The pi-calculus

Jesper Bengtson

September 1, 2025

Abstract

Contents

1 Overview **1**

2 Formalisation **2**

1 Overview

These theories formalise the following results for psi-calculi. Note that there is only an early semantics for psi-calculi, although a late one may appear later.

- strong bisimilarity is preserved by all operators except the input-prefix
- strong equivalence is a congruence
- weak bisimilarity is preserved by all operators except case and the input-prefix
- weak congruence is a congruence
- strong equivalence respect the laws of structural congruence
- all strongly equivalent agents are also weakly congruent which in turn are weakly bisimilar. Moreover, strongly equivalent agents are also strongly bisimilar
- as a corollary of the last two points, all mentioned equivalences respect the law of structural congruence
- for instances of psi-calculi where assertion composition satisfies weakening, the definition of weak bisimilarity can be simplified significantly and proven equivalent to the version that applies when weakening does not hold

- for certain versions of psi-calculi, sum can be encoded
- for certain versions of psi-calculi, the tau-prefix can be encoded and when weakening is satisfied, all of the tau-laws hold.

The file naming convention is hopefully self explanatory, where the prefixes *Strong* and *Weak* denote that the file covers theories required to formalise properties of strong and weak bisimilarity respectively; files with the prefix *Weaken* cover theories where weakening holds for the static implication; if the file name contains *Sim* the theories cover simulation, file names containing *Bisim* cover bisimulation, and file names containing *Cong* cover weak congruence; files with the suffix *Pres* deal with theories that reason about preservation properties of operators such as a simulation or bisimulation being preserved by a certain operator; files with the suffix *StructCong* reason about structural congruence.

For a complete exposition of all of theories, please consult Bengtson's Ph. D. thesis [1]. A shorter presentation can be found in our TPHOLs paper 'Psi-calculi in Isabelle' from 2009 [3]. There are also two LICS-papers that focus on the mathematical theories, rather than the Isabelle formalisations [2, 4].

2 Formalisation

```
theory Chain
  imports HOL−Nominal.Nominal
begin

lemma pt-set-nil:
  fixes Xs :: 'a set
  assumes pt: pt TYPE('a) TYPE ('x)
  and     at: at TYPE('x)

  shows ([]::'x prm)•Xs = Xs
by(auto simp add: perm-set-def pt1[OF pt])

lemma pt-set-append:
  fixes pi1 :: 'x prm
  and   pi2 :: 'x prm
  and   Xs  :: 'a set
  assumes pt: pt TYPE('a) TYPE ('x)
  and     at: at TYPE('x)

  shows (pi1@pi2)•Xs = pi1•(pi2•Xs)
by(auto simp add: perm-set-def pt2[OF pt])

lemma pt-set-prm-eq:
```

fixes *pi1* :: *'x prm*
and *pi2* :: *'x prm*
and *Xs* :: *'a set*
assumes *pt*: *pt TYPE('a) TYPE ('x)*
and *at*: *at TYPE('x)*

shows $pi1 \triangleq pi2 \Longrightarrow pi1 \cdot Xs = pi2 \cdot Xs$
by(*auto simp add*: *perm-set-def pt3*[*OF pt*])

lemma *pt-set-inst*:
assumes *pt*: *pt TYPE('a) TYPE ('x)*
and *at*: *at TYPE('x)*

shows *pt TYPE('a set) TYPE('x)*
apply(*simp add*: *pt-def pt-set-nil*[*OF pt*, *OF at*] *pt-set-append*[*OF pt*, *OF at*])
apply(*clarify*)
by(*rule pt-set-prm-eq*[*OF pt*, *OF at*])

lemma *pt-ball-eqvt*:
fixes *pi* :: *'a prm*
and *Xs* :: *'b set*
and *P* :: *'b* $\Rightarrow$ *bool*

assumes *pt*: *pt TYPE('b) TYPE('a)*
and *at*: *at TYPE('a)*

shows $(pi \cdot (\forall x \in Xs.\ P\ x)) = (\forall x \in (pi \cdot Xs).\ pi \cdot P\ (rev\ pi \cdot x))$
apply(*auto simp add*: *perm-bool*)
apply(*drule-tac pi=rev pi* **in** *pt-set-bij2*[*OF pt*, *OF at*])
apply(*simp add*: *pt-rev-pi*[*OF pt-set-inst*[*OF pt*, *OF at*], *OF at*])
apply(*drule-tac pi=pi* **in** *pt-set-bij2*[*OF pt*, *OF at*])
apply(*erule-tac x=pi* $\cdot$ *x* **in** *ballE*)
apply(*simp add*: *pt-rev-pi*[*OF pt*, *OF at*])
by *simp*

lemma *perm-cart-prod*:
fixes *Xs* :: *'b set*
and *Ys* :: *'c set*
and *p* :: *'a prm*

assumes *pt1*: *pt TYPE('b) TYPE('a)*
and *pt2*: *pt TYPE('c) TYPE('a)*
and *at*: *at TYPE('a)*

shows $(p \cdot (Xs \times Ys)) = (((p \cdot Xs) \times (p \cdot Ys))::(('b \times 'c)\ set))$
by(*auto simp add*: *perm-set-def*)

lemma *supp-member*:
fixes *Xs* :: *'b set*

```
  and   x :: 'a

  assumes pt: pt TYPE('b) TYPE('a)
  and     at: at TYPE('a)
  and     fs: fs TYPE('b) TYPE('a)
  and     finite Xs
  and     x ∈ ((supp Xs)::'a set)

  obtains X where (X::'b) ∈ Xs and x ∈ supp X
proof −
  from ‹finite Xs› ‹x ∈ supp Xs› have ∃ X::'b. (X ∈ Xs) ∧ (x ∈ (supp X))
  proof(induct rule: finite-induct)
    case empty
    from ‹x ∈ ((supp {})::'a set)› have False
      by(simp add: supp-set-empty)
    thus ?case by simp
  next
    case(insert y Xs)
    show ?case
    proof(case-tac x ∈ supp y)
      assume x ∈ supp y
      thus ?case by force
    next
      assume x ∉ supp y
      with ‹x ∈ supp(insert y Xs)› have x ∈ supp Xs
        by(simp add: supp-fin-insert[OF pt, OF at, OF fs, OF ‹finite Xs›])
      with ‹x ∈ supp Xs ⟹ ∃ X. X ∈ Xs ∧ x ∈ supp X›
      show ?case by force
    qed
  qed
  with that show ?thesis by blast
qed

lemma supp-cart-prod-empty[simp]:
  fixes Xs :: 'b set

  shows supp (Xs × {}) = ({}::'a set)
  and   supp ({} × Xs) = ({}::'a set)
by(auto simp add: supp-set-empty)

lemma supp-cart-prod:
  fixes Xs :: 'b set
  and   Ys :: 'c set

  assumes pt1: pt TYPE('b) TYPE('a)
  and     pt2: pt TYPE('c) TYPE('a)
  and     fs1: fs TYPE('b) TYPE('a)
  and     fs2: fs TYPE('c) TYPE('a)
  and     at:  at TYPE('a)
```

```
  and     f1: finite Xs
  and     f2: finite Ys
  and     a:  Xs ≠ {}
  and     b:  Ys ≠ {}

  shows ((supp (Xs × Ys))::'a set) = ((supp Xs) ∪ (supp Ys))
proof −
  from f1 f2 have f3: finite(Xs × Ys) by simp
  show ?thesis
    apply(simp add: supp-of-fin-sets[OF pt-prod-inst[OF pt1, OF pt2], OF at, OF
fs-prod-inst[OF fs1, OF fs2], OF f3] supp-prod)
    apply(rule equalityI)
   using Union-included-in-supp[OF pt1, OF at, OF fs1, OF f1] Union-included-in-supp[OF
pt2, OF at, OF fs2, OF f2]
    apply(force simp add: supp-prod)
    using a b
    apply(auto simp add: supp-prod)
    using supp-member[OF pt1, OF at, OF fs1, OF f1]
    apply blast
    using supp-member[OF pt2, OF at, OF fs2, OF f2]
    by blast
qed

lemma fresh-cart-prod:
  fixes x  :: 'a
  and   Xs :: 'b set
  and   Ys :: 'c set

  assumes pt1: pt TYPE('b) TYPE('a)
  and     pt2: pt TYPE('c) TYPE('a)
  and     fs1: fs TYPE('b) TYPE('a)
  and     fs2: fs TYPE('c) TYPE('a)
  and     at:  at TYPE('a)
  and     f1:  finite Xs
  and     f2:  finite Ys
  and     a:   Xs ≠ {}
  and     b:   Ys ≠ {}

  shows (x ♯ (Xs × Ys)) = (x ♯ Xs ∧ x ♯ Ys)
using assms
by(simp add: supp-cart-prod fresh-def)

lemma fresh-star-cart-prod:
  fixes Zs   :: 'a set
  and   xvec :: 'a list
  and   Xs   :: 'b set
  and   Ys   :: 'c set

  assumes pt1: pt TYPE('b) TYPE('a)
```

```
  and     pt2: pt TYPE('c) TYPE('a)
  and     fs1: fs TYPE('b) TYPE('a)
  and     fs2: fs TYPE('c) TYPE('a)
  and     at:  at TYPE('a)
  and     f1:  finite Xs
  and     f2:  finite Ys
  and     a:   Xs ≠ {}
  and     b:   Ys ≠ {}

  shows (Zs ♯* (Xs × Ys)) = (Zs ♯* Xs ∧ Zs ♯* Ys)
  and   (xvec ♯* (Xs × Ys)) = (xvec ♯* Xs ∧ xvec ♯* Ys)
using assms
by(force simp add: fresh-cart-prod fresh-star-def)+

lemma permCommute:
  fixes p  :: 'a prm
  and   q  :: 'a prm
  and   P  :: 'x
  and   Xs :: 'a set
  and   Ys :: 'a set

  assumes pt: pt TYPE('x) TYPE('a)
  and     at: at TYPE('a)
  and     a: (set p) ⊆ Xs × Ys
  and     b: Xs ♯* q
  and     c: Ys ♯* q

  shows p · q · P = q · p · P
proof −
  have p · q · P = (p · q) · p · P
    by(rule pt-perm-compose[OF pt, OF at])
  moreover from at have pt TYPE('a) TYPE('a)
    by(rule at-pt-inst)
  hence pt TYPE(('a × 'a) list) TYPE('a)
    by(force intro: pt-prod-inst pt-list-inst)
  hence p · q = q using at a b c
    by(rule pt-freshs-freshs)
  ultimately show ?thesis by simp
qed


definition
  distinctPerm :: 'a prm ⇒ bool where
  distinctPerm p ≡ distinct((map fst p)@(map snd p))

lemma at-set-avoiding-aux':
  fixes Xs::'a set
  and   As::'a set
  assumes at: at TYPE('a)
```

```
  and     a: finite Xs
  and     b: Xs ⊆ As
  and     c: finite As
  and     d: finite ((supp c)::'a set)
  shows ∃(Ys::'a set) (pi::'a prm). Ys♯*c ∧ Ys ∩ As = {} ∧ (pi•Xs=Ys) ∧
        set pi ⊆ Xs × Ys ∧ finite Ys ∧ (distinctPerm pi)
using a b c
proof (induct)
  case empty
  have ({}::'a set)♯*c by (simp add: fresh-star-def)
  moreover
  have ({}::'a set) ∩ As = {} by simp
  moreover
  have ([]::'a prm)•{} = ({}::'a set)
    by(rule pt1) (metis Nominal.pt-set-inst at at-pt-inst)
  moreover
  have set ([]::'a prm) ⊆ {} × {} by simp
  moreover
  have finite ({}::'a set) by simp
  moreover have distinctPerm([]::'a prm)
    by(simp add: distinctPerm-def)
  ultimately show ?case by blast
next
  case (insert x Xs)
  then have ih: ∃ Ys pi. Ys♯*c ∧ Ys ∩ As = {} ∧ pi•Xs = Ys ∧ set pi ⊆ Xs ×
Ys ∧ finite Ys ∧ distinctPerm pi by simp
  then obtain Ys pi where a1: Ys♯*c and a2: Ys ∩ As = {} and a3: (pi::'a
prm)•Xs = Ys and
                    a4: set pi ⊆ Xs × Ys and a5: finite Ys
                  and a6: distinctPerm pi by blast
  have b: x∉Xs by fact
  have d1: finite As by fact
  have d2: finite Xs by fact
  have d3: insert x Xs ⊆ As by fact
  have d4: finite((supp pi)::'a set)
    by(induct pi)
      (auto simp add: supp-list-nil supp-prod at-fin-set-supp[OF at]
                  supp-list-cons at-supp[OF at])
  have ∃ y::'a. y♯(c,x,Ys,As,pi) using d d1 a5 d4
    by (rule-tac at-exists-fresh'[OF at])
       (simp add: supp-prod at-supp[OF at] at-fin-set-supp[OF at])
  then obtain y::'a where  e: y♯(c,x,Ys,As,pi) by blast
  have ({y}∪Ys)♯*c using a1 e by (simp add: fresh-star-def)
  moreover
 have ({y}∪Ys)∩As = {} using a2 d1 e by (simp add: fresh-prod at-fin-set-fresh[OF
at])
  moreover
  have (((pi•x,y)#pi)•(insert x Xs)) = {y}∪Ys
  proof −
```

```
    have eq: [(pi•x,y)]•Ys = Ys
    proof −
      have (pi•x)♯Ys using a3[symmetric] b d2
        by(simp add: pt-fresh-bij[OF pt-set-inst, OF at-pt-inst[OF at], OF at, OF
at]
                    at-fin-set-fresh[OF at d2])
      moreover
      have y♯Ys using e by simp
      ultimately show [(pi•x,y)]•Ys = Ys
        by (simp add: pt-fresh-fresh[OF pt-set-inst, OF at-pt-inst[OF at], OF at,
OF at])
    qed
    have (((pi•x,y)#pi)•({x}∪Xs)) = ([(pi•x,y)]•(pi•({x}∪Xs)))
      by (simp add: pt2[symmetric, OF pt-set-inst, OF at-pt-inst[OF at], OF at])
    also have ... = {y}∪([(pi•x,y)]•(pi•Xs))
      by (simp only: union-eqvt perm-set-def at-calc[OF at])(auto)
    also have ... = {y}∪([(pi•x,y)]•Ys) using a3 by simp
    also have ... = {y}∪Ys using eq by simp
    finally show (((pi•x,y)#pi)•(insert x Xs)) = {y}∪Ys by auto
  qed
  moreover
  have pi•x=x using a4 b a2 a3 d3 by (rule-tac at-prm-fresh2[OF at]) (auto)
  then have set ((pi•x,y)#pi) ⊆ (insert x Xs) × ({y}∪Ys) using a4 by auto
  moreover
  have finite ({y}∪Ys) using a5 by simp
  moreover from ‹Ys ∩ As = {}› ‹(insert x Xs) ⊆ As› ‹finite Ys› have x ∉ Ys
    by(auto simp add: fresh-def at-fin-set-supp[OF at])
  with a6 ‹pi • x = x› ‹x ∉ Xs› ‹(set pi) ⊆ Xs × Ys› e have distinctPerm((pi •
x, y)#pi)
    apply(auto simp add: distinctPerm-def fresh-prod at-fresh[OF at])
    proof −
      fix a b
      assume b ♯ pi and (a, b) ∈ set pi
      thus False
        by(induct pi)
            (auto simp add: supp-list-cons supp-prod at-supp[OF at] fresh-list-cons
fresh-prod at-fresh[OF at])
    next
      fix a b
      assume a ♯ pi and (a, b) ∈ set pi
      thus False
        by(induct pi)
            (auto simp add: supp-list-cons supp-prod at-supp[OF at] fresh-list-cons
fresh-prod at-fresh[OF at])
    qed
  ultimately
  show ?case by blast
qed
```

lemma *at-set-avoiding*:
  fixes *Xs*::*'a set*
  assumes *at*: *at TYPE('a)*
  and *a*: *finite Xs*
  and *b*: *finite ((supp c)::'a set)*
  obtains *pi*::*'a prm* **where** *(pi · Xs) ♯∗ c* **and** *set pi ⊆ Xs × (pi · Xs)* **and** *distinctPerm pi*
  using *a b*
  by (*frule-tac As=Xs* **in** *at-set-avoiding-aux'[OF at]*) *auto*

lemma *pt-swap*:
  fixes *x* :: *'a*
  and *a* :: *'x*
  and *b* :: *'x*

  assumes *pt*: *pt TYPE('a) TYPE('x)*
  and *at*: *at TYPE('x)*

  shows *[(a, b)] · x = [(b, a)] · x*
proof −
  show *?thesis* **by**(*simp add*: *pt3[OF pt] at-ds5[OF at]*)
qed

atom-decl *name*

lemma *supp-subset*:
  fixes *Xs* :: *'a::fs-name set*
  and *Ys* :: *'a::fs-name set*

  assumes *Xs ⊆ Ys*
  and *finite Xs*
  and *finite Ys*

  shows *(supp Xs) ⊆ ((supp Ys)::name set)*
proof(*rule subsetI*)
  fix *x*
  assume *x ∈ ((supp Xs)::name set)*
  with ‹*finite Xs*› **obtain** *X* **where** *X ∈ Xs* **and** *x ∈ supp X*
    by(*rule supp-member[OF pt-name-inst, OF at-name-inst, OF fs-name-inst]*)
  from ‹*X ∈ Xs*› ‹*Xs ⊆ Ys*› **have** *X ∈ Ys* **by** *auto*
  with ‹*finite Ys*› ‹*x ∈ supp X*› **show** *x ∈ supp Ys*
    by(*induct rule*: *finite-induct*)
      (*auto simp add*: *supp-fin-insert[OF pt-name-inst, OF at-name-inst, OF fs-name-inst]*)
qed

abbreviation *mem-def* :: *'a ⇒ 'a list ⇒ bool* (‹- *mem* -› *[80, 80] 80*) **where**
  x mem xs ≡ x ∈ set xs

lemma *memFresh*:
  fixes x :: *name*
  and p :: $'a$::*fs-name*
  and l :: ($'a$::*fs-name*) *list*

  assumes $x \sharp l$
  and p *mem* l

  shows $x \sharp p$
using *assms*
by(*induct l*, *auto simp add*: *fresh-list-cons*)

lemma *memFreshChain*:
  fixes *xvec* :: *name list*
  and p :: $'a$::*fs-name*
  and l :: $'a$::*fs-name list*
  and *Xs* :: *name set*

  assumes p *mem* l

  shows $xvec \sharp* l \Longrightarrow xvec \sharp* p$
  and $Xs \sharp* l \Longrightarrow Xs \sharp* p$
using *assms*
by(*auto simp add*: *fresh-star-def intro*: *memFresh*)

lemma *fresh-star-list-append*[*simp*]:
  fixes A :: *name list*
  and B :: *name list*
  and C :: *name list*

  shows $(A \sharp* (B @ C)) = ((A \sharp* B) \wedge (A \sharp* C))$
by(*auto simp add*: *fresh-star-def fresh-list-append*)

lemma *unionSimps*[*simp*]:
  fixes *Xs* :: *name set*
  and *Ys* :: *name set*
  and C :: $'a$::*fs-name*

  shows $((Xs \cup Ys) \sharp* C) = ((Xs \sharp* C) \wedge (Ys \sharp* C))$
by(*auto simp add*: *fresh-star-def*)

lemma *substFreshAux*[*simp*]:
  fixes C :: $'a$::*fs-name*
  and *xvec* :: *name list*

  shows $xvec \sharp* (supp\ C - set\ xvec)$
by(*auto simp add*: *fresh-star-def fresh-def at-fin-set-supp*[*OF at-name-inst*] *fs-name1*)

lemma *fresh-star-perm-app*[*simp*]:
  fixes Xs :: *name set*
  and $xvec$ :: *name list*
  and p :: *name prm*
  and C :: $'d$::*fs-name*

  shows $[\![Xs \sharp* p;\ Xs \sharp* C]\!] \Longrightarrow Xs \sharp* (p \cdot C)$
  and $[\![xvec \sharp* p;\ xvec \sharp* C]\!] \Longrightarrow xvec \sharp* (p \cdot C)$
by(*auto simp add*: *fresh-star-def fresh-perm-app*)

lemma *freshSets*[*simp*]:
  fixes x :: *name*
  and y :: *name*
  and $xvec$ :: *name list*
  and X :: *name set*
  and C :: $'a$

  shows ([]::*name list*) $\sharp* C$
  and ([]::*name list*) $\sharp* [y].C$
  and ({}::*name set*) $\sharp* C$
  and ({}::*name set*) $\sharp* [y].C$
  and $((x\#xvec) \sharp* C) = (x \sharp C \wedge xvec \sharp* C)$
  and $((x\#xvec) \sharp* ([y].C)) = (x \sharp ([y].C) \wedge xvec \sharp* ([y].C))$
  and $((insert\ x\ X) \sharp* C) = (x \sharp C \wedge X \sharp* C)$
  and $((insert\ x\ X) \sharp* ([y].C)) = (x \sharp ([y].C) \wedge X \sharp* ([y].C))$
by(*auto simp add*: *fresh-star-def*)

lemma *freshStarAtom*[*simp*]: ($xvec$::*name list*) $\sharp*$ (x::*name*) $= x \sharp xvec$
by(*induct xvec*)
  (*auto simp add*: *fresh-list-nil fresh-list-cons fresh-atm*)

lemma *name-list-set-fresh*[*simp*]:
  fixes $xvec$ :: *name list*
  and x :: $'a$::*fs-name*

  shows (*set xvec*) $\sharp* x = xvec \sharp* x$
by(*auto simp add*: *fresh-star-def*)

lemma *name-list-supp*:
  fixes $xvec$ :: *name list*

  shows *set xvec* = *supp xvec*
proof −
  have *set xvec* = *supp*(*set xvec*)
    by(*simp add*: *at-fin-set-supp*[*OF at-name-inst*])
  moreover have ... = *supp xvec*
   by(*simp add*: *pt-list-set-supp*[*OF pt-name-inst*, *OF at-name-inst*, *OF fs-name-inst*])
  ultimately show *?thesis*
    by *simp*

qed

lemma *abs-fresh-list-star*:
  fixes *xvec* :: *name list*
  and *a* :: *name*
  and *P* :: *'a::fs-name*

  shows $(xvec \sharp* [a].P) = ((set\ xvec) - \{a\}) \sharp* P$
by(*induct xvec*) (*auto simp add*: *fresh-star-def abs-fresh*)

lemma *abs-fresh-set-star*:
  fixes *X* :: *name set*
  and *a* :: *name*
  and *P* :: *'a::fs-name*

  shows $(X \sharp* [a].P) = (X - \{a\}) \sharp* P$
by(*auto simp add*: *fresh-star-def abs-fresh*)

lemmas *abs-fresh-star* = *abs-fresh-list-star abs-fresh-set-star*

lemma *abs-fresh-list-star'*[*simp*]:
  fixes *xvec* :: *name list*
  and *a* :: *name*
  and *P* :: *'a::fs-name*

  assumes $a \sharp xvec$

  shows $xvec \sharp* [a].P = xvec \sharp* P$
using *assms*
by(*induct xvec*) (*auto simp add*: *abs-fresh fresh-list-cons fresh-atm*)

lemma *freshChainSym*[*simp*]:
  fixes *xvec* :: *name list*
  and *yvec* :: *name list*

  shows $xvec \sharp* yvec = yvec \sharp* xvec$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)

lemmas [*eqvt*] = *perm-cart-prod*[*OF pt-name-inst*, *OF pt-name-inst*, *OF at-name-inst*]

lemma *name-set-avoiding*:
  fixes *c* :: *'a::fs-name*
  and *X* :: *name set*

  assumes *finite X*
  and $\bigwedge pi{::}name\ prm.\ [\![(pi \cdot X) \sharp* c;\ distinctPerm\ pi;\ set\ pi \subseteq X \times (pi \cdot X)]\!]$
$\Longrightarrow$ *thesis*

  shows *thesis*

using *assms*
by(*rule-tac c=c* **in** *at-set-avoiding*[*OF at-name-inst*]) (*simp-all add*: *fs-name1*)

lemmas *simps*[*simp*] = *fresh-atm fresh-prod*
pt3[*OF pt-name-inst*, *OF at-ds1*, *OF at-name-inst*]
pt-fresh-fresh[*OF pt-name-inst*, *OF at-name-inst*]
pt-rev-pi[*OF pt-name-inst*, *OF at-name-inst*]
pt-pi-rev[*OF pt-name-inst*, *OF at-name-inst*]

lemmas *name-supp-cart-prod* = *supp-cart-prod*[*OF pt-name-inst*, *OF pt-name-inst*, *OF fs-name-inst*, *OF fs-name-inst*, *OF at-name-inst*]
lemmas *name-fresh-cart-prod* = *fresh-cart-prod*[*OF pt-name-inst*, *OF pt-name-inst*, *OF fs-name-inst*, *OF fs-name-inst*, *OF at-name-inst*]
lemmas *name-fresh-star-cart-prod* = *fresh-star-cart-prod*[*OF pt-name-inst*, *OF pt-name-inst*, *OF fs-name-inst*, *OF fs-name-inst*, *OF at-name-inst*]

lemmas *name-swap-bij*[*simp*] = *pt-swap-bij*[*OF pt-name-inst*, *OF at-name-inst*]
lemmas *name-swap* = *pt-swap*[*OF pt-name-inst*, *OF at-name-inst*]
lemmas *name-set-fresh-fresh*[*simp*] = *pt-freshs-freshs*[*OF pt-name-inst*, *OF at-name-inst*]
lemmas *list-fresh*[*simp*] = *fresh-list-nil fresh-list-cons fresh-list-append*

definition *eqvt* :: *'a*::*fs-name set* ⇒ *bool* **where**
eqvt X ≡ ∀ *x* ∈ *X*. ∀ *p*::*name prm*. *p* · *x* ∈ *X*

lemma *eqvtUnion*[*intro*]:
fixes *Rel* :: (*'d*::*fs-name*) *set*
and *Rel'* :: *'d set*

assumes *EqvtRel*: *eqvt Rel*
and *EqvtRel'*: *eqvt Rel'*

shows *eqvt* (*Rel* ∪ *Rel'*)
using *assms*
by(*force simp add*: *eqvt-def*)

lemma *eqvtPerm*[*simp*]:
fixes *X* :: (*'d*::*fs-name*) *set*
and *x* :: *name*
and *y* :: *name*

assumes *eqvt X*

shows ([(*x*, *y*)] · *X*) = *X*
using *assms*
apply(*auto simp add*: *eqvt-def*)
apply(*erule-tac x*=[(*x*, *y*)] · *xa* **in** *ballE*)
apply(*erule-tac x*=[(*x*, *y*)] **in** *allE*)
apply *simp*

apply(*drule-tac pi*=[(*x*, *y*)] **in** *pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*])
apply *simp*
apply(*erule-tac x*=*xa* **in** *ballE*)
apply(*erule-tac x*=[(*x*, *y*)] **in** *allE*)
apply(*drule-tac pi*=[(*x*, *y*)] **and** *x*=[(*x*, *y*)] · *xa* **in** *pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*])
apply *simp*
by *simp*

lemma *eqvtI*:
  fixes X :: *'d*::*fs-name set*
  and x :: *'d*
  and p :: *name prm*

  assumes *eqvt X*
  and $x \in X$

  shows $(p \cdot x) \in X$
using *assms*
by(*unfold eqvt-def*) *auto*

lemma *fresh-star-list-nil*[*simp*]:
  fixes *xvec* :: *name list*
  and *Xs* :: *name set*

  shows *xvec* $\sharp*$ []
  and *Xs* $\sharp*$ []
by(*auto simp add*: *fresh-star-def*)

lemma *fresh-star-list-cons*[*simp*]:
  fixes *xvec* :: *name list*
  and *Xs* :: *name set*
  and x :: *'a*::*fs-name*
  and *xs* :: *'a list*

  shows (*xvec* $\sharp*$ (*x*#*xs*)) = ((*xvec* $\sharp*$ *x*) $\wedge$ *xvec* $\sharp*$ *xs*)
  and (*Xs* $\sharp*$ (*x*#*xs*)) = ((*Xs* $\sharp*$ *x*) $\wedge$ (*Xs* $\sharp*$ *xs*))
by(*auto simp add*: *fresh-star-def*)

lemma *freshStarPair*[*simp*]:
  fixes X :: *name set*
  and *xvec* :: *name list*
  and x :: *'a*::*fs-name*
  and y :: *'b*::*fs-name*

  shows (X $\sharp*$ (x, y)) = (X $\sharp*$ x $\wedge$ X $\sharp*$ y)
  and (*xvec* $\sharp*$ (x, y)) = (*xvec* $\sharp*$ x $\wedge$ *xvec* $\sharp*$ y)
by(*auto simp add*: *fresh-star-def*)
lemma *name-list-avoiding*:

fixes c :: $'a$::*fs-name*
and *xvec* :: *name list*

assumes $\bigwedge$*pi*::*name prm*. $[\![$(*pi* · *xvec*) $\sharp*$ *c*; *distinctPerm pi*; *set pi* $\subseteq$ (*set xvec*) $\times$ (*set* (*pi* · *xvec*))$]\!] \Longrightarrow$ *thesis*

shows *thesis*
proof −
have *finite*(*set xvec*) **by** *simp*
thus *?thesis* **using** *assms*
by(*rule name-set-avoiding*) (*auto simp add*: *eqvts fresh-star-def*)
qed

lemma *distinctPermSimps*[*simp*]:
fixes *p* :: *name prm*
and *a* :: *name*
and *b* :: *name*

shows *distinctPerm*([]::*name prm*)
and (*distinctPerm*((*a*, *b*)#*p*)) = (*distinctPerm p* $\wedge$ *a* $\neq$ *b* $\wedge$ *a* $\sharp$ *p* $\wedge$ *b* $\sharp$ *p*)
apply(*simp add*: *distinctPerm-def*)
apply(*induct p*)
apply(*unfold distinctPerm-def*)
apply(*clarsimp*)
apply(*rule iffI*, *erule iffE*)
by(*clarsimp*)+

lemma *map-eqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *lst* :: $'a$::*pt-name list*

shows (*p* · (*map f lst*)) = *map* (*p* · *f*) (*p* · *lst*)
apply(*induct lst*, *auto*)
by(*simp add*: *pt-fun-app-eq*[*OF pt-name-inst*, *OF at-name-inst*])

lemma *consPerm*:
fixes *x* :: *name*
and *y* :: *name*
and *p* :: *name prm*
and *C* :: $'a$::*pt-name*

shows ((*x*, *y*)#*p*) · *C* = [(*x*, *y*)] · *p* · *C*
by(*simp add*: *pt2*[*OF pt-name-inst*, *THEN sym*])

simproc-setup *consPerm* (((*x*, *y*)#*p*) · *C*) = ‹
fn - => *fn* - => *fn ct* =>
case Thm.term-of ct of
Const (@{*const-name perm*}, -) $ (*Const* (@{*const-name Cons*}, -) $ - $ *p*)
$ - =>

```
      (case p of
        Const (@{const-name Nil}, -) => NONE
      | - => SOME(mk-meta-eq @{thm consPerm}))
    | - => NONE
›

lemma distinctEqvt[eqvt]:
  fixes p  :: name prm
  and   xs :: 'a::pt-name list

  shows (p • (distinct xs)) = distinct (p • xs)
by(induct xs) (auto simp add: eqvts)

lemma distinctClosed[simp]:
  fixes p  :: name prm
  and   xs :: 'a::pt-name list

  shows distinct (p • xs) = distinct xs
apply(induct xs)
apply(auto simp add: eqvts)
apply(drule-tac pi=p in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
apply(simp add: eqvts)
apply(drule-tac pi=rev p in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
by(simp add: eqvts)

lemma lengthEqvt[eqvt]:
  fixes p  :: name prm
  and   xs :: 'a::pt-name list

  shows p • (length xs) = length (p • xs)
by(induct xs) (auto simp add: eqvts)

lemma lengthClosed[simp]:
  fixes p  :: name prm
  and   xs :: 'a::pt-name list

  shows length (p • xs) = length xs
by(induct xs) (auto simp add: eqvts)

lemma subsetEqvt[eqvt]:
  fixes p :: name prm
  and   S :: ('a::pt-name) set
  and   T :: ('a::pt-name) set

  shows (p • (S ⊆ T)) = ((p • S) ⊆ (p • T))
by(rule pt-subseteq-eqvt[OF pt-name-inst, OF at-name-inst])

lemma subsetClosed[simp]:
  fixes p :: name prm
```

```
  and   S :: ('a::pt-name) set
  and   T :: ('a::pt-name) set

  shows ((p · S) ⊆ (p · T)) = (S ⊆ T)
apply auto
apply(drule-tac pi=p in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
apply(insert pt-set-bij[OF pt-name-inst, OF at-name-inst])
apply auto
apply(drule-tac pi=rev p in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
apply auto
apply(subgoal-tac rev p · x ∈ T)
apply auto
apply(drule-tac pi=p in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
apply auto
apply(drule-tac pi=p in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
by auto

lemma subsetClosed'[simp]:
  fixes p    :: name prm
  and   xvec :: name list
  and   P    :: 'a::fs-name

  shows (set (p · xvec) ⊆ supp (p · P)) = (set xvec ⊆ supp P)
by(simp add: eqvts[THEN sym])

lemma memEqvt[eqvt]:
  fixes p  :: name prm
  and   x  :: 'a::pt-name
  and   xs :: ('a::pt-name) list

  shows (p · (x mem xs)) = ((p · x) mem (p · xs))
by(induct xs)
  (auto simp add: pt-bij[OF pt-name-inst, OF at-name-inst] eqvts)

lemma memClosed[simp]:
  fixes p  :: name prm
  and   x  :: 'a::pt-name
  and   xs :: ('a::pt-name) list

  shows (p · x) mem (p · xs) = (x mem xs)
proof −
  have x mem xs = p · (x mem xs)
    by(case-tac x mem xs) auto
  thus ?thesis by(simp add: eqvts)
qed

lemma memClosed'[simp]:
  fixes p  :: name prm
  and   x  :: 'a::pt-name
```

and y :: $'b$::*pt-name*
and xs :: $('a \times 'b)$ *list*

shows $((p \cdot x,\ p \cdot y)$ *mem* $(p \cdot xs)) = ((x,\ y)$ *mem* $xs)$
apply(*subgoal-tac* $((x,\ y)$ *mem* $xs) = (p \cdot (x,\ y))$ *mem* $(p \cdot xs))$
apply *force*
by(*force simp del*: *eqvts*)

lemma *freshPerm*:
fixes x :: *name*
and p :: *name prm*

assumes $x \sharp p$

shows $p \cdot x = x$
using *assms*
apply(*rule-tac pt-pi-fresh-fresh*[*OF pt-name-inst*, *OF at-name-inst*])
by(*induct p*, *auto simp add*: *fresh-list-cons fresh-prod*)

lemma *freshChainPermSimp*:
fixes *xvec* :: *name list*
and p :: *name prm*

assumes $xvec \sharp* p$

shows $p \cdot xvec = xvec$
and $rev\ p \cdot xvec = xvec$
using *assms*
by(*induct xvec*) (*auto simp add*: *freshPerm pt-bij1*[*OF pt-name-inst*, *OF at-name-inst*, *THEN sym*])

lemma *freshChainAppend*[*simp*]:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and C :: $'a$::*fs-name*

shows $(xvec@yvec) \sharp* C = ((xvec \sharp* C) \wedge (yvec \sharp* C))$
by(*force simp add*: *fresh-star-def*)

lemma *subsetFresh*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and C :: $'d$::*fs-name*

assumes $set\ xvec \subseteq set\ yvec$
and $yvec \sharp* C$

shows $xvec \sharp* C$

```
using assms
by(auto simp add: fresh-star-def)

lemma distinctPermCancel[simp]:
  fixes p :: name prm
  and   T :: 'a::pt-name

  assumes distinctPerm p

  shows (p · (p · T)) = T
using assms
proof(induct p)
  case Nil
  show ?case by simp
next
  case(Cons a p)
  thus ?case
  proof(case-tac a, auto)
    fix a b
    assume (a::name) ♯ (p::name prm) b ♯ p p · p · T = T a ≠ b
    thus [(a, b)] · p · [(a, b)] · p · T = T
      by(subst pt-perm-compose[OF pt-name-inst, OF at-name-inst]) simp
  qed
qed

fun composePerm :: name list ⇒ name list ⇒ name prm
where
  Base:  composePerm [] [] = []
| Step:  composePerm (x#xs) (y#ys) = (x, y)#(composePerm xs ys)
| Empty: composePerm - -= []

lemma composePermInduct[consumes 1, case-names cBase cStep]:
  fixes xvec :: name list
  and   yvec :: name list
  and   P    :: name list ⇒ name list ⇒ bool

  assumes L: length xvec = length yvec
  and     rBase: P [] []
  and     rStep: ⋀x xvec y yvec. ⟦length xvec = length yvec; P xvec yvec⟧ ⟹ P (x
# xvec) (y # yvec)

  shows P xvec yvec
using assms
by(induct rule: composePerm.induct) auto

lemma composePermEqvt[eqvt]:
  fixes p    :: name prm
  and   xvec :: name list
  and   yvec :: name list
```

shows $(p \cdot (composePerm\ xvec\ yvec)) = composePerm\ (p \cdot xvec)\ (p \cdot yvec)$
by(*induct xvec yvec rule*: *composePerm.induct*) *auto*

abbreviation
composePermJudge (‹[- -] $\cdot_v$ -› [*80*, *80*, *80*] *80*) **where** $[xvec\ yvec] \cdot_v p \equiv (composePerm\ xvec\ yvec) \cdot p$

abbreviation
composePermInvJudge (‹[- -]$^-$ $\cdot_v$ -› [*80*, *80*, *80*] *80*) **where** $[xvec\ yvec]^- \cdot_v p \equiv (rev\ (composePerm\ xvec\ yvec)) \cdot p$

lemma *permChainSimps*[*simp*]:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and *perm* :: *name prm*
and *p* :: *'a::pt-name*

shows $((composePerm\ xvec\ yvec)\ @\ perm) \cdot p = [xvec\ yvec] \cdot_v (perm \cdot p)$
by(*simp add*: *pt2*[*OF pt-name-inst*])

lemma *permChainEqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *xvec* :: *name list*
and *yvec* :: *name list*
and *x* :: *'a::pt-name*

shows $(p \cdot ([xvec\ yvec] \cdot_v x)) = [(p \cdot xvec)\ (p \cdot yvec)] \cdot_v (p \cdot x)$
and $(p \cdot ([xvec\ yvec]^- \cdot_v x)) = [(p \cdot xvec)\ (p \cdot yvec)]^- \cdot_v (p \cdot x)$
by(*subst pt-perm-compose*[*OF pt-name-inst*, *OF at-name-inst*], *simp add*: *eqvts rev-eqvt*)+

lemma *permChainBij*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and *p* :: *'a::pt-name*
and *q* :: *'a::pt-name*

assumes *length xvec* = *length yvec*

shows $(([xvec\ yvec] \cdot_v p) = ([xvec\ yvec] \cdot_v q)) = (p = q)$
and $(([xvec\ yvec]^- \cdot_v p) = ([xvec\ yvec]^- \cdot_v q)) = (p = q)$
using *assms*
by(*induct rule*: *composePermInduct*)
(*auto simp add*: *pt-bij*[*OF pt-name-inst*, *OF at-name-inst*])

lemma *permChainAppend*:
fixes *xvec1* :: *name list*
and *yvec1* :: *name list*

and *xvec2* :: *name list*
and *yvec2* :: *name list*
and *p* :: *'a::pt-name*

assumes *length xvec1* = *length yvec1*

shows $([(xvec1@xvec2)\ (yvec1@yvec2)] \cdot_v p) = [xvec1\ yvec1] \cdot_v [xvec2\ yvec2] \cdot_v p$
and $([(xvec1@xvec2)\ (yvec1@yvec2)]^- \cdot_v p) = [xvec2\ yvec2]^- \cdot_v [xvec1\ yvec1]^- \cdot_v p$
using *assms*
by(*induct arbitrary*: *p rule*: *composePermInduct*, *auto*) (*simp add*: *pt2*[*OF pt-name-inst*])

lemma *calcChainAtom*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and *x* :: *name*

assumes *length xvec* = *length yvec*
and $x \sharp xvec$
and $x \sharp yvec$

shows $[xvec\ yvec] \cdot_v x = x$
using *assms*
by(*induct rule*: *composePermInduct*, *auto*)

lemma *calcChainAtomRev*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and *x* :: *name*

assumes *length xvec* = *length yvec*
and $x \sharp xvec$
and $x \sharp yvec$

shows $[xvec\ yvec]^- \cdot_v x = x$
using *assms*
by(*induct rule*: *composePermInduct*, *auto*)
(*auto simp add*: *pt2*[*OF pt-name-inst*] *fresh-list-cons calc-atm*)

lemma *permChainFresh*[*simp*]:
fixes *x* :: *name*
and *xvec* :: *name list*
and *yvec* :: *name list*
and *p* :: *'a::pt-name*

assumes $x \sharp xvec$
and $x \sharp yvec$
and *length xvec* = *length yvec*

shows $x \sharp [xvec\ yvec] \cdot_v p = x \sharp p$
and $x \sharp [xvec\ yvec]^- \cdot_v p = x \sharp p$
using *assms*
by(*auto simp add*: *fresh-left calcChainAtomRev calcChainAtom*)

lemma *chainFreshFresh*:
fixes x :: *name*
and y :: *name*
and *xvec* :: *name list*
and p :: $'a$::*pt-name*

assumes $x \sharp xvec$
and $y \sharp xvec$

shows $xvec \sharp* ([(x, y)] \cdot p) = (xvec \sharp* p)$
using *assms*
by(*induct xvec*) (*auto simp add*: *fresh-list-cons fresh-left*)

lemma *permChainFreshFresh*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and p :: $'a$::*pt-name*

assumes $xvec \sharp* p$
and $yvec \sharp* p$
and *length xvec* = *length yvec*

shows $[xvec\ yvec] \cdot_v p = p$
and $[xvec\ yvec]^- \cdot_v p = p$
using *assms*
by(*induct rule*: *composePerm.induct*, *auto*) (*simp add*: *pt2*[*OF pt-name-inst*])

lemma *setFresh*[*simp*]:
fixes x :: *name*
and *xvec* :: *name list*

shows $x \notin set\ xvec = x \sharp xvec$
by(*simp add*: *name-list-supp fresh-def*)

lemma *calcChain*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*

assumes $yvec \sharp* xvec$
and *length xvec* = *length yvec*
and *distinct xvec*
and *distinct yvec*

```
  shows [xvec yvec] ·v xvec = yvec
using assms
by(induct xvec yvec rule: composePerm.induct, auto)
  (subst consPerm, simp add: calcChainAtom calc-atm name-list-supp fresh-def[symmetric])+

lemma freshChainPerm:
  fixes xvec :: name list
  and   yvec :: name list
  and   x    :: name
  and   C    :: 'a::pt-name

  assumes length xvec = length yvec
  and     yvec ♯* C
  and     xvec ♯* yvec
  and     x mem xvec
  and     distinct yvec

  shows x ♯ [xvec yvec] ·v C
using assms
proof(induct rule: composePermInduct)
  case cBase
  have x mem [] by fact
  hence False by simp
  thus ?case by simp
next
  case(cStep x' xvec y yvec)
  have (y # yvec) ♯* C by fact
  hence yFreshC: y ♯ C and yvecFreshp: yvec ♯* C by simp+
  have (x' # xvec) ♯* (y # yvec) by fact
  hence x'ineqy: x' ≠ y and xvecFreshyvec: xvec ♯* yvec
    and x'Freshyvec: x' ♯ yvec and yFreshxvec: y ♯ xvec
    by(auto simp add: fresh-list-cons)
  have distinct (y#yvec) by fact
  hence yFreshyvec: y ♯ yvec and yvecDist: distinct yvec
    by simp+
  have L: length xvec = length yvec by fact
  have x ♯ [(x', y)] · [xvec yvec] ·v C
  proof(case-tac x = x')
    assume xeqx': x = x'
    moreover from yFreshxvec yFreshyvec yFreshC L have y ♯ [xvec yvec] ·v C
      by simp
    hence ([(x, y)] · y) ♯ [(x, y)] · [xvec yvec] ·v C
      by(rule pt-fresh-bij1[OF pt-name-inst, OF at-name-inst])
    with x'ineqy xeqx' show ?thesis by(simp add: calc-atm)
  next
    assume xineqx': x ≠ x'
    have x mem (x' # xvec) by fact
    with xineqx' have xmemxvec: x mem xvec by simp
    moreover have ⟦yvec ♯* C; xvec ♯* yvec; x mem xvec; distinct yvec⟧ ⟹ x ♯
```

```
[xvec yvec] ·v C by fact
    ultimately have x ♯ [xvec yvec] ·v C using yvecFreshp xvecFreshyvec yvecDist
      by simp
    hence ([(x′, y)] · x) ♯ [(x′, y)] · [xvec yvec] ·v C
      by(rule pt-fresh-bij1[OF pt-name-inst, OF at-name-inst])
    moreover from xmemxvec yFreshxvec have x ≠ y
      by(induct xvec) (auto simp add: fresh-list-cons)
    ultimately show ?thesis using xineqx′ x′ineqy by(simp add: calc-atm)
  qed
  thus ?case by simp
qed

lemma memFreshSimp[simp]:
  fixes y    :: name
  and   yvec :: name list

  shows (¬(y mem yvec)) = y ♯ yvec
by(induct yvec)
  (auto simp add: fresh-list-nil fresh-list-cons)

lemma freshChainPerm′:
  fixes xvec :: name list
  and   yvec :: name list
  and   p    :: ′a::pt-name

  assumes length xvec = length yvec
  and     yvec ♯∗ p
  and     xvec ♯∗ yvec
  and     distinct yvec

  shows xvec ♯∗ ([xvec yvec] ·v p)
using assms
proof(induct rule: composePermInduct)
  case cBase
  show ?case by simp
next
  case(cStep x xvec y yvec)
  have (y # yvec) ♯∗ p by fact
  hence yFreshp: y ♯ p and yvecFreshp: yvec ♯∗ p
    by simp+
  moreover have (x # xvec) ♯∗ (y # yvec) by fact
  hence xineqy: x ≠ y and xvecFreshyvec: xvec ♯∗ yvec
    and xFreshyvec: x ♯ yvec and yFreshxvec: y ♯ xvec
    by(auto simp add: fresh-list-cons)
  have distinct (y # yvec) by fact
  hence yFreshyvec: y ♯ yvec and yvecDist: distinct yvec
    by simp+
  have L: length xvec = length yvec by fact
  have ⟦yvec ♯∗ p; xvec ♯∗ yvec; distinct yvec⟧ ⟹ xvec ♯∗ ([xvec yvec] ·v p) by fact
```

```
  with yvecFreshp xvecFreshyvec yvecDist have IH: xvec ♯* ([xvec yvec] ·v p) by
simp
  show ?case
  proof(auto)
   from L yFreshp yvecFreshp xineqy xvecFreshyvec yvecDist yFreshyvec yFreshxvec
xFreshyvec
    have x ♯ [(x # xvec) (y # yvec)] ·v p
      by(rule-tac freshChainPerm) (auto simp add: fresh-list-cons)
    thus x ♯ [(x, y)] · [xvec yvec] ·v p by simp
  next
    show xvec ♯* ([(x, y)] · [xvec yvec] ·v p)
    proof(case-tac x mem xvec)
      assume x mem xvec
      with L yvecFreshp xvecFreshyvec yvecDist xFreshyvec
      havex ♯ [xvec yvec] ·v p
        by(rule-tac freshChainPerm) (auto simp add: fresh-list-cons)
      moreover from yFreshxvec yFreshyvec yFreshp L
      have y ♯ [xvec yvec] ·v p by simp
      ultimately show ?thesis using IH
        by(subst consPerm) (simp add: perm-fresh-fresh)
    next
      assume ¬(x mem xvec)
      hence xFreshxvec: x ♯ xvec by simp
      from IH have ([(x, y)] · xvec) ♯* ([(x, y)] · [xvec yvec] ·v p)
        by(simp add: pt-fresh-star-bij[OF pt-name-inst, OF at-name-inst])
      with xFreshxvec yFreshxvec show ?thesis by simp
    qed
  qed
qed

lemma permSym:
  fixes x    :: name
  and   y    :: name
  and   xvec :: name list
  and   yvec :: name list
  and   p    :: 'a::pt-name

  assumes x ♯ xvec
  and     x ♯ yvec
  and     y ♯ xvec
  and     y ♯ yvec
  and     length xvec = length yvec

  shows ([(x, y)] · [xvec yvec] ·v p) = [xvec yvec] ·v [(x, y)] · p
using assms
apply(induct rule: composePerm.induct, auto)
by(subst pt-perm-compose[OF pt-name-inst, OF at-name-inst]) simp

lemma distinctPermClosed[simp]:
```

fixes p :: *name prm*
and q :: *name prm*

assumes *distinctPerm* p

shows *distinctPerm*$(q \cdot p)$
using *assms*
by(*induct* p) (*auto simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*] *dest*: *pt-bij4*[*OF pt-name-inst*, *OF at-name-inst*])

lemma *freshStarSimps*:
fixes x :: *name*
and Xs :: *name set*
and Ys :: *name set*
and C :: *'a::fs-name*
and p :: *name prm*

assumes *set* $p \subseteq Xs \times Ys$
and $Xs \sharp* x$
and $Ys \sharp* x$

shows $x \sharp (p \cdot C) = x \sharp C$
using *assms*
by(*subst pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*, *symmetric*, *of - C p*]) *simp*

lemma *freshStarChainSimps*:
fixes *xvec* :: *name list*
and Xs :: *name set*
and Ys :: *name set*
and C :: *'a::fs-name*
and p :: *name prm*

assumes *set* $p \subseteq Xs \times Ys$
and $Xs \sharp* xvec$
and $Ys \sharp* xvec$

shows $xvec \sharp* (p \cdot C) = xvec \sharp* C$
using *assms*
by(*induct xvec*) (*auto simp add*: *freshStarSimps*)

lemma *permStarFresh*:
fixes *xvec* :: *name list*
and p :: *name prm*
and T :: *'a::pt-name*

assumes $xvec \sharp* p$

shows $xvec \sharp* (p \cdot T) = xvec \sharp* T$

```
using assms
by(induct p) (auto simp add: chainFreshFresh)

lemma swapStarFresh:
  fixes x :: name
  and   p :: name prm
  and   T :: 'a::pt-name

  assumes x ♯ p

  shows x ♯ (p • T) = x ♯ T
proof −
  from assms have [x] ♯* (p • T) = [x] ♯* T
    by(rule-tac permStarFresh) auto
  thus ?thesis by simp
qed

lemmas freshChainSimps = freshStarSimps freshStarChainSimps permStarFresh
swapStarFresh chainFreshFresh freshPerm subsetFresh

lemma freshAlphaPerm:
  fixes xvec :: name list
  and   Xs   :: name set
  and   Ys   :: name set
  and   p    :: name prm

  assumes S: set p ⊆ Xs × Ys
  and     Xs ♯* xvec
  and     Ys ♯* xvec

  shows xvec ♯* p
using assms
apply(induct p)
by auto (simp add: fresh-star-def fresh-def name-list-supp supp-list-nil)+

lemma freshAlphaSwap:
  fixes x  :: name
  and   Xs :: name set
  and   Ys :: name set
  and   p  :: name prm

  assumes S: set p ⊆ Xs × Ys
  and     Xs ♯* x
  and     Ys ♯* x

  shows x ♯ p
proof −
  from assms have [x] ♯* p
    apply(rule-tac freshAlphaPerm)
```

apply *assumption*
by *auto*
thus *?thesis* **by** *simp*
qed

lemma *setToListFresh*[*simp*]:
fixes *xvec* :: *name list*
and *C* :: *'a::fs-name*
and *yvec* :: *name list*
and *Xs* :: *name set*
and *x* :: *name*

shows *xvec* $\sharp*$ (*set yvec*) = *xvec* $\sharp*$ *yvec*
and *Xs* $\sharp*$ (*set yvec*) = *Xs* $\sharp*$ *yvec*
and *x* $\sharp$ (*set yvec*) = *x* $\sharp$ *yvec*
and *set xvec* $\sharp*$ *Xs* = *xvec* $\sharp*$ *Xs*
by(*auto simp add*: *fresh-star-def name-list-supp fresh-def fs-name1 at-fin-set-supp*[*OF at-name-inst*])

end

theory *Subst-Term*
imports *Chain*
begin

locale *substType* =
fixes *subst* :: *'a::fs-name* ⇒ *name list* ⇒ *'b::fs-name list* ⇒ *'a* (‹-[-::=-]› [*80*, *80*, *80*] *130*)

assumes *eq*[*eqvt*]: ⋀*p*::*name prm*. (*p* · (*M*[*xvec*::=*Tvec*])) = ((*p* · *M*)[(*p* · *xvec*)::=(*p* · *Tvec*)])

and *subst3*: ⋀*xvec Tvec T x*. ⟦*length xvec* = *length Tvec*; *distinct xvec*; *set*(*xvec*) ⊆ *supp*(*T*); (*x*::*name*) $\sharp$ *T*[*xvec*::=*Tvec*]⟧ ⟹ *x* $\sharp$ *Tvec*

and *renaming*: ⋀*xvec Tvec p T*. ⟦*length xvec* = *length Tvec*; (*set p*) ⊆ *set xvec* × *set* (*p* · *xvec*);
distinctPerm p; (*p* · *xvec*) $\sharp*$ *T*⟧ ⟹
T[*xvec*::=*Tvec*] = (*p* · *T*)[(*p* · *xvec*)::=*Tvec*]
begin

lemma *suppSubst*:
fixes *M* :: *'a*
and *xvec* :: *name list*
and *Tvec* :: *'b list*

shows (*supp*(*M*[*xvec*::=*Tvec*])::*name set*) ⊆ ((*supp M*) ∪ (*supp xvec*) ∪ (*supp Tvec*))
proof(*auto simp add*: *eqvts supp-def*)

fix x::*name*
let $?P = \lambda y.\ ([(x,\ y)] \cdot M)[([(x,\ y)] \cdot xvec)::=([(x,\ y)] \cdot Tvec)] \neq M[xvec::=Tvec]$
let $?Q = \lambda y\ M.\ ([(x,\ y)] \cdot M) \neq (M::'a)$
let $?R = \lambda y\ xvec.\ ([(x,\ y)] \cdot xvec) \neq (xvec::name\ list)$
let $?S = \lambda y\ Tvec.\ ([(x,\ y)] \cdot Tvec) \neq (Tvec::'b\ list)$
assume A: *finite* $\{y.\ ?Q\ y\ M\}$ **and** B: *finite* $\{y.\ ?R\ y\ xvec\}$ **and** C: *finite* $\{y.\ ?S\ y\ Tvec\}$ **and** D: *infinite* $\{y.\ ?P(y)\}$
hence *infinite*$(\{y.\ ?P(y)\} - \{y.\ ?Q\ y\ M\} - \{y.\ ?R\ y\ xvec\} - \{y.\ ?S\ y\ Tvec\})$
by(*auto intro*: *Diff-infinite-finite*)
hence *infinite*$(\{y.\ ?P(y) \wedge \neg(?Q\ y\ M) \wedge \neg\ (?R\ y\ xvec) \wedge \neg\ (?S\ y\ Tvec)\})$ **by**(*simp add*: *set-diff-eq*)
moreover have $\{y.\ ?P(y) \wedge \neg(?Q\ y\ M) \wedge \neg\ (?R\ y\ xvec) \wedge \neg\ (?S\ y\ Tvec)\} = \{\}$ **by** *auto*
ultimately have *infinite* $\{\}$ **by**(*drule-tac Infinite-cong*) *auto*
thus *False* **by** *simp*
qed

lemma *subst2*[*intro*]:
fixes x :: *name*
and M :: $'a$
and $xvec$ :: *name list*
and $Tvec$ :: $'b$ *list*

assumes $x \sharp M$
and $x \sharp xvec$
and $x \sharp Tvec$

shows $x \sharp M[xvec::=Tvec]$
using *assms suppSubst*
by(*auto simp add*: *fresh-def*)

lemma *subst2Chain*[*intro*]:
fixes $yvec$ :: *name list*
and M :: $'a$
and $xvec$ :: *name list*
and $Tvec$ :: $'b$ *list*

assumes $yvec \sharp* M$
and $yvec \sharp* xvec$
and $yvec \sharp* Tvec$

shows $yvec \sharp* M[xvec::=Tvec]$
using *assms*
by(*induct yvec*) *auto*

lemma *fs*[*simp*]: *finite* ((*supp subst*)::*name set*)
by(*simp add*: *supp-def perm-fun-def eqvts*)

lemma *subst3Chain*:

fixes *xvec* :: *name list*
and *Tvec* :: *'b list*
and *Xs* :: *name set*
and *T* :: *'a*

assumes *length xvec* = *length Tvec*
and *distinct xvec*
and *set xvec* ⊆ *supp T*
and *Xs* ♯∗ *T*[*xvec*::=*Tvec*]

shows *Xs* ♯∗ *Tvec*
using *assms*
by(*auto intro*: *subst3 simp add*: *fresh-star-def*)

lemma *subst4Chain*:
fixes *xvec* :: *name list*
and *Tvec* :: *'b list*
and *T* :: *'a*

assumes *length xvec* = *length Tvec*
and *distinct xvec*
and *xvec* ♯∗ *Tvec*

shows *xvec* ♯∗ *T*[*xvec*::=*Tvec*]
proof −
obtain *p* **where** ((*p*::*name prm*) · (*xvec*::*name list*)) ♯∗ *T* **and** (*p* · *xvec*) ♯∗ *xvec*
and *S*: (*set p*) ⊆ *set xvec* × *set* (*p* · *xvec*)
and *distinctPerm p*
by(*rule-tac xvec*=*xvec* **and** *c*=(*T*, *xvec*) **in** *name-list-avoiding*) *auto*

from ‹*length xvec* = *length Tvec*› **have** *length*(*p* · *xvec*) = *length Tvec* **by** *simp*
moreover from ‹(*p* · *xvec*) ♯∗ *T*› **have** (*p* · *p* · *xvec*) ♯∗ (*p* · *T*)
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** *xvec* ♯∗ (*p* · *T*) **by** *simp*
ultimately have (*set xvec*) ♯∗ (*p* · *T*)[(*p* · *xvec*)::=*Tvec*] **using** ‹*xvec* ♯∗ *Tvec*› ‹(*p* · *xvec*) ♯∗ *xvec*›
by *auto*

thus *?thesis* **using** ‹*length xvec* = *length Tvec*› ‹*distinct xvec*› *S* ‹(*p* · *xvec*) ♯∗ *T*› ‹*distinctPerm p*›
by(*simp add*: *renaming*)
qed

definition *seqSubst* :: *'a* ⇒ (*name list* × *'b list*) *list* ⇒ *'a* (‹-[<->]› [*80*, *80*] *130*)
where *M*[<*σ*>] ≡ *foldl* (λ*N*. λ(*xvec*, *Tvec*). *N*[*xvec*::=*Tvec*]) *M σ*

lemma *seqSubstNil*[*simp*]:
seqSubst M [] = *M*
by(*simp add*: *seqSubst-def*)

lemma *seqSubstCons*[*simp*]:
shows *seqSubst M* ((*xvec*, *Tvec*)#σ) = *seqSubst*(*M*[*xvec*::=*Tvec*]) σ
by(*simp add*: *seqSubst-def*)

lemma *seqSubstTermAppend*[*simp*]:
shows *seqSubst M* (σ@σ') = *seqSubst* (*seqSubst M* σ) σ'
by(*induct* σ) (*auto simp add*: *seqSubst-def*)

definition *wellFormedSubst* :: (('*d*::*fs-name*) *list* × ('*e*::*fs-name*) *list*) *list* ⇒ *bool*
where *wellFormedSubst* σ = ((*filter* (λ(*xvec*, *Tvec*). ¬(*length xvec* = *length Tvec* ∧ *distinct xvec*)) σ) = [])

lemma *wellFormedSubstEqvt*[*eqvt*]:
fixes σ :: (('*d*::*fs-name*) *list* × ('*e*::*fs-name*) *list*) *list*
and *p* :: *name prm*

shows *p* • (*wellFormedSubst* σ) = *wellFormedSubst*(*p* • σ)
by(*induct* σ *arbitrary*: *p*) (*auto simp add*: *eqvts wellFormedSubst-def*)

lemma *wellFormedSimp*[*simp*]:
fixes σ :: (('*d*::*fs-name*) *list* × ('*e*::*fs-name*) *list*) *list*
and *p* :: *name prm*

shows *wellFormedSubst*(*p* • σ) = *wellFormedSubst* σ
by(*induct* σ) (*auto simp add*: *eqvts wellFormedSubst-def*)

lemma *wellFormedNil*[*simp*]:
wellFormedSubst []
by(*simp add*: *wellFormedSubst-def*)

lemma *wellFormedCons*[*simp*]:
shows *wellFormedSubst*((*xvec*, *Tvec*)#σ) = (*length xvec* = *length Tvec* ∧ *distinct xvec* ∧ *wellFormedSubst* σ)
by(*simp add*: *wellFormedSubst-def*) *auto*

lemma *wellFormedAppend*[*simp*]:
fixes σ :: (('*d*::*fs-name*) *list* × ('*e*::*fs-name*) *list*) *list*
and σ' :: (('*d*::*fs-name*) *list* × ('*e*::*fs-name*) *list*) *list*

shows *wellFormedSubst*(σ@σ') = (*wellFormedSubst* σ ∧ *wellFormedSubst* σ')
by(*simp add*: *wellFormedSubst-def*)

lemma *seqSubst2*[*intro*]:
fixes σ :: (*name list* × '*b list*) *list*
and *T* :: '*a*
and *x* :: *name*

assumes *x* ♯ σ

```
  and     x ♯ T

  shows x ♯ T[<σ>]
using assms
by(induct σ arbitrary: T) (clarsimp | blast)+

lemma seqSubst2Chain[intro]:
  fixes σ    :: (name list × 'b list) list
  and   T    :: 'a
  and   xvec :: name list

  assumes xvec ♯* σ
  and     xvec ♯* T

  shows xvec ♯* T[<σ>]
using assms
by(induct xvec) auto

end

end

theory Agent
  imports Subst-Term
begin

nominal-datatype ('term, 'assertion, 'condition) psi =
  PsiNil (‹0› 190)


| Output 'term::fs-name 'term ('term, 'assertion::fs-name, 'condition::fs-name) psi
(‹-⟨-⟩.-› [120, 120, 110] 110)
| Input 'term ('term, 'assertion, 'condition) input                    (‹-⦇-›
[120, 120] 110)
| Case (('term, 'assertion, 'condition) psiCase)                       (‹Case
-› [120] 120)
| Par ('term, 'assertion, 'condition) psi ('term, 'assertion, 'condition) psi      (infixl
‹‖› 90)
| Res «name»(('term, 'assertion, 'condition) psi)                      (‹⦇ν-⦈-›
[120, 120] 110)
| Assert 'assertion                                                    (‹⦃-⦄›
[120] 120)
| Bang ('term, 'assertion, 'condition) psi                             (‹!-›
[110] 110)

and ('term, 'assertion, 'condition) input =
  Trm 'term (('term, 'assertion, 'condition) psi)                       (‹⦈-.-›
[130, 130] 130)
| Bind «name»(('term, 'assertion, 'condition) input)                   (‹ν--›
```

[*120*, *120*] *120*)

and (*'term*, *'assertion*, *'condition*) *psiCase* =
  EmptyCase (‹$\bot_c$› *120*)
| *Cond 'condition* ((*'term*, *'assertion*, *'condition*) *psi*)
  ((*'term*, *'assertion*, *'condition*) *psiCase*) (‹□
- ⇒ - - › [*120*, *120*, *120*] *120*)

lemma *psiFreshSet*[*simp*]:
  fixes X :: *name set*
  and M :: *'a*::*fs-name*
  and N :: *'a*
  and P :: (*'a*, *'b*::*fs-name*, *'c*::*fs-name*) *psi*
  and I :: (*'a*, *'b*, *'c*) *input*
  and C :: (*'a*, *'b*, *'c*) *psiCase*
  and Q :: (*'a*, *'b*, *'c*) *psi*
  and x :: *name*
  and Ψ :: *'b*
  and Φ :: *'c*

  shows $X \sharp* (M\langle N\rangle.P) = (X \sharp* M \wedge X \sharp* N \wedge X \sharp* P)$
  and $X \sharp* M(\!|I = (X \sharp* M \wedge X \sharp* I)$
  and $X \sharp* Case\ C = X \sharp* C$
  and $X \sharp* (P \parallel Q) = (X \sharp* P \wedge X \sharp* Q)$
  and $X \sharp* (\!|\nu x|\!)P = (X \sharp* [x].P)$
  and $X \sharp* \{\!|\Psi|\!\} = X \sharp* \Psi$
  and $X \sharp* !P = X \sharp* P$
  and $X \sharp* \mathbf{0}$
  and $X \sharp* Trm\ N\ P = (X \sharp* N \wedge X \sharp* P)$
  and $X \sharp* Bind\ x\ I = X \sharp* ([x].I)$

  and $X \sharp* \bot_c$
  and $X \sharp* \Box\ \Phi \Rightarrow P\ C = (X \sharp* \Phi \wedge X \sharp* P \wedge X \sharp* C)$
by(*auto simp add*: *fresh-star-def psi.fresh*)+

lemma *psiFreshVec*[*simp*]:
  fixes *xvec* :: *name list*

  shows $xvec \sharp* (M\langle N\rangle.P) = (xvec \sharp* M \wedge xvec \sharp* N \wedge xvec \sharp* P)$
  and $xvec \sharp* M(\!|I = (xvec \sharp* M \wedge xvec \sharp* I)$
  and $xvec \sharp* Case\ C = xvec \sharp* C$
  and $xvec \sharp* (P \parallel Q) = (xvec \sharp* P \wedge xvec \sharp* Q)$
  and $xvec \sharp* (\!|\nu x|\!)P = (xvec \sharp* [x].P)$
  and $xvec \sharp* \{\!|\Psi|\!\} = xvec \sharp* \Psi$
  and $xvec \sharp* !P = xvec \sharp* P$
  and $xvec \sharp* \mathbf{0}$

  and $xvec \sharp* Trm\ N\ P = (xvec \sharp* N \wedge xvec \sharp* P)$
  and $xvec \sharp* Bind\ x\ I = xvec \sharp* ([x].I)$

and $xvec \sharp* \bot_c$
and $xvec \sharp* \Box\ \Phi \Rightarrow P\ C = (xvec \sharp* \Phi \wedge xvec \sharp* P \wedge xvec \sharp* C)$
by(*auto simp add*: *fresh-star-def*)

fun *psiCases* :: ('*c*::*fs-name* × ('*a*::*fs-name*, '*b*::*fs-name*, '*c*) *psi*) *list* ⇒ ('*a*, '*b*, '*c*) *psiCase*
where
base: *psiCases* [] = $\bot_c$
| *step*: *psiCases* ((Φ, *P*)#*xs*) = *Cond* Φ *P* (*psiCases xs*)

lemma *psiCasesEqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *Cs* :: ('*c*::*fs-name* × ('*a*::*fs-name*, '*b*::*fs-name*, '*c*) *psi*) *list*

shows (*p* • (*psiCases Cs*)) = *psiCases*(*p* • *Cs*)
by(*induct Cs*) *auto*

lemma *psiCasesFresh*[*simp*]:
fixes *x* :: *name*
and *Cs* :: ('*c*::*fs-name* × ('*a*::*fs-name*, '*b*::*fs-name*, '*c*) *psi*) *list*

shows *x* ♯ *psiCases Cs* = *x* ♯ *Cs*
by(*induct Cs*)
(*auto simp add*: *fresh-list-nil fresh-list-cons*)

lemma *psiCasesFreshChain*[*simp*]:
fixes *xvec* :: *name list*
and *Cs* :: ('*c*::*fs-name* × ('*a*::*fs-name*, '*b*::*fs-name*, '*c*) *psi*) *list*
and *Xs* :: *name set*

shows (*xvec* ♯* *psiCases Cs*) = *xvec* ♯* *Cs*
and (*Xs* ♯* *psiCases Cs*) = *Xs* ♯* *Cs*
by(*auto simp add*: *fresh-star-def*)

abbreviation
psiCasesJudge (‹*Cases* -› [*80*] *80*) **where** *Cases Cs* ≡ *Case*(*psiCases Cs*)

primrec *resChain* :: *name list* ⇒ ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi* ⇒ ('*a*, '*b*, '*c*) *psi* **where**
base: *resChain* [] *P* = *P*
| *step*: *resChain* (*x*#*xs*) *P* = ⦇ν*x*⦈(*resChain xs P*)

notation *resChain* (‹⦇ν*-⦈-› [*80*, *80*] *80*)

lemma *resChainEqvt*[*eqvt*]:
fixes *perm* :: *name prm*
and *lst* :: *name list*
and *P* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

shows $perm \cdot ((\!|\nu{*}xvec|\!)P) = (\!|\nu{*}(perm \cdot xvec)|\!)(perm \cdot P)$
by(*induct-tac xvec*, *auto*)

lemma *resChainSupp*:
fixes $xvec$ :: *name list*
and P :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $supp((\!|\nu{*}xvec|\!)P) = (supp\ P) - set\ xvec$
by(*induct xvec*) (*auto simp add*: *psi.supp abs-supp*)

lemma *resChainFresh*:
fixes x :: *name*
and $xvec$ :: *name list*
and P :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $x \sharp (\!|\nu{*}xvec|\!)P = (x \in set\ xvec \vee x \sharp P)$
by (*induct xvec*) (*simp-all add*: *abs-fresh*)

lemma *resChainFreshSet*:
fixes Xs :: *name set*
and $xvec$ :: *name list*
and $yvec$ :: *name list*
and P :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $Xs \sharp{*} ((\!|\nu{*}xvec|\!)P) = (\forall x{\in}Xs.\ x \in set\ xvec \vee x \sharp P)$
and $yvec \sharp{*} ((\!|\nu{*}xvec|\!)P) = (\forall x{\in}(set\ yvec).\ x \in set\ xvec \vee x \sharp P)$
by (*simp add*: *fresh-star-def resChainFresh*)+

lemma *resChainFreshSimps*[*simp*]:
fixes Xs :: *name set*
and $xvec$ :: *name list*
and P :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psi*
and $yvec$ :: *name list*

shows $Xs \sharp{*}\ xvec \Longrightarrow Xs \sharp{*} ((\!|\nu{*}xvec|\!)P) = (Xs \sharp{*}\ P)$
and $yvec \sharp{*}\ xvec \Longrightarrow yvec \sharp{*} ((\!|\nu{*}xvec|\!)P) = (yvec \sharp{*}\ P)$
and $xvec \sharp{*} ((\!|\nu{*}xvec|\!)P)$
apply(*simp add*: *resChainFreshSet*) **apply**(*force simp add*: *fresh-star-def name-list-supp fresh-def*)
apply(*simp add*: *resChainFreshSet*) **apply**(*force simp add*: *fresh-star-def name-list-supp fresh-def*)
by(*simp add*: *resChainFreshSet*)

lemma *resChainAlpha*:
fixes p :: *name prm*
and $xvec$ :: *name list*
and P :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

assumes *xvecFreshP*: $(p \cdot xvec) \sharp* P$
and *S*: $set\ p \subseteq set\ xvec \times set\ (p \cdot xvec)$

shows $(\!|\nu*xvec|\!)P = (\!|\nu*(p \cdot xvec)|\!)(p \cdot P)$
proof −
note *pt-name-inst at-name-inst S*
moreover have $set\ xvec \sharp* ((\!|\nu*xvec|\!)P)$
by (*simp add*: *resChainFreshSet*)
moreover from *xvecFreshP* **have** $set\ (p \cdot xvec) \sharp* ((\!|\nu*xvec|\!)P)$
by (*simp add*: *resChainFreshSet*) (*simp add*: *fresh-star-def*)
ultimately have $(\!|\nu*xvec|\!)P = p \cdot ((\!|\nu*xvec|\!)P)$
by (*rule-tac pt-freshs-freshs* [*symmetric*])
then show *?thesis* **by**(*simp add*: *eqvts*)
qed

lemma *resChainAppend*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and *P* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

shows $(\!|\nu*(xvec@yvec)|\!)P = (\!|\nu*xvec|\!)((\!|\nu*yvec|\!)P)$
by(*induct xvec*) *auto*

lemma *resChainSimps*[*dest*]:
fixes *xvec* :: *name list*
and *P* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
and *Q* :: ('*a*, '*b*, '*c*) *psi*
and *P'* :: ('*a*, '*b*, '*c*) *psi*
and *Q'* :: ('*a*, '*b*, '*c*) *psi*

shows $((\!|\nu*xvec|\!)(P \parallel Q)) = P' \parallel Q') \Longrightarrow (P = P' \wedge Q = Q')$
and $(P \parallel Q = (\!|\nu*xvec|\!)(P' \parallel Q')) \Longrightarrow (P = P' \wedge Q = Q')$
by(*case-tac xvec*, *simp-all add*: *psi.inject*)+

primrec *inputChain* :: *name list* ⇒ '*a*::*fs-name* ⇒ ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi* ⇒ ('*a*, '*b*, '*c*) *input* **where**
base: $inputChain\ []\ N\ P = |\!)(N).P$
| *step*: $inputChain\ (x\#xs)\ N\ P = \nu\ x\ (inputChain\ xs\ N\ P)$

abbreviation
inputChainJudge (‹-(|λ*- -|).-› [*80*, *80*, *80*, *80*] *80*) **where** $M(\!|\lambda*xvec\ N|\!).P \equiv M(\!|(inputChain\ xvec\ N\ P)$

lemma *inputChainEqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *xvec* :: *name list*
and *N* :: '*a*::*fs-name*
and *P* :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

shows $p \cdot (inputChain\ xvec\ N\ P) = inputChain\ (p \cdot xvec)\ (p \cdot N)\ (p \cdot P)$
by(*induct-tac xvec*) *auto*

lemma *inputChainFresh*:
fixes x :: *name*
and *xvec* :: *name list*
and N :: $'a$::*fs-name*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $x \sharp (inputChain\ xvec\ N\ P) = (x \in set\ xvec \vee (x \sharp N \wedge x \sharp P))$
by (*induct xvec*) (*simp-all add*: *abs-fresh*)

lemma *inductChainSimps*[*simp*]:
fixes *xvec* :: *name list*
and N :: $'a$::*fs-name*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $xvec \sharp* (inputChain\ xvec\ N\ P)$
by(*induct xvec*) (*auto simp add*: *abs-fresh abs-fresh-star fresh-star-def*)

lemma *inputChainFreshSet*:
fixes *Xs* :: *name set*
and *xvec* :: *name list*
and N :: $'a$::*fs-name*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $Xs \sharp* (inputChain\ xvec\ N\ P) = (\forall x \in Xs.\ x \in set\ xvec \vee (x \sharp N \wedge x \sharp P))$
by (*simp add*: *fresh-star-def inputChainFresh*)

lemma *inputChainAlpha*:
fixes p :: *name prm*
and *Xs* :: *name set*
and *Ys* :: *name set*

assumes *XsFreshP*: $Xs \sharp* (inputChain\ xvec\ N\ P)$
and *YsFreshN*: $Ys \sharp* N$
and *YsFreshP*: $Ys \sharp* P$
and *S*: $set\ p \subseteq Xs \times Ys$

shows $(inputChain\ xvec\ N\ P) = (inputChain\ (p \cdot xvec)\ (p \cdot N)\ (p \cdot P))$
proof −
note *pt-name-inst at-name-inst XsFreshP S*
moreover from *YsFreshN YsFreshP* **have** $Ys \sharp* (inputChain\ xvec\ N\ P)$
by (*simp add*: *inputChainFreshSet*) (*simp add*: *fresh-star-def*)
ultimately have $(inputChain\ xvec\ N\ P) = p \cdot (inputChain\ xvec\ N\ P)$
by (*rule-tac pt-freshs-freshs* [*symmetric*])
then show *?thesis* **by**(*simp add*: *eqvts*)
qed

```
lemma inputChainAlpha':
  fixes p    :: name prm
  and   xvec :: name list
  and   N    :: 'a::fs-name
  and   P    :: ('a, 'b::fs-name, 'c::fs-name) psi

  assumes xvecFreshP: (p · xvec) ♯* P
  and     xvecFreshN: (p · xvec) ♯* N
  and     S: set p ⊆ set xvec × set (p · xvec)

  shows (inputChain xvec N P) = (inputChain (p · xvec) (p · N) (p · P))
proof −
  note pt-name-inst at-name-inst S
  moreover have set xvec ♯* (inputChain xvec N P)
    by (simp add: inputChainFreshSet)
  ultimately show ?thesis using xvecFreshN xvecFreshP
    by(rule-tac inputChainAlpha) (simp add: fresh-star-def)+
qed

lemma alphaRes:
  fixes M :: 'a::fs-name
  and   x :: name
  and   P :: ('a, 'b::fs-name, 'c::fs-name) psi
  and   y :: name

  assumes yFreshP: y ♯ P

  shows ⦇νx⦈P = ⦇νy⦈([(x, y)] · P)
proof(cases x = y)
  assume x=y
  thus ?thesis by simp
next
  assume x ≠ y
  with yFreshP show ?thesis
    by(perm-simp add: psi.inject alpha calc-atm fresh-left)
qed

lemma alphaInput:
  fixes x :: name
  and   I :: ('a::fs-name, 'b::fs-name, 'c::fs-name) input
  and   c :: name

  assumes A1: c ♯ I

  shows ν x I = ν c([(x, c)] · I)
proof(cases x = c)
  assume x=c
  thus ?thesis by simp
next
```

assume $x \neq c$
with *A1* **show** *?thesis*
by(*perm-simp add*: *input.inject alpha calc-atm fresh-left*)
qed

lemma *inputChainLengthEq*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and *M* :: *'a::fs-name*
and *P* :: (*'a*, *'b::fs-name*, *'c::fs-name*) *psi*

assumes *length xvec* = *length yvec*
and *xvec* ♯∗ *yvec*
and *distinct yvec*
and *yvec* ♯∗ *M*
and *yvec* ♯∗ *P*

obtains *N Q* **where** *inputChain xvec M P* = *inputChain yvec N Q*
proof −
assume ⋀*N Q*. *inputChain xvec M P* = *inputChain yvec N Q* ⟹ *thesis*
moreover obtain *n* **where** *n* = *length xvec* **by** *auto*
with *assms* **have** ∃ *N Q*. *inputChain xvec M P* = *inputChain yvec N Q*
proof(*induct n arbitrary*: *xvec yvec M P*)
case *0*
thus *?case* **by** *auto*
next
case(*Suc n xvec yvec M P*)
from ‹*Suc n* = *length xvec*›
obtain *x xvec'* **where** *xvec* = *x*#*xvec'* **and** *length xvec'* = *n*
by(*case-tac xvec*) *auto*
with ‹*length xvec* = *length yvec*›
obtain *y yvec'* **where** *yvec* = *y*#*yvec'* **by**(*case-tac yvec*) *auto*
from ‹*yvec* = *y*#*yvec'*› ‹*xvec*=*x*#*xvec'*› ‹*xvec* ♯∗ *yvec*› ‹*distinct yvec*› ‹*length xvec* = *length yvec*› ‹*yvec* ♯∗ *M*› ‹*yvec* ♯∗ *P*›
have *length xvec'* = *length yvec'* **and** *xvec'* ♯∗ *yvec'* **and** *distinct yvec'* **and** *yvec'* ♯∗ *M* **and** *yvec'* ♯∗ *P*
by *simp+*
then obtain *N Q* **where** *Eq*: *inputChain xvec' M P* = *inputChain yvec' N Q* **using** ‹*length xvec'* = *n*›
by(*drule-tac Suc*) *auto*
moreover from ‹*distinct yvec*› ‹*yvec* = *y*#*yvec'*› **have** *y* ♯ *yvec'* **by** *auto*
moreover from ‹*xvec* ♯∗ *yvec*› ‹*xvec* = *x*#*xvec'*› ‹*yvec*=*y*#*yvec'*› **have** $x \neq y$ **and** *x* ♯ *yvec'*
by *auto*
moreover from ‹*yvec* ♯∗ *M*› ‹*yvec* ♯∗ *P*› ‹*yvec* = *y*#*yvec'*› **have** *y* ♯ *M* **and** *y* ♯ *P* **by** *auto*
hence *y* ♯ *inputChain xvec' M P* **by**(*simp add*: *inputChainFresh*)
with *Eq* **have** *y* ♯ *inputChain yvec' N Q* **by**(*simp add*: *inputChainFresh*)
ultimately have *ν x* (*inputChain xvec' M P*) = *ν y* (*inputChain yvec'* ([(*x*,

$y)] \cdot N)$ $([(x, y)] \cdot Q))$
by(*simp add*: *input.inject alpha′ eqvts name-swap*)
thus *?case* **using** ‹*xvec* = *x*#*xvec′*› ‹*yvec*=*y*#*yvec′*› **by** *force*
qed
ultimately show *?thesis*
by *blast*
qed

lemma *inputChainEq*:
fixes *xvec* :: *name list*
and *M* :: *′a*::*fs-name*
and *P* :: (*′a*, *′b*::*fs-name*, *′c*::*fs-name*) *psi*
and *yvec* :: *name list*
and *N* :: *′a*
and *Q* :: (*′a*, *′b*, *′c*) *psi*

assumes *inputChain xvec M P* = *inputChain yvec N Q*
and *xvec* ♯∗ *yvec*
and *distinct xvec*
and *distinct yvec*

obtains *p* **where** (*set p*) ⊆ (*set xvec*) × *set* (*p* · *xvec*) **and** *distinctPerm p* **and** *yvec* = *p* · *xvec* **and** *N* = *p* · *M* **and** *Q* = *p* · *P*
proof −
assume ⋀*p*. ⟦*set p* ⊆ *set xvec* × *set* (*p* · *xvec*); *distinctPerm p*; *yvec* = *p* · *xvec*; *N* = *p* · *M*; *Q* = *p* · *P*⟧ ⟹ *thesis*
moreover obtain *n* **where** *n* = *length xvec* **by** *auto*
with *assms* **have** ∃*p*. (*set p*) ⊆ (*set xvec*) × *set* (*yvec*) ∧ *distinctPerm p* ∧ *yvec* = *p* · *xvec* ∧ *N* = *p* · *M* ∧ *Q* = *p* · *P*
proof(*induct n arbitrary*: *xvec yvec M N P Q*)
case(*0 xvec yvec M N P Q*)
have *Eq*: *inputChain xvec M P* = *inputChain yvec N Q* **by** *fact*
from ‹*0* = *length xvec*› **have** *xvec* = [] **by** *auto*
moreover with *Eq* **have** *yvec* = []
by(*case-tac yvec*) *auto*
ultimately show *?case* **using** *Eq*
by(*simp add*: *input.inject*)
next
case(*Suc n xvec yvec M N P Q*)
from ‹*Suc n* = *length xvec*›
obtain *x xvec′* **where** *xvec* = *x*#*xvec′* **and** *length xvec′* = *n*
by(*case-tac xvec*) *auto*
from ‹*inputChain xvec M P* = *inputChain yvec N Q*› ‹*xvec* = *x* # *xvec′*›
obtain *y yvec′* **where** *inputChain* (*x*#*xvec′*) *M P* = *inputChain* (*y*#*yvec′*) *N Q*
and *yvec* = *y*#*yvec′*
by(*case-tac yvec*) *auto*
hence *EQ*: *ν x* (*inputChain xvec′ M P*) = *ν y* (*inputChain yvec′ N Q*)
by *simp*

from ‹*xvec* = *x*#*xvec*′› ‹*yvec*=*y*#*yvec*′› ‹*xvec* ♯∗ *yvec*›
have $x \neq y$ **and** *xvec*′ ♯∗ *yvec*′ **and** *x* ♯ *yvec*′ **and** *y* ♯ *xvec*′
by(*auto simp add*: *fresh-list-cons*)
from ‹*distinct xvec*› ‹*distinct yvec*› ‹*xvec*=*x*#*xvec*′› ‹*yvec*=*y*#*yvec*′› **have** *x* ♯ *xvec*′ **and** *y* ♯ *yvec*′ **and** *distinct xvec*′ **and** *distinct yvec*′
by *simp*+
have *IH*: $\bigwedge$*xvec yvec M N P Q*. ⟦*inputChain xvec* (*M*::′*a*) (*P*::(′*a*, ′*b*, ′*c*) *psi*) = *inputChain yvec* (*N*::′*a*) (*Q*::(′*a*, ′*b*, ′*c*) *psi*); *xvec* ♯∗ *yvec*; *distinct xvec*; *distinct yvec*; *n* = *length xvec*⟧ $\Longrightarrow$ ∃ *p*. (*set p*) ⊆ (*set xvec*) × (*set yvec*) ∧ *distinctPerm p* ∧ *yvec* = *p* · *xvec* ∧ *N* = *p* · *M* ∧ *Q* = *p* · *P*
by *fact*
from *EQ* ‹$x \neq y$› ‹*x* ♯ *yvec*′› ‹*y* ♯ *yvec*′› **have** *inputChain xvec*′ *M P* = *inputChain yvec*′ ([(*x*, *y*)] · *N*) ([(*x*, *y*)] · *Q*)
by(*simp add*: *input.inject alpha eqvts*)
with ‹*xvec*′ ♯∗ *yvec*′› ‹*distinct xvec*′› ‹*distinct yvec*′› ‹*length xvec*′ = *n*› *IH*
obtain *p* **where** *S*: (*set p*) ⊆ (*set xvec*′) × (*set yvec*′) **and** *distinctPerm p* **and** *yvec*′ = *p* · *xvec*′ **and** ([(*x*, *y*)] · *N*) = *p* · *M* **and** ([(*x*, *y*)] · *Q*) = *p* · *P*
by *metis*
from *S* **have** *set*((*x*, *y*)#*p*) ⊆ *set*(*x*#*xvec*′) × *set*(*y*#*yvec*′) **by** *auto*
moreover from ‹*x* ♯ *xvec*′› ‹*x* ♯ *yvec*′› ‹*y* ♯ *xvec*′› ‹*y* ♯ *yvec*′› *S* **have** *x* ♯ *p* **and** *y* ♯ *p*
apply(*induct p*)
by(*auto simp add*: *fresh-list-nil fresh-list-cons fresh-prod name-list-supp*) (*auto simp add*: *fresh-def*)

with *S* ‹*distinctPerm p*› ‹$x \neq y$› **have** *distinctPerm*((*x*, *y*)#*p*) **by** *auto*
moreover from ‹*yvec*′ = *p* · *xvec*′› ‹*x* ♯ *p*› ‹*y* ♯ *p*› ‹*x* ♯ *xvec*′› ‹*y* ♯ *xvec*′› **have** (*y*#*yvec*′) = ((*x*, *y*)#*p*) · (*x*#*xvec*′)
by(*simp add*: *calc-atm freshChainSimps*)
moreover from ‹([(*x*, *y*)] · *N*) = *p* · *M*› **have** ([(*x*, *y*)] · [(*x*, *y*)] · *N*) = [(*x*, *y*)] · *p* · *M*
by(*simp add*: *pt-bij*)
hence *N* = ((*x*, *y*)#*p*) · *M* **by** *simp*
moreover from ‹([(*x*, *y*)] · *Q*) = *p* · *P*› **have** ([(*x*, *y*)] · [(*x*, *y*)] · *Q*) = [(*x*, *y*)] · *p* · *P*
by(*simp add*: *pt-bij*)
hence *Q* = ((*x*, *y*)#*p*) · *P* **by** *simp*
ultimately show *?case* **using** ‹*xvec*=*x*#*xvec*′› ‹*yvec*=*y*#*yvec*′›
by *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *inputChainEqLength*:
fixes *xvec* :: *name list*
and *M* :: ′*a*::*fs-name*
and *P* :: (′*a*, ′*b*::*fs-name*, ′*c*::*fs-name*) *psi*
and *yvec* :: *name list*
and *N* :: ′*a*

```
  and   Q   :: ('a, 'b, 'c) psi

  assumes inputChain xvec M P = inputChain yvec N Q

  shows length xvec = length yvec
proof −
  obtain n where n = length xvec by auto
  with assms show ?thesis
  proof(induct n arbitrary: xvec yvec M P N Q)
    case(0 xvec yvec M P N Q)
    from ‹0 = length xvec› have xvec = [] by auto
    moreover with ‹inputChain xvec M P = inputChain yvec N Q› have yvec =
[]
      by(case-tac yvec) auto
    ultimately show ?case by simp
  next
    case(Suc n xvec yvec M P N Q)
    from ‹Suc n = length xvec›
    obtain x xvec' where xvec = x#xvec' and length xvec' = n
      by(case-tac xvec) auto
    from ‹inputChain xvec M P = inputChain yvec N Q› ‹xvec = x # xvec'›
    obtain y yvec' where inputChain (x#xvec') M P = inputChain (y#yvec') N
Q
      and yvec = y#yvec'
      by(case-tac yvec) auto
    hence EQ: ν x (inputChain xvec' M P) = ν y (inputChain yvec' N Q)
      by simp
    have IH: ⋀xvec yvec M P N Q. ⟦inputChain xvec (M::'a) (P::('a, 'b, 'c) psi)
= inputChain yvec N Q; n = length xvec⟧ ⟹ length xvec = length yvec
      by fact
    show ?case
    proof(case-tac x = y)
      assume x = y
      with EQ have inputChain xvec' M P = inputChain yvec' N Q
        by(simp add: alpha input.inject)
      with IH ‹length xvec' = n› have length xvec' = length yvec'
        by blast
      with ‹xvec = x#xvec'› ‹yvec=y#yvec'›
      show ?case by simp
    next
      assume x ≠ y
      with EQ have inputChain xvec' M P = inputChain ([(x, y)] • yvec') ([(x, y)]
• N) ([(x, y)] • Q)
        by(simp add: alpha input.inject eqvts)
      with IH ‹length xvec' = n› have length xvec' = length ([(x, y)] • yvec')
        by blast
      hence length xvec' = length yvec'
        by simp
      with ‹xvec = x#xvec'› ‹yvec=y#yvec'›
```

```
      show ?case by simp
    qed
  qed
qed

lemma alphaInputChain:
  fixes yvec :: name list
  and   xvec :: name list
  and   M    :: 'a::fs-name
  and   P    :: ('a, 'b::fs-name, 'c::fs-name) psi

  assumes length xvec = length yvec
  and     yvec ♯* M
  and     yvec ♯* P
  and     yvec ♯* xvec
  and     distinct yvec

  shows inputChain xvec M P = inputChain yvec ([xvec yvec] ·v M) ([xvec yvec]
·v P)
using assms
proof(induct rule: composePermInduct)
  case cBase
  show ?case by simp
next
  case(cStep x xvec y yvec)
  thus ?case
    apply auto
    by(subst alphaInput[of y]) (auto simp add: inputChainFresh eqvts)
qed

lemma inputChainInject[simp]:

  shows (inputChain xvec M P = inputChain xvec N Q) = ((M = N) ∧ (P = Q))
by(induct xvec) (auto simp add: input.inject alpha)

lemma alphaInputDistinct:
  fixes xvec :: name list
  and   M    :: 'a::fs-name
  and   P    :: ('a, 'b::fs-name, 'c::fs-name) psi
  and   yvec :: name list
  and   N    :: 'a
  and   Q    :: ('a, 'b, 'c) psi

  assumes Eq: inputChain xvec M P = inputChain yvec N Q
  and     xvecDist: distinct xvec
  and     Mem: ⋀x. x ∈ set xvec ⟹ x ∈ supp M
  and     xvecFreshyvec: xvec ♯* yvec
  and     xvecFreshN: xvec ♯* N
  and     xvecFreshQ: xvec ♯* Q
```

shows *distinct yvec*
proof −
from *Eq* **have** *length xvec* = *length yvec*
by(*rule inputChainEqLength*)
with *assms* **show** *?thesis*
proof(*induct n*==*length xvec arbitrary*: *xvec yvec N Q rule*: *nat.induct*)
case(*zero xvec yvec N Q*)
thus *?case* **by** *simp*
next
case(*Suc n xvec yvec N Q*)
have *L*: *length xvec* = *length yvec* **and** *Suc n* = *length xvec* **by** *fact*+
then obtain *x xvec′ y yvec′* **where** *xEq*: *xvec* = *x*#*xvec′* **and** *yEq*: *yvec* = *y*#*yvec′*
and *L′*: *length xvec′* = *length yvec′*
by(*cases xvec*, *auto*, *cases yvec*, *auto*)
have *xvecFreshyvec*: *xvec* ♯∗ *yvec* **and** *xvecDist*: *distinct xvec* **by** *fact*+
with *xEq yEq* **have** *xineqy*: *x* ≠ *y* **and** *xvec′Freshyvec′*: *xvec′* ♯∗ *yvec′*
and *xvec′Dist*: *distinct xvec′* **and** *xFreshxvec′*: *x* ♯ *xvec′*
and *xFreshyvec′*: *x* ♯ *yvec′* **and** *yFreshxvec′*: *y* ♯ *xvec′*
by(*auto simp add*: *fresh-list-cons*)
have *Eq*: *inputChain xvec M P* = *inputChain yvec N Q* **by** *fact*
with *xEq yEq xineqy* **have** *Eq′*: *inputChain xvec′ M P* = *inputChain* ([(*x*, *y*)] · *yvec′*) ([(*x*, *y*)] · *N*) ([(*x*, *y*)] · *Q*)
by(*simp add*: *input.inject alpha eqvts*)
moreover have *Mem*:⋀*x*. *x* ∈ *set xvec* ⟹ *x* ∈ *supp M* **by** *fact*
with *xEq* **have** ⋀*x*. *x* ∈ *set xvec′* ⟹ *x* ∈ *supp M* **by** *simp*
moreover have *xvecFreshN*: *xvec* ♯∗ *N* **by** *fact*
with *xEq xFreshxvec′ yFreshxvec′* **have** *xvec′* ♯∗ ([(*x*, *y*)] · *N*) **by** *simp*
moreover have *xvecFreshQ*: *xvec* ♯∗ *Q* **by** *fact*
with *xEq xFreshxvec′ yFreshxvec′* **have** *xvec′* ♯∗ ([(*x*, *y*)] · *Q*) **by** *simp*
moreover have *Suc n* = *length xvec* **by** *fact*
with *xEq* **have** *n* = *length xvec′* **by** *simp*
moreover from *xvec′Freshyvec′ xFreshxvec′ yFreshxvec′* **have** *xvec′* ♯∗ ([(*x*, *y*)] · *yvec′*)
by *simp*
moreover from *L′* **have** *length xvec′* = *length*([(*x*, *y*)] · *yvec′*) **by** *simp*
ultimately have *distinct*([(*x*, *y*)] · *yvec′*) **using** *xvec′Dist*
by(*rule-tac Suc*)
hence *distinct yvec′* **by** *simp*
from *Mem xEq* **have** *xSuppM*: *x* ∈ *supp M* **by** *simp*
from *L xvecFreshyvec xvecDist xvecFreshN xvecFreshQ*
have *inputChain yvec N Q* = *inputChain xvec* ([*yvec xvec*] $\cdot_v$ *N*) ([*yvec xvec*] $\cdot_v$ *Q*)
by(*simp add*: *alphaInputChain*)
with *Eq* **have** *M* = [*yvec xvec*] $\cdot_v$ *N* **by** *auto*
with *xEq yEq* **have** *M* = [(*y*, *x*)] · [*yvec′ xvec′*] $\cdot_v$ *N*
by *simp*
with *xSuppM* **have** *ySuppN*: *y* ∈ *supp*([*yvec′ xvec′*] $\cdot_v$ *N*)

```
      by(drule-tac pi=[(x, y)] in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
        (simp add: calc-atm eqvts name-swap)
    have y ♯ yvec'
    proof(simp add: fresh-def, rule notI)
      assume y ∈ supp yvec'
      hence y mem yvec'
        by(induct yvec') (auto simp add: supp-list-nil supp-list-cons supp-atm)
      moreover from xvecFreshN xEq xFreshxvec' have xvec' ♯* N by simp
      ultimately have y ♯ [yvec' xvec'] •v  N using L' xvec'Freshyvec' xvec'Dist
        by(force intro: freshChainPerm simp add: freshChainSym)
      with ySuppN show False by(simp add: fresh-def)
    qed
    with ‹distinct yvec'›  yEq show ?case by simp
  qed
qed

lemma psiCasesInject[simp]:
  fixes CsP  :: ('c::fs-name × ('a::fs-name, 'b::fs-name, 'c) psi) list
  and   CsQ  :: ('c × ('a, 'b, 'c) psi) list

  shows (psiCases CsP = psiCases CsQ) = (CsP = CsQ)
proof(induct CsP arbitrary: CsQ)
  case(Nil CsQ)
  thus ?case by(case-tac CsQ) (auto)
next
  case(Cons a CsP CsQ)
  thus ?case
    by(case-tac a, case-tac CsQ) (clarsimp simp add: psiCase.inject)+
qed

lemma casesInject[simp]:
  fixes CsP :: ('c::fs-name × ('a::fs-name, 'b::fs-name, 'c) psi) list
  and   CsQ :: ('c × ('a, 'b, 'c) psi) list

  shows (Cases CsP = Cases CsQ) = (CsP = CsQ)
apply(induct CsP)
apply(auto simp add: psiCase.inject)
apply(case-tac CsQ)
apply(simp add: psiCase.inject psi.inject)
apply(force simp add: psiCase.inject psi.inject)
apply(case-tac CsQ)
apply(force simp add: psiCase.inject psi.inject)
apply(auto simp add: psiCase.inject psi.inject)
apply(simp only: psiCases.simps[symmetric])
apply(simp only: psiCasesInject)
apply simp
apply(case-tac CsQ)
by(auto simp add: psiCase.inject psi.inject)
```

nominal-primrec
  guarded :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi* ⇒ *bool*
  and *guarded'* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *input* ⇒ *bool*
  and *guarded''* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psiCase* ⇒ *bool*

where
  guarded (**0**) = *True*
| *guarded* ($M\langle N\rangle.P$) = *True*
| *guarded* ($M(\!|I$) = *True*
| *guarded* (*Case C*) = *guarded''* *C*
| *guarded* ($P \parallel Q$) = ((*guarded P*) ∧ (*guarded Q*))
| *guarded* ($(\!|\nu x|\!)P$) = (*guarded P*)
| *guarded* ($\{\!|\Psi|\!\}$) = *False*
| *guarded* ($!P$) = *guarded P*

| *guarded'* (*Trm M P*) = *False*
| *guarded'* (ν *y I*) = *False*

| *guarded''* ($\bot_c$) = *True*
| *guarded''* ($\Box\varphi \Rightarrow P\ C$) = (*guarded P* ∧ *guarded''* *C*)
apply(*finite-guess*)+
apply(*rule TrueI*)+
by(*fresh-guess add*: *fresh-bool*)+

lemma *guardedEqvt*[*eqvt*]:
  fixes *p* :: *name prm*
  and *P* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
  and *I* :: ('*a*, '*b*, '*c*) *input*
  and *C* :: ('*a*, '*b*, '*c*) *psiCase*

  shows ($p \cdot$ (*guarded P*)) = *guarded* ($p \cdot P$)
  and ($p \cdot$ (*guarded' I*)) = *guarded'* ($p \cdot I$)
  and ($p \cdot$ (*guarded'' C*)) = *guarded''* ($p \cdot C$)
by(*nominal-induct P* **and** *I* **and** *C rule*: *psi-input-psiCase.strong-inducts*)
  (*simp add*: *eqvts*)+

lemma *guardedClosed*[*simp*]:
  fixes *P* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
  and *p* :: *name prm*

  assumes *guarded P*

  shows *guarded*($p \cdot P$)
proof −
  from ‹*guarded P*› **have** $p \cdot$ (*guarded P*)
    by(*simp add*: *perm-bool*)
  thus *?thesis* **by**(*simp add*: *eqvts*)
qed

locale *substPsi* =
  substTerm?: *substType substTerm* +
  substAssert?: *substType substAssert* +
  substCond?: *substType substCond*

  for *substTerm* :: ('*a*::*fs-name*) ⇒ *name list* ⇒ '*a*::*fs-name list* ⇒ '*a*
  and *substAssert* :: ('*b*::*fs-name*) ⇒ *name list* ⇒ '*a*::*fs-name list* ⇒ '*b*
  and *substCond* :: ('*c*::*fs-name*) ⇒ *name list* ⇒ '*a*::*fs-name list* ⇒ '*c*
begin

nominal-primrec
  subs :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi* ⇒ *name list* ⇒ '*a list* ⇒ ('*a*, '*b*, '*c*) *psi*
and *subs'* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *input* ⇒ *name list* ⇒ '*a list* ⇒ ('*a*, '*b*, '*c*) *input*
and *subs''* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psiCase* ⇒ *name list* ⇒ '*a list* ⇒ ('*a*, '*b*, '*c*) *psiCase*

where
  subs (**0**) *xvec Tvec* = **0**
| (*subs* (*M*⟨*N*⟩.*P*) *xvec Tvec*) = (*substTerm M xvec Tvec*)⟨(*substTerm N xvec Tvec*)⟩.(*subs P xvec Tvec*)
| (*subs* (*M*⦇*I*) *xvec Tvec*) = (*substTerm M xvec Tvec*)⦇(*subs' I xvec Tvec*)

| (*subs* (*Case C*) *xvec Tvec*) = (*Case* (*subs'' C xvec Tvec*))
| (*subs* (*P* ∥ *Q*) *xvec Tvec*) = (*subs P xvec Tvec*) ∥ (*subs Q xvec Tvec*)
| ⟦*y* ♯ *xvec*; *y* ♯ *Tvec*⟧ ⟹ (*subs* ((⦇*νy*⦈*P*) *xvec Tvec*) = (⦇*νy*⦈(*subs P xvec Tvec*)
| (*subs* (⦃Ψ⦄) *xvec Tvec*) = ⦃(*substAssert* Ψ *xvec Tvec*)⦄
| (*subs* (!*P*) *xvec Tvec*) = !(*subs P xvec Tvec*)

| (*subs'* ((*Trm M P*)::('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *input*) *xvec Tvec*) = (⦈(*substTerm M xvec Tvec*).(*subs P xvec Tvec*))
| ⟦*y* ♯ *xvec*; *y* ♯ *Tvec*⟧ ⟹ (*subs'* (*ν y I*) *xvec Tvec*) = (*ν y* (*subs' I xvec Tvec*))

| (*subs''* ($\bot_c$::('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psiCase*) *xvec Tvec*) = $\bot_c$
| (*subs''* (□Φ ⇒ *P C*) *xvec Tvec*) = (□(*substCond* Φ *xvec Tvec*) ⇒ (*subs P xvec Tvec*) (*subs'' C xvec Tvec*))
apply(*finite-guess add*: *substTerm.fs substAssert.fs substCond.fs*)+
apply(*rule TrueI*)+
apply(*simp add*: *abs-fresh*)
apply(*simp add*: *abs-fresh*)
apply(*simp add*: *abs-fresh*)
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)

apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
apply(*rule supports-fresh*[*of supp*(*xvec*, *Tvec*)])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
done

lemma *substEqvt*[*eqvt*]:
  fixes *p* :: *name prm*
  and *P* :: ('*a*, '*b*, '*c*) *psi*
  and *xvec* :: *name list*
  and *Tvec* :: '*a list*
  and *I* :: ('*a*, '*b*, '*c*) *input*
  and *C* :: ('*a*, '*b*, '*c*) *psiCase*

shows $(p \cdot (subs\ P\ xvec\ Tvec)) = subs\ (p \cdot P)\ (p \cdot xvec)\ (p \cdot Tvec)$
and $(p \cdot (subs'\ I\ xvec\ Tvec)) = subs'\ (p \cdot I)\ (p \cdot xvec)\ (p \cdot Tvec)$
and $(p \cdot (subs''\ C\ xvec\ Tvec)) = subs''\ (p \cdot C)\ (p \cdot xvec)\ (p \cdot Tvec)$
apply(*nominal-induct P* **and** *I* **and** *C avoiding*: *xvec Tvec rule*: *psi-input-psiCase.strong-inducts*)
apply(*auto simp add*: *eqvts*)
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply *simp*
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
by *simp*

lemma *subst2*[*intro*]:
fixes $xvec :: name\ list$
and $Tvec :: 'a\ list$
and $x :: name$
and $P :: ('a, 'b, 'c)\ psi$
and $I :: ('a, 'b, 'c)\ input$
and $C :: ('a, 'b, 'c)\ psiCase$

assumes $x \sharp Tvec$
and $x \sharp xvec$

shows $x \sharp P \Longrightarrow x \sharp (subs\ P\ xvec\ Tvec)$
and $x \sharp I \Longrightarrow x \sharp (subs'\ I\ xvec\ Tvec)$
and $x \sharp C \Longrightarrow x \sharp (subs''\ C\ xvec\ Tvec)$
using *assms*
by(*nominal-induct P* **and** *I* **and** *C avoiding*: *xvec Tvec rule*: *psi-input-psiCase.strong-inducts*)
(*auto intro*: *substTerm.subst2 substCond.subst2 substAssert.subst2 simp add*: *abs-fresh*)

lemma *subst2Chain*[*intro*]:
fixes $xvec :: name\ list$
and $Tvec :: 'a\ list$
and $Xs :: name\ set$
and $P :: ('a, 'b, 'c)\ psi$
and $I :: ('a, 'b, 'c)\ input$
and $C :: ('a, 'b, 'c)\ psiCase$

assumes $Xs \sharp* xvec$
and $Xs \sharp* Tvec$

shows $Xs \sharp* P \Longrightarrow Xs \sharp* (subs\ P\ xvec\ Tvec)$
and $Xs \sharp* I \Longrightarrow Xs \sharp* (subs'\ I\ xvec\ Tvec)$
and $Xs \sharp* C \Longrightarrow Xs \sharp* (subs''\ C\ xvec\ Tvec)$
using *assms*
by(*auto intro*: *subst2 simp add*: *fresh-star-def*)

lemma *renaming*:

fixes *xvec* :: *name list*
and *Tvec* :: *'a list*
and *p* :: *name prm*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *I* :: (*'a* ,*'b*, *'c*) *input*
and *C* :: (*'a*, *'b*, *'c*) *psiCase*

assumes *length xvec* = *length Tvec*
and *set p* ⊆ *set xvec* × *set* (*p* · *xvec*)
and *distinctPerm p*

shows ⟦(*p* · *xvec*) ♯∗ *P*⟧ ⟹ (*subs P xvec Tvec*) = *subs* (*p* · *P*) (*p* · *xvec*) *Tvec*
and ⟦(*p* · *xvec*) ♯∗ *I*⟧ ⟹ (*subs' I xvec Tvec*) = *subs'* (*p* · *I*) (*p* · *xvec*) *Tvec*
and ⟦(*p* · *xvec*) ♯∗ *C*⟧ ⟹ (*subs'' C xvec Tvec*) = *subs''* (*p* · *C*) (*p* · *xvec*) *Tvec*
using *assms*
by(*nominal-induct P* **and** *I* **and** *C avoiding*: *xvec p Tvec rule*: *psi-input-psiCase.strong-inducts*)
(*auto intro*: *substTerm.renaming substCond.renaming substAssert.renaming simp add*: *freshChainSimps psi.inject input.inject psiCase.inject*)

lemma *subst4Chain*:
fixes *xvec* :: *name list*
and *Tvec* :: *'a list*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *I* :: (*'a*, *'b*, *'c*) *input*
and *C* :: (*'a*, *'b*, *'c*) *psiCase*

assumes *length xvec* = *length Tvec*
and *distinct xvec*
and *xvec* ♯∗ *Tvec*

shows *xvec* ♯∗ (*subs P xvec Tvec*)
and *xvec* ♯∗ (*subs' I xvec Tvec*)
and *xvec* ♯∗ (*subs'' C xvec Tvec*)
using *assms*
by(*nominal-induct P* **and** *I* **and** *C avoiding*: *xvec Tvec rule*: *psi-input-psiCase.strong-inducts*)
(*auto intro*: *substTerm.subst4Chain substCond.subst4Chain substAssert.subst4Chain simp add*: *abs-fresh*)

lemma *guardedSubst*[*simp*]:
fixes *P* :: (*'a*, *'b*, *'c*) *psi*
and *I* :: (*'a*, *'b*, *'c*) *input*
and *C* :: (*'a*, *'b*, *'c*) *psiCase*
and *xvec* :: *name list*
and *Tvec* :: *'a list*

assumes *length xvec* = *length Tvec*
and *distinct xvec*

shows *guarded P* ⟹ *guarded*(*subs P xvec Tvec*)

and *guarded′ I* ⟹ *guarded′*(*subs′ I xvec Tvec*)
and *guarded″ C* ⟹ *guarded″*(*subs″ C xvec Tvec*)
using *assms*
by(*nominal-induct P* **and** *I* **and** *C avoiding*: *xvec Tvec rule*: *psi-input-psiCase.strong-inducts*) *auto*

definition *seqSubs* :: (′*a*, ′*b*, ′*c*) *psi* ⇒ (*name list* × ′*a list*) *list* ⇒ (′*a*, ′*b*, ′*c*) *psi* (‹-[<->]› [*80*, *80*] *130*)
where *P*[<*σ*>] ≡ *foldl* (λ*Q*. λ(*xvec*, *Tvec*). *subs Q xvec Tvec*) *P σ*

definition *seqSubs′* :: (′*a*, ′*b*, ′*c*) *input* ⇒ (*name list* × ′*a list*) *list* ⇒ (′*a*, ′*b*, ′*c*) *input*
where *seqSubs′ I σ* ≡ *foldl* (λ*Q*. λ(*xvec*, *Tvec*). *subs′ Q xvec Tvec*) *I σ*

definition *seqSubs″* :: (′*a*, ′*b*, ′*c*) *psiCase* ⇒ (*name list* × ′*a list*) *list* ⇒ (′*a*, ′*b*, ′*c*) *psiCase*
where *seqSubs″ C σ* ≡ *foldl* (λ*Q*. λ(*xvec*, *Tvec*). *subs″ Q xvec Tvec*) *C σ*

lemma *substInputChain*[*simp*]:
fixes *xvec* :: *name list*
and *N* :: ′*a*
and *P* :: (′*a*, ′*b*, ′*c*) *psi*
and *yvec* :: *name list*
and *Tvec* :: ′*a list*

assumes *xvec* ♯∗ *yvec*
and *xvec* ♯∗ *Tvec*

shows *subs′* (*inputChain xvec N P*) *yvec Tvec* = *inputChain xvec* (*substTerm N yvec Tvec*) (*subs P yvec Tvec*)
using *assms*
by(*induct xvec*) (*auto simp add*: *psi.inject*)

fun *caseListSubst* :: (′*c* × (′*a*, ′*b*, ′*c*) *psi*) *list* ⇒ *name list* ⇒ ′*a list* ⇒ (′*c* × (′*a*, ′*b*, ′*c*) *psi*) *list*
where
caseListSubst [] - - = []
| *caseListSubst* ((*φ*, *P*)#*Cs*) *xvec Tvec* = (*substCond φ xvec Tvec*, (*subs P xvec Tvec*))#(*caseListSubst Cs xvec Tvec*)

lemma *substCases*[*simp*]:
fixes *Cs* :: (′*c* × (′*a*, ′*b*, ′*c*) *psi*) *list*
and *xvec* :: *name list*
and *Tvec* :: ′*a list*

shows *subs* (*Cases Cs*) *xvec Tvec* = *Cases*(*caseListSubst Cs xvec Tvec*)
by(*induct Cs*) (*auto simp add*: *psi.inject*)

lemma *substCases′*[*simp*]:

fixes Cs :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and $xvec$ :: $name\ list$
and $Tvec$ :: $'a\ list$

shows $(subs''\ (psiCases\ Cs)\ xvec\ Tvec) = psiCases(caseListSubst\ Cs\ xvec\ Tvec)$
by$(induct\ Cs)\ auto$

lemma $seqSubstSimps[simp]$:
shows $seqSubs\ (\mathbf{0})\ \sigma = \mathbf{0}$
and $(seqSubs\ (M\langle N\rangle.P)\ \sigma) = (substTerm.seqSubst\ M\ \sigma)\langle(substTerm.seqSubst\ N\ \sigma)\rangle.(seqSubs\ P\ \sigma)$
and $(seqSubs\ (M(\!|I)\ \sigma) = (substTerm.seqSubst\ M\ \sigma)(\!|(seqSubs'\ I\ \sigma)$

and $(seqSubs\ (Case\ C)\ \sigma) = (Case\ (seqSubs''\ C\ \sigma))$
and $(seqSubs\ (P \parallel Q)\ \sigma) = (seqSubs\ P\ \sigma) \parallel (seqSubs\ Q\ \sigma)$
and $[\![y \sharp \sigma]\!] \Longrightarrow (seqSubs\ ((\!|\nu y|\!)P)\ \sigma) = (\!|\nu y|\!)(seqSubs\ P\ \sigma)$
and $(seqSubs\ (\{\!|\Psi|\!\})\ \sigma) = \{\!|(substAssert.seqSubst\ \Psi\ \sigma)|\!\}$
and $(seqSubs\ (!P)\ \sigma) = !(seqSubs\ P\ \sigma)$

and $(seqSubs'\ ((Trm\ M\ P)::('a::fs\text{-}name, 'b::fs\text{-}name, 'c::fs\text{-}name)\ input)\ \sigma) = (|)(substTerm.seqSubst\ M\ \sigma).(seqSubs\ P\ \sigma))$
and $[\![y \sharp \sigma]\!] \Longrightarrow (seqSubs'\ (\nu\ y\ I)\ \sigma) = (\nu\ y\ (seqSubs'\ I\ \sigma))$

and $(seqSubs''\ (\bot_c::('a::fs\text{-}name, 'b::fs\text{-}name, 'c::fs\text{-}name)\ psiCase)\ \sigma) = \bot_c$
and $(seqSubs''\ (\Box\Phi \Rightarrow P\ C)\ \sigma) = (\Box(substCond.seqSubst\ \Phi\ \sigma) \Rightarrow (seqSubs\ P\ \sigma)\ (seqSubs''\ C\ \sigma))$
by$(induct\ \sigma\ arbitrary: M\ N\ P\ I\ C\ Q\ \Psi\ \Phi, auto\ simp\ add: seqSubs\text{-}def\ seqSubs'\text{-}def\ seqSubs''\text{-}def)$

lemma $seqSubsNil[simp]$:
$seqSubs\ P\ [] = P$
by$(simp\ add: seqSubs\text{-}def)$

lemma $seqSubsCons[simp]$:
shows $seqSubs\ P\ ((xvec, Tvec)\#\sigma) = seqSubs(subs\ P\ xvec\ Tvec)\ \sigma$
by$(simp\ add: seqSubs\text{-}def)$

lemma $seqSubsTermAppend[simp]$:
shows $seqSubs\ P\ (\sigma@\sigma') = seqSubs\ (seqSubs\ P\ \sigma)\ \sigma'$
by$(induct\ \sigma)\ (auto\ simp\ add: seqSubs\text{-}def)$

fun $caseListSeqSubst$ :: $('c \times ('a, 'b, 'c)\ psi)\ list \Rightarrow (name\ list \times 'a\ list)\ list \Rightarrow ('c \times ('a, 'b, 'c)\ psi)\ list$
where
$caseListSeqSubst\ []\ \text{-} = []$
$|\ caseListSeqSubst\ ((\varphi, P)\#Cs)\ \sigma = (substCond.seqSubst\ \varphi\ \sigma, (seqSubs\ P\ \sigma))\#(caseListSeqSubst\ Cs\ \sigma)$

lemma $seqSubstCases[simp]$:

fixes Cs :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and σ :: $(name\ list \times 'a\ list)\ list$

shows $seqSubs\ (Cases\ Cs)\ \sigma = Cases(caseListSeqSubst\ Cs\ \sigma)$
by(*induct Cs*) (*auto simp add*: *psi.inject*)

lemma *seqSubstCases'*[*simp*]:
fixes Cs :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and σ :: $(name\ list \times 'a\ list)\ list$

shows $(seqSubs''\ (psiCases\ Cs)\ \sigma) = psiCases(caseListSeqSubst\ Cs\ \sigma)$
by(*induct Cs*) *auto*

lemma *seqSubstEqvt*[*eqvt*]:
fixes P :: $('a, 'b, 'c)\ psi$
and σ :: $(name\ list \times 'a\ list)\ list$
and p :: $name\ prm$

shows $(p \cdot (P[<\sigma>])) = (p \cdot P)[<(p \cdot \sigma)>]$
by(*induct* σ *arbitrary*: P) (*auto simp add*: *eqvts seqSubs-def*)

lemma *guardedSeqSubst*:
assumes *guarded P*
and *wellFormedSubst* σ

shows $guarded(seqSubs\ P\ \sigma)$
using *assms*
by(*induct* σ *arbitrary*: P) (*auto dest*: *guardedSubst*)

end

lemma *inter-eqvt*:
shows $(pi::name\ prm) \cdot ((X::name\ set) \cap Y) = (pi \cdot X) \cap (pi \cdot Y)$
by(*auto simp add*: *perm-set-def perm-bij*)

lemma *delete-eqvt*:
fixes p :: $name\ prm$
and X :: $name\ set$
and Y :: $name\ set$

shows $p \cdot (X - Y) = (p \cdot X) - (p \cdot Y)$
by(*auto simp add*: *perm-set-def perm-bij*)

lemma *perm-singleton*[*simp*]:
shows $(p::name\ prm) \cdot \{(x::name)\} = \{p \cdot x\}$
by(*auto simp add*: *perm-set-def*)

end

theory *Frame*
  imports *Agent*
begin

lemma *permLength*[*simp*]:
  fixes p :: *name prm*
  and *xvec* :: *'a::pt-name list*

  shows *length*($p \cdot$ *xvec*) = *length xvec*
by(*induct xvec*) *auto*

nominal-datatype *'assertion frame* =
    FAssert 'assertion::fs-name
  | *FRes* «*name*» (*'assertion frame*) (‹⦇ν-⦈-› [*80*, *80*] *80*)

primrec *frameResChain* :: *name list* ⇒ (*'a::fs-name*) *frame* ⇒ *'a frame* **where**
  base: *frameResChain* [] F = F
| *step*: *frameResChain* (x#*xs*) F = ⦇νx⦈(*frameResChain xs* F)

notation *frameResChain* (‹⦇ν*-⦈-› [*80*, *80*] *80*)
notation *FAssert* (‹⟨ε, -⟩› [*80*] *80*)
abbreviation *FAssertJudge* (‹⟨-, -⟩› [*80*, *80*] *80*) **where** $\langle A_F, \Psi_F\rangle \equiv$ *frameResChain* A_F (*FAssert* Ψ_F)

lemma *frameResChainEqvt*[*eqvt*]:
  fixes *perm* :: *name prm*
  and *lst* :: *name list*
  and F :: *'a::fs-name frame*

  shows *perm* $\cdot$ (⦇ν*xvec⦈F) = ⦇ν*(*perm* $\cdot$ *xvec*)⦈(*perm* $\cdot$ F)
by(*induct-tac xvec*, *auto*)

lemma *frameResChainFresh*:
  fixes x :: *name*
  and *xvec* :: *name list*
  and F :: *'a::fs-name frame*

  shows $x \sharp$ ⦇ν*xvec⦈F = ($x \in$ *set xvec* ∨ $x \sharp F$)
by (*induct xvec*) (*simp-all add*: *abs-fresh*)

lemma *frameResChainFreshSet*:
  fixes *Xs* :: *name set*
  and *xvec* :: *name list*
  and F :: *'a::fs-name frame*

  shows *Xs* ♯* (⦇ν*xvec⦈F) = (∀ $x \in$*Xs*. $x \in$ *set xvec* ∨ $x \sharp F$)
by (*simp add*: *fresh-star-def frameResChainFresh*)

lemma *frameChainAlpha*:

fixes p :: *name prm*
and *xvec* :: *name list*
and F :: *'a::fs-name frame*

assumes *xvecFreshF*: $(p \cdot xvec) \sharp* F$
and S: $set\ p \subseteq set\ xvec \times set\ (p \cdot xvec)$

shows $(\!|\nu{*}xvec|\!)F = (\!|\nu{*}(p \cdot xvec)|\!)(p \cdot F)$
proof −
note *pt-name-inst at-name-inst S*
moreover have $set\ xvec \sharp* ((\!|\nu{*}xvec|\!)F)$
by (*simp add*: *frameResChainFreshSet*)
moreover from *xvecFreshF* **have** $set\ (p \cdot xvec) \sharp* ((\!|\nu{*}xvec|\!)F)$
by (*simp add*: *frameResChainFreshSet*) (*simp add*: *fresh-star-def*)
ultimately have $(\!|\nu{*}xvec|\!)F = p \cdot ((\!|\nu{*}xvec|\!)F)$
by (*rule-tac pt-freshs-freshs* [*symmetric*])
then show *?thesis* **by**(*simp add*: *eqvts*)
qed

lemma *frameChainAlpha'*:
fixes p :: *name prm*
and A_P :: *name list*
and Ψ_P :: *'a::fs-name*

assumes $(p \cdot A_P) \sharp* \Psi_P$
and S: $set\ p \subseteq set\ A_P \times set\ (p \cdot A_P)$

shows $\langle A_P, \Psi_P\rangle = \langle (p \cdot A_P), p \cdot \Psi_P\rangle$
using *assms*
by(*subst frameChainAlpha*) (*auto simp add*: *fresh-star-def*)

lemma *alphaFrameRes*:
fixes x :: *name*
and F :: *'a::fs-name frame*
and y :: *name*

assumes $y \sharp F$

shows $(\!|\nu x|\!)F = (\!|\nu y|\!)([(x, y)] \cdot F)$
proof(*cases* $x = y$)
assume $x{=}y$
thus *?thesis* **by** *simp*
next
assume $x \neq y$
with ‹$y \sharp F$› **show** *?thesis*
by(*perm-simp add*: *frame.inject alpha calc-atm fresh-left*)
qed

lemma *frameChainAppend*:

```
  fixes xvec :: name list
  and   yvec :: name list
  and   F    :: 'a::fs-name frame

  shows (|ν*(xvec@yvec)|)F = (|ν*xvec|)((|ν*yvec|)F)
by(induct xvec) auto

lemma frameChainEqLength:
  fixes xvec :: name list
  and   Ψ    :: 'a::fs-name
  and   yvec :: name list
  and   Ψ'   :: 'a::fs-name

  assumes ⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩

  shows length xvec = length yvec
proof −
  obtain n where n = length xvec by auto
  with assms show ?thesis
  proof(induct n arbitrary: xvec yvec Ψ Ψ')
    case(0 xvec yvec Ψ Ψ')
    from ‹0 = length xvec› have xvec = [] by auto
    moreover with ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩› have yvec = []
      by(case-tac yvec) auto
    ultimately show ?case by simp
  next
    case(Suc n xvec yvec Ψ Ψ')
    from ‹Suc n = length xvec›
    obtain x xvec' where xvec = x#xvec' and length xvec' = n
      by(case-tac xvec) auto
    from ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩› ‹xvec = x # xvec'›
    obtain y yvec' where ⟨(x#xvec'), Ψ⟩ = ⟨(y#yvec'), Ψ'⟩
      and yvec = y#yvec'
      by(case-tac yvec) auto
    hence EQ: (|νx|)(|ν*xvec'|)(FAssert Ψ) = (|νy|)(|ν*yvec'|)(FAssert Ψ')
      by simp
    have IH: ⋀xvec yvec Ψ Ψ'. ⟦⟨xvec, (Ψ::'a)⟩ = ⟨yvec, (Ψ'::'a)⟩; n = length xvec⟧
⟹ length xvec = length yvec
      by fact
    show ?case
    proof(case-tac x = y)
      assume x = y
      with EQ have ⟨xvec', Ψ⟩ = ⟨yvec', Ψ'⟩
        by(simp add: alpha frame.inject)
      with IH ‹length xvec' = n› have length xvec' = length yvec'
        by blast
      with ‹xvec = x#xvec'› ‹yvec=y#yvec'›
      show ?case by simp
    next
```

```
      assume x ≠ y
      with EQ have ⟨xvec', Ψ⟩ = [(x, y)] • ⟨yvec', Ψ'⟩
        by(simp add: alpha frame.inject)
      hence ⟨xvec', Ψ⟩ = ⟨([(x, y)] • yvec'), ([(x, y)] • Ψ')⟩
        by(simp add: eqvts)
      with IH ‹length xvec' = n› have length xvec' = length ([(x, y)] • yvec')
        by blast
      hence length xvec' = length yvec'
        by simp
      with ‹xvec = x#xvec'› ‹yvec=y#yvec'›
      show ?case by simp
    qed
  qed
qed

lemma frameEqFresh:
  fixes F :: ('a::fs-name) frame
  and   G :: 'a frame
  and   x :: name
  and   y :: name

  assumes ⦇νx⦈F = ⦇νy⦈G
  and     x ♯ F

  shows y ♯ G
using assms
by(auto simp add: frame.inject alpha fresh-left calc-atm)

lemma frameEqSupp:
  fixes F :: ('a::fs-name) frame
  and   G :: 'a frame
  and   x :: name
  and   y :: name

  assumes ⦇νx⦈F = ⦇νy⦈G
  and     x ∈ supp F

  shows y ∈ supp G
using assms
apply(auto simp add: frame.inject alpha fresh-left calc-atm)
apply(drule-tac pi=[(x, y)] in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
by(simp add: eqvts calc-atm)

lemma frameChainEqSuppEmpty[dest]:
  fixes xvec :: name list
  and   Ψ    :: 'a::fs-name
  and   yvec :: name list
  and   Ψ'   :: 'a::fs-name
```

```
  assumes ⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩
  and     supp Ψ = ({}::name set)

  shows Ψ = Ψ'
proof −
  obtain n where n = length xvec by auto
  with assms show ?thesis
  proof(induct n arbitrary: xvec yvec Ψ Ψ')
    case(0 xvec yvec Ψ Ψ')
    from ‹0 = length xvec› have xvec = [] by auto
    moreover with ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩› have yvec = []
      by(case-tac yvec) auto
    ultimately show ?case  using ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩›
      by(simp add: frame.inject)
  next
    case(Suc n xvec yvec Ψ Ψ')
    from ‹Suc n = length xvec›
    obtain x xvec' where xvec = x#xvec' and length xvec' = n
      by(case-tac xvec) auto
    from ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩› ‹xvec = x # xvec'›
    obtain y yvec' where ⟨(x#xvec'), Ψ⟩ = ⟨(y#yvec'), Ψ'⟩
      and yvec = y#yvec'
      by(case-tac yvec) auto
    hence EQ: ⦇νx⦈⦇ν*xvec'⦈(FAssert Ψ) = ⦇νy⦈⦇ν*yvec'⦈(FAssert Ψ')
      by simp
      have IH: ⋀xvec yvec Ψ Ψ'. ⟦⟨xvec, (Ψ::'a)⟩ = ⟨yvec, (Ψ'::'a)⟩; supp Ψ =
({}::name set); n = length xvec⟧ ⟹ Ψ = Ψ'
      by fact
    show ?case
    proof(case-tac x = y)
      assume x = y
      with EQ have ⟨xvec', Ψ⟩ = ⟨yvec', Ψ'⟩
        by(simp add: alpha frame.inject)
      with IH ‹length xvec' = n› ‹supp Ψ = {}› show ?case
        by simp
    next
      assume x ≠ y
      with EQ have ⟨xvec', Ψ⟩ = [(x, y)] • ⟨yvec', Ψ'⟩
        by(simp add: alpha frame.inject)
      hence ⟨xvec', Ψ⟩ = ⟨([(x, y)] • yvec'), ([(x, y)] • Ψ')⟩
        by(simp add: eqvts)
      with IH ‹length xvec' = n› ‹supp Ψ = {}› have Ψ = [(x, y)] • Ψ'
        by(simp add: eqvts)
      moreover with ‹supp Ψ = {}› have supp([(x, y)] • Ψ') = ({}::name set)
        by simp
      hence x ♯ ([(x, y)] • Ψ') and y ♯ ([(x, y)] • Ψ')
        by(simp add: fresh-def)+
      with ‹x ≠ y› have x ♯ Ψ' and y ♯ Ψ'
        by(simp add: fresh-left calc-atm)+
```

```
      ultimately show ?case by simp
    qed
  qed
qed

lemma frameChainEq:
  fixes xvec :: name list
  and   Ψ    :: 'a::fs-name
  and   yvec :: name list
  and   Ψ'   :: 'a::fs-name

  assumes ⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩
  and     xvec ♯* yvec

  obtains p where (set p) ⊆ (set xvec) × set (yvec) and distinctPerm p and Ψ'
= p • Ψ
proof −
  assume ⋀p. ⟦set p ⊆ set xvec × set yvec; distinctPerm p; Ψ' = p • Ψ⟧ ⟹ thesis
  moreover obtain n where n = length xvec by auto
  with assms have ∃p. (set p) ⊆ (set xvec) × set (yvec) ∧ distinctPerm p ∧  Ψ'
= p • Ψ
  proof(induct n arbitrary: xvec yvec Ψ Ψ')
    case(0 xvec yvec Ψ Ψ')
    have Eq: ⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩ by fact
    from ‹0 = length xvec› have xvec = [] by auto
    moreover with Eq have yvec = []
      by(case-tac yvec) auto
    ultimately show ?case using Eq
      by(simp add: frame.inject)
  next
    case(Suc n xvec yvec Ψ Ψ')
    from ‹Suc n = length xvec›
    obtain x xvec' where xvec = x#xvec' and length xvec' = n
      by(case-tac xvec) auto
    from ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩› ‹xvec = x # xvec'›
    obtain y yvec' where ⟨(x#xvec'), Ψ⟩ = ⟨(y#yvec'), Ψ'⟩
      and yvec = y#yvec'
      by(case-tac yvec) auto
    hence EQ: ⦇νx⦈⦇ν*xvec'⦈(FAssert Ψ) = ⦇νy⦈⦇ν*yvec'⦈(FAssert Ψ')
      by simp
    from ‹xvec = x#xvec'› ‹yvec=y#yvec'› ‹xvec ♯* yvec›
    have x ≠ y and xvec' ♯* yvec' and x ♯ yvec' and y ♯ xvec'
      by auto
    have IH: ⋀xvec yvec Ψ Ψ'. ⟦⟨xvec, (Ψ::'a)⟩ = ⟨yvec, (Ψ'::'a)⟩; xvec ♯* yvec; n
= length xvec⟧ ⟹
                                  ∃p. (set p) ⊆ (set xvec) × (set yvec) ∧ distinctPerm p
∧  Ψ' = p • Ψ
      by fact
```

from EQ ‹$x \neq y$› **have** EQ': $\langle xvec', \Psi\rangle = ([(x, y)] \cdot \langle yvec', \Psi'\rangle)$
and $xFresh\Psi'$: $x \sharp (\!|\nu{*}yvec'|\!)(FAssert\ \Psi')$
by(*simp add*: *frame.inject alpha*)+

show *?case*
proof(*case-tac* $x \sharp \langle xvec', \Psi\rangle$)
assume $x \sharp \langle xvec', \Psi\rangle$
with EQ **have** $y \sharp \langle yvec', \Psi'\rangle$
by(*rule frameEqFresh*)
with $xFresh\Psi'$ EQ' **have** $\langle xvec', \Psi\rangle = \langle yvec', \Psi'\rangle$
by(*simp*)
with ‹$xvec' \sharp{*}\ yvec'$› ‹$length\ xvec' = n$› *IH*
obtain p **where** S: $(set\ p) \subseteq (set\ xvec') \times (set\ yvec')$ **and** $distinctPerm\ p$ **and** $\Psi' = p \cdot \Psi$
by *blast*
from S **have** $(set\ p) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
with ‹$xvec = x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹$distinctPerm\ p$› ‹$\Psi' = p \cdot \Psi$›
show *?case* **by** *blast*
next
assume $\neg(x \sharp (\!|\nu{*}xvec'|\!)(FAssert\ \Psi))$
hence $xSupp\Psi$: $x \in supp(\langle xvec', \Psi\rangle)$
by(*simp add*: *fresh-def*)
with EQ **have** $y \in supp\ (\langle yvec', \Psi'\rangle)$
by(*rule frameEqSupp*)
hence $y \sharp yvec'$
by(*induct yvec'*) (*auto simp add*: *frame.supp abs-supp*)
with ‹$x \sharp yvec'$› EQ' **have** $\langle xvec', \Psi\rangle = \langle yvec', ([(x, y)] \cdot \Psi')\rangle$
by(*simp add*: *eqvts*)
with ‹$xvec' \sharp{*}\ yvec'$› ‹$length\ xvec' = n$› *IH*
obtain p **where** S: $(set\ p) \subseteq (set\ xvec') \times (set\ yvec')$ **and** $distinctPerm\ p$ **and** $([(x, y)] \cdot \Psi') = p \cdot \Psi$
by *blast*

from $xSupp\Psi$ **have** $x \sharp xvec'$
by(*induct xvec'*) (*auto simp add*: *frame.supp abs-supp*)
with ‹$x \sharp yvec'$› ‹$y \sharp xvec'$› ‹$y \sharp yvec'$› S **have** $x \sharp p$ **and** $y \sharp p$
apply(*induct p*)
by(*auto simp add*: *name-list-supp*) (*auto simp add*: *fresh-def*)
from S **have** $(set\ ((x, y)\#p)) \subseteq (set(x\#xvec')) \times (set(y\#yvec'))$
by *force*
moreover from ‹$x \neq y$› ‹$x \sharp p$› ‹$y \sharp p$› S ‹$distinctPerm\ p$›
have $distinctPerm((x,y)\#p)$ **by** *simp*
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp xvec'$› ‹$y \sharp xvec'$› **have** $y\#(p \cdot xvec') = ((x, y)\#p) \cdot (x\#xvec')$
by(*simp add*: *eqvts calc-atm freshChainSimps*)
moreover from ‹$([(x, y)] \cdot \Psi') = p \cdot \Psi$›
have $([(x, y)] \cdot [(x, y)] \cdot \Psi') = [(x, y)] \cdot p \cdot \Psi$
by(*simp add*: *pt-bij*)
hence $\Psi' = ((x, y)\#p) \cdot \Psi$ **by** *simp*

```
      ultimately show ?case using ‹xvec=x#xvec'› ‹yvec=y#yvec'›
        by blast
    qed
  qed
  ultimately show ?thesis by blast
qed

lemma frameChainEq':
  fixes xvec :: name list
  and   Ψ    :: 'a::fs-name
  and   yvec :: name list
  and   Ψ'   :: 'a::fs-name

  assumes ⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩
  and     xvec ♯* yvec
  and     distinct xvec
  and     distinct yvec

  obtains p where (set p) ⊆ (set xvec) × set (p • xvec) and distinctPerm p and
yvec = p • xvec and Ψ' = p • Ψ
proof −
  assume ⋀p. ⟦set p ⊆ set xvec × set (p • xvec); distinctPerm p; yvec = p • xvec;
Ψ' = p • Ψ⟧ ⟹ thesis
  moreover obtain n where n = length xvec by auto
  with assms have ∃p. (set p) ⊆ (set xvec) × set (yvec) ∧ distinctPerm p ∧  yvec
= p • xvec ∧ Ψ' = p • Ψ
  proof(induct n arbitrary: xvec yvec Ψ Ψ')
    case(0 xvec yvec Ψ Ψ')
    have Eq: ⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩ by fact
    from ‹0 = length xvec› have xvec = [] by auto
    moreover with Eq have yvec = []
      by(case-tac yvec) auto
    ultimately show ?case using Eq
      by(simp add: frame.inject)
  next
    case(Suc n xvec yvec Ψ Ψ')
    from ‹Suc n = length xvec›
    obtain x xvec' where xvec = x#xvec' and length xvec' = n
      by(case-tac xvec) auto
    from ‹⟨xvec, Ψ⟩ = ⟨yvec, Ψ'⟩› ‹xvec = x # xvec'›
    obtain y yvec' where ⟨(x#xvec'), Ψ⟩ = ⟨(y#yvec'), Ψ'⟩
      and yvec = y#yvec'
      by(case-tac yvec) auto
    hence EQ: ⦇νx⦈⦇ν*xvec'⦈(FAssert Ψ) = ⦇νy⦈⦇ν*yvec'⦈(FAssert Ψ')
      by simp
    from ‹xvec = x#xvec'› ‹yvec=y#yvec'› ‹xvec ♯* yvec›
    have x ≠ y and xvec' ♯* yvec' and x ♯ yvec' and y ♯ xvec'
      by auto
    from ‹distinct xvec› ‹distinct yvec› ‹xvec=x#xvec'› ‹yvec=y#yvec'› have x ♯
```

xvec′ **and** $y \sharp$ *yvec′* **and** *distinct xvec′* **and** *distinct yvec′*
by *simp+*
have *IH*: $\bigwedge$*xvec yvec* Ψ Ψ'. $[\![\langle xvec, (\Psi::'a)\rangle = \langle yvec, (\Psi'::'a)\rangle; xvec \sharp* yvec;$ *distinct xvec*; *distinct yvec*; $n = length\ xvec]\!] \Longrightarrow \exists\, p.\ (set\ p) \subseteq (set\ xvec) \times (set\ yvec) \wedge distinctPerm\ p \wedge\ yvec = p \cdot xvec \wedge \Psi' = p \cdot \Psi$
by *fact*
from *EQ* ‹$x \neq y$› ‹$x \sharp yvec'$› ‹$y \sharp yvec'$› **have** $\langle xvec', \Psi\rangle = \langle yvec', ([(x, y)] \cdot \Psi')\rangle$
by(*simp add*: *frame.inject alpha eqvts*)
with ‹$xvec' \sharp* yvec'$› ‹*distinct xvec′*› ‹*distinct yvec′*› ‹$length\ xvec' = n$› *IH*
obtain *p* **where** *S*: $(set\ p) \subseteq (set\ xvec') \times (set\ yvec')$ **and** *distinctPerm p* **and** $yvec' = p \cdot xvec'$ **and** $[(x, y)] \cdot \Psi' = p \cdot \Psi$
by *metis*
from *S* **have** $set((x, y)\#p) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
moreover from ‹$x \sharp xvec'$› ‹$x \sharp yvec'$› ‹$y \sharp xvec'$› ‹$y \sharp yvec'$› *S* **have** $x \sharp p$ **and** $y \sharp p$
apply(*induct p*)
by(*auto simp add*: *name-list-supp*) (*auto simp add*: *fresh-def*)

with *S* ‹*distinctPerm p*› ‹$x \neq y$› **have** $distinctPerm((x, y)\#p)$ **by** *auto*
moreover from ‹$yvec' = p \cdot xvec'$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp xvec'$› ‹$y \sharp xvec'$› **have** $(y\#yvec') = ((x, y)\#p) \cdot (x\#xvec')$
by(*simp add*: *freshChainSimps calc-atm*)
moreover from ‹$([(x, y)] \cdot \Psi') = p \cdot \Psi$›
have $([(x, y)] \cdot [(x, y)] \cdot \Psi') = [(x, y)] \cdot p \cdot \Psi$
by(*simp add*: *pt-bij*)
hence $\Psi' = ((x, y)\#p) \cdot \Psi$
by *simp*
ultimately show *?case* **using** ‹*xvec=x#xvec′*› ‹*yvec=y#yvec′*›
by *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *frameEq*[*simp*]:
fixes A_F :: *name list*
and Ψ :: *′a::fs-name*
and Ψ' :: *′a*

shows $\langle A_F, \Psi\rangle = \langle \varepsilon, \Psi'\rangle = (A_F = [] \wedge \Psi = \Psi')$
and $\langle \varepsilon, \Psi'\rangle = \langle A_F, \Psi\rangle = (A_F = [] \wedge \Psi = \Psi')$
proof −
{
assume $\langle A_F, \Psi\rangle = \langle \varepsilon, \Psi'\rangle$
hence *A*: $\langle A_F, \Psi\rangle = \langle [], \Psi'\rangle$ **by** *simp*
hence $length\ A_F = length\ ([]::name\ list)$
by(*rule frameChainEqLength*)
with *A* **have** $A_F = []$ **and** $\Psi = \Psi'$ **by**(*auto simp add*: *frame.inject*)
}

thus $\langle A_F, \Psi\rangle = \langle \varepsilon, \Psi'\rangle = (A_F = [] \wedge \Psi = \Psi')$
and $\langle \varepsilon, \Psi'\rangle = \langle A_F, \Psi\rangle = (A_F = [] \wedge \Psi = \Psi')$
by *auto*
qed

lemma *distinctFrame*:
fixes A_F :: *name list*
and Ψ_F :: *'a::fs-name*
and C :: *'b::fs-name*

assumes $A_F \sharp* C$

obtains $A_F{}'$ **where** $\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F\rangle$ **and** *distinct* $A_F{}'$ **and** $A_F{}' \sharp* C$
proof −
assume $\bigwedge A_F{}'.\ [\![\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F\rangle;\ \textit{distinct}\ A_F{}';\ A_F{}' \sharp* C]\!] \Longrightarrow \textit{thesis}$
moreover from *assms* **have** $\exists A_F{}'.\ \langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F\rangle \wedge \textit{distinct}\ A_F{}' \wedge A_F{}' \sharp* C$
proof(*induct* A_F)
case *Nil*
thus *?case* **by** *simp*
next
case(*Cons a* A_F)
then obtain $A_F{}'$ **where** *Eq*: $\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F\rangle$ **and** *distinct* $A_F{}'$ **and** $A_F{}' \sharp* C$ **by** *force*
from ‹$(a\#A_F) \sharp* C$› **have** $a \sharp C$ **and** $A_F \sharp* C$ **by** *simp+*
show *?case*
proof(*case-tac* $a \sharp \langle A_F{}', \Psi_F\rangle$)
assume $a \sharp \langle A_F{}', \Psi_F\rangle$
obtain *b::name* **where** $b \sharp A_F{}'$ **and** $b \sharp \Psi_F$ **and** $b \sharp C$ **by**(*generate-fresh name*, *auto*)
have $\langle (a\#A_F), \Psi_F\rangle = \langle (b\#A_F{}'), \Psi_F\rangle$
proof −
from *Eq* **have** $\langle (a\#A_F), \Psi_F\rangle = \langle (a\#A_F{}'), \Psi_F\rangle$ **by**(*simp add*: *frame.inject*)
moreover from ‹$b \sharp \Psi_F$› **have** $\ldots = (\!|\nu b|\!)([(a, b)] \cdot (\!|\nu* A_F{}'|\!)(\textit{FAssert}\ \Psi_F))$
by(*force intro*: *alphaFrameRes simp add*: *frameResChainFresh*)
ultimately show *?thesis* **using** ‹$a \sharp \langle A_F{}', \Psi_F\rangle$› ‹$b \sharp \Psi_F$›
by(*simp add*: *frameResChainFresh*)
qed
moreover from ‹*distinct* $A_F{}'$› ‹$b \sharp A_F{}'$› **have** *distinct*$(b\#A_F{}')$ **by** *simp*
moreover from ‹$A_F{}' \sharp* C$› ‹$b \sharp C$› **have** $(b\#A_F{}') \sharp* C$ **by** *simp+*
ultimately show *?case* **by** *blast*
next
from *Eq* **have** $\langle (a\#A_F), \Psi_F\rangle = \langle (a\#A_F{}'), \Psi_F\rangle$ **by**(*simp add*: *frame.inject*)
moreover assume $\neg(a \sharp \langle A_F{}', \Psi_F\rangle)$
hence $a \sharp A_F{}'$ **apply**(*simp add*: *fresh-def*)
by(*induct* $A_F{}'$) (*auto simp add*: *supp-list-nil supp-list-cons supp-atm frame.supp abs-supp*)
with ‹*distinct* $A_F{}'$› **have** *distinct*$(a\#A_F{}')$ **by** *simp*
moreover from ‹$A_F{}' \sharp* C$› ‹$a \sharp C$› **have** $(a\#A_F{}') \sharp* C$ **by** *simp+*

```
      ultimately show ?case by blast
    qed
  qed
  ultimately show ?thesis using ‹A_F ♯* C›
    by blast
qed

lemma freshFrame:
  fixes F :: ('a::fs-name) frame
  and   C :: 'b ::fs-name

  obtains A_F Ψ_F where F = ⟨A_F, Ψ_F⟩ and distinct A_F and A_F ♯* C
proof −
  assume ⋀A_F Ψ_F. ⟦F = ⟨A_F, Ψ_F⟩; distinct A_F; A_F ♯* C⟧ ⟹ thesis
  moreover have ∃ A_F Ψ_F. F = ⟨A_F, Ψ_F⟩ ∧ A_F ♯* C
  proof(nominal-induct F avoiding: C rule: frame.strong-induct)
    case(FAssert Ψ_F)
    have FAssert Ψ_F = ⟨[], Ψ_F⟩ by simp
    moreover have ([]::name list) ♯* C by simp
    ultimately show ?case by force
  next
    case(FRes a F)
    from ‹⋀C. ∃ A_F Ψ_F. F = ⟨A_F, Ψ_F⟩ ∧ A_F ♯* C›
    obtain A_F Ψ_F  where F = ⟨A_F, Ψ_F⟩ and A_F ♯* C
      by blast
    with ‹a ♯ C› have ⦇νa⦈F = ⦇ν*(a#A_F)⦈(FAssert Ψ_F) and (a#A_F) ♯* C
      by simp+
    thus ?case by blast
  qed
  ultimately show ?thesis
    by(auto, rule-tac distinctFrame) auto
qed

locale assertionAux =
  fixes SCompose :: 'b::fs-name ⇒ 'b ⇒ 'b   (infixr ‹⊗› 80)
  and   SImp     :: 'b ⇒ 'c::fs-name ⇒ bool (‹- ⊢ -› [70, 70] 70)
  and   SBottom  :: 'b                       (‹⊥› 90)
  and   SChanEq  :: 'a::fs-name ⇒ 'a ⇒ 'c   (‹- ↔ -› [80, 80] 80)

  assumes statEqvt[eqvt]:   ⋀p::name prm. p • (Ψ ⊢ Φ) = (p • Ψ) ⊢ (p • Φ)
  and     statEqvt'[eqvt]:  ⋀p::name prm. p • (Ψ ⊗ Ψ') = (p • Ψ) ⊗ (p • Ψ')
  and     statEqvt''[eqvt]: ⋀p::name prm. p • (M ↔ N) = (p • M) ↔ (p • N)
  and     permBottom[eqvt]: ⋀p::name prm. (p • SBottom) = SBottom

begin

lemma statClosed:
  fixes Ψ :: 'b
  and   φ :: 'c
```

and p :: *name prm*

assumes $\Psi \vdash \varphi$

shows $(p \cdot \Psi) \vdash (p \cdot \varphi)$
using *assms statEqvt*
by(*simp add*: *perm-bool*)

lemma *compSupp*:
fixes Ψ :: $'b$
and Ψ' :: $'b$

shows $(supp(\Psi \otimes \Psi')::name\ set) \subseteq ((supp\ \Psi) \cup (supp\ \Psi'))$
proof(*auto simp add*: *eqvts supp-def*)
fix x::*name*
let $?P = \lambda y.\ ([(x,\ y)] \cdot \Psi) \otimes [(x,\ y)] \cdot \Psi' \neq \Psi \otimes \Psi'$
let $?Q = \lambda y\ \Psi.\ ([(x,\ y)] \cdot \Psi) \neq \Psi$
assume *finite* $\{y.\ ?Q\ y\ \Psi'\}$
moreover assume *finite* $\{y.\ ?Q\ y\ \Psi\}$ **and** *infinite* $\{y.\ ?P(y)\}$
hence *infinite*$(\{y.\ ?P(y)\} - \{y.\ ?Q\ y\ \Psi\})$ **by**(*rule Diff-infinite-finite*)
ultimately have *infinite*$((\{y.\ ?P(y)\} - \{y.\ ?Q\ y\ \Psi\}) - \{y.\ ?Q\ y\ \Psi'\})$ **by**(*rule Diff-infinite-finite*)
hence *infinite*$(\{y.\ ?P(y) \wedge \neg(?Q\ y\ \Psi) \wedge \neg\ (?Q\ y\ \Psi')\})$ **by**(*simp add*: *set-diff-eq*)
moreover have $\{y.\ ?P(y) \wedge \neg(?Q\ y\ \Psi) \wedge \neg\ (?Q\ y\ \Psi')\} = \{\}$ **by** *auto*
ultimately have *infinite* $\{\}$ **by**(*drule-tac Infinite-cong*) *auto*
thus *False* **by** *simp*
qed

lemma *chanEqSupp*:
fixes M :: $'a$
and N :: $'a$

shows $(supp(M \leftrightarrow N)::name\ set) \subseteq ((supp\ M) \cup (supp\ N))$
proof(*auto simp add*: *eqvts supp-def*)
fix x::*name*
let $?P = \lambda y.\ ([(x,\ y)] \cdot M) \leftrightarrow [(x,\ y)] \cdot N \neq M \leftrightarrow N$
let $?Q = \lambda y\ M.\ ([(x,\ y)] \cdot M) \neq M$
assume *finite* $\{y.\ ?Q\ y\ N\}$
moreover assume *finite* $\{y.\ ?Q\ y\ M\}$ **and** *infinite* $\{y.\ ?P(y)\}$
hence *infinite*$(\{y.\ ?P(y)\} - \{y.\ ?Q\ y\ M\})$ **by**(*rule Diff-infinite-finite*)
ultimately have *infinite*$((\{y.\ ?P(y)\} - \{y.\ ?Q\ y\ M\}) - \{y.\ ?Q\ y\ N\})$ **by**(*rule Diff-infinite-finite*)
hence *infinite*$(\{y.\ ?P(y) \wedge \neg(?Q\ y\ M) \wedge \neg\ (?Q\ y\ N)\})$ **by**(*simp add*: *set-diff-eq*)
moreover have $\{y.\ ?P(y) \wedge \neg(?Q\ y\ M) \wedge \neg\ (?Q\ y\ N)\} = \{\}$ **by** *auto*
ultimately have *infinite* $\{\}$ **by**(*drule-tac Infinite-cong*) *auto*
thus *False* **by** *simp*
qed

lemma *freshComp*[*intro*]:

fixes x :: *name*
and Ψ :: $'b$
and Ψ' :: $'b$

assumes $x \sharp \Psi$
and $x \sharp \Psi'$

shows $x \sharp \Psi \otimes \Psi'$
using *assms compSupp*
by(*auto simp add*: *fresh-def*)

lemma *freshCompChain*[*intro*]:
fixes *xvec* :: *name list*
and *Xs* :: *name set*
and Ψ :: $'b$
and Ψ' :: $'b$

shows $[\![xvec \sharp* \Psi;\ xvec \sharp* \Psi']\!] \Longrightarrow xvec \sharp* (\Psi \otimes \Psi')$
and $[\![Xs \sharp* \Psi;\ Xs \sharp* \Psi']\!] \Longrightarrow Xs \sharp* (\Psi \otimes \Psi')$
by(*auto simp add*: *fresh-star-def*)

lemma *freshChanEq*[*intro*]:
fixes x :: *name*
and M :: $'a$
and N :: $'a$

assumes $x \sharp M$
and $x \sharp N$

shows $x \sharp M \leftrightarrow N$
using *assms chanEqSupp*
by(*auto simp add*: *fresh-def*)

lemma *freshChanEqChain*[*intro*]:
fixes *xvec* :: *name list*
and *Xs* :: *name set*
and M :: $'a$
and N :: $'a$

shows $[\![xvec \sharp* M;\ xvec \sharp* N]\!] \Longrightarrow xvec \sharp* (M \leftrightarrow N)$
and $[\![Xs \sharp* M;\ Xs \sharp* N]\!] \Longrightarrow Xs \sharp* (M \leftrightarrow N)$
by(*auto simp add*: *fresh-star-def*)

lemma *suppBottom*[*simp*]:
shows ((*supp SBottom*)::*name set*) = {}
by(*auto simp add*: *supp-def permBottom*)

lemma *freshBottom*[*simp*]:
fixes x :: *name*

shows $x \sharp \bot$
by(*simp add*: *fresh-def*)

lemma *freshBottoChain*[*simp*]:
fixes *xvec* :: *name list*
and *Xs* :: *name set*

shows $xvec \sharp* (\bot)$
and $Xs \sharp* (\bot)$
by(*auto simp add*: *fresh-star-def*)

lemma *chanEqClosed*:
fixes Ψ :: $'b$
and M :: $'a$
and N :: $'a$
and p :: *name prm*

assumes $\Psi \vdash M \leftrightarrow N$

shows $(p \cdot \Psi) \vdash (p \cdot M) \leftrightarrow (p \cdot N)$
proof −
from ‹$\Psi \vdash M \leftrightarrow N$› **have** $(p \cdot \Psi) \vdash p \cdot (M \leftrightarrow N)$
by(*rule statClosed*)
thus *?thesis* **by**(*simp add*: *eqvts*)
qed

definition
AssertionStatImp :: $'b \Rightarrow 'b \Rightarrow$ *bool* (**infix** ‹$\hookrightarrow$› *70*)
where $(\Psi \hookrightarrow \Psi') \equiv (\forall \Phi.\ \Psi \vdash \Phi \longrightarrow \Psi' \vdash \Phi)$

definition
AssertionStatEq :: $'b \Rightarrow 'b \Rightarrow$ *bool* (**infix** ‹$\simeq$› *70*)
where $(\Psi \simeq \Psi') \equiv \Psi \hookrightarrow \Psi' \wedge \Psi' \hookrightarrow \Psi$

lemma *statImpEnt*:
fixes Ψ :: $'b$
and Ψ' :: $'b$
and Φ :: $'c$

assumes $\Psi \hookrightarrow \Psi'$
and $\Psi \vdash \Phi$

shows $\Psi' \vdash \Phi$
using *assms*
by(*simp add*: *AssertionStatImp-def*)

lemma *statEqEnt*:
fixes Ψ :: $'b$

and $\Psi' :: 'b$
and $\Phi :: 'c$

assumes $\Psi \simeq \Psi'$
and $\Psi \vdash \Phi$

shows $\Psi' \vdash \Phi$
using *assms*
by(*auto simp add*: *AssertionStatEq-def intro*: *statImpEnt*)

lemma *AssertionStatImpClosed*:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$
and p :: *name prm*

assumes $\Psi \hookrightarrow \Psi'$

shows $(p \cdot \Psi) \hookrightarrow (p \cdot \Psi')$
proof(*auto simp add*: *AssertionStatImp-def*)
fix φ
assume $(p \cdot \Psi) \vdash \varphi$
hence $\Psi \vdash$ *rev* $p \cdot \varphi$ **by**(*drule-tac p=rev p* **in** *statClosed*) *auto*
with ‹$\Psi \hookrightarrow \Psi'$› **have** $\Psi' \vdash$ *rev* $p \cdot \varphi$ **by**(*simp add*: *AssertionStatImp-def*)
thus $(p \cdot \Psi') \vdash \varphi$ **by**(*drule-tac p=p* **in** *statClosed*) *auto*
qed

lemma *AssertionStatEqClosed*:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$
and p :: *name prm*

assumes $\Psi \simeq \Psi'$

shows $(p \cdot \Psi) \simeq (p \cdot \Psi')$
using *assms*
by(*auto simp add*: *AssertionStatEq-def intro*: *AssertionStatImpClosed*)

lemma *AssertionStatImpEqvt*[*eqvt*]:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$
and p :: *name prm*

shows $(p \cdot (\Psi \hookrightarrow \Psi')) = ((p \cdot \Psi) \hookrightarrow (p \cdot \Psi'))$
by(*simp add*: *AssertionStatImp-def eqvts*)

lemma *AssertionStatEqEqvt*[*eqvt*]:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$
and p :: *name prm*

shows $(p \cdot (\Psi \simeq \Psi')) = ((p \cdot \Psi) \simeq (p \cdot \Psi'))$
by(*simp add*: *AssertionStatEq-def eqvts*)

lemma *AssertionStatImpRefl*[*simp*]:
fixes $\Psi :: 'b$

shows $\Psi \hookrightarrow \Psi$
by(*simp add*: *AssertionStatImp-def*)

lemma *AssertionStatEqRefl*[*simp*]:
fixes $\Psi :: 'b$

shows $\Psi \simeq \Psi$
by(*simp add*: *AssertionStatEq-def*)

lemma *AssertionStatEqSym*:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$

assumes $\Psi \simeq \Psi'$

shows $\Psi' \simeq \Psi$
using *assms*
by(*auto simp add*: *AssertionStatEq-def*)

lemma *AssertionStatImpTrans*:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$
and $\Psi'' :: 'b$

assumes $\Psi \hookrightarrow \Psi'$
and $\Psi' \hookrightarrow \Psi''$

shows $\Psi \hookrightarrow \Psi''$
using *assms*
by(*simp add*: *AssertionStatImp-def*)

lemma *AssertionStatEqTrans*:
fixes $\Psi :: 'b$
and $\Psi' :: 'b$
and $\Psi'' :: 'b$

assumes $\Psi \simeq \Psi'$
and $\Psi' \simeq \Psi''$

shows $\Psi \simeq \Psi''$
using *assms*
by(*auto simp add*: *AssertionStatEq-def intro*: *AssertionStatImpTrans*)

definition
  FrameImp :: *'b::fs-name frame* ⇒ *'c* ⇒ *bool* (**infixl** ‹$\vdash_F$› *70*)
  where $(F \vdash_F \Phi) = (\exists A_F\ \Psi_F.\ F = \langle A_F, \Psi_F\rangle \wedge A_F \sharp* \Phi \wedge (\Psi_F \vdash \Phi))$

lemma *frameImpI*:
  fixes F :: *'b frame*
  and φ :: *'c*
  and A_F :: *name list*
  and Ψ_F :: *'b*

  assumes $F = \langle A_F, \Psi_F\rangle$
  and $A_F \sharp* \varphi$
  and $\Psi_F \vdash \varphi$

  shows $F \vdash_F \varphi$
using *assms*
by(*force simp add*: *FrameImp-def*)

lemma *frameImpAlphaEnt*:
  fixes A_F :: *name list*
  and Ψ_F :: *'b*
  and $A_F{}'$ :: *name list*
  and $\Psi_F{}'$ :: *'b*
  and φ :: *'c*

  assumes $\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F{}'\rangle$
  and $A_F \sharp* \varphi$
  and $A_F{}' \sharp* \varphi$
  and $\Psi_F{}' \vdash \varphi$

  shows $\Psi_F \vdash \varphi$
proof −
  from ‹$\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F{}'\rangle$›
  obtain n **where** $n = length\ A_F$ **by** *blast*
  moreover from ‹$\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F{}'\rangle$›
  have $length\ A_F = length\ A_F{}'$
    by(*rule frameChainEqLength*)
  ultimately show *?thesis* **using** *assms*
  proof(*induct n arbitrary*: $A_F\ A_F{}'\ \Psi_F{}'$ *rule*: *nat.induct*)
    case(*zero* $A_F\ A_F{}'\ \Psi_F{}'$)
    thus *?case* **by**(*auto simp add*: *frame.inject*)
  next
    case(*Suc n* $A_F\ A_F{}'\ \Psi_F{}'$)
    from ‹$Suc\ n = length\ A_F$›
    obtain *x xs* **where** $A_F = x\#xs$ **and** $n = length\ xs$
      by(*case-tac* A_F) *auto*
    from ‹$\langle A_F, \Psi_F\rangle = \langle A_F{}', \Psi_F{}'\rangle$› ‹$A_F = x\ \#\ xs$›
    obtain *y ys* **where** $\langle (x\#xs), \Psi_F\rangle = \langle (y\#ys), \Psi_F{}'\rangle$ **and** $A_F{}' = y\#ys$

by(*case-tac* $A_F{}'$) *auto*
hence *EQ*: $(\!|\nu x|\!)(\!|\nu{*}xs|\!)(\mathit{FAssert}\ \Psi_F) = (\!|\nu y|\!)(\!|\nu{*}ys|\!)(\mathit{FAssert}\ \Psi_F{}')$
by *simp*
from ‹$A_F = x\ \#\ xs$› ‹$A_F{}' = y\ \#\ ys$› ‹$\mathit{length}\ A_F = \mathit{length}\ A_F{}'$› ‹$A_F \sharp{*}\ \varphi$› ‹$A_F{}' \sharp{*}\ \varphi$›
have $\mathit{length}\ xs = \mathit{length}\ ys$ **and** $xs \sharp{*}\ \varphi$ **and** $ys \sharp{*}\ \varphi$ **and** $x \sharp\ \varphi$ **and** $y \sharp\ \varphi$
by *auto*

have *IH*: $\bigwedge xs\ ys\ \Psi_F{}'.\ [\![n = \mathit{length}\ xs;\ \mathit{length}\ xs = \mathit{length}\ ys;\ \langle xs,\ \Psi_F\rangle = \langle ys,\ (\Psi_F{}'::'b)\rangle;\ xs \sharp{*}\ \varphi;\ ys \sharp{*}\ \varphi;\ \Psi_F{}' \vdash \varphi]\!] \Longrightarrow \Psi_F \vdash \varphi$
by *fact*
show *?case*
proof(*case-tac* $x = y$)
assume $x = y$
with *EQ* **have** $\langle xs,\ \Psi_F\rangle = \langle ys,\ \Psi_F{}'\rangle$ **by**(*simp add*: *alpha frame.inject*)
with *IH* ‹$n = \mathit{length}\ xs$› ‹$\mathit{length}\ xs = \mathit{length}\ ys$› ‹$xs \sharp{*}\ \varphi$› ‹$ys \sharp{*}\ \varphi$› ‹$\Psi_F{}' \vdash \varphi$›
show *?case* **by** *blast*
next
assume $x \neq y$
with *EQ* **have** $\langle xs,\ \Psi_F\rangle = [(x,\ y)] \cdot \langle ys,\ \Psi_F{}'\rangle$ **by**(*simp add*: *alpha frame.inject*)
hence $\langle xs,\ \Psi_F\rangle = \langle ([(x,\ y)] \cdot ys),\ ([(x,\ y)] \cdot \Psi_F{}')\rangle$ **by**(*simp add*: *eqvts*)
moreover from ‹$\mathit{length}\ xs = \mathit{length}\ ys$› **have** $\mathit{length}\ xs = \mathit{length}([(x,\ y)] \cdot ys)$
by *auto*
moreover from ‹$ys \sharp{*}\ \varphi$› **have** $([(x,\ y)] \cdot ys) \sharp{*}\ ([(x,\ y)] \cdot \varphi)$
by(*simp add*: *fresh-star-bij*)
with ‹$x \sharp\ \varphi$› ‹$y \sharp\ \varphi$› **have** $([(x,\ y)] \cdot ys) \sharp{*}\ \varphi$
by *simp*
moreover with ‹$\Psi_F{}' \vdash \varphi$› **have** $([(x,\ y)] \cdot \Psi_F{}') \vdash ([(x,\ y)] \cdot \varphi)$
by(*simp add*: *statClosed*)
with ‹$x \sharp\ \varphi$› ‹$y \sharp\ \varphi$› **have** $([(x,\ y)] \cdot \Psi_F{}') \vdash \varphi$
by *simp*
ultimately show *?case* **using** *IH* ‹$n = \mathit{length}\ xs$› ‹$xs \sharp{*}\ \varphi$›
by *blast*
qed
qed
qed

lemma *frameImpEAux*:
fixes F :: $'b$ *frame*
and Φ :: $'c$

assumes $F \vdash_F \Phi$
and $F = \langle A_F,\ \Psi_F\rangle$
and $A_F \sharp{*}\ \Phi$

shows $\Psi_F \vdash \Phi$
using *assms*
by(*auto simp add*: *FrameImp-def dest*: *frameImpAlphaEnt*)

lemma *frameImpE*:
  fixes F :: $'b$ *frame*
  and Φ :: $'c$

  assumes $\langle A_F, \Psi_F\rangle \vdash_F \Phi$
  and $A_F \sharp* \Phi$

  shows $\Psi_F \vdash \Phi$
using *assms*
by(*auto elim*: *frameImpEAux*)

lemma *frameImpClosed*:
  fixes F :: $'b$ *frame*
  and Φ :: $'c$
  and p :: *name prm*

  assumes $F \vdash_F \Phi$

  shows $(p \cdot F) \vdash_F (p \cdot \Phi)$
using *assms*
by(*force simp add*: *FrameImp-def eqvts pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*]
    intro: *statClosed*)

lemma *frameImpEqvt*[*eqvt*]:
  fixes F :: $'b$ *frame*
  and Φ :: $'c$
  and p :: *name prm*

  shows $(p \cdot (F \vdash_F \Phi)) = (p \cdot F) \vdash_F (p \cdot \Phi)$
proof −
  have $F \vdash_F \Phi \Longrightarrow (p \cdot F) \vdash_F (p \cdot \Phi)$
    by(*rule frameImpClosed*)
  moreover have $(p \cdot F) \vdash_F (p \cdot \Phi) \Longrightarrow F \vdash_F \Phi$
    by(*drule-tac* p = *rev* p **in** *frameImpClosed*) *simp*
  ultimately show *?thesis*
    by(*auto simp add*: *perm-bool*)
qed

lemma *frameImpEmpty*[*simp*]:
  fixes Ψ :: $'b$
  and φ :: $'c$

  shows $\langle \varepsilon, \Psi\rangle \vdash_F \varphi = \Psi \vdash \varphi$
by(*auto simp add*: *FrameImp-def*)

definition
  FrameStatImp :: $'b$ *frame* $\Rightarrow$ $'b$ *frame*$\Rightarrow$ *bool* (**infix** ‹$\hookrightarrow_F$› *70*)
  where $(F \hookrightarrow_F G) \equiv (\forall \varphi.\ F \vdash_F \varphi \longrightarrow G \vdash_F \varphi)$

definition
FrameStatEq :: *'b frame* ⇒ *'b frame*⇒ *bool* (**infix** ‹$\simeq_F$› *70*)
where $(F \simeq_F G) \equiv F \hookrightarrow_F G \wedge G \hookrightarrow_F F$

lemma *FrameStatImpClosed*:
fixes F :: *'b frame*
and G :: *'b frame*
and p :: *name prm*

assumes $F \hookrightarrow_F G$

shows $(p \cdot F) \hookrightarrow_F (p \cdot G)$
proof(*auto simp add*: *FrameStatImp-def*)
fix φ
assume $(p \cdot F) \vdash_F \varphi$
hence $F \vdash_F$ *rev* $p \cdot \varphi$ **by**(*drule-tac p=rev p* **in** *frameImpClosed*) *auto*
with ‹$F \hookrightarrow_F G$› **have** $G \vdash_F$ *rev* $p \cdot \varphi$ **by**(*simp add*: *FrameStatImp-def*)
thus $(p \cdot G) \vdash_F \varphi$ **by**(*drule-tac p=p* **in** *frameImpClosed*) *auto*
qed

lemma *FrameStatEqClosed*:
fixes F :: *'b frame*
and G :: *'b frame*
and p :: *name prm*

assumes $F \simeq_F G$

shows $(p \cdot F) \simeq_F (p \cdot G)$
using *assms*
by(*auto simp add*: *FrameStatEq-def intro*: *FrameStatImpClosed*)

lemma *FrameStatImpEqvt*[*eqvt*]:
fixes F :: *'b frame*
and G :: *'b frame*
and p :: *name prm*

shows $(p \cdot (F \hookrightarrow_F G)) = ((p \cdot F) \hookrightarrow_F (p \cdot G))$
by(*simp add*: *FrameStatImp-def eqvts*)

lemma *FrameStatEqEqvt*[*eqvt*]:
fixes F :: *'b frame*
and G :: *'b frame*
and p :: *name prm*

shows $(p \cdot (F \simeq_F G)) = ((p \cdot F) \simeq_F (p \cdot G))$
by(*simp add*: *FrameStatEq-def eqvts*)

lemma *FrameStatImpRefl*[*simp*]:

fixes F :: $'b$ *frame*

shows $F \hookrightarrow_F F$
by(*simp add*: *FrameStatImp-def*)

lemma *FrameStatEqRefl*[*simp*]:
fixes F :: $'b$ *frame*

shows $F \simeq_F F$
by(*simp add*: *FrameStatEq-def*)

lemma *FrameStatEqSym*:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*

assumes $F \simeq_F G$

shows $G \simeq_F F$
using *assms*
by(*auto simp add*: *FrameStatEq-def*)

lemma *FrameStatImpTrans*:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*
and H :: $'b$ *frame*

assumes $F \hookrightarrow_F G$
and $G \hookrightarrow_F H$

shows $F \hookrightarrow_F H$
using *assms*
by(*simp add*: *FrameStatImp-def*)

lemma *FrameStatEqTrans*:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*
and H :: $'b$ *frame*

assumes $F \simeq_F G$
and $G \simeq_F H$

shows $F \simeq_F H$
using *assms*
by(*auto simp add*: *FrameStatEq-def intro*: *FrameStatImpTrans*)

lemma *fsCompose*[*simp*]: *finite*((*supp SCompose*)::*name set*)
by(*simp add*: *supp-def perm-fun-def eqvts*)

nominal-primrec

insertAssertion :: *'b frame* ⇒ *'b* ⇒ *'b frame*
where
insertAssertion (*FAssert* Ψ) Ψ′ = *FAssert* (Ψ′ ⊗ Ψ)
| *x* ♯ Ψ′ ⟹ *insertAssertion* ((⦇νx⦈*F*) Ψ′ = (⦇νx⦈(*insertAssertion F* Ψ′)
apply(*finite-guess add*: *fsCompose*)+
apply(*rule TrueI*)+
apply(*simp add*: *abs-fresh*)
apply(*rule supports-fresh*[*of supp* Ψ′])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
done

lemma *insertAssertionEqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *F* :: *'b frame*
and Ψ :: *'b*

shows *p* • (*insertAssertion F* Ψ) = *insertAssertion* (*p* • *F*) (*p* • Ψ)
by(*nominal-induct F avoiding*: *p* Ψ *rule*: *frame.strong-induct*)
(*auto simp add*: *at-prm-fresh*[*OF at-name-inst*]
pt-fresh-perm-app[*OF pt-name-inst*, *OF at-name-inst*] *eqvts*)

nominal-primrec
mergeFrame :: *'b frame* ⇒ *'b frame* ⇒ *'b frame*
where
mergeFrame (*FAssert* Ψ) *G* = *insertAssertion G* Ψ
| *x* ♯ *G* ⟹ *mergeFrame* ((⦇νx⦈*F*) *G* = (⦇νx⦈(*mergeFrame F G*)
apply(*finite-guess add*: *fsCompose*)+
apply(*rule TrueI*)+
apply(*simp add*: *abs-fresh*)
apply(*simp add*: *fs-name1*)
apply(*rule supports-fresh*[*of supp G*])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess*)+
done

notation *mergeFrame* (**infixr** ‹$\otimes_F$› *80*)

abbreviation
frameBottomJudge (‹$\bot_F$›) **where** $\bot_F$ ≡ (*FAssert SBottom*)

lemma *mergeFrameEqvt*[*eqvt*]:
fixes *p* :: *name prm*
and *F* :: *'b frame*

and G :: $'b$ *frame*

shows $p \cdot$ (*mergeFrame* F G) = *mergeFrame* ($p \cdot F$) ($p \cdot G$)
by(*nominal-induct* F *avoiding*: p G *rule*: *frame.strong-induct*)
(*auto simp add*: *at-prm-fresh*[*OF at-name-inst*]
pt-fresh-perm-app[*OF pt-name-inst*, *OF at-name-inst*] *eqvts*)

nominal-primrec
extractFrame :: ($'a$, $'b$, $'c$) *psi* $\Rightarrow$ $'b$ *frame*
and *extractFrame'* :: ($'a$, $'b$, $'c$) *input* $\Rightarrow$ $'b$ *frame*
and *extractFrame''* :: ($'a$, $'b$, $'c$) *psiCase* $\Rightarrow$ $'b$ *frame*

where
extractFrame (**0**) = $\langle\varepsilon, \bot\rangle$
| *extractFrame* (M⦇I) = $\langle\varepsilon, \bot\rangle$
| *extractFrame* ($M\langle N\rangle.P$) = $\langle\varepsilon, \bot\rangle$
| *extractFrame* (*Case* C) = $\langle\varepsilon, \bot\rangle$
| *extractFrame* ($P \parallel Q$) = (*extractFrame* P) $\otimes_F$ (*extractFrame* Q)
| *extractFrame* ((⦃Ψ⦄::($'a$, $'b$, $'c$) *psi*)) = $\langle\varepsilon, \Psi\rangle$

| *extractFrame* (⦇νx⦈P) = ⦇νx⦈(*extractFrame* P)
| *extractFrame* (!P) = $\langle\varepsilon, \bot\rangle$

| *extractFrame'* ((*Trm* M P)::($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *input*) = $\langle\varepsilon, \bot\rangle$

| *extractFrame'* (*Bind* x I) = $\langle\varepsilon, \bot\rangle$

| *extractFrame''* ($\bot_c$::($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psiCase*) = $\langle\varepsilon, \bot\rangle$
| *extractFrame''* ($\Box\Phi \Rightarrow P$ C) = $\langle\varepsilon, \bot\rangle$
apply(*finite-guess add*: *fsCompose*)+
apply(*rule TrueI*)+
apply(*simp add*: *abs-fresh*)+
apply(*fresh-guess add*: *freshBottom*)+
apply(*rule supports-fresh*[*of* {}])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess add*: *freshBottom*)+
apply(*rule supports-fresh*[*of* {}])
apply(*force simp add*: *perm-fun-def eqvts fresh-def*[*symmetric*] *supports-def*)
apply(*simp add*: *fs-name1*)
apply(*simp add*: *fresh-def*[*symmetric*])
apply(*fresh-guess add*: *freshBottom*)+
done

lemmas *extractFrameSimps* = *extractFrame-extractFrame'-extractFrame''.simps*

lemma *extractFrameEqvt*[*eqvt*]:
fixes p :: *name prm*

and P :: $('a, 'b, 'c)$ *psi*
and I :: $('a, 'b, 'c)$ *input*
and C :: $('a, 'b, 'c)$ *psiCase*

shows $p \cdot (extractFrame\ P) = extractFrame\ (p \cdot P)$
and $p \cdot (extractFrame'\ I) = extractFrame'\ (p \cdot I)$
and $p \cdot (extractFrame''\ C) = extractFrame''\ (p \cdot C)$
by(*nominal-induct P* **and** *I* **and** *C avoiding*: *p rule*: *psi-input-psiCase.strong-inducts*)
(*auto simp add*: *at-prm-fresh*[*OF at-name-inst*] *eqvts permBottom*
pt-fresh-perm-app[*OF pt-name-inst*, *OF at-name-inst*])

lemma *insertAssertionFresh*[*intro*]:
fixes F :: $'b$ *frame*
and Ψ :: $'b$
and x :: *name*

assumes $x \sharp F$
and $x \sharp \Psi$

shows $x \sharp (insertAssertion\ F\ \Psi)$
using *assms*
by(*nominal-induct F avoiding*: $x\ \Psi$ *rule*: *frame.strong-induct*)
(*auto simp add*: *abs-fresh*)

lemma *insertAssertionFreshChain*[*intro*]:
fixes F :: $'b$ *frame*
and Ψ :: $'b$
and *xvec* :: *name list*
and *Xs* :: *name set*

shows $[\![xvec \sharp* F;\ xvec \sharp* \Psi]\!] \Longrightarrow xvec \sharp* (insertAssertion\ F\ \Psi)$
and $[\![Xs \sharp* F;\ Xs \sharp* \Psi]\!] \Longrightarrow Xs \sharp* (insertAssertion\ F\ \Psi)$
by(*auto simp add*: *fresh-star-def*)

lemma *mergeFrameFresh*[*intro*]:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*
and x :: *name*

shows $[\![x \sharp F;\ x \sharp G]\!] \Longrightarrow x \sharp (mergeFrame\ F\ G)$
by(*nominal-induct F avoiding*: $x\ G$ *rule*: *frame.strong-induct*)
(*auto simp add*: *abs-fresh*)

lemma *mergeFrameFreshChain*[*intro*]:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*
and *xvec* :: *name list*
and *Xs* :: *name set*

shows $[\![xvec \sharp* F; xvec \sharp* G]\!] \Longrightarrow xvec \sharp* (mergeFrame\ F\ G)$
and $[\![Xs \sharp* F; Xs \sharp* G]\!] \Longrightarrow Xs \sharp* (mergeFrame\ F\ G)$
by(*auto simp add*: *fresh-star-def*)

lemma *extractFrameFresh*:
fixes P :: $('a, 'b, 'c)$ *psi*
and I :: $('a, 'b, 'c)$ *input*
and C :: $('a, 'b, 'c)$ *psiCase*
and x :: *name*

shows $x \sharp P \Longrightarrow x \sharp extractFrame\ P$
and $x \sharp I \Longrightarrow x \sharp extractFrame'\ I$
and $x \sharp C \Longrightarrow x \sharp extractFrame''\ C$
by(*nominal-induct* P **and** I **and** C *avoiding*: x *rule*: *psi-input-psiCase.strong-inducts*)
(*auto simp add*: *abs-fresh*)

lemma *extractFrameFreshChain*:
fixes P :: $('a, 'b, 'c)$ *psi*
and I :: $('a, 'b, 'c)$ *input*
and C :: $('a, 'b, 'c)$ *psiCase*
and $xvec$ :: *name list*
and Xs :: *name set*

shows $xvec \sharp* P \Longrightarrow xvec \sharp* extractFrame\ P$
and $xvec \sharp* I \Longrightarrow xvec \sharp* extractFrame'\ I$
and $xvec \sharp* C \Longrightarrow xvec \sharp* extractFrame''\ C$
and $Xs \sharp* P \Longrightarrow Xs \sharp* extractFrame\ P$
and $Xs \sharp* I \Longrightarrow Xs \sharp* extractFrame'\ I$
and $Xs \sharp* C \Longrightarrow Xs \sharp* extractFrame''\ C$
by(*auto simp add*: *fresh-star-def intro*: *extractFrameFresh*)

lemma *guardedFrameSupp*[*simp*]:
fixes P :: $('a, 'b, 'c)$ *psi*
and I :: $('a, 'b, 'c)$ *input*
and C :: $('a, 'b, 'c)$ *psiCase*
and x :: *name*

shows $guarded\ P \Longrightarrow x \sharp (extractFrame\ P)$
and $guarded'\ I \Longrightarrow x \sharp (extractFrame'\ I)$
and $guarded''\ C \Longrightarrow x \sharp (extractFrame''\ C)$
by(*nominal-induct* P **and** I **and** C *arbitrary*: x *rule*: *psi-input-psiCase.strong-inducts*)
(*auto simp add*: *frameResChainFresh abs-fresh*)

lemma *frameResChainFresh'*:
fixes $xvec$ :: *name list*
and $yvec$ :: *name list*
and F :: $'b$ *frame*

shows $(xvec \sharp* ((\!|\nu*yvec|\!)F)) = (\forall x \in set\ xvec.\ x \in set\ yvec \vee x \sharp F)$
by(*simp add*: *frameResChainFresh fresh-star-def*)

lemma *frameChainFresh*[*simp*]:
fixes $xvec$ :: *name list*
and Ψ :: $'b$
and Xs :: *name set*

shows $xvec \sharp* (FAssert\ \Psi) = xvec \sharp* \Psi$
and $Xs \sharp* (FAssert\ \Psi) = Xs \sharp* \Psi$
by(*simp add*: *fresh-star-def*)+

lemma *frameResChainFresh''*[*simp*]:
fixes $xvec$ :: *name list*
and $yvec$ :: *name list*
and F :: $'b$ *frame*

assumes $xvec \sharp* yvec$

shows $xvec \sharp* ((\!|\nu*yvec|\!)F) = xvec \sharp* F$

using *assms*
by(*simp-all add*: *frameResChainFresh'*)
(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)

lemma *frameResChainFresh'''*[*simp*]:
fixes x :: *name*
and $xvec$ :: *name list*
and F :: $'b$ *frame*

assumes $x \sharp xvec$

shows $x \sharp ((\!|\nu*xvec|\!)F) = x \sharp F$
using *assms*
by(*induct xvec*) (*auto simp add*: *abs-fresh*)

lemma *FFreshBottom*[*simp*]:
fixes $xvec$ :: *name list*
and Xs :: *name set*

shows $xvec \sharp* (\bot_F)$
and $Xs \sharp* (\bot_F)$
by(*auto simp add*: *fresh-star-def*)

lemma *SFreshBottom*[*simp*]:
fixes $xvec$ :: *name list*
and Xs :: *name set*

shows $xvec \sharp* (SBottom)$

and $Xs \sharp* (SBottom)$
by(*auto simp add*: *fresh-star-def*)

lemma *freshFrameDest*[*dest*]:
fixes A_F :: *name list*
and Ψ_F :: $'b$
and *xvec* :: *name list*

assumes $xvec \sharp* (\langle A_F, \Psi_F\rangle)$

shows $xvec \sharp* A_F \Longrightarrow xvec \sharp* \Psi_F$
and $A_F \sharp* xvec \Longrightarrow xvec \sharp* \Psi_F$
proof −
from *assms* **have** $(set\ xvec) \sharp* (\langle A_F, \Psi_F\rangle)$
by(*simp add*: *fresh-star-def*)
moreover assume $xvec \sharp* A_F$
ultimately show $xvec \sharp* \Psi_F$
by(*simp add*: *frameResChainFreshSet*) (*force simp add*: *fresh-def name-list-supp fresh-star-def*)
next
from *assms* **have** $(set\ xvec) \sharp* (\langle A_F, \Psi_F\rangle)$
by(*simp add*: *fresh-star-def*)
moreover assume $A_F \sharp* xvec$
ultimately show $xvec \sharp* \Psi_F$
by(*simp add*: *frameResChainFreshSet*) (*force simp add*: *fresh-def name-list-supp fresh-star-def*)
qed

lemma *insertAssertionSimps*[*simp*]:
fixes A_F :: *name list*
and Ψ_F :: $'b$
and Ψ :: $'b$

assumes $A_F \sharp* \Psi$

shows $insertAssertion\ (\langle A_F, \Psi_F\rangle)\ \Psi = \langle A_F, \Psi \otimes \Psi_F\rangle$
using *assms*
by(*induct* A_F *arbitrary*: F) *auto*

lemma *mergeFrameSimps*[*simp*]:
fixes A_F :: *name list*
and Ψ_F :: $'b$
and Ψ :: $'b$

assumes $A_F \sharp* \Psi$

shows $(\langle A_F, \Psi_F\rangle) \otimes_F \langle \varepsilon, \Psi\rangle = \langle A_F, \Psi_F \otimes \Psi\rangle$
using *assms*
by(*induct* A_F *arbitrary*: F) *auto*

lemma *mergeFrames*[*simp*]:
  fixes A_F :: *name list*
  and Ψ_F :: $'b$
  and A_G :: *name list*
  and Ψ_G :: $'b$

  assumes $A_F \sharp* A_G$
  and $A_F \sharp* \Psi_G$
  and $A_G \sharp* \Psi_F$

  shows $(\langle A_F, \Psi_F\rangle) \otimes_F (\langle A_G, \Psi_G\rangle) = (\langle (A_F @ A_G), \Psi_F \otimes \Psi_G\rangle)$
using *assms*
by(*induct* A_F) *auto*

lemma *frameImpResFreshLeft*:
  fixes F :: $'b$ *frame*
  and x :: *name*

  assumes $x \sharp F$

  shows $(\!|\nu x|\!)F \hookrightarrow_F F$
proof(*auto simp add*: *FrameStatImp-def*)
  fix φ::$'c$
  obtain A_F Ψ_F **where** *Feq*: $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* (x, \varphi)$
    by(*rule freshFrame*)
  from ‹$A_F \sharp* (x, \varphi)$› **have** $x \sharp A_F$ **and** $A_F \sharp* \varphi$ **by** *simp*+
  obtain y **where** $y \sharp \varphi$ **and** $y \sharp F$ **and** $x \neq y$
    by(*generate-fresh name*, *auto*)

  assume $(\!|\nu x|\!)F \vdash_F \varphi$
  with ‹$y \sharp F$› **have** $(\!|\nu y|\!)([(x, y)] \cdot F) \vdash_F \varphi$ **by**(*simp add*: *alphaFrameRes*)
  with ‹$x \sharp F$› ‹$y \sharp F$› **have** $(\!|\nu y|\!)F \vdash_F \varphi$ **by** *simp*
  with *Feq* **have** $\langle (y\#A_F), \Psi_F\rangle \vdash_F \varphi$ **by** *simp*
  with *Feq* ‹$A_F \sharp* \varphi$› ‹$y \sharp \varphi$› **show** $F \vdash_F \varphi$
    by(*force intro*: *frameImpI dest*: *frameImpE simp del*: *frameResChain.simps*)
qed

lemma *frameImpResFreshRight*:
  fixes F :: $'b$ *frame*
  and x :: *name*

  assumes $x \sharp F$

  shows $F \hookrightarrow_F (\!|\nu x|\!)F$
proof(*auto simp add*: *FrameStatImp-def*)
  fix φ::$'c$
  obtain A_F Ψ_F **where** *Feq*: $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* (x, \varphi)$
    by(*rule freshFrame*)

from ‹A_F ♯* (x, φ)› **have** x ♯ A_F **and** A_F ♯* φ **by** *simp+*
obtain y **where** y ♯ φ **and** y ♯ F **and** $x \neq y$
by(*generate-fresh name*, *auto*)

assume $F \vdash_F \varphi$
with *Feq* ‹A_F ♯* φ› ‹y ♯ φ› **have** $\langle (y\#A_F), \Psi_F\rangle \vdash_F \varphi$
by(*force intro*: *frameImpI dest*: *frameImpE simp del*: *frameResChain.simps*)
moreover with ‹y ♯ F› ‹x ♯ F› *Feq* **show** $(\!|\nu x|\!)F \vdash_F \varphi$
by(*subst alphaFrameRes*) *auto*
qed

lemma *frameResFresh*:
fixes F :: *'b frame*
and x :: *name*

assumes x ♯ F

shows $(\!|\nu x|\!)F \simeq_F F$
using *assms*
by(*auto simp add*: *FrameStatEq-def intro*: *frameImpResFreshLeft frameImpRes-FreshRight*)

lemma *frameImpResPres*:
fixes F :: *'b frame*
and G :: *'b frame*
and x :: *name*

assumes $F \hookrightarrow_F G$

shows $(\!|\nu x|\!)F \hookrightarrow_F (\!|\nu x|\!)G$
proof(*auto simp add*: *FrameStatImp-def*)
fix φ::*'c*
obtain A_F Ψ_F **where** *Feq*: $F = \langle A_F, \Psi_F\rangle$ **and** A_F ♯* (x, φ)
by(*rule freshFrame*)
from ‹A_F ♯* (x, φ)› **have** x ♯ A_F **and** A_F ♯* φ **by** *simp+*
obtain y **where** y ♯ A_F **and** y ♯ F **and** y ♯ G
and $x \neq y$ **and** y ♯ φ
by(*generate-fresh name*, *auto*)
assume $(\!|\nu x|\!)F \vdash_F \varphi$
with ‹y ♯ F› **have** $(\!|\nu y|\!)([(x, y)] \cdot F) \vdash_F \varphi$ **by**(*simp add*: *alphaFrameRes*)
with *Feq* ‹x ♯ A_F› ‹y ♯ A_F› **have** $\langle (y\#A_F), [(x, y)] \cdot \Psi_F\rangle \vdash_F \varphi$ **by**(*simp add*: *eqvts*)
with ‹y ♯ φ› ‹A_F ♯* φ› **have** $\langle A_F, [(x, y)] \cdot \Psi_F\rangle \vdash_F \varphi$
by(*force intro*: *frameImpI dest*: *frameImpE simp del*: *frameResChain.simps*)
hence $([(x, y)] \cdot \langle A_F, [(x, y)] \cdot \Psi_F\rangle) \vdash_F ([(x, y)] \cdot \varphi)$
by(*rule frameImpClosed*)
with ‹x ♯ A_F› ‹y ♯ A_F› *Feq* **have** $F \vdash_F [(x, y)] \cdot \varphi$
by(*simp add*: *eqvts*)
with ‹$F \hookrightarrow_F G$› **have** $G \vdash_F [(x, y)] \cdot \varphi$ **by**(*simp add*: *FrameStatImp-def*)

obtain A_G Ψ_G **where** *Geq*: $G = \langle A_G, \Psi_G\rangle$ **and** $A_G \sharp* (x, y, \varphi)$
by(*rule freshFrame*)
from ‹$A_G \sharp* (x, y, \varphi)$› **have** $x \sharp A_G$ **and** $y \sharp A_G$ **and** $A_G \sharp* \varphi$ **by** *simp*+
from ‹$G \vdash_F [(x, y)] \cdot \varphi$› **have** $([(x, y)] \cdot G) \vdash_F [(x, y)] \cdot [(x, y)] \cdot \varphi$
by(*rule frameImpClosed*)
with *Geq* ‹$x \sharp A_G$› ‹$y \sharp A_G$› **have** $\langle A_G, [(x, y)] \cdot \Psi_G\rangle \vdash_F \varphi$ **by**(*simp add*: *eqvts*)
with ‹$y \sharp \varphi$› ‹$A_G \sharp* \varphi$› **have** $\langle (y\#A_G), [(x, y)] \cdot \Psi_G\rangle \vdash_F \varphi$
by(*force intro*: *frameImpI dest*: *frameImpE simp del*: *frameResChain.simps*)
with ‹$y \sharp G$› ‹$x \sharp A_G$› ‹$y \sharp A_G$› *Geq* **show** $(\!|\nu x|\!) G \vdash_F \varphi$
by(*subst alphaFrameRes*) (*fastforce simp add*: *eqvts*)+
qed

lemma *frameResPres*:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*
and x :: *name*

assumes $F \simeq_F G$

shows $(\!|\nu x|\!) F \simeq_F (\!|\nu x|\!) G$
using *assms*
by(*auto simp add*: *FrameStatEq-def intro*: *frameImpResPres*)

lemma *frameImpResComm*:
fixes x :: *name*
and y :: *name*
and F :: $'b$ *frame*

shows $(\!|\nu x|\!)((\!|\nu y|\!) F) \hookrightarrow_F (\!|\nu y|\!)((\!|\nu x|\!) F)$
proof(*case-tac* $x = y$)
assume $x = y$
thus *?thesis* **by** *simp*
next
assume $x \neq y$
show *?thesis*
proof(*auto simp add*: *FrameStatImp-def*)
fix φ::$'c$
obtain A_F Ψ_F **where** *Feq*: $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* (x, y, \varphi)$
by(*rule freshFrame*)
then have $x \sharp A_F$ **and** $y \sharp A_F$ **and** $A_F \sharp* \varphi$ **by** *simp*+

obtain x'::*name* **where** $x' \neq x$ **and** $x' \neq y$ **and** $x' \sharp F$ **and** $x' \sharp \varphi$ **and** $x' \sharp A_F$
by(*generate-fresh name*) *auto*
obtain y'::*name* **where** $y' \neq x$ **and** $y' \neq y$ **and** $y' \neq x'$ **and** $y' \sharp F$ **and** $y' \sharp \varphi$ **and** $y' \sharp A_F$
by(*generate-fresh name*) *auto*

from ‹$y' \sharp F$› **have** $(\!|\nu x|\!)((\!|\nu y|\!)F) = (\!|\nu x|\!)((\!|\nu y'|\!)([(y, y')] \cdot F))$
by(*simp add*: *alphaFrameRes*)
moreover from ‹$x' \sharp F$› ‹$x' \neq y$› ‹$y' \neq x'$› **have** $\ldots = (\!|\nu x'|\!)([(x, x')] \cdot ((\!|\nu y'|\!)([(y, y')] \cdot F)))$
by(*rule-tac alphaFrameRes*) (*simp add*: *abs-fresh fresh-left*)
moreover with ‹$y' \neq x'$› ‹$y' \neq x$› **have** $\ldots = (\!|\nu x'|\!)((\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot F))$
by(*simp add*: *eqvts calc-atm*)
ultimately have *A*: $(\!|\nu x|\!)((\!|\nu y|\!)F) = (\!|\nu x'|\!)((\!|\nu y'|\!)((\!|\nu * A_F|\!)(FAssert([(x, x')] \cdot [(y, y')] \cdot \Psi_F))))$
using *Feq* ‹$x \sharp A_F$› ‹$x' \sharp A_F$› ‹$y \sharp A_F$› ‹$y' \sharp A_F$›
by(*simp add*: *eqvts*)

from ‹$x' \sharp F$› **have** $(\!|\nu y|\!)((\!|\nu x|\!)F) = (\!|\nu y|\!)((\!|\nu x'|\!)([(x, x')] \cdot F))$
by(*simp add*: *alphaFrameRes*)
moreover from ‹$y' \sharp F$› ‹$y' \neq x$› ‹$y' \neq x'$› **have** $\ldots = (\!|\nu y'|\!)([(y, y')] \cdot ((\!|\nu x'|\!)([(x, x')] \cdot F)))$
by(*rule-tac alphaFrameRes*) (*simp add*: *abs-fresh fresh-left*)
moreover with ‹$y' \neq x'$› ‹$x' \neq y$› **have** $\ldots = (\!|\nu y'|\!)((\!|\nu x'|\!)([(y, y')] \cdot [(x, x')] \cdot F))$
by(*simp add*: *eqvts calc-atm*)
moreover with ‹$x' \neq x$› ‹$x' \neq y$› ‹$y' \neq x$› ‹$y' \neq y$› ‹$y' \neq x'$› ‹$x \neq y$›
have $\ldots = (\!|\nu y'|\!)((\!|\nu x'|\!)([(x, x')] \cdot [(y, y')] \cdot F))$
apply(*simp add*: *eqvts*)
by(*subst perm-compose*) (*simp add*: *calc-atm*)
ultimately have *B*: $(\!|\nu y|\!)((\!|\nu x|\!)F) = (\!|\nu y'|\!)((\!|\nu x'|\!)((\!|\nu * A_F|\!)(FAssert([(x, x')] \cdot [(y, y')] \cdot \Psi_F))))$
using *Feq* ‹$x \sharp A_F$› ‹$x' \sharp A_F$› ‹$y \sharp A_F$› ‹$y' \sharp A_F$›
by(*simp add*: *eqvts*)

from ‹$x' \sharp \varphi$› ‹$y' \sharp \varphi$› ‹$A_F \sharp * \varphi$›
have $\langle (x' \# y' \# A_F), [(x, x')] \cdot [(y, y')] \cdot \Psi_F \rangle \vdash_F \varphi = \langle (y' \# x' \# A_F), [(x, x')] \cdot [(y, y')] \cdot \Psi_F \rangle \vdash_F \varphi$
by(*force dest*: *frameImpE intro*: *frameImpI simp del*: *frameResChain.simps*)
with *A B* **have** $((\!|\nu x|\!)((\!|\nu y|\!)F)) \vdash_F \varphi = ((\!|\nu y|\!)((\!|\nu x|\!)F)) \vdash_F \varphi$
by *simp*
moreover assume $((\!|\nu x|\!)((\!|\nu y|\!)F)) \vdash_F \varphi$
ultimately show $((\!|\nu y|\!)((\!|\nu x|\!)F)) \vdash_F \varphi$ **by** *simp*
qed
qed

lemma *frameResComm*:
fixes x :: *name*
and y :: *name*
and F :: *'b frame*

shows $(\!|\nu x|\!)((\!|\nu y|\!)F) \simeq_F (\!|\nu y|\!)((\!|\nu x|\!)F)$
by(*auto simp add*: *FrameStatEq-def intro*: *frameImpResComm*)

lemma *frameImpResCommLeft'*:
fixes x :: *name*
and *xvec* :: *name list*
and F :: *'b frame*

shows $(\!|\nu x|\!)((\!|\nu{*}xvec|\!)F) \hookrightarrow_F (\!|\nu{*}xvec|\!)((\!|\nu x|\!)F)$
by(*induct xvec*) (*auto intro*: *frameImpResComm FrameStatImpTrans frameImpResPres*)

lemma *frameImpResCommRight'*:
fixes x :: *name*
and *xvec* :: *name list*
and F :: *'b frame*

shows $(\!|\nu{*}xvec|\!)((\!|\nu x|\!)F) \hookrightarrow_F (\!|\nu x|\!)((\!|\nu{*}xvec|\!)F)$
by(*induct xvec*) (*auto intro*: *frameImpResComm FrameStatImpTrans frameImpResPres*)

lemma *frameResComm'*:
fixes x :: *name*
and *xvec* :: *name list*
and F :: *'b frame*

shows $(\!|\nu x|\!)((\!|\nu{*}xvec|\!)F) \simeq_F (\!|\nu{*}xvec|\!)((\!|\nu x|\!)F)$
by(*induct xvec*) (*auto intro*: *frameResComm FrameStatEqTrans frameResPres*)

lemma *frameImpChainComm*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and F :: *'b frame*

shows $(\!|\nu{*}xvec|\!)((\!|\nu{*}yvec|\!)F) \hookrightarrow_F (\!|\nu{*}yvec|\!)((\!|\nu{*}xvec|\!)F)$
by(*induct xvec*) (*auto intro*: *frameImpResCommLeft' FrameStatImpTrans frameImpResPres*)

lemma *frameResChainComm*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and F :: *'b frame*

shows $(\!|\nu{*}xvec|\!)((\!|\nu{*}yvec|\!)F) \simeq_F (\!|\nu{*}yvec|\!)((\!|\nu{*}xvec|\!)F)$
by(*induct xvec*) (*auto intro*: *frameResComm' FrameStatEqTrans frameResPres*)

lemma *frameImpNilStatEq*[*simp*]:
fixes Ψ :: *'b*
and Ψ' :: *'b*

shows $(\langle\varepsilon, \Psi\rangle \hookrightarrow_F \langle\varepsilon, \Psi'\rangle) = (\Psi \hookrightarrow \Psi')$
by(*simp add*: *FrameStatImp-def AssertionStatImp-def FrameImp-def*)

lemma *frameNilStatEq*[*simp*]:
fixes Ψ :: $'b$
and Ψ' :: $'b$

shows $(\langle\varepsilon, \Psi\rangle \simeq_F \langle\varepsilon, \Psi'\rangle) = (\Psi \simeq \Psi')$
by(*simp add*: *FrameStatEq-def AssertionStatEq-def FrameImp-def*)

lemma *extractFrameChainStatImp*:
fixes *xvec* :: *name list*
and P :: $('a, 'b, 'c)$ *psi*

shows $extractFrame((\!|\nu{*}xvec|\!)P) \hookrightarrow_F (\!|\nu{*}xvec|\!)(extractFrame\ P)$
by(*induct xvec*) (*auto intro*: *frameImpResPres*)

lemma *extractFrameChainStatEq*:
fixes *xvec* :: *name list*
and P :: $('a, 'b, 'c)$ *psi*

shows $extractFrame((\!|\nu{*}xvec|\!)P) \simeq_F (\!|\nu{*}xvec|\!)(extractFrame\ P)$
by(*induct xvec*) (*auto intro*: *frameResPres*)

lemma *insertAssertionExtractFrameFreshImp*:
fixes *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

assumes $xvec \sharp* \Psi$

shows $insertAssertion(extractFrame((\!|\nu{*}xvec|\!)P))\ \Psi \hookrightarrow_F (\!|\nu{*}xvec|\!)(insertAssertion\ (extractFrame\ P)\ \Psi)$
using *assms*
by(*induct xvec*) (*auto intro*: *frameImpResPres*)

lemma *insertAssertionExtractFrameFresh*:
fixes *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

assumes $xvec \sharp* \Psi$

shows $insertAssertion(extractFrame((\!|\nu{*}xvec|\!)P))\ \Psi \simeq_F (\!|\nu{*}xvec|\!)(insertAssertion\ (extractFrame\ P)\ \Psi)$
using *assms*
by(*induct xvec*) (*auto intro*: *frameResPres*)

lemma *frameImpResChainPres*:
fixes F :: $'b$ *frame*

and G :: *'b frame*
and *xvec* :: *name list*

assumes $F \hookrightarrow_F G$

shows $(\!|\nu{*}xvec|\!)F \hookrightarrow_F (\!|\nu{*}xvec|\!)G$
using *assms*
by(*induct xvec*) (*auto intro*: *frameImpResPres*)

lemma *frameResChainPres*:
fixes F :: *'b frame*
and G :: *'b frame*
and *xvec* :: *name list*

assumes $F \simeq_F G$

shows $(\!|\nu{*}xvec|\!)F \simeq_F (\!|\nu{*}xvec|\!)G$
using *assms*
by(*induct xvec*) (*auto intro*: *frameResPres*)

lemma *insertAssertionE*:
fixes F :: (*'b*::*fs-name*) *frame*
and Ψ :: *'b*
and Ψ' :: *'b*
and A_F :: *name list*

assumes *insertAssertion* $F\ \Psi = \langle A_F, \Psi' \rangle$
and $A_F \sharp{*}\ F$
and $A_F \sharp{*}\ \Psi$
and *distinct* A_F

obtains Ψ_F **where** $F = \langle A_F, \Psi_F \rangle$ **and** $\Psi' = \Psi \otimes \Psi_F$
proof −
assume A: $\bigwedge \Psi_F.\ [\![F = \langle A_F, \Psi_F \rangle;\ \Psi' = \Psi \otimes \Psi_F]\!] \Longrightarrow$ *thesis*
from *assms* **have** $\exists\, \Psi_F.\ F = \langle A_F, \Psi_F \rangle \wedge \Psi' = \Psi \otimes \Psi_F$
proof(*nominal-induct F avoiding*: $\Psi\ A_F\ \Psi'$ *rule*: *frame.strong-induct*)
case(*FAssert* $\Psi\ A_F\ \Psi'$)
thus *?case* **by** *auto*
next
case(*FRes x F* $\Psi\ A_F\ \Psi'$)
from ‹*insertAssertion* $((\!|\nu x|\!)F)\ \Psi = \langle A_F, \Psi' \rangle$› ‹$x \sharp \Psi$›
obtain $y\ A_F{}'$ **where** $A_F = y\#A_F{}'$ **by**(*induct* A_F) *auto*
with ‹*insertAssertion* $((\!|\nu x|\!)F)\ \Psi = \langle A_F, \Psi' \rangle$› ‹$x \sharp \Psi$› ‹$x \sharp A_F$›
have A: *insertAssertion* $F\ \Psi = \langle ([(x, y)] \cdot A_F{}'), [(x, y)] \cdot \Psi' \rangle$
by(*simp add*: *frame.inject alpha eqvts*)
from ‹$A_F = y\#A_F{}'$› ‹$A_F \sharp{*}\ \Psi$› **have** $y \sharp \Psi$ **and** $A_F{}' \sharp{*}\ \Psi$ **by** *simp+*
from ‹*distinct* A_F› ‹$A_F = y\#A_F{}'$› **have** $y \sharp A_F{}'$ **and** *distinct* $A_F{}'$ **by** *auto*
from ‹$A_F \sharp{*}\ ((\!|\nu x|\!)F)$› ‹$x \sharp A_F$› ‹$A_F = y\#A_F{}'$› **have** $y \sharp F$ **and** $A_F{}' \sharp{*}\ F$
and $x \sharp A_F{}'$

apply −
apply(*auto simp add*: *abs-fresh*)
apply(*hypsubst-thin*)
apply(*subst fresh-star-def*)
apply(*erule rev-mp*)
apply(*subst fresh-star-def*)
apply(*clarify*)
apply(*erule-tac x=xa* **in** *ballE*)
apply(*simp add*: *abs-fresh*)
apply *auto*
by(*simp add*: *fresh-def name-list-supp*)
with ‹$x \sharp A_F{}'$› ‹$y \sharp A_F{}'$› **have** $([(x, y)] \cdot A_F{}') \sharp* F$ **by** *simp*
from ‹$A_F{}' \sharp* \Psi$› **have** $([(x, y)] \cdot A_F{}') \sharp* ([(x, y)] \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $([(x, y)] \cdot A_F{}') \sharp* \Psi$ **by** *simp*
with ‹$\bigwedge \Psi\ A_F\ \Psi'.\ [\![insertAssertion\ F\ \Psi = \langle A_F, \Psi'\rangle; A_F \sharp* F; A_F \sharp* \Psi; distinct\ A_F]\!] \Longrightarrow \exists \Psi_F.\ F = \langle A_F, \Psi_F\rangle \wedge \Psi' = \Psi \otimes \Psi_F$› A
‹$([(x, y)] \cdot A_F{}') \sharp* F$› ‹$distinct\ A_F{}'$› ‹$x \sharp A_F{}'$› ‹$y \sharp A_F{}'$›
obtain Ψ_F **where** *Feq*: $F = \langle A_F{}', \Psi_F\rangle$ **and** Ψeq: $([(x, y)] \cdot \Psi') = \Psi \otimes \Psi_F$
by *force*

from *Feq* **have** $(\!|\nu x|\!)F = \langle (x\#A_F{}'), \Psi_F\rangle$ **by**(*simp add*: *frame.inject*)
hence $([(x, y)] \cdot (\!|\nu x|\!)F) = [(x, y)] \cdot \langle (x\#A_F{}'), \Psi_F\rangle$ **by** *simp*
hence $(\!|\nu x|\!)F = \langle A_F, [(x, y)] \cdot \Psi_F\rangle$ **using** ‹$y \sharp F$› ‹$A_F = y\#A_F{}'$› ‹$x \sharp A_F$› ‹$y \sharp A_F{}'$›
by(*simp add*: *eqvts calc-atm alphaFrameRes*)

moreover from Ψeq **have** $[(x, y)] \cdot ([(x, y)] \cdot \Psi') = [(x, y)] \cdot (\Psi \otimes \Psi_F)$
by *simp*
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi' = \Psi \otimes ([(x, y)] \cdot \Psi_F)$ **by**(*simp add*: *eqvts*)
ultimately show *?case*
by *blast*
qed
with A **show** *?thesis*
by *blast*
qed

lemma *mergeFrameE*:
fixes F :: *'b frame*
and G :: *'b frame*
and A_{FG} :: *name list*
and Ψ_{FG} :: *'b*

assumes *mergeFrame* $F\ G = \langle A_{FG}, \Psi_{FG}\rangle$
and *distinct* A_{FG}
and $A_{FG} \sharp* F$
and $A_{FG} \sharp* G$

obtains $A_F\ \Psi_F\ A_G\ \Psi_G$ **where** $A_{FG} = A_F@A_G$ **and** $\Psi_{FG} = \Psi_F \otimes \Psi_G$ **and** F

$= \langle A_F, \Psi_F\rangle$ **and** $G = \langle A_G, \Psi_G\rangle$ **and** $A_F \sharp* \Psi_G$ **and** $A_G \sharp* \Psi_F$
proof −
assume A: $\bigwedge A_F\ A_G\ \Psi_F\ \Psi_G.\ [\![A_{FG} = A_F@A_G;\ \Psi_{FG} = \Psi_F \otimes \Psi_G;\ F = \langle A_F, \Psi_F\rangle;\ G = \langle A_G, \Psi_G\rangle;\ A_F \sharp* \Psi_G;\ A_G \sharp* \Psi_F]\!] \Longrightarrow thesis$
from *assms* **have** $\exists A_F\ \Psi_F\ A_G\ \Psi_G.\ A_{FG} = A_F@A_G \wedge \Psi_{FG} = \Psi_F \otimes \Psi_G \wedge F = \langle A_F, \Psi_F\rangle \wedge G = \langle A_G, \Psi_G\rangle \wedge A_F \sharp* \Psi_G \wedge A_G \sharp* \Psi_F$
proof(*nominal-induct F avoiding*: $G\ A_{FG}\ \Psi_{FG}$ *rule*: *frame.strong-induct*)
case(*FAssert* $\Psi\ G\ A_{FG}\ \Psi_{FG}$)
thus *?case*
apply *auto*
apply(*rule-tac x*=[] **in** *exI*)
by(*drule-tac insertAssertionE*) *auto*
next
case(*FRes x F G* $A_{FG}\ \Psi_{FG}$)
from ‹*mergeFrame* $((\!|\nu x|\!)F)\ G = \langle A_{FG}, \Psi_{FG}\rangle$› ‹$x \sharp G$›
obtain $y\ A_{FG}'$ **where** $A_{FG} = y\#A_{FG}'$ **by**(*induct* A_{FG}) *auto*
with ‹$A_{FG} \sharp* ((\!|\nu x|\!)F)$› ‹$x \sharp A_{FG}$› **have** $A_{FG}' \sharp* F$ **and** $x \sharp A_{FG}'$
by(*auto simp add*: *supp-list-cons fresh-star-def fresh-def name-list-supp abs-supp frame.supp*)
from ‹$A_{FG} = y\#A_{FG}'$› ‹$A_{FG} \sharp* G$› **have** $y \sharp G$ **and** $A_{FG}' \sharp* G$ **by** *simp+*
from ‹$A_{FG} = y\#A_{FG}'$› ‹$A_{FG} \sharp* ((\!|\nu x|\!)F)$› ‹$x \sharp A_{FG}$› **have** $y \sharp F$ **and** $A_{FG}' \sharp* F$
apply(*auto simp add*: *abs-fresh frameResChainFreshSet*)
apply(*hypsubst-thin*)
by(*induct* A_{FG}') (*auto simp add*: *abs-fresh*)
from ‹*distinct* A_{FG}› ‹$A_{FG} = y\#A_{FG}'$› **have** $y \sharp A_{FG}'$ **and** *distinct* A_{FG}' **by** *auto*

with ‹$A_{FG} = y\#A_{FG}'$› ‹*mergeFrame* $((\!|\nu x|\!)F)\ G = \langle A_{FG}, \Psi_{FG}\rangle$› ‹$x \sharp G$› ‹$x \sharp A_{FG}$› ‹$y \sharp A_{FG}'$›
have *mergeFrame* $F\ G = \langle A_{FG}', [(x, y)] \cdot \Psi_{FG}\rangle$
by(*simp add*: *frame.inject alpha eqvts*)
with ‹*distinct* A_{FG}'› ‹$A_{FG}' \sharp* F$› ‹$A_{FG}' \sharp* G$›
‹$\bigwedge G\ A_{FG}\ \Psi_{FG}.\ [\![mergeFrame\ F\ G = \langle A_{FG}, \Psi_{FG}\rangle;\ distinct\ A_{FG};\ A_{FG} \sharp* F;\ A_{FG} \sharp* G]\!] \Longrightarrow \exists A_F\ \Psi_F\ A_G\ \Psi_G.\ A_{FG} = A_F@A_G \wedge \Psi_{FG} = \Psi_F \otimes \Psi_G \wedge F = \langle A_F, \Psi_F\rangle \wedge G = \langle A_G, \Psi_G\rangle \wedge A_F \sharp* \Psi_G \wedge A_G \sharp* \Psi_F$›
obtain $A_F\ \Psi_F\ A_G\ \Psi_G$ **where** $A_{FG}' = A_F@A_G$ **and** $([(x, y)] \cdot \Psi_{FG}) = \Psi_F \otimes \Psi_G$ **and** *FrF*: $F = \langle A_F, \Psi_F\rangle$ **and** *FrG*: $G = \langle A_G, \Psi_G\rangle$ **and** $A_F \sharp* \Psi_G$ **and** $A_G \sharp* \Psi_F$
by *metis*

from ‹$A_{FG}' = A_F@A_G$› ‹$A_{FG} = y\#A_{FG}'$› **have** $A_{FG} = (y\#A_F)@A_G$ **by** *simp*
moreover from ‹$A_{FG}' = A_F@A_G$› ‹$y \sharp A_{FG}'$› ‹$x \sharp A_{FG}'$› **have** $x \sharp A_F$ **and** $y \sharp A_F$ **and** $x \sharp A_G$ **and** $y \sharp A_G$ **by** *simp+*
with ‹$y \sharp G$› ‹$x \sharp G$› ‹$x \sharp A_{FG}$› *FrG* **have** $y \sharp \Psi_G$ **and** $x \sharp \Psi_G$
by *auto*
from ‹$([(x, y)] \cdot \Psi_{FG}) = \Psi_F \otimes \Psi_G$› **have** $([(x, y)] \cdot [(x, y)] \cdot \Psi_{FG}) = [(x, y)] \cdot (\Psi_F \otimes \Psi_G)$

by *simp*
with ‹$x \sharp \Psi_G$› ‹$y \sharp \Psi_G$› **have** $\Psi_{FG} = ([(x, y)] \cdot \Psi_F) \otimes \Psi_G$ **by**(*simp add*: *eqvts*)
moreover from *FrF* **have** $([(x, y)] \cdot F) = [(x, y)] \cdot \langle A_F, \Psi_F\rangle$ **by** *simp*
with ‹$x \sharp A_F$› ‹$y \sharp A_F$› **have** $([(x, y)] \cdot F) = \langle A_F, [(x, y)] \cdot \Psi_F\rangle$ **by**(*simp add*: *eqvts*)
hence $(\!|\nu y|\!)([(x, y)] \cdot F) = \langle (y\#A_F), [(x, y)] \cdot \Psi_F\rangle$ **by**(*simp add*: *frame.inject*)
with ‹$y \sharp F$› **have** $(\!|\nu x|\!)F = \langle (y\#A_F), [(x, y)] \cdot \Psi_F\rangle$ **by**(*simp add*: *alphaFrameRes*)
moreover with ‹$A_G \sharp\!* \Psi_F$› **have** $([(x, y)] \cdot A_G) \sharp\!* ([(x, y)] \cdot \Psi_F)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_G$› ‹$y \sharp A_G$› **have** $A_G \sharp\!* ([(x, y)] \cdot \Psi_F)$ **by** *simp*
moreover from ‹$A_F \sharp\!* \Psi_G$› ‹$y \sharp \Psi_G$› **have** $(y\#A_F) \sharp\!* \Psi_G$ **by** *simp*
ultimately show *?case* **using** *FrG*
by *blast*
qed
with *A* **show** *?thesis* **by** *blast*
qed

lemma *mergeFrameRes1*[*simp*]:
fixes A_F :: *name list*
and Ψ_F :: *'b*
and x :: *name*
and A_G :: *name list*
and Ψ_G :: *'b*

assumes $A_F \sharp\!* \Psi_G$
and $A_F \sharp\!* A_G$
and $x \sharp A_F$
and $x \sharp \Psi_F$
and $A_G \sharp\!* \Psi_F$

shows $(\langle A_F, \Psi_F\rangle) \otimes_F ((\!|\nu x|\!)(\langle A_G, \Psi_G\rangle)) = (\langle (A_F@x\#A_G), \Psi_F \otimes \Psi_G\rangle)$
using *assms*
apply(*fold frameResChain.simps*)
by(*rule mergeFrames*) *auto*

lemma *mergeFrameRes2*[*simp*]:
fixes A_F :: *name list*
and Ψ_F :: *'b*
and x :: *name*
and A_G :: *name list*
and Ψ_G :: *'b*

assumes $A_F \sharp\!* \Psi_G$
and $A_G \sharp\!* A_F$
and $x \sharp A_F$
and $x \sharp \Psi_F$
and $A_G \sharp\!* \Psi_F$

shows $(\langle A_F,\ \Psi_F\rangle) \otimes_F (\llparenthesis \nu x \rrparenthesis(\langle A_G,\ \Psi_G\rangle)) = (\langle (A_F @ x \# A_G),\ \Psi_F \otimes \Psi_G\rangle)$
using *assms*
apply(*fold frameResChain.simps*)
by(*rule mergeFrames*) *auto*

lemma *insertAssertionResChain*[*simp*]:
fixes *xvec* :: *name list*
and F :: $'b$ *frame*
and Ψ :: $'b$

assumes *xvec* $\sharp*$ Ψ

shows *insertAssertion* $(\llparenthesis \nu{*}xvec \rrparenthesis F)\ \Psi = \llparenthesis \nu{*}xvec \rrparenthesis(insertAssertion\ F\ \Psi)$
using *assms*
by(*induct xvec*) *auto*

lemma *extractFrameResChain*[*simp*]:
fixes *xvec* :: *name list*
and P :: $('a,\ 'b,\ 'c)$ *psi*

shows $extractFrame(\llparenthesis \nu{*}xvec \rrparenthesis P) = \llparenthesis \nu{*}xvec \rrparenthesis(extractFrame\ P)$
by(*induct xvec*) *auto*

lemma *frameResFreshChain*:
fixes *xvec* :: *name list*
and F :: $'b$ *frame*

assumes *xvec* $\sharp*$ F

shows $\llparenthesis \nu{*}xvec \rrparenthesis F \simeq_F F$
using *assms*
proof(*induct xvec*)
case *Nil*
thus *?case* **by** *simp*
next
case(*Cons x xvec*)
thus *?case*
by *auto* (*metis frameResPres frameResFresh FrameStatEqTrans*)
qed

end

locale *assertion* = *assertionAux SCompose SImp SBottom SChanEq*
for *SCompose* :: $'b$::*fs-name* $\Rightarrow$ $'b$ $\Rightarrow$ $'b$
and *SImp* :: $'b$ $\Rightarrow$ $'c$::*fs-name* $\Rightarrow$ *bool*
and *SBottom* :: $'b$
and *SChanEq* :: $'a$::*fs-name* $\Rightarrow$ $'a$ $\Rightarrow$ $'c$ +

assumes *chanEqSym*: *SImp* Ψ (*SChanEq M N*) $\Longrightarrow$ *SImp* Ψ (*SChanEq N M*)

and *chanEqTrans*: $[\![SImp\ \Psi\ (SChanEq\ M\ N);\ SImp\ \Psi\ (SChanEq\ N\ L)]\!] \Longrightarrow SImp\ \Psi\ (SChanEq\ M\ L)$
and *Composition*: $assertionAux.AssertionStatEq\ SImp\ \Psi\ \Psi' \Longrightarrow assertionAux.AssertionStatEq\ SImp\ (SCompose\ \Psi\ \Psi'')\ (SCompose\ \Psi'\ \Psi'')$
and *Identity*: $assertionAux.AssertionStatEq\ SImp\ (SCompose\ \Psi\ SBottom)\ \Psi$
and *Associativity*: $assertionAux.AssertionStatEq\ SImp\ (SCompose\ (SCompose\ \Psi\ \Psi')\ \Psi'')\ (SCompose\ \Psi\ (SCompose\ \Psi'\ \Psi''))$
and *Commutativity*: $assertionAux.AssertionStatEq\ SImp\ (SCompose\ \Psi\ \Psi')\ (SCompose\ \Psi'\ \Psi)$

begin

notation *SCompose* (**infixr** ‹⊗› *90*)
notation *SImp* (‹- ⊢ -› [*85*, *85*] *85*)
notation *SChanEq* (‹- ↔ -› [*90*, *90*] *90*)
notation *SBottom* (‹⊥› *90*)

lemma *compositionSym*:
fixes Ψ :: *'b*
and Ψ' :: *'b*
and Ψ'' :: *'b*

assumes $\Psi \simeq \Psi'$

shows $\Psi'' \otimes \Psi \simeq \Psi'' \otimes \Psi'$
proof −
have $\Psi'' \otimes \Psi \simeq \Psi \otimes \Psi''$ **by**(*rule Commutativity*)
moreover from *assms* **have** $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$ **by**(*rule Composition*)
moreover have $\Psi' \otimes \Psi'' \simeq \Psi'' \otimes \Psi'$ **by**(*rule Commutativity*)
ultimately show *?thesis* **by**(*blast intro*: *AssertionStatEqTrans*)
qed

lemma *Composition'*:
fixes Ψ :: *'b*
and Ψ' :: *'b*
and Ψ'' :: *'b*
and Ψ''' :: *'b*

assumes $\Psi \simeq \Psi'$
and $\Psi'' \simeq \Psi'''$

shows $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi'''$
using *assms*
by(*metis Composition Commutativity AssertionStatEqTrans*)

lemma *composition'*:
fixes Ψ :: *'b*

and Ψ' :: $'b$
and Ψ'' :: $'b$
and Ψ''' :: $'b$

assumes $\Psi \simeq \Psi'$

shows $(\Psi \otimes \Psi'') \otimes \Psi''' \simeq (\Psi' \otimes \Psi'') \otimes \Psi'''$
proof −
have $(\Psi \otimes \Psi'') \otimes \Psi''' \simeq \Psi \otimes (\Psi'' \otimes \Psi''')$
by(*rule Associativity*)
moreover from *assms* **have** $\Psi \otimes (\Psi'' \otimes \Psi''') \simeq \Psi' \otimes (\Psi'' \otimes \Psi''')$
by(*rule Composition*)
moreover have $\Psi' \otimes (\Psi'' \otimes \Psi''') \simeq (\Psi' \otimes \Psi'') \otimes \Psi'''$
by(*rule Associativity*[*THEN AssertionStatEqSym*])
ultimately show *?thesis* **by**(*blast dest*: *AssertionStatEqTrans*)
qed

lemma *associativitySym*:
fixes Ψ :: $'b$
and Ψ' :: $'b$
and Ψ'' :: $'b$

shows $(\Psi \otimes \Psi') \otimes \Psi'' \simeq (\Psi \otimes \Psi'') \otimes \Psi'$
proof −
have $(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$
by(*rule Associativity*)
moreover have $\Psi \otimes (\Psi' \otimes \Psi'') \simeq \Psi \otimes (\Psi'' \otimes \Psi')$
by(*rule compositionSym*[*OF Commutativity*])
moreover have $\Psi \otimes (\Psi'' \otimes \Psi') \simeq (\Psi \otimes \Psi'') \otimes \Psi'$
by(*rule AssertionStatEqSym*[*OF Associativity*])
ultimately show *?thesis*
by(*blast dest*: *AssertionStatEqTrans*)
qed

lemma *frameIntAssociativity*:
fixes A_F :: *name list*
and Ψ :: $'b$
and Ψ' :: $'b$
and Ψ'' :: $'b$

shows $\langle A_F, (\Psi \otimes \Psi') \otimes \Psi'' \rangle \simeq_F \langle A_F, \Psi \otimes (\Psi' \otimes \Psi'') \rangle$
by(*induct* A_F) (*auto intro*: *Associativity frameResPres*)

lemma *frameIntCommutativity*:
fixes A_F :: *name list*
and Ψ :: $'b$
and Ψ' :: $'b$

shows $\langle A_F, \Psi \otimes \Psi' \rangle \simeq_F \langle A_F, \Psi' \otimes \Psi \rangle$

by(*induct* A_F) (*auto intro*: *Commutativity frameResPres*)

lemma *frameIntIdentity*:
fixes A_F :: *name list*
and Ψ_F :: $'b$

shows $\langle A_F, \Psi_F \otimes SBottom\rangle \simeq_F \langle A_F, \Psi_F\rangle$
by(*induct* A_F) (*auto intro*: *Identity frameResPres*)

lemma *frameIntComposition*:
fixes Ψ :: $'b$
and Ψ' :: $'b$
and A_F :: *name list*
and Ψ_F :: $'b$

assumes $\Psi \simeq \Psi'$

shows $\langle A_F, \Psi \otimes \Psi_F\rangle \simeq_F \langle A_F, \Psi' \otimes \Psi_F\rangle$
using *assms*
by(*induct* A_F) (*auto intro*: *Composition frameResPres*)

lemma *frameIntCompositionSym*:
fixes Ψ :: $'b$
and Ψ' :: $'b$
and A_F :: *name list*
and Ψ_F :: $'b$

assumes $\Psi \simeq \Psi'$

shows $\langle A_F, \Psi_F \otimes \Psi\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi'\rangle$
using *assms*
by(*induct* A_F) (*auto intro*: *compositionSym frameResPres*)

lemma *frameCommutativity*:
fixes F :: $'b$ *frame*
and G :: $'b$ *frame*

shows $F \otimes_F G \simeq_F G \otimes_F F$
proof −
obtain A_F Ψ_F **where** $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* G$
by(*rule freshFrame*)
moreover obtain A_G Ψ_G **where** $G = \langle A_G, \Psi_G\rangle$ **and** $A_G \sharp* \Psi_F$ **and** $A_G \sharp* A_F$
by(*rule-tac* C=(A_F, Ψ_F) **in** *freshFrame*) *auto*
moreover from ‹$A_F \sharp* G$› ‹$G = \langle A_G, \Psi_G\rangle$› ‹$A_G \sharp* A_F$› **have** $A_F \sharp* \Psi_G$
by *auto*
ultimately show *?thesis*
by *auto* (*metis FrameStatEqTrans frameChainAppend frameResChainComm frameIntCommutativity*)
qed

lemma *frameScopeExt*:
fixes x :: *name*
and F :: $'b$ *frame*
and G :: $'b$ *frame*

assumes $x \sharp F$

shows $(\!|\nu x|\!)(F \otimes_F G) \simeq_F F \otimes_F ((\!|\nu x|\!)G)$
proof −
have $(\!|\nu x|\!)(F \otimes_F G) \simeq_F (\!|\nu x|\!)(G \otimes_F F)$
by(*metis frameResPres frameCommutativity*)
with ‹$x \sharp F$› **have** $(\!|\nu x|\!)(F \otimes_F G) \simeq_F ((\!|\nu x|\!)G) \otimes_F F$
by *simp*
moreover have $((\!|\nu x|\!)G) \otimes_F F \simeq_F F \otimes_F ((\!|\nu x|\!)G)$
by(*rule frameCommutativity*)
ultimately show *?thesis* **by**(*rule FrameStatEqTrans*)
qed

lemma *insertDoubleAssertionStatEq*:
fixes F :: $'b$ *frame*
and Ψ :: $'b$
and Ψ' :: $'b$

shows *insertAssertion*(*insertAssertion* F Ψ) $\Psi' \simeq_F$ (*insertAssertion* F) ($\Psi \otimes \Psi'$)
proof −
obtain A_F Ψ_F **where** $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* \Psi$ **and** $A_F \sharp* \Psi'$ **and** $A_F \sharp* (\Psi \otimes \Psi')$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*
thus *?thesis*
by *auto* (*metis frameIntComposition Commutativity frameIntAssociativity FrameStatEqTrans FrameStatEqSym*)
qed

lemma *guardedStatEq*:
fixes P :: ($'a$, $'b$, $'c$) *psi*
and I :: ($'a$, $'b$, $'c$) *input*
and C :: ($'a$, $'b$, $'c$) *psiCase*
and A_P :: *name list*
and Ψ_P :: $'b$

shows $[\![$*guarded* P; *extractFrame* $P = \langle A_P, \Psi_P\rangle]\!] \Longrightarrow \Psi_P \simeq \bot \wedge$ *supp* $\Psi_P = (\{\}$::*name set*)
and $[\![$*guarded'* I; *extractFrame'* $I = \langle A_P, \Psi_P\rangle]\!] \Longrightarrow \Psi_P \simeq \bot \wedge$ *supp* $\Psi_P = (\{\}$::*name set*)
and $[\![$*guarded''* C; *extractFrame''* $C = \langle A_P, \Psi_P\rangle]\!] \Longrightarrow \Psi_P \simeq \bot \wedge$ *supp* $\Psi_P = (\{\}$::*name set*)
proof(*nominal-induct* P **and** I **and** C *arbitrary*: A_P Ψ_P *rule*: *psi-input-psiCase.strong-inducts*)

case(*PsiNil* A_P Ψ_P)
thus *?case* **by** *simp*
next
case(*Output M N P* A_P Ψ_P)
thus *?case* **by** *simp*
next
case(*Input M In* A_P Ψ_P)
thus *?case* **by** *simp*
next
case(*Case psiCase* A_P Ψ_P)
thus *?case* **by** *simp*
next
case(*Par P Q* A_{PQ} Ψ_{PQ})
from ‹*guarded*($P \parallel Q$)› **have** *guarded P* **and** *guarded Q* **by** *simp+*
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* Q$ **by**(*rule freshFrame*)
obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* A_P$ **and** $A_Q \sharp* \Psi_P$
by(*rule-tac* C=(A_P, Ψ_P) **in** *freshFrame*) *auto*

from ‹$\bigwedge A_P\ \Psi_P.\ [\![$*guarded P*; *extractFrame* $P = \langle A_P, \Psi_P\rangle]\!] \Longrightarrow \Psi_P \simeq \bot \wedge$ (*supp* $\Psi_P = (\{\}$::*name set*))› ‹*guarded P*› *FrP*
have $\Psi_P \simeq \bot$ **and** *supp* $\Psi_P = (\{\}$::*name set*) **by** *simp+*
from ‹$\bigwedge A_Q\ \Psi_Q.\ [\![$*guarded Q*; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle]\!] \Longrightarrow \Psi_Q \simeq \bot \wedge$ (*supp* $\Psi_Q = (\{\}$::*name set*))› ‹*guarded Q*› *FrQ*
have $\Psi_Q \simeq \bot$ **and** *supp* $\Psi_Q = (\{\}$::*name set*) **by** *simp+*

from ‹$A_P \sharp* Q$› *FrQ* ‹$A_Q \sharp* A_P$› **have** $A_P \sharp* \Psi_Q$ **by**(*drule-tac extractFrameFreshChain*) *auto*
with ‹$A_Q \sharp* A_P$› ‹$A_Q \sharp* \Psi_P$› *FrP FrQ* ‹*extractFrame*($P \parallel Q$) $= \langle A_{PQ}, \Psi_{PQ}\rangle$› **have** $\langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$
by *auto*
with ‹*supp* $\Psi_P = \{\}$› ‹*supp* $\Psi_Q = \{\}$› *compSupp* **have** $\Psi_{PQ} = \Psi_P \otimes \Psi_Q$
by *blast*
moreover from ‹$\Psi_P \simeq \bot$› ‹$\Psi_Q \simeq \bot$› **have** $\Psi_P \otimes \Psi_Q \simeq \bot$
by(*metis Composition Identity Associativity Commutativity AssertionStatEqTrans*)
ultimately show *?case* **using** ‹*supp* $\Psi_P = \{\}$› ‹*supp* $\Psi_Q = \{\}$› *compSupp*
by *blast*
next
case(*Res x P* A_{xP} Ψ_{xP})
from ‹*guarded*($(\!|\nu x|\!)P$)› **have** *guarded P* **by** *simp*
moreover obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by**(*rule freshFrame*)
moreover note ‹$\bigwedge A_P\ \Psi_P.\ [\![$*guarded P*; *extractFrame* $P = \langle A_P, \Psi_P\rangle]\!] \Longrightarrow \Psi_P \simeq \bot \wedge$ (*supp* $\Psi_P = (\{\}$::*name set*))›
ultimately have $\Psi_P \simeq \bot$ **and** *supp* $\Psi_P = (\{\}$::*name set*) **by** *auto*
from *FrP* ‹*extractFrame*($(\!|\nu x|\!)P$) $= \langle A_{xP}, \Psi_{xP}\rangle$› **have** $\langle (x\#A_P), \Psi_P\rangle = \langle A_{xP}, \Psi_{xP}\rangle$ **by** *simp*

with ‹*supp* $\Psi_P = \{\}$› **have** $\Psi_P = \Psi_{xP}$ **by**(*auto simp del*: *frameResChain.simps*)
with ‹$\Psi_P \simeq \bot$› ‹*supp* $\Psi_P = \{\}$› **show** *?case*
by *simp*
next
case(*Assert* Ψ A_P Ψ_P)
thus *?case* **by** *simp*
next
case(*Bang P* A_P Ψ_P)
thus *?case* **by** *simp*
next
case(*Trm M P*)
thus *?case* **by** *simp*
next
case(*Bind x I*)
thus *?case* **by** *simp*
next
case *EmptyCase*
thus *?case* **by** *simp*
next
case(*Cond* φ *P psiCase*)
thus *?case* **by** *simp*
qed

end

end

theory *Semantics*
imports *Frame*
begin

nominal-datatype (*'a*, *'b*, *'c*) *boundOutput* =
BOut 'a::*fs-name* (*'a*, *'b*::*fs-name*, *'c*::*fs-name*) *psi* (‹- ≺'' -› [*110*, *110*] *110*)
| *BStep* «*name*» (*'a*, *'b*, *'c*) *boundOutput* (‹⦇ν-⦈-› [*110*, *110*] *110*)

primrec *BOresChain* :: *name list* ⇒ (*'a*::*fs-name*, *'b*::*fs-name*, *'c*::*fs-name*) *boundOutput* ⇒
(*'a*, *'b*, *'c*) *boundOutput* **where**
Base: *BOresChain* [] *B* = *B*
| *Step*: *BOresChain* (*x*#*xs*) *B* = ⦇νx⦈(*BOresChain xs B*)

abbreviation
BOresChainJudge (‹⦇ν*-⦈-› [*80*, *80*] *80*) **where** ⦇ν*xvec*⦈*B* ≡ *BOresChain xvec B*

lemma *BOresChainEqvt*[*eqvt*]:
fixes *perm* :: *name prm*
and *lst* :: *name list*
and *B* :: (*'a*::*fs-name*, *'b*::*fs-name*, *'c*::*fs-name*) *boundOutput*

shows $perm \cdot ((\!|\nu{*}xvec|\!)B) = (\!|\nu{*}(perm \cdot xvec)|\!)(perm \cdot B)$
by(*induct-tac xvec*, *auto*)

lemma *BOresChainSimps*[*simp*]:
fixes $xvec$ :: *name list*
and N :: $'a$::*fs-name*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*
and N' :: $'a$
and P' :: ($'a$, $'b$, $'c$) *psi*
and B :: ($'a$, $'b$, $'c$) *boundOutput*
and B' :: ($'a$, $'b$, $'c$) *boundOutput*

shows $((\!|\nu{*}xvec|\!)N \prec' P = N' \prec' P') = (xvec = [] \wedge N = N' \wedge P = P')$
and $(N' \prec' P' = (\!|\nu{*}xvec|\!)N \prec' P) = (xvec = [] \wedge N = N' \wedge P = P')$
and $(N' \prec' P' = N \prec' P) = (N = N' \wedge P = P')$
and $((\!|\nu{*}xvec|\!)B = (\!|\nu{*}xvec|\!)B') = (B = B')$
by(*induct xvec*) (*auto simp add*: *boundOutput.inject alpha*)

lemma *outputFresh*[*simp*]:
fixes Xs :: *name set*
and $xvec$ :: *name list*
and N :: $'a$::*fs-name*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $(Xs \sharp* (N \prec' P)) = ((Xs \sharp* N) \wedge (Xs \sharp* P))$
and $(xvec \sharp* (N \prec' P)) = ((xvec \sharp* N) \wedge (xvec \sharp* P))$
by(*auto simp add*: *fresh-star-def*)

lemma *boundOutputFresh*:
fixes x :: *name*
and $xvec$ :: *name list*
and B :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *boundOutput*

shows $(x \sharp ((\!|\nu{*}xvec|\!)B)) = (x \in set\ xvec \vee x \sharp B)$
by (*induct xvec*) (*simp-all add*: *abs-fresh*)

lemma *boundOutputFreshSet*:
fixes Xs :: *name set*
and $xvec$ :: *name list*
and B :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *boundOutput*
and $yvec$ :: *name list*
and x :: *name*

shows $Xs \sharp* ((\!|\nu{*}xvec|\!)B) = (\forall x{\in}Xs.\ x \in set\ xvec \vee x \sharp B)$
and $yvec \sharp* ((\!|\nu{*}xvec|\!)B) = (\forall x{\in}(set\ yvec).\ x \in set\ xvec \vee x \sharp B)$
and $Xs \sharp* ((\!|\nu x|\!)B) = Xs \sharp* [x].B$
and $xvec \sharp* ((\!|\nu x|\!)B) = xvec \sharp* [x].B$
by(*simp add*: *fresh-star-def boundOutputFresh*)+

lemma *BOresChainSupp*:
  fixes *xvec* :: *name list*
  and *B* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*

  shows (*supp*(⦇ν∗*xvec*⦈*B*)::*name set*) = (*supp B*) − (*supp xvec*)
by(*induct xvec*)
  (*auto simp add*: *boundOutput.supp supp-list-nil supp-list-cons abs-supp supp-atm*)

lemma *boundOutputFreshSimps*[*simp*]:
  fixes *Xs* :: *name set*
  and *xvec* :: *name list*
  and *B* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*
  and *yvec* :: *name list*
  and *x* :: *name*

  shows *Xs* ♯∗ *xvec* ⟹ (*Xs* ♯∗ (⦇ν∗*xvec*⦈*B*)) = (*Xs* ♯∗ *B*)
  and *yvec* ♯∗ *xvec* ⟹ *yvec* ♯∗ (⦇ν∗*xvec*⦈*B*) = *yvec* ♯∗ *B*
  and *xvec* ♯∗ (⦇ν∗*xvec*⦈*B*)
  and *x* ♯ *xvec* ⟹ *x* ♯ ⦇ν∗*xvec*⦈*B* = *x* ♯ *B*
apply(*simp add*: *boundOutputFreshSet*) **apply**(*force simp add*: *fresh-star-def name-list-supp fresh-def*)
apply(*simp add*: *boundOutputFreshSet*) **apply**(*force simp add*: *fresh-star-def name-list-supp fresh-def*)
apply(*simp add*: *boundOutputFreshSet*)
by(*simp add*: *BOresChainSupp fresh-def*)

lemma *boundOutputChainAlpha*:
  fixes *p* :: *name prm*
  and *xvec* :: *name list*
  and *B* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*
  and *yvec* :: *name list*

  assumes *xvecFreshB*: (*p* · *xvec*) ♯∗ *B*
  and *S*: *set p* ⊆ *set xvec* × *set* (*p* · *xvec*)
  and (*set xvec*) ⊆ (*set yvec*)

  shows (⦇ν∗*yvec*⦈*B*) = (⦇ν∗(*p* · *yvec*)⦈(*p* · *B*))
proof −
  note *pt-name-inst at-name-inst S*
  moreover from ‹(*set xvec*) ⊆ (*set yvec*)› **have** *set xvec* ♯∗ (⦇ν∗*yvec*⦈*B*)
    by(*force simp add*: *boundOutputFreshSet*)
  moreover from *xvecFreshB* ‹(*set xvec*) ⊆ (*set yvec*)› **have** *set* (*p* · *xvec*) ♯∗ (⦇ν∗*yvec*⦈*B*)
    by (*simp add*: *boundOutputFreshSet*) (*simp add*: *fresh-star-def*)
  ultimately have (⦇ν∗*yvec*⦈*B*) = *p* · (⦇ν∗*yvec*⦈*B*)
    by (*rule-tac pt-freshs-freshs* [*symmetric*])
  then show *?thesis* **by**(*simp add*: *eqvts*)
qed

lemma *boundOutputChainAlpha'*:
  fixes p :: *name prm*
  and *xvec* :: *name list*
  and B :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*
  and *yvec* :: *name list*
  and *zvec* :: *name list*

  assumes *xvecFreshB*: *xvec* $\sharp*$ B
  and *S*: *set* $p \subseteq$ *set xvec* $\times$ *set yvec*
  and *yvec* $\sharp*$ $((\!|\nu*zvec|\!)B)$

  shows $((\!|\nu*zvec|\!)B) = ((\!|\nu*(p \cdot zvec)|\!)(p \cdot B))$
proof −
  note *pt-name-inst at-name-inst S* ‹*yvec* $\sharp*$ $((\!|\nu*zvec|\!)B)$›
  moreover from *xvecFreshB* **have** *set* (*xvec*) $\sharp*$ $((\!|\nu*zvec|\!)B)$
    by (*simp add*: *boundOutputFreshSet*) (*simp add*: *fresh-star-def*)
  ultimately have $((\!|\nu*zvec|\!)B) = p \cdot ((\!|\nu*zvec|\!)B)$
    by (*rule-tac pt-freshs-freshs* [*symmetric*]) *auto*
  then show *?thesis* **by**(*simp add*: *eqvts*)
qed

lemma *boundOutputChainAlpha''*:
  fixes p :: *name prm*
  and *xvec* :: *name list*
  and M :: '*a*::*fs-name*
  and P :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
  and *yvec* :: *name list*

  assumes $(p \cdot xvec) \sharp* M$
  and $(p \cdot xvec) \sharp* P$
  and *set* $p \subseteq$ *set xvec* $\times$ *set* $(p \cdot xvec)$
  and (*set xvec*) $\subseteq$ (*set yvec*)

  shows $((\!|\nu*yvec|\!)M \prec' P) = ((\!|\nu*(p \cdot yvec)|\!)(p \cdot M) \prec' (p \cdot P))$
using *assms*
by(*subst boundOutputChainAlpha*) *auto*

lemma *boundOutputChainSwap*:
  fixes x :: *name*
  and y :: *name*
  and N :: '*a*::*fs-name*
  and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
  and *xvec* :: *name list*

  assumes $y \sharp N$
  and $y \sharp P$
  and $x \in$ (*set xvec*)

shows $(\!|\nu{*}xvec|\!)N \prec' P = (\!|\nu{*}([(x,\ y)] \cdot xvec)|\!)([(x\ ,y)] \cdot N) \prec' ([(x,\ y)] \cdot P)$
proof(*case-tac* $x{=}y$)
assume $x{=}y$
thus *?thesis* **by** *simp*
next
assume $x \neq y$
with *assms* **show** *?thesis*
by(*rule-tac xvec*=[x] **in** *boundOutputChainAlpha''*) (*auto simp add*: *calc-atm*)
qed

lemma *alphaBoundOutput*:
fixes x :: *name*
and y :: *name*
and B :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*

assumes $y \sharp B$

shows $(\!|\nu x|\!)B = (\!|\nu y|\!)([(x,\ y)] \cdot B)$
using *assms*
by(*auto simp add*: *boundOutput.inject alpha fresh-left calc-atm*)

lemma *boundOutputEqFresh*:
fixes B :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*
and C :: ('*a*, '*b*, '*c*) *boundOutput*
and x :: *name*
and y :: *name*

assumes $(\!|\nu x|\!)B = (\!|\nu y|\!)C$
and $x \sharp B$

shows $y \sharp C$
using *assms*
by(*auto simp add*: *boundOutput.inject alpha fresh-left calc-atm*)

lemma *boundOutputEqSupp*:
fixes B :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*
and C :: ('*a*, '*b*, '*c*) *boundOutput*
and x :: *name*
and y :: *name*

assumes $(\!|\nu x|\!)B = (\!|\nu y|\!)C$
and $x \in supp\ B$

shows $y \in supp\ C$
using *assms*
apply(*auto simp add*: *boundOutput.inject alpha fresh-left calc-atm*)
apply(*drule-tac pi*=[(x, y)] **in** *pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*])
by(*simp add*: *eqvts calc-atm*)

lemma *boundOutputChainEq*:
  fixes *xvec* :: *name list*
  and *B* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *boundOutput*
  and *yvec* :: *name list*
  and *B'* :: ('*a*, '*b*, '*c*) *boundOutput*

  assumes $(\!|\nu*xvec|\!)B = (\!|\nu*yvec|\!)B'$
  and *xvec* $\sharp*$ *yvec*
  and *length xvec* = *length yvec*

  shows $\exists p.$ (*set p*) $\subseteq$ (*set xvec*) $\times$ *set* (*yvec*) $\wedge$ *distinctPerm p* $\wedge$ $B = p \cdot B' \wedge$ (*set* (*map fst p*)) $\subseteq$ (*supp B*) $\wedge$ *xvec* $\sharp*$ *B'* $\wedge$ *yvec* $\sharp*$ *B*
proof −
  obtain *n* **where** *n* = *length xvec* **by** *auto*
  with *assms* **show** *?thesis*
  proof(*induct n arbitrary*: *xvec yvec B B'*)
    case(*0 xvec yvec B B'*)
    have *Eq*: $(\!|\nu*xvec|\!)B = (\!|\nu*yvec|\!)B'$ **by** *fact*
    from ‹*0* = *length xvec*› **have** *xvec* = [] **by** *auto*
    moreover with ‹*length xvec* = *length yvec*› **have** *yvec* = []
      by(*case-tac yvec*) *auto*
    ultimately show *?case* **using** *Eq*
      by(*simp add*: *boundOutput.inject*)
  next
    case(*Suc n xvec yvec B B'*)
    from ‹*Suc n* = *length xvec*›
    obtain *x xvec'* **where** *xvec* = *x*#*xvec'* **and** *length xvec'* = *n*
      by(*case-tac xvec*) *auto*
    from ‹$(\!|\nu*xvec|\!)B = (\!|\nu*yvec|\!)B'$› ‹*xvec* = *x* # *xvec'*› ‹*length xvec* = *length yvec*›
    obtain *y yvec'* **where** $(\!|\nu*(x\#xvec')|\!)B = (\!|\nu*(y\#yvec')|\!)B'$
      and *yvec* = *y*#*yvec'* **and** *length xvec'* = *length yvec'*
      by(*case-tac yvec*) *auto*
    hence *EQ*: $(\!|\nu x|\!)((\!|\nu*xvec'|\!)B) = (\!|\nu y|\!)((\!|\nu*yvec'|\!)B')$
      by *simp*
    from ‹*xvec* = *x*#*xvec'*› ‹*yvec*=*y*#*yvec'*› ‹*xvec* $\sharp*$ *yvec*›
    have $x \neq y$ **and** *xvec'* $\sharp*$ *yvec'* **and** *x* $\sharp$ *yvec'* **and** *y* $\sharp$ *xvec'*
      by *auto*
    have *IH*: $\bigwedge$*xvec yvec B B'*. $[\![(\!|\nu*xvec|\!)(B::('a::fs\text{-}name, 'b::fs\text{-}name, 'c::fs\text{-}name)\ boundOutput) = (\!|\nu*yvec|\!)B'; xvec \sharp* yvec; length\ xvec = length\ yvec; n = length\ xvec]\!] \Longrightarrow \exists p.\ (set\ p) \subseteq (set\ xvec) \times (set\ yvec) \wedge distinctPerm\ p \wedge B = p \cdot B' \wedge set(map\ fst\ p) \subseteq supp\ B \wedge xvec \sharp* B' \wedge yvec \sharp* B$
      by *fact*
    from *EQ* ‹$x \neq y$› **have** *EQ'*: $(\!|\nu*xvec'|\!)B = ([(x, y)] \cdot ((\!|\nu*yvec'|\!)B'))$
                and *xFreshB'*: $x \sharp ((\!|\nu*yvec'|\!)B')$
                and *yFreshB*: $y \sharp ((\!|\nu*xvec'|\!)B)$
      by(*metis boundOutput.inject alpha*)+
    from *xFreshB'* ‹*x* $\sharp$ *yvec'*› **have** *x* $\sharp$ *B'*
      by(*auto simp add*: *boundOutputFresh*) (*simp add*: *fresh-def name-list-supp*)+
    from *yFreshB* ‹*y* $\sharp$ *xvec'*› **have** *y* $\sharp$ *B*

by(*auto simp add*: *boundOutputFresh*) (*simp add*: *fresh-def name-list-supp*)+
show *?case*
proof(*case-tac* $x \sharp (\!|\nu{*}xvec'|\!)B$)
assume *xFreshB*: $x \sharp (\!|\nu{*}xvec'|\!)B$
with *EQ* **have** *yFreshB'*: $y \sharp (\!|\nu{*}yvec'|\!)B'$
by(*rule boundOutputEqFresh*)
with *xFreshB' EQ'* **have** $(\!|\nu{*}xvec'|\!)B = (\!|\nu{*}yvec'|\!)B'$
by(*simp*)
with ‹$xvec' \sharp{*}\ yvec'$› ‹$length\ xvec' = length\ yvec'$› ‹$length\ xvec' = n$› *IH*
obtain *p* **where** *S*: $(set\ p) \subseteq (set\ xvec') \times (set\ yvec')$ **and** *distinctPerm p* **and** $B = p \cdot B'$
and $set(map\ fst\ p) \subseteq supp\ B$ **and** $xvec' \sharp{*}\ B'$ **and** $yvec' \sharp{*}\ B$
by *blast*
from *S* **have** $(set\ p) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
moreover note ‹$xvec = x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹*distinctPerm p*› ‹$B = p \cdot B'$›
‹$xvec' \sharp{*}\ B'$› ‹$x \sharp B'$› ‹$x \sharp B'$› ‹$yvec' \sharp{*}\ B$› ‹$y \sharp B$› ‹$set(map\ fst\ p) \subseteq supp\ B$›

ultimately show *?case* **by** *auto*
next
assume $\neg(x \sharp (\!|\nu{*}xvec'|\!)B)$
hence *xSuppB*: $x \in supp((\!|\nu{*}xvec'|\!)B)$
by(*simp add*: *fresh-def*)
with *EQ* **have** *ySuppB'*: $y \in supp\ ((\!|\nu{*}yvec'|\!)B')$
by(*rule boundOutputEqSupp*)
hence $y \sharp yvec'$
by(*induct yvec'*) (*auto simp add*: *boundOutput.supp abs-supp*)
with ‹$x \sharp yvec'$› *EQ'* **have** $(\!|\nu{*}xvec'|\!)B = (\!|\nu{*}yvec'|\!)([(x,\ y)] \cdot B')$
by(*simp add*: *eqvts*)
with ‹$xvec' \sharp{*}\ yvec'$› ‹$length\ xvec' = length\ yvec'$› ‹$length\ xvec' = n$› *IH*
obtain *p* **where** *S*: $(set\ p) \subseteq (set\ xvec') \times (set\ yvec')$ **and** *distinctPerm p* **and** $B = p \cdot [(x,\ y)] \cdot B'$
and $set(map\ fst\ p) \subseteq supp\ B$ **and** $xvec' \sharp{*}\ ([(x,\ y)] \cdot B')$ **and** $yvec' \sharp{*}\ B$
by *blast*

from *xSuppB* **have** $x \sharp xvec'$
by(*induct xvec'*) (*auto simp add*: *boundOutput.supp abs-supp*)
with ‹$x \sharp yvec'$› ‹$y \sharp xvec'$› ‹$y \sharp yvec'$› *S* **have** $x \sharp p$ **and** $y \sharp p$
apply(*induct p*)
by(*auto simp add*: *name-list-supp*) (*auto simp add*: *fresh-def*)
from *S* **have** $(set\ ((x,\ y)\#p)) \subseteq (set(x\#xvec')) \times (set(y\#yvec'))$
by *force*
moreover from ‹$x \neq y$› ‹$x \sharp p$› ‹$y \sharp p$› *S* ‹*distinctPerm p*›
have $distinctPerm((x,y)\#p)$ **by** *simp*
moreover from ‹$B = p \cdot [(x,\ y)] \cdot B'$› ‹$x \sharp p$› ‹$y \sharp p$› **have** $B = [(x,\ y)] \cdot p \cdot B'$
by(*subst perm-compose*) *simp*

hence $B = ((x, y)\#p) \cdot B'$ **by** *simp*
moreover from ‹$xvec' \sharp* ([(x, y)] \cdot B')$› **have** $([(x, y)] \cdot xvec') \sharp* ([(x, y)] \cdot [(x, y)] \cdot B')$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp xvec'$› ‹$y \sharp xvec'$› ‹$x \sharp B'$› **have** $(x\#xvec') \sharp* B'$ **by** *simp*
moreover from ‹$y \sharp B$› ‹$yvec' \sharp* B$› **have** $(y\#yvec') \sharp* B$ **by** *simp*
moreover from ‹$set(map\ fst\ p) \subseteq supp\ B$› *xSuppB* ‹$x \sharp xvec'$›
have $set(map\ fst\ ((x, y)\#p)) \subseteq supp\ B$
by(*simp add*: *BOresChainSupp*)
ultimately show *?case* **using** ‹$xvec=x\#xvec'$› ‹$yvec=y\#yvec'$›
by *metis*
qed
qed
qed

lemma *boundOutputChainEqLength*:
fixes $xvec$ :: *name list*
and M :: $'a$::*fs-name*
and P :: $('a, 'b$::*fs-name*, $'c$::*fs-name*) *psi*
and $yvec$ :: *name list*
and N :: $'a$::*fs-name*
and Q :: $('a, 'b$::*fs-name*, $'c$::*fs-name*) *psi*

assumes $(\!|\nu*xvec|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' Q$

shows $length\ xvec = length\ yvec$
proof −
obtain n **where** $n = length\ xvec$ **by** *auto*
with *assms* **show** *?thesis*
proof(*induct n arbitrary*: *xvec yvec M P N Q*)
case(*0 xvec yvec M P N Q*)
from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
moreover with ‹$(\!|\nu*xvec|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' Q$› **have** $yvec = []$
by(*case-tac yvec*) *auto*
ultimately show *?case* **by** *simp*
next
case(*Suc n xvec yvec M P N Q*)
from ‹$Suc\ n = length\ xvec$›
obtain $x\ xvec'$ **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
by(*case-tac xvec*) *auto*
from ‹$(\!|\nu*xvec|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' Q$› ‹$xvec = x\ \#\ xvec'$›
obtain $y\ yvec'$ **where** $(\!|\nu*(x\#xvec')|\!)M \prec' P = (\!|\nu*(y\#yvec')|\!)N \prec' Q$
and $yvec = y\#yvec'$
by(*case-tac yvec*) *auto*
hence *EQ*: $(\!|\nu x|\!)((\!|\nu*xvec'|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu*yvec'|\!)N \prec' Q)$
by *simp*
have *IH*: $\bigwedge xvec\ yvec\ M\ P\ N\ Q.\ [\![(\!|\nu*xvec|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' (Q::('a, 'b, 'c)\ psi);\ n = length\ xvec]\!] \Longrightarrow length\ xvec = length\ yvec$
by *fact*

```
    show ?case
    proof(case-tac x = y)
      assume x = y
      with EQ have ⦇ν*xvec'⦈M ≺' P  = ⦇ν*yvec'⦈N ≺' Q
        by(simp add: alpha boundOutput.inject)
      with IH ‹length xvec' = n› have length xvec' = length yvec'
        by blast
      with ‹xvec = x#xvec'› ‹yvec=y#yvec'›
      show ?case by simp
    next
      assume x ≠ y
      with EQ have ⦇ν*xvec'⦈M ≺' P = [(x, y)] • ⦇ν*yvec'⦈N ≺' Q
        by(simp add: alpha boundOutput.inject)
     hence ⦇ν*xvec'⦈M ≺' P = ⦇ν*([(x, y)] • yvec')⦈([(x, y)] • N) ≺' ([(x, y)] • Q)
        by(simp add: eqvts)
      with IH ‹length xvec' = n› have length xvec' = length ([(x, y)] • yvec')
        by blast
      hence length xvec' = length yvec'
        by simp
      with ‹xvec = x#xvec'› ‹yvec=y#yvec'›
      show ?case by simp
    qed
  qed
qed

lemma boundOutputChainEq':
  fixes xvec :: name list
  and   M    :: 'a::fs-name
  and   P    :: ('a, 'b::fs-name, 'c::fs-name) psi
  and   yvec :: name list
  and   N    :: 'a
  and   Q    :: ('a::fs-name, 'b::fs-name, 'c::fs-name) psi

  assumes ⦇ν*xvec⦈M ≺' P = ⦇ν*yvec⦈N ≺' Q
  and     xvec ♯* yvec

  shows ∃p. (set p) ⊆ (set xvec) × set (yvec) ∧ distinctPerm p ∧  M = p • N ∧
P = p • Q ∧ xvec ♯* N ∧ xvec ♯* Q ∧ yvec ♯* M ∧ yvec ♯* P
using assms
apply(frule-tac boundOutputChainEqLength)
apply(drule-tac boundOutputChainEq)
by(auto simp add: boundOutput.inject)

lemma boundOutputChainEq'':
  fixes xvec :: name list
  and   M    :: 'a::fs-name
  and   P    :: ('a, 'b::fs-name, 'c::fs-name) psi
  and   yvec :: name list
  and   N    :: 'a
```

and Q :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

assumes $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' Q$
and $xvec \sharp* yvec$
and *distinct xvec*
and *distinct yvec*

obtains p **where** $(set\ p) \subseteq (set\ xvec) \times set\ (p \cdot xvec)$ **and** *distinctPerm* p **and** $yvec = p \cdot xvec$ **and** $N = p \cdot M$ **and** $Q = p \cdot P$ **and** $xvec \sharp* N$ **and** $xvec \sharp* Q$ **and** $(p \cdot xvec) \sharp* M$ **and** $(p \cdot xvec) \sharp* P$
proof −

assume $\bigwedge p.\ [\![set\ p \subseteq set\ xvec \times set\ (p \cdot xvec);\ distinctPerm\ p;\ yvec = p \cdot xvec;\ N = p \cdot M;\ Q = p \cdot P;\ xvec \sharp* N;\ xvec \sharp* Q;\ (p \cdot xvec) \sharp* M;\ (p \cdot xvec) \sharp* P]\!] \Longrightarrow thesis$

moreover obtain n **where** $n = length\ xvec$ **by** *auto*
with *assms* **have** $\exists p.\ (set\ p) \subseteq (set\ xvec) \times set\ (yvec) \wedge distinctPerm\ p \wedge\ yvec = p \cdot xvec \wedge N = p \cdot M \wedge Q = p \cdot P \wedge xvec \sharp* N \wedge xvec \sharp* Q \wedge (p \cdot xvec) \sharp* M \wedge (p \cdot xvec) \sharp* P$
proof(*induct n arbitrary*: *xvec yvec M P N Q*)
case(*0 xvec yvec M P N Q*)
have *Eq*: $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' Q$ **by** *fact*
from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
moreover with *Eq* **have** $yvec = []$
by(*case-tac yvec*) *auto*
ultimately show *?case* **using** *Eq*
by(*simp add*: *boundOutput.inject*)
next
case(*Suc n xvec yvec M P N Q*)
from ‹$Suc\ n = length\ xvec$›
obtain $x\ xvec'$ **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
by(*case-tac xvec*) *auto*
from ‹$(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' Q$› ‹$xvec = x\ \#\ xvec'$›
obtain $y\ yvec'$ **where** $(\!|\nu{*}(x\#xvec')|\!)M \prec' P = (\!|\nu{*}(y\#yvec')|\!)N \prec' Q$
and $yvec = y\#yvec'$
by(*case-tac yvec*) *auto*
hence *EQ*: $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' Q)$
by *simp*
from ‹$xvec = x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹$xvec \sharp* yvec$›
have $x \neq y$ **and** $xvec' \sharp* yvec'$ **and** $x \sharp yvec'$ **and** $y \sharp xvec'$
by *auto*
from ‹*distinct xvec*› ‹*distinct yvec*› ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$› **have** $x \sharp xvec'$ **and** $y \sharp yvec'$ **and** *distinct* $xvec'$ **and** *distinct* $yvec'$
by *simp+*
have *IH*: $\bigwedge xvec\ yvec\ M\ P\ N\ Q.\ [\![(\!|\nu{*}xvec|\!)(M::'a) \prec' (P::('a,\ 'b,\ 'c)\ psi) = (\!|\nu{*}yvec|\!)N \prec' Q;\ xvec \sharp* yvec;\ distinct\ xvec;\ distinct\ yvec;\ n = length\ xvec]\!] \Longrightarrow \exists p.\ (set\ p) \subseteq (set\ xvec) \times (set\ yvec) \wedge distinctPerm\ p \wedge\ yvec = p \cdot xvec \wedge N = p \cdot M \wedge Q = p \cdot P \wedge xvec \sharp* N \wedge xvec \sharp* Q \wedge (p \cdot xvec) \sharp* M \wedge (p \cdot xvec) \sharp* P$

by *fact*
from EQ ‹$x \neq y$› ‹$x \sharp yvec'$› ‹$y \sharp yvec'$› ‹$y \sharp xvec'$› ‹$x \sharp xvec'$› **have** $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$ **and** $x \sharp N$ **and** $x \sharp Q$ **and** $y \sharp M$ **and** $y \sharp P$
apply −
apply(*simp add*: *boundOutput.inject alpha eqvts*)
apply(*simp add*: *boundOutput.inject alpha eqvts*)
apply(*simp add*: *boundOutput.inject alpha eqvts*)
by(*simp add*: *boundOutput.inject alpha′ eqvts*)+
with ‹$xvec' \sharp{*}\ yvec'$› ‹*distinct xvec′*› ‹*distinct yvec′*› ‹*length xvec′* = *n*› *IH*
obtain p **where** S: $(set\ p) \subseteq (set\ xvec') \times (set\ yvec')$ **and** *distinctPerm p* **and** $yvec' = p \cdot xvec'$ **and** $([(x, y)] \cdot N) = p \cdot M$ **and** $([(x, y)] \cdot Q) = p \cdot P$ **and** $xvec' \sharp{*}\ ([(x, y)] \cdot N)$ **and** $xvec' \sharp{*}\ ([(x, y)] \cdot Q)$ **and** $yvec' \sharp{*}\ M$ **and** $yvec' \sharp{*}\ P$
by *metis*
from S **have** $set((x, y)\#p) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
moreover from ‹$x \sharp xvec'$› ‹$x \sharp yvec'$› ‹$y \sharp xvec'$› ‹$y \sharp yvec'$› S **have** $x \sharp p$ **and** $y \sharp p$
apply(*induct p*)
by(*auto simp add*: *fresh-prod name-list-supp*) (*auto simp add*: *fresh-def*)

with S ‹*distinctPerm p*› ‹$x \neq y$› **have** $distinctPerm((x, y)\#p)$ **by** *auto*
moreover from ‹$yvec' = p \cdot xvec'$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp xvec'$› ‹$y \sharp xvec'$› **have** $(y\#yvec') = ((x, y)\#p) \cdot (x\#xvec')$
by(*simp add*: *eqvts calc-atm perm-compose freshChainSimps*)
moreover from ‹$([(x, y)] \cdot N) = p \cdot M$›
have $([(x, y)] \cdot [(x, y)] \cdot N) = [(x, y)] \cdot p \cdot M$
by(*simp add*: *pt-bij*)
hence $N = ((x, y)\#p) \cdot M$ **by** *simp*
moreover from ‹$([(x, y)] \cdot Q) = p \cdot P$›
have $([(x, y)] \cdot [(x, y)] \cdot Q) = [(x, y)] \cdot p \cdot P$
by(*simp add*: *pt-bij*)
hence $Q = ((x, y)\#p) \cdot P$ **by** *simp*
moreover from ‹$xvec' \sharp{*}\ ([(x, y)] \cdot N)$› **have** $([(x, y)] \cdot xvec') \sharp{*}\ ([(x, y)] \cdot [(x, y)] \cdot N)$
by(*subst fresh-star-bij*)
with ‹$x \sharp xvec'$› ‹$y \sharp xvec'$› **have** $xvec' \sharp{*}\ N$ **by** *simp*
with ‹$x \sharp N$› **have** $(x\#xvec') \sharp{*}\ N$ **by** *simp*
moreover from ‹$xvec' \sharp{*}\ ([(x, y)] \cdot Q)$› **have** $([(x, y)] \cdot xvec') \sharp{*}\ ([(x, y)] \cdot [(x, y)] \cdot Q)$
by(*subst fresh-star-bij*)
with ‹$x \sharp xvec'$› ‹$y \sharp xvec'$› **have** $xvec' \sharp{*}\ Q$ **by** *simp*
with ‹$x \sharp Q$› **have** $(x\#xvec') \sharp{*}\ Q$ **by** *simp*
moreover from ‹$y \sharp M$› ‹$yvec' \sharp{*}\ M$› **have** $(y\#yvec') \sharp{*}\ M$ **by** *simp*
moreover from ‹$y \sharp P$› ‹$yvec' \sharp{*}\ P$› **have** $(y\#yvec') \sharp{*}\ P$ **by** *simp*
ultimately show *?case* **using** ‹*xvec*=*x*#*xvec′*› ‹*yvec*=*y*#*yvec′*›
by *metis*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *boundOutputEqSupp'*:
fixes x :: *name*
and *xvec* :: *name list*
and M :: *'a::fs-name*
and P :: (*'a*, *'b::fs-name*, *'c::fs-name*) *psi*
and y :: *name*
and *yvec* :: *name list*
and N :: *'a*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes *Eq*: $(\!|\nu x|\!)((\!|\nu{*}xvec|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu{*}yvec|\!)N \prec' Q)$
and $x \neq y$
and $x \sharp yvec$
and $x \sharp xvec$
and $y \sharp xvec$
and $y \sharp yvec$
and $xvec \sharp{*}\ yvec$
and $x \in supp\ M$

shows $y \in supp\ N$
proof −
from *Eq* ‹$x \neq y$› ‹$x \sharp yvec$› ‹$y \sharp yvec$› **have** $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
by(*simp add*: *boundOutput.inject alpha eqvts*)
then obtain p **where** S: $set\ p \subseteq set\ xvec \times set\ yvec$ **and** $M = p \cdot [(x, y)] \cdot N$ **and** *distinctPerm p* **using** ‹$xvec \sharp{*}\ yvec$›
by(*blast dest*: *boundOutputChainEq'*)
with ‹$x \in supp\ M$› **have** $x \in supp(p \cdot [(x, y)] \cdot N)$ **by** *simp*
hence $(p \cdot x) \in p \cdot supp(p \cdot [(x, y)] \cdot N)$
by(*simp add*: *pt-set-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp xvec$› ‹$x \sharp yvec$› S ‹*distinctPerm p*› **have** $x \in supp([(x, y)] \cdot N)$
by(*simp add*: *eqvts*)
hence $([(x, y)] \cdot x) \in ([(x, y)] \cdot (supp([(x, y)] \cdot N)))$
by(*simp add*: *pt-set-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \neq y$› **show** *?thesis* **by**(*simp add*: *calc-atm eqvts*)
qed

lemma *boundOutputChainOpenIH*:
fixes *xvec* :: *name list*
and x :: *name*
and B :: (*'a::fs-name*, *'b::fs-name*, *'c::fs-name*) *boundOutput*
and *yvec* :: *name list*
and y :: *name*
and B' :: (*'a*, *'b*, *'c*) *boundOutput*

assumes *Eq*: $(\!|\nu{*}xvec|\!)((\!|\nu x|\!)B) = (\!|\nu{*}yvec|\!)((\!|\nu y|\!)B')$
and *L*: $length\ xvec = length\ yvec$
and *xFreshB'*: $x \sharp B'$

and *xFreshxvec*: $x \sharp xvec$
and *xFreshyvec*: $x \sharp yvec$

shows $(\!|\nu{*}xvec|\!)B = (\!|\nu{*}yvec|\!)([(x,\ y)] \cdot B')$
using *assms*
proof(*induct n==length xvec arbitrary*: *xvec yvec y B′ rule*: *nat.induct*)
case(*zero xvec yvec y B′*)
have *0* = *length xvec* **and** *length xvec* = *length yvec* **by** *fact+*
moreover have $(\!|\nu{*}xvec|\!)(\!|\nu x|\!)B = (\!|\nu{*}yvec|\!)(\!|\nu y|\!)B'$ **by** *fact*
ultimately show *?case* **by**(*auto simp add*: *boundOutput.inject alpha*)
next
case(*Suc n xvec yvec y B′*)
have *L*: *length xvec* = *length yvec* **and** *Suc n* = *length xvec* **by** *fact+*
then obtain *x′ xvec′ y′ yvec′* **where** *xEq*: *xvec* = *x′#xvec′* **and** *yEq*: *yvec* = *y′#yvec′*
and *L′*: *length xvec′* = *length yvec′*
by(*cases xvec*, *auto*, *cases yvec*, *auto*)
have *xFreshB′*: $x \sharp B'$ **by** *fact*
have $x \sharp xvec$ **and** $x \sharp yvec$ **by** *fact+*
with *xEq yEq* **have** *xineqx′*: $x \neq x'$ **and** *xFreshxvec′*: $x \sharp xvec'$
and *xineqy′*: $x \neq y'$ **and** *xFreshyvec′*: $x \sharp yvec'$
by *simp+*
have $(\!|\nu{*}xvec|\!)(\!|\nu x|\!)B = (\!|\nu{*}yvec|\!)(\!|\nu y|\!)B'$ **by** *fact*
with *xEq yEq* **have** *Eq*: $(\!|\nu x'|\!)((\!|\nu{*}xvec'|\!)(\!|\nu x|\!)B) = (\!|\nu y'|\!)((\!|\nu{*}yvec'|\!)(\!|\nu y|\!)B')$ **by** *simp*
have *Suc n* = *length xvec* **by** *fact*
with *xEq* **have** *L″*: *n* = *length xvec′* **by** *simp*
have $(\!|\nu x'|\!)((\!|\nu{*}xvec'|\!)B) = (\!|\nu y'|\!)((\!|\nu{*}yvec'|\!)([(x,\ y)] \cdot B'))$
proof(*case-tac x′=y′*)
assume *x′eqy′*: $x' = y'$
with *Eq* **have** $(\!|\nu{*}xvec'|\!)(\!|\nu x|\!)B = (\!|\nu{*}yvec'|\!)(\!|\nu y|\!)B'$ **by**(*simp add*: *boundOutput.inject alpha*)
hence $(\!|\nu{*}xvec'|\!)B = (\!|\nu{*}yvec'|\!)([(x,\ y)] \cdot B')$ **using** *L′ xFreshB′ xFreshxvec′ xFreshyvec′ L″*
by(*rule-tac Suc*)
with *x′eqy′* **show** *?thesis* **by**(*simp add*: *boundOutput.inject alpha*)
next
assume *x′ineqy′*: $x' \neq y'$
with *Eq* **have** *Eq′*: $(\!|\nu{*}xvec'|\!)(\!|\nu x|\!)B = (\!|\nu{*}([(x',\ y')] \cdot yvec')|\!)(\!|\nu([(x',\ y')] \cdot y)|\!)([(x',\ y')] \cdot B')$
and *x′FreshB′*: $x' \sharp (\!|\nu{*}yvec'|\!)(\!|\nu y|\!)B'$
by(*simp add*: *boundOutput.inject alpha eqvts*)+
from *L′* **have** *length xvec′* = *length* $([(x',\ y')] \cdot yvec')$ **by** *simp*
moreover from *xineqx′ xineqy′ xFreshB′* **have** $x \sharp [(x',\ y')] \cdot B'$ **by**(*simp add*: *fresh-left calc-atm*)
moreover from *xineqx′ xineqy′ xFreshyvec′* **have** $x \sharp [(x',\ y')] \cdot yvec'$ **by**(*simp add*: *fresh-left calc-atm*)
ultimately have $(\!|\nu{*}xvec'|\!)B = (\!|\nu{*}([(x',\ y')] \cdot yvec')|\!)([(x,\ ([(x',\ y')] \cdot y))] \cdot [(x',\ y')] \cdot B')$ **using** *Eq′ xFreshxvec′ L″*

```
      by(rule-tac Suc)
    moreover from x'FreshB' have x' ♯ ⦇ν*yvec'⦈([(x, y)] • B')
    proof(case-tac x' ♯ yvec')
      assume x' ♯ yvec'
      with x'FreshB' have x'FreshB': x' ♯ ⦇νy⦈B'
        by(simp add: fresh-def BOresChainSupp)
      show ?thesis
      proof(case-tac x'=y)
        assume x'eqy: x' = y
        show ?thesis
        proof(case-tac x=y)
          assume x=y
        with xFreshB' x'eqy show ?thesis by(simp add: BOresChainSupp fresh-def)
        next
          assume x ≠ y
          with ‹x ♯ B'› have y ♯ [(x, y)] • B' by(simp add: fresh-left calc-atm)
          with x'eqy show ?thesis by(simp add: BOresChainSupp fresh-def)
        qed
      next
        assume x'ineqy: x' ≠ y
        with x'FreshB' have x' ♯ B' by(simp add: abs-fresh)
       with xineqx' x'ineqy have x' ♯ ([(x, y)] • B') by(simp add: fresh-left calc-atm)
        thus ?thesis by(simp add: BOresChainSupp fresh-def)
      qed
    next
      assume ¬x' ♯ yvec'
      thus ?thesis by(simp add: BOresChainSupp fresh-def)
    qed
    ultimately show ?thesis using x'ineqy' xineqx' xineqy'
      apply(simp add: boundOutput.inject alpha eqvts)
      apply(subst perm-compose[of [(x', y')]])
      by(simp add: calc-atm)
  qed
  with xEq yEq show ?case by simp
qed

lemma boundOutputPar1Dest:
  fixes xvec :: name list
  and   M    :: 'a::fs-name
  and   P    :: ('a, 'b::fs-name, 'c::fs-name) psi
  and   yvec :: name list
  and   N    :: 'a
  and   Q    :: ('a, 'b, 'c) psi
  and   R    :: ('a, 'b, 'c) psi

  assumes ⦇ν*xvec⦈M ≺' P = ⦇ν*yvec⦈N ≺' (Q ∥ R)
  and     xvec ♯* R
  and     yvec ♯* R
```

obtains T **where** $P = T \parallel R$ **and** $(\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' Q$
proof −
assume $\bigwedge T.\ [\![P = T \parallel R;\ (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' Q]\!] \Longrightarrow thesis$
moreover obtain n **where** $n = length\ xvec$ **by** *auto*
with *assms* **have** $\exists\, T.\ P = T \parallel R \wedge (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' Q$
proof(*induct n arbitrary*: *xvec yvec M N P Q R*)
case(*0 xvec yvec M N P Q R*)
have *Eq*: $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$ **by** *fact*
from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
moreover with *Eq* **have** $yvec = []$
by(*case-tac yvec*) *auto*
ultimately show *?case* **using** *Eq*
by(*simp add*: *boundOutput.inject*)
next
case(*Suc n xvec yvec M N P Q R*)
from ‹$Suc\ n = length\ xvec$›
obtain $x\ xvec'$ **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
by(*case-tac xvec*) *auto*
from ‹$(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$› ‹$xvec = x\ \#\ xvec'$›
obtain $y\ yvec'$ **where** $(\!|\nu{*}(x\#xvec')|\!)M \prec' P = (\!|\nu{*}(y\#yvec')|\!)N \prec' (Q \parallel R)$
and $yvec = y\#yvec'$
by(*case-tac yvec*) *auto*
hence *EQ*: $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R))$
by *simp*
from ‹$xvec \sharp{*}\ R$› ‹$yvec \sharp{*}\ R$› ‹$xvec = x\#xvec'$› ‹$yvec = y\#yvec'$›
have $x \sharp R$ **and** $xvec' \sharp{*}\ R$ **and** $y \sharp R$ **and** $yvec' \sharp{*}\ R$ **by** *auto*
show *?case*
proof(*case-tac* $x = y$)
assume $x = y$
with *EQ* **have** $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R)$
by(*simp add*: *boundOutput.inject alpha*)
with ‹$xvec' \sharp{*}\ R$› ‹$yvec' \sharp{*}\ R$› ‹$length\ xvec' = n$›
obtain T **where** $P = T \parallel R$ **and** $(\!|\nu{*}xvec'|\!)M \prec' T = (\!|\nu{*}yvec'|\!)N \prec' Q$
by(*drule-tac Suc*) *auto*
with ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹$x{=}y$› **show** *?case*
by(*force simp add*: *boundOutput.inject alpha*)
next
assume $x \neq y$
with *EQ* ‹$x \sharp R$› ‹$y \sharp R$›
have $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}([(x,\ y)] \cdot yvec')|\!)([(x,\ y)] \cdot N) \prec' (([(x,\ y)] \cdot Q) \parallel R)$
and *xFreshQR*: $x \sharp (\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R)$
by(*simp add*: *boundOutput.inject alpha eqvts*)+
moreover from ‹$yvec' \sharp{*}\ R$› **have** $([(x,\ y)] \cdot yvec') \sharp{*}\ ([(x,\ y)] \cdot R)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp R$› ‹$y \sharp R$› **have** $([(x,\ y)] \cdot yvec') \sharp{*}\ R$ **by** *simp*
moreover note ‹$xvec' \sharp{*}\ R$› ‹$length\ xvec' = n$›
ultimately obtain T **where** $P = T \parallel R$ **and** A: $(\!|\nu{*}xvec'|\!)M \prec' T = (\!|\nu{*}([(x,\ y)] \cdot yvec')|\!)([(x,\ y)] \cdot N) \prec' ([(x,\ y)] \cdot Q)$

by(*drule-tac Suc*) *auto*

from *A* **have** $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' T) = (\!|\nu x|\!)((\!|\nu{*}([(x,\ y)] \cdot yvec')|\!)([(x,\ y)] \cdot N) \prec' ([(x,\ y)] \cdot Q))$
by(*simp add*: *boundOutput.inject alpha*)
moreover from *xFreshQR* **have** $x \sharp (\!|\nu{*}yvec'|\!)N \prec' Q$
by(*force simp add*: *boundOutputFresh*)
ultimately show *?thesis* **using** ‹$P = T \parallel R$› ‹*xvec=x#xvec'*› ‹*yvec=y#yvec'*› *xFreshQR*
by(*force simp add*: *alphaBoundOutput name-swap eqvts*)
qed
qed
ultimately show *?thesis*
by *blast*
qed

lemma *boundOutputPar1Dest'*:
fixes *xvec* :: *name list*
and *M* :: *'a::fs-name*
and *P* :: (*'a*, *'b::fs-name*, *'c::fs-name*) *psi*
and *yvec* :: *name list*
and *N* :: *'a*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *R* :: (*'a*, *'b*, *'c*) *psi*

assumes $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$
and $xvec \sharp{*}\ yvec$

obtains *T p* **where** $set\ p \subseteq set\ xvec \times set\ yvec$ **and** $P = T \parallel (p \cdot R)$ **and** $(\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' Q$
proof −
assume $\bigwedge p\ T.\ [\![set\ p \subseteq set\ xvec \times set\ yvec;\ P = T \parallel (p \cdot R);\ (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' Q]\!] \Longrightarrow thesis$
moreover obtain *n* **where** $n = length\ xvec$ **by** *auto*
with *assms* **have** $\exists p\ T.\ set\ p \subseteq set\ xvec \times set\ yvec \wedge P = T \parallel (p \cdot R) \wedge (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' Q$
proof(*induct n arbitrary*: *xvec yvec M N P Q R*)
case(*0 xvec yvec M N P Q R*)
have *Eq*: $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$ **by** *fact*
from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
moreover with *Eq* **have** $yvec = []$
by(*case-tac yvec*) *auto*
ultimately show *?case* **using** *Eq*
by(*simp add*: *boundOutput.inject*)
next
case(*Suc n xvec yvec M N P Q R*)
from ‹$Suc\ n = length\ xvec$›
obtain *x xvec'* **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
by(*case-tac xvec*) *auto*

from ‹$(\!|\nu*xvec|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' (Q \parallel R)$› ‹$xvec = x \# xvec'$›
obtain y $yvec'$ **where** $(\!|\nu*(x\#xvec')|\!)M \prec' P = (\!|\nu*(y\#yvec')|\!)N \prec' (Q \parallel R)$
and $yvec = y\#yvec'$
by(*case-tac yvec*) *auto*
hence *Eq*: $(\!|\nu x|\!)((\!|\nu*xvec'|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu*yvec'|\!)N \prec' (Q \parallel R))$
by *simp*
from ‹$xvec = x\#xvec'$› ‹$yvec=y\#yvec'$› ‹$xvec \sharp* yvec$› **have** $x \neq y$ **and** $x \sharp yvec'$ **and** $y \sharp xvec'$ **and** $xvec' \sharp* yvec'$
by *auto*
from *Eq* ‹$x \neq y$› **have** *Eq'*: $(\!|\nu*xvec'|\!)M \prec' P = [(x, y)] \cdot (\!|\nu*yvec'|\!)N \prec' (Q \parallel R)$
and *xFreshQR*: $x \sharp (\!|\nu*yvec'|\!)N \prec' (Q \parallel R)$
by(*simp add*: *boundOutput.inject alpha*)+
have *IH*: $\bigwedge xvec\ yvec\ M\ N\ P\ Q\ R.\ [\![(\!|\nu*xvec|\!)M \prec' (P::('a, 'b, 'c)\ psi) = (\!|\nu*yvec|\!)N \prec' (Q \parallel R);\ xvec \sharp* yvec;\ n = length\ xvec]\!] \Longrightarrow \exists p\ T.\ set\ p \subseteq set\ xvec \times set\ yvec \wedge P = T \parallel (p \cdot R) \wedge (\!|\nu*xvec|\!)M \prec' T = (\!|\nu*yvec|\!)N \prec' Q$
by *fact*
show *?case*
proof(*case-tac* $x \sharp (\!|\nu*xvec'|\!)M \prec' P$)
assume $x \sharp (\!|\nu*xvec'|\!)M \prec' P$
with *Eq* **have** *yFreshQR*: $y \sharp (\!|\nu*yvec'|\!)N \prec' (Q \parallel R)$
by(*rule boundOutputEqFresh*)
with *Eq'* *xFreshQR* **have** $(\!|\nu*xvec'|\!)M \prec' P = (\!|\nu*yvec'|\!)N \prec' (Q \parallel R)$
by *simp*
with ‹$xvec' \sharp* yvec'$› ‹$length\ xvec' = n$›
obtain p T **where** S: $set\ p \subseteq set\ xvec' \times set\ yvec'$ **and** $P = T \parallel (p \cdot R)$ **and** A: $(\!|\nu*xvec'|\!)M \prec' T = (\!|\nu*yvec'|\!)N \prec' Q$
by(*drule-tac IH*) *auto*
from *yFreshQR* *xFreshQR* **have** *yFreshQ*: $y \sharp (\!|\nu*yvec'|\!)N \prec' Q$ **and** *xFreshQ*: $x \sharp (\!|\nu*yvec'|\!)N \prec' Q$
by(*force simp add*: *BOresChainSupp fresh-def boundOutput.supp psi.supp*)+
hence $(\!|\nu x|\!)((\!|\nu*yvec'|\!)N \prec' Q) = (\!|\nu y|\!)((\!|\nu*yvec'|\!)N \prec' Q)$ **by** (*subst alphaBoundOutput*) *simp*+
with *A* **have** $(\!|\nu x|\!)((\!|\nu*xvec'|\!)M \prec' T) = (\!|\nu y|\!)((\!|\nu*yvec'|\!)N \prec' Q)$ **by** *simp*
with ‹$xvec=x\#xvec'$› ‹$yvec=y\#yvec'$› S ‹$P = T \parallel (p \cdot R)$› **show** *?case*
by *auto*
next
assume $\neg(x \sharp (\!|\nu*xvec'|\!)M \prec' P)$
hence $x \in supp((\!|\nu*xvec'|\!)M \prec' P)$ **by**(*simp add*: *fresh-def*)
with *Eq* **have** $y \in supp((\!|\nu*yvec'|\!)N \prec' (Q \parallel R))$
by(*rule boundOutputEqSupp*)
hence $y \sharp yvec'$ **by**(*simp add*: *BOresChainSupp fresh-def*)
with *Eq'* ‹$x \sharp yvec'$› **have** $(\!|\nu*xvec'|\!)M \prec' P = (\!|\nu*yvec'|\!)([(x, y)] \cdot N) \prec' (([(x, y)] \cdot Q) \parallel ([(x, y)] \cdot R))$
by(*simp add*: *eqvts*)
moreover note ‹$xvec' \sharp* yvec'$› ‹$length\ xvec' = n$›
ultimately obtain p T **where** S: $set\ p \subseteq set\ xvec' \times set\ yvec'$ **and** $P = T \parallel (p \cdot [(x, y)] \cdot R)$ **and** A: $(\!|\nu*xvec'|\!)M \prec' T = (\!|\nu*yvec'|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$

by(*drule-tac IH*) *auto*

from S **have** $set(p@[(x, y)]) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
moreover from ‹$P = T \parallel (p \cdot [(x, y)] \cdot R)$› **have** $P = T \parallel ((p @ [(x, y)]) \cdot R)$
by(*simp add*: *pt2*[*OF pt-name-inst*])
moreover from *xFreshQR* **have** *xFreshQ*: $x \sharp (\!|\nu{*}yvec'|\!)N \prec' Q$
by(*force simp add*: *BOresChainSupp fresh-def boundOutput.supp psi.supp*)+
with ‹$x \sharp yvec'$› ‹$y \sharp yvec'$› ‹$x \neq y$› **have** $y \sharp (\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
by(*simp add*: *fresh-left calc-atm*)
with ‹$x \sharp yvec'$› ‹$y \sharp yvec'$› **have** $(\!|\nu x|\!)((\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' Q)$
by(*subst alphaBoundOutput*) (*assumption* | *simp add*: *eqvts*)+
with A **have** $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' T) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' Q)$ **by** *simp*
ultimately show *?thesis* **using** ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$›
by(*rule-tac* $x{=}p@[(x, y)]$ **in** *exI*) *force*
qed
qed
ultimately show *?thesis*
by *blast*
qed

lemma *boundOutputPar2Dest*:
fixes *xvec* :: *name list*
and M :: $'a$::*fs-name*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*
and *yvec* :: *name list*
and N :: $'a$
and Q :: ($'a$, $'b$, $'c$) *psi*
and R :: ($'a$, $'b$, $'c$) *psi*

assumes $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$
and $xvec \sharp* Q$
and $yvec \sharp* Q$

obtains T **where** $P = Q \parallel T$ **and** $(\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R$
proof −
assume $\bigwedge T.\ [\![P = Q \parallel T;\ (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R]\!] \Longrightarrow thesis$
moreover obtain n **where** $n = length\ xvec$ **by** *auto*
with *assms* **have** $\exists T.\ P = Q \parallel T \wedge (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R$
proof(*induct n arbitrary*: *xvec yvec M N P Q R*)
case(*0 xvec yvec M N P Q R*)
have *Eq*: $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$ **by** *fact*
from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
moreover with *Eq* **have** $yvec = []$
by(*case-tac yvec*) *auto*
ultimately show *?case* **using** *Eq*
by(*simp add*: *boundOutput.inject*)

next
case(*Suc n xvec yvec M N P Q R*)
from ‹*Suc n = length xvec*›
obtain *x xvec′* **where** *xvec = x#xvec′* **and** *length xvec′ = n*
by(*case-tac xvec*) *auto*
from ‹⦇ν*xvec⦈M ≺′ P = ⦇ν*yvec⦈N ≺′ (Q ∥ R)› ‹*xvec = x # xvec′*›
obtain *y yvec′* **where** ⦇ν*(*x#xvec′*)⦈M ≺′ P = ⦇ν*(*y#yvec′*)⦈N ≺′ (Q ∥ R)*
and *yvec = y#yvec′*
by(*case-tac yvec*) *auto*
hence *EQ*: ⦇νx⦈(⦇ν*xvec′⦈M ≺′ P) = ⦇νy⦈(⦇ν*yvec′⦈N ≺′ (Q ∥ R))*
by *simp*
from ‹*xvec ♯∗ Q*› ‹*yvec ♯∗ Q*› ‹*xvec = x#xvec′*› ‹*yvec = y#yvec′*›
have *x ♯ Q* **and** *xvec′ ♯∗ Q* **and** *y ♯ Q* **and** *yvec′ ♯∗ Q* **by** *auto*
have *IH*: ⋀*xvec yvec M N P Q R*. ⟦⦇ν*xvec⦈M ≺′ (P::('a, 'b, 'c) psi) = ⦇ν*yvec⦈N ≺′ (Q ∥ R); xvec ♯∗ Q; yvec ♯∗ Q; n = length xvec⟧ ⟹ ∃ T. P = Q ∥ T ∧ ⦇ν*xvec⦈M ≺′ T = ⦇ν*yvec⦈N ≺′ R*
by *fact*
show *?case*
proof(*case-tac x = y*)
assume *x = y*
with *EQ* **have** ⦇ν*xvec′⦈M ≺′ P = ⦇ν*yvec′⦈N ≺′ (Q ∥ R)*
by(*simp add*: *boundOutput.inject alpha*)
with ‹*xvec′ ♯∗ Q*› ‹*yvec′ ♯∗ Q*› ‹*length xvec′ = n*›
obtain *T* **where** *P = Q ∥ T* **and** ⦇ν*xvec′⦈M ≺′ T = ⦇ν*yvec′⦈N ≺′ R*
by(*drule-tac IH*) *auto*
with ‹*xvec=x#xvec′*› ‹*yvec=y#yvec′*› ‹*x=y*› **show** *?case*
by(*force simp add*: *boundOutput.inject alpha*)
next
assume *x ≠ y*
with *EQ* ‹*x ♯ Q*› ‹*y ♯ Q*›
have ⦇ν*xvec′⦈M ≺′ P = ⦇ν*([(x, y)] · yvec′)⦈([(x, y)] · N) ≺′ (Q ∥ ([(x, y)] · R))*
and *xFreshQR*: *x ♯* ⦇ν*yvec′⦈N ≺′ (Q ∥ R)*
by(*simp add*: *boundOutput.inject alpha eqvts*)+
moreover from ‹*yvec′ ♯∗ Q*› **have** ([(x, y)] · *yvec′*) ♯∗ ([(x, y)] · *Q*)
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*x ♯ Q*› ‹*y ♯ Q*› **have** ([(x, y)] · *yvec′*) ♯∗ *Q* **by** *simp*
moreover note ‹*xvec′ ♯∗ Q*› ‹*length xvec′ = n*›
ultimately obtain *T* **where** *P = Q ∥ T* **and** *A*: ⦇ν*xvec′⦈M ≺′ T = ⦇ν*([(x, y)] · yvec′)⦈([(x, y)] · N) ≺′ ([(x, y)] · R)*
by(*drule-tac IH*) *auto*

from *A* **have** ⦇νx⦈(⦇ν*xvec′⦈M ≺′ T) = ⦇νx⦈(⦇ν*([(x, y)] · yvec′)⦈([(x, y)] · N) ≺′ ([(x, y)] · R))*
by(*simp add*: *boundOutput.inject alpha*)
moreover from *xFreshQR* **have** *x ♯* ⦇ν*yvec′⦈N ≺′ R*
by(*force simp add*: *boundOutputFresh*)
ultimately show *?thesis* **using** ‹*P = Q ∥ T*› ‹*xvec=x#xvec′*› ‹*yvec=y#yvec′*› *xFreshQR*

by(*force simp add*: *alphaBoundOutput name-swap eqvts*)
qed
qed
ultimately show *?thesis*
by *blast*
qed

lemma *boundOutputPar2Dest'*:
fixes *xvec* :: *name list*
and *M* :: *'a::fs-name*
and *P* :: (*'a*, *'b::fs-name*, *'c::fs-name*) *psi*
and *yvec* :: *name list*
and *N* :: *'a*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *R* :: (*'a*, *'b*, *'c*) *psi*

assumes $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$
and $xvec \sharp* yvec$

obtains $T\ p$ **where** $set\ p \subseteq set\ xvec \times set\ yvec$ **and** $P = (p \cdot Q) \parallel T$ **and** $(\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R$
proof −
assume $\bigwedge p\ T.\ [\![set\ p \subseteq set\ xvec \times set\ yvec;\ P = (p \cdot Q) \parallel T;\ (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R]\!] \Longrightarrow thesis$
moreover obtain *n* **where** $n = length\ xvec$ **by** *auto*
with *assms* **have** $\exists p\ T.\ set\ p \subseteq set\ xvec \times set\ yvec \wedge P = (p \cdot Q) \parallel T \wedge (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R$
proof(*induct n arbitrary*: *xvec yvec M N P Q R*)
case(*0 xvec yvec M N P Q R*)
have *Eq*: $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$ **by** *fact*
from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
moreover with *Eq* **have** $yvec = []$
by(*case-tac yvec*) *auto*
ultimately show *?case* **using** *Eq*
by(*simp add*: *boundOutput.inject*)
next
case(*Suc n xvec yvec M N P Q R*)
from ‹$Suc\ n = length\ xvec$›
obtain *x xvec'* **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
by(*case-tac xvec*) *auto*
from ‹$(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R)$› ‹$xvec = x\ \#\ xvec'$›
obtain *y yvec'* **where** $(\!|\nu{*}(x\#xvec')|\!)M \prec' P = (\!|\nu{*}(y\#yvec')|\!)N \prec' (Q \parallel R)$
and $yvec = y\#yvec'$
by(*case-tac yvec*) *auto*
hence *Eq*: $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R))$
by *simp*
from ‹$xvec = x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹$xvec \sharp* yvec$› **have** $x \neq y$ **and** $x \sharp yvec'$ **and** $y \sharp xvec'$ **and** $xvec' \sharp* yvec'$
by *auto*

from *Eq* ‹$x \neq y$› **have** *Eq'*: $(\!|\nu{*}xvec'|\!)M \prec' P = [(x, y)] \cdot (\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R)$
and *xFreshQR*: $x \sharp (\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R)$
by(*simp add*: *boundOutput.inject alpha*)+
have *IH*: $\bigwedge xvec\ yvec\ M\ N\ P\ Q\ R.\ [\![(\!|\nu{*}xvec|\!)M \prec' (P::('a, 'b, 'c)\ psi) = (\!|\nu{*}yvec|\!)N \prec' (Q \parallel R);\ xvec \sharp{*}\ yvec;\ n = length\ xvec]\!] \Longrightarrow \exists p\ T.\ set\ p \subseteq set\ xvec \times set\ yvec \wedge P = (p \cdot Q) \parallel T \wedge (\!|\nu{*}xvec|\!)M \prec' T = (\!|\nu{*}yvec|\!)N \prec' R$
by *fact*
show *?case*
proof(*case-tac* $x \sharp (\!|\nu{*}xvec'|\!)M \prec' P$)
assume $x \sharp (\!|\nu{*}xvec'|\!)M \prec' P$
with *Eq* **have** *yFreshQR*: $y \sharp (\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R)$
by(*rule boundOutputEqFresh*)
with *Eq' xFreshQR* **have** $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R)$
by *simp*
with ‹$xvec' \sharp{*}\ yvec'$› ‹$length\ xvec' = n$›
obtain *p T* **where** *S*: $set\ p \subseteq set\ xvec' \times set\ yvec'$ **and** $P = (p \cdot Q) \parallel T$ **and** *A*: $(\!|\nu{*}xvec'|\!)M \prec' T = (\!|\nu{*}yvec'|\!)N \prec' R$
by(*drule-tac IH*) *auto*
from *yFreshQR xFreshQR* **have** *yFreshR*: $y \sharp (\!|\nu{*}yvec'|\!)N \prec' R$ **and** *xFreshQ*: $x \sharp (\!|\nu{*}yvec'|\!)N \prec' R$
by(*force simp add*: *BOresChainSupp fresh-def boundOutput.supp psi.supp*)+
hence $(\!|\nu x|\!)((\!|\nu{*}yvec'|\!)N \prec' R) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' R)$ **by** (*subst alphaBoundOutput*) *simp*+
with *A* **have** $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' T) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' R)$ **by** *simp*
with ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$› *S* ‹$P = (p \cdot Q) \parallel T$› **show** *?case*
by *auto*
next
assume $\neg(x \sharp (\!|\nu{*}xvec'|\!)M \prec' P)$
hence $x \in supp((\!|\nu{*}xvec'|\!)M \prec' P)$ **by**(*simp add*: *fresh-def*)
with *Eq* **have** $y \in supp((\!|\nu{*}yvec'|\!)N \prec' (Q \parallel R))$
by(*rule boundOutputEqSupp*)
hence $y \sharp yvec'$ **by**(*simp add*: *BOresChainSupp fresh-def*)
with *Eq'* ‹$x \sharp yvec'$› **have** $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' (([(x, y)] \cdot Q) \parallel ([(x, y)] \cdot R))$
by(*simp add*: *eqvts*)
moreover note ‹$xvec' \sharp{*}\ yvec'$› ‹$length\ xvec' = n$›
ultimately obtain *p T* **where** *S*: $set\ p \subseteq set\ xvec' \times set\ yvec'$ **and** $P = (p \cdot [(x, y)] \cdot Q) \parallel T$ **and** *A*: $(\!|\nu{*}xvec'|\!)M \prec' T = (\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot R)$
by(*drule-tac IH*) *auto*

from *S* **have** $set(p@[(x, y)]) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
moreover from ‹$P = (p \cdot [(x, y)] \cdot Q) \parallel T$› **have** $P = ((p\ @\ [(x, y)]) \cdot Q) \parallel T$
by(*simp add*: *pt2*[*OF pt-name-inst*])
moreover from *xFreshQR* **have** *xFreshR*: $x \sharp (\!|\nu{*}yvec'|\!)N \prec' R$
by(*force simp add*: *BOresChainSupp fresh-def boundOutput.supp psi.supp*)+
with ‹$x \sharp yvec'$› ‹$y \sharp yvec'$› ‹$x \neq y$› **have** $y \sharp (\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' ([(x,$

$y)] \cdot R)$
by(*simp add*: *fresh-left calc-atm*)
with ‹$x \sharp yvec'$› ‹$y \sharp yvec'$› **have** $(\!|\nu x|\!)((\!|\nu{*}yvec'|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot R)) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' R)$
by(*subst alphaBoundOutput*) (*assumption* | *simp add*: *eqvts*)+
with A **have** $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' T) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' R)$ **by** *simp*
ultimately show *?thesis* **using** ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$›
by(*rule-tac* $x{=}p@[(x, y)]$ **in** *exI*) *force*
qed
qed
ultimately show *?thesis*
by *blast*
qed

lemma *boundOutputApp*:
fixes *xvec* :: *name list*
and *yvec* :: *name list*
and B :: ($'a$::*fs-name*, $'b$::*fs-name*, $'c$::*fs-name*) *boundOutput*

shows $(\!|\nu{*}(xvec@yvec)|\!)B = (\!|\nu{*}xvec|\!)((\!|\nu{*}yvec|\!)B)$
by(*induct xvec*) *auto*

lemma *openInjectAuxAuxAux*:
fixes x :: *name*
and *xvec* :: *name list*

shows $\exists\, y\ yvec.\ x \# xvec = yvec @ [y] \wedge length\ xvec = length\ yvec$
apply(*induct xvec arbitrary*: x)
apply *auto*
apply(*subgoal-tac* $\exists\, y\ yvec.\ a \# xvec = yvec @ [y] \wedge length\ xvec = length\ yvec$)
apply(*clarify*)
apply(*rule-tac* $x{=}y$ **in** *exI*)
by *auto*

lemma *openInjectAuxAux*:
fixes *xvec1* :: *name list*
and *xvec2* :: *name list*
and *yvec* :: *name list*

assumes $length(xvec1@xvec2) = length\ yvec$

shows $\exists\, yvec1\ yvec2.\ yvec = yvec1@yvec2 \wedge length\ xvec1 = length\ yvec1 \wedge length\ xvec2 = length\ yvec2$
using *assms*
apply(*induct yvec arbitrary*: *xvec1*)
apply *simp*
apply *simp*
apply(*case-tac xvec1*)
apply *simp*

apply *simp*
apply(*subgoal-tac* ∃ *yvec1 yvec2*.
yvec = *yvec1* @ *yvec2* ∧ *length list* = *length yvec1* ∧ *length xvec2* = *length yvec2*)
apply(*clarify*)
apply(*rule-tac x=a#yvec1* **in** *exI*)
apply(*rule-tac x=yvec2* **in** *exI*)
by *auto*

lemma *openInjectAux*:
fixes *xvec1* :: *name list*
and *x* :: *name*
and *xvec2* :: *name list*
and *yvec* :: *name list*

assumes *length*(*xvec1@x#xvec2*) = *length yvec*

shows ∃ *yvec1 y yvec2*. *yvec* = *yvec1@y#yvec2* ∧ *length xvec1* = *length yvec1* ∧ *length xvec2* = *length yvec2*
using *assms*
apply(*case-tac yvec*)
apply *simp*
apply *simp*
apply(*subgoal-tac* ∃ (*yvec1*::*name list*) (*yvec2*::*name list*). *yvec1@yvec2* = *list* ∧ *length xvec1* = *length yvec1* ∧ *length xvec2* = *length yvec2*)
apply(*clarify*)
apply *hypsubst-thin*
apply *simp*
apply(*subgoal-tac* ∃ *y* (*yvec*::*name list*). *a* # *yvec1* = *yvec* @ [*y*] ∧ *length yvec1* = *length yvec*)
apply(*clarify*)
apply(*rule-tac x=yvec* **in** *exI*)
apply(*rule-tac x=y* **in** *exI*)
apply *simp*
apply(*rule-tac x=yvec2* **in** *exI*)
apply *simp*
apply(*rule openInjectAuxAuxAux*)
apply(*insert openInjectAuxAux*)
apply *simp*
by *blast*

lemma *boundOutputOpenDest*:
fixes *yvec* :: *name list*
and *M* :: *'a*::*fs-name*
and *P* :: (*'a*, *'b*::*fs-name*, *'c*::*fs-name*) *psi*
and *xvec1* :: *name list*
and *x* :: *name*
and *xvec2* :: *name list*
and *N* :: *'a*

and Q :: $('a, 'b, 'c)$ *psi*

assumes *Eq*: $(\!|\nu*(xvec1@x\#xvec2)|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' Q$
and $x \sharp xvec1$
and $x \sharp yvec$
and $x \sharp N$
and $x \sharp Q$
and *distinct yvec*

obtains *yvec1 y yvec2* **where** *yvec=yvec1@y#yvec2* **and** *length xvec1 = length yvec1* **and** *length xvec2 = length yvec2*
and $(\!|\nu*(xvec1@xvec2)|\!)M \prec' P = (\!|\nu*(yvec1@yvec2)|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
proof −
assume *Ass*: $\bigwedge yvec1\ y\ yvec2.$
$[\![yvec = yvec1\ @\ y\ \#\ yvec2;\ length\ xvec1 = length\ yvec1;\ length\ xvec2 = length\ yvec2;$
$(\!|\nu*(xvec1\ @\ xvec2)|\!)M \prec' P = (\!|\nu*(yvec1\ @\ yvec2)|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)]\!]$
$\Longrightarrow$ *thesis*
from *Eq* **have** $length(xvec1@x\#xvec2) = length\ yvec$ **by**(*rule boundOutputChainEqLength*)
then obtain *yvec1 y yvec2* **where** *A*: $yvec = yvec1@y\#yvec2$ **and** *length xvec1 = length yvec1*
and *length xvec2 = length yvec2*
by(*metis openInjectAux sym*)

from ‹*distinct yvec*› *A* **have** $y \sharp yvec2$ **by** *simp*
from *A* ‹$x \sharp yvec$› **have** $x \sharp yvec2$ **and** $x \sharp yvec1$ **by** *simp+*
with *Eq* ‹*length xvec1 = length yvec1*› ‹$x \sharp N$› ‹$x \sharp Q$› ‹$y \sharp yvec2$› ‹$x \sharp xvec1$› *A*
have $(\!|\nu*(xvec1@xvec2)|\!)M \prec' P = (\!|\nu*(yvec1@yvec2)|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
by(*force dest*: *boundOutputChainOpenIH simp add*: *boundOutputApp BOresChainSupp fresh-def boundOutput.supp eqvts*)
with ‹*length xvec1 = length yvec1*› ‹*length xvec2 = length yvec2*› *A Ass* **show** *?thesis*
by *blast*
qed

lemma *boundOutputOpenDest'*:
fixes *yvec* :: *name list*
and M :: $'a$::*fs-name*
and P :: $('a, 'b$::*fs-name*, $'c$::*fs-name*$)$ *psi*
and *xvec1* :: *name list*
and x :: *name*
and *xvec2* :: *name list*
and N :: $'a$
and Q :: $('a, 'b, 'c)$ *psi*

assumes *Eq*: $(\!|\nu*(xvec1@x\#xvec2)|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' Q$
and $x \sharp xvec1$
and $x \sharp yvec$
and $x \sharp N$
and $x \sharp Q$

obtains *yvec1 y yvec2* **where** *yvec=yvec1@y#yvec2* **and** *length xvec1 = length yvec1* **and** *length xvec2 = length yvec2*
and $(\!|\nu*(xvec1@xvec2)|\!)M \prec' P = (\!|\nu*(yvec1@[(x, y)] \cdot yvec2)|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
proof −
assume *Ass*: $\bigwedge yvec1\ y\ yvec2.$
$[\![yvec = yvec1\ @\ y\ \#\ yvec2;\ length\ xvec1 = length\ yvec1;\ length\ xvec2 = length\ yvec2;$
$(\!|\nu*(xvec1\ @\ xvec2)|\!)M \prec' P = (\!|\nu*(yvec1\ @\ ([(x, y)] \cdot yvec2))|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)]\!]$
$\Longrightarrow thesis$
from *Eq* **have** *length*$(xvec1@x\#xvec2) = length\ yvec$ **by**(*rule boundOutputChainEqLength*)
then obtain *yvec1 y yvec2* **where** *A*: $yvec = yvec1@y\#yvec2$ **and** *length xvec1 = length yvec1*
and *length xvec2 = length yvec2*
by(*metis openInjectAux sym*)

from *A* ‹$x \sharp yvec$› **have** $x \sharp yvec2$ **and** $x \sharp yvec1$ **by** *simp+*
with *Eq* ‹*length xvec1 = length yvec1*› ‹$x \sharp N$› ‹$x \sharp Q$› ‹$x \sharp xvec1$› *A*
have $(\!|\nu*(xvec1@xvec2)|\!)M \prec' P = (\!|\nu*(yvec1@([(x, y)] \cdot yvec2))|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
by(*force dest*: *boundOutputChainOpenIH simp add*: *boundOutputApp BOresChainSupp fresh-def boundOutput.supp eqvts*)
with ‹*length xvec1 = length yvec1*› ‹*length xvec2 = length yvec2*› *A Ass* **show** *?thesis*
by *blast*
qed

lemma *boundOutputScopeDest*:
fixes *xvec* :: *name list*
and *M* :: *'a::fs-name*
and *P* :: (*'a*, *'b::fs-name*, *'c::fs-name*) *psi*
and *yvec* :: *name list*
and *N* :: *'a*
and *x* :: *name*
and *Q* :: (*'a*, *'b*, *'c*) *psi*

assumes $(\!|\nu*xvec|\!)M \prec' P = (\!|\nu*yvec|\!)N \prec' (\!|\nu z|\!)Q$
and $z \sharp xvec$
and $z \sharp yvec$

obtains R **where** $P = (\!|\nu z|\!)R$ **and** $(\!|\nu*xvec|\!)M \prec' R = (\!|\nu*yvec|\!)N \prec' Q$

proof −
  assume $\bigwedge R.\ [\![P = (\!|\nu z|\!)R;\ (\!|\nu{*}xvec|\!)M \prec' R = (\!|\nu{*}yvec|\!)N \prec' Q]\!] \Longrightarrow thesis$
  moreover obtain n **where** $n = length\ xvec$ **by** *auto*
  with *assms* **have** $\exists R.\ P = (\!|\nu z|\!)R \wedge (\!|\nu{*}xvec|\!)M \prec' R = (\!|\nu{*}yvec|\!)N \prec' Q$
  proof(*induct n arbitrary*: *xvec yvec M N P Q z*)
    case(*0 xvec yvec M N P Q z*)
    have *Eq*: $(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' (\!|\nu z|\!)Q$ **by** *fact*
    from ‹$0 = length\ xvec$› **have** $xvec = []$ **by** *auto*
    moreover with *Eq* **have** $yvec = []$
      by(*case-tac yvec*) *auto*
    ultimately show *?case* **using** *Eq*
      by(*simp add*: *boundOutput.inject*)
  next
    case(*Suc n xvec yvec M N P Q z*)
    from ‹$Suc\ n = length\ xvec$›
    obtain $x\ xvec'$ **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
      by(*case-tac xvec*) *auto*
    from ‹$(\!|\nu{*}xvec|\!)M \prec' P = (\!|\nu{*}yvec|\!)N \prec' ((\!|\nu z|\!)Q)$› ‹$xvec = x\ \#\ xvec'$›
    obtain $y\ yvec'$ **where** $(\!|\nu{*}(x\#xvec')|\!)M \prec' P = (\!|\nu{*}(y\#yvec')|\!)N \prec' (\!|\nu z|\!)Q$
      and $yvec = y\#yvec'$
      by(*case-tac yvec*) *auto*
    hence *EQ*: $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' P) = (\!|\nu y|\!)((\!|\nu{*}yvec'|\!)N \prec' (\!|\nu z|\!)Q)$
      by *simp*
    from ‹$z \sharp xvec$› ‹$z \sharp yvec$› ‹$xvec = x\#xvec'$› ‹$yvec = y\#yvec'$›
    have $z \neq x$ **and** $z \neq y$ **and** $z \sharp xvec'$ **and** $z \sharp yvec'$
      by *simp+*
     have *IH*: $\bigwedge xvec\ yvec\ M\ N\ P\ Q\ z.\ [\![(\!|\nu{*}xvec|\!)M \prec' (P::('a,\ 'b,\ 'c)\ psi) = (\!|\nu{*}yvec|\!)N \prec' (\!|\nu z|\!)Q;\ z \sharp xvec;\ z \sharp yvec;\ n = length\ xvec]\!] \Longrightarrow \exists R.\ P = (\!|\nu z|\!)R \wedge (\!|\nu{*}xvec|\!)M \prec' R = (\!|\nu{*}yvec|\!)N \prec' Q$
      by *fact*
    show *?case*
    proof(*case-tac* $x = y$)
      assume $x = y$
      with *EQ* **have** $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}yvec'|\!)N \prec' (\!|\nu z|\!)Q$
        by(*simp add*: *boundOutput.inject alpha*)
      with ‹$z \sharp xvec'$› ‹$z \sharp yvec'$› ‹$length\ xvec' = n$›
      obtain R **where** $P = (\!|\nu z|\!)R$ **and** $(\!|\nu{*}xvec'|\!)M \prec' R = (\!|\nu{*}yvec'|\!)N \prec' Q$
        by(*drule-tac IH*) *auto*
      with ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹$x{=}y$› **show** *?case*
        by(*force simp add*: *boundOutput.inject alpha*)
    next
      assume $x \neq y$
      with *EQ* ‹$z \neq x$› ‹$z \neq y$›
      have $(\!|\nu{*}xvec'|\!)M \prec' P = (\!|\nu{*}([(x,\ y)] \cdot yvec')|\!)([(x,\ y)] \cdot N) \prec' (\!|\nu z|\!)([(x,\ y)] \cdot Q)$
       and *xFreshzQ*: $x \sharp (\!|\nu{*}yvec'|\!)N \prec' (\!|\nu z|\!)Q$
        by(*simp add*: *boundOutput.inject alpha eqvts*)+
     moreover from ‹$z \neq x$› ‹$z \neq y$› ‹$z \sharp yvec'$› ‹$x \neq y$› **have** $z \sharp ([(x,\ y)] \cdot yvec')$
        by(*simp add*: *fresh-left calc-atm*)

moreover note ‹$z \sharp xvec'$› ‹$length\ xvec' = n$›
ultimately obtain R **where** $P = (\!|\nu z|\!)R$ **and** A: $(\!|\nu{*}xvec'|\!)M \prec' R = (\!|\nu{*}([(x, y)] \cdot yvec')|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q)$
by(*drule-tac IH*) *auto*

from A **have** $(\!|\nu x|\!)((\!|\nu{*}xvec'|\!)M \prec' R) = (\!|\nu x|\!)((\!|\nu{*}([(x, y)] \cdot yvec')|\!)([(x, y)] \cdot N) \prec' ([(x, y)] \cdot Q))$
by(*simp add*: *boundOutput.inject alpha*)
moreover from *xFreshzQ* ‹$z \neq x$› **have** $x \sharp (\!|\nu{*}yvec'|\!)N \prec' Q$
by(*simp add*: *boundOutputFresh abs-fresh*)
ultimately show *?thesis* **using** ‹$P = (\!|\nu z|\!)R$› ‹*xvec=x#xvec'*› ‹*yvec=y#yvec'*› *xFreshzQ*
by(*force simp add*: *alphaBoundOutput name-swap eqvts*)
qed
qed
ultimately show *?thesis*
by *blast*
qed

nominal-datatype ('*a*, '*b*, '*c*) *residual* =
RIn '*a*::*fs-name* '*a* ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
| *ROut* '*a* ('*a*, '*b*, '*c*) *boundOutput*
| *RTau* ('*a*, '*b*, '*c*) *psi*

nominal-datatype '*a action* = *In* '*a*::*fs-name* '*a* (‹-(|-|)› [*90*, *90*] *90*)
| *Out* '*a*::*fs-name name list* '*a* (‹-(|ν*-|)⟨-⟩› [*90*, *90*, *90*] *90*)
| *Tau* (‹τ› *90*)

nominal-primrec *bn* :: ('*a*::*fs-name*) *action* ⇒ *name list*
where
bn ($M(\!|N|\!)$) = []
| *bn* ($M(\!|\nu{*}xvec|\!)\langle N\rangle$) = *xvec*
| *bn* (τ) = []
by(*rule TrueI*)+

lemma *bnEqvt*[*eqvt*]:
fixes p :: *name prm*
and α :: ('*a*::*fs-name*) *action*

shows $(p \cdot bn\ \alpha) = bn(p \cdot \alpha)$
by(*nominal-induct* α *rule*: *action.strong-induct*) *auto*

nominal-primrec *create-residual* :: ('*a*::*fs-name*) *action* ⇒ ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi* ⇒ ('*a*, '*b*, '*c*) *residual* (‹- ≺ -› [*80*, *80*] *80*)
where
$(M(\!|N|\!)) \prec P = RIn\ M\ N\ P$
| $M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P = ROut\ M\ ((\!|\nu{*}xvec|\!)(N \prec' P))$
| $\tau \prec P = (RTau\ P)$
by(*rule TrueI*)+

nominal-primrec *subject* :: ($'a$::*fs-name*) *action* $\Rightarrow$ $'a$ *option*
where
subject $(M(\!|N|\!)) = Some\ M$
| *subject* $(M(\!|\nu{*}xvec|\!)\langle N\rangle) = Some\ M$
| *subject* $(\tau) = None$
by(*rule TrueI*)+

nominal-primrec *object* :: ($'a$::*fs-name*) *action* $\Rightarrow$ $'a$ *option*
where
object $(M(\!|N|\!)) = Some\ N$
| *object* $(M(\!|\nu{*}xvec|\!)\langle N\rangle) = Some\ N$
| *object* $(\tau) = None$
by(*rule TrueI*)+

lemma *optionFreshChain*[*simp*]:
fixes *xvec* :: *name list*
and X :: *name set*

shows $xvec \sharp{*}\ (Some\ x) = xvec \sharp{*}\ x$
and $X \sharp{*}\ (Some\ x) = X \sharp{*}\ x$
and $xvec \sharp{*}\ None$
and $X \sharp{*}\ None$
by(*auto simp add*: *fresh-star-def fresh-some fresh-none*)

lemmas [*simp*] = *fresh-some fresh-none*

lemma *actionFresh*[*simp*]:
fixes x :: *name*
and α :: ($'a$::*fs-name*) *action*

shows $(x \sharp \alpha) = (x \sharp (subject\ \alpha) \wedge x \sharp (bn\ \alpha) \wedge x \sharp (object\ \alpha))$
by(*nominal-induct* α *rule*: *action.strong-induct*) *auto*

lemma *actionFreshChain*[*simp*]:
fixes X :: *name set*
and α :: ($'a$::*fs-name*) *action*
and *xvec* :: *name list*

shows $(X \sharp{*}\ \alpha) = (X \sharp{*}\ (subject\ \alpha) \wedge X \sharp{*}\ (bn\ \alpha) \wedge X \sharp{*}\ (object\ \alpha))$
and $(xvec \sharp{*}\ \alpha) = (xvec \sharp{*}\ (subject\ \alpha) \wedge xvec \sharp{*}\ (bn\ \alpha) \wedge xvec \sharp{*}\ (object\ \alpha))$
by(*auto simp add*: *fresh-star-def*)

lemma *subjectEqvt*[*eqvt*]:
fixes p :: *name prm*
and α :: ($'a$::*fs-name*) *action*

shows $(p \cdot subject\ \alpha) = subject(p \cdot \alpha)$
by(*nominal-induct* α *rule*: *action.strong-induct*) *auto*

lemma *okjectEqvt*[*eqvt*]:
fixes p :: *name prm*
and α :: ('*a*::*fs-name*) *action*

shows $(p \cdot$ *object* $\alpha)$ = *object*$(p \cdot \alpha)$
by(*nominal-induct* α *rule*: *action.strong-induct*) *auto*

lemma *create-residualEqvt*[*eqvt*]:
fixes p :: *name prm*
and α :: ('*a*::*fs-name*) *action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

shows $(p \cdot (\alpha \prec P)) = (p \cdot \alpha) \prec (p \cdot P)$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *eqvts*)

lemma *residualFresh*:
fixes x :: *name*
and α :: '*a*::*fs-name action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

shows $(x \sharp (\alpha \prec P)) = (x \sharp (\textit{subject}\ \alpha) \wedge (x \in (\textit{set}(\textit{bn}(\alpha))) \vee (x \sharp \textit{object}(\alpha) \wedge$
$x \sharp P)))$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *fresh-some fresh-none boundOutputFresh*)

lemma *residualFresh2*[*simp*]:
fixes x :: *name*
and α :: ('*a*::*fs-name*) *action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

assumes $x \sharp \alpha$
and $x \sharp P$

shows $x \sharp \alpha \prec P$
using *assms*
by(*nominal-induct* α *rule*: *action.strong-induct*) *auto*

lemma *residualFreshChain2*[*simp*]:
fixes *xvec* :: *name list*
and X :: *name set*
and α :: ('*a*::*fs-name*) *action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*

shows $[\![\textit{xvec} \sharp* \alpha;\ \textit{xvec} \sharp* P]\!] \Longrightarrow \textit{xvec} \sharp* (\alpha \prec P)$
and $[\![X \sharp* \alpha;\ X \sharp* P]\!] \Longrightarrow X \sharp* (\alpha \prec P)$
by(*auto simp add*: *fresh-star-def*)

lemma *residualFreshSimp*[*simp*]:
fixes x :: *name*
and M :: $'a$::*fs-name*
and N :: $'a$
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $x \sharp (M(\!|N|\!) \prec P) = (x \sharp M \wedge x \sharp N \wedge x \sharp P)$
and $x \sharp (M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P) = (x \sharp M \wedge x \sharp ((\!|\nu{*}xvec|\!)(N \prec' P)))$
and $x \sharp (\tau \prec P) = (x \sharp P)$
by(*auto simp add*: *residualFresh*)

lemma *residualInject'*:

shows $(\alpha \prec P = RIn\ M\ N\ Q) = (P = Q \wedge \alpha = M(\!|N|\!))$
and $(\alpha \prec P = ROut\ M\ B) = (\exists\, xvec\ N.\ \alpha = M(\!|\nu{*}xvec|\!)\langle N\rangle \wedge B = (\!|\nu{*}xvec|\!)(N \prec' P))$
and $(\alpha \prec P = RTau\ Q) = (\alpha = \tau \wedge P = Q)$
and $(RIn\ M\ N\ Q = \alpha \prec P) = (P = Q \wedge \alpha = M(\!|N|\!))$
and $(ROut\ M\ B = \alpha \prec P) = (\exists\, xvec\ N.\ \alpha = M(\!|\nu{*}xvec|\!)\langle N\rangle \wedge B = (\!|\nu{*}xvec|\!)(N \prec' P))$
and $(RTau\ Q = \alpha \prec P) = (\alpha = \tau \wedge P = Q)$
proof −
show $(\alpha \prec P = RIn\ M\ N\ Q) = (P = Q \wedge \alpha = M(\!|N|\!))$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residual.inject action.inject*)
next
show $(\alpha \prec P = ROut\ M\ B) = (\exists\, xvec\ N.\ \alpha = M(\!|\nu{*}xvec|\!)\langle N\rangle \wedge B = (\!|\nu{*}xvec|\!)(N \prec' P))$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residual.inject action.inject*)
next
show $(\alpha \prec P = RTau\ Q) = (\alpha = \tau \wedge P = Q)$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residual.inject action.inject*)
next
show $(RIn\ M\ N\ Q = \alpha \prec P) = (P = Q \wedge \alpha = M(\!|N|\!))$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residual.inject action.inject*)
next
show $(ROut\ M\ B = \alpha \prec P) = (\exists\, xvec\ N.\ \alpha = M(\!|\nu{*}xvec|\!)\langle N\rangle \wedge B = (\!|\nu{*}xvec|\!)(N \prec' P))$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residual.inject action.inject*)
next
show $(RTau\ Q = \alpha \prec P) = (\alpha = \tau \wedge P = Q)$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residual.inject action.inject*)
qed

lemma *residualFreshChainSimp*[*simp*]:
fixes *xvec* :: *name list*
and X :: *name set*
and M :: $'a$::*fs-name*
and N :: $'a$
and *yvec* :: *name list*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $xvec \sharp* (M(\!|N|\!) \prec P) = (xvec \sharp* M \wedge xvec \sharp* N \wedge xvec \sharp* P)$
and $xvec \sharp* (M(\!|\nu*yvec|\!)\langle N\rangle \prec P) = (xvec \sharp* M \wedge xvec \sharp* ((\!|\nu*yvec|\!)(N \prec' P)))$
and $xvec \sharp* (\tau \prec P) = (xvec \sharp* P)$
and $X \sharp* (M(\!|N|\!) \prec P) = (X \sharp* M \wedge X \sharp* N \wedge X \sharp* P)$
and $X \sharp* (M(\!|\nu*yvec|\!)\langle N\rangle \prec P) = (X \sharp* M \wedge X \sharp* ((\!|\nu*yvec|\!)(N \prec' P)))$
and $X \sharp* (\tau \prec P) = (X \sharp* P)$
by(*auto simp add*: *fresh-star-def*)

lemma *residualFreshChainSimp2*[*simp*]:
fixes *xvec* :: *name list*
and X :: *name set*
and M :: $'a$::*fs-name*
and N :: $'a$
and *yvec* :: *name list*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

shows $xvec \sharp* (RIn\ M\ N\ P) = (xvec \sharp* M \wedge xvec \sharp* N \wedge xvec \sharp* P)$
and $xvec \sharp* (ROut\ M\ B) = (xvec \sharp* M \wedge xvec \sharp* B)$
and $xvec \sharp* (RTau\ P) = (xvec \sharp* P)$
and $X \sharp* (RIn\ M\ N\ P) = (X \sharp* M \wedge X \sharp* N \wedge X \sharp* P)$
and $X \sharp* (ROut\ M\ B) = (X \sharp* M \wedge X \sharp* B)$
and $X \sharp* (RTau\ P) = (X \sharp* P)$
by(*auto simp add*: *fresh-star-def*)

lemma *freshResidual3*[*dest*]:
fixes x :: *name*
and α :: ($'a$::*fs-name*) *action*
and P :: ($'a$, $'b$::*fs-name*, $'c$::*fs-name*) *psi*

assumes $x \sharp bn\ \alpha$
and $x \sharp \alpha \prec P$

shows $x \sharp \alpha$ **and** $x \sharp P$
using *assms*
by(*nominal-induct rule*: *action.strong-induct*) *auto*

lemma *freshResidualChain3*[*dest*]:
fixes *xvec* :: *name list*
and α :: ($'a$::*fs-name*) *action*

and P :: ('a, 'b::*fs-name*, 'c::*fs-name*) *psi*

assumes *xvec* $\sharp*$ ($\alpha \prec P$)
and *xvec* $\sharp*$ *bn* α

shows *xvec* $\sharp*$ α **and** *xvec* $\sharp*$ P
using *assms*
by(*nominal-induct rule*: *action.strong-induct*) *auto*

lemma *freshResidual4*[*dest*]:
fixes x :: *name*
and α :: ('a::*fs-name*) *action*
and P :: ('a, 'b::*fs-name*, 'c::*fs-name*) *psi*

assumes $x \sharp \alpha \prec P$

shows $x \sharp$ *subject* α
using *assms*
by(*nominal-induct rule*: *action.strong-induct*) *auto*

lemma *freshResidualChain4*[*dest*]:
fixes *xvec* :: *name list*
and α :: ('a::*fs-name*) *action*
and P :: ('a, 'b::*fs-name*, 'c::*fs-name*) *psi*

assumes *xvec* $\sharp*$ ($\alpha \prec P$)

shows *xvec* $\sharp*$ *subject* α
using *assms*
by(*nominal-induct rule*: *action.strong-induct*) *auto*

lemma *alphaOutputResidual*:
fixes M :: 'a::*fs-name*
and *xvec* :: *name list*
and N :: 'a
and P :: ('a, 'b::*fs-name*, 'c::*fs-name*) *psi*
and p :: *name prm*

assumes ($p \cdot$ *xvec*) $\sharp*$ N
and ($p \cdot$ *xvec*) $\sharp*$ P
and *set* $p \subseteq$ *set xvec* $\times$ *set*($p \cdot$ *xvec*)
and *set xvec* $\subseteq$ *set yvec*

shows $M(\!|\nu{*}yvec|\!)\langle N\rangle \prec P = M(\!|\nu{*}(p \cdot yvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot P)$
using *assms*
by(*simp add*: *boundOutputChainAlpha''*)

lemmas[*simp del*] = *create-residual.simps*

lemma *residualInject''*:

assumes $bn\ \alpha = bn\ \beta$

shows $(\alpha \prec P = \beta \prec Q) = (\alpha = \beta \wedge P = Q)$
using *assms*
apply(*nominal-induct* α *rule*: *action.strong-induct*)
apply(*auto simp add*: *residual.inject create-residual.simps residualInject' action.inject boundOutput.inject*)
by(*rule-tac x=bn* β **in** *exI*) *auto*

lemmas *residualInject* = *residual.inject create-residual.simps residualInject' residualInject''*

lemma *bnFreshResidual*[*simp*]:
fixes α :: ('*a*::*fs-name*) *action*

shows $(bn\ \alpha) \sharp* (\alpha \prec P) = bn\ \alpha \sharp* (subject\ \alpha)$
by(*nominal-induct* α *rule*: *action.strong-induct*)
(*auto simp add*: *residualFresh fresh-some fresh-star-def*)

lemma *actionCases*[*case-names cInput cOutput cTau*]:
fixes α :: ('*a*::*fs-name*) *action*

assumes $\bigwedge M\ N.\ \alpha = M(\!|N|\!) \Longrightarrow Prop$
and $\bigwedge M\ xvec\ N.\ \alpha = M(\!|\nu{*}xvec|\!)\langle N\rangle \Longrightarrow Prop$
and $\alpha = \tau \Longrightarrow Prop$

shows *Prop*
using *assms*
by(*nominal-induct* α *rule*: *action.strong-induct*) *auto*

lemma *actionPar1Dest*:
fixes α :: ('*a*::*fs-name*) *action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
and β :: ('*a*::*fs-name*) *action*
and Q :: ('*a*, '*b*, '*c*) *psi*
and R :: ('*a*, '*b*, '*c*) *psi*

assumes $\alpha \prec P = \beta \prec (Q \parallel R)$
and $bn\ \alpha \sharp* bn\ \beta$

obtains $T\ p$ **where** $set\ p \subseteq set(bn\ \alpha) \times set(bn\ \beta)$ **and** $P = T \parallel (p \cdot R)$ **and** $\alpha \prec T = \beta \prec Q$
using *assms*
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
by(*drule-tac boundOutputPar1Dest'*) *auto*

lemma *actionPar2Dest*:
  fixes α :: ('*a*::*fs-name*) *action*
  and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
  and β :: ('*a*::*fs-name*) *action*
  and Q :: ('*a*, '*b*, '*c*) *psi*
  and R :: ('*a*, '*b*, '*c*) *psi*

  assumes $\alpha \prec P = \beta \prec (Q \parallel R)$
  and $bn\ \alpha \sharp* bn\ \beta$

  obtains $T\ p$ **where** $set\ p \subseteq set(bn\ \alpha) \times set(bn\ \beta)$ **and** $P = (p \cdot Q) \parallel T$ **and** $\alpha \prec T = \beta \prec R$
using *assms*
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
by(*drule-tac boundOutputPar2Dest'*) *auto*

lemma *actionScopeDest*:
  fixes α :: ('*a*::*fs-name*) *action*
  and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
  fixes β :: ('*a*::*fs-name*) *action*
  and x :: *name*
  and Q :: ('*a*, '*b*, '*c*) *psi*

  assumes $\alpha \prec P = \beta \prec (\!|\nu x|\!)Q$
  and $x \sharp bn\ \alpha$
  and $x \sharp bn\ \beta$

  obtains R **where** $P = (\!|\nu x|\!)R$ **and** $\alpha \prec R = \beta \prec Q$
using *assms*
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
by(*drule-tac boundOutputScopeDest*) *auto*

abbreviation
  outputJudge (‹-⟨-⟩› [*110*, *110*] *110*) **where** $M\langle N\rangle \equiv M(\!|\nu*([])|\!)\langle N\rangle$

declare [[*unify-trace-bound*=*100*]]

locale *env* = *substPsi substTerm substAssert substCond* +
        assertion SCompose' SImp' SBottom' SChanEq'
  for *substTerm* :: ('*a*::*fs-name*) ⇒ *name list* ⇒ '*a*::*fs-name list* ⇒ '*a*
  and *substAssert* :: ('*b*::*fs-name*) ⇒ *name list* ⇒ '*a*::*fs-name list* ⇒ '*b*
  and *substCond* :: ('*c*::*fs-name*) ⇒ *name list* ⇒ '*a*::*fs-name list* ⇒ '*c*
  and *SCompose'* :: '*b* ⇒ '*b* ⇒ '*b*
  and *SImp'* :: '*b* ⇒ '*c* ⇒ *bool*
  and *SBottom'* :: '*b*
  and *SChanEq'* :: '*a* ⇒ '*a* ⇒ '*c*
begin

notation *SCompose′* (**infixr** ‹⊗› *90*)
notation *SImp′* (‹- ⊢ -› [*85*, *85*] *85*)
notation *FrameImp* (‹- $\vdash_F$ -› [*85*, *85*] *85*)
abbreviation
FBottomJudge (‹$\bot_F$› *90*) **where** $\bot_F \equiv$ (*FAssert SBottom′*)
notation *SChanEq′* (‹- ↔ -› [*90*, *90*] *90*)
notation *substTerm* (‹-[-::=-]› [*100*, *100*, *100*] *100*)
notation *subs* (‹-[-::=-]› [*100*, *100*, *100*] *100*)
notation *AssertionStatEq* (‹- ≃ -› [*80*, *80*] *80*)
notation *FrameStatEq* (‹- $\simeq_F$ -› [*80*, *80*] *80*)
notation *SBottom′* (‹**1**› *190*)
abbreviation *insertAssertion′* (‹*insertAssertion*›) **where** *insertAssertion′* ≡ *assertionAux.insertAssertion* (⊗)

inductive *semantics* :: *′b* ⇒ (*′a*, *′b*, *′c*) *psi* ⇒ (*′a*, *′b*, *′c*) *residual* ⇒ *bool*
(‹- ▷ - ⟼ -› [*50*, *50*, *50*] *50*)
where
cInput: ⟦$\Psi \vdash M \leftrightarrow K$; *distinct xvec*; *set xvec* ⊆ *supp N*; *xvec* ♯* *Tvec*;
length xvec = *length Tvec*;
xvec ♯* Ψ; *xvec* ♯* *M*; *xvec* ♯* *K*⟧ ⟹ Ψ ▷ *M*⦇λ*xvec *N*⦈.*P* ⟼ *K*⦇(*N*[*xvec*::=*Tvec*])⦈ ≺ *P*[*xvec*::=*Tvec*]
| *Output*: ⟦$\Psi \vdash M \leftrightarrow K$⟧ ⟹ Ψ ▷ *M*⟨*N*⟩.*P* ⟼ *K*⟨*N*⟩ ≺ *P*
| *Case*: ⟦Ψ ▷ *P* ⟼ *Rs*; (φ, *P*) *mem Cs*; Ψ ⊢ φ; *guarded P*⟧ ⟹ Ψ ▷ *Cases Cs* ⟼ *Rs*

| *cPar1*: ⟦(Ψ ⊗ Ψ_Q) ▷ *P* ⟼α ≺ *P′*; *extractFrame Q* = ⟨A_Q, Ψ_Q⟩; *distinct* A_Q;
A_Q ♯* *P*; A_Q ♯* *Q*; A_Q ♯* Ψ; A_Q ♯* α; A_Q ♯* *P′*; *distinct*(*bn* α);
bn α ♯* Ψ; *bn* α ♯* Ψ_Q; *bn* α ♯* *Q*; *bn* α ♯* *P*; *bn* α ♯* (*subject* α)⟧ ⟹
Ψ ▷ *P* ∥ *Q* ⟼α ≺ (*P′* ∥ *Q*)
| *cPar2*: ⟦(Ψ ⊗ Ψ_P) ▷ *Q* ⟼α ≺ *Q′*; *extractFrame P* = ⟨A_P, Ψ_P⟩; *distinct* A_P;
A_P ♯* *P*; A_P ♯* *Q*; A_P ♯* Ψ; A_P ♯* α; A_P ♯* *Q′*; *distinct*(*bn* α);
bn α ♯* Ψ; *bn* α ♯* Ψ_P; *bn* α ♯* *P*; *bn* α ♯* *Q*; *bn* α ♯* (*subject* α)⟧ ⟹
Ψ ▷ *P* ∥ *Q* ⟼α ≺ (*P* ∥ *Q′*)
| *cComm1*: ⟦Ψ ⊗ Ψ_Q ▷ *P* ⟼ *M*⦇*N*⦈ ≺ *P′*; *extractFrame P* = ⟨A_P, Ψ_P⟩; *distinct* A_P;
Ψ ⊗ Ψ_P ▷ *Q* ⟼ *K*⦇ν*xvec⦈⟨*N*⟩ ≺ *Q′*; *extractFrame Q* = ⟨A_Q, Ψ_Q⟩; *distinct* A_Q;
Ψ ⊗ Ψ_P ⊗ Ψ_Q ⊢ *M* ↔ *K*;
A_P ♯* Ψ; A_P ♯* Ψ_Q; A_P ♯* *P*; A_P ♯* *M*; A_P ♯* *N*; A_P ♯* *P′*;
A_P ♯* *Q*; A_P ♯* *Q′*; A_P ♯* A_Q; A_P ♯* *xvec*;
A_Q ♯* Ψ; A_Q ♯* Ψ_P; A_Q ♯* *P*; A_Q ♯* *N*; A_Q ♯* *P′*;
A_Q ♯* *Q*; A_Q ♯* *K*; A_Q ♯* *Q′*; A_Q ♯* *xvec*; *distinct xvec*;
xvec ♯* Ψ; *xvec* ♯* Ψ_P; *xvec* ♯* Ψ_Q; *xvec* ♯* *P*; *xvec* ♯* *M*;
xvec ♯* *Q*; *xvec* ♯* *K*⟧ ⟹
Ψ ▷ *P* ∥ *Q* ⟼ τ ≺ ⦇ν*xvec⦈(*P′* ∥ *Q′*)
| *cComm2*: ⟦Ψ ⊗ Ψ_Q ▷ *P* ⟼ *M*⦇ν*xvec⦈⟨*N*⟩ ≺ *P′*; *extractFrame P* = ⟨A_P, Ψ_P⟩; *distinct* A_P;
Ψ ⊗ Ψ_P ▷ *Q* ⟼ *K*⦇*N*⦈ ≺ *Q′*; *extractFrame Q* = ⟨A_Q, Ψ_Q⟩; *distinct* A_Q;

$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* xvec$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$;
$A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$; $A_Q \sharp* xvec$; *distinct xvec*;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* \Psi_Q$; $xvec \sharp* P$; $xvec \sharp* M$;
$xvec \sharp* Q$; $xvec \sharp* K]\!] \Longrightarrow$
$\Psi \rhd P \parallel Q \longmapsto \tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$
| *cOpen*: $[\![\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$; $x \in supp\ N$; $x \sharp xvec$; $x \sharp yvec$; $x \sharp M$; $x \sharp \Psi$;
distinct xvec; *distinct yvec*;
$xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* yvec$; $yvec \sharp* \Psi$; $yvec \sharp* P$; $yvec \sharp* M]\!] \Longrightarrow$
$\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$
| *cScope*: $[\![\Psi \rhd P \longmapsto \alpha \prec P'$; $x \sharp \Psi$; $x \sharp \alpha$; $bn\ \alpha \sharp* \Psi$; $bn\ \alpha \sharp* P$; $bn\ \alpha \sharp* (subject\ \alpha)$; $distinct(bn\ \alpha)]\!] \Longrightarrow \Psi \rhd (\!|\nu x|\!)P \longmapsto \alpha \prec ((\!|\nu x|\!)P')$
| *Bang*: $[\![\Psi \rhd P \parallel !P \longmapsto Rs$; *guarded* $P]\!] \Longrightarrow \Psi \rhd !P \longmapsto Rs$

abbreviation
semanticsBottomJudge (‹- ⟼ -› [*50*, *50*] *50*) **where** $P \longmapsto Rs \equiv \mathbf{1} \rhd P \longmapsto Rs$

equivariance *env.semantics*

nominal-inductive2 *env.semantics*
avoids *cInput*: *set xvec*
| *cPar1*: *set* $A_Q \cup$ *set*(*bn* α)
| *cPar2*: *set* $A_P \cup$ *set*(*bn* α)
| *cComm1*: *set* $A_P \cup$ *set* $A_Q \cup$ *set xvec*
| *cComm2*: *set* $A_P \cup$ *set* $A_Q \cup$ *set xvec*
| *cOpen*: $\{x\} \cup$ *set xvec* $\cup$ *set yvec*
| *cScope*: $\{x\} \cup$ *set*(*bn* α)
apply(*auto intro*: *substTerm.subst4Chain subst4Chain simp add*: *abs-fresh residualFresh*)
apply(*force simp add*: *fresh-star-def abs-fresh*)
apply(*simp add*: *boundOutputFresh*)
apply(*simp add*: *boundOutputFreshSet*)
apply(*simp add*: *boundOutputFreshSet*)
by(*simp add*: *fresh-star-def abs-fresh*)

lemma *nilTrans*[*dest*]:
fixes Ψ :: *'b*
and *Rs* :: (*'a*, *'b*, *'c*) *residual*
and *M* :: *'a*
and *xvec* :: *name list*
and *N* :: *'a*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *K* :: *'a*
and *yvec* :: *name list*

and N' :: $'a$
and P' :: $('a,\ 'b,\ 'c)\ psi$
and CsP :: $('c \times\ ('a,\ 'b,\ 'c)\ psi)\ list$
and Ψ' :: $'b$

shows $\Psi \rhd \mathbf{0} \longmapsto Rs \Longrightarrow False$
and $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto K(\!|\nu{*}yvec|\!)\langle N'\rangle \prec P' \Longrightarrow False$
and $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto \tau \prec P' \Longrightarrow False$
and $\Psi \rhd M\langle N\rangle.P \longmapsto K(\!|N'|\!) \prec P' \Longrightarrow False$
and $\Psi \rhd M\langle N\rangle.P \longmapsto \tau \prec P' \Longrightarrow False$
and $\Psi \rhd \{\!|\Psi'|\!\} \longmapsto Rs \Longrightarrow False$
apply(*cases rule*: *semantics.cases*) **apply** *auto*
apply(*cases rule*: *semantics.cases*) **apply**(*auto simp add*: *residualInject*)
apply(*cases rule*: *semantics.cases*) **apply**(*auto simp add*: *residualInject*)
apply(*cases rule*: *semantics.cases*) **apply**(*auto simp add*: *residualInject*)
apply(*cases rule*: *semantics.cases*) **apply**(*auto simp add*: *residualInject*)
by(*cases rule*: *semantics.cases*) (*auto simp add*: *residualInject*)

lemma *residualEq*:
fixes α :: $'a\ action$
and P :: $('a,\ 'b,\ 'c)\ psi$
and β :: $'a\ action$
and Q :: $('a,\ 'b,\ 'c)\ psi$

assumes $\alpha \prec P = \beta \prec Q$
and $bn\ \alpha \sharp* (bn\ \beta)$
and $distinct(bn\ \alpha)$
and $distinct(bn\ \beta)$
and $bn\ \alpha \sharp* (\alpha \prec P)$
and $bn\ \beta \sharp* (\beta \prec Q)$

obtains p **where** $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** $distinctPerm\ p$ **and** $\beta = p \cdot \alpha$ **and** $Q = p \cdot P$ **and** $bn\ \alpha \sharp* \beta$ **and** $bn\ \alpha \sharp* Q$ **and** $bn(p \cdot \alpha) \sharp* \alpha$ **and** $bn(p \cdot \alpha) \sharp* P$
using *assms*
proof(*nominal-induct* α *rule*: *action.strong-induct*)
case(*In M N*)
thus *?case* **by**(*simp add*: *residualInject*)
next
case(*Out M xvec N*)
thus *?case*
by(*auto simp add*: *residualInject*)
(*drule-tac boundOutputChainEq''*, *auto*)
next
case *Tau*
thus *?case* **by**(*simp add*: *residualInject*)
qed

lemma *semanticsInduct*[*consumes 3*, *case-names cAlpha cInput cOutput cCase*

cPar1 cPar2 cComm1 cComm2 cOpen cScope cBang]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and *Prop* :: $'d$::*fs-name* $\Rightarrow$ $'b$ $\Rightarrow$ $('a, 'b, 'c)$ *psi* $\Rightarrow$
$'a$ *action* $\Rightarrow$ $('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool*
and C :: $'d$::*fs-name*

assumes $\Psi \rhd P \longmapsto \alpha \prec P'$
and $bn\ \alpha \sharp* (subject\ \alpha)$
and $distinct(bn\ \alpha)$
and *rAlpha*: $\bigwedge \Psi\ P\ \alpha\ P'\ p\ C.\ [\![bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* (subject\ \alpha);$
$bn\ \alpha \sharp* C;\ bn\ \alpha \sharp* (bn(p \cdot \alpha));$
$set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha));\ distinctPerm\ p;$
$(bn(p \cdot \alpha)) \sharp* \alpha;\ (bn(p \cdot \alpha)) \sharp* P';\ Prop\ C\ \Psi\ P\ \alpha\ P'] \Longrightarrow$
$Prop\ C\ \Psi\ P\ (p \cdot \alpha)\ (p \cdot P')$
and *rInput*: $\bigwedge \Psi\ M\ K\ xvec\ N\ Tvec\ P\ C.$
$[\![\Psi \vdash M \leftrightarrow K;\ distinct\ xvec;\ set\ xvec \subseteq supp\ N;$
$length\ xvec = length\ Tvec;\ xvec \sharp* \Psi;$
$xvec \sharp* M;\ xvec \sharp* K;\ xvec \sharp* C] \Longrightarrow$
$Prop\ C\ \Psi\ (M(\!|\lambda* xvec\ N|\!).P)$
$(K(\!|(N[xvec::=Tvec])|\!))\ (P[xvec::=Tvec])$
and *rOutput*: $\bigwedge \Psi\ M\ K\ N\ P\ C.\ [\![\Psi \vdash M \leftrightarrow K]\!] \Longrightarrow Prop\ C\ \Psi\ (M\langle N\rangle.P)$
$(K\langle N\rangle)\ P$
and *rCase*: $\bigwedge \Psi\ P\ \alpha\ P'\ \varphi\ Cs\ C.\ [\![\Psi \rhd P \longmapsto \alpha \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ P\ \alpha\ P';$
$(\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P] \Longrightarrow$
$Prop\ C\ \Psi\ (Cases\ Cs)\ \alpha\ P'$
and *rPar1*: $\bigwedge \Psi\ \Psi_Q\ P\ \alpha\ P'\ A_Q\ Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct$
$A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ \alpha\ P';$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* \alpha;\ A_Q \sharp* P';\ A_Q \sharp* C;$
$distinct(bn\ \alpha);\ bn\ \alpha \sharp* Q;$
$bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* \Psi_Q;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* C]$
$\Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ \alpha\ (P' \parallel Q)$
and *rPar2*: $\bigwedge \Psi\ \Psi_P\ Q\ \alpha\ Q'\ A_P\ P\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct$
$A_P;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ \alpha\ Q';$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* \alpha;\ A_P \sharp* Q';\ A_P \sharp* C;$
$distinct(bn\ \alpha);\ bn\ \alpha \sharp* Q;$
$bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* \Psi_P;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* C]$
$\Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ \alpha\ (P \parallel Q')$
and *rComm1*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ \bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|N|\!))$

P';
$extractFrame\ P = \langle A_P,\ \Psi_P\rangle$; $distinct\ A_P$;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$; $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|\nu{*}xvec|\!)\langle N\rangle)\ Q'$;
$extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$; $distinct\ A_Q$;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$;
$A_P \sharp{*}\ \Psi$; $A_P \sharp{*}\ \Psi_Q$; $A_P \sharp{*}\ P$; $A_P \sharp{*}\ M$; $A_P \sharp{*}\ N$; $A_P \sharp{*}\ P'$;
$A_P \sharp{*}\ Q$; $A_P \sharp{*}\ Q'$; $A_P \sharp{*}\ A_Q$; $A_P \sharp{*}\ xvec$; $A_Q \sharp{*}\ \Psi$; $A_Q \sharp{*}\ \Psi_P$;
$A_Q \sharp{*}\ P$; $A_Q \sharp{*}\ N$; $A_Q \sharp{*}\ P'$; $A_Q \sharp{*}\ Q$; $A_Q \sharp{*}\ K$; $A_Q \sharp{*}\ Q'$; $distinct\ xvec$;
$A_Q \sharp{*}\ xvec$; $xvec \sharp{*}\ \Psi$; $xvec \sharp{*}\ \Psi_P$; $xvec \sharp{*}\ \Psi_Q$; $xvec \sharp{*}\ P$; $xvec \sharp{*}\ M$;
$xvec \sharp{*}\ Q$; $xvec \sharp{*}\ K$; $A_P \sharp{*}\ C$; $A_Q \sharp{*}\ C$; $xvec \sharp{*}\ C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (\tau)\ ((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

and *rComm2*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|\nu{*}xvec|\!)\langle N\rangle)\ P'$;
$extractFrame\ P = \langle A_P,\ \Psi_P\rangle$; $distinct\ A_P$;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|N|\!))\ Q'$;
$extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$; $distinct\ A_Q$;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$;
$A_P \sharp{*}\ \Psi$; $A_P \sharp{*}\ \Psi_Q$; $A_P \sharp{*}\ P$; $A_P \sharp{*}\ M$; $A_P \sharp{*}\ N$; $A_P \sharp{*}\ P'$;
$A_P \sharp{*}\ Q$; $A_P \sharp{*}\ Q'$; $A_P \sharp{*}\ A_Q$; $A_P \sharp{*}\ xvec$; $A_Q \sharp{*}\ \Psi$; $A_Q \sharp{*}\ \Psi_P$;
$A_Q \sharp{*}\ P$; $A_Q \sharp{*}\ N$; $A_Q \sharp{*}\ P'$; $A_Q \sharp{*}\ Q$; $A_Q \sharp{*}\ K$; $A_Q \sharp{*}\ Q'$; $distinct\ xvec$;
$A_Q \sharp{*}\ xvec$; $xvec \sharp{*}\ \Psi$; $xvec \sharp{*}\ \Psi_P$; $xvec \sharp{*}\ \Psi_Q$; $xvec \sharp{*}\ P$; $xvec \sharp{*}\ M$;
$xvec \sharp{*}\ Q$; $xvec \sharp{*}\ K$; $A_P \sharp{*}\ C$; $A_Q \sharp{*}\ C$; $xvec \sharp{*}\ C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (\tau)\ ((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

and *rOpen*: $\bigwedge \Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ C.$
$[\![\Psi \rhd P \longmapsto M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P'$; $x \in supp\ N$; $\bigwedge C.\ Prop\ C\ \Psi\ P\ (M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle)\ P'$;
$x \sharp \Psi$; $x \sharp M$; $x \sharp xvec$; $x \sharp yvec$; $xvec \sharp{*}\ \Psi$; $xvec \sharp{*}\ P$; $xvec \sharp{*}\ M$; $distinct\ xvec$; $distinct\ yvec$;
$yvec \sharp{*}\ \Psi$; $yvec \sharp{*}\ P$; $yvec \sharp{*}\ M$; $yvec \sharp{*}\ C$; $x \sharp C$; $xvec \sharp{*}\ C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (M(\!|\nu{*}(xvec@x\#yvec)|\!)\langle N\rangle)\ P'$

and *rScope*: $\bigwedge \Psi\ P\ \alpha\ P'\ x\ C.$
$[\![\Psi \rhd P \longmapsto \alpha \prec P'$; $\bigwedge C.\ Prop\ C\ \Psi\ P\ \alpha\ P'$;
$x \sharp \Psi$; $x \sharp \alpha$; $bn\ \alpha \sharp{*}\ \Psi$;
$bn\ \alpha \sharp{*}\ P$; $bn\ \alpha \sharp{*}\ (subject\ \alpha)$; $x \sharp C$; $bn\ \alpha \sharp{*}\ C$; $distinct(bn\ \alpha)]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ \alpha\ ((\!|\nu x|\!)P')$

and *rBang*: $\bigwedge \Psi\ P\ \alpha\ P'\ C.$
$[\![\Psi \rhd P \parallel !P \longmapsto \alpha \prec P'$; $guarded\ P$; $\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ \alpha\ P']\!] \Longrightarrow$
$Prop\ C\ \Psi\ (!P)\ \alpha\ P'$

shows $Prop\ C\ \Psi\ P\ \alpha\ P'$

using ‹$\Psi \rhd P \longmapsto\alpha \prec P'$› ‹$bn\ \alpha \sharp* (subject\ \alpha)$› ‹$distinct(bn\ \alpha)$›
proof(*nominal-induct x3==*$\alpha \prec P'$ *avoiding*: α C *arbitrary*: P' *rule*: *semantics.strong-induct*)
  case(*cInput* Ψ M K *xvec* N *Tvec* P α C P')
  thus *?case* **by**(*force intro*: *rInput simp add*: *residualInject*)
next
  case(*Output* Ψ M K N P α C P')
  thus *?case* **by**(*force intro*: *rOutput simp add*: *residualInject*)
next
  case(*Case* Ψ P *Rs* φ *Cs* α C)
  thus *?case* **by**(*auto intro*: *rCase*)
next
  case(*cPar1* Ψ Ψ_Q P α P' Q A_Q α' C P'')
  note ‹$\alpha \prec (P' \parallel Q) = \alpha' \prec P''$›
  moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* (bn\ \alpha')$ **by** *auto*
  moreover note ‹$distinct\ (bn\ \alpha)$› ‹$distinct(bn\ \alpha')$›
  moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha' \sharp* subject\ \alpha'$›
  have $bn\ \alpha \sharp* (\alpha \prec P' \parallel Q)$ **and** $bn\ \alpha' \sharp* (\alpha' \prec P'')$ **by** *simp+*
  ultimately obtain p **where** S: $(set\ p) \subseteq (set(bn\ \alpha)) \times (set(bn(p \cdot \alpha)))$ **and** *distinctPerm* p
    and αEq: $\alpha' = p \cdot \alpha$ **and** $P'eq$: $P'' = p \cdot (P' \parallel Q)$ **and** $(bn(p \cdot \alpha)) \sharp* \alpha$
    and $(bn(p \cdot \alpha)) \sharp* (P' \parallel Q)$
    by(*rule residualEq*)

  note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$› ‹$extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$› ‹$distinct\ A_Q$›
  moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
  have $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ \alpha\ P'$ **by**(*rule-tac cPar1*) *auto*
  moreover note ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \alpha$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* C$›
    ‹$bn\ \alpha \sharp* Q$› ‹$distinct(bn\ \alpha)$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_Q$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* C$›
  ultimately have $Prop\ C\ \Psi\ (P \parallel Q)\ \alpha\ (P' \parallel Q)$
    by(*rule-tac rPar1*)

  with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* bn\ \alpha'$› S ‹$distinctPerm\ p$› ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn(p \cdot \alpha) \sharp* (P' \parallel Q)$› ‹$A_Q \sharp* C$›
  have $Prop\ C\ \Psi\ (P \parallel Q)\ (p \cdot \alpha)\ (p \cdot (P' \parallel Q))$
    by(*rule-tac rAlpha*) *auto*
  with αEq $P'eq$ ‹$distinctPerm\ p$› **show** *?case* **by** *simp*
next
  case(*cPar2* Ψ Ψ_P Q α Q' P A_P α' C Q'')
  note ‹$\alpha \prec (P \parallel Q') = \alpha' \prec Q''$›
  moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* (bn\ \alpha')$ **by** *auto*
  moreover note ‹$distinct\ (bn\ \alpha)$› ‹$distinct(bn\ \alpha')$›
  moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha' \sharp* subject\ \alpha'$›
  have $bn\ \alpha \sharp* (\alpha \prec P \parallel Q')$ **and** $bn\ \alpha' \sharp* (\alpha' \prec Q'')$ **by** *simp+*
  ultimately obtain p **where** S: $(set\ p) \subseteq (set(bn\ \alpha)) \times (set(bn(p \cdot \alpha)))$ **and** *distinctPerm* p

and αEq: $\alpha' = p \cdot \alpha$ **and** $Q'eq$: $Q'' = p \cdot (P \parallel Q')$ **and** $(bn(p \cdot \alpha)) \sharp* \alpha$
and $(bn(p \cdot \alpha)) \sharp* (P \parallel Q')$
by(*rule residualEq*)

note ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$› ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› ‹*distinct* A_P›
moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
have $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ \alpha\ Q'$ **by**(*rule-tac cPar2*) *auto*

moreover note ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* Q'$› ‹$A_P \sharp* C$›
‹$bn\ \alpha \sharp* Q$› ‹$distinct(bn\ \alpha)$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_P$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* C$›
ultimately have $Prop\ C\ \Psi\ (P \parallel Q)\ \alpha\ (P \parallel Q')$
by(*rule-tac rPar2*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* (bn\ \alpha')$› S ‹$distinctPerm\ p$› ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn(p \cdot \alpha) \sharp* (P \parallel Q')$›
have $Prop\ C\ \Psi\ (P \parallel Q)\ (p \cdot \alpha)\ (p \cdot (P \parallel Q'))$
by(*rule-tac rAlpha*) *auto*
with $\alpha Eq\ Q'eq$ ‹$distinctPerm\ p$› **show** *?case* **by** *simp*
next
case($cComm1\ \Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q\ \alpha\ C\ P''$)
hence $Prop\ C\ \Psi\ (P \parallel Q)\ (\tau)\ ((\!|\nu*xvec|\!)(P' \parallel Q'))$
by(*rule-tac rComm1*) (*assumption* | *simp*)+
thus *?case* **using** ‹$\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q') = \alpha \prec P''$›
by(*simp add*: *residualInject*)
next
case($cComm2\ \Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ \alpha\ C\ P''$)
hence $Prop\ C\ \Psi\ (P \parallel Q)\ (\tau)\ ((\!|\nu*xvec|\!)(P' \parallel Q'))$
by(*rule-tac rComm2*) (*assumption* | *simp*)+
thus *?case* **using** ‹$\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q') = \alpha \prec P''$›
by(*simp add*: *residualInject*)
next
case($cOpen\ \Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ \alpha\ C\ P''$)
note ‹$M(\!|\nu*(xvec@x\#yvec)|\!)\langle N \rangle \prec P' = \alpha \prec P''$›
moreover from ‹$xvec \sharp* \alpha$› ‹$x \sharp \alpha$› ‹$yvec \sharp* \alpha$› **have** $(xvec@x\#yvec) \sharp* (bn\ \alpha)$
by *auto*
moreover from ‹$xvec \sharp* yvec$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› ‹$distinct\ xvec$› ‹$distinct\ yvec$›
have $distinct(xvec@x\#yvec)$
by(*auto simp add*: *fresh-star-def*) (*simp add*: *fresh-def name-list-supp*)
moreover note ‹$distinct(bn\ \alpha)$›
moreover from ‹$xvec \sharp* M$› ‹$x \sharp M$› ‹$yvec \sharp* M$› **have** $(xvec@x\#yvec) \sharp* M$ **by** *auto*
hence $(xvec@x\#yvec) \sharp* (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N \rangle \prec P')$ **by** *auto*
moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› **have** $bn\ \alpha \sharp* (\alpha \prec P'')$ **by** *simp*
ultimately obtain p **where** S: $(set\ p) \subseteq (set(xvec@x\#yvec)) \times (set(p \cdot (xvec@x\#yvec)))$ **and** $distinctPerm\ p$
and αeq: $\alpha = (p \cdot M)(\!|\nu*(p \cdot (xvec@x\#yvec))|\!)\langle (p \cdot N) \rangle$ **and** $P'eq$: $P'' = (p \cdot P')$

and A: $(xvec@x\#yvec) \sharp* ((p \cdot M)(\!|\nu*(p \cdot (xvec@x\#yvec))|\!)\langle(p \cdot N)\rangle)$
and B: $(p \cdot (xvec@x\#yvec)) \sharp* (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle)$
and C: $(p \cdot (xvec@x\#yvec)) \sharp* P'$
by(*rule-tac residualEq*) (*assumption* | *simp*)+

note ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$› ‹$x \in (supp\ N)$›

moreover {
fix C
from ‹$xvec \sharp* M$› ‹$yvec \sharp* M$› **have** $(xvec@yvec) \sharp* M$ **by** *simp*
moreover from ‹*distinct xvec*› ‹*distinct yvec*› ‹$xvec \sharp* yvec$› **have** $distinct(xvec@yvec)$
by *auto* (*simp add*: *fresh-star-def name-list-supp fresh-def*)
ultimately have $Prop\ C\ \Psi\ P\ (M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle)\ P'$ **by**(*rule-tac cOpen*) *auto*
}

moreover note ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$xvec \sharp* M$›
‹$yvec \sharp* \Psi$› ‹$yvec \sharp* P$› ‹$yvec \sharp* M$› ‹$yvec \sharp* C$› ‹$x \sharp C$› ‹$xvec \sharp* C$› ‹*distinct xvec*› ‹*distinct yvec*›
ultimately have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle)\ P'$
by(*rule-tac rOpen*)

with ‹$xvec \sharp* \Psi$› ‹$yvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$yvec \sharp* P$› ‹$xvec \sharp* M$› ‹$yvec \sharp* M$›
‹$yvec \sharp* C$› S ‹*distinctPerm p*› ‹$x \sharp C$› ‹$xvec \sharp* C$›
‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› $A\ B\ C$
have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (p \cdot (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle))\ (p \cdot P')$
apply(*rule-tac* $\alpha{=}M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$ **in** *rAlpha*)
apply(*assumption* | *simp*)+
apply(*fastforce simp add*: *fresh-star-def abs-fresh*)
by(*assumption* | *simp*)+
with $\alpha eq\ P'eq$ **show** *?case* **by** *simp*
next
case($cScope\ \Psi\ P\ \alpha\ P'\ x\ \alpha'\ C\ P''$)
note ‹$\alpha \prec ((\!|\nu x|\!)P') = \alpha' \prec P''$›
moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* (bn\ \alpha')$ **by** *auto*
moreover note ‹$distinct\ (bn\ \alpha)$› ‹$distinct(bn\ \alpha')$›
moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha' \sharp* subject\ \alpha'$›
have $bn\ \alpha \sharp* (\alpha \prec (\!|\nu x|\!)P')$ **and** $bn\ \alpha' \sharp* (\alpha' \prec P'')$ **by** *simp*+
ultimately obtain p **where** S: $(set\ p) \subseteq (set(bn\ \alpha)) \times (set(bn(p \cdot \alpha)))$ **and** *distinctPerm p*
and αEq: $\alpha' = p \cdot \alpha$ **and** $P'eq$: $P'' = p \cdot ((\!|\nu x|\!)P')$ **and** $(bn(p \cdot \alpha)) \sharp* \alpha$
and $(bn(p \cdot \alpha)) \sharp* ((\!|\nu x|\!)P')$
by(*rule residualEq*)

note ‹$\Psi \rhd P \longmapsto \alpha \prec P'$›
moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
have $\bigwedge C.\ Prop\ C\ \Psi\ P\ \alpha\ P'$ **by**(*rule-tac cScope*) *auto*

moreover note ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*\ subject\ \alpha$›
‹$x \sharp C$› ‹$bn\ \alpha \sharp* C$› ‹$distinct(bn\ \alpha)$›
ultimately have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ \alpha\ ((\!|\nu x|\!)P')$
by(*rule rScope*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp*\ subject\ \alpha$› ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* (bn\ \alpha')$› S ‹$distinctPerm\ p$› ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn(p \cdot \alpha) \sharp* ((\!|\nu x|\!)P')$›
have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (p \cdot \alpha)\ (p \cdot ((\!|\nu x|\!)P'))$
by(*rule-tac rAlpha*) *simp+*
with $\alpha Eq\ P'eq$ ‹$distinctPerm\ p$› **show** *?case* **by** *simp*
next
case(*Bang* $\Psi\ P\ Rs\ \alpha\ C$)
thus *?case* **by**(*rule-tac rBang*) *auto*
qed

lemma *outputInduct*[*consumes 1*, *case-names cOutput cCase cPar1 cPar2 cOpen cScope cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and M :: $'a$
and B :: $('a, 'b, 'c)\ boundOutput$
and $Prop$ :: $'d::fs\text{-}name \Rightarrow 'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow$
$'a \Rightarrow ('a, 'b, 'c)\ boundOutput \Rightarrow bool$
and C :: $'d::fs\text{-}name$

assumes $\Psi \rhd P \longmapsto ROut\ M\ B$
and *rOutput*: $\bigwedge \Psi\ M\ K\ N\ P\ C.\ [\![\Psi \vdash M \leftrightarrow K]\!] \Longrightarrow Prop\ C\ \Psi\ (M\langle N\rangle.P)\ K\ (N \prec' P)$
and *rCase*: $\bigwedge \Psi\ P\ M\ B\ \varphi\ Cs\ C.$
$[\![\Psi \rhd P \longmapsto (ROut\ M\ B);\ \bigwedge C.\ Prop\ C\ \Psi\ P\ M\ B;\ (\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (Cases\ Cs)\ M\ B$
and *rPar1*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_Q\ Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ M\ ((\!|\nu{*}xvec|\!)N \prec' P');$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* M;$
$A_Q \sharp* xvec;\ A_Q \sharp* N;\ A_Q \sharp* P';\ A_Q \sharp* C;\ xvec \sharp* Q;$
$xvec \sharp* \Psi;\ xvec \sharp* \Psi_Q;\ xvec \sharp* P;\ xvec \sharp* M;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ ((\!|\nu{*}xvec|\!)N \prec' (P' \parallel Q))$
and *rPar2*: $\bigwedge \Psi\ \Psi_P\ Q\ M\ xvec\ N\ Q'\ A_P\ P\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ M\ ((\!|\nu{*}xvec|\!)N \prec' Q');$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* M;$
$A_P \sharp* xvec;\ A_P \sharp* N;\ A_P \sharp* Q';\ A_P \sharp* C;\ xvec \sharp* P;$
$xvec \sharp* \Psi;\ xvec \sharp* \Psi_P;\ xvec \sharp* Q;\ xvec \sharp* M;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ ((\!|\nu{*}xvec|\!)N \prec' (P \parallel Q'))$
and *rOpen*: $\bigwedge \Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ C.$

$[\![\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$; $x \in supp\ N$; $\bigwedge C.\ Prop\ C\ \Psi\ P\ M\ ((\!|\nu*(xvec@yvec)|\!)N \prec' P')$;
$x \sharp \Psi$; $x \sharp M$; $x \sharp xvec$; $x \sharp yvec$; $xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$;
$xvec \sharp* yvec$; $yvec \sharp* \Psi$; $yvec \sharp* P$; $yvec \sharp* M$; $yvec \sharp* C$; $x \sharp C$; $xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ M\ ((\!|\nu*(xvec@x\#yvec)|\!)N \prec' P')$

and *rScope*: $\bigwedge \Psi\ P\ M\ xvec\ N\ P'\ x\ C.$
$[\![\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$; $\bigwedge C.\ Prop\ C\ \Psi\ P\ M\ ((\!|\nu*xvec|\!)N \prec' P')$;
$x \sharp \Psi$; $x \sharp M$; $x \sharp xvec$; $x \sharp N$; $xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$;
$x \sharp C$; $xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ M\ ((\!|\nu*xvec|\!)N \prec' (\!|\nu x|\!)P')$

and *rBang*: $\bigwedge \Psi\ P\ M\ B\ C.$
$[\![\Psi \rhd P \parallel !P \longmapsto (ROut\ M\ B)$; $guarded\ P$; $\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ M\ B]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (!P)\ M\ B$

shows $Prop\ C\ \Psi\ P\ M\ B$
using ‹$\Psi \rhd P \longmapsto (ROut\ M\ B)$›
proof(*nominal-induct* $\Psi\ P\ Rs$==$(ROut\ M\ B)$ *avoiding*: C *arbitrary*: B *rule*: *semantics.strong-induct*)
  case(*cInput* $\Psi\ M\ K\ xvec\ N\ Tvec\ P\ C$)
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*Output* $\Psi\ M\ K\ N\ P\ C$)
  thus *?case* **by**(*force simp add*: *residualInject intro*: *rOutput*)
next
  case(*Case* $\Psi\ P\ Rs\ \varphi\ Cs\ C$)
  thus *?case* **by**(*force intro*: *rCase*)
next
  case(*cPar1* $\Psi\ \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q\ C$)
  thus *?case* **by**(*force intro*: *rPar1 simp add*: *residualInject*)
next
  case(*cPar2* $\Psi\ \Psi_P\ Q\ \alpha\ Q'\ P\ A_P\ C$)
  thus *?case* **by**(*force intro*: *rPar2 simp add*: *residualInject*)
next
  case *cComm1*
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case *cComm2*
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*cOpen* $\Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ C\ B$)
  thus *?case* **by**(*force intro*: *rOpen simp add*: *residualInject*)
next
  case(*cScope* $\Psi\ P\ M\ \alpha\ P'\ x\ C$)
  thus *?case* **by**(*force intro*: *rScope simp add*: *residualInject*)
next
  case(*Bang* $\Psi\ P\ Rs\ C$)
  thus *?case* **by**(*force intro*: *rBang*)

```
qed

lemma boundOutputBindObject:
  fixes Ψ   :: 'b
  and   P   :: ('a, 'b, 'c) psi
  and   M   :: 'a
  and   yvec :: name list
  and   N   :: 'a
  and   P'  :: ('a, 'b, 'c) psi
  and   y   :: name

  assumes Ψ ▷ P ⟼α ≺ P'
  and     bn α ♯* subject α
  and     distinct(bn α)
  and     y ∈ set(bn α)

  shows y ∈ supp(object α)
using assms
proof(nominal-induct avoiding: P' arbitrary: y rule: semanticsInduct)
  case(cAlpha Ψ P α P' p P'' y)
  from ‹y ∈ set(bn(p • α))› have (p • y) ∈ (p • set(bn(p • α)))
    by(rule pt-set-bij2[OF pt-name-inst, OF at-name-inst])
  hence (p • y) ∈ set(bn α) using ‹distinctPerm p›
    by(simp add: eqvts)
  hence (p • y) ∈ supp(object α) by(rule cAlpha)
  hence (p • p • y) ∈ (p • supp(object α))
    by(rule pt-set-bij2[OF pt-name-inst, OF at-name-inst])
  thus ?case using ‹distinctPerm p›
    by(simp add: eqvts)
next
  case cInput
  thus ?case by(simp add: supp-list-nil)
next
  case cOutput
  thus ?case by(simp add: supp-list-nil)
next
  case cCase
  thus ?case by simp
next
  case cPar1
  thus ?case by simp
next
  case cPar2
  thus ?case by simp
next
  case cComm1
  thus ?case by(simp add: supp-list-nil)
next
  case cComm2
```

thus *?case* **by**(*simp add*: *supp-list-nil*)
next
case *cOpen*
thus *?case* **by**(*auto simp add*: *supp-list-cons supp-list-append supp-atm supp-some*)
next
case *cScope*
thus *?case* **by** *simp*
next
case *cBang*
thus *?case* **by** *simp*
qed

lemma *alphaBoundOutputChain'*:
fixes *yvec* :: *name list*
and *xvec* :: *name list*
and *B* :: ('*a*, '*b*, '*c*) *boundOutput*

assumes *length xvec* = *length yvec*
and *yvec* $\sharp*$ *B*
and *yvec* $\sharp*$ *xvec*
and *distinct yvec*

shows $(\!|\nu*xvec|\!)B = (\!|\nu*yvec|\!)([xvec\ yvec] \cdot_v B)$
using *assms*
proof(*induct rule*: *composePermInduct*)
case *cBase*
show *?case* **by** *simp*
next
case(*cStep x xvec y yvec*)
thus *?case*
apply *auto*
by(*subst alphaBoundOutput*[*of y*]) (*auto simp add*: *eqvts*)
qed

lemma *alphaBoundOutputChain''*:
fixes *yvec* :: *name list*
and *xvec* :: *name list*
and *N* :: '*a*
and *P* :: ('*a*, '*b*, '*c*) *psi*

assumes *length xvec* = *length yvec*
and *yvec* $\sharp*$ *N*
and *yvec* $\sharp*$ *P*
and *yvec* $\sharp*$ *xvec*
and *distinct yvec*

shows $(\!|\nu*xvec|\!)(N \prec' P) = (\!|\nu*yvec|\!)(([xvec\ yvec] \cdot_v N) \prec' ([xvec\ yvec] \cdot_v P))$
proof −
from *assms* **have** $(\!|\nu*xvec|\!)(N \prec' P) = (\!|\nu*yvec|\!)([xvec\ yvec] \cdot_v (N \prec' P))$

by(*simp add*: *alphaBoundOutputChain'*)
thus *?thesis* **by** *simp*
qed

lemma *alphaDistinct*:
fixes *xvec* :: *name list*
and N :: $'a$
and P :: $('a, 'b, 'c)$ *psi*
and *yvec* :: *name list*
and M :: $'a$
and Q :: $('a, 'b, 'c)$ *psi*

assumes $\alpha \prec P = \beta \prec Q$
and $distinct(bn\ \alpha)$
and $\bigwedge x.\ x \in set(bn\ \alpha) \Longrightarrow x \in supp(object\ \alpha)$
and $bn\ \alpha \sharp* bn\ \beta$
and $bn\ \alpha \sharp* (object\ \beta)$
and $bn\ \alpha \sharp* Q$

shows $distinct(bn\ \beta)$
using *assms*
proof(*rule-tac actionCases*[**where** α=α], *auto simp add*: *residualInject supp-some*)
fix *xvec M yvec N*
assume *Eq*: $(\!|\nu*xvec|\!)N \prec' P = (\!|\nu*yvec|\!)M \prec' Q$
assume *distinct xvec* **and** $xvec \sharp* M$ **and** $xvec \sharp* yvec$ **and** $xvec \sharp* Q$
assume *Mem*: $\bigwedge x.\ x \in set\ xvec \Longrightarrow x \in (supp\ N)$
show *distinct yvec*
proof −
from *Eq* **have** *length xvec* = *length yvec*
by(*rule boundOutputChainEqLength*)
with *Eq* ‹*distinct xvec*› ‹$xvec \sharp* yvec$› ‹$xvec \sharp* M$› ‹$xvec \sharp* Q$› *Mem* **show** *?thesis*
proof(*induct n*==*length xvec arbitrary*: *xvec yvec M Q rule*: *nat.induct*)
case(*zero xvec yvec M Q*)
thus *?case* **by** *simp*
next
case(*Suc n xvec yvec M Q*)
have *L*: *length xvec* = *length yvec* **and** *Suc n* = *length xvec* **by** *fact*+
then obtain *x xvec' y yvec'* **where** *xEq*: *xvec* = *x*#*xvec'* **and** *yEq*: *yvec* = *y*#*yvec'*
and *L'*: *length xvec'* = *length yvec'*
by(*cases xvec*, *auto*, *cases yvec*, *auto*)
have *xvecFreshyvec*: $xvec \sharp* yvec$ **and** *xvecDist*: *distinct xvec* **by** *fact*+
with *xEq yEq* **have** *xineqy*: $x \neq y$ **and** $xvec'Freshyvec'$: $xvec' \sharp* yvec'$
and *xvec'Dist*: *distinct xvec'* **and** $xFreshxvec'$: $x \sharp xvec'$
and $xFreshyvec'$: $x \sharp yvec'$ **and** $yFreshxvec'$: $y \sharp xvec'$
by *auto*
have *Eq*: $(\!|\nu*xvec|\!)N \prec' P = (\!|\nu*yvec|\!)M \prec' Q$ **by** *fact*
with *xEq yEq xineqy* **have** *Eq'*: $(\!|\nu*xvec'|\!)N \prec' P = (\!|\nu*([(x, y)] \cdot yvec')|\!)([(x, y)] \cdot M) \prec' ([(x, y)] \cdot Q)$

```
      by(simp add: boundOutput.inject alpha eqvts)
    moreover have Mem:⋀x. x ∈ set xvec ⟹ x ∈ supp N by fact
    with xEq have ⋀x. x ∈ set xvec' ⟹ x ∈ supp N by simp
    moreover have xvec ♯* M by fact
    with xEq xFreshxvec' yFreshxvec' have xvec' ♯* ([(x, y)] • M) by simp
    moreover have xvecFreshQ: xvec ♯* Q by fact
    with xEq xFreshxvec' yFreshxvec' have xvec' ♯* ([(x, y)] • Q) by simp
    moreover have Suc n = length xvec by fact
    with xEq have n = length xvec' by simp
    moreover from xvec'Freshyvec' xFreshxvec' yFreshxvec' have xvec' ♯* ([(x, y)]
• yvec')
      by simp
    moreover from L' have length xvec' = length([(x, y)] • yvec') by simp
    ultimately have distinct([(x, y)] • yvec') using xvec'Dist
      by(rule-tac Suc) (assumption | simp)+
    hence distinct yvec' by simp
    from Mem xEq have xSuppN: x ∈ supp N by simp
    from L ‹distinct xvec› ‹xvec ♯* yvec› ‹xvec ♯* M› ‹xvec ♯* Q›
    have ⦇ν*yvec⦈M ≺' Q = ⦇ν*xvec⦈([yvec xvec] •v M) ≺' ([yvec xvec] •v Q)
      by(simp add: alphaBoundOutputChain'')
    with Eq have N = [yvec xvec] •v M by simp
    with xEq yEq have N = [(y, x)] • [yvec' xvec'] •v M
      by simp
    with xSuppN have ySuppM: y ∈ supp([yvec' xvec'] •v M)
      by(drule-tac pi=[(x, y)] in pt-set-bij2[OF pt-name-inst, OF at-name-inst])
        (simp add: calc-atm eqvts name-swap)
    have y ♯ yvec'
    proof(simp add: fresh-def, rule notI)
      assume y ∈ supp yvec'
      hence y mem yvec'
        by(induct yvec') (auto simp add: supp-list-nil supp-list-cons supp-atm)
      moreover from ‹xvec ♯* M› xEq xFreshxvec' have xvec' ♯* M by simp
      ultimately have y ♯ [yvec' xvec'] •v  M using L' xvec'Freshyvec' xvec'Dist
        by(force intro: freshChainPerm)
      with ySuppM show False by(simp add: fresh-def)
    qed
    with ‹distinct yvec'›  yEq show ?case by simp
  qed
qed
qed

lemma boundOutputDistinct:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   α    :: 'a action
  and   P'   :: ('a, 'b, 'c) psi

  assumes Ψ ▷ P ⟼α ≺ P'
```

shows *distinct*(*bn* α)
using *assms*
proof(*nominal-induct* Ψ *P x3*==$\alpha \prec P'$ *avoiding*: α P' *rule*: *semantics.strong-induct*)
case *cInput*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *Output*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *Case*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *cPar1*
thus *?case* **by**(*force intro*: *alphaDistinct boundOutputBindObject*)
next
case *cPar2*
thus *?case* **by**(*force intro*: *alphaDistinct boundOutputBindObject*)
next
case *cComm1*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *cComm2*
thus *?case* **by**(*simp add*: *residualInject*)
next
case(*cOpen* Ψ *P M xvec yvec N* P' *x* α P'')
note ‹$M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P' = \alpha \prec P''$›
moreover from ‹*xvec* $\sharp*$ *yvec*› ‹*x* $\sharp$ *xvec*› ‹*x* $\sharp$ *yvec*› ‹*distinct xvec*› ‹*distinct yvec*›
have *distinct*(*bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$))
by *auto* (*simp add*: *fresh-star-def fresh-def name-list-supp*)
moreover {
fix *y*
from ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$› ‹$x \in$ *supp N*› ‹*x* $\sharp$ *xvec*› ‹*x* $\sharp$ *yvec*› ‹*x* $\sharp$ *M*› ‹*x* $\sharp$ Ψ› ‹*distinct xvec*› ‹*distinct yvec*› ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ *P*› ‹*xvec* $\sharp*$ *M*› ‹*xvec* $\sharp*$ *yvec*› ‹*yvec* $\sharp*$ Ψ› ‹*yvec* $\sharp*$ *P*› ‹*yvec* $\sharp*$ *M*›
have $\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$ **by**(*rule semantics.cOpen*)
moreover moreover from ‹*xvec* $\sharp*$ *M*› ‹*x* $\sharp$ *M*› ‹*yvec* $\sharp*$ *M*›
have *bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$) $\sharp*$ (*subject*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$))
by *simp*
moreover note ‹*distinct*(*bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$))›
moreover assume $y \in$ *set*(*bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$))

ultimately have $y \in$ *supp*(*object*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$))
by(*rule-tac boundOutputBindObject*)
}
moreover from ‹*xvec* $\sharp*$ α› ‹*x* $\sharp$ α› ‹*yvec* $\sharp*$ α›
have *bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$) $\sharp*$ *bn* α **and** *bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$) $\sharp*$ *object* α **by** *simp+*
moreover from ‹*xvec* $\sharp*$ P''› ‹*x* $\sharp$ P''› ‹*yvec* $\sharp*$ P''›
have *bn*($M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle$) $\sharp*$ P'' **by** *simp*

ultimately show *?case* **by**(*rule alphaDistinct*)
next
case *cScope*
thus *?case*
by(*rule-tac alphaDistinct*, *auto*) (*rule-tac boundOutputBindObject*, *auto*)
next
case *Bang*
thus *?case* **by** *simp*
qed

lemma *inputDistinct*:
fixes Ψ :: $'b$
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P :: $('a, 'b, 'c)$ *psi*
and *Rs* :: $('a, 'b, 'c)$ *residual*

assumes $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto Rs$

shows *distinct xvec*
using *assms*
by(*nominal-induct* Ψ $P{=}{=}M(\!|\lambda{*}xvec\ N|\!).P$ *Rs avoiding*: *xvec N P rule*: *semantics.strong-induct*)
(*auto simp add*: *psi.inject intro*: *alphaInputDistinct*)

lemma *outputInduct'*[*consumes 2*, *case-names cAlpha cOutput cCase cPar1 cPar2 cOpen cScope cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *yvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and *Prop* :: $'d$::*fs-name* $\Rightarrow$ $'b$ $\Rightarrow$ $('a, 'b, 'c)$ *psi* $\Rightarrow$
$'a$ $\Rightarrow$ *name list* $\Rightarrow$ $'a$ $\Rightarrow$ $('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool*
and C :: $'d$::*fs-name*

assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $xvec \sharp* M$
and *rAlpha*: $\bigwedge\Psi\ P\ M\ xvec\ N\ P'\ p\ C.\ [\![xvec \sharp* \Psi;\ xvec \sharp* P;\ xvec \sharp* M;\ xvec \sharp* C;\ xvec \sharp* (p \cdot xvec);$
$set\ p \subseteq set\ xvec \times set(p \cdot xvec);\ distinctPerm\ p;$
$(p \cdot xvec) \sharp* N;\ (p \cdot xvec) \sharp* P';\ Prop\ C\ \Psi\ P\ M\ xvec\ N\ P']\!] \Longrightarrow$
$Prop\ C\ \Psi\ P\ M\ (p \cdot xvec)\ (p \cdot N)\ (p \cdot P')$
and *rOutput*: $\bigwedge\Psi\ M\ K\ N\ P\ C.\ [\![\Psi \vdash M \leftrightarrow K]\!] \Longrightarrow Prop\ C\ \Psi\ (M\langle N\rangle.P)\ K\ ([])\ N\ P$
and *rCase*: $\bigwedge\Psi\ P\ M\ xvec\ N\ P'\ \varphi\ Cs\ C.\ [\![\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';$

$\bigwedge C.$ *Prop* C Ψ P M *xvec* N P'; (φ, P) *mem Cs*; $\Psi \vdash \varphi$; *guarded* P⟧ $\Longrightarrow$
Prop C Ψ (*Cases Cs*) M *xvec* N P'
and *rPar1*: $\bigwedge \Psi$ Ψ_Q P M *xvec* N P' A_Q Q C.
⟦$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\bigwedge C.$ *Prop* C $(\Psi \otimes \Psi_Q)$ P M *xvec* N P';
$A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* \Psi$; $A_Q \sharp* M$;
$A_Q \sharp*$ *xvec*; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* C$; *xvec* $\sharp* Q$;
xvec $\sharp* \Psi$; *xvec* $\sharp* \Psi_Q$; *xvec* $\sharp* P$; *xvec* $\sharp* M$; *xvec* $\sharp* C$⟧ $\Longrightarrow$
Prop C Ψ $(P \parallel Q)$ M *xvec* N $(P' \parallel Q)$
and *rPar2*: $\bigwedge \Psi$ Ψ_P Q M *xvec* N Q' A_P P C.
⟦$\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\bigwedge C.$ *Prop* C $(\Psi \otimes \Psi_P)$ Q M *xvec* N Q';
$A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* \Psi$; $A_P \sharp* M$;
$A_P \sharp*$ *xvec*; $A_P \sharp* N$; $A_P \sharp* Q'$; $A_P \sharp* C$; *xvec* $\sharp* Q$;
xvec $\sharp* \Psi$; *xvec* $\sharp* \Psi_P$; *xvec* $\sharp* P$; *xvec* $\sharp* M$; *xvec* $\sharp* C$⟧ $\Longrightarrow$
Prop C Ψ $(P \parallel Q)$ M *xvec* N $(P \parallel Q')$
and *rOpen*: $\bigwedge \Psi$ P M *xvec* *yvec* N P' x C.
⟦$\Psi \rhd P \longmapsto M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P'$; $x \in$ *supp* N; $\bigwedge C.$ *Prop* C Ψ P M $(xvec@yvec)$ N P';
$x \sharp \Psi$; $x \sharp M$; $x \sharp$ *xvec*; $x \sharp$ *yvec*; *xvec* $\sharp* \Psi$; *xvec* $\sharp* P$; *xvec* $\sharp* M$;
yvec $\sharp* \Psi$; *yvec* $\sharp* P$; *yvec* $\sharp* M$; *yvec* $\sharp* C$; $x \sharp C$; *xvec* $\sharp* C$⟧ $\Longrightarrow$
Prop C Ψ $((\!|\nu x|\!)P)$ M $(xvec@x\#yvec)$ N P'
and *rScope*: $\bigwedge \Psi$ P M *xvec* N P' x C.
⟦$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; $\bigwedge C.$ *Prop* C Ψ P M *xvec* N P';
$x \sharp \Psi$; $x \sharp M$; $x \sharp$ *xvec*; $x \sharp N$; *xvec* $\sharp* \Psi$;
xvec $\sharp* P$; *xvec* $\sharp* M$; $x \sharp C$; *xvec* $\sharp* C$⟧ $\Longrightarrow$
Prop C Ψ $((\!|\nu x|\!)P)$ M *xvec* N $((\!|\nu x|\!)P')$
and *rBang*: $\bigwedge \Psi$ P M *xvec* N P' C.
⟦$\Psi \rhd P \parallel !P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; *guarded* P; $\bigwedge C.$ *Prop* C Ψ $(P \parallel !P)$ M *xvec* N P'⟧ $\Longrightarrow$
Prop C Ψ $(!P)$ M *xvec* N P'
shows *Prop* C Ψ P M *xvec* N P'
proof −
note ‹$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$›
moreover from ‹*xvec* $\sharp* M$› **have** $bn(M(\!|\nu{*}xvec|\!)\langle N\rangle) \sharp*$ *subject*$(M(\!|\nu{*}xvec|\!)\langle N\rangle)$ **by** *simp*
moreover from ‹$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› **have** *distinct*$(bn(M(\!|\nu{*}xvec|\!)\langle N\rangle))$
by(*rule boundOutputDistinct*)
ultimately show *?thesis*
proof(*nominal-induct* Ψ P $\alpha{=}{=}M(\!|\nu{*}xvec|\!)\langle N\rangle$ P' *avoiding*: C *arbitrary*: M *xvec* N *rule*: *semanticsInduct*)
case(*cAlpha* Ψ P α P' p C M *xvec* N)
from ‹$(p \cdot \alpha) = M(\!|\nu{*}xvec|\!)\langle N\rangle$› **have** $(p \cdot p \cdot \alpha) = p \cdot (M(\!|\nu{*}xvec|\!)\langle N\rangle)$
by(*simp add*: *fresh-bij*)
with ‹*distinctPerm* p› **have** A: $\alpha = (p \cdot M)(\!|\nu{*}(p \cdot xvec)|\!)\langle(p \cdot N)\rangle$
by(*simp add*: *eqvts*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*$ *subject* α › ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* bn(p$

```
· α)› ‹distinctPerm p›
    have (p · xvec) ♯* Ψ and (p · xvec) ♯* P and (p · xvec) ♯* (p · M) and (p
· xvec) ♯* C and (p · xvec) ♯* (p · p · xvec)
      by auto
    moreover from A ‹set p ⊆ set(bn α) × set(bn(p · α))› ‹distinctPerm p›
    have S: set p ⊆ set(p · xvec) × set(p · p · xvec) by simp
    moreover note ‹distinctPerm p›
    moreover from A ‹bn(p · α) ♯* α› ‹bn(p · α) ♯* P′›
    have (p · p · xvec) ♯* (p · N) and (p · p · xvec) ♯* P′ by simp+
    moreover from A have Prop C Ψ P (p · M) (p · xvec) (p · N) P′
      by(rule cAlpha)
    ultimately have Prop C Ψ P (p · M) (p · p · xvec) (p · p · N) (p · P′)
      by(rule rAlpha)
    moreover from A ‹bn α ♯* subject α› have (p · xvec) ♯* (p · M) by simp
    hence xvec ♯* M by(simp add: fresh-star-bij)
    from A ‹bn(p · α) ♯* α› ‹distinctPerm p› have xvec ♯* (p · M) by simp
    hence (p · xvec) ♯* (p · p · M) by(simp add: fresh-star-bij)
    with ‹distinctPerm p› have (p · xvec) ♯* M by simp
    with ‹xvec ♯* M› S ‹distinctPerm p› have (p · M) = M by simp
    ultimately show ?case using S ‹distinctPerm p› by simp
  next
    case cInput
    thus ?case by(simp add: residualInject)
  next
    case cOutput
    thus ?case by(force dest: rOutput simp add: action.inject)
  next
    case cCase
    thus ?case by(force intro: rCase)
  next
    case cPar1
    thus ?case by(force intro: rPar1)
  next
    case cPar2
    thus ?case by(force intro: rPar2)
  next
    case cComm1
    thus ?case by(simp add: action.inject)
  next
    case cComm2
    thus ?case by(simp add: action.inject)
  next
    case cOpen
    thus ?case by(fastforce intro: rOpen simp add: action.inject)
  next
    case cScope
    thus ?case by(fastforce intro: rScope)
  next
    case cBang
```

thus *?case* **by**(*fastforce intro*: *rBang*)
qed
qed

lemma *inputInduct*[*consumes 1*, *case-names cInput cCase cPar1 cPar2 cScope cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and *Prop* :: $'d$::*fs-name* $\Rightarrow$ $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$
$'a \Rightarrow 'a \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and *rInput*: $\bigwedge \Psi\ M\ K\ xvec\ N\ Tvec\ P\ C.$
$[\![\Psi \vdash M \leftrightarrow K;\ distinct\ xvec;\ set\ xvec \subseteq supp\ N;$
$length\ xvec = length\ Tvec;\ xvec \sharp* \Psi;$
$xvec \sharp* M;\ xvec \sharp* K;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (M(\!|\lambda{*}xvec\ N|\!).P)$
$K\ (N[xvec::=Tvec])\ (P[xvec::=Tvec])$
and *rCase*: $\bigwedge \Psi\ P\ M\ N\ P'\ \varphi\ Cs\ C.\ [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ P\ M\ N\ P';\ (\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (Cases\ Cs)\ M\ N\ P'$
and *rPar1*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ N\ P'\ A_Q\ Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;$
$distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ M\ N\ P';\ distinct\ A_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* M;\ A_Q \sharp* N;$
$A_Q \sharp* P';\ A_Q \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ N\ (P' \parallel Q)$
and *rPar2*: $\bigwedge \Psi\ \Psi_P\ Q\ M\ N\ Q'\ A_P\ P\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;$
$distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ M\ N\ Q';\ distinct\ A_P;$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* M;\ A_P \sharp* N;$
$A_P \sharp* Q';\ A_P \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ N\ (P \parallel Q')$
and *rScope*: $\bigwedge \Psi\ P\ M\ N\ P'\ x\ C.$
$[\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ P\ M\ N\ P';\ x \sharp \Psi;\ x \sharp M;\ x \sharp N;\ x \sharp C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ M\ N\ ((\!|\nu x|\!)P')$
and *rBang*: $\bigwedge \Psi\ P\ M\ N\ P'\ C.$
$[\![\Psi \rhd P \parallel !P \longmapsto M(\!|N|\!) \prec P';\ guarded\ P;\ \bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ M\ N\ P']\!] \Longrightarrow Prop\ C\ \Psi\ (!P)\ M\ N\ P'$
shows $Prop\ C\ \Psi\ P\ M\ N\ P'$
using *Trans*
proof(*nominal-induct* $\Psi\ P\ Rs{==}M(\!|N|\!) \prec P'$ *avoiding*: C *arbitrary*: P' *rule*: *se-*

mantics.strong-induct)
  case(*cInput* Ψ *M K xvec N Tvec P C*)
  thus *?case*
    by(*force intro*: *rInput simp add*: *residualInject action.inject*)
next
  case(*Output* Ψ *M K N P C*)
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*Case* Ψ *P Rs* φ *CS C*)
  thus *?case* **by**(*force intro*: *rCase*)
next
  case(*cPar1* Ψ Ψ_Q *P* α *P′ Q* A_Q *C P″*)
  thus *?case* **by**(*force intro*: *rPar1 simp add*: *residualInject*)
next
  case(*cPar2* Ψ ΨP *Q* α *Q′ xvec P C Q″*)
  thus *?case* **by**(*force intro*: *rPar2 simp add*: *residualInject*)
next
  case(*cComm1* Ψ ΨQ *P M N P′ xvec* ΨP *Q K zvec Q′ yvec C PQ*)
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*cComm2* Ψ ΨQ *P M zvec N P′ xvec* ΨP *Q K yvec Q′ C PQ*)
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*cOpen* Ψ *P M xvec N P′ x yvec C P″*)
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*cScope* Ψ *P* α *P′ x C P″*)
  thus *?case* **by**(*force intro*: *rScope simp add*: *residualInject*)
next
  case(*Bang* Ψ *P Rs C*)
  thus *?case* **by**(*force intro*: *rBang*)
qed

lemma *tauInduct*[*consumes 1*, *case-names cCase cPar1 cPar2 cComm1 cComm2 cScope cBang*]:
  fixes Ψ :: *′b*
  and *P* :: (*′a*, *′b*, *′c*) *psi*
  and *Rs* :: (*′a*, *′b*, *′c*) *residual*
  and *Prop* :: *′d*::*fs-name* $\Rightarrow$ *′b* $\Rightarrow$ (*′a*, *′b*, *′c*) *psi* $\Rightarrow$
        (*′a*, *′b*, *′c*) *psi* $\Rightarrow$ *bool*
  and *C* :: *′d*::*fs-name*

  assumes *Trans*: $\Psi \rhd P \longmapsto\tau \prec P'$
  and *rCase*: $\bigwedge\Psi$ *P P′* φ *Cs C*. $[\![\Psi \rhd P \longmapsto\tau \prec P'$; $\bigwedge C$. *Prop C* Ψ *P P′*; (φ, *P*) *mem Cs*; $\Psi \vdash \varphi$; *guarded P*$]\!] \Longrightarrow$
        Prop C Ψ (*Cases Cs*) *P′*
  and *rPar1*: $\bigwedge\Psi$ Ψ_Q *P P′* A_Q *Q C*.
        $[\![\Psi \otimes \Psi_Q \rhd P \longmapsto\tau \prec P'$; *extractFrame Q* = $\langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;

$\bigwedge C.$ *Prop* C $(\Psi \otimes \Psi_Q)$ P P';
$A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* \Psi$;
$A_Q \sharp* P'$; $A_Q \sharp* C]\!] \Longrightarrow$
Prop C Ψ $(P \parallel Q)$ $(P' \parallel Q)$

and *rPar2*: $\bigwedge \Psi$ Ψ_P Q Q' A_P P C.
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto \tau \prec Q'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\bigwedge C.$ *Prop* C $(\Psi \otimes \Psi_P)$ Q Q';
$A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* \Psi$;
$A_P \sharp* Q'$; $A_P \sharp* C]\!] \Longrightarrow$
Prop C Ψ $(P \parallel Q)$ $(P \parallel Q')$

and *rComm1*: $\bigwedge \Psi$ Ψ_Q P M N P' A_P Ψ_P Q K *xvec* Q' A_Q C.
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp*$ *xvec*; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$;
$A_Q \sharp*$ *xvec*; *xvec* $\sharp* \Psi$; *xvec* $\sharp* \Psi_P$; *xvec* $\sharp* \Psi_Q$; *xvec* $\sharp* P$; *xvec* $\sharp* M$;
xvec $\sharp* Q$; *xvec* $\sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; *xvec* $\sharp* C]\!] \Longrightarrow$
Prop C Ψ $(P \parallel Q)$ $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

and *rComm2*: $\bigwedge \Psi$ Ψ_Q P M *xvec* N P' A_P Ψ_P Q K Q' A_Q C.
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp*$ *xvec*; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$;
$A_Q \sharp*$ *xvec*; *xvec* $\sharp* \Psi$; *xvec* $\sharp* \Psi_P$; *xvec* $\sharp* \Psi_Q$; *xvec* $\sharp* P$; *xvec* $\sharp* M$;
xvec $\sharp* Q$; *xvec* $\sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; *xvec* $\sharp* C]\!] \Longrightarrow$
Prop C Ψ $(P \parallel Q)$ $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

and *rScope*: $\bigwedge \Psi$ P P' x C.
$[\![\Psi \rhd P \longmapsto \tau \prec P'$; $\bigwedge C.$ *Prop* C Ψ P P'; $x \sharp \Psi$; $x \sharp C]\!] \Longrightarrow$
Prop C Ψ $((\!|\nu x|\!)P)$ $((\!|\nu x|\!)P')$

and *rBang*: $\bigwedge \Psi$ P P' C.
$[\![\Psi \rhd P \parallel !P \longmapsto \tau \prec P'$; *guarded* P; $\bigwedge C.$ *Prop* C Ψ $(P \parallel !P)$ $P']\!]$
$\Longrightarrow$ *Prop* C Ψ $(!P)$ P'

shows *Prop* C Ψ P P'
using *Trans*
proof(*nominal-induct* Ψ P *Rs*$==\tau \prec P'$ *avoiding*: C *arbitrary*: P' *rule*: *semantics.strong-induct*)
case(*cInput* M K *xvec* N *Tvec* P C)
thus *?case* **by**(*simp add*: *residualInject*)

next
  case(*Output* Ψ *M K N P C*)
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*Case* Ψ *P Rs* φ *Cs C*)
  thus *?case* **by**(*force intro*: *rCase simp add*: *residualInject*)
next
  case(*cPar1* Ψ Ψ_Q *P* α P' A_Q *Q C* P'')
  thus *?case* **by**(*force intro*: *rPar1 simp add*: *residualInject*)
next
  case(*cPar2* Ψ Ψ_P *Q* α Q' A_P *P C* Q'')
  thus *?case* **by**(*force intro*: *rPar2 simp add*: *residualInject*)
next
  case(*cComm1* Ψ Ψ_Q *P M N* P' A_P Ψ_P *Q K xvec* Q' A_Q *C PQ*)
  thus *?case* **by**(*force intro*: *rComm1 simp add*: *residualInject*)
next
  case(*cComm2* Ψ Ψ_Q *P M xvec N* P' A_P ΨP Q' A_Q *C PQ*)
  thus *?case* **by**(*force intro*: *rComm2 simp add*: *residualInject*)
next
  case(*cOpen* Ψ *P M xvec N* P' *x yvec C* P'')
  thus *?case* **by**(*simp add*: *residualInject*)
next
  case(*cScope* Ψ *P* α P' *x C* P'')
  thus *?case* **by**(*force intro*: *rScope simp add*: *residualInject*)
next
  case(*Bang* Ψ *P Rs C*)
  thus *?case* **by**(*force intro*: *rBang simp add*: *residualInject*)
qed

lemma *semanticsFrameInduct*[*consumes 3*, *case-names cAlpha cInput cOutput cCase cPar1 cPar2 cComm1 cComm2 cOpen cScope cBang*]:
  fixes Ψ :: *'b*
  and *P* :: (*'a*, *'b*, *'c*) *psi*
  and *Rs* :: (*'a*, *'b*, *'c*) *residual*
  and A_P :: *name list*
  and Ψ_P :: *'b*
  and *Prop* :: *'d*::*fs-name* $\Rightarrow$ *'b* $\Rightarrow$ (*'a*, *'b*, *'c*) *psi* $\Rightarrow$
    (*'a*, *'b*, *'c*) *residual* $\Rightarrow$ *name list* $\Rightarrow$ *'b* $\Rightarrow$ *bool*
  and *C* :: *'d*::*fs-name*

  assumes *Trans*: $\Psi \rhd P \longmapsto Rs$
  and *FrP*: *extractFrame* $P = \langle A_P, \Psi_P \rangle$
  and *distinct* A_P
  and *rAlpha*: $\bigwedge \Psi\ P\ A_P\ \Psi_P\ p\ Rs\ C.\ [\![A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* (p \cdot A_P);\ A_P \sharp* Rs;\ A_P \sharp* C;$
    set $p \subseteq$ *set* $A_P \times$ *set*$(p \cdot A_P)$; *distinctPerm p*;
    Prop C Ψ *P Rs* A_P $\Psi_P]\!] \Longrightarrow$ *Prop C* Ψ *P Rs* $(p \cdot A_P)$ $(p \cdot \Psi_P)$
  and *rInput*: $\bigwedge \Psi$ *M K xvec N Tvec P C*.

$[\![\Psi \vdash M \leftrightarrow K;\ \mathit{distinct}\ \mathit{xvec};\ \mathit{set}\ \mathit{xvec} \subseteq \mathit{supp}\ N;$
$\mathit{length}\ \mathit{xvec} = \mathit{length}\ \mathit{Tvec};\ \mathit{xvec} \sharp* \Psi;$
$\mathit{xvec} \sharp* M;\ \mathit{xvec} \sharp* K;\ \mathit{xvec} \sharp* C]\!] \Longrightarrow$
$\mathit{Prop}\ C\ \Psi\ (M(\!|\lambda{*}\mathit{xvec}\ N|\!).P)$
$(K(\!|(N[\mathit{xvec}::=\mathit{Tvec}])|\!) \prec (P[\mathit{xvec}::=\mathit{Tvec}]))\ ([])\ (\mathbf{1})$

and *rOutput*: $\bigwedge \Psi\ M\ K\ N\ P\ C.\ \Psi \vdash M \leftrightarrow K \Longrightarrow \mathit{Prop}\ C\ \Psi\ (M\langle N\rangle.P)\ (K\langle N\rangle \prec P)\ ([])\ (\mathbf{1})$

and *rCase*: $\bigwedge \Psi\ P\ \mathit{Rs}\ \varphi\ \mathit{Cs}\ A_P\ \Psi_P\ C.\ [\![\Psi \rhd P \longmapsto \mathit{Rs};\ \mathit{extractFrame}\ P = \langle A_P, \Psi_P\rangle;\ \mathit{distinct}\ A_P;\ \bigwedge C.\ \mathit{Prop}\ C\ \Psi\ P\ \mathit{Rs}\ A_P\ \Psi_P;$
$(\varphi, P)\ \mathit{mem}\ \mathit{Cs};\ \Psi \vdash \varphi;\ \mathit{guarded}\ P;\ \Psi_P \simeq \mathbf{1};\ (\mathit{supp}\ \Psi_P) = (\{\}::\mathit{name}\ \mathit{set});$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* \mathit{Rs};\ A_P \sharp* C]\!] \Longrightarrow \mathit{Prop}\ C\ \Psi\ (\mathit{Cases}\ \mathit{Cs})\ \mathit{Rs}\ ([])\ (\mathbf{1})$

and *rPar1*: $\bigwedge \Psi\ \Psi_Q\ P\ \alpha\ P'\ A_Q\ Q\ A_P\ \Psi_P\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P';$
$\mathit{extractFrame}\ P = \langle A_P, \Psi_P\rangle;\ \mathit{distinct}\ A_P;$
$\mathit{extractFrame}\ Q = \langle A_Q, \Psi_Q\rangle;\ \mathit{distinct}\ A_Q;$
$\bigwedge C.\ \mathit{Prop}\ C\ (\Psi \otimes \Psi_Q)\ P\ (\alpha \prec P')\ A_P\ \Psi_P;\ \mathit{distinct}(\mathit{bn}\ \alpha);$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* \alpha;\ A_P \sharp* P';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* \alpha;\ A_Q \sharp* P';\ A_Q \sharp* \Psi_P;$
$\mathit{bn}\ \alpha \sharp* \Psi;\ \mathit{bn}\ \alpha \sharp* P;\ \mathit{bn}\ \alpha \sharp* Q;\ \mathit{bn}\ \alpha \sharp* \mathit{subject}\ \alpha;\ \mathit{bn}\ \alpha \sharp* \Psi_P;\ \mathit{bn}\ \alpha \sharp* \Psi_Q;$
$A_P \sharp* C;\ A_Q \sharp* C;\ \mathit{bn}\ \alpha \sharp* C]\!] \Longrightarrow$
$\mathit{Prop}\ C\ \Psi\ (P \parallel Q)\ (\alpha \prec (P' \parallel Q))\ (A_P @ A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rPar2*: $\bigwedge \Psi\ \Psi_P\ Q\ \alpha\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q';$
$\mathit{extractFrame}\ P = \langle A_P, \Psi_P\rangle;\ \mathit{distinct}\ A_P;$
$\mathit{extractFrame}\ Q = \langle A_Q, \Psi_Q\rangle;\ \mathit{distinct}\ A_Q;$
$\bigwedge C.\ \mathit{Prop}\ C\ (\Psi \otimes \Psi_P)\ Q\ (\alpha \prec Q')\ A_Q\ \Psi_Q;\ \mathit{distinct}(\mathit{bn}\ \alpha);$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* \alpha;\ A_P \sharp* Q';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* \alpha;\ A_Q \sharp* Q';\ A_Q \sharp* \Psi_P;$
$\mathit{bn}\ \alpha \sharp* \Psi;\ \mathit{bn}\ \alpha \sharp* P;\ \mathit{bn}\ \alpha \sharp* Q;\ \mathit{bn}\ \alpha \sharp* \mathit{subject}\ \alpha;\ \mathit{bn}\ \alpha \sharp* \Psi_P;\ \mathit{bn}\ \alpha \sharp* \Psi_Q;$
$A_P \sharp* C;\ A_Q \sharp* C;\ \mathit{bn}\ \alpha \sharp* C]\!] \Longrightarrow$
$\mathit{Prop}\ C\ \Psi\ (P \parallel Q)\ (\alpha \prec (P \parallel Q'))\ (A_P @ A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rComm1*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ \mathit{xvec}\ Q'\ A_Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ \mathit{extractFrame}\ P = \langle A_P, \Psi_P\rangle;\ \mathit{distinct}\ A_P;$
$\bigwedge C.\ \mathit{Prop}\ C\ (\Psi \otimes \Psi_Q)\ P\ ((M(\!|N|\!)) \prec P')\ A_P\ \Psi_P;$
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}\mathit{xvec}|\!)\langle N\rangle \prec Q';\ \mathit{extractFrame}\ Q = \langle A_Q, \Psi_Q\rangle;\ \mathit{distinct}\ A_Q;$
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$
$\bigwedge C.\ \mathit{Prop}\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|\nu{*}\mathit{xvec}|\!)\langle N\rangle \prec Q')\ A_Q\ \Psi_Q;\ \mathit{distinct}\ \mathit{xvec};$
$A_P \sharp* \Psi;\ A_P \sharp* \Psi_Q;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* P';$
$A_P \sharp* Q;\ A_P \sharp* Q';\ A_P \sharp* A_Q;\ A_P \sharp* \mathit{xvec};\ A_Q \sharp* \Psi;\ A_Q \sharp* \Psi_P;$
$A_Q \sharp* P;\ A_Q \sharp* N;\ A_Q \sharp* P';\ A_Q \sharp* Q;\ A_Q \sharp* K;\ A_Q \sharp* Q';$

$A_Q \sharp* xvec$; $xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* \Psi_Q$; $xvec \sharp* P$; $xvec \sharp* M$;
$xvec \sharp* Q$; $xvec \sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; $xvec \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rComm2*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ C.$
$\llbracket \Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; $extractFrame\ P = \langle A_P, \Psi_P\rangle$; $distinct\ A_P$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P')\ A_P\ \Psi_P$;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; $extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$; $distinct\ A_Q$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|N|\!) \prec Q')\ A_Q\ \Psi_Q$;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; $distinct\ xvec$;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* xvec$; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$;
$A_Q \sharp* xvec$; $xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* \Psi_Q$; $xvec \sharp* P$; $xvec \sharp* M$;
$xvec \sharp* Q$; $xvec \sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; $xvec \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rOpen*: $\bigwedge \Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ A_P\ \Psi_P\ C.$
$\llbracket \Psi \rhd P \longmapsto M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P'$; $extractFrame\ P = \langle A_P, \Psi_P\rangle$; $distinct\ A_P$;
$\bigwedge C.\ Prop\ C\ \Psi\ P\ (M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P')\ A_P\ \Psi_P$; $x \in supp\ N$; $x \sharp \Psi$; $x \sharp M$;
$x \sharp A_P$; $x \sharp xvec$; $x \sharp yvec$; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* xvec$; $A_P \sharp* yvec$; $xvec \sharp* yvec$; $distinct\ xvec$; $distinct\ yvec$;
$xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* \Psi_P$; $yvec \sharp* \Psi_P$;
$yvec \sharp* \Psi$; $yvec \sharp* P$; $yvec \sharp* M$; $A_P \sharp* C$; $x \sharp C$; $xvec \sharp* C$; $yvec \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (M(\!|\nu{*}(xvec@x\#yvec)|\!)\langle N\rangle \prec P')\ (x\#A_P)\ \Psi_P$
and *rScope*: $\bigwedge \Psi\ P\ \alpha\ P'\ x\ A_P\ \Psi_P\ C.$
$\llbracket \Psi \rhd P \longmapsto \alpha \prec P'$; $extractFrame\ P = \langle A_P, \Psi_P\rangle$; $distinct\ A_P$;
$\bigwedge C.\ Prop\ C\ \Psi\ P\ (\alpha \prec P')\ A_P\ \Psi_P$;
$x \sharp \Psi$; $x \sharp \alpha$; $x \sharp A_P$; $A_P \sharp* \Psi$; $A_P \sharp* P$;
$A_P \sharp* \alpha$; $A_P \sharp* P'$; $distinct(bn\ \alpha)$;
$bn\ \alpha \sharp* \Psi$; $bn\ \alpha \sharp* P$; $bn\ \alpha \sharp* subject\ \alpha$; $bn\ \alpha \sharp* \Psi_P$;
$A_P \sharp* C$; $x \sharp C$; $bn\ \alpha \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (\alpha \prec ((\!|\nu x|\!)P'))\ (x\#A_P)\ \Psi_P$
and *rBang*: $\bigwedge \Psi\ P\ Rs\ A_P\ \Psi_P\ C.$
$\llbracket \Psi \rhd P \parallel !P \longmapsto Rs$; $guarded\ P$; $extractFrame\ P = \langle A_P, \Psi_P\rangle$; $distinct\ A_P$;
$\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ Rs\ A_P\ (\Psi_P \otimes \mathbf{1})$; $\Psi_P \simeq \mathbf{1}$; $supp\ \Psi_P = (\{\}::name\ set)$;
$A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* Rs$; $A_P \sharp* C\rrbracket \Longrightarrow Prop\ C\ \Psi\ (!P)\ Rs\ ([])\ (\mathbf{1})$
shows $Prop\ C\ \Psi\ P\ Rs\ A_P\ \Psi_P$
using *Trans FrP* ‹*distinct* A_P›
proof(*nominal-induct avoiding*: $A_P\ \Psi_P\ C$ *rule*: *semantics.strong-induct*)

case(*cInput* Ψ M K *xvec* N *Tvec* P A_P Ψ_P C)
from ‹*extractFrame* (M⦇λ*$xvec$ N⦈.P) = $\langle A_P, \Psi_P\rangle$›
have $A_P = []$ **and** $\Psi_P = \mathbf{1}$
by *auto*
with ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp* N› ‹*length xvec* = *length Tvec*›
‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ M› ‹*xvec* $\sharp*$ K› ‹*xvec* $\sharp*$ C›
show *?case* **by**(*blast intro*: *rInput*)
next
case(*Output* Ψ M K N P A_P Ψ_P)
from ‹*extractFrame* ($M\langle N\rangle.P$) = $\langle A_P, \Psi_P\rangle$›
have $A_P = []$ **and** $\Psi_P = \mathbf{1}$
by *auto*
with ‹$\Psi \vdash M \leftrightarrow K$› **show** *?case*
by(*blast intro*: *rOutput*)
next
case(*Case* Ψ P *Rs* φ *Cs* A_{cP} Ψ_{cP} C)
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *distinct* A_P
and $A_P \sharp* (\Psi, P, Rs, C)$
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* P$ **and** $A_P \sharp* Rs$ **and** $A_P \sharp* C$
by *simp*+
note ‹$\Psi \rhd P \longmapsto Rs$› *FrP* ‹*distinct* A_P›
moreover from *FrP* ‹*distinct* A_P› ‹$\bigwedge A_P\ \Psi_P\ C.$ ⟦*extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P⟧ $\Longrightarrow$ *Prop* C Ψ P *Rs* A_P Ψ_P›
have $\bigwedge C.$ *Prop* C Ψ P *Rs* A_P Ψ_P **by** *simp*
moreover note ‹(φ, P) *mem Cs*› ‹$\Psi \vdash \varphi$› ‹*guarded* P›
moreover from ‹*guarded* P› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **and** *supp* Ψ_P = ({}::*name set*)
by(*metis guardedStatEq*)+
moreover note ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Rs$› ‹$A_P \sharp* C$›
ultimately have *Prop* C Ψ (*Cases Cs*) *Rs* ([]) (**1**)
by(*rule rCase*)
thus *?case* **using** ‹*extractFrame*(*Cases Cs*) = $\langle A_{cP}, \Psi_{cP}\rangle$› **by** *simp*
next
case(*cPar1* Ψ Ψ_Q P α P' Q A_Q A_{PQ} Ψ_{PQ} C)
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *distinct* A_P
$A_P \sharp* (P, Q, \Psi, \alpha, P', A_Q, A_{PQ}, C, \Psi_Q)$
by(*rule freshFrame*)
hence $A_P \sharp* P$ **and** $A_P \sharp* Q$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* \alpha$ **and** $A_P \sharp* P'$
and $A_P \sharp* A_Q$ **and** $A_P \sharp* A_{PQ}$ **and** $A_P \sharp* C$ **and** $A_P \sharp* \Psi_Q$
by *simp*+

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*

from ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_Q$› *FrP* **have** $A_Q \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)

from ‹*bn* $\alpha \sharp* P$› ‹$A_P \sharp* \alpha$› *FrP* **have** *bn* $\alpha \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)

from ‹*extractFrame*$(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› *FrP FrQ* ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›
have $\langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$
by *simp*
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› ‹$A_P \sharp* A_Q$› **have** *distinct*$(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
ultimately obtain p **where** S: *set* $p \subseteq$ *set*$(A_P@A_Q) \times$ *set*$((p \cdot A_P)@(p \cdot A_Q))$ **and** *distinctPerm* p
and Ψeq: $\Psi_{PQ} = p \cdot (\Psi_P \otimes \Psi_Q)$ **and** Aeq: $A_{PQ} = (p \cdot A_P)@(p \cdot A_Q)$
using ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$› ‹*distinct* A_{PQ}›
by(*rule-tac frameChainEq′*) (*assumption* | *simp add*: *eqvts*)+

note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$› *FrP* ‹*distinct* A_P› *FrQ* ‹*distinct* A_Q›

moreover from *FrP* ‹*distinct* A_P› ‹$\bigwedge A_P\ \Psi_P\ C.\ [\![$*extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* $A_P]\!] \Longrightarrow$ *Prop* $C\ (\Psi \otimes \Psi_Q)\ P\ (\alpha \prec P')\ A_P\ \Psi_P$›
have $\bigwedge C.$ *Prop* $C\ (\Psi \otimes \Psi_Q)\ P\ (\alpha \prec P')\ A_P\ \Psi_P$ **by** *simp*

moreover note ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$›
‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \alpha$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* \Psi_P$› ‹*distinct*(*bn* α)›
‹*bn* $\alpha \sharp* \Psi$› ‹*bn* $\alpha \sharp* P$› ‹*bn* $\alpha \sharp* Q$› ‹*bn* $\alpha \sharp*$ *subject* α› ‹*bn* $\alpha \sharp* \Psi_P$› ‹*bn* $\alpha \sharp* \Psi_Q$›
‹$A_P \sharp* C$› ‹$A_Q \sharp* C$› ‹*bn* $\alpha \sharp* C$›
ultimately have *Prop* $C\ \Psi\ (P \parallel Q)\ (\alpha \prec (P' \parallel Q))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
by(*rule-tac rPar1*)
with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* A_{PQ}$› ‹$A_P \sharp* C$›
‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* \alpha$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* A_{PQ}$› ‹$A_Q \sharp* C$›
S ‹*distinctPerm* p› Aeq
have *Prop* $C\ \Psi\ (P \parallel Q)\ (\alpha \prec (P' \parallel Q))\ (p \cdot (A_P@A_Q))\ (p \cdot (\Psi_P \otimes \Psi_Q))$
by(*rule-tac rAlpha*) (*assumption* | *simp add*: *eqvts*)+
with Ψeq Aeq **show** *?case* **by**(*simp add*: *eqvts*)
next
case(*cPar2* $\Psi\ \Psi_P\ Q\ \alpha\ Q'\ P\ A_P\ A_{PQ}\ \Psi_{PQ}\ C$)
obtain $A_Q\ \Psi_Q$ **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **and** *distinct* A_Q
$A_Q \sharp* (P, Q, \Psi, \alpha, Q', A_P, A_{PQ}, C, \Psi_P)$
by(*rule freshFrame*)
hence $A_Q \sharp* P$ **and** $A_Q \sharp* Q$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* \alpha$ **and** $A_Q \sharp* Q'$
and $A_Q \sharp* A_P$ **and** $A_Q \sharp* A_{PQ}$ **and** $A_Q \sharp* C$ **and** $A_Q \sharp* \Psi_P$
by *simp*+

from ‹$A_Q \sharp* A_P$› **have** $A_P \sharp* A_Q$ **by** *simp*
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*

from ‹A_P ♯* Q› ‹A_Q ♯* A_P› FrQ **have** A_P ♯* Ψ_Q
by(*force dest*: *extractFrameFreshChain*)
from ‹$bn\ \alpha$ ♯* Q› ‹A_Q ♯* α› FrQ **have** $bn\ \alpha$ ♯* Ψ_Q
by(*force dest*: *extractFrameFreshChain*)

from ‹$extractFrame(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› $FrP\ FrQ$ ‹A_P ♯* A_Q› ‹A_P ♯* Ψ_Q› ‹A_Q ♯* Ψ_P›
have $\langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$
by *simp*
moreover from ‹$distinct\ A_P$› ‹$distinct\ A_Q$› ‹A_P ♯* A_Q› **have** $distinct(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
ultimately obtain p **where** S: $(set\ p \subseteq (set(A_P@A_Q)) \times (set\ A_{PQ}))$ **and** $distinctPerm\ p$
and Ψeq: $\Psi_{PQ} = p \cdot (\Psi_P \otimes \Psi_Q)$ **and** Aeq: $A_{PQ} = ((p \cdot A_P)@(p \cdot A_Q))$
using ‹A_P ♯* A_{PQ}› ‹A_Q ♯* A_{PQ}› ‹$distinct\ A_{PQ}$›
by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

note ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$› FrP ‹$distinct\ A_P$› FrQ ‹$distinct\ A_Q$›

moreover from FrQ ‹$distinct\ A_Q$› ‹$\bigwedge A_Q\ \Psi_Q\ C.\ [\![extractFrame\ Q = \langle A_Q, \Psi_Q\rangle; distinct\ A_Q]\!] \Longrightarrow Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (\alpha \prec Q')\ A_Q\ \Psi_Q$›
have $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (\alpha \prec Q')\ A_Q\ \Psi_Q$ **by** *simp*

moreover note ‹A_P ♯* P› ‹A_P ♯* Q› ‹A_P ♯* Ψ› ‹A_P ♯* α› ‹A_P ♯* Q'› ‹A_P ♯* A_Q› ‹A_P ♯* Ψ_Q›
‹A_Q ♯* P› ‹A_Q ♯* Q› ‹A_Q ♯* Ψ› ‹A_Q ♯* α› ‹A_Q ♯* Q'› ‹A_Q ♯* Ψ_P› ‹$distinct(bn\ \alpha)$›
‹$bn\ \alpha$ ♯* Ψ› ‹$bn\ \alpha$ ♯* P› ‹$bn\ \alpha$ ♯* Q› ‹$bn\ \alpha$ ♯* $subject\ \alpha$› ‹$bn\ \alpha$ ♯* Ψ_P› ‹$bn\ \alpha$ ♯* Ψ_Q›
‹A_P ♯* C› ‹A_Q ♯* C› ‹$bn\ \alpha$ ♯* C›
ultimately have $Prop\ C\ \Psi\ (P \parallel Q)\ (\alpha \prec (P \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
by(*rule-tac rPar2*)

with ‹A_P ♯* Ψ› ‹A_P ♯* P› ‹A_P ♯* Q› ‹A_P ♯* α› ‹A_P ♯* Q'› ‹A_P ♯* A_{PQ}› ‹A_P ♯* C›
‹A_Q ♯* Ψ› ‹A_Q ♯* P› ‹A_Q ♯* Q› ‹A_Q ♯* α› ‹A_Q ♯* Q'› ‹A_Q ♯* A_{PQ}› ‹A_Q ♯* C›
S ‹$distinctPerm\ p$› Aeq
have $Prop\ C\ \Psi\ (P \parallel Q)\ (\alpha \prec (P \parallel Q'))\ (p \cdot (A_P@A_Q))\ (p \cdot (\Psi_P \otimes \Psi_Q))$
by(*rule-tac rAlpha*) (*assumption* | *simp add*: *eqvts*)+
with $\Psi eq\ Aeq$ **show** *?case* **by**(*simp add*: *eqvts*)
next
case(*cComm1* $\Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q\ A_{PQ}\ \Psi_{PQ}\ C$)
from ‹$distinct\ A_P$› ‹$distinct\ A_Q$› ‹A_P ♯* A_Q› **have** $distinct(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
from *cComm1* **have** $Prop\ C\ \Psi\ (P \parallel Q)\ (\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
by(*rule-tac rComm1*)

moreover from ‹*extractFrame*$(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$›
‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›
have $\langle (A_P@A_Q), (\Psi_P \otimes \Psi_Q)\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$
by *simp*
with ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$› ‹*distinct*$(A_P@A_Q)$› ‹*distinct* A_{PQ}›
obtain p **where** S: (*set* $p \subseteq$ (*set*$(A_P@A_Q)) \times$ (*set* A_{PQ})) **and** *distinctPerm* p
and Ψeq: $\Psi_{PQ} = p \cdot (\Psi_P \otimes \Psi_Q)$ **and** Aeq: $A_{PQ} = p \cdot (A_P@A_Q)$
by(*rule-tac frameChainEq′*) (*assumption* | *simp*)+
moreover note ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_P \sharp*$ *xvec*›
‹$A_Q \sharp*$ *xvec*› ‹$A_P \sharp* P'$› ‹$A_Q \sharp* P'$› ‹$A_P \sharp* Q'$› ‹$A_Q \sharp* Q'$› ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$›
‹$A_P \sharp* C$› ‹$A_Q \sharp* C$›
ultimately have *Prop* C Ψ $(P \parallel Q)$ $(\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q'))$ $(p \cdot (A_P@A_Q))$ $(p \cdot (\Psi_P \otimes \Psi_Q))$
by(*rule-tac rAlpha*) *auto*
with Ψeq Aeq **show** *?case* **by** *simp*
next
case(*cComm2* Ψ Ψ_Q P M *xvec* N P' A_P Ψ_P Q K Q' A_Q A_{PQ} Ψ_{PQ} C)
from ‹*distinct* A_P› ‹*distinct* A_Q› ‹$A_P \sharp* A_Q$› **have** *distinct*$(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
from *cComm2* **have** *Prop* C Ψ $(P \parallel Q)$ $(\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q'))$ $(A_P@A_Q)$ $(\Psi_P \otimes \Psi_Q)$
by(*rule-tac rComm2*)
moreover from ‹*extractFrame*$(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$›
‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›
have $\langle (A_P@A_Q), (\Psi_P \otimes \Psi_Q)\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$
by *simp*
with ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$› ‹*distinct*$(A_P@A_Q)$› ‹*distinct* A_{PQ}›
obtain p **where** S: (*set* $p \subseteq$ (*set*$(A_P@A_Q)) \times$ (*set* A_{PQ})) **and** *distinctPerm* p
and Ψeq: $\Psi_{PQ} = p \cdot (\Psi_P \otimes \Psi_Q)$ **and** Aeq: $A_{PQ} = p \cdot (A_P@A_Q)$
by(*rule-tac frameChainEq′*) (*assumption* | *simp*)+
moreover note ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_P \sharp*$ *xvec*›
‹$A_Q \sharp*$ *xvec*› ‹$A_P \sharp* P'$› ‹$A_Q \sharp* P'$› ‹$A_P \sharp* Q'$› ‹$A_Q \sharp* Q'$› ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$›
‹$A_P \sharp* C$› ‹$A_Q \sharp* C$›
ultimately have *Prop* C Ψ $(P \parallel Q)$ $(\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q'))$ $(p \cdot (A_P@A_Q))$ $(p \cdot (\Psi_P \otimes \Psi_Q))$
by(*rule-tac rAlpha*) *auto*
with Ψeq Aeq **show** *?case* **by** *simp*
next
case(*cOpen* Ψ P M *xvec* *yvec* N P' x A_{xP} Ψ_{xP} C)
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *distinct* A_P
and $A_P \sharp* (\Psi, P, M, xvec, yvec, N, P', A_{xP}, \Psi_{xP}, C, x)$
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* P$ **and** $A_P \sharp* M$ **and** $A_P \sharp*$ *xvec***and** $A_P \sharp*$ *yvec*

and $A_P \sharp* N$ **and** $A_P \sharp* P'$
and $A_P \sharp* A_{xP}$ **and** $A_P \sharp* \Psi_{xP}$ **and** $A_P \sharp* C$ **and** $x \sharp A_P$
by *simp*+

from ‹$xvec \sharp* P$› ‹$A_P \sharp* xvec$› FrP **have** $xvec \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)
from ‹$yvec \sharp* P$› ‹$A_P \sharp* yvec$› FrP **have** $yvec \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)

from ‹$extractFrame((\!|\nu x|\!)P) = \langle A_{xP}, \Psi_{xP}\rangle$› FrP
have $\langle (x\#A_P), \Psi_P\rangle = \langle A_{xP}, \Psi_{xP}\rangle$
by *simp*
moreover from ‹$x \sharp A_P$› ‹$distinct\ A_P$› **have** $distinct(x\#A_P)$ **by** *simp*
ultimately obtain p **where** S: $set\ p \subseteq set\ (x\#A_P) \times set\ (p \cdot (x\#A_P))$ **and** $distinctPerm\ p$
and Ψeq: $\Psi_{xP} = p \cdot \Psi_P$ **and** Aeq: $A_{xP} = (p \cdot x)\#(p \cdot A_P)$
using ‹$A_P \sharp* A_{xP}$›‹$x \sharp A_{xP}$› ‹$distinct\ A_{xP}$›
by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

note ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$› FrP ‹$distinct\ A_P$›
moreover from FrP ‹$distinct\ A_P$› ‹$\bigwedge A_P\ \Psi_P\ C.\ [\![extractFrame\ P = \langle A_P, \Psi_P\rangle; distinct\ A_P]\!] \Longrightarrow Prop\ C\ \Psi\ P\ (M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P')\ A_P\ \Psi_P$›
have $\bigwedge C.\ Prop\ C\ \Psi\ P\ (M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P')\ A_P\ \Psi_P$ **by** *simp*
moreover note ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› ‹$x \in supp\ N$› ‹$x \sharp A_P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* xvec$› ‹$A_P \sharp* yvec$› ‹$A_P \sharp* N$› ‹$A_P \sharp* P'$›
‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$xvec \sharp* M$› ‹$xvec \sharp* \Psi_P$› ‹$yvec \sharp* \Psi$› ‹$yvec \sharp* P$› ‹$yvec \sharp* M$› ‹$yvec \sharp* \Psi_P$›
‹$A_P \sharp* C$› ‹$x \sharp C$› ‹$xvec \sharp* C$› ‹$yvec \sharp* C$› ‹$xvec \sharp* yvec$› ‹$distinct\ xvec$› ‹$distinct\ yvec$›
ultimately have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P')\ (x\#A_P)\ \Psi_P$
by(*rule-tac rOpen*)

with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* xvec$› ‹$A_P \sharp* yvec$› ‹$A_P \sharp* N$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* A_{xP}$› ‹$A_P \sharp* C$› ‹$x \sharp A_{xP}$› ‹$A_P \sharp* A_{xP}$› ‹$x \sharp A_P$›
‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp C$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› Aeq
S ‹$distinctPerm\ p$›
have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P')\ (p \cdot (x\#A_P))\ (p \cdot \Psi_P)$
by(*rule-tac* $A_P{=}x\#A_P$ **in** *rAlpha*) (*assumption* | *simp add*: *abs-fresh fresh-star-def boundOutputFresh*)+
with $\Psi eq\ Aeq$ **show** *?case* **by**(*simp add*: *eqvts*)
next
case(*cScope* $\Psi\ P\ \alpha\ P'\ x\ A_{xP}\ \Psi_{xP}\ C$)
obtain $A_P\ \Psi_P$ **where** FrP: $extractFrame\ P = \langle A_P, \Psi_P\rangle$ **and** $distinct\ A_P$
and $A_P \sharp* (\Psi, P, \alpha, P', A_{xP}, \Psi_{xP}, C, x)$
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* P$ **and** $A_P \sharp* \alpha$ **and** $A_P \sharp* P'$

and $A_P \sharp* A_{xP}$ **and** $A_P \sharp* \Psi_{xP}$ **and** $A_P \sharp* C$ **and** $x \sharp A_P$
by *simp+*

from ‹$bn\ \alpha \sharp* P$› ‹$A_P \sharp* \alpha$› FrP **have** $bn\ \alpha \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)

from ‹$extractFrame((\!|\nu x|\!)P) = \langle A_{xP}, \Psi_{xP}\rangle$› FrP
have $\langle (x\#A_P), \Psi_P\rangle = \langle A_{xP}, \Psi_{xP}\rangle$
by *simp*
moreover from ‹$x \sharp A_P$› ‹$distinct\ A_P$› **have** $distinct(x\#A_P)$ **by** *simp*
ultimately obtain p **where** S: $set\ p \subseteq set\ (x\#A_P) \times set\ (p \cdot (x\#A_P))$ **and** $distinctPerm\ p$
and Ψeq: $\Psi_{xP} = p \cdot \Psi_P$ **and** Aeq: $A_{xP} = (p \cdot x)\#(p \cdot A_P)$
using ‹$A_P \sharp* A_{xP}$›‹$x \sharp A_{xP}$› ‹$distinct\ A_{xP}$›
by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

note ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› FrP ‹$distinct\ A_P$›
moreover from FrP ‹$distinct\ A_P$› ‹$\bigwedge A_P\ \Psi_P\ C.\ [\![extractFrame\ P = \langle A_P, \Psi_P\rangle; distinct\ A_P]\!] \Longrightarrow Prop\ C\ \Psi\ P\ (\alpha \prec P')\ A_P\ \Psi_P$›
have $\bigwedge C.\ Prop\ C\ \Psi\ P\ (\alpha \prec P')\ A_P\ \Psi_P$ **by** *simp*
moreover note ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp A_P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* P'$› ‹$distinct(bn\ \alpha)$›
‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* \Psi_P$› ‹$A_P \sharp* C$› ‹$x \sharp C$› ‹$bn\ \alpha \sharp* C$›
ultimately have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (\alpha \prec ((\!|\nu x|\!)P'))\ (x\#A_P)\ \Psi_P$
by(*rule-tac rScope*)

with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* A_{xP}$› ‹$A_P \sharp* C$› ‹$x \sharp A_{xP}$› ‹$A_P \sharp* A_{xP}$› ‹$x \sharp A_P$›
‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp C$› Aeq
S ‹$distinctPerm\ p$›
have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (\alpha \prec ((\!|\nu x|\!)P'))\ (p \cdot (x\#A_P))\ (p \cdot \Psi_P)$
by(*rule-tac* $A_P{=}x\#A_P$ **in** *rAlpha*) (*assumption* | *simp add*: *abs-fresh fresh-star-def*)+
with $\Psi eq\ Aeq$ **show** *?case* **by**(*simp add*: *eqvts*)
next
case(*Bang* $\Psi\ P\ Rs\ A_{bP}\ \Psi_{bP}\ C$)

obtain $A_P\ \Psi_P$ **where** FrP: $extractFrame\ P = \langle A_P, \Psi_P\rangle$ **and** $distinct\ A_P$
and $A_P \sharp* (\Psi, P, Rs, C)$
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* P$ **and** $A_P \sharp* Rs$ **and** $A_P \sharp* C$
by *simp+*

note ‹$\Psi \rhd P \parallel !P \longmapsto Rs$› ‹$guarded\ P$› FrP ‹$distinct\ A_P$›
moreover from FrP **have** $extractFrame\ (P \parallel !P) = \langle A_P, \Psi_P \otimes \mathbf{1}\rangle$
by *simp*
with ‹$distinct\ A_P$› ‹$\bigwedge A_P\ \Psi_P\ C.\ [\![extractFrame\ (P \parallel !P) = \langle A_P, \Psi_P\rangle; distinct\ A_P]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ Rs\ A_P\ \Psi_P$›
have $\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ Rs\ A_P\ (\Psi_P \otimes \mathbf{1})$ **by** *simp*

moreover from ‹*guarded P*› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **and** *supp* $\Psi_P = (\{\}::name\ set)$ **by**(*metis guardedStatEq*)+
moreover note ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Rs$› ‹$A_P \sharp* C$›
ultimately have *Prop C* Ψ (!*P*) *Rs* ([]) (**1**)
by(*rule rBang*)
thus *?case* **using** ‹*extractFrame*(!*P*) = $\langle A_{bP}, \Psi_{bP}\rangle$› **by** *simp*
qed

lemma *semanticsFrameInduct'*[*consumes 5*, *case-names cAlpha cFrameAlpha cInput cOutput cCase cPar1 cPar2 cComm1 cComm2 cOpen cScope cBang*]:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rs* :: (*'a*, *'b*, *'c*) *residual*
and A_P :: *name list*
and Ψ_P :: *'b*
and *Prop* :: *'d*::*fs-name* ⇒ *'b* ⇒ (*'a*, *'b*, *'c*) *psi* ⇒ *'a action* ⇒
(*'a*, *'b*, *'c*) *psi* ⇒ *name list* ⇒ *'b* ⇒ *bool*
and *C* :: *'d*::*fs-name*

assumes *Trans*: $\Psi \rhd P \longmapsto\alpha \prec P'$
and *FrP*: *extractFrame P* = $\langle A_P, \Psi_P\rangle$
and *distinct* A_P
and *bn* $\alpha \sharp*$ *subject* α
and *distinct*(*bn* α)
and *rAlpha*: $\bigwedge\Psi\ P\ \alpha\ P'\ p\ A_P\ \Psi_P\ C.\ [\![bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* \Psi_P;$
$bn\ \alpha \sharp* C;\ bn\ \alpha \sharp* (p \cdot \alpha);\ A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* \alpha;\ A_P \sharp* P';\ A_P \sharp* C;$
$set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha));\ distinctPerm\ p;$
$bn(p \cdot \alpha) \sharp* \alpha;\ (bn(p \cdot \alpha)) \sharp* P';\ Prop\ C\ \Psi\ P\ \alpha\ P'\ A_P\ \Psi_P]\!] \Longrightarrow$
$Prop\ C\ \Psi\ P\ (p \cdot \alpha)\ (p \cdot P')\ A_P\ \Psi_P$
and *rFrameAlpha*: $\bigwedge\Psi\ P\ A_P\ \Psi_P\ p\ \alpha\ P'\ C.\ [\![A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* (p \cdot A_P);\ A_P \sharp* \alpha;\ A_P \sharp* P';\ A_P \sharp* C;$
$set\ p \subseteq set\ A_P \times set(p \cdot A_P);\ distinctPerm\ p;\ A_P \sharp* subject\ \alpha;$
$Prop\ C\ \Psi\ P\ \alpha\ P'\ A_P\ \Psi_P]\!] \Longrightarrow Prop\ C\ \Psi\ P\ \alpha\ P'\ (p \cdot A_P)\ (p \cdot \Psi_P)$
and *rInput*: $\bigwedge\Psi\ M\ K\ xvec\ N\ Tvec\ P\ C.$
$[\![\Psi \vdash M \leftrightarrow K;\ distinct\ xvec;\ set\ xvec \subseteq supp\ N;$
$length\ xvec = length\ Tvec;\ xvec \sharp* \Psi;$
$xvec \sharp* M;\ xvec \sharp* K;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (M(\!|\lambda* xvec\ N|\!).P)$
$(K(\!|(N[xvec::=Tvec])|\!))\ (P[xvec::=Tvec])\ ([])\ (\mathbf{1})$
and *rOutput*: $\bigwedge\Psi\ M\ K\ N\ P\ C.\ \Psi \vdash M \leftrightarrow K \Longrightarrow Prop\ C\ \Psi\ (M\langle N\rangle.P)\ (K\langle N\rangle)\ P\ ([])\ (\mathbf{1})$
and *rCase*: $\bigwedge\Psi\ P\ \alpha\ P'\ \varphi\ Cs\ A_P\ \Psi_P\ C.\ [\![\Psi \rhd P \longmapsto\alpha \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;\ \bigwedge C.\ Prop\ C\ \Psi\ P\ \alpha\ P'\ A_P\ \Psi_P;$

$(\varphi,\ P)$ *mem Cs*; $\Psi \vdash \varphi$; *guarded* P; $\Psi_P \simeq \mathbf{1}$; $(supp\ \Psi_P) = (\{\}::name\ set)$;
$A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* \alpha$; $A_P \sharp* P'$; $A_P \sharp* C\rrbracket \Longrightarrow Prop\ C\ \Psi\ (Cases\ Cs)\ \alpha\ P'\ ([])\ (\mathbf{1})$

and *rPar1*: $\bigwedge \Psi\ \Psi_Q\ P\ \alpha\ P'\ A_Q\ Q\ A_P\ \Psi_P\ C.$
$\llbracket \Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$;
$extractFrame\ P = \langle A_P,\ \Psi_P\rangle$; $distinct\ A_P$;
$extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$; $distinct\ A_Q$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ \alpha\ P'\ A_P\ \Psi_P$;
$A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* \Psi$; $A_P \sharp* \alpha$; $A_P \sharp* P'$; $A_P \sharp* A_Q$; $A_P \sharp* \Psi_Q$;
$A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* \Psi$; $A_Q \sharp* \alpha$; $A_Q \sharp* P'$; $A_Q \sharp* \Psi_P$;
$bn\ \alpha \sharp* \Psi$; $bn\ \alpha \sharp* P$; $bn\ \alpha \sharp* Q$; $bn\ \alpha \sharp* subject\ \alpha$; $bn\ \alpha \sharp* \Psi_P$; $bn\ \alpha \sharp* \Psi_Q$;
$A_P \sharp* C$; $A_Q \sharp* C$; $bn\ \alpha \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ \alpha\ (P' \parallel Q)\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rPar2*: $\bigwedge \Psi\ \Psi_P\ Q\ \alpha\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ C.$
$\llbracket \Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$;
$extractFrame\ P = \langle A_P,\ \Psi_P\rangle$; $distinct\ A_P$;
$extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$; $distinct\ A_Q$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ \alpha\ Q'\ A_Q\ \Psi_Q$;
$A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* \Psi$; $A_P \sharp* \alpha$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* \Psi_Q$;
$A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* \Psi$; $A_Q \sharp* \alpha$; $A_Q \sharp* Q'$; $A_Q \sharp* \Psi_P$;
$bn\ \alpha \sharp* \Psi$; $bn\ \alpha \sharp* P$; $bn\ \alpha \sharp* Q$; $bn\ \alpha \sharp* subject\ \alpha$; $bn\ \alpha \sharp* \Psi_P$; $bn\ \alpha \sharp* \Psi_Q$;
$A_P \sharp* C$; $A_Q \sharp* C$; $bn\ \alpha \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ \alpha\ (P \parallel Q')\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rComm1*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q\ C.$
$\llbracket \Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$; $extractFrame\ P = \langle A_P,\ \Psi_P\rangle$; $distinct\ A_P$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|N|\!))\ P'\ A_P\ \Psi_P$;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$; $extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$; $distinct\ A_Q$;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; $distinct\ xvec$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|\nu{*}xvec|\!)\langle N\rangle)\ Q'\ A_Q\ \Psi_Q$;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* xvec$; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$;
$A_Q \sharp* xvec$; $xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* \Psi_Q$; $xvec \sharp* P$; $xvec \sharp* M$;
$xvec \sharp* Q$; $xvec \sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; $xvec \sharp* C\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (\tau)\ ((\!|\nu{*}xvec|\!)(P' \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rComm2*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ C.$
$\llbracket \Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; $extractFrame\ P = \langle A_P,\ \Psi_P\rangle$; $distinct\ A_P$;
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|\nu{*}xvec|\!)\langle N\rangle)\ P'\ A_P\ \Psi_P$;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; $extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$; $distinct\ A_Q$;

$\bigwedge C.$ *Prop* C $(\Psi \otimes \Psi_P)$ Q $(K(\!|N|\!))$ Q' A_Q Ψ_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* xvec$; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$;
$A_Q \sharp* xvec$; $xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* \Psi_Q$; $xvec \sharp* P$; $xvec \sharp* M$;
$xvec \sharp* Q$; $xvec \sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; $xvec \sharp* C$⟧ $\Longrightarrow$
Prop C Ψ $(P \parallel Q)$ (τ) $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$ $(A_P@A_Q)$ $(\Psi_P \otimes \Psi_Q)$
and *rOpen*: $\bigwedge \Psi$ P M *xvec yvec* N P' x A_P Ψ_P y C.
⟦$\Psi \rhd P \longmapsto M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\bigwedge C.$ *Prop* C Ψ P $(M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle)$ P' A_P Ψ_P; $x \in$ *supp* N; $x \sharp \Psi$; $x \sharp M$;
$x \sharp A_P$; $x \sharp xvec$; $x \sharp yvec$; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* xvec$; $A_P \sharp* yvec$; $xvec \sharp* yvec$; *distinct xvec*; *distinct yvec*;
$xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* \Psi_P$;
$yvec \sharp* \Psi$; $yvec \sharp* P$; $yvec \sharp* M$; $A_P \sharp* C$; $x \sharp C$; $xvec \sharp* C$; $yvec \sharp* C$;
$y \neq x$; $y \sharp \Psi$; $y \sharp P$; $y \sharp M$; $y \sharp xvec$; $y \sharp yvec$; $y \sharp N$; $y \sharp P'$; $y \sharp A_P$; $y \sharp \Psi_P$; $y \sharp C$⟧ $\Longrightarrow$
Prop C Ψ $((\!|\nu x|\!)P)$ $(M(\!|\nu{*}(xvec@y\#yvec)|\!)\langle([(x, y)] \cdot N)\rangle)$ $([(x, y)] \cdot P')$ $(x\#A_P)$ Ψ_P
and *rScope*: $\bigwedge \Psi$ P α P' x A_P Ψ_P C.
⟦$\Psi \rhd P \longmapsto \alpha \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\bigwedge C.$ *Prop* C Ψ P α P' A_P Ψ_P;
$x \sharp \Psi$; $x \sharp \alpha$; $x \sharp A_P$; $A_P \sharp* \Psi$; $A_P \sharp* P$;
$A_P \sharp* \alpha$; $A_P \sharp* P'$;
bn $\alpha \sharp* \Psi$; *bn* $\alpha \sharp* P$; *bn* $\alpha \sharp*$ *subject* α; *bn* $\alpha \sharp* \Psi_P$;
$A_P \sharp* C$; $x \sharp C$; *bn* $\alpha \sharp* C$⟧ $\Longrightarrow$
Prop C Ψ $((\!|\nu x|\!)P)$ α $((\!|\nu x|\!)P')$ $(x\#A_P)$ Ψ_P
and *rBang*: $\bigwedge \Psi$ P α P' A_P Ψ_P C.
⟦$\Psi \rhd P \parallel !P \longmapsto \alpha \prec P'$; *guarded* P; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\bigwedge C.$ *Prop* C Ψ $(P \parallel !P)$ α P' A_P $(\Psi_P \otimes \mathbf{1})$; $\Psi_P \simeq \mathbf{1}$; *supp* Ψ_P $= (\{\}::$*name set*$)$;
$A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* \alpha$; $A_P \sharp* P'$; $A_P \sharp* C$⟧ $\Longrightarrow$ *Prop* C Ψ $(!P)$ α P' $([])$ $(\mathbf{1})$
shows *Prop* C Ψ P α P' A_P Ψ_P
using *Trans FrP* ‹*distinct* A_P› ‹*bn* $\alpha \sharp*$ *subject* α› ‹*distinct*(*bn* α)›
proof(*nominal-induct* Ψ P *Rs*==$\alpha \prec P'$ A_P Ψ_P *avoiding*: C α P' *rule*: *semanticsFrameInduct*)
case *cAlpha*
thus *?case* **using** *rFrameAlpha*
by *auto*
next
case *cInput*
thus *?case* **using** *rInput*

by(*auto simp add*: *residualInject*)
next
case *cOutput*
thus *?case* **using** *rOutput*
by(*auto simp add*: *residualInject*)
next
case *cCase*
thus *?case* **using** *rCase*
by(*auto simp add*: *residualInject*)
next
case(*cPar1* Ψ Ψ_Q P α P' A_Q Q A_P Ψ_P C α' P'')
note ‹$\alpha \prec (P' \parallel Q) = \alpha' \prec P''$›
moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* (bn\ \alpha')$ **by** *auto*
moreover note ‹*distinct* $(bn\ \alpha)$› ‹*distinct*$(bn\ \alpha')$›
moreover from ‹$bn\ \alpha \sharp*$ *subject* α› ‹$bn\ \alpha' \sharp*$ *subject* α'›
have $bn\ \alpha \sharp* (\alpha \prec P' \parallel Q)$ **and** $bn\ \alpha' \sharp* (\alpha' \prec P'')$ **by** *simp+*
ultimately obtain p **where** S: $(set\ p) \subseteq (set(bn\ \alpha)) \times (set(bn(p \cdot \alpha)))$ **and** *distinctPerm* p
and αEq: $\alpha' = p \cdot \alpha$ **and** $P'eq$: $P'' = p \cdot (P' \parallel Q)$ **and** $(bn(p \cdot \alpha)) \sharp* \alpha$
and $(bn(p \cdot \alpha)) \sharp* (P' \parallel Q)$
by(*rule residualEq*)

note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$› ‹*extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$› ‹*distinct* A_Q›
moreover from ‹$bn\ \alpha \sharp*$ *subject* α› ‹*distinct*$(bn\ \alpha)$› ‹$A_P \sharp* \alpha$›
have $\bigwedge C.$ *Prop* C $(\Psi \otimes \Psi_Q)$ P α P' A_P Ψ_P **by**(*rule-tac cPar1*) *auto*

moreover note ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \alpha$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* C$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* C$›
‹$bn\ \alpha \sharp* Q$› ‹*distinct*$(bn\ \alpha)$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_Q$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$bn\ \alpha \sharp* C$›
‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› ‹*distinct* A_P› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$› ‹$bn\ \alpha \sharp* \Psi_P$›
ultimately have *Prop* C Ψ $(P \parallel Q)$ α $(P' \parallel Q)$ $(A_P @ A_Q)$ $(\Psi_P \otimes \Psi_Q)$
by(*rule-tac rPar1*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* (bn\ \alpha')$› S ‹*distinctPerm* p› ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn(p \cdot \alpha) \sharp* (P' \parallel Q)$› ‹$bn\ \alpha \sharp* \Psi_P$› ‹$bn\ \alpha \sharp* \Psi_Q$› ‹$A_P \sharp* \alpha$› ‹$A_Q \sharp* \alpha$› ‹$A_P \sharp* \alpha'$› ‹$A_Q \sharp* \alpha'$› αEq ‹$bn\ \alpha \sharp* \Psi_P$› ‹$bn\ \alpha \sharp* \alpha'$› ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_P \sharp* P'$› ‹$A_Q \sharp* P'$› ‹$A_P \sharp* C$› ‹$A_Q \sharp* C$›
have *Prop* C Ψ $(P \parallel Q)$ $(p \cdot \alpha)$ $(p \cdot (P' \parallel Q))$ $(A_P @ A_Q)$ $(\Psi_P \otimes \Psi_Q)$
by(*rule-tac rAlpha*) *auto*
with αEq $P'eq$ ‹*distinctPerm* p› **show** *?case* **by** *simp*
next
case(*cPar2* Ψ Ψ_P Q α Q' A_P P A_Q Ψ_Q C α' Q'')
note ‹$\alpha \prec (P \parallel Q') = \alpha' \prec Q''$›
moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* (bn\ \alpha')$ **by** *auto*
moreover note ‹*distinct* $(bn\ \alpha)$› ‹*distinct*$(bn\ \alpha')$›
moreover from ‹$bn\ \alpha \sharp*$ *subject* α› ‹$bn\ \alpha' \sharp*$ *subject* α'›

have $bn\ \alpha \sharp* (\alpha \prec P \parallel Q')$ **and** $bn\ \alpha' \sharp* (\alpha' \prec Q'')$ **by** *simp+*
ultimately obtain p **where** S: $(set\ p) \subseteq (set(bn\ \alpha)) \times (set(bn(p \cdot \alpha)))$ **and** *distinctPerm p*
and αEq: $\alpha' = p \cdot \alpha$ **and** $Q'eq$: $Q'' = p \cdot (P \parallel Q')$ **and** $(bn(p \cdot \alpha)) \sharp* \alpha$
and $(bn(p \cdot \alpha)) \sharp* (P \parallel Q')$
by(*rule residualEq*)

note ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$› ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› ‹*distinct* A_P›
moreover from ‹$bn\ \alpha \sharp*$ *subject* α› ‹$distinct(bn\ \alpha)$› ‹$A_Q \sharp* \alpha$›
have $\bigwedge C.$ *Prop* $C\ (\Psi \otimes \Psi_P)\ Q\ \alpha\ Q'\ A_Q\ \Psi_Q$ **by**(*rule-tac cPar2*) *auto*

moreover note ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \alpha$› ‹$A_Q \sharp* Q'$› ‹$A_Q \sharp* C$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* Q'$› ‹$A_P \sharp* C$›
‹$bn\ \alpha \sharp* Q$› ‹$distinct(bn\ \alpha)$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_Q$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$bn\ \alpha \sharp* C$›
‹*extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$› ‹*distinct* A_Q› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$› ‹$bn\ \alpha \sharp* \Psi_P$›
ultimately have *Prop* $C\ \Psi\ (P \parallel Q)\ \alpha\ (P \parallel Q')\ (A_P @ A_Q)\ (\Psi_P \otimes \Psi_Q)$
by(*rule-tac rPar2*) *auto*
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* (bn\ \alpha')$› S ‹*distinctPerm* p› ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn(p \cdot \alpha) \sharp* (P \parallel Q')$› ‹$bn\ \alpha \sharp* \Psi_P$› ‹$bn\ \alpha \sharp* \Psi_Q$› ‹$A_P \sharp* \alpha$› ‹$A_Q \sharp* \alpha$› ‹$A_P \sharp* \alpha'$› ‹$A_Q \sharp* \alpha'$› αEq ‹$bn\ \alpha \sharp* \alpha'$› ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_P \sharp* Q'$› ‹$A_Q \sharp* Q'$› ‹$A_P \sharp* C$› ‹$A_Q \sharp* C$›
have *Prop* $C\ \Psi\ (P \parallel Q)\ (p \cdot \alpha)\ (p \cdot (P \parallel Q'))\ (A_P @ A_Q)\ (\Psi_P \otimes \Psi_Q)$
by(*rule-tac rAlpha*) *auto*
with αEq $Q'eq$ ‹*distinctPerm* p› **show** *?case* **by** *simp*
next
case(*cComm1* $\Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q\ C\ \alpha\ P''$)
thus *?case* **using** *rComm1*
apply(*auto*)
apply(*drule-tac* $x{=}M(\!|N|\!)$ **in** *meta-spec*)
apply(*drule-tac* $x{=}K(\!|\nu{*}xvec|\!)\langle N \rangle$ **in** *meta-spec*)
apply(*drule-tac* $x{=}P'$ **in** *meta-spec*)
apply(*drule-tac* $x{=}Q'$ **in** *meta-spec*)
apply *auto*
apply(*drule-tac* $x{=}\Psi$ **in** *meta-spec*)
apply(*drule-tac* $x{=}\Psi_Q$ **in** *meta-spec*)
apply(*drule-tac* $x{=}P$ **in** *meta-spec*)
apply(*drule-tac* $x{=}M$ **in** *meta-spec*)
apply(*drule-tac* $x{=}N$ **in** *meta-spec*)
apply(*drule-tac* $x{=}P'$ **in** *meta-spec*)
apply(*drule-tac* $x{=}A_P$ **in** *meta-spec*)
apply(*drule-tac* $x{=}\Psi_P$ **in** *meta-spec*)
apply(*drule-tac* $x{=}Q$ **in** *meta-spec*)
apply(*drule-tac* $x{=}K$ **in** *meta-spec*)
apply(*drule-tac* $x{=}xvec$ **in** *meta-spec*)
apply(*drule-tac* $x{=}Q'$ **in** *meta-spec*)

apply(*drule-tac* $x{=}A_Q$ **in** *meta-spec*)
apply *auto*
apply(*subgoal-tac* $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|\nu{*}xvec|\!)\langle N\rangle)\ Q'\ A_Q\ \Psi_Q$)
apply(*subgoal-tac* $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|N|\!))\ P'\ A_P\ \Psi_P$)
by(*auto simp add*: *residualInject*)
next
case(*cComm2* $\Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ C\ \alpha\ Q''$)
thus *?case* **using** *rComm2*
apply(*drule-tac* $x{=}M(\!|\nu{*}xvec|\!)\langle N\rangle$ **in** *meta-spec*)
apply(*drule-tac* $x{=}K(\!|N|\!)$ **in** *meta-spec*)
apply(*drule-tac* $x{=}P'$ **in** *meta-spec*)
apply(*drule-tac* $x{=}Q'$ **in** *meta-spec*)
apply *auto*
apply(*drule-tac* $x{=}\Psi$ **in** *meta-spec*)
apply(*drule-tac* $x{=}\Psi_Q$ **in** *meta-spec*)
apply(*drule-tac* $x{=}P$ **in** *meta-spec*)
apply(*drule-tac* $x{=}M$ **in** *meta-spec*)
apply(*drule-tac* $x{=}xvec$ **in** *meta-spec*)
apply(*drule-tac* $x{=}N$ **in** *meta-spec*)
apply(*drule-tac* $x{=}P'$ **in** *meta-spec*)
apply(*drule-tac* $x{=}A_P$ **in** *meta-spec*)
apply(*drule-tac* $x{=}\Psi_P$ **in** *meta-spec*)
apply(*drule-tac* $x{=}Q$ **in** *meta-spec*)
apply(*drule-tac* $x{=}K$ **in** *meta-spec*)
apply(*drule-tac* $x{=}Q'$ **in** *meta-spec*)
apply(*drule-tac* $x{=}A_Q$ **in** *meta-spec*)
apply *auto*
apply(*subgoal-tac* $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ (M(\!|\nu{*}xvec|\!)\langle N\rangle)\ P'\ A_P\ \Psi_P$)
apply(*subgoal-tac* $\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ (K(\!|N|\!))\ Q'\ A_Q\ \Psi_Q$)
by(*auto simp add*: *residualInject*)
next
case(*cOpen* $\Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ A_P\ \Psi_P\ C\ \alpha\ P''$)
note ‹$M(\!|\nu{*}(xvec@x\#yvec)|\!)\langle N\rangle \prec P' = \alpha \prec P''$›
moreover from ‹$xvec\ \sharp{*}\ \alpha$› ‹$x\ \sharp\ \alpha$› ‹$yvec\ \sharp{*}\ \alpha$› **have** $(xvec@x\#yvec)\ \sharp{*}\ (bn\ \alpha)$
by *auto*
moreover from ‹$xvec\ \sharp{*}\ yvec$› ‹$x\ \sharp\ xvec$› ‹$x\ \sharp\ yvec$› ‹*distinct xvec*› ‹*distinct yvec*›
have $distinct(xvec@x\#yvec)$
by(*auto simp add*: *fresh-star-def*) (*simp add*: *fresh-def name-list-supp*)
moreover note ‹$distinct(bn\ \alpha)$›
moreover from ‹$xvec\ \sharp{*}\ M$› ‹$x\ \sharp\ M$› ‹$yvec\ \sharp{*}\ M$› **have** $(xvec@x\#yvec)\ \sharp{*}\ M$
by *auto*
hence $(xvec@x\#yvec)\ \sharp{*}\ (M(\!|\nu{*}(xvec@x\#yvec)|\!)\langle N\rangle \prec P')$ **by** *auto*
moreover from ‹$bn\ \alpha\ \sharp{*}\ subject\ \alpha$› **have** $bn\ \alpha\ \sharp{*}\ (\alpha \prec P'')$ **by** *simp*
ultimately obtain p **where** S: $(set\ p) \subseteq (set(xvec@x\#yvec)) \times (set(p \cdot (xvec@x\#yvec)))$
and *distinctPerm p*
and αeq: $\alpha = (p \cdot M)(\!|\nu{*}(p \cdot (xvec@x\#yvec))|\!)\langle(p \cdot N)\rangle$ **and** $P'eq$: $P'' = (p \cdot P')$
and A: $(xvec@x\#yvec)\ \sharp{*}\ ((p \cdot M)(\!|\nu{*}(p \cdot (xvec@x\#yvec))|\!)\langle(p \cdot N)\rangle)$
and B: $(p \cdot (xvec@x\#yvec))\ \sharp{*}\ (M(\!|\nu{*}(xvec@x\#yvec)|\!)\langle N\rangle)$

and C: $(p \cdot (xvec@x\#yvec))$ $\sharp*$ P'
by(*rule-tac residualEq*) (*assumption* | *simp*)+

note ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$› ‹$x \in (supp\ N)$›

moreover {
fix C
from ‹$xvec$ $\sharp*$ M› ‹$yvec$ $\sharp*$ M› **have** $(xvec@yvec)$ $\sharp*$ M **by** *simp*
moreover from ‹*distinct xvec*› ‹*distinct yvec*› ‹$xvec$ $\sharp*$ $yvec$› **have** $distinct(xvec@yvec)$
by *auto* (*simp add*: *fresh-star-def name-list-supp fresh-def*)
ultimately have *Prop* C Ψ P $(M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle)$ P' A_P Ψ_P **using** ‹A_P $\sharp*$ $xvec$› ‹A_P $\sharp*$ $yvec$› ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ N›
by(*rule-tac cOpen*) *auto*
}
moreover obtain y::*name* **where** $y \sharp \Psi$ **and** $y \neq x$ **and** $y \sharp P$ **and** $y \sharp xvec$ **and** $y \sharp yvec$ **and** $y \sharp \alpha$ **and** $y \sharp P'$ **and** $y \sharp A_P$ **and** $y \sharp \Psi_P$ **and** $y \sharp M$ **and** $y \sharp N$ **and** $y \sharp C$ **and** $y \sharp p$
by(*generate-fresh name*) *auto*
moreover note ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› ‹$xvec$ $\sharp*$ Ψ› ‹$xvec$ $\sharp*$ P› ‹$xvec$ $\sharp*$ M›
‹$yvec$ $\sharp*$ Ψ› ‹$yvec$ $\sharp*$ P› ‹$yvec$ $\sharp*$ M› ‹$yvec$ $\sharp*$ C› ‹$x \sharp C$› ‹$xvec$ $\sharp*$ C› ‹*distinct xvec*› ‹*distinct yvec*›
‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹*distinct* A_P› ‹$x \sharp A_P$› ‹$xvec$ $\sharp*$ $yvec$› ‹$xvec$ $\sharp*$ Ψ_P›
‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ $xvec$› ‹A_P $\sharp*$ $yvec$› ‹A_P $\sharp*$ N› ‹A_P $\sharp*$ P'› ‹A_P $\sharp*$ C›
ultimately have *Prop* C Ψ $((\!|\nu x|\!)P)$ $(M(\!|\nu*(xvec@y\#yvec)|\!)\langle([(x, y)] \cdot N)\rangle)$ $([(x, y)] \cdot P')$ $(x\#A_P)$ Ψ_P
by(*rule-tac rOpen*)
moreover have $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot M) = [(x, y)] \cdot p \cdot M$
by(*subst perm-compose*[*symmetric*]) *simp*
with ‹$y \sharp M$› ‹$x \sharp \alpha$› αeq ‹$y \sharp p$› ‹$x \sharp M$› **have** D: $(([(x, y)] \cdot p) \cdot M) = p \cdot M$
by(*auto simp add*: *eqvts freshChainSimps*)
moreover have $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot xvec) = [(x, y)] \cdot p \cdot xvec$
by(*subst perm-compose*[*symmetric*]) *simp*
with ‹$y \sharp xvec$› ‹$x \sharp \alpha$› αeq ‹$y \sharp p$› ‹$x \sharp xvec$› **have** E: $(([(x, y)] \cdot p) \cdot xvec) = p \cdot xvec$
by(*auto simp add*: *eqvts freshChainSimps*)
moreover have $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot yvec) = [(x, y)] \cdot p \cdot yvec$
by(*subst perm-compose*[*symmetric*]) *simp*
with ‹$y \sharp yvec$› ‹$x \sharp \alpha$› αeq ‹$y \sharp p$› ‹$x \sharp yvec$› **have** F: $(([(x, y)] \cdot p) \cdot yvec) = p \cdot yvec$
by(*auto simp add*: *eqvts freshChainSimps*)
moreover have $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot x) = [(x, y)] \cdot p \cdot x$
by(*subst perm-compose*[*symmetric*]) *simp*
with ‹$y \neq x$› ‹$y \sharp p$› **have** G: $(([(x, y)] \cdot p) \cdot y) = p \cdot x$
apply(*simp add*: *freshChainSimps calc-atm*)
apply(*subgoal-tac* $y \neq p \cdot x$)
apply(*clarsimp*)

using A αeq
apply(*simp add*: *eqvts*)
apply(*subst fresh-atm*[*symmetric*])
apply(*simp only*: *freshChainSimps*)
by *simp*
moreover have $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot N) = [(x, y)] \cdot p \cdot N$
by(*subst perm-compose*[*symmetric*]) *simp*
with ‹$y \sharp N$› ‹$x \sharp \alpha$› ‹$y \sharp p$› αeq **have** H: $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot N) = p \cdot N$
by(*auto simp add*: *eqvts freshChainSimps*)
moreover have $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot P') = [(x, y)] \cdot p \cdot P'$
by(*subst perm-compose*[*symmetric*]) *simp*
with ‹$y \sharp P'$› ‹$x \sharp P''$› ‹$y \sharp p$› $P'eq$ **have** I: $(([(x, y)] \cdot p) \cdot [(x, y)] \cdot P') = p \cdot P'$
by(*auto simp add*: *eqvts freshChainSimps*)
from ‹$y \sharp p$› ‹$y \neq x$› **have** $y \neq p \cdot x$
apply(*subst fresh-atm*[*symmetric*])
apply(*simp only*: *freshChainSimps*)
by *simp*
moreover from S **have** $([(x, y)] \cdot set\ p) \subseteq [(x, y)] \cdot (set(xvec@x\#yvec) \times set(p \cdot (xvec@x\#yvec)))$
by(*simp*)
with ‹$y \neq p \cdot x$› ‹$(([(x, y)] \cdot p) \cdot y) = p \cdot x$› ‹$x \sharp xvec$› ‹$y \sharp xvec$› ‹$x \sharp yvec$› ‹$y \sharp yvec$› ‹$y \sharp p$› ‹$x \sharp \alpha$› αeq **have**
$set([(x, y)] \cdot p) \subseteq set(xvec@y\#yvec) \times set(([(x, y)] \cdot p) \cdot (xvec@y\#yvec))$
by(*simp add*: *eqvts calc-atm perm-compose*)
moreover note ‹$xvec \sharp* \Psi$› ‹$yvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$yvec \sharp* P$› ‹$xvec \sharp* M$› ‹$yvec \sharp* M$›
‹$yvec \sharp* C$› S ‹$distinctPerm\ p$› ‹$x \sharp C$› ‹$xvec \sharp* C$› ‹$xvec \sharp* \Psi_P$› ‹$yvec \sharp* \Psi_P$› ‹$x \sharp \Psi$›
‹$A_P \sharp* xvec$› ‹$x \sharp A_P$› ‹$A_P \sharp* yvec$› ‹$A_P \sharp* M$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› ‹$x \sharp M$› ‹$x \sharp A_P$› ‹$A_P \sharp* N$›
$A\ B\ C$ αeq ‹$A_P \sharp* \alpha$› ‹$y \sharp \Psi$› ‹$y \neq x$› ‹$y \sharp P$› ‹$y \sharp M$› ‹$y \sharp \Psi_P$› ‹$y \sharp C$› ‹$xvec \sharp* \alpha$› ‹$x \sharp \alpha$› ‹$yvec \sharp* \alpha$› ‹$y \sharp \alpha$› ‹$A_P \sharp* P$› ‹$A_P \sharp* \Psi$› ‹$y \sharp A_P$› ‹$y \sharp N$› ‹$A_P \sharp* P'$› ‹$y \sharp P'$› ‹$A_P \sharp* C$› $P'eq$
ultimately have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (([(x, y)] \cdot p) \cdot (M(\!|\nu*(xvec@y\#yvec)|\!)\langle([(x, y)] \cdot N)\rangle))\ ([(x, y)] \cdot p) \cdot [(x, y)] \cdot P')\ (x\#A_P)\ \Psi_P$
apply(*rule-tac* α=$M(\!|\nu*(xvec@y\#yvec)|\!)\langle([(x, y)] \cdot N)\rangle$ **in** *rAlpha*)
apply(*assumption* | *simp*)+
apply(*simp add*: *eqvts*)
apply(*assumption* | *simp add*: *abs-fresh*)+
apply(*simp add*: *fresh-left calc-atm*)
apply(*assumption* | *simp*)+
apply(*simp add*: *fresh-left calc-atm*)
apply(*assumption* | *simp*)+
by(*simp add*: *eqvts fresh-left*)+
with αeq $P'eq$ D E F G H I **show** *?case*
by(*simp add*: *eqvts*)
next
case(*cScope* Ψ P α P' x A_P Ψ_P C α' P'')

note ‹$\alpha \prec ((\!|\nu x|\!)P') = \alpha' \prec P''$›
moreover from ‹$bn\ \alpha\ \sharp*\ \alpha'$› **have** $bn\ \alpha\ \sharp*\ (bn\ \alpha')$ **by** *auto*
moreover note ‹$distinct\ (bn\ \alpha)$› ‹$distinct(bn\ \alpha')$›
moreover from ‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$bn\ \alpha'\ \sharp*\ subject\ \alpha'$›
have $bn\ \alpha\ \sharp*\ (\alpha \prec (\!|\nu x|\!)P')$ **and** $bn\ \alpha'\ \sharp*\ (\alpha' \prec P'')$ **by** *simp+*
ultimately obtain p **where** S: $(set\ p) \subseteq (set(bn\ \alpha)) \times (set(bn(p \cdot \alpha)))$ **and** $distinctPerm\ p$
and αEq: $\alpha' = p \cdot \alpha$ **and** $P'eq$: $P'' = p \cdot ((\!|\nu x|\!)P')$ **and** $(bn(p \cdot \alpha))\ \sharp*\ \alpha$
and $(bn(p \cdot \alpha))\ \sharp*\ ((\!|\nu x|\!)P')$
by(*rule residualEq*)

note ‹$\Psi \rhd P \longmapsto \alpha \prec P'$›
moreover from ‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
have $\bigwedge C.\ Prop\ C\ \Psi\ P\ \alpha\ P'\ A_P\ \Psi_P$ **by**(*rule-tac cScope*) *auto*

moreover note ‹$x\ \sharp\ \Psi$› ‹$x\ \sharp\ \alpha$› ‹$bn\ \alpha\ \sharp*\ \Psi$› ‹$bn\ \alpha\ \sharp*\ P$› ‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$bn\ \alpha\ \sharp*\ \Psi_P$›
‹$x\ \sharp\ C$› ‹$bn\ \alpha\ \sharp*\ C$› ‹$distinct(bn\ \alpha)$› ‹$extractFrame\ P = \langle A_P, \Psi_P\rangle$›
‹$distinct\ A_P$› ‹$x\ \sharp\ A_P$› ‹$A_P\ \sharp*\ \Psi$› ‹$A_P\ \sharp*\ P$› ‹$A_P\ \sharp*\ \alpha$› ‹$A_P\ \sharp*\ P'$› ‹$A_P\ \sharp*\ C$›
ultimately have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ \alpha\ ((\!|\nu x|\!)P')\ (x\#A_P)\ \Psi_P$
by(*rule-tac rScope*)
with ‹$bn\ \alpha\ \sharp*\ \Psi$› ‹$bn\ \alpha\ \sharp*\ P$› ‹$x\ \sharp\ \alpha$› ‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$bn\ \alpha\ \sharp*\ C$› ‹$bn\ \alpha\ \sharp*\ (bn\ \alpha')$› S ‹$distinctPerm\ p$› ‹$bn(p \cdot \alpha)\ \sharp*\ \alpha$› ‹$bn(p \cdot \alpha)\ \sharp*\ ((\!|\nu x|\!)P')$› ‹$A_P\ \sharp*\ \alpha$› ‹$A_P\ \sharp*\ \alpha'$› αEq ‹$x\ \sharp\ \alpha'$› ‹$bn\ \alpha\ \sharp*\ \Psi_P$› ‹$bn\ \alpha\ \sharp*\ \alpha'$› ‹$x\ \sharp\ \Psi$› ‹$A_P\ \sharp*\ \Psi$› ‹$x\ \sharp\ A_P$› ‹$A_P\ \sharp*\ P$› ‹$A_P\ \sharp*\ P'$› ‹$x\ \sharp\ C$› ‹$A_P\ \sharp*\ C$›
have $Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ (p \cdot \alpha)\ (p \cdot ((\!|\nu x|\!)P'))\ \ (x\#A_P)\ \Psi_P$
by(*rule-tac rAlpha*) (*simp add*: *abs-fresh*)+
with $\alpha Eq\ P'eq$ ‹$distinctPerm\ p$› **show** *?case* **by** *simp*
next
case($cBang\ \Psi\ P\ Rs\ A_P\ \Psi_P\ C\ \alpha$)
thus *?case* **by**(*rule-tac rBang*) *auto*
qed

lemma *inputFrameInduct*[*consumes 3*, *case-names cAlpha cInput cCase cPar1 cPar2 cScope cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)\ psi$
and $Prop$ :: $'d$::*fs-name* $\Rightarrow 'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow$
$'a \Rightarrow 'a \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow name\ list \Rightarrow 'b \Rightarrow bool$
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and *FrP*: $extractFrame\ P = \langle A_P, \Psi_P\rangle$
and $distinct\ A_P$

and *rAlpha*: $\bigwedge\Psi\ P\ M\ N\ P'\ A_P\ \Psi_P\ p\ C.\ [\![A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* P';\ A_P \sharp* (p \cdot A_P);\ A_P \sharp* C;$
$set\ p \subseteq set\ A_P \times set(p \cdot A_P);\ distinctPerm\ p;$
$Prop\ C\ \Psi\ P\ M\ N\ P'\ A_P\ \Psi_P]\!] \Longrightarrow Prop\ C\ \Psi\ P\ M\ N\ P'\ (p \cdot A_P)\ (p \cdot \Psi_P)$

and *rInput*: $\bigwedge\Psi\ M\ K\ xvec\ N\ Tvec\ P\ C.$
$[\![\Psi \vdash M \leftrightarrow K;\ distinct\ xvec;\ set\ xvec \subseteq supp\ N;$
$length\ xvec = length\ Tvec;\ xvec \sharp* \Psi;$
$xvec \sharp* M;\ xvec \sharp* K;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (M(\!|\lambda{*}xvec\ N|\!).P)$
$K\ (N[xvec::=Tvec])\ (P[xvec::=Tvec])\ ([])\ (\mathbf{1})$

and *rCase*: $\bigwedge\Psi\ P\ M\ N\ P'\ \varphi\ Cs\ A_P\ \Psi_P\ C.\ [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ P = \langle A_P,\ \Psi_P\rangle;\ distinct\ A_P;\ \bigwedge C.\ Prop\ C\ \Psi\ P\ M\ N\ P'\ A_P\ \Psi_P;$
$(\varphi,\ P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P;\ \Psi_P \simeq \mathbf{1};\ (supp\ \Psi_P) = (\{\}::name\ set);$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* P';\ A_P \sharp* C]\!] \Longrightarrow Prop\ C\ \Psi\ (Cases\ Cs)\ M\ N\ P'\ ([])\ (\mathbf{1})$

and *rPar1*: $\bigwedge\Psi\ \Psi_Q\ P\ M\ N\ P'\ A_Q\ Q\ A_P\ \Psi_P\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';$
$extractFrame\ P = \langle A_P,\ \Psi_P\rangle;\ distinct\ A_P;$
$extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ M\ N\ P'\ A_P\ \Psi_P;$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* P';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* M;\ A_Q \sharp* N;\ A_Q \sharp* P';\ A_Q \sharp* \Psi_P;$
$A_P \sharp* C;\ A_Q \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ N\ (P' \parallel Q)\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rPar2*: $\bigwedge\Psi\ \Psi_P\ Q\ M\ N\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q';$
$extractFrame\ P = \langle A_P,\ \Psi_P\rangle;\ distinct\ A_P;$
$extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ M\ N\ Q'\ A_Q\ \Psi_Q;$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* Q';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* M;\ A_Q \sharp* N;\ A_Q \sharp* Q';\ A_Q \sharp* \Psi_P;$
$A_P \sharp* C;\ A_Q \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ N\ (P \parallel Q')\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$

and *rScope*: $\bigwedge\Psi\ P\ M\ N\ P'\ x\ A_P\ \Psi_P\ C.$
$[\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ P = \langle A_P,\ \Psi_P\rangle;\ distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ \Psi\ P\ M\ N\ P'\ A_P\ \Psi_P;\ x \sharp \Psi;\ x \sharp M;\ x \sharp N;$
$x \sharp A_P;\ A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* P';$
$A_P \sharp* C;\ x \sharp C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ M\ N\ ((\!|\nu x|\!)P')\ (x\#A_P)\ \Psi_P$

and *rBang*: $\bigwedge\Psi\ P\ M\ N\ P'\ A_P\ \Psi_P\ C.$
$[\![\Psi \rhd P \parallel !P \longmapsto M(\!|N|\!) \prec P';\ guarded\ P;\ extractFrame\ P = \langle A_P,\ \Psi_P\rangle;\ distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ M\ N\ P'\ A_P\ (\Psi_P \otimes \mathbf{1});\ \Psi_P \simeq \mathbf{1};\ (supp$

$\Psi_P) = (\{\}::name\ set)$;
$A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (!P)\ M\ N\ P'\ ([])\ (\mathbf{1})$
shows $Prop\ C\ \Psi\ P\ M\ N\ P'\ A_P\ \Psi_P$
using *assms*
by(*nominal-induct* $\Psi\ P\ Rs{=}{=}M(\!|N|\!) \prec P'\ A_P\ \Psi_P$ *avoiding*: C *arbitrary*: P' *rule*: *semanticsFrameInduct*)
(*auto simp add*: *residualInject*)

lemma *outputFrameInduct*[*consumes 3*, *case-names cAlpha cOutput cCase cPar1 cPar2 cOpen cScope cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and M :: $'a$
and B :: $('a, 'b, 'c)\ boundOutput$
and A_P :: $name\ list$
and Ψ_P :: $'b$
and $Prop$ :: $'d::fs\text{-}name \Rightarrow 'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow$
$'a \Rightarrow ('a, 'b, 'c)\ boundOutput \Rightarrow name\ list \Rightarrow 'b \Rightarrow bool$
and C :: $'d::fs\text{-}name$

assumes *Trans*: $\Psi \rhd P \longmapsto ROut\ M\ B$
and *FrP*: $extractFrame\ P = \langle A_P, \Psi_P\rangle$
and $distinct\ A_P$
and *rAlpha*: $\bigwedge \Psi\ P\ M\ A_P\ \Psi_P\ p\ B\ C.\ [\![A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* (p \cdot A_P);\ A_P \sharp* B;\ A_P \sharp* C;$
$set\ p \subseteq set\ A_P \times set(p \cdot A_P);\ distinctPerm\ p;$
$Prop\ C\ \Psi\ P\ M\ B\ A_P\ \Psi_P]\!] \Longrightarrow Prop\ C\ \Psi\ P\ M\ B\ (p \cdot A_P)\ (p \cdot \Psi_P)$
and *rOutput*: $\bigwedge \Psi\ M\ K\ N\ P\ C.\ \Psi \vdash M \leftrightarrow K \Longrightarrow Prop\ C\ \Psi\ (M\langle N\rangle.P)\ K\ (N \prec' P)\ ([])\ (\mathbf{1})$
and *rCase*: $\bigwedge \Psi\ P\ M\ B\ \varphi\ Cs\ A_P\ \Psi_P\ C.\ [\![\Psi \rhd P \longmapsto (ROut\ M\ B);\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;\ \bigwedge C.\ Prop\ C\ \Psi\ P\ M\ B\ A_P\ \Psi_P;$
$(\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P;\ \Psi_P \simeq \mathbf{1};$
$(supp\ \Psi_P) = (\{\}::name\ set);$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* B;\ A_P \sharp* C]\!] \Longrightarrow Prop\ C\ \Psi\ (Cases\ Cs)\ M\ B\ ([])\ (\mathbf{1})$
and *rPar1*: $\bigwedge \Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_Q\ Q\ A_P\ \Psi_P\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';$
$extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ M\ ((\!|\nu{*}xvec|\!)N \prec' P')\ A_P\ \Psi_P;$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* M;\ A_P \sharp* xvec;\ A_P \sharp* N;\ A_P \sharp* P';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* M;\ A_Q \sharp* xvec;\ A_Q \sharp* N;\ A_Q \sharp* P';\ A_Q \sharp* \Psi_P;$
$xvec \sharp* \Psi;\ xvec \sharp* P;\ xvec \sharp* Q;\ xvec \sharp* M;\ xvec \sharp* \Psi_P;\ xvec \sharp* \Psi_Q;$
$A_P \sharp* C;\ A_Q \sharp* C;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ ((\!|\nu{*}xvec|\!)N \prec' (P' \parallel Q))\ (A_P@A_Q)\ (\Psi_P \otimes$

Ψ_Q)
and *rPar2*: $\bigwedge\Psi\ \Psi_P\ Q\ M\ xvec\ N\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';$
$extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ M\ ((\!|\nu{*}xvec|\!)N \prec' Q')\ A_Q\ \Psi_Q;$
$A_P \sharp{*}\ P;\ A_P \sharp{*}\ Q;\ A_P \sharp{*}\ \Psi;\ A_P \sharp{*}\ M;\ A_P \sharp{*}\ xvec;\ A_P \sharp{*}\ N;\ A_P \sharp{*}\ Q';\ A_P \sharp{*}\ A_Q;\ A_P \sharp{*}\ \Psi_Q;$
$A_Q \sharp{*}\ P;\ A_Q \sharp{*}\ Q;\ A_Q \sharp{*}\ \Psi;\ A_Q \sharp{*}\ M;\ A_Q \sharp{*}\ xvec;\ A_Q \sharp{*}\ N;\ A_Q \sharp{*}\ Q';\ A_Q \sharp{*}\ \Psi_P;$
$xvec \sharp{*}\ \Psi;\ xvec \sharp{*}\ P;\ xvec \sharp{*}\ Q;\ xvec \sharp{*}\ M;\ xvec \sharp{*}\ \Psi_P;\ xvec \sharp{*}\ \Psi_Q;$
$A_P \sharp{*}\ C;\ A_Q \sharp{*}\ C;\ xvec \sharp{*}\ C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ M\ ((\!|\nu{*}xvec|\!)N \prec' (P \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rOpen*: $\bigwedge\Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ A_P\ \Psi_P\ C.$
$[\![\Psi \rhd P \longmapsto M(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ \Psi\ P\ M\ ((\!|\nu{*}(xvec@yvec)|\!)N \prec' P')\ A_P\ \Psi_P;\ x \in supp\ N;\ x \sharp \Psi;\ x \sharp M;$
$x \sharp A_P;\ x \sharp xvec;\ x \sharp yvec;\ A_P \sharp{*}\ \Psi;\ A_P \sharp{*}\ P;\ A_P \sharp{*}\ M;\ A_P \sharp{*}\ N;\ A_P \sharp{*}\ P';$
$A_P \sharp{*}\ xvec;\ A_P \sharp{*}\ yvec;$
$xvec \sharp{*}\ \Psi;\ xvec \sharp{*}\ P;\ xvec \sharp{*}\ M;\ xvec \sharp{*}\ \Psi_P;$
$yvec \sharp{*}\ \Psi;\ yvec \sharp{*}\ P;\ yvec \sharp{*}\ M;\ A_P \sharp{*}\ C;\ x \sharp C;\ xvec \sharp{*}\ C;\ yvec \sharp{*}\ C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ M\ ((\!|\nu{*}(xvec@x\#yvec)|\!)N \prec' P')\ (x\#A_P)\ \Psi_P$
and *rScope*: $\bigwedge\Psi\ P\ M\ xvec\ N\ P'\ x\ A_P\ \Psi_P\ C.$
$[\![\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ \Psi\ P\ M\ ((\!|\nu{*}xvec|\!)N \prec' P')\ A_P\ \Psi_P;$
$x \sharp \Psi;\ x \sharp M;\ x \sharp xvec;\ x \sharp N;\ x \sharp A_P;\ A_P \sharp{*}\ \Psi;\ A_P \sharp{*}\ P;$
$A_P \sharp{*}\ M;\ A_P \sharp{*}\ N;\ A_P \sharp{*}\ P';\ A_P \sharp{*}\ xvec;$
$xvec \sharp{*}\ \Psi;\ xvec \sharp{*}\ P;\ xvec \sharp{*}\ M;\ xvec \sharp{*}\ \Psi_P;$
$A_P \sharp{*}\ C;\ x \sharp C;\ xvec \sharp{*}\ C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ ((\!|\nu x|\!)P)\ M\ ((\!|\nu{*}xvec|\!)N \prec' ((\!|\nu x|\!)P'))\ (x\#A_P)\ \Psi_P$
and *rBang*: $\bigwedge\Psi\ P\ M\ B\ A_P\ \Psi_P\ C.$
$[\![\Psi \rhd P \parallel !P \longmapsto ROut\ M\ B;\ guarded\ P;\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ M\ B\ A_P\ (\Psi_P \otimes \mathbf{1});\ \Psi_P \simeq \mathbf{1};\ supp\ \Psi_P = (\{\}::name\ set);$
$A_P \sharp{*}\ \Psi;\ A_P \sharp{*}\ P;\ A_P \sharp{*}\ M;\ A_P \sharp{*}\ C]\!] \Longrightarrow Prop\ C\ \Psi\ (!P)\ M\ B\ ([])\ (\mathbf{1})$
shows $Prop\ C\ \Psi\ P\ M\ B\ A_P\ \Psi_P$
proof −
{
fix B
assume $\Psi \rhd P \longmapsto ROut\ M\ B$
hence $Prop\ C\ \Psi\ P\ M\ B\ A_P\ \Psi_P$ **using** *FrP* ‹*distinct* A_P›
proof(*nominal-induct* $\Psi\ P\ Rs{=}{=}ROut\ M\ B\ A_P\ \Psi_P$ *avoiding*: C *arbitrary*: B

```
rule: semanticsFrameInduct)
      case cAlpha
      thus ?case by(fastforce intro: rAlpha)
    next
      case cInput
      thus ?case by(simp add: residualInject)
    next
      case cOutput
      thus ?case by(force intro: rOutput simp add: residualInject)
    next
      case cCase
      thus ?case by(force intro: rCase simp add: residualInject)
    next
      case cPar1
      thus ?case
        by(fastforce intro: rPar1 simp add: residualInject)
    next
      case cPar2
      thus ?case
        by(fastforce intro: rPar2 simp add: residualInject)
    next
      case cComm1
      thus ?case by(simp add: residualInject)
    next
      case cComm2
      thus ?case by(simp add: residualInject)
    next
      case cOpen
      thus ?case by(fastforce intro: rOpen simp add: residualInject)
    next
      case cScope
      thus ?case by(force intro: rScope simp add: residualInject)
    next
      case cBang
      thus ?case by(force intro: rBang simp add: residualInject)
    qed
  }
  with Trans show ?thesis by(simp add: residualInject)
qed

lemma tauFrameInduct[consumes 3, case-names cAlpha cCase cPar1 cPar2 cComm1
cComm2 cScope cBang]:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   P'   :: ('a, 'b, 'c) psi
  and   Prop :: 'd::fs-name ⇒ 'b ⇒ ('a, 'b, 'c) psi ⇒
                ('a, 'b, 'c) psi ⇒ name list ⇒ 'b ⇒ bool
  and   C    :: 'd::fs-name
```

assumes *Trans*: $\Psi \rhd P \longmapsto\tau \prec P'$
and *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and *rAlpha*: $\bigwedge\Psi\ P\ P'\ A_P\ \Psi_P\ p\ C.\ [\![A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* P';\ A_P \sharp* (p \cdot A_P);\ A_P \sharp* C;$
$set\ p \subseteq set\ A_P \times set\ (p \cdot A_P);\ distinctPerm\ p;$
$Prop\ C\ \Psi\ P\ P'\ A_P\ \Psi_P]\!] \Longrightarrow Prop\ C\ \Psi\ P\ P'\ (p \cdot A_P)\ (p \cdot \Psi_P)$
and *rCase*: $\bigwedge\Psi\ P\ P'\ \varphi\ Cs\ A_P\ \Psi_P\ C.\ [\![\Psi \rhd P \longmapsto\tau \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;\ \bigwedge C.\ Prop\ C\ \Psi\ P\ P'\ A_P\ \Psi_P;$
$(\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P;\ \Psi_P \simeq \mathbf{1};\ (supp\ \Psi_P) = (\{\}::name\ set);$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* P';\ A_P \sharp* C]\!] \Longrightarrow Prop\ C\ \Psi\ (Cases\ Cs)\ P'\ ([])\ (\mathbf{1})$
and *rPar1*: $\bigwedge\Psi\ \Psi_Q\ P\ P'\ A_Q\ Q\ A_P\ \Psi_P\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto\tau \prec P';$
$extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_Q)\ P\ P'\ A_P\ \Psi_P;$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* P';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* P';\ A_Q \sharp* \Psi_P;$
$A_P \sharp* C;\ A_Q \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (P' \parallel Q)\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rPar2*: $\bigwedge\Psi\ \Psi_P\ Q\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ C.$
$[\![\Psi \otimes \Psi_P \rhd Q \longmapsto\tau \prec Q';$
$extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$\bigwedge C.\ Prop\ C\ (\Psi \otimes \Psi_P)\ Q\ Q'\ A_Q\ \Psi_Q;$
$A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* \Psi;\ A_P \sharp* Q';\ A_P \sharp* A_Q;\ A_P \sharp* \Psi_Q;$
$A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* \Psi;\ A_Q \sharp* Q';\ A_Q \sharp* \Psi_P;$
$A_P \sharp* C;\ A_Q \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ (P \parallel Q')\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rComm1*: $\bigwedge\Psi\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;\ distinct\ xvec;$
$A_P \sharp* \Psi;\ A_P \sharp* \Psi_Q;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* P';$
$A_P \sharp* Q;\ A_P \sharp* Q';\ A_P \sharp* A_Q;\ A_P \sharp* xvec;\ A_Q \sharp* \Psi;\ A_Q \sharp* \Psi_P;$
$A_Q \sharp* P;\ A_Q \sharp* N;\ A_Q \sharp* P';\ A_Q \sharp* Q;\ A_Q \sharp* K;\ A_Q \sharp* Q';$
$A_Q \sharp* xvec;\ xvec \sharp* \Psi;\ xvec \sharp* \Psi_P;\ xvec \sharp* \Psi_Q;\ xvec \sharp* P;\ xvec \sharp* M;$
$xvec \sharp* Q;\ xvec \sharp* K;\ A_P \sharp* C;\ A_Q \sharp* C;\ xvec \sharp* C]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel Q)\ ((\!|\nu*xvec|\!)(P' \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rComm2*: $\bigwedge\Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ C.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;$

distinct A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$;
$A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* xvec$; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* K$; $A_Q \sharp* Q'$;
$A_Q \sharp* xvec$; $xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* \Psi_Q$; $xvec \sharp* P$; $xvec \sharp*$
M;
$xvec \sharp* Q$; $xvec \sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; $xvec \sharp* C\rrbracket \Longrightarrow$
Prop $C\ \Psi\ (P \parallel Q)\ ((\!|\nu{*}xvec|\!)(P' \parallel Q'))\ (A_P@A_Q)\ (\Psi_P \otimes \Psi_Q)$
and *rScope*: $\bigwedge\Psi\ P\ P'\ x\ A_P\ \Psi_P\ C.$
$\llbracket\Psi \rhd P \longmapsto\tau \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\bigwedge C.$ *Prop* $C\ \Psi\ P\ P'\ A_P\ \Psi_P$; $x \sharp \Psi$;
$x \sharp A_P$; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* P'$;
$A_P \sharp* C$; $x \sharp C\rrbracket \Longrightarrow$
Prop $C\ \Psi\ ((\!|\nu x|\!)P)\ ((\!|\nu x|\!)P')\ (x\#A_P)\ \Psi_P$
and *rBang*: $\bigwedge\Psi\ P\ P'\ A_P\ \Psi_P\ C.$
$\llbracket\Psi \rhd P \parallel !P \longmapsto\tau \prec P'$; *guarded* P; *extractFrame* $P = \langle A_P, \Psi_P\rangle$;
distinct A_P;
$\bigwedge C.$ *Prop* $C\ \Psi\ (P \parallel !P)\ P'\ A_P\ (\Psi_P \otimes \mathbf{1})$; $\Psi_P \simeq \mathbf{1}$; *supp* $\Psi_P =$
$(\{\}::$*name set*$)$;
$A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* P'$; $A_P \sharp* C\rrbracket \Longrightarrow$ *Prop* $C\ \Psi\ (!P)\ P'$
$([])\ (\mathbf{1})$
shows *Prop* $C\ \Psi\ P\ P'\ A_P\ \Psi_P$
using *Trans FrP* ‹*distinct* A_P›
proof(*nominal-induct* $\Psi\ P\ Rs{=}{=}\tau \prec P'\ A_P\ \Psi_P$ *avoiding*: C *arbitrary*: P' *rule*: *semanticsFrameInduct*)
case *cAlpha*
thus *?case* **by**(*force intro*: *rAlpha simp add*: *residualInject*)
next
case *cInput*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *cOutput*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *cCase*
thus *?case* **by**(*force intro*: *rCase simp add*: *residualInject*)
next
case *cPar1*
thus *?case* **by**(*force intro*: *rPar1 simp add*: *residualInject*)
next
case *cPar2*
thus *?case* **by**(*force intro*: *rPar2 simp add*: *residualInject*)
next
case *cComm1*
thus *?case* **by**(*force intro*: *rComm1 simp add*: *residualInject*)
next
case *cComm2*
thus *?case* **by**(*force intro*: *rComm2 simp add*: *residualInject*)

```
next
  case cOpen
  thus ?case by(simp add: residualInject)
next
  case cScope
  thus ?case by(force intro: rScope simp add: residualInject)
next
  case cBang
  thus ?case by(force intro: rBang simp add: residualInject)
qed

lemma inputFreshDerivative:
  fixes Ψ  :: 'b
  and   P  :: ('a, 'b, 'c) psi
  and   M  :: 'a
  and   N  :: 'a
  and   P' :: ('a, 'b, 'c) psi
  and   x  :: name

  assumes Ψ ▷ P ⟼M⦇N⦈ ≺ P'
  and     x ♯ P
  and     x ♯ N

  shows x ♯ P'
proof −
  have bn(M⦇N⦈) ♯* subject(M⦇N⦈) and distinct(bn(M⦇N⦈)) by simp+
  with ‹Ψ ▷ P ⟼M⦇N⦈ ≺ P'› show ?thesis using ‹x ♯ P› ‹x ♯ N›
  proof(nominal-induct Ψ P α==M⦇N⦈ P' avoiding: x rule: semanticsInduct)
    case(cAlpha Ψ P α P' p x)
    thus ?case by simp
  next
    case(cInput Ψ M' K xvec N' Tvec P x)
      from ‹K⦇(N'[xvec::=Tvec])⦈ = M⦇N⦈› have M = K and NeqN': N =
N'[xvec::=Tvec] by(simp add: action.inject)+
    note ‹length xvec = length Tvec› ‹distinct xvec› then
    moreover have x ♯ Tvec using ‹set xvec ⊆ supp N'› ‹x ♯ N› NeqN'
      by(blast intro: substTerm.subst3)
    moreover from ‹xvec ♯* x› ‹x ♯ M'⦇λ*xvec N'⦈.P›
   have x ♯ P by(simp add: inputChainFresh) (simp add: name-list-supp fresh-def)
    ultimately show ?case using ‹xvec ♯* x› by auto
  next
    case(cOutput Ψ M  K N P x)
    thus ?case by simp
  next
    case(cCase Ψ P P' φ Cs x)
    thus ?case by(induct Cs, auto)
  next
    case(cPar1 Ψ Ψ_Q P P' xvec Q x)
    thus ?case by simp
```

```
  next
    case(cPar2 Ψ Ψ_P Q Q' xvec P x)
    thus ?case by simp
  next
    case(cComm1 Ψ Ψ_Q P M N P' A_P Ψ_P Q K xvec Q' A_Q x)
    thus ?case by simp
  next
    case(cComm2 Ψ Ψ_Q P M xwec N P' A_P Ψ_P Q K Q' A_Q x)
    thus ?case by simp
  next
    case(cOpen Ψ P M xvec yvec N P' x y)
    thus ?case by simp
  next
    case(cScope Ψ P P' x y)
    thus ?case by(simp add: abs-fresh)
  next
    case(cBang Ψ P P' x)
    thus ?case by simp
  qed
qed

lemma inputFreshChainDerivative:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   M    :: 'a
  and   N    :: 'a
  and   P'   :: ('a, 'b, 'c) psi
  and   xvec :: name list

  assumes Ψ ▷ P ⟼M⦇N⦈ ≺ P'
  and     xvec ♯* P
  and     xvec ♯* N

  shows xvec ♯* P'
using assms
by(induct xvec)
  (auto intro: inputFreshDerivative)

lemma outputFreshDerivative:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   M    :: 'a
  and   xvec :: name list
  and   N    :: 'a
  and   P'   :: ('a, 'b, 'c) psi
  and   x    :: name

  assumes Ψ ▷ P ⟼M⦇ν*xvec⦈⟨N⟩ ≺ P'
  and     xvec ♯* M
```

```
  and     distinct xvec
  and     x ♯ P
  and     x ♯ xvec

  shows x ♯ N
  and   x ♯ P′
proof −
  note ‹Ψ ▷ P ⟼M⦇ν∗xvec⦈⟨N⟩ ≺ P′›
 moreover from ‹xvec ♯∗ M› have bn(M⦇ν∗xvec⦈⟨N⟩) ♯∗ subject(M⦇ν∗xvec⦈⟨N⟩)
by simp
  moreover from ‹distinct xvec› have distinct(bn(M⦇ν∗xvec⦈⟨N⟩)) by simp
  ultimately show x ♯ N using ‹x ♯ P› ‹x ♯ xvec›
  proof(nominal-induct Ψ P α==M⦇ν∗xvec⦈⟨N⟩ P′ avoiding: x arbitrary: M xvec
N rule: semanticsInduct)
    case(cAlpha Ψ P α P′ p x M xvec N)
    have S: set p ⊆ set(bn α) × set(bn(p · α)) by fact
     from ‹(p · α) = M⦇ν∗xvec⦈⟨N⟩› have (p · p · α) = p · (M⦇ν∗xvec⦈⟨N⟩)
by(simp add: fresh-star-bij)
    with ‹distinctPerm p› have α  = (p · M)⦇ν∗(p · xvec)⦈⟨(p · N)⟩ by simp
    moreover from ‹(p · α) = M⦇ν∗xvec⦈⟨N⟩› ‹x ♯ xvec› have x ♯ (bn(p · α)) by
simp
    with ‹(bn α) ♯∗ x› ‹x ♯ xvec› S have x ♯ (p · xvec)
       by(drule-tac pt-fresh-bij1[OF pt-name-inst, OF at-name-inst, where pi=p
and x=xvec]) simp
    ultimately have x ♯ (p · N) using ‹x ♯ P› by(rule-tac cAlpha)
     hence (p · x) ♯ (p · p · N) by(simp add: pt-fresh-bij1[OF pt-name-inst, OF
at-name-inst])
    with ‹distinctPerm p› ‹bn(α) ♯∗ x› ‹x ♯ (bn(p · α))›S show ?case by simp
  next
    case cInput
    thus ?case by simp
  next
    case cOutput
    thus ?case by(simp add: action.inject)
  next
    case cCase
    thus ?case
      by(rule-tac cCase) (auto dest: memFresh)
  next
    case cPar1
    thus ?case by simp
  next
    case cPar2
    thus ?case by simp
  next
    case cComm1
    thus ?case by simp
  next
    case cComm2
```

thus *?case* **by** *simp*
next
case(*cOpen* Ψ P M *xvec yvec* N P' x y M' *zvec* N')
from ‹$M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle = M'(\!|\nu*zvec|\!)\langle N'\rangle$› **have** $zvec = xvec@x\#yvec$ **and** $N = N'$
by(*simp add*: *action.inject*)+
from ‹$y \sharp (\!|\nu x|\!)P$› ‹$x \sharp y$› **have** $y \sharp P$ **by**(*simp add*: *abs-fresh*)
moreover from ‹$y \sharp zvec$› ‹$zvec = xvec@x\#yvec$›**have** $y \sharp (xvec@yvec)$
by *simp*
ultimately have $y \sharp N$ **by**(*rule-tac cOpen*) *auto*
with ‹$N = N'$› **show** *?case* **by** *simp*
next
case *cScope*
thus *?case* **by**(*auto simp add*: *abs-fresh*)
next
case *cBang*
thus *?case* **by** *simp*
qed
next
note ‹$\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$›
moreover from ‹$xvec \sharp* M$› **have** $bn(M(\!|\nu*xvec|\!)\langle N\rangle) \sharp* subject(M(\!|\nu*xvec|\!)\langle N\rangle)$ **by** *simp*
moreover from ‹*distinct xvec*› **have** $distinct(bn(M(\!|\nu*xvec|\!)\langle N\rangle))$ **by** *simp*
ultimately show $x \sharp P'$ **using** ‹$x \sharp P$› ‹$x \sharp xvec$›
proof(*nominal-induct* Ψ P α==$M(\!|\nu*xvec|\!)\langle N\rangle$ P' *avoiding*: x *arbitrary*: M *xvec* N *rule*: *semanticsInduct*)
case(*cAlpha* Ψ P α P' p x M *xvec* N)
have S: $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **by** *fact*
from ‹$(p \cdot \alpha) = M(\!|\nu*xvec|\!)\langle N\rangle$› **have** $(p \cdot p \cdot \alpha) = p \cdot (M(\!|\nu*xvec|\!)\langle N\rangle)$ **by**(*simp add*: *fresh-star-bij*)
with ‹*distinctPerm p*› **have** $\alpha = (p \cdot M)(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle$ **by** *simp*
moreover from ‹$(p \cdot \alpha) = M(\!|\nu*xvec|\!)\langle N\rangle$› ‹$x \sharp xvec$› **have** $x \sharp (bn(p \cdot \alpha))$ **by** *simp*
with ‹$(bn\ \alpha) \sharp* x$› ‹$x \sharp xvec$› S **have** $x \sharp (p \cdot xvec)$
by(*drule-tac pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*, **where** *pi*=*p* **and** *x*=*xvec*]) *simp*
ultimately have $x \sharp P'$ **using** ‹$x \sharp P$› **by**(*rule-tac cAlpha*)
hence $(p \cdot x) \sharp (p \cdot P')$ **by**(*simp add*: *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› ‹$bn(\alpha) \sharp* x$› ‹$x \sharp (bn(p \cdot \alpha))$›$S$ **show** *?case* **by** *simp*
next
case *cInput*
thus *?case* **by** *simp*
next
case *cOutput*
thus *?case* **by**(*simp add*: *action.inject*)
next
case *cCase*
thus *?case* **by**(*fastforce simp add*: *action.inject dest*: *memFresh*)

```
  next
    case cPar1
    thus ?case by simp
  next
    case cPar2
    thus ?case by simp
  next
    case cComm1
    thus ?case by simp
  next
    case cComm2
    thus ?case by simp
  next
    case(cOpen Ψ P M xvec yvec N P' x y M' zvec N')
   from ‹M⦇ν*(xvec@x#yvec)⦈⟨N⟩ = M'⦇ν*zvec⦈⟨N'⟩› have zvec = xvec@x#yvec

      by(simp add: action.inject)
    from ‹y ♯ ⦇νx⦈P› ‹x ♯ y›  have y ♯ P by(simp add: abs-fresh)
    moreover from ‹y ♯ zvec› ‹zvec = xvec@x#yvec›have y ♯ (xvec@yvec)
      by simp
    ultimately show y ♯ P' by(rule-tac cOpen) auto
  next
    case cScope
    thus ?case by(auto simp add: abs-fresh)
  next
    case cBang
    thus ?case by simp
  qed
qed

lemma outputFreshChainDerivative:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   M    :: 'a
  and   xvec :: name list
  and   N    :: 'a
  and   P'   :: ('a, 'b, 'c) psi
  and   yvec :: name list

  assumes Ψ ▷ P ⟼M⦇ν*xvec⦈⟨N⟩ ≺ P'
  and     xvec ♯* M
  and     distinct xvec
  and     yvec ♯* P
  and     yvec ♯* xvec

  shows yvec ♯* N
  and   yvec ♯* P'
using assms
by(induct yvec) (auto intro: outputFreshDerivative)
```

lemma *tauFreshDerivative*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*
  and x :: *name*

  assumes $\Psi \rhd P \longmapsto\tau \prec P'$
  and $x \sharp P$

  shows $x \sharp P'$
proof −
  have $bn(\tau) \sharp* subject(\tau)$ **and** $distinct(bn(\tau))$ **by** *simp*+
  with ‹$\Psi \rhd P \longmapsto\tau \prec P'$› **show** *?thesis* **using** ‹$x \sharp P$›
  proof(*nominal-induct* Ψ P α==(τ::($'a$ *action*)) P' *avoiding*: x *rule*: *semanticsInduct*)
    case *cAlpha*
    thus *?case* **by** *simp*
  next
    case *cInput*
    thus *?case* **by** *simp*
  next
    case *cOutput*
    thus *?case* **by** *simp*
  next
    case *cCase*
    thus *?case* **by**(*auto dest*: *memFresh*)
  next
    case *cPar1*
    thus *?case* **by** *simp*
  next
    case *cPar2*
    thus *?case* **by** *simp*
  next
    case *cComm1*
    thus *?case*
    by(*fastforce dest*: *inputFreshDerivative outputFreshDerivative simp add*: *resChainFresh*)
  next
    case *cComm2*
    thus *?case*
    by(*fastforce dest*: *inputFreshDerivative outputFreshDerivative simp add*: *resChainFresh*)
  next
    case *cOpen*
    thus *?case* **by** *simp*
  next
    case *cScope*
    thus *?case* **by**(*simp add*: *abs-fresh*)

```
  next
    case cBang
    thus ?case by simp
  qed
qed

lemma tauFreshChainDerivative:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   M    :: 'a
  and   N    :: 'a
  and   P'   :: ('a, 'b, 'c) psi
  and   xvec :: name list

  assumes Ψ ▷ P ⟼τ ≺ P'
  and     xvec ♯* P

  shows xvec ♯* P'
using assms
by(induct xvec) (auto intro: tauFreshDerivative)

lemma freeFreshDerivative:
  fixes Ψ  :: 'b
  and   P  :: ('a, 'b, 'c) psi
  and   α  :: 'a action
  and   P' :: ('a, 'b, 'c) psi
  and   x  :: name

  assumes Ψ ▷ P ⟼α ≺ P'
  and     bn α ♯* subject α
  and     distinct(bn α)
  and     x ♯ α
  and     x ♯ P

  shows   x ♯ P'
using assms
by(rule-tac actionCases[where α=α])
  (auto intro: inputFreshDerivative tauFreshDerivative outputFreshDerivative)

lemma freeFreshChainDerivative:
  fixes Ψ    :: 'b
  and   P    :: ('a, 'b, 'c) psi
  and   α    :: 'a action
  and   P'   :: ('a, 'b, 'c) psi
  and   xvec :: name list

  assumes Ψ ▷ P ⟼α ≺ P'
  and     bn α ♯* subject α
  and     distinct(bn α)
```

and $xvec \sharp* P$
and $xvec \sharp* \alpha$

shows $xvec \sharp* P'$
using *assms*
by(*auto intro*: *freeFreshDerivative simp add*: *fresh-star-def*)

lemma *Input*:
fixes Ψ :: $'b$
and M :: $'a$
and K :: $'a$
and $xvec$ :: *name list*
and N :: $'a$
and $Tvec$ :: $'a$ *list*

assumes $\Psi \vdash M \leftrightarrow K$
and *distinct xvec*
and $set\ xvec \subseteq supp\ N$
and $length\ xvec = length\ Tvec$

shows $\Psi \rhd M(\!|\lambda*xvec\ N|\!).P \longmapsto K(\!|N[xvec::=Tvec]|\!) \prec P[xvec::=Tvec]$
proof −
obtain p **where** *xvecFreshPsi*: $((p::name\ prm) \cdot (xvec::name\ list)) \sharp* \Psi$
and *xvecFreshM*: $(p \cdot xvec) \sharp* M$
and *xvecFreshN*: $(p \cdot xvec) \sharp* N$
and *xvecFreshK*: $(p \cdot xvec) \sharp* K$
and *xvecFreshTvec*: $(p \cdot xvec) \sharp* Tvec$
and *xvecFreshP*: $(p \cdot xvec) \sharp* P$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$
and *dp*: *distinctPerm p*
by(*rule-tac xvec=xvec* **and** $c=(\Psi, M, K, N, P, Tvec)$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
note ‹$\Psi \vdash M \leftrightarrow K$›
moreover from ‹*distinct xvec*› **have** $distinct(p \cdot xvec)$
by *simp*
moreover from ‹$(set\ xvec) \subseteq (supp\ N)$› **have** $(p \cdot (set\ xvec)) \subseteq (p \cdot (supp\ N))$
by *simp*
hence $set(p \cdot xvec) \subseteq supp(p \cdot N)$
by(*simp add*: *eqvts*)
moreover from ‹$length\ xvec = length\ Tvec$› **have** $length(p \cdot xvec) = length\ Tvec$
by *simp*
ultimately have $\Psi \rhd M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P) \longmapsto K(\!|(p \cdot N)[(p \cdot xvec)::=Tvec]|\!) \prec (p \cdot P)[(p \cdot xvec)::=Tvec]$
using *xvecFreshPsi xvecFreshM xvecFreshK xvecFreshTvec*
by(*rule-tac cInput*)
thus *?thesis* **using** *xvecFreshN xvecFreshP S* ‹$length\ xvec = length\ Tvec$› *dp*
by(*auto simp add*: *inputChainAlpha′ substTerm.renaming renaming*)
qed

lemma *residualAlpha*:
  fixes p :: *name prm*
  and α :: *'a action*
  and P :: *('a, 'b, 'c) psi*

  assumes $bn(p \cdot \alpha) \sharp* object\ \alpha$
  and $bn(p \cdot \alpha) \sharp* P$
  and $bn\ \alpha \sharp* subject\ \alpha$
  and $bn(p \cdot \alpha) \sharp* subject\ \alpha$
  and $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$

  shows $\alpha \prec P = (p \cdot \alpha) \prec (p \cdot P)$
using *assms*
apply(*rule-tac* α=α **in** *actionCases*)
apply(*simp only*: *eqvts bn.simps*)
apply *simp*
apply(*simp add*: *boundOutputChainAlpha'' residualInject*)
by *simp*

lemma *Par1*:
  fixes Ψ :: *'b*
  and Ψ_Q :: *'b*
  and P :: *('a, 'b, 'c) psi*
  and α :: *'a action*
  and P' :: *('a, 'b, 'c) psi*
  and A_Q :: *name list*
  and Q :: *('a, 'b, 'c) psi*

  assumes *Trans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$
  and $extractFrame\ Q = \langle A_Q, \Psi_Q \rangle$
  and $bn\ \alpha \sharp* Q$
  and $A_Q \sharp* \Psi$
  and $A_Q \sharp* P$
  and $A_Q \sharp* \alpha$

  shows $\Psi \rhd P \parallel Q \longmapsto\alpha \prec (P' \parallel Q)$
proof −
  {
    fix Ψ :: *'b*
    and Ψ_Q :: *'b*
    and P :: *('a, 'b, 'c) psi*
    and α :: *'a action*
    and P' :: *('a, 'b, 'c) psi*
    and A_Q :: *name list*
    and Q :: *('a, 'b, 'c) psi*

    assume $\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$
    and $extractFrame\ Q = \langle A_Q, \Psi_Q \rangle$
    and $bn\ \alpha \sharp* Q$

and $bn\ \alpha \sharp* subject\ \alpha$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* \alpha$
and $distinct\ A_Q$

have $\Psi \rhd P \parallel Q \longmapsto\alpha \prec (P' \parallel Q)$
proof −
from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$› **have** $distinct(bn\ \alpha)$ **by**(*rule boundOutputDistinct*)
obtain q::*name prm* **where** $bn(q \cdot \alpha) \sharp* \Psi$ **and** $bn(q \cdot \alpha) \sharp* P$ **and** $bn(q \cdot \alpha) \sharp* Q$ **and** $bn(q \cdot \alpha) \sharp* \alpha$
and $bn(q \cdot \alpha) \sharp* A_Q$ **and** $bn(q \cdot \alpha) \sharp* P'$ **and** $bn(q \cdot \alpha) \sharp* \Psi_Q$
and Sq: $(set\ q) \subseteq (set\ (bn\ \alpha)) \times (set(bn(q \cdot \alpha)))$
by(*rule-tac xvec*=$bn\ \alpha$ **and** c=(Ψ, P, Q, α, A_Q, Ψ_Q, P') **in** *name-list-avoiding*) (*auto simp add*: *eqvts*)
obtain p::*name prm* **where** $(p \cdot A_Q) \sharp* \Psi$ **and** $(p \cdot A_Q) \sharp* P$ **and** $(p \cdot A_Q) \sharp* Q$ **and** $(p \cdot A_Q) \sharp* \alpha$
and $(p \cdot A_Q) \sharp* \alpha$ **and** $(p \cdot A_Q) \sharp* (q \cdot \alpha)$ **and** $(p \cdot A_Q) \sharp* P'$
and $(p \cdot A_Q) \sharp* (q \cdot P')$ **and** $(p \cdot A_Q) \sharp* \Psi_Q$ **and** Sp: $(set\ p) \subseteq (set\ A_Q) \times (set(p \cdot A_Q))$
by(*rule-tac xvec*=A_Q **and** c=(Ψ, P, Q, α, $bn\ \alpha$, $q \cdot \alpha$, P', $(q \cdot P')$, Ψ_Q) **in** *name-list-avoiding*) *auto*
from ‹$distinct(bn\ \alpha)$› **have** $distinct(bn(q \cdot \alpha))$
by(*rule-tac* α=α **in** *actionCases*) (*auto simp add*: *eqvts*)
from ‹$A_Q \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* A_Q$› Sq **have** $A_Q \sharp* (q \cdot \alpha)$
apply(*rule-tac* α=α **in** *actionCases*)
apply(*simp only*: *bn.simps eqvts*, *simp*)
apply(*simp add*: *freshChainSimps*)
by *simp*
from ‹$bn\ \alpha \sharp* subject\ \alpha$› **have** $(q \cdot (bn\ \alpha)) \sharp* (q \cdot (subject\ \alpha))$
by(*simp add*: *fresh-star-bij*)
hence $bn(q \cdot \alpha) \sharp* subject(q \cdot \alpha)$ **by**(*simp add*: *eqvts*)
from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$› ‹$bn(q \cdot \alpha) \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* P'$› ‹$bn\ \alpha \sharp* (subject\ \alpha)$› Sq
have $Trans$: $\Psi \otimes \Psi_Q \rhd P \longmapsto(q \cdot \alpha) \prec (q \cdot P')$
by(*force simp add*: *residualAlpha*)
hence $A_Q \sharp* (q \cdot P')$ **using** ‹$bn(q \cdot \alpha) \sharp* subject(q \cdot \alpha)$› ‹$distinct(bn(q \cdot \alpha))$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* (q \cdot \alpha)$›
by(*auto intro*: *freeFreshChainDerivative*)
from $Trans$ **have** $(p \cdot (\Psi \otimes \Psi_Q)) \rhd (p \cdot P) \longmapsto p \cdot ((q \cdot \alpha) \prec (q \cdot P'))$
by(*rule semantics.eqvt*)
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* (q \cdot \alpha)$› ‹$(p \cdot A_Q) \sharp* (q \cdot \alpha)$› ‹$A_Q \sharp* (q \cdot P')$›
‹$(p \cdot A_Q) \sharp* \Psi$› ‹$(p \cdot A_Q) \sharp* P$› ‹$(p \cdot A_Q) \sharp* (q \cdot P')$› Sp
have $\Psi \otimes (p \cdot \Psi_Q) \rhd P \longmapsto(q \cdot \alpha) \prec (q \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from ‹$extractFrame\ Q = \langle A_Q, \Psi_Q \rangle$› ‹$(p \cdot A_Q) \sharp* \Psi_Q$› Sp **have**

extractFrame $Q = \langle(p \cdot A_Q), (p \cdot \Psi_Q)\rangle$
by(*simp add*: *frameChainAlpha′ eqvts*)
moreover from ‹$(bn(q \cdot \alpha)) \sharp* \Psi_Q$› ‹$(bn(q \cdot \alpha)) \sharp* A_Q$› ‹$(p \cdot A_Q) \sharp* (q \cdot \alpha)$› *Sp*
have $(bn(q \cdot \alpha)) \sharp* (p \cdot \Psi_Q)$
by(*simp add*: *freshAlphaPerm*)
moreover from ‹*distinct* A_Q› **have** *distinct*$(p \cdot A_Q)$ **by** *simp*
ultimately have $\Psi \rhd P \parallel Q \longmapsto (q \cdot \alpha) \prec ((q \cdot P') \parallel Q)$
using ‹$(p \cdot A_Q) \sharp* P$› ‹$(p \cdot A_Q) \sharp* Q$› ‹$(p \cdot A_Q) \sharp* \Psi$› ‹$(p \cdot A_Q) \sharp* (q \cdot \alpha)$› ‹$(p \cdot A_Q) \sharp* (q \cdot P')$› ‹$(bn(q \cdot \alpha)) \sharp* \Psi$› ‹$(bn(q \cdot \alpha)) \sharp* Q$› ‹$(bn(q \cdot \alpha)) \sharp* P$› ‹$(bn(q \cdot \alpha)) \sharp* (subject\ (q \cdot \alpha))$› ‹*distinct*$(bn(q \cdot \alpha))$›
by(*rule-tac cPar1*)

thus *?thesis* **using** ‹$bn(q \cdot \alpha) \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* P'$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn(q \cdot \alpha) \sharp* Q$› ‹$bn\ \alpha \sharp* Q$› *Sq*
by(*force simp add*: *residualAlpha*)
qed
}
note *Goal* = *this*
from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$›
obtain $A_Q{}'$ **where** *FrQ*: *extractFrame* $Q = \langle A_Q{}', \Psi_Q\rangle$ **and** *distinct* $A_Q{}'$ **and** $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* \alpha$
by(*rule-tac* C=(Ψ, P, α) **in** *distinctFrame*) *auto*
show *?thesis*
proof(*induct rule*: *actionCases*[**where** α=α])
case(*cInput M N*)
from *Trans FrQ* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P$› ‹$A_Q{}' \sharp* \alpha$› ‹*distinct* $A_Q{}'$› ‹$bn\ \alpha \sharp* Q$›
show *?case* **using** ‹$\alpha = M(\!|N|\!)$› **by**(*force intro*: *Goal*)
next
case *cTau*
from *Trans FrQ* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P$› ‹$A_Q{}' \sharp* \alpha$› ‹*distinct* $A_Q{}'$› ‹$bn\ \alpha \sharp* Q$›
show *?case* **using** ‹$\alpha = \tau$› **by**(*force intro*: *Goal*)
next
case(*cOutput M xvec N*)
from ‹$\alpha = M(\!|\nu{*}xvec|\!)\langle N\rangle$› ‹$A_Q{}' \sharp* \alpha$› ‹$bn\ \alpha \sharp* Q$› **have** $xvec \sharp* A_Q{}'$ **and** $xvec \sharp* Q$
by *simp+*
obtain p **where** $(p \cdot xvec) \sharp* N$ **and** $(p \cdot xvec) \sharp* P'$ **and** $(p \cdot xvec) \sharp* Q$
and $(p \cdot xvec) \sharp* M$ **and** $(p \cdot xvec) \sharp* A_Q{}'$
and S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$
by(*rule-tac xvec*=*xvec* **and** c=(N, P', Q, M, $A_Q{}'$) **in** *name-list-avoiding*) *auto*
from *Trans* ‹α=$M(\!|\nu{*}xvec|\!)\langle N\rangle$› **have** $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by** *simp*
with ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* P'$› S
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot P')$
by(*simp add*: *boundOutputChainAlpha″ create-residual.simps*)
moreover from ‹$xvec \sharp* A_Q{}'$› ‹$(p \cdot xvec) \sharp* A_Q{}'$› ‹$A_Q{}' \sharp* \alpha$› S
have $A_Q{}' \sharp* (p \cdot \alpha)$ **by**(*simp add*: *freshChainSimps del*: *actionFreshChain*)

ultimately have $\Psi \rhd P \parallel Q \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot P') \parallel Q$
using *FrQ* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P$› ‹*distinct* $A_Q{}'$› ‹$(p \cdot xvec) \sharp* Q$› ‹$A_Q{}' \sharp* \alpha$› ‹$(p \cdot xvec) \sharp* M$› ‹$\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$›
by(*force intro*: *Goal*)
with ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* P'$› ‹$(p \cdot xvec) \sharp* Q$› ‹$xvec \sharp* Q$› S ‹$\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$›
show *?case*
by(*simp add*: *boundOutputChainAlpha''* *eqvts* *create-residual.simps*)
qed
qed

lemma *Par2*:
fixes Ψ :: $'b$
and Ψ_P :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and Q' :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*
and P :: $('a, 'b, 'c)$ *psi*

assumes *Trans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *bn* $\alpha \sharp* P$
and $A_P \sharp* \Psi$
and $A_P \sharp* Q$
and $A_P \sharp* \alpha$

shows $\Psi \rhd P \parallel Q \longmapsto \alpha \prec (P \parallel Q')$
proof −
{
fix Ψ :: $'b$
and Ψ_P :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and Q' :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*
and P :: $('a, 'b, 'c)$ *psi*

assume $\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *bn* $\alpha \sharp* P$
and *bn* $\alpha \sharp*$ *subject* α
and $A_P \sharp* \Psi$
and $A_P \sharp* Q$
and $A_P \sharp* \alpha$
and *distinct* A_P

have $\Psi \rhd P \parallel Q \longmapsto \alpha \prec (P \parallel Q')$
proof −

from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$› **have** *distinct*($bn\ \alpha$) **by**(*rule boundOutputDistinct*)

obtain q::*name prm* **where** $bn(q \cdot \alpha) \sharp* \Psi$ **and** $bn(q \cdot \alpha) \sharp* P$ **and** $bn(q \cdot \alpha) \sharp* Q$ **and** $bn(q \cdot \alpha) \sharp* \alpha$

and $bn(q \cdot \alpha) \sharp* A_P$ **and** $bn(q \cdot \alpha) \sharp* Q'$ **and** $bn(q \cdot \alpha) \sharp* \Psi_P$

and *Sq*: $(set\ q) \subseteq (set\ (bn\ \alpha)) \times (set(bn(q \cdot \alpha)))$

by(*rule-tac xvec*=$bn\ \alpha$ **and** c=(Ψ, P, Q, α, A_P, Ψ_P, Q') **in** *name-list-avoiding*) (*auto simp add*: *eqvts*)

obtain p::*name prm* **where** $(p \cdot A_P) \sharp* \Psi$ **and** $(p \cdot A_P) \sharp* P$ **and** $(p \cdot A_P) \sharp* Q$ **and** $(p \cdot A_P) \sharp* \alpha$

and $(p \cdot A_P) \sharp* \alpha$ **and** $(p \cdot A_P) \sharp* (q \cdot \alpha)$ **and** $(p \cdot A_P) \sharp* Q'$

and $(p \cdot A_P) \sharp* (q \cdot Q')$ **and** $(p \cdot A_P) \sharp* \Psi_P$

and *Sp*: $(set\ p) \subseteq (set\ A_P) \times (set(p \cdot A_P))$

by(*rule-tac xvec*=A_P **and** c=(Ψ, P, Q, α, $q \cdot \alpha$, Q', $(q \cdot Q')$, Ψ_P) **in** *name-list-avoiding*) *auto*

from ‹*distinct*($bn\ \alpha$)› **have** *distinct*($bn(q \cdot \alpha)$)

by(*rule-tac* α=α **in** *actionCases*) (*auto simp add*: *eqvts*)

from ‹$A_P \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* A_P$› *Sq* **have** $A_P \sharp* (q \cdot \alpha)$

apply(*rule-tac* α=α **in** *actionCases*)

apply(*simp only*: *bn.simps eqvts*, *simp*)

apply(*simp add*: *freshChainSimps*)

by *simp*

from ‹$bn\ \alpha \sharp* subject\ \alpha$› **have** $(q \cdot (bn\ \alpha)) \sharp* (q \cdot (subject\ \alpha))$

by(*simp add*: *fresh-star-bij*)

hence $bn(q \cdot \alpha) \sharp* subject(q \cdot \alpha)$ **by**(*simp add*: *eqvts*)

from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$› ‹$bn(q \cdot \alpha) \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* Q'$› ‹$bn\ \alpha \sharp* (subject\ \alpha)$› *Sq*

have *Trans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto (q \cdot \alpha) \prec (q \cdot Q')$

by(*force simp add*: *residualAlpha*)

hence $A_P \sharp* (q \cdot Q')$ **using** ‹$bn(q \cdot \alpha) \sharp* subject(q \cdot \alpha)$› ‹*distinct*($bn(q \cdot \alpha)$)› ‹$A_P \sharp* Q$› ‹$A_P \sharp* (q \cdot \alpha)$›

by(*auto intro*: *freeFreshChainDerivative*)

from *Trans* **have** $(p \cdot (\Psi \otimes \Psi_P)) \rhd (p \cdot Q) \longmapsto p \cdot ((q \cdot \alpha) \prec (q \cdot Q'))$

by(*rule semantics.eqvt*)

with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* (q \cdot \alpha)$› ‹$(p \cdot A_P) \sharp* (q \cdot \alpha)$› ‹$A_P \sharp* (q \cdot Q')$›

‹$(p \cdot A_P) \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* Q$› ‹$(p \cdot A_P) \sharp* (q \cdot Q')$› *Sp*

have $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto (q \cdot \alpha) \prec (q \cdot Q')$ **by**(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› ‹$(p \cdot A_P) \sharp* \Psi_P$› *Sp* **have** *extractFrame* $P = \langle (p \cdot A_P), (p \cdot \Psi_P) \rangle$

by(*simp add*: *frameChainAlpha′ eqvts*)

moreover from ‹$(bn(q \cdot \alpha)) \sharp* \Psi_P$› ‹$(bn(q \cdot \alpha)) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (q \cdot \alpha)$› *Sp*

have $(bn(q \cdot \alpha)) \sharp* (p \cdot \Psi_P)$

by(*simp add*: *freshAlphaPerm*)

moreover from ‹*distinct* A_P› **have** *distinct*($p \cdot A_P$) **by** *simp*

ultimately have $\Psi \rhd P \parallel Q \longmapsto (q \cdot \alpha) \prec (P \parallel (q \cdot Q'))$

using ‹$(p \cdot A_P) \sharp* P$› ‹$(p \cdot A_P) \sharp* Q$› ‹$(p \cdot A_P) \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* (q \cdot \alpha)$› ‹$(p \cdot A_P) \sharp* (q \cdot Q')$› ‹$(bn(q \cdot \alpha)) \sharp* \Psi$› ‹$(bn(q \cdot \alpha)) \sharp* Q$› ‹$(bn(q \cdot \alpha)) \sharp* P$›
‹$(bn(q \cdot \alpha)) \sharp* (subject\ (q \cdot \alpha))$› ‹$distinct(bn(q \cdot \alpha))$›
by(*rule-tac cPar2*)

thus *?thesis* **using** ‹$bn(q \cdot \alpha) \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* Q'$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn(q \cdot \alpha) \sharp* P$› ‹$bn\ \alpha \sharp* P$› *Sq*
by(*force simp add*: *residualAlpha*)
qed
}
note *Goal* = *this*
from ‹$extractFrame\ P = \langle A_P, \Psi_P \rangle$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \alpha$›
obtain $A_P{}'$ **where** *FrP*: $extractFrame\ P = \langle A_P{}', \Psi_P \rangle$ **and** $distinct\ A_P{}'$ **and** $A_P{}' \sharp* \Psi$ **and** $A_P{}' \sharp* Q$ **and** $A_P{}' \sharp* \alpha$
by(*rule-tac* $C=(\Psi, Q, \alpha)$ **in** *distinctFrame*) *auto*
show *?thesis*
proof(*induct rule*: *actionCases*[**where** $\alpha=\alpha$])
case(*cInput M N*)
from *Trans FrP* ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* Q$› ‹$A_P{}' \sharp* \alpha$› ‹$distinct\ A_P{}'$› ‹$bn\ \alpha \sharp* P$›
show *?case* **using** ‹$\alpha = M(\!|N|\!)$› **by**(*force intro*: *Goal*)
next
case *cTau*
from *Trans FrP* ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* Q$› ‹$A_P{}' \sharp* \alpha$› ‹$distinct\ A_P{}'$› ‹$bn\ \alpha \sharp* P$›
show *?case* **using** ‹$\alpha = \tau$› **by**(*force intro*: *Goal*)
next
case(*cOutput M xvec N*)
from ‹$\alpha = M(\!|\nu*xvec|\!)\langle N \rangle$› ‹$A_P{}' \sharp* \alpha$› ‹$bn\ \alpha \sharp* P$› **have** $xvec \sharp* A_P{}'$ **and** $xvec \sharp* P$
by *simp+*
obtain p **where** $(p \cdot xvec) \sharp* N$ **and** $(p \cdot xvec) \sharp* Q'$ **and** $(p \cdot xvec) \sharp* P$
and $(p \cdot xvec) \sharp* M$ **and** $(p \cdot xvec) \sharp* A_P{}'$
and S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$
by(*rule-tac xvec=xvec* **and** $c=(N, Q', P, M, A_P{}')$ **in** *name-list-avoiding*) *auto*
from *Trans* ‹$\alpha=M(\!|\nu*xvec|\!)\langle N \rangle$› **have** $\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N \rangle \prec Q'$ **by** *simp*
with ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* Q'$› S
have $\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle (p \cdot N) \rangle \prec (p \cdot Q')$
by(*simp add*: *boundOutputChainAlpha''* *create-residual.simps*)
moreover from ‹$xvec \sharp* A_P{}'$› ‹$(p \cdot xvec) \sharp* A_P{}'$› ‹$A_P{}' \sharp* \alpha$› S
have $A_P{}' \sharp* (p \cdot \alpha)$ **by**(*simp add*: *freshChainSimps del*: *actionFreshChain*)
ultimately have $\Psi \rhd P \parallel Q \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle (p \cdot N) \rangle \prec P \parallel (p \cdot Q')$
using *FrP* ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* Q$› ‹$distinct\ A_P{}'$› ‹$(p \cdot xvec) \sharp* P$› ‹$A_P{}' \sharp* \alpha$› ‹$(p \cdot xvec) \sharp* M$› ‹$\alpha = M(\!|\nu*xvec|\!)\langle N \rangle$›
by(*force intro*: *Goal*)
with ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* Q'$› ‹$(p \cdot xvec) \sharp* P$› ‹$xvec \sharp* P$› S ‹$\alpha = M(\!|\nu*xvec|\!)\langle N \rangle$›
show *?case*
by(*simp add*: *boundOutputChainAlpha''* *eqvts* *create-residual.simps*)

qed
qed

lemma *Open*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and *yvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes *Trans*: $\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$
and $x \in supp\ N$
and $x \sharp \Psi$
and $x \sharp M$
and $x \sharp xvec$
and $x \sharp yvec$

shows $\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$
proof −
from *Trans* **have** *distinct*(*xvec@yvec*) **by**(*force dest*: *boundOutputDistinct*)
hence *xvec* $\sharp*$ *yvec* **by**(*induct xvec*) *auto*

obtain p **where** $(p \cdot yvec) \sharp* \Psi$ **and** $(p \cdot yvec) \sharp* P$ **and** $(p \cdot yvec) \sharp* M$
and $(p \cdot yvec) \sharp* yvec$ **and** $(p \cdot yvec) \sharp* N$ **and** $(p \cdot yvec) \sharp* P'$
and $x \sharp (p \cdot yvec)$ **and** $(p \cdot yvec) \sharp* xvec$
and *Sp*: $(set\ p) \subseteq (set\ yvec) \times (set(p \cdot yvec))$
by(*rule-tac xvec=yvec* **and** $c=(\Psi, P, M, xvec, yvec, N, P', x)$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain q **where** $(q \cdot xvec) \sharp* \Psi$ **and** $(q \cdot xvec) \sharp* P$ **and** $(q \cdot xvec) \sharp* M$
and $(q \cdot xvec) \sharp* xvec$ **and** $(q \cdot xvec) \sharp* N$ **and** $(q \cdot xvec) \sharp* P'$
and $x \sharp (q \cdot xvec)$ **and** $(q \cdot xvec) \sharp* yvec$
and $(q \cdot xvec) \sharp* p$ **and** $(q \cdot xvec) \sharp* (p \cdot yvec)$
and *Sq*: $(set\ q) \subseteq (set\ xvec) \times (set(q \cdot xvec))$
by(*rule-tac xvec=xvec* **and** $c=(\Psi, P, M, xvec, yvec, p \cdot yvec, N, P', x, p)$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)

note ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$›
moreover from ‹$(p \cdot yvec) \sharp* N$› ‹$(q \cdot xvec) \sharp* N$› ‹$xvec \sharp* yvec$› ‹$(q \cdot xvec) \sharp* yvec$› ‹$(q \cdot xvec) \sharp* (p \cdot yvec)$› ‹$(p \cdot yvec) \sharp* xvec$› *Sp Sq*
have $((p@q) \cdot (xvec\ @\ yvec)) \sharp* N$ **apply**(*simp only*: *eqvts*) **apply**(*simp only*: *pt2*[*OF pt-name-inst*])
by *simp*
moreover from ‹$(p \cdot yvec) \sharp* P'$› ‹$(q \cdot xvec) \sharp* P'$› ‹$xvec \sharp* yvec$› ‹$(q \cdot xvec) \sharp* yvec$› ‹$(q \cdot xvec) \sharp* (p \cdot yvec)$› ‹$(p \cdot yvec) \sharp* xvec$› *Sp Sq*
have $((p@q) \cdot (xvec\ @\ yvec)) \sharp* P'$ **by**(*simp del*: *freshAlphaPerm add*: *eqvts*

pt2[*OF pt-name-inst*])
moreover from *Sp Sq* ‹*xvec* ♯∗ *yvec*› ‹(*q* · *xvec*) ♯∗ *yvec*› ‹(*q* · *xvec*) ♯∗ (*p* · *yvec*)› ‹(*p* · *yvec*) ♯∗ *xvec*›
have *Spq*: *set*(*p*@*q*) ⊆ *set*(*xvec*@*yvec*) × *set*((*p*@*q*) · (*xvec*@*yvec*))
by(*simp add*: *pt2*[*OF pt-name-inst*] *eqvts*) *blast*
ultimately have Ψ ▷ *P* ⟼*M*⦇ν∗((*p*@*q*) · (*xvec*@*yvec*))⦈⟨((*p*@*q*) · *N*)⟩ ≺ ((*p*@*q*) · *P*′)
apply(*simp add*: *create-residual.simps*)
by(*erule-tac rev-mp*) (*subst boundOutputChainAlpha*, *auto*)

with *Sp Sq* ‹*xvec* ♯∗ *yvec*› ‹(*q* · *xvec*) ♯∗ *yvec*› ‹(*q* · *xvec*) ♯∗ (*p* · *yvec*)› ‹(*p* · *yvec*) ♯∗ *xvec*›
have Ψ ▷ *P* ⟼*M*⦇ν∗((*q* · *xvec*)@(*p* · *yvec*))⦈⟨((*p*@*q*) · *N*)⟩ ≺ ((*p*@*q*) · *P*′)
by(*simp add*: *eqvts pt2*[*OF pt-name-inst*] *del*: *freshAlphaPerm*)
moreover from ‹*x* ∈ *supp N*› **have** ((*p*@*q*) · *x*) ∈ (*p*@*q*) · (*supp N*)
by(*simp add*: *pt-set-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*x* ♯ *xvec*› ‹*x* ♯ *yvec*› ‹*x* ♯ (*q* · *xvec*)› ‹*x* ♯ (*p* · *yvec*)› *Sp Sq*
have *x* ∈ *supp*((*p*@*q*)· *N*) **by**(*simp add*: *eqvts pt2*[*OF pt-name-inst*])
moreover from ‹*distinct*(*xvec*@*yvec*)› **have** *distinct*(*q* · *xvec*) **and** *distinct*(*p* · *yvec*)
by *auto*
moreover note ‹*x* ♯ (*q* · *xvec*)› ‹*x* ♯ (*p* · *yvec*)› ‹*x* ♯ *M*› ‹*x* ♯ Ψ›
‹(*q* · *xvec*) ♯∗ Ψ› ‹(*q* · *xvec*) ♯∗ *P*› ‹(*q* · *xvec*) ♯∗ *M*› ‹(*q* · *xvec*) ♯∗ (*p* · *yvec*)›
‹(*p* · *yvec*) ♯∗ Ψ› ‹(*p* · *yvec*) ♯∗ *P*› ‹(*p* · *yvec*) ♯∗ *M*› ‹*distinct*(*q* · *xvec*)›
ultimately have Ψ ▷ ⦇ν*x*⦈*P* ⟼*M*⦇ν∗((*q* · *xvec*)@*x*#(*p* · *yvec*))⦈⟨((*p*@*q*) · *N*)⟩ ≺ ((*p*@*q*) · *P*′)
by(*rule-tac cOpen*)
with ‹*x* ♯ *xvec*› ‹*x* ♯ *yvec*› ‹*x* ♯ (*q* · *xvec*)› ‹*x* ♯ (*p* · *yvec*)›
‹*xvec* ♯∗ *yvec*› ‹(*q* · *xvec*) ♯∗ *yvec*› ‹(*q* · *xvec*) ♯∗ (*p* · *yvec*)› ‹(*p* · *yvec*) ♯∗ *xvec*› *Sp Sq*
have Ψ ▷ ⦇ν*x*⦈*P* ⟼*M*⦇ν∗((*p*@*q*) · (*xvec*@*x*#*yvec*))⦈⟨((*p*@*q*) · *N*)⟩ ≺ ((*p*@*q*) · *P*′)
by(*simp add*: *eqvts pt2*[*OF pt-name-inst*] *del*: *freshAlphaPerm*)
thus *?thesis* **using** ‹((*p*@*q*) · (*xvec* @ *yvec*)) ♯∗ *N*› ‹((*p*@*q*) · (*xvec* @ *yvec*)) ♯∗ *P*′› *Spq*
apply(*simp add*: *create-residual.simps*)
by(*erule-tac rev-mp*) (*subst boundOutputChainAlpha*, *auto*)
qed

lemma *Scope*:
fixes Ψ :: ′*b*
and *P* :: (′*a*, ′*b*, ′*c*) *psi*
and α :: ′*a action*
and *P*′ :: (′*a*, ′*b*, ′*c*) *psi*
and *x* :: *name*

assumes Ψ ▷ *P* ⟼α ≺ *P*′
and *x* ♯ Ψ

and $x \sharp \alpha$

shows $\Psi \rhd (\!|\nu x|\!)P \longmapsto\alpha \prec (\!|\nu x|\!)P'$
proof −
{
fix Ψ P M $xvec$ N P' x

assume $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $(x{::}name) \sharp \Psi$
and $x \sharp M$
and $x \sharp xvec$
and $x \sharp N$

obtain $p{::}name$ prm **where** $(p \cdot xvec) \sharp{*} \Psi$ **and** $(p \cdot xvec) \sharp{*} P$ **and** $(p \cdot xvec) \sharp{*} M$ **and** $(p \cdot xvec) \sharp{*} xvec$
and $(p \cdot xvec) \sharp{*} N$ **and** $(p \cdot xvec) \sharp{*} P'$ **and** $x \sharp (p \cdot xvec)$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$
by(*rule-tac xvec=xvec* **and** c=(Ψ, P, M, $xvec$, N, P', x) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
from ‹$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹$(p \cdot xvec) \sharp{*} N$› ‹$(p \cdot xvec) \sharp{*} P'$› S
have $\Psi \rhd P \longmapsto M(\!|\nu{*}(p \cdot xvec)|\!)\langle (p \cdot N)\rangle \prec (p \cdot P')$
by(*simp add*: *boundOutputChainAlpha''* *create-residual.simps*)
moreover hence *distinct*$(p \cdot xvec)$ **by**(*force dest*: *boundOutputDistinct*)
moreover note ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp (p \cdot xvec)$›
moreover from ‹$x \sharp xvec$› ‹$x \sharp p \cdot xvec$› ‹$x \sharp N$› S **have** $x \sharp (p \cdot N)$
by(*simp add*: *fresh-left del*: *freshAlphaSwap*)
ultimately have $\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu{*}(p \cdot xvec)|\!)\langle (p \cdot N)\rangle \prec (\!|\nu x|\!)(p \cdot P')$ **using** ‹$(p \cdot xvec) \sharp{*} \Psi$› ‹$(p \cdot xvec) \sharp{*} P$› ‹$(p \cdot xvec) \sharp{*} M$›
by(*rule-tac cScope*) *auto*
moreover from ‹$x \sharp xvec$› ‹$x \sharp p \cdot xvec$› S **have** $p \cdot x = x$ **by** *simp*
ultimately have $\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu{*}(p \cdot xvec)|\!)\langle (p \cdot N)\rangle \prec (p \cdot ((\!|\nu x|\!)P'))$ **by** *simp*
moreover from ‹$(p \cdot xvec) \sharp{*} P'$› ‹$x \sharp xvec$› ‹$x \sharp (p \cdot xvec)$› **have** $(p \cdot xvec) \sharp{*} (\!|\nu x|\!)P'$
by(*simp add*: *abs-fresh-star*)
ultimately have$\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec (\!|\nu x|\!)P'$ **using** ‹$(p \cdot xvec) \sharp{*} N$› S
by(*simp add*: *boundOutputChainAlpha''* *create-residual.simps*)
}
note *Goal* = *this*
show *?thesis*
proof(*induct rule*: *actionCases*[**where** α=α])
case(*cInput M N*)
with *assms* **show** *?case* **by**(*force intro*: *cScope*)
next
case(*cOutput M xvec N*)
with *assms* **show** *?case* **by**(*force intro*: *Goal*)
next
case *cTau*

with *assms* **show** *?case* **by**(*force intro*: *cScope*)
qed
qed

lemma *inputSwapFrameSubject*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*
and y :: *name*

assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and $x \sharp P$
and $y \sharp P$

shows $([(x, y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$
using *assms*
proof(*nominal-induct avoiding*: *x y rule*: *inputInduct*)
case(*cInput* Ψ *M K xvec N Tvec P x y*)
from ‹$x \sharp M(\!|\lambda{*}xvec\ N|\!).P$› **have** $x \sharp M$ **by** *simp*
from ‹$y \sharp M(\!|\lambda{*}xvec\ N|\!).P$› **have** $y \sharp M$ **by** *simp*
from ‹$\Psi \vdash M \leftrightarrow K$› **have** $([(x, y)] \cdot \Psi) \vdash ([(x, y)] \cdot M) \leftrightarrow ([(x, y)] \cdot K)$
by(*rule chanEqClosed*)
with ‹$x \sharp M$› ‹$y \sharp M$› **have** $([(x, y)] \cdot \Psi) \vdash M \leftrightarrow ([(x, y)] \cdot K)$
by(*simp*)
thus *?case* **using** ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*›
by(*rule Input*)
next
case(*cCase* Ψ *P M N* P' φ *Cs x y*)
from ‹$x \sharp$ *Cases Cs*› ‹$y \sharp$ *Cases Cs*› ‹(φ, P) *mem Cs*› **have** $x \sharp \varphi$ **and** $x \sharp P$ **and** $y \sharp \varphi$ **and** $y \sharp P$
by(*auto dest*: *memFresh*)
from ‹$x \sharp P$› ‹$y \sharp P$› **have** $([(x ,y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$ **by**(*rule cCase*)
moreover note ‹(φ, P) *mem Cs*›
moreover from ‹$\Psi \vdash \varphi$› **have** $([(x, y)] \cdot \Psi) \vdash ([(x, y)] \cdot \varphi)$ **by**(*rule statClosed*)
with ‹$x \sharp \varphi$› ‹$y \sharp \varphi$› **have** $([(x, y)] \cdot \Psi) \vdash \varphi$ **by** *simp*
ultimately show *?case* **using** ‹*guarded P*› **by**(*rule Case*)
next
case(*cPar1* Ψ Ψ_Q *P M N* P' A_Q *Q x y*)
from ‹$x \sharp P \parallel Q$› **have** $x \sharp P$ **and** $x \sharp Q$ **by** *simp*+
from ‹$y \sharp P \parallel Q$› **have** $y \sharp P$ **and** $y \sharp Q$ **by** *simp*+
from ‹$x \sharp P$› ‹$y \sharp P$› ‹$\bigwedge x\ y.\ [\![x \sharp P;\ y \sharp P]\!] \Longrightarrow ([(x, y)] \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$›
have $([(x, y)] \cdot \Psi) \otimes ([(x, y)] \cdot \Psi_Q) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$
by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› **have** $([(x, y)] \cdot (\textit{extractFrame}\ Q)) = ([(x, y)] \cdot \langle A_Q, \Psi_Q\rangle)$
by *simp*
with ‹$A_Q \sharp* x$› ‹$x \sharp Q$› ‹$A_Q \sharp* y$› ‹$y \sharp Q$› **have** $\langle A_Q, ([(x, y)] \cdot \Psi_Q)\rangle = \textit{extractFrame}\ Q$
by(*simp add*: *eqvts*)
moreover from ‹$A_Q \sharp* \Psi$› **have** $([(x, y)] \cdot A_Q) \sharp* ([(x, y)] \cdot \Psi)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_Q \sharp* x$› ‹$A_Q \sharp* y$› **have** $A_Q \sharp* ([(x, y)] \cdot \Psi)$ **by** *simp*
moreover from ‹$A_Q \sharp* M$› **have** $([(x, y)] \cdot A_Q) \sharp* ([(x, y)] \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_Q \sharp* x$› ‹$A_Q \sharp* y$› **have** $A_Q \sharp* ([(x, y)] \cdot M)$ **by** *simp*
ultimately show *?case* **using** ‹$A_Q \sharp* P$› ‹$A_Q \sharp* N$›
by(*rule-tac Par1*) *auto*
next
case(*cPar2* $\Psi\ \Psi_P\ Q\ M\ N\ Q'\ A_P\ P\ x\ y$)
from ‹$x \sharp P \parallel Q$› **have** $x \sharp P$ **and** $x \sharp Q$ **by** *simp*+
from ‹$y \sharp P \parallel Q$› **have** $y \sharp P$ **and** $y \sharp Q$ **by** *simp*+
from ‹$x \sharp Q$› ‹$y \sharp Q$› ‹$\bigwedge x\ y.\ [\![x \sharp Q;\ y \sharp Q]\!] \Longrightarrow ([(x, y)] \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec Q'$›
have $([(x, y)] \cdot \Psi) \otimes ([(x, y)] \cdot \Psi_P) \rhd Q \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec Q'$
by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› **have** $([(x, y)] \cdot (\textit{extractFrame}\ P)) = ([(x, y)] \cdot \langle A_P, \Psi_P\rangle)$
by *simp*
with ‹$A_P \sharp* x$› ‹$x \sharp P$› ‹$A_P \sharp* y$› ‹$y \sharp P$› **have** $\langle A_P, ([(x, y)] \cdot \Psi_P)\rangle = \textit{extractFrame}\ P$
by(*simp add*: *eqvts*)
moreover from ‹$A_P \sharp* \Psi$› **have** $([(x, y)] \cdot A_P) \sharp* ([(x, y)] \cdot \Psi)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* x$› ‹$A_P \sharp* y$› **have** $A_P \sharp* ([(x, y)] \cdot \Psi)$ **by** *simp*
moreover from ‹$A_P \sharp* M$› **have** $([(x, y)] \cdot A_P) \sharp* ([(x, y)] \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* x$› ‹$A_P \sharp* y$› **have** $A_P \sharp* ([(x, y)] \cdot M)$ **by** *simp*
ultimately show *?case* **using** ‹$A_P \sharp* Q$› ‹$A_P \sharp* N$›
by(*rule-tac Par2*) *auto*
next
case(*cScope* $\Psi\ P\ M\ N\ P'\ z\ x\ y$)
from ‹$x \sharp (\!|\nu z|\!)P$› ‹$z \sharp x$› **have** $x \sharp P$ **by**(*simp add*: *abs-fresh*)
from ‹$y \sharp (\!|\nu z|\!)P$› ‹$z \sharp y$› **have** $y \sharp P$ **by**(*simp add*: *abs-fresh*)
from ‹$x \sharp P$› ‹$y \sharp P$› ‹$\bigwedge x\ y.\ [\![x \sharp P;\ y \sharp P]\!] \Longrightarrow ([(x, y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$›
have $([(x, y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$ **by** *simp*
moreover with ‹$z \sharp \Psi$› **have** $([(x, y)] \cdot z) \sharp [(x, y)] \cdot \Psi$
by(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$z \sharp x$› ‹$z \sharp y$› **have** $z \sharp [(x, y)] \cdot \Psi$ **by** *simp*
moreover with ‹$z \sharp M$› **have** $([(x, y)] \cdot z) \sharp [(x, y)] \cdot M$
by(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])

```
  with ‹z ♯ x› ‹z ♯ y› have z ♯ [(x, y)] • M by simp
  ultimately show ?case using ‹z ♯ N›
    by(rule-tac Scope) (assumption | simp)+
next
  case(cBang Ψ P M N P′ x y)
  thus ?case by(force intro: Bang)
qed

lemma inputPermFrameSubject:
  fixes Ψ  :: ′b
  and   P  :: (′a, ′b, ′c) psi
  and   M  :: ′a
  and   N  :: ′a
  and   P′ :: (′a, ′b, ′c) psi
  and   p  :: name prm
  and   Xs :: name set
  and   Ys :: name set

  assumes Ψ ▷ P ⟼M⦇N⦈ ≺ P′
  and     S: set p ⊆ Xs × Ys
  and     Xs ♯* P
  and     Ys ♯* P

  shows (p • Ψ) ▷ P ⟼ (p • M)⦇N⦈ ≺ P′
using S
proof(induct p)
  case Nil
  from ‹Ψ ▷ P ⟼M⦇N⦈ ≺ P′›
  show ?case by simp
next
  case(Cons a p)
  from ‹set(a#p) ⊆ Xs × Ys› have set p ⊆ Xs × Ys by auto
  with ‹set p ⊆ Xs × Ys ⟹ (p • Ψ) ▷ P ⟼ (p • M)⦇N⦈ ≺ P′›
  have Trans: (p • Ψ) ▷ P ⟼ (p • M)⦇N⦈ ≺ P′ by simp
  from ‹set(a#p) ⊆ Xs × Ys› show ?case
  proof(cases a, clarsimp)
    fix a b
    assume a ∈ Xs and b ∈ Ys
    with ‹Xs ♯* P› ‹Ys ♯* P›
    have a ♯ P and b ♯ P
      by(auto simp add: fresh-star-def)
    with Trans show ([(a, b)] • p • Ψ) ▷ P ⟼ ([(a, b)] • p • M)⦇N⦈ ≺ P′
      by(rule inputSwapFrameSubject)
  qed
qed

lemma inputSwapSubject:
  fixes Ψ  :: ′b
  and   P  :: (′a, ′b, ′c) psi
```

and M :: $'a$
and N :: $'a$
and P' :: $('a,\ 'b,\ 'c)$ *psi*
and x :: *name*
and y :: *name*

assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and $x \sharp P$
and $y \sharp P$
and $x \sharp \Psi$
and $y \sharp \Psi$

shows $\Psi \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|N|\!) \prec P'$
proof −
from ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› ‹$x \sharp P$› ‹$y \sharp P$›
have $([(x,\ y)] \cdot \Psi) \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|N|\!) \prec P'$
by(*rule inputSwapFrameSubject*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **show** *?thesis*
by *simp*
qed

lemma *inputPermSubject*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a,\ 'b,\ 'c)$ *psi*
and p :: *name prm*
and Xs :: *name set*
and Ys :: *name set*

assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and S: *set* $p \subseteq Xs \times Ys$
and $Xs \sharp* P$
and $Ys \sharp* P$
and $Xs \sharp* \Psi$
and $Ys \sharp* \Psi$

shows $\Psi \rhd P \longmapsto (p \cdot M)(\!|N|\!) \prec P'$
proof −
from ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› S ‹$Xs \sharp* P$› ‹$Ys \sharp* P$›
have $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|N|\!) \prec P'$
by(*rule inputPermFrameSubject*)
with ‹$Xs \sharp* \Psi$› ‹$Ys \sharp* \Psi$› S **show** *?thesis*
by *simp*
qed

lemma *inputSwapFrame*:
fixes Ψ :: $'b$

and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*
and y :: *name*

assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and $x \sharp P$
and $y \sharp P$
and $x \sharp M$
and $y \sharp M$

shows $([(x, y)] \cdot \Psi) \rhd P \longmapsto M(\!|N|\!) \prec P'$
proof –
from ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› ‹$x \sharp P$› ‹$y \sharp P$›
have $([(x, y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|N|\!) \prec P'$
by(*rule inputSwapFrameSubject*)
with ‹$x \sharp M$› ‹$y \sharp M$› **show** *?thesis*
by *simp*
qed

lemma *inputPermFrame*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*
and Xs :: *name set*
and Ys :: *name set*

assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and S: *set* $p \subseteq Xs \times Ys$
and $Xs \sharp* P$
and $Ys \sharp* P$
and $Xs \sharp* M$
and $Ys \sharp* M$

shows $(p \cdot \Psi) \rhd P \longmapsto M(\!|N|\!) \prec P'$
proof –
from ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› S ‹$Xs \sharp* P$› ‹$Ys \sharp* P$›
have $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|N|\!) \prec P'$
by(*rule inputPermFrameSubject*)
with ‹$Xs \sharp* M$› ‹$Ys \sharp* M$› S **show** *?thesis*
by *simp*
qed

lemma *inputAlpha*:

fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*
and *xvec* :: *name list*

assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
and $set\ p \subseteq (set\ xvec) \times (set\ (p \cdot xvec))$
and *distinctPerm p*
and $xvec \sharp* P$
and $(p \cdot xvec) \sharp* P$

shows $\Psi \rhd P \longmapsto M(\!|(p \cdot N)|\!) \prec (p \cdot P')$
proof −
from ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› ‹$set\ p \subseteq (set\ xvec) \times (set\ (p \cdot xvec))$› ‹$xvec \sharp* P$› ‹$(p \cdot xvec) \sharp* P$›
have $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|N|\!) \prec P'$ **by**(*rule-tac inputPermFrameSubject*) *auto*
hence $(p \cdot p \cdot \Psi) \rhd (p \cdot P) \longmapsto (p \cdot ((p \cdot M)(\!|N|\!) \prec P'))$ **by**(*rule eqvts*)
with ‹*distinctPerm p*› ‹$xvec \sharp* P$› ‹$(p \cdot xvec) \sharp* P$› ‹$set\ p \subseteq (set\ xvec) \times (set\ (p \cdot xvec))$›
show *?thesis* **by**(*simp add*: *eqvts*)
qed

lemma *frameFresh*[*dest*]:
fixes x :: *name*
and A_F :: *name list*
and Ψ_F :: $'b$

assumes $x \sharp A_F$
and $x \sharp \langle A_F, \Psi_F \rangle$

shows $x \sharp \Psi_F$
using *assms*
by(*simp add*: *frameResChainFresh*) (*simp add*: *fresh-def name-list-supp*)

lemma *outputSwapFrameSubject*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and x :: *name*
and y :: *name*

assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N \rangle \prec P'$
and $xvec \sharp* M$

and $x \sharp P$
and $y \sharp P$

shows $([(x, y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
using *assms*
proof(*nominal-induct avoiding*: *x y rule*: *outputInduct'*)
case *cAlpha*
thus *?case* **by**(*simp add*: *create-residual.simps boundOutputChainAlpha''*)
next
case(*cOutput* Ψ M K N P x y)
from ‹$x \sharp M\langle N\rangle.P$› **have** $x \sharp M$ **by** *simp*
from ‹$y \sharp M\langle N\rangle.P$› **have** $y \sharp M$ **by** *simp*
from ‹$\Psi \vdash M \leftrightarrow K$› **have** $([(x, y)] \cdot \Psi) \vdash ([(x, y)] \cdot M) \leftrightarrow ([(x, y)] \cdot K)$
by(*rule chanEqClosed*)
with ‹$x \sharp M$› ‹$y \sharp M$› **have** $([(x, y)] \cdot \Psi) \vdash M \leftrightarrow ([(x, y)] \cdot K)$
by(*simp*)
thus *?case* **by**(*rule Output*)
next
case(*cCase* Ψ P M *xvec* N P' φ *Cs* x y)
from ‹$x \sharp$ *Cases Cs*› ‹$y \sharp$ *Cases Cs*› ‹(φ, P) *mem Cs*› **have** $x \sharp \varphi$ **and** $x \sharp P$ **and** $y \sharp \varphi$ **and** $y \sharp P$
by(*auto dest*: *memFresh*)
from ‹$x \sharp P$› ‹$y \sharp P$› **have** $([(x ,y)] \cdot \Psi) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by**(*rule cCase*)
moreover note ‹(φ, P) *mem Cs*›
moreover from ‹$\Psi \vdash \varphi$› **have** $([(x, y)] \cdot \Psi) \vdash ([(x, y)] \cdot \varphi)$ **by**(*rule statClosed*)
with ‹$x \sharp \varphi$› ‹$y \sharp \varphi$› **have** $([(x, y)] \cdot \Psi) \vdash \varphi$ **by** *simp*
ultimately show *?case* **using** ‹*guarded P*› **by**(*rule Case*)
next
case(*cPar1* Ψ Ψ_Q P M *xvec* N P' A_Q Q x y)
from ‹$x \sharp P \parallel Q$› **have** $x \sharp P$ **and** $x \sharp Q$ **by** *simp+*
from ‹$y \sharp P \parallel Q$› **have** $y \sharp P$ **and** $y \sharp Q$ **by** *simp+*
from ‹$x \sharp P$› ‹$y \sharp P$› ‹$\bigwedge x\ y.\ [\![x \sharp P;\ y \sharp P]\!] \Longrightarrow ([(x, y)] \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$›
have $([(x, y)] \cdot \Psi) \otimes ([(x, y)] \cdot \Psi_Q) \rhd P \longmapsto ([(x, y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› **have** $([(x, y)] \cdot \langle A_Q, \Psi_Q\rangle) = ([(x, y)] \cdot (\mathit{extractFrame}\ Q))$
by *simp*
with ‹$A_Q \sharp{*}\ x$› ‹$x \sharp Q$› ‹$A_Q \sharp{*}\ y$› ‹$y \sharp Q$› **have** $\langle A_Q, ([(x, y)] \cdot \Psi_Q)\rangle = \mathit{extractFrame}\ Q$
by(*simp add*: *eqvts*)
moreover from ‹$A_Q \sharp{*}\ \Psi$› **have** $([(x, y)] \cdot A_Q) \sharp{*} ([(x, y)] \cdot \Psi)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_Q \sharp{*}\ x$› ‹$A_Q \sharp{*}\ y$› **have** $A_Q \sharp{*} ([(x, y)] \cdot \Psi)$ **by** *simp*
moreover from ‹$A_Q \sharp{*}\ M$› **have** $([(x, y)] \cdot A_Q) \sharp{*} ([(x, y)] \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_Q \sharp{*}\ x$› ‹$A_Q \sharp{*}\ y$› **have** $A_Q \sharp{*} ([(x, y)] \cdot M)$ **by** *simp*

ultimately show *?case* **using** ‹A_Q ♯* P› ‹A_Q ♯* N› ‹$xvec$ ♯* Q› ‹A_Q ♯* $xvec$›
by(*rule-tac Par1*) *auto*
next
case(*cPar2* Ψ Ψ_P Q M $xvec$ N Q' A_P P x y)
from ‹x ♯ $P \parallel Q$› **have** x ♯ P **and** x ♯ Q **by** *simp*+
from ‹y ♯ $P \parallel Q$› **have** y ♯ P **and** y ♯ Q **by** *simp*+
from ‹x ♯ Q› ‹y ♯ Q› ‹$\bigwedge x\ y.\ [\![x \sharp Q;\ y \sharp Q]\!] \Longrightarrow ([(x,\ y)] \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$›
have $([(x,\ y)] \cdot \Psi) \otimes ([(x,\ y)] \cdot \Psi_P) \rhd Q \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
by(*simp add*: *eqvts*)

moreover from ‹$extractFrame\ P = \langle A_P,\ \Psi_P\rangle$› **have** $([(x,\ y)] \cdot \langle A_P,\ \Psi_P\rangle) = ([(x,\ y)] \cdot (extractFrame\ P))$
by *simp*
with ‹A_P ♯* x› ‹x ♯ P› ‹A_P ♯* y› ‹y ♯ P› **have** $\langle A_P,\ ([(x,\ y)] \cdot \Psi_P)\rangle = extractFrame\ P$
by(*simp add*: *eqvts*)
moreover from ‹A_P ♯* Ψ› **have** $([(x,\ y)] \cdot A_P)$ ♯* $([(x,\ y)] \cdot \Psi)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* x› ‹A_P ♯* y› **have** A_P ♯* $([(x,\ y)] \cdot \Psi)$ **by** *simp*
moreover from ‹A_P ♯* M› **have** $([(x,\ y)] \cdot A_P)$ ♯* $([(x,\ y)] \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* x› ‹A_P ♯* y› **have** A_P ♯* $([(x,\ y)] \cdot M)$ **by** *simp*
ultimately show *?case* **using** ‹A_P ♯* Q› ‹A_P ♯* N› ‹$xvec$ ♯* P› ‹A_P ♯* $xvec$›
by(*rule-tac Par2*) *auto*
next
case(*cOpen* Ψ P M $xvec$ $yvec$ N P' z x y)
from ‹x ♯ $(\!|\nu z|\!)P$› ‹z ♯ x› **have** x ♯ P **by**(*simp add*: *abs-fresh*)
from ‹y ♯ $(\!|\nu z|\!)P$› ‹z ♯ y› **have** y ♯ P **by**(*simp add*: *abs-fresh*)
from ‹x ♯ P› ‹y ♯ P› ‹$\bigwedge x\ y.\ [\![x \sharp P;\ y \sharp P]\!] \Longrightarrow ([(x,\ y)] \cdot \Psi) \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P'$›
have $([(x,\ y)] \cdot \Psi) \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}(xvec@yvec)|\!)\langle N\rangle \prec P'$ **by** *simp*
moreover with ‹z ♯ Ψ› **have** $([(x,\ y)] \cdot z)$ ♯ $[(x,\ y)] \cdot \Psi$
by(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹z ♯ x› ‹z ♯ y› **have** z ♯ $[(x,\ y)] \cdot \Psi$ **by** *simp*
moreover with ‹z ♯ M› **have** $([(x,\ y)] \cdot z)$ ♯ $[(x,\ y)] \cdot M$
by(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹z ♯ x› ‹z ♯ y› **have** z ♯ $[(x,\ y)] \cdot M$ **by** *simp*
ultimately show *?case* **using** ‹$z \in supp\ N$› ‹z ♯ $xvec$› ‹z ♯ $yvec$›
by(*rule-tac Open*) (*assumption* | *simp*)+
next
case(*cScope* Ψ P M $xvec$ N P' z x y)
from ‹x ♯ $(\!|\nu z|\!)P$› ‹z ♯ x› **have** x ♯ P **by**(*simp add*: *abs-fresh*)
from ‹y ♯ $(\!|\nu z|\!)P$› ‹z ♯ y› **have** y ♯ P **by**(*simp add*: *abs-fresh*)
from ‹x ♯ P› ‹y ♯ P› ‹$\bigwedge x\ y.\ [\![x \sharp P;\ y \sharp P]\!] \Longrightarrow ([(x,\ y)] \cdot \Psi) \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$›
have $([(x,\ y)] \cdot \Psi) \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by** *simp*
moreover with ‹z ♯ Ψ› **have** $([(x,\ y)] \cdot z)$ ♯ $[(x,\ y)] \cdot \Psi$
by(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])

with ‹$z \sharp x$› ‹$z \sharp y$› **have** $z \sharp [(x, y)] \cdot \Psi$ **by** *simp*
moreover with ‹$z \sharp M$› **have** $([(x, y)] \cdot z) \sharp [(x, y)] \cdot M$
by(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$z \sharp x$› ‹$z \sharp y$› **have** $z \sharp [(x, y)] \cdot M$ **by** *simp*
ultimately show *?case* **using** ‹$z \sharp N$› ‹$z \sharp xvec$›
by(*rule-tac Scope*) (*assumption* | *simp*)+
next
case(*cBang* Ψ P M B x y)
thus *?case* **by**(*force intro*: *Bang*)
qed

lemma *outputPermFrameSubject*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*
and *yvec* :: *name list*
and *zvec* :: *name list*

assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and S: *set* $p \subseteq$ *set yvec* $\times$ *set zvec*
and *yvec* $\sharp{*}$ P
and *zvec* $\sharp{*}$ P

shows $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
proof −
{
fix *xvec* N P' *Xs* *YS*
assume $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **and** *xvec* $\sharp{*}$ M **and** *xvec* $\sharp{*}$ *yvec* **and** *xvec* $\sharp{*}$ *zvec*
have $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **using** S
proof(*induct p*)
case *Nil*
from ‹$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$›
show *?case* **by** *simp*
next
case(*Cons a p*)
from ‹*set*($a\#p$) $\subseteq$ *set yvec* $\times$ *set zvec*› **have** *set* $p \subseteq$ *set yvec* $\times$ *set zvec* **by** *auto*
then have *Trans*: $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by**(*rule Cons*)
from ‹*set*($a\#p$) $\subseteq$ *set yvec* $\times$ *set zvec*› **show** *?case*
proof(*cases a*, *clarsimp*)
fix x y
note *Trans*
moreover from ‹*xvec* $\sharp{*}$ *yvec*› ‹*xvec* $\sharp{*}$ *zvec*› ‹*set* $p \subseteq$ *set yvec* $\times$ *set zvec*› ‹*xvec* $\sharp{*}$ M› **have** *xvec* $\sharp{*}$ $(p \cdot M)$

by(*simp add*: *freshChainSimps*)
moreover assume $x \in$ *set yvec* **and** $y \in$ *set zvec*
with ‹*yvec* $\sharp*$ P› ‹*zvec* $\sharp*$ P› **have** $x \sharp P$ **and** $y \sharp P$
by(*auto simp add*: *fresh-star-def*)
ultimately show $([(x, y)] \cdot p \cdot \Psi) \rhd P \longmapsto([(x, y)] \cdot p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*rule outputSwapFrameSubject*)
qed
qed
}
note *Goal* = *this*
obtain *q*::*name prm* **where** $(q \cdot xvec) \sharp* yvec$ **and** $(q \cdot xvec) \sharp* zvec$ **and** $(q \cdot xvec) \sharp* xvec$
and $(q \cdot xvec) \sharp* N$ **and** $(q \cdot xvec) \sharp* P'$ **and** $(q \cdot xvec) \sharp* M$
and *Sq*: $(set\ q) \subseteq (set\ xvec) \times (set(q \cdot xvec))$
by(*rule-tac xvec*=*xvec* **and** *c*=(P, *xvec*, *yvec*, *zvec*, N, M, P') **in** *name-list-avoiding*) *auto*
with ‹$\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› **have** $\Psi \rhd P \longmapsto M(\!|\nu*(q \cdot xvec)|\!)\langle(q \cdot N)\rangle \prec (q \cdot P')$
by(*simp add*: *boundOutputChainAlpha''* *residualInject*)
hence $(p \cdot \Psi) \rhd P \longmapsto(p \cdot M)(\!|\nu*(q \cdot xvec)|\!)\langle(q \cdot N)\rangle \prec (q \cdot P')$
using ‹$(q \cdot xvec) \sharp* M$› ‹$(q \cdot xvec) \sharp* yvec$› ‹$(q \cdot xvec) \sharp* zvec$›
by(*rule Goal*)
with ‹$(q \cdot xvec) \sharp* N$› ‹$(q \cdot xvec) \sharp* P'$› *Sq* **show** *?thesis*
by(*simp add*: *boundOutputChainAlpha''* *residualInject*)
qed

lemma *outputSwapSubject*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and M :: *'a*
and B :: (*'a*, *'b*, *'c*) *boundOutput*
and x :: *name*
and y :: *name*

assumes $\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
and $xvec \sharp* M$
and $x \sharp P$
and $y \sharp P$
and $x \sharp \Psi$
and $y \sharp \Psi$

shows $\Psi \rhd P \longmapsto([(x, y)] \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
proof −
from ‹$\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› ‹$xvec \sharp* M$› ‹$x \sharp P$› ‹$y \sharp P$›
have $([(x, y)] \cdot \Psi) \rhd P \longmapsto([(x, y)] \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*rule outputSwapFrameSubject*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **show** *?thesis*
by *simp*

qed

lemma *outputPermSubject*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and M :: $'a$
  and B :: $('a,\ 'b,\ 'c)$ *boundOutput*
  and p :: *name prm*
  and *yvec* :: *name list*
  and *zvec* :: *name list*

  assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
  and S: *set* $p \subseteq$ *set yvec* $\times$ *set zvec*
  and *yvec* $\sharp*$ P
  and *zvec* $\sharp*$ P
  and *yvec* $\sharp*$ Ψ
  and *zvec* $\sharp*$ Ψ

  shows $\Psi \rhd P \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
proof −
  from *assms* **have** $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
   by(*rule-tac outputPermFrameSubject*)
  with S ‹*yvec* $\sharp*$ Ψ› ‹*zvec* $\sharp*$ Ψ› **show** *?thesis*
    by *simp*
qed

lemma *outputSwapFrame*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and M :: $'a$
  and B :: $('a,\ 'b,\ 'c)$ *boundOutput*
  and x :: *name*
  and y :: *name*

  assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
  and *xvec* $\sharp*$ M
  and $x \sharp P$
  and $y \sharp P$
  and $x \sharp M$
  and $y \sharp M$

  shows $([(x,\ y)] \cdot \Psi) \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
proof −
  from ‹$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹*xvec* $\sharp*$ M› ‹$x \sharp P$› ‹$y \sharp P$›
  have $([(x,\ y)] \cdot \Psi) \rhd P \longmapsto ([(x,\ y)] \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
    by(*rule outputSwapFrameSubject*)
  with ‹$x \sharp M$› ‹$y \sharp M$› **show** *?thesis*
    by *simp*
qed

lemma *outputPermFrame*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and B :: $('a, 'b, 'c)$ *boundOutput*
  and p :: *name prm*
  and *yvec* :: *name list*
  and *zvec* :: *name list*

  assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
  and S: *set* $p \subseteq$ *set yvec* $\times$ *set zvec*
  and *yvec* $\sharp*$ P
  and *zvec* $\sharp*$ P
  and *yvec* $\sharp*$ M
  and *zvec* $\sharp*$ M

  shows $(p \cdot \Psi) \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
proof −
  from *assms* **have** $(p \cdot \Psi) \rhd P \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
    by(*rule-tac outputPermFrameSubject*)
  with S ‹*yvec* $\sharp*$ M› ‹*zvec* $\sharp*$ M› **show** *?thesis*
    by *simp*
qed

lemma *Comm1*:
  fixes Ψ :: $'b$
  and Ψ_Q :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$
  and P' :: $('a, 'b, 'c)$ *psi*
  and A_P :: *name list*
  and Ψ_P :: $'b$
  and Q :: $('a, 'b, 'c)$ *psi*
  and K :: $'a$
  and *xvec* :: *name list*
  and Q' :: $('a, 'b, 'c)$ *psi*
  and A_Q :: *name list*

  assumes $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$
  and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
  and $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
  and *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
  and $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$
  and $A_P \sharp* \Psi$
  and $A_P \sharp* P$
  and $A_P \sharp* Q$
  and $A_P \sharp* M$

and $A_P \sharp* A_Q$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* Q$
and $A_Q \sharp* K$
and $xvec \sharp* P$

shows $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$
proof −
{
fix Ψ :: $'b$
and Ψ_Q :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and K :: $'a$
and *xvec* :: *name list*
and Q' :: $('a, 'b, 'c)$ *psi*
and A_Q :: *name list*

assume $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$
and *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
and *distinct* A_Q
and $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$
and $A_P \sharp* \Psi$
and $A_P \sharp* P$
and $A_P \sharp* Q$
and $A_P \sharp* M$
and $A_P \sharp* A_Q$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* Q$
and $A_Q \sharp* K$
and $xvec \sharp* P$

have $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$
proof −

obtain r::*name prm* **where** $(r \cdot xvec) \sharp* \Psi$ **and** $(r \cdot xvec) \sharp* P$ **and** $(r \cdot xvec) \sharp* Q$ **and** $(r \cdot xvec) \sharp* M$
and $(r \cdot xvec) \sharp* K$ **and** $(r \cdot xvec) \sharp* N$ **and** $(r \cdot xvec) \sharp* A_P$ **and** $(r \cdot xvec) \sharp* A_Q$

and $(r \cdot xvec) \sharp* P'$ **and** $(r \cdot xvec) \sharp* Q'$ **and** $(r \cdot xvec) \sharp* \Psi_P$ **and** $(r \cdot xvec) \sharp* \Psi_Q$
and *Sr*: $(set\ r) \subseteq (set\ xvec) \times (set(r \cdot xvec))$ **and** *distinctPerm r*
by(*rule-tac xvec=xvec* **and** $c=(\Psi, P, Q, M, K, N, A_P, A_Q, \Psi_P, \Psi_Q, P', Q')$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain *q*::*name prm* **where** $(q \cdot A_Q) \sharp* \Psi$ **and** $(q \cdot A_Q) \sharp* P$ **and** $(q \cdot A_Q) \sharp* Q$ **and** $(q \cdot A_Q) \sharp* K$
and $(q \cdot A_Q) \sharp* (r \cdot N)$ **and** $(q \cdot A_Q) \sharp* (r \cdot xvec)$ **and** $(q \cdot A_Q) \sharp* (r \cdot Q')$
and $(q \cdot A_Q) \sharp* (r \cdot P')$ **and** $(q \cdot A_Q) \sharp* \Psi_P$ **and** $(q \cdot A_Q) \sharp* A_P$ **and** $(q \cdot A_Q) \sharp* \Psi_Q$
and *Sq*: $set\ q \subseteq set\ A_Q \times set(q \cdot A_Q)$
by(*rule-tac xvec=*A_Q **and** $c=(\Psi, P, Q, K, r \cdot N, r \cdot xvec, \Psi_Q, A_P, \Psi_P, r \cdot Q', r \cdot P')$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain *p*::*name prm* **where** $(p \cdot A_P) \sharp* \Psi$ **and** $(p \cdot A_P) \sharp* P$ **and** $(p \cdot A_P) \sharp* Q$ **and** $(p \cdot A_P) \sharp* M$
and $(p \cdot A_P) \sharp* (r \cdot N)$ **and** $(p \cdot A_P) \sharp* (r \cdot xvec)$ **and** $(p \cdot A_P) \sharp* (r \cdot Q')$
and $(p \cdot A_P) \sharp* (r \cdot P')$ **and** $(p \cdot A_P) \sharp* \Psi_P$ **and** $(p \cdot A_P) \sharp* \Psi_Q$ **and** $(p \cdot A_P) \sharp* A_Q$
and $(p \cdot A_P) \sharp* (q \cdot A_Q)$ **and** *Sp*: $(set\ p) \subseteq (set\ A_P) \times (set(p \cdot A_P))$
by(*rule-tac xvec=*A_P **and** $c=(\Psi, P, Q, M, r \cdot N, r \cdot xvec, A_Q, q \cdot A_Q, \Psi_Q, \Psi_P, r \cdot Q', r \cdot P')$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)

have *FrP*: $extractFrame\ P = \langle A_P, \Psi_P \rangle$ **by** *fact*
have *FrQ*: $extractFrame\ Q = \langle A_Q, \Psi_Q \rangle$ **by** *fact*

from ‹$A_P \sharp* Q$› *FrQ* ‹$A_P \sharp* A_Q$› **have** $A_P \sharp* \Psi_Q$
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹$A_Q \sharp* P$› *FrP* ‹$A_P \sharp* A_Q$› **have** $A_Q \sharp* \Psi_P$
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹$(r \cdot xvec) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* A_P$› *Sp* **have** $(r \cdot xvec) \sharp* (p \cdot A_P)$
by(*simp add*: *freshChainSimps*)

from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› *Sr* ‹*distinctPerm r*› ‹$xvec \sharp* P$› ‹$(r \cdot xvec) \sharp* P$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|(r \cdot N)|\!) \prec (r \cdot P')$
by(*rule inputAlpha*)
hence $(q \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto (q \cdot M)(\!|(r \cdot N)|\!) \prec (r \cdot P')$ **using** *Sq* ‹$A_Q \sharp* P$› ‹$(q \cdot A_Q) \sharp* P$›
by(*rule-tac inputPermFrameSubject*) (*assumption* | *simp*)+
hence *PTrans*: $\Psi \otimes (q \cdot \Psi_Q) \rhd P \longmapsto (q \cdot M)(\!|(r \cdot N)|\!) \prec (r \cdot P')$ **using** *Sq* ‹$A_Q \sharp* \Psi$› ‹$(q \cdot A_Q) \sharp* \Psi$›

by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› *Sp* ‹$(p \cdot A_P) \sharp* \Psi_P$›
have *FrP*: *extractFrame* $P = \langle (p \cdot A_P), (p \cdot \Psi_P)\rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹*distinct* A_P› **have** *distinct*$(p \cdot A_P)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› *Sr* ‹$(r \cdot xvec) \sharp* N$› ‹$(r \cdot xvec) \sharp* Q'$›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*(r \cdot xvec)|\!)\langle (r \cdot N)\rangle \prec (r \cdot Q')$
by(*simp add*: *boundOutputChainAlpha''* *create-residual.simps*)
hence $(p \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto (p \cdot K)(\!|\nu*(r \cdot xvec)|\!)\langle (r \cdot N)\rangle \prec (r \cdot Q')$ **using** *Sp* ‹$A_P \sharp* Q$› ‹$(p \cdot A_P) \sharp* Q$› ‹$(r \cdot xvec) \sharp* K$› ‹$(r \cdot xvec) \sharp* A_P$› ‹$(r \cdot xvec) \sharp* (p \cdot A_P)$›
by(*rule-tac outputPermFrameSubject*) (*assumption* | *auto*)
hence *QTrans*: $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto (p \cdot K)(\!|\nu*(r \cdot xvec)|\!)\langle (r \cdot N)\rangle \prec (r \cdot Q')$ **using** *Sp* ‹$A_P \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* \Psi$›
by(*simp add*: *eqvts*)
moreover hence *distinct*$(r \cdot xvec)$ **by**(*force dest*: *boundOutputDistinct*)
moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› *Sq* ‹$(q \cdot A_Q) \sharp* \Psi_Q$›
have *FrQ*: *extractFrame* $Q = \langle (q \cdot A_Q), (q \cdot \Psi_Q)\rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹*distinct* A_Q› **have** *distinct*$(q \cdot A_Q)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(p \cdot q \cdot (\Psi \otimes \Psi_P \otimes \Psi_Q)) \vdash (p \cdot q \cdot M) \leftrightarrow (p \cdot q \cdot K)$
by(*rule-tac chanEqClosed*)+
with ‹$A_P \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$(q \cdot A_Q) \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$(q \cdot A_Q) \sharp* \Psi_P$›
‹$A_P \sharp* \Psi_Q$› ‹$(p \cdot A_P) \sharp* \Psi_Q$› ‹$A_P \sharp* M$› ‹$(p \cdot A_P) \sharp* M$› ‹$(q \cdot A_Q) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$›
‹$A_Q \sharp* K$› ‹$(q \cdot A_Q) \sharp* K$› ‹$A_P \sharp* A_Q$› ‹$(p \cdot A_P) \sharp* A_Q$› *Sp Sq*
have $\Psi \otimes (p \cdot \Psi_P) \otimes (q \cdot \Psi_Q) \vdash (q \cdot M) \leftrightarrow (p \cdot K)$ **by**(*simp add*: *eqvts freshChainSimps*)
moreover note ‹$(p \cdot A_P) \sharp* \Psi$›
moreover from ‹$(p \cdot A_P) \sharp* A_Q$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$› ‹$(p \cdot A_P) \sharp* \Psi_Q$› *Sq* **have** $(p \cdot A_P) \sharp* (q \cdot \Psi_Q)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P) \sharp* P$›
moreover from ‹$(p \cdot A_P) \sharp* A_Q$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$› ‹$(p \cdot A_P) \sharp* M$› *Sq* **have** $(p \cdot A_P) \sharp* (q \cdot M)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P) \sharp* (r \cdot N)$› ‹$(p \cdot A_P) \sharp* (r \cdot P')$› ‹$(p \cdot A_P) \sharp* Q$› ‹$(p \cdot A_P) \sharp* (r \cdot Q')$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$›
‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(q \cdot A_Q) \sharp* \Psi$›
moreover from ‹$(q \cdot A_Q) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* A_Q$› ‹$(q \cdot A_Q) \sharp* \Psi_P$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$› *Sp Sq* **have** $(q \cdot A_Q) \sharp* (p \cdot \Psi_P)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q) \sharp* P$› ‹$(q \cdot A_Q) \sharp* (r \cdot N)$›‹$(q \cdot A_Q) \sharp* (r \cdot P')$›

‹$(q \cdot A_Q) \sharp* Q$›
moreover from ‹$(q \cdot A_Q) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* A_Q$› ‹$(q \cdot A_Q) \sharp* K$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$› *Sp Sq* **have** $(q \cdot A_Q) \sharp* (p \cdot K)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q) \sharp* (r \cdot Q')$› ‹$(q \cdot A_Q) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* \Psi$›
moreover from ‹$(r \cdot xvec) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* \Psi_P$› *Sp* **have** $(r \cdot xvec) \sharp* (p \cdot \Psi_P)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(r \cdot xvec) \sharp* A_Q$› ‹$(q \cdot A_Q) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* \Psi_Q$› *Sq* **have** $(r \cdot xvec) \sharp* (q \cdot \Psi_Q)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(r \cdot xvec) \sharp* P$›
moreover from ‹$(r \cdot xvec) \sharp* A_Q$› ‹$(q \cdot A_Q) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* M$› *Sq* **have** $(r \cdot xvec) \sharp* (q \cdot M)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(r \cdot xvec) \sharp* Q$›
moreover from ‹$(r \cdot xvec) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* K$› *Sp* **have** $(r \cdot xvec) \sharp* (p \cdot K)$
by(*simp add*: *freshChainSimps*)
ultimately have $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu*(r \cdot xvec)|\!)((r \cdot P') \parallel (r \cdot Q'))$
by(*rule-tac cComm1*)
with ‹$(r \cdot xvec) \sharp* P'$› ‹$(r \cdot xvec) \sharp* Q'$› *Sr*
show *?thesis*
by(*subst resChainAlpha*) *auto*
qed
}
note *Goal* = *this*
note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$›
moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* M$› ‹$A_P \sharp* A_Q$›
obtain A_P' **where** *extractFrame* $P = \langle A_P', \Psi_P\rangle$ **and** *distinct* A_P' **and** $A_P' \sharp* \Psi$ **and** $A_P' \sharp* P$ **and** $A_P' \sharp* Q$ **and** $A_P' \sharp* M$ **and** $A_P' \sharp* A_Q$
by(*rule-tac* C=(Ψ, P, Q, M, A_Q) **in** *distinctFrame*) *auto*
moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_P' \sharp* A_Q$›
obtain A_Q' **where** *extractFrame* $Q = \langle A_Q', \Psi_Q\rangle$ **and** *distinct* A_Q' **and** $A_Q' \sharp* \Psi$ **and** $A_Q' \sharp* P$ **and** $A_Q' \sharp* Q$ **and** $A_Q' \sharp* K$ **and** $A_P' \sharp* A_Q'$
apply(*rule-tac* C=(Ψ, P, Q, K, A_P') **in** *distinctFrame*) **by** *auto*
ultimately show *?thesis* **using** ‹$xvec \sharp* P$›
by(*rule-tac Goal*)
qed

lemma *Comm2*:
fixes Ψ :: $'b$
and Ψ_Q :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$

and $xvec$ :: *name list*
and N :: $'a$
and P' :: $('a,\ 'b,\ 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and K :: $'a$
and Q' :: $('a,\ 'b,\ 'c)$ *psi*
and A_Q :: *name list*

assumes $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $extractFrame\ P = \langle A_P,\ \Psi_P\rangle$
and $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$
and $extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$
and $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$
and $A_P \sharp{*}\ \Psi$
and $A_P \sharp{*}\ P$
and $A_P \sharp{*}\ Q$
and $A_P \sharp{*}\ M$
and $A_P \sharp{*}\ A_Q$
and $A_Q \sharp{*}\ \Psi$
and $A_Q \sharp{*}\ P$
and $A_Q \sharp{*}\ Q$
and $A_Q \sharp{*}\ K$
and $xvec \sharp{*}\ Q$

shows $\Psi \rhd P \parallel Q \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q')$
proof −
{
fix Ψ :: $'b$
and Ψ_Q :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and M :: $'a$
and $xvec$ :: *name list*
and N :: $'a$
and P' :: $('a,\ 'b,\ 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and K :: $'a$
and Q' :: $('a,\ 'b,\ 'c)$ *psi*
and A_Q :: *name list*

assume $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $extractFrame\ P = \langle A_P,\ \Psi_P\rangle$
and $distinct\ A_P$
and $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$
and $extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$
and $distinct\ A_Q$

and $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$
and $A_P \sharp* \Psi$
and $A_P \sharp* P$
and $A_P \sharp* Q$
and $A_P \sharp* M$
and $A_P \sharp* A_Q$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* Q$
and $A_Q \sharp* K$
and $xvec \sharp* Q$

have $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$
proof −

obtain r::*name prm* **where** $(r \cdot xvec) \sharp* \Psi$ **and** $(r \cdot xvec) \sharp* P$ **and** $(r \cdot xvec) \sharp* Q$ **and** $(r \cdot xvec) \sharp* M$
and $(r \cdot xvec) \sharp* K$ **and** $(r \cdot xvec) \sharp* N$ **and** $(r \cdot xvec) \sharp* A_P$ **and** $(r \cdot xvec) \sharp* A_Q$
and $(r \cdot xvec) \sharp* P'$ **and** $(r \cdot xvec) \sharp* Q'$ **and** $(r \cdot xvec) \sharp* \Psi_P$ **and** $(r \cdot xvec) \sharp* \Psi_Q$
and *Sr*: $(set\ r) \subseteq (set\ xvec) \times (set(r \cdot xvec))$ **and** *distinctPerm r*
by(*rule-tac xvec*=*xvec* **and** *c*=(Ψ, P, Q, M, K, N, A_P, A_Q, Ψ_P, Ψ_Q, P', Q') **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain q::*name prm* **where** $(q \cdot A_Q) \sharp* \Psi$ **and** $(q \cdot A_Q) \sharp* P$ **and** $(q \cdot A_Q) \sharp* Q$ **and** $(q \cdot A_Q) \sharp* K$
and $(q \cdot A_Q) \sharp* (r \cdot N)$ **and** $(q \cdot A_Q) \sharp* (r \cdot xvec)$ **and** $(q \cdot A_Q) \sharp* (r \cdot Q')$
and $(q \cdot A_Q) \sharp* (r \cdot P')$ **and** $(q \cdot A_Q) \sharp* \Psi_P$ **and** $(q \cdot A_Q) \sharp* A_P$ **and** $(q \cdot A_Q) \sharp* \Psi_Q$
and *Sq*: $set\ q \subseteq set\ A_Q \times set(q \cdot A_Q)$
by(*rule-tac xvec*=A_Q **and** *c*=(Ψ, P, Q, K, $r \cdot N$, $r \cdot xvec$, Ψ_Q, A_P, Ψ_P, $r \cdot Q'$, $r \cdot P'$) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain p::*name prm* **where** $(p \cdot A_P) \sharp* \Psi$ **and** $(p \cdot A_P) \sharp* P$ **and** $(p \cdot A_P) \sharp* Q$ **and** $(p \cdot A_P) \sharp* M$
and $(p \cdot A_P) \sharp* (r \cdot N)$ **and** $(p \cdot A_P) \sharp* (r \cdot xvec)$ **and** $(p \cdot A_P) \sharp* (r \cdot Q')$
and $(p \cdot A_P) \sharp* (r \cdot P')$ **and** $(p \cdot A_P) \sharp* \Psi_P$ **and** $(p \cdot A_P) \sharp* \Psi_Q$ **and** $(p \cdot A_P) \sharp* A_Q$
and $(p \cdot A_P) \sharp* (q \cdot A_Q)$ **and** *Sp*: $(set\ p) \subseteq (set\ A_P) \times (set(p \cdot A_P))$
by(*rule-tac xvec*=A_P **and** *c*=(Ψ, P, Q, M, $r \cdot N$, $r \cdot xvec$, A_Q, $q \cdot A_Q$, Ψ_Q, Ψ_P, $r \cdot Q'$, $r \cdot P'$) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)

have *FrP*: $extractFrame\ P = \langle A_P, \Psi_P \rangle$ **by** *fact*

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$ **by** *fact*

from ‹$A_P \sharp* Q$› *FrQ* ‹$A_P \sharp* A_Q$› **have** $A_P \sharp* \Psi_Q$
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹$A_Q \sharp* P$› *FrP* ‹$A_P \sharp* A_Q$› **have** $A_Q \sharp* \Psi_P$
by(*drule-tac extractFrameFreshChain*) *auto*

from ‹$(r \cdot xvec) \sharp* A_Q$› ‹$(q \cdot A_Q) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* A_Q$› *Sq* **have** $(r \cdot xvec) \sharp* (q \cdot A_Q)$
by(*simp add*: *freshChainSimps*)

from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N \rangle \prec P'$› *Sr* ‹$(r \cdot xvec) \sharp* N$› ‹$(r \cdot xvec) \sharp* P'$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}(r \cdot xvec)|\!)\langle (r \cdot N) \rangle \prec (r \cdot P')$
by(*simp add*: *boundOutputChainAlpha''* *create-residual.simps*)
hence $(q \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto (q \cdot M)(\!|\nu{*}(r \cdot xvec)|\!)\langle (r \cdot N) \rangle \prec (r \cdot P')$ **using** *Sq* ‹$A_Q \sharp* P$› ‹$(q \cdot A_Q) \sharp* P$› ‹$(r \cdot xvec) \sharp* M$› ‹$(r \cdot xvec) \sharp* A_Q$› ‹$(r \cdot xvec) \sharp* (q \cdot A_Q)$›
by(*rule-tac outputPermFrameSubject*) (*assumption* | *auto*)
hence *PTrans*: $\Psi \otimes (q \cdot \Psi_Q) \rhd P \longmapsto (q \cdot M)(\!|\nu{*}(r \cdot xvec)|\!)\langle (r \cdot N) \rangle \prec (r \cdot P')$ **using** *Sq* ‹$A_Q \sharp* \Psi$› ‹$(q \cdot A_Q) \sharp* \Psi$›
by(*simp add*: *eqvts*)
moreover hence *distinct*$(r \cdot xvec)$ **by**(*force dest*: *boundOutputDistinct*)

moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› *Sp* ‹$(p \cdot A_P) \sharp* \Psi_P$›
have *FrP*: *extractFrame* $P = \langle (p \cdot A_P), (p \cdot \Psi_P) \rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹*distinct* A_P› **have** *distinct*$(p \cdot A_P)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› *Sr* ‹*distinctPerm r*› ‹$xvec \sharp* Q$› ‹$(r \cdot xvec) \sharp* Q$›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|(r \cdot N)|\!) \prec (r \cdot Q')$
by(*rule inputAlpha*)
hence $(p \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto (p \cdot K)(\!|(r \cdot N)|\!) \prec (r \cdot Q')$ **using** *Sp* ‹$A_P \sharp* Q$› ‹$(p \cdot A_P) \sharp* Q$›
by(*rule-tac inputPermFrameSubject*) (*assumption* | *simp*)+
hence *QTrans*: $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto (p \cdot K)(\!|(r \cdot N)|\!) \prec (r \cdot Q')$ **using** *Sp* ‹$A_P \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* \Psi$›
by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$› *Sq* ‹$(q \cdot A_Q) \sharp* \Psi_Q$›
have *FrQ*: *extractFrame* $Q = \langle (q \cdot A_Q), (q \cdot \Psi_Q) \rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹*distinct* A_Q› **have** *distinct*$(q \cdot A_Q)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(p \cdot q \cdot (\Psi \otimes \Psi_P \otimes \Psi_Q)) \vdash (p \cdot q \cdot M) \leftrightarrow (p \cdot q \cdot K)$
by(*rule-tac chanEqClosed*)+
with ‹$A_P \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$(q \cdot A_Q) \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹(q

$\cdot\ A_Q)\ \sharp*\ \Psi_P$›
‹$A_P\ \sharp*\ \Psi_Q$› ‹$(p \cdot A_P)\ \sharp*\ \Psi_Q$› ‹$A_P\ \sharp*\ M$› ‹$(p \cdot A_P)\ \sharp*\ M$› ‹$(q \cdot A_Q)\ \sharp*\ A_P$› ‹$(p \cdot A_P)\ \sharp*\ (q \cdot A_Q)$›
‹$A_Q\ \sharp*\ K$› ‹$(q \cdot A_Q)\ \sharp*\ K$› ‹$A_P\ \sharp*\ A_Q$› ‹$(p \cdot A_P)\ \sharp*\ A_Q$› *Sp Sq*
have $\Psi \otimes (p \cdot \Psi_P) \otimes (q \cdot \Psi_Q) \vdash (q \cdot M) \leftrightarrow (p \cdot K)$
by(*simp add*: *eqvts freshChainSimps*)
moreover note ‹$(p \cdot A_P)\ \sharp*\ \Psi$›
moreover from ‹$(p \cdot A_P)\ \sharp*\ A_Q$› ‹$(p \cdot A_P)\ \sharp*\ (q \cdot A_Q)$› ‹$(p \cdot A_P)\ \sharp*\ \Psi_Q$› *Sq* **have** $(p \cdot A_P)\ \sharp*\ (q \cdot \Psi_Q)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P)\ \sharp*\ P$›
moreover from ‹$(p \cdot A_P)\ \sharp*\ A_Q$› ‹$(p \cdot A_P)\ \sharp*\ (q \cdot A_Q)$› ‹$(p \cdot A_P)\ \sharp*\ M$› *Sq* **have** $(p \cdot A_P)\ \sharp*\ (q \cdot M)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P)\ \sharp*\ (r \cdot N)$› ‹$(p \cdot A_P)\ \sharp*\ (r \cdot P')$› ‹$(p \cdot A_P)\ \sharp*\ Q$› ‹$(p \cdot A_P)\ \sharp*\ (r \cdot Q')$› ‹$(p \cdot A_P)\ \sharp*\ (q \cdot A_Q)$›
‹$(p \cdot A_P)\ \sharp*\ (r \cdot xvec)$› ‹$(q \cdot A_Q)\ \sharp*\ \Psi$›
moreover from ‹$(q \cdot A_Q)\ \sharp*\ A_P$› ‹$(p \cdot A_P)\ \sharp*\ A_Q$› ‹$(q \cdot A_Q)\ \sharp*\ \Psi_P$› ‹$(p \cdot A_P)\ \sharp*\ (q \cdot A_Q)$› *Sp Sq* **have** $(q \cdot A_Q)\ \sharp*\ (p \cdot \Psi_P)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q)\ \sharp*\ P$› ‹$(q \cdot A_Q)\ \sharp*\ (r \cdot N)$›‹$(q \cdot A_Q)\ \sharp*\ (r \cdot P')$› ‹$(q \cdot A_Q)\ \sharp*\ Q$›
moreover from ‹$(q \cdot A_Q)\ \sharp*\ A_P$› ‹$(p \cdot A_P)\ \sharp*\ A_Q$› ‹$(q \cdot A_Q)\ \sharp*\ K$› ‹$(p \cdot A_P)\ \sharp*\ (q \cdot A_Q)$› *Sp Sq* **have** $(q \cdot A_Q)\ \sharp*\ (p \cdot K)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q)\ \sharp*\ (r \cdot Q')$› ‹$(q \cdot A_Q)\ \sharp*\ (r \cdot xvec)$› ‹$(r \cdot xvec)\ \sharp*\ \Psi$›
moreover from ‹$(r \cdot xvec)\ \sharp*\ A_P$› ‹$(p \cdot A_P)\ \sharp*\ (r \cdot xvec)$› ‹$(r \cdot xvec)\ \sharp*\ \Psi_P$› *Sp* **have** $(r \cdot xvec)\ \sharp*\ (p \cdot \Psi_P)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(r \cdot xvec)\ \sharp*\ A_Q$› ‹$(q \cdot A_Q)\ \sharp*\ (r \cdot xvec)$› ‹$(r \cdot xvec)\ \sharp*\ \Psi_Q$› *Sq* **have** $(r \cdot xvec)\ \sharp*\ (q \cdot \Psi_Q)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(r \cdot xvec)\ \sharp*\ P$›
moreover from ‹$(r \cdot xvec)\ \sharp*\ A_Q$› ‹$(q \cdot A_Q)\ \sharp*\ (r \cdot xvec)$› ‹$(r \cdot xvec)\ \sharp*\ M$› *Sq* **have** $(r \cdot xvec)\ \sharp*\ (q \cdot M)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(r \cdot xvec)\ \sharp*\ Q$›
moreover from ‹$(r \cdot xvec)\ \sharp*\ A_P$› ‹$(p \cdot A_P)\ \sharp*\ (r \cdot xvec)$› ‹$(r \cdot xvec)\ \sharp*\ K$› *Sp* **have** $(r \cdot xvec)\ \sharp*\ (p \cdot K)$
by(*simp add*: *freshChainSimps*)
ultimately have $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu*(r \cdot xvec)|\!)((r \cdot P') \parallel (r \cdot Q'))$
by(*rule-tac cComm2*)
with ‹$(r \cdot xvec)\ \sharp*\ P'$› ‹$(r \cdot xvec)\ \sharp*\ Q'$› *Sr*
show *?thesis*
by(*subst resChainAlpha*) *auto*
qed
}
note *Goal* = *this*

note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$›
moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹$A_P \sharp{*} \Psi$› ‹$A_P \sharp{*} P$› ‹$A_P \sharp{*} Q$› ‹$A_P \sharp{*} M$› ‹$A_P \sharp{*} A_Q$›
obtain $A_P{}'$ **where** *extractFrame* $P = \langle A_P{}', \Psi_P\rangle$ **and** *distinct* $A_P{}'$ **and** $A_P{}' \sharp{*} \Psi$ **and** $A_P{}' \sharp{*} P$ **and** $A_P{}' \sharp{*} Q$ **and** $A_P{}' \sharp{*} M$ **and** $A_P{}' \sharp{*} A_Q$
by(*rule-tac* C=(Ψ, P, Q, M, A_Q) **in** *distinctFrame*) *auto*
moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp{*} \Psi$› ‹$A_Q \sharp{*} P$› ‹$A_Q \sharp{*} Q$› ‹$A_Q \sharp{*} K$› ‹$A_P{}' \sharp{*} A_Q$›
obtain $A_Q{}'$ **where** *extractFrame* $Q = \langle A_Q{}', \Psi_Q\rangle$ **and** *distinct* $A_Q{}'$ **and** $A_Q{}' \sharp{*} \Psi$ **and** $A_Q{}' \sharp{*} P$ **and** $A_Q{}' \sharp{*} Q$ **and** $A_Q{}' \sharp{*} K$ **and** $A_P{}' \sharp{*} A_Q{}'$
by(*rule-tac* C=(Ψ, P, Q, K, $A_P{}'$) **in** *distinctFrame*) *auto*
ultimately show *?thesis* **using** ‹*xvec* $\sharp{*}$ Q›
by(*rule-tac Goal*)
qed

lemma *semanticsCasesAux*[*consumes 1*, *case-names cInput cOutput cCase cPar1 cPar2 cComm1 cComm2 cOpen cScope cBang*]:
fixes Ψ :: *'b*
and *cP* :: (*'a*, *'b*, *'c*) *psi*
and *cRs* :: (*'a*, *'b*, *'c*) *residual*
and *C* :: *'d*::*fs-name*
and *x* :: *name*

assumes $\Psi \rhd cP \longmapsto cRs$
and *rInput*: $\bigwedge M\ K\ xvec\ N\ Tvec\ P.\ [\![cP = M(\!|\lambda{*}xvec\ N|\!).P;\ cRs = K(\!|(N[xvec::=Tvec])|\!) \prec P[xvec::=Tvec];$
$\Psi \vdash M \leftrightarrow K;\ distinct\ xvec;\ set\ xvec \subseteq supp\ N;$
$length\ xvec{=}length\ Tvec;$
$xvec \sharp{*} Tvec;\ xvec \sharp{*} \Psi;\ xvec \sharp{*} M;\ xvec \sharp{*} K;$
$xvec \sharp{*} C]\!] \Longrightarrow Prop$
and *rOutput*: $\bigwedge M\ K\ N\ P.\ [\![cP = M\langle N\rangle.P;\ cRs = K\langle N\rangle \prec P;\ \Psi \vdash M \leftrightarrow K]\!] \Longrightarrow Prop$
and *rCase*: $\bigwedge Cs\ P\ \varphi.\ [\![cP = Cases\ Cs;\ \Psi \rhd P \longmapsto cRs;\ (\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P]\!] \Longrightarrow Prop$

and *rPar1*: $\bigwedge \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q.\ [\![cP = P \parallel Q;\ cRs = \alpha \prec (P' \parallel Q);$
$(\Psi \otimes \Psi_Q) \rhd P \longmapsto (\alpha \prec P');\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$A_Q \sharp{*} P;\ A_Q \sharp{*} Q;\ A_Q \sharp{*} \Psi;\ A_Q \sharp{*} \alpha;\ A_Q \sharp{*} C;\ A_Q \sharp{*} P';\ bn\ \alpha \sharp{*} \Psi;\ bn\ \alpha \sharp{*} \Psi_Q;$
$bn\ \alpha \sharp{*} Q;\ bn\ \alpha \sharp{*} P;\ bn\ \alpha \sharp{*} subject\ \alpha;\ bn\ \alpha \sharp{*} C;\ distinct(bn\ \alpha)]\!] \Longrightarrow$
$Prop$
and *rPar2*: $\bigwedge \Psi_P\ Q\ \alpha\ Q'\ P\ A_P.\ [\![cP = P \parallel Q;\ cRs = \alpha \prec (P \parallel Q');$
$(\Psi \otimes \Psi_P) \rhd Q \longmapsto \alpha \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$A_P \sharp{*} P;\ A_P \sharp{*} Q;\ A_P \sharp{*} \Psi;\ A_P \sharp{*} \alpha;\ A_P \sharp{*} C;$
$A_P \sharp{*} Q';\ bn\ \alpha \sharp{*} \Psi;\ bn\ \alpha \sharp{*} \Psi_P;\ bn\ \alpha \sharp{*} P;\ bn\ \alpha \sharp{*} Q;\ bn\ \alpha \sharp{*} subject$

α; *bn* α $\sharp*$ C; *distinct*(*bn* α)$\rrbracket \Longrightarrow$ *Prop*

and *rComm1*: $\bigwedge \Psi_Q$ P M N P' A_P Ψ_P Q K *xvec* Q' A_Q.
$\llbracket cP = P \parallel Q$; $cRs = \tau \prec (\!|\nu{*}xvec|\!) P' \parallel Q'$;
$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P \rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N \rangle \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; $A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$;
$A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp*$ *xvec*; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* Q'$; $A_Q \sharp*$ *xvec*;
xvec $\sharp* \Psi$; *xvec* $\sharp* \Psi_P$; *xvec* $\sharp* \Psi_Q$; *xvec* $\sharp* P$; *xvec* $\sharp* M$; *xvec* $\sharp* Q$;
xvec $\sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; *xvec* $\sharp* C$; *distinct xvec*$\rrbracket \Longrightarrow$ *Prop*

and *rComm2*: $\bigwedge \Psi_Q$ P M *xvec* N P' A_P Ψ_P Q K Q' A_Q.
$\llbracket cP = P \parallel Q$; $cRs = \tau \prec (\!|\nu{*}xvec|\!) P' \parallel Q'$;
$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N \rangle \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P \rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; $A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$;
$A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp*$ *xvec*; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* Q'$; $A_Q \sharp*$ *xvec*;
xvec $\sharp* \Psi$; *xvec* $\sharp* \Psi_P$; *xvec* $\sharp* \Psi_Q$; *xvec* $\sharp* P$; *xvec* $\sharp* M$; *xvec* $\sharp* Q$;
xvec $\sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; *xvec* $\sharp* C$; *distinct xvec*$\rrbracket \Longrightarrow$ *Prop*

and *rOpen*: $\bigwedge P$ M *xvec* *yvec* N P' x.
$\llbracket cP = (\!|\nu x|\!) P$; $cRs = M(\!|\nu{*}(xvec@x\#yvec)|\!)\langle N \rangle \prec P'$;
$\Psi \rhd P \longmapsto M(\!|\nu{*}(xvec@yvec)|\!)\langle N \rangle \prec P'$; $x \in$ *supp* N; $x \sharp$ *xvec*; $x \sharp$ *yvec*; $x \sharp M$; $x \sharp \Psi$; *distinct xvec*; *distinct yvec*;
xvec $\sharp* \Psi$; *xvec* $\sharp* P$; *xvec* $\sharp* M$; *xvec* $\sharp*$ *yvec*; *yvec* $\sharp* \Psi$; *yvec* $\sharp* P$; *yvec* $\sharp* M$; *xvec* $\sharp* C$; $x \sharp C$; *yvec* $\sharp* C\rrbracket \Longrightarrow$
Prop

and *rScope*: $\bigwedge P$ α P' x. $\llbracket cP = (\!|\nu x|\!) P$; $cRs = \alpha \prec (\!|\nu x|\!) P'$;
$\Psi \rhd P \longmapsto \alpha \prec P'$; $x \sharp \Psi$; $x \sharp \alpha$; $x \sharp C$; *bn* α $\sharp* \Psi$; *bn* α $\sharp* P$; *bn* α $\sharp*$ *subject* α; *bn* α $\sharp* C$; *distinct*(*bn* α)$\rrbracket \Longrightarrow$ *Prop*

and *rBang*: $\bigwedge P$. $\llbracket cP = !P$; $\Psi \rhd P \parallel !P \longmapsto cRs$; *guarded* $P\rrbracket \Longrightarrow$ *Prop*

shows *Prop*

using ‹$\Psi \rhd cP \longmapsto cRs$›

proof(*cases rule*: *semantics.cases*)

case(*cInput* M K *xvec* N *Tvec* P)

obtain *p*::*name prm* **where** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* M$ **and** $(p \cdot xvec) \sharp* N$ **and** $(p \cdot xvec) \sharp* K$

and $(p \cdot xvec) \sharp* Tvec$ **and** $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* C$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm p*
by(*rule-tac xvec=xvec* **and** $c=(\Psi, M, K, N, P, C, Tvec)$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
from ‹$cP = M(\!|\lambda*xvec\ N|\!).P$› ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* P$› S
have $cP = M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P)$
by(*simp add*: *inputChainAlpha'*)
moreover from ‹$cRs = K(\!|(N[xvec::=Tvec])|\!) \prec P[xvec::=Tvec]$› ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* P$› S ‹$length\ xvec = length\ Tvec$› ‹*distinctPerm p*›
have $cRs = K(\!|((p \cdot N)[(p \cdot xvec)::=Tvec])|\!) \prec (p \cdot P)[(p \cdot xvec)::=Tvec]$
by(*auto simp add*: *substTerm.renaming renaming residualInject*)

moreover note ‹$\Psi \vdash M \leftrightarrow K$›
moreover from ‹*distinct xvec*› **have** $distinct(p \cdot xvec)$
by *simp*
moreover from ‹$(set\ xvec) \subseteq (supp\ N)$› **have** $(p \cdot (set\ xvec)) \subseteq (p \cdot (supp\ N))$
by *simp*
hence $set(p \cdot xvec) \subseteq supp(p \cdot N)$
by(*simp add*: *eqvts*)
moreover from ‹$length\ xvec = length\ Tvec$› **have** $length(p \cdot xvec) = length\ Tvec$
by *simp*
ultimately show *?thesis* **using** ‹$(p \cdot xvec) \sharp* Tvec$› ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* M$› ‹$(p \cdot xvec) \sharp* K$›
‹$(p \cdot xvec) \sharp* C$›
by(*rule rInput*)
next
case(*Output M K N P*)
thus *?thesis* **by**(*rule rOutput*)
next
case(*Case P* φ *Cs*)
thus *?thesis* **by**(*rule rCase*)
next
case($cPar1\ \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q$)
obtain *q*::*name prm* **where** $(bn(q \cdot \alpha)) \sharp* \Psi$ **and** $(bn(q \cdot \alpha)) \sharp* P$ **and** $(bn(q \cdot \alpha)) \sharp* Q$
and $(bn(q \cdot \alpha)) \sharp* \alpha$ **and** $(bn(q \cdot \alpha)) \sharp* A_Q$ **and** $(bn(q \cdot \alpha)) \sharp* P'$ **and** $(bn(q \cdot \alpha)) \sharp* \Psi_Q$
and *distinctPerm q*
and $(bn(q \cdot \alpha)) \sharp* C$ **and** *Sq*: $(set\ q) \subseteq set(bn\ \alpha) \times (set(bn(q \cdot \alpha)))$
by(*rule-tac xvec=bn* α **and** $c=(\Psi, P, Q, \alpha, A_Q, \Psi_Q, P', C)$ **in** *name-list-avoiding*) (*auto simp add*: *eqvts*)
obtain *p*::*name prm* **where** $(p \cdot A_Q) \sharp* \Psi$ **and** $(p \cdot A_Q) \sharp* P$ **and** $(p \cdot A_Q) \sharp* Q$
and $(p \cdot A_Q) \sharp* \alpha$ **and** $(p \cdot A_Q) \sharp* (q \cdot \alpha)$ **and** $(p \cdot A_Q) \sharp* P'$
and $(p \cdot A_Q) \sharp* (q \cdot P')$ **and** $(p \cdot A_Q) \sharp* \Psi_Q$ **and** $(p \cdot A_Q)$

$\sharp*$ C
and *Sp*: $(set\ p) \subseteq (set\ A_Q) \times (set(p \cdot A_Q))$ **and** *distinctPerm p*
by(*rule-tac xvec*=A_Q **and** *c*=$(\Psi,\ P,\ Q,\ \alpha,\ q \cdot \alpha,\ P',\ (q \cdot P'),\ \Psi_Q,\ C)$ **in** *name-list-avoiding*) *auto*
from ‹$A_Q \sharp* \alpha$› ‹$bn(q \cdot \alpha) \sharp* A_Q$› *Sq* ‹*distinctPerm q*› **have** $A_Q \sharp* (q \cdot \alpha)$
by(*subst fresh-star-bij*[*symmetric*, *of* - - *q*]) (*simp add*: *eqvts*)
from ‹$bn\ \alpha \sharp*$ *subject* α› ‹*distinctPerm q*› **have** $bn(q \cdot \alpha) \sharp*$ *subject*$(q \cdot \alpha)$
by(*subst fresh-star-bij*[*symmetric*, *of* - - *q*]) (*simp add*: *eqvts*)
from ‹*distinct*(*bn* α)› ‹*distinctPerm q*› **have** *distinct*$(bn(q \cdot \alpha))$
by(*subst distinctClosed*[*symmetric*, *of* - *q*]) (*simp add*: *eqvts*)
note ‹$cP = P \parallel Q$›

moreover from ‹$cRs = \alpha \prec (P' \parallel Q)$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$(bn(q \cdot \alpha)) \sharp* \alpha$› ‹$(bn(q \cdot \alpha)) \sharp* P'$› ‹$(bn(q \cdot \alpha)) \sharp* Q$› ‹$bn\ \alpha \sharp* Q$› *Sq*
have $cRs = (q \cdot \alpha) \prec (q \cdot P') \parallel Q$
by(*force simp add*: *residualAlpha*)
moreover from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$(bn(q \cdot \alpha)) \sharp* \alpha$› ‹$(bn(q \cdot \alpha)) \sharp* P'$› *Sq*
have *Trans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto (q \cdot \alpha) \prec (q \cdot P')$
by(*force simp add*: *residualAlpha*)
hence $A_Q \sharp* (q \cdot P')$ **using** ‹$bn(q \cdot \alpha) \sharp*$ *subject*$(q \cdot \alpha)$› ‹*distinct*$(bn(q \cdot \alpha))$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* (q \cdot \alpha)$›
by(*drule-tac freeFreshChainDerivative*) *auto*

from *Trans* **have** $(p \cdot (\Psi \otimes \Psi_Q)) \rhd (p \cdot P) \longmapsto p \cdot ((q \cdot \alpha) \prec (q \cdot P'))$
by(*rule semantics.eqvt*)
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* (q \cdot \alpha)$› ‹$A_Q \sharp* (q \cdot P')$› ‹$(bn(q \cdot \alpha)) \sharp* A_Q$› ‹$(p \cdot A_Q) \sharp* (q \cdot \alpha)$›
‹$(p \cdot A_Q) \sharp* \Psi$› ‹$(p \cdot A_Q) \sharp* P$› ‹$(p \cdot A_Q) \sharp* (q \cdot P')$› *Sp*
have $\Psi \otimes (p \cdot \Psi_Q) \rhd P \longmapsto (q \cdot \alpha) \prec (q \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from ‹*extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$› ‹$(p \cdot A_Q) \sharp* \Psi_Q$› *Sp* **have** *extractFrame* $Q = \langle (p \cdot A_Q),\ (p \cdot \Psi_Q)\rangle$
by(*simp add*: *frameChainAlpha′ eqvts*)
moreover from ‹$(bn(q \cdot \alpha)) \sharp* \Psi_Q$› ‹$(bn(q \cdot \alpha)) \sharp* A_Q$› ‹$(p \cdot A_Q) \sharp* (q \cdot \alpha)$› *Sp* **have** $(bn(q \cdot \alpha)) \sharp* (p \cdot \Psi_Q)$
by(*simp add*: *freshAlphaPerm*)
moreover from ‹*distinct* A_Q› **have** *distinct*$(p \cdot A_Q)$ **by** *simp*
ultimately show *?thesis*
using ‹$(p \cdot A_Q) \sharp* P$› ‹$(p \cdot A_Q) \sharp* Q$› ‹$(p \cdot A_Q) \sharp* \Psi$› ‹$(p \cdot A_Q) \sharp* (q \cdot \alpha)$›
‹$(p \cdot A_Q) \sharp* (q \cdot P')$› ‹$(bn(q \cdot \alpha)) \sharp* \Psi$› ‹$(bn(q \cdot \alpha)) \sharp* Q$› ‹$(bn(q \cdot \alpha)) \sharp* P$›
‹$(bn(q \cdot \alpha)) \sharp* C$› ‹$(p \cdot A_Q) \sharp* C$› ‹$bn\ (q \cdot \alpha) \sharp*$ *subject* $(q \cdot \alpha)$› ‹*distinct*$(bn(q \cdot \alpha))$›
by(*rule-tac rPar1*)
next
case(*cPar2* $\Psi_P\ Q\ \alpha\ Q'\ P\ A_P$)
obtain *q*::*name prm* **where** $(bn(q \cdot \alpha)) \sharp* \Psi$ **and** $(bn(q \cdot \alpha)) \sharp* P$ **and** $(bn(q \cdot \alpha)) \sharp* Q$
and $(bn(q \cdot \alpha)) \sharp* \alpha$ **and** $(bn(q \cdot \alpha)) \sharp* A_P$ **and** $(bn(q \cdot \alpha))$

$\sharp*$ Q' **and** $(bn(q \cdot \alpha))$ $\sharp*$ Ψ_P
and *distinctPerm q*
and $(bn(q \cdot \alpha))$ $\sharp*$ C **and** *Sq*: $(set\ q) \subseteq set(bn\ \alpha) \times (set(bn(q \cdot \alpha)))$
by(*rule-tac xvec*=*bn* α **and** *c*=$(\Psi, P, Q, \alpha, A_P, \Psi_P, Q', C)$ **in** *name-list-avoiding*) (*auto simp add*: *eqvts*)
obtain *p*::*name prm* **where** $(p \cdot A_P)$ $\sharp*$ Ψ **and** $(p \cdot A_P)$ $\sharp*$ P **and** $(p \cdot A_P)$ $\sharp*$ Q
and $(p \cdot A_P)$ $\sharp*$ α **and** $(p \cdot A_P)$ $\sharp*$ $(q \cdot \alpha)$ **and** $(p \cdot A_P)$ $\sharp*$ Q'
and $(p \cdot A_P)$ $\sharp*$ $(q \cdot Q')$ **and** $(p \cdot A_P)$ $\sharp*$ Ψ_P **and** $(p \cdot A_P)$ $\sharp*$ C
and *Sp*: $(set\ p) \subseteq (set\ A_P) \times (set(p \cdot A_P))$ **and** *distinctPerm p*
by(*rule-tac xvec*=A_P **and** *c*=$(\Psi, P, Q, \alpha, q \cdot \alpha, Q', (q \cdot Q'), \Psi_P, C)$ **in** *name-list-avoiding*) *auto*
from ‹A_P $\sharp*$ α› ‹$bn(q \cdot \alpha)$ $\sharp*$ A_P› *Sq* ‹*distinctPerm q*› **have** A_P $\sharp*$ $(q \cdot \alpha)$
by(*subst fresh-star-bij*[*symmetric*, *of* - - *q*]) (*simp add*: *eqvts*)
from ‹*bn* α $\sharp*$ *subject* α› ‹*distinctPerm q*› **have** $bn(q \cdot \alpha)$ $\sharp*$ $subject(q \cdot \alpha)$
by(*subst fresh-star-bij*[*symmetric*, *of* - - *q*]) (*simp add*: *eqvts*)
from ‹$distinct(bn\ \alpha)$› ‹*distinctPerm q*› **have** $distinct(bn(q \cdot \alpha))$
by(*subst distinctClosed*[*symmetric*, *of* - *q*]) (*simp add*: *eqvts*)
note ‹$cP = P \parallel Q$›

moreover from ‹$cRs = \alpha \prec (P \parallel Q')$› ‹*bn* α $\sharp*$ *subject* α› ‹$(bn(q \cdot \alpha))$ $\sharp*$ α› ‹$(bn(q \cdot \alpha))$ $\sharp*$ Q'› ‹$(bn(q \cdot \alpha))$ $\sharp*$ P› ‹*bn* α $\sharp*$ P› *Sq*
have $cRs = (q \cdot \alpha) \prec P \parallel (q \cdot Q')$
by(*force simp add*: *residualAlpha*)
moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$› ‹*bn* α $\sharp*$ *subject* α› ‹$(bn(q \cdot \alpha))$ $\sharp*$ α› ‹$(bn(q \cdot \alpha))$ $\sharp*$ Q'› *Sq*
have *Trans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto (q \cdot \alpha) \prec (q \cdot Q')$
by(*force simp add*: *residualAlpha*)
hence A_P $\sharp*$ $(q \cdot Q')$ **using** ‹$bn(q \cdot \alpha)$ $\sharp*$ $subject(q \cdot \alpha)$› ‹$distinct(bn(q \cdot \alpha))$› ‹$A_P$ $\sharp*$ Q› ‹A_P $\sharp*$ $(q \cdot \alpha)$›
by(*drule-tac freeFreshChainDerivative*) *auto*

from *Trans* **have** $(p \cdot (\Psi \otimes \Psi_P)) \rhd (p \cdot Q) \longmapsto p \cdot ((q \cdot \alpha) \prec (q \cdot Q'))$
by(*rule semantics.eqvt*)
with ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ Q› ‹A_P $\sharp*$ $(q \cdot \alpha)$› ‹A_P $\sharp*$ $(q \cdot Q')$› ‹$(bn(q \cdot \alpha))$ $\sharp*$ A_P› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot \alpha)$›
‹$(p \cdot A_P)$ $\sharp*$ Ψ› ‹$(p \cdot A_P)$ $\sharp*$ Q› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot Q')$› *Sp*
have $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto (q \cdot \alpha) \prec (q \cdot Q')$ **by**(*simp add*: *eqvts*)
moreover from ‹$extractFrame\ P = \langle A_P, \Psi_P \rangle$› ‹$(p \cdot A_P)$ $\sharp*$ Ψ_P› *Sp* **have** $extractFrame\ P = \langle (p \cdot A_P), (p \cdot \Psi_P) \rangle$
by(*simp add*: *frameChainAlpha′ eqvts*)
moreover from ‹$(bn(q \cdot \alpha))$ $\sharp*$ Ψ_P› ‹$(bn(q \cdot \alpha))$ $\sharp*$ A_P› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot \alpha)$› *Sp* **have** $(bn(q \cdot \alpha))$ $\sharp*$ $(p \cdot \Psi_P)$
by(*simp add*: *freshAlphaPerm*)
moreover from ‹$distinct\ A_P$› **have** $distinct(p \cdot A_P)$ **by** *simp*
ultimately show *?thesis*
using ‹$(p \cdot A_P)$ $\sharp*$ P› ‹$(p \cdot A_P)$ $\sharp*$ Q› ‹$(p \cdot A_P)$ $\sharp*$ Ψ› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot \alpha)$›

‹$(p \cdot A_P)$ ♯* $(q \cdot Q')$› ‹$(bn(q \cdot \alpha))$ ♯* Ψ› ‹$(bn(q \cdot \alpha))$ ♯* Q› ‹$(bn(q \cdot \alpha))$ ♯* P›
‹$(bn(q \cdot \alpha))$ ♯* C› ‹$(p \cdot A_P)$ ♯* C› ‹$bn\ (q \cdot \alpha)$ ♯* *subject* $(q \cdot \alpha)$› ‹*distinct*$(bn(q \cdot \alpha))$›
by(*rule-tac rPar2*)
next
case(*cComm1* Ψ_Q P M N P' A_P Ψ_P Q K *xvec* Q' A_Q)
obtain *r*::*name prm* **where** $(r \cdot xvec)$ ♯* Ψ **and** $(r \cdot xvec)$ ♯* P **and** $(r \cdot xvec)$ ♯* Q **and** $(r \cdot xvec)$ ♯* M
and $(r \cdot xvec)$ ♯* K **and** $(r \cdot xvec)$ ♯* N **and** $(r \cdot xvec)$ ♯* A_P **and** $(r \cdot xvec)$ ♯* A_Q
and $(r \cdot xvec)$ ♯* P' **and** $(r \cdot xvec)$ ♯* Q' **and** $(r \cdot xvec)$ ♯* Ψ_P **and** $(r \cdot xvec)$ ♯* Ψ_Q
and $(r \cdot xvec)$ ♯* C **and** *Sr*: $(set\ r) \subseteq (set\ xvec) \times (set(r \cdot xvec))$ **and** *distinctPerm r*
by(*rule-tac xvec=xvec* **and** *c*=(Ψ, P, Q, M, K, N, A_P, A_Q, Ψ_P, Ψ_Q, P', Q', C) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts*)

obtain *q*::*name prm* **where** $(q \cdot A_Q)$ ♯* Ψ **and** $(q \cdot A_Q)$ ♯* P **and** $(q \cdot A_Q)$ ♯* Q **and** $(q \cdot A_Q)$ ♯* K
and $(q \cdot A_Q)$ ♯* N **and** $(q \cdot A_Q)$ ♯* *xvec* **and** $(q \cdot A_Q)$ ♯* Q' **and** $(q \cdot A_Q)$ ♯* P'
and $(q \cdot A_Q)$ ♯* Ψ_P **and** $(q \cdot A_Q)$ ♯* A_P **and** $(q \cdot A_Q)$ ♯* Ψ_Q **and** $(q \cdot A_Q)$ ♯* $(r \cdot xvec)$
and $(q \cdot A_Q)$ ♯* C **and** *Sq*: $(set\ q) \subseteq (set\ A_Q) \times (set(q \cdot A_Q))$
by(*rule-tac xvec=*A_Q **and** *c*=(Ψ, P, Q, K, N, *xvec*, $r \cdot xvec$, Ψ_Q, A_P, Ψ_P, Q', P', C) **in** *name-list-avoiding*) *clarsimp*

obtain *p*::*name prm* **where** $(p \cdot A_P)$ ♯* Ψ **and** $(p \cdot A_P)$ ♯* P **and** $(p \cdot A_P)$ ♯* Q **and** $(p \cdot A_P)$ ♯* M
and $(p \cdot A_P)$ ♯* N **and** $(p \cdot A_P)$ ♯* *xvec* **and** $(p \cdot A_P)$ ♯* Q' **and** $(p \cdot A_P)$ ♯* A_Q
and $(p \cdot A_P)$ ♯* P' **and** $(p \cdot A_P)$ ♯* Ψ_P **and** $(p \cdot A_P)$ ♯* Ψ_Q **and** $(p \cdot A_P)$ ♯* $(q \cdot A_Q)$
and $(p \cdot A_P)$ ♯* C **and** $(p \cdot A_P)$ ♯* $(r \cdot xvec)$ **and** *Sp*: $(set\ p) \subseteq (set\ A_P) \times (set(p \cdot A_P))$
by(*rule-tac xvec=*A_P **and** *c*=(Ψ, P, Q, M, N, *xvec*, $r \cdot xvec$, A_Q, $q \cdot A_Q$, Ψ_Q, Ψ_P, Q', P', C) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)

have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P \rangle$ **by** *fact*
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$ **by** *fact*

from ‹A_P ♯* Q› *FrQ* ‹A_P ♯* A_Q› **have** A_P ♯* Ψ_Q
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹A_Q ♯* P› *FrP* ‹A_P ♯* A_Q› **have** A_Q ♯* Ψ_P
by(*drule-tac extractFrameFreshChain*) *auto*
note ‹$cP = P \parallel Q$›

moreover from ‹$(r \cdot xvec) \sharp* P'$› ‹$(r \cdot xvec) \sharp* Q'$› **have** $(r \cdot xvec) \sharp* (P' \parallel Q')$
by *simp*
with ‹$cRs = \tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$› ‹$(r \cdot xvec) \sharp* N$› *Sr*
have $cRs = \tau \prec (\!|\nu*(r \cdot xvec)|\!)(r \cdot (P' \parallel Q'))$ **by**(*simp add*: *resChainAlpha residualInject*)
hence $cRs = \tau \prec (\!|\nu*(r \cdot xvec)|\!)((r \cdot P') \parallel (r \cdot Q'))$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› *Sr* ‹*distinctPerm r*› ‹$xvec \sharp* P$› ‹$(r \cdot xvec) \sharp* P$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|(r \cdot N)|\!) \prec (r \cdot P')$
by(*rule inputAlpha*)
hence $(q \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto (q \cdot M)(\!|(r \cdot N)|\!) \prec (r \cdot P')$ **using** *Sq* ‹$A_Q \sharp* P$› ‹$(q \cdot A_Q) \sharp* P$›
by(*rule-tac inputPermFrameSubject*) (*assumption* | *simp*)+
hence *PTrans*: $\Psi \otimes (q \cdot \Psi_Q) \rhd P \longmapsto (q \cdot M)(\!|(r \cdot N)|\!) \prec (r \cdot P')$ **using** *Sq* ‹$A_Q \sharp* \Psi$› ‹$(q \cdot A_Q) \sharp* \Psi$›
by(*simp add*: *eqvts*)

moreover from ‹$extractFrame\ P = \langle A_P, \Psi_P \rangle$› *Sp* ‹$(p \cdot A_P) \sharp* \Psi_P$›
have *FrP*: $extractFrame\ P = \langle (p \cdot A_P), (p \cdot \Psi_P) \rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹$distinct\ A_P$› **have** $distinct(p \cdot A_P)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N \rangle \prec Q'$› *Sr* ‹$(r \cdot xvec) \sharp* N$› ‹$(r \cdot xvec) \sharp* Q'$›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*(r \cdot xvec)|\!)\langle (r \cdot N) \rangle \prec (r \cdot Q')$
by(*simp add*: *boundOutputChainAlpha''* *residualInject*)
hence $(p \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto (p \cdot K)(\!|\nu*(r \cdot xvec)|\!)\langle (r \cdot N) \rangle \prec (r \cdot Q')$ **using** *Sp* ‹$A_P \sharp* Q$› ‹$(p \cdot A_P) \sharp* Q$› ‹$(r \cdot xvec) \sharp* K$›‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* A_P$›
by(*rule-tac outputPermFrameSubject*) (*assumption* | *auto*)

hence *QTrans*: $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto (p \cdot K)(\!|\nu*(r \cdot xvec)|\!)\langle (r \cdot N) \rangle \prec (r \cdot Q')$ **using** *Sp* ‹$A_P \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* \Psi$›
by(*simp add*: *eqvts*)

moreover from ‹$extractFrame\ Q = \langle A_Q, \Psi_Q \rangle$› *Sq* ‹$(q \cdot A_Q) \sharp* \Psi_Q$›
have *FrQ*: $extractFrame\ Q = \langle (q \cdot A_Q), (q \cdot \Psi_Q) \rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹$distinct\ A_Q$› **have** $distinct(q \cdot A_Q)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(p \cdot q \cdot (\Psi \otimes \Psi_P \otimes \Psi_Q)) \vdash (p \cdot q \cdot M) \leftrightarrow (p \cdot q \cdot K)$
by(*rule-tac chanEqClosed*)+
with ‹$A_P \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$(q \cdot A_Q) \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$(q \cdot A_Q) \sharp* \Psi_P$›
‹$A_P \sharp* \Psi_Q$› ‹$(p \cdot A_P) \sharp* \Psi_Q$› ‹$A_P \sharp* M$› ‹$(p \cdot A_P) \sharp* M$› ‹$(q \cdot A_Q) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (q \cdot A_Q)$›

‹A_Q ♯* K› ‹(q · A_Q) ♯* K› ‹A_P ♯* A_Q› ‹(p · A_P) ♯* A_Q› *Sp Sq*
have $\Psi \otimes (p \cdot \Psi_P) \otimes (q \cdot \Psi_Q) \vdash (q \cdot M) \leftrightarrow (p \cdot K)$
by(*simp add*: *eqvts freshChainSimps*)
moreover note ‹(p · A_P) ♯* Ψ›
moreover from ‹(p · A_P) ♯* A_Q› ‹(p · A_P) ♯* (q · A_Q)› ‹(p · A_P) ♯* Ψ_Q› *Sq*
have (p · A_P) ♯* (q · Ψ_Q)
by(*simp add*: *freshChainSimps*)
moreover note ‹(p · A_P) ♯* P›
moreover from ‹(p · A_P) ♯* A_Q› ‹(p · A_P) ♯* (q · A_Q)› ‹(p · A_P) ♯* M› *Sq*
have (p · A_P) ♯* (q · M)
by(*simp add*: *freshChainSimps*)
moreover from ‹(p · A_P) ♯* *xvec*› ‹(p · A_P) ♯* (r · *xvec*)› ‹(p · A_P) ♯* N› *Sr*
have (p · A_P) ♯* (r · N)
by(*simp add*: *freshChainSimps*)
moreover from ‹(p · A_P) ♯* *xvec*› ‹(p · A_P) ♯* (r · *xvec*)› ‹(p · A_P) ♯* P'› *Sr*
have (p · A_P) ♯* (r · P')
by(*simp add*: *freshChainSimps*)
moreover note ‹(p · A_P) ♯* Q›
moreover from ‹(p · A_P) ♯* *xvec*› ‹(p · A_P) ♯* (r · *xvec*)› ‹(p · A_P) ♯* Q'› *Sr*
have (p · A_P) ♯* (r · Q')
by(*simp add*: *freshChainSimps*)
moreover note ‹(p · A_P) ♯* (q · A_Q)› ‹(p · A_P) ♯* (r · *xvec*)› ‹(q · A_Q) ♯* Ψ›
moreover from ‹(q · A_Q) ♯* A_P› ‹(p · A_P) ♯* A_Q› ‹(q · A_Q) ♯* Ψ_P› ‹(p · A_P) ♯* (q · A_Q)› *Sp Sq* **have** (q · A_Q) ♯* (p · Ψ_P)
by(*simp add*: *freshChainSimps*)
moreover note ‹(q · A_Q) ♯* P›
moreover from ‹(q · A_Q) ♯* A_P› ‹(p · A_P) ♯* A_Q› ‹(q · A_Q) ♯* K› ‹(p · A_P) ♯* (q · A_Q)› *Sp Sq* **have** (q · A_Q) ♯* (p · K)
by(*simp add*: *freshChainSimps*)
moreover from ‹(q · A_Q) ♯* *xvec*› ‹(q · A_Q) ♯* (r · *xvec*)› ‹(q · A_Q) ♯* N› *Sr*
have (q · A_Q) ♯* (r · N)
by(*simp add*: *freshChainSimps*)
moreover from ‹(q · A_Q) ♯* *xvec*› ‹(q · A_Q) ♯* (r · *xvec*)› ‹(q · A_Q) ♯* P'› *Sr*
have (q · A_Q) ♯* (r · P')
by(*simp add*: *freshChainSimps*)
moreover note ‹(q · A_Q) ♯* Q›
moreover from ‹(q · A_Q) ♯* *xvec*› ‹(q · A_Q) ♯* (r · *xvec*)› ‹(q · A_Q) ♯* Q'› *Sr*
have (q · A_Q) ♯* (r · Q')
by(*simp add*: *freshChainSimps*)
moreover note ‹(q · A_Q) ♯* (r · *xvec*)› ‹(r · *xvec*) ♯* Ψ›
moreover from ‹(r · *xvec*) ♯* A_P› ‹(p · A_P) ♯* (r · *xvec*)› ‹(r · *xvec*) ♯* Ψ_P› *Sp* **have** (r · *xvec*) ♯* (p · Ψ_P)
by(*simp add*: *freshChainSimps*)
moreover from ‹(r · *xvec*) ♯* A_Q› ‹(q · A_Q) ♯* (r · *xvec*)› ‹(r · *xvec*) ♯* Ψ_Q› *Sq* **have** (r · *xvec*) ♯* (q · Ψ_Q)
by(*simp add*: *freshChainSimps*)
moreover note ‹(r · *xvec*) ♯* P›
moreover from ‹(r · *xvec*) ♯* A_Q› ‹(q · A_Q) ♯* (r · *xvec*)› ‹(r · *xvec*) ♯* M› *Sq*
have (r · *xvec*) ♯* (q · M)

by(*simp add*: *freshChainSimps*)
moreover note ‹$(r \cdot xvec) \sharp* Q$›
moreover from ‹$(r \cdot xvec) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* K$› *Sp*
have $(r \cdot xvec) \sharp* (p \cdot K)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P) \sharp* C$› ‹$(q \cdot A_Q) \sharp* C$› ‹$(r \cdot xvec) \sharp* C$›
moreover from ‹*distinct xvec*› **have** *distinct*$(r \cdot xvec)$ **by** *simp*
ultimately show *?thesis* **by**(*rule rComm1*)
next
case(*cComm2* Ψ_Q P M *xvec* N P' A_P Ψ_P Q K Q' A_Q)
obtain *r*::*name prm* **where** $(r \cdot xvec) \sharp* \Psi$ **and** $(r \cdot xvec) \sharp* P$ **and** $(r \cdot xvec) \sharp* Q$ **and** $(r \cdot xvec) \sharp* M$
and $(r \cdot xvec) \sharp* K$ **and** $(r \cdot xvec) \sharp* N$ **and** $(r \cdot xvec) \sharp* A_P$ **and** $(r \cdot xvec) \sharp* A_Q$
and $(r \cdot xvec) \sharp* P'$ **and** $(r \cdot xvec) \sharp* Q'$ **and** $(r \cdot xvec) \sharp* \Psi_P$ **and** $(r \cdot xvec) \sharp* \Psi_Q$
and $(r \cdot xvec) \sharp* C$ **and** *Sr*: $(set\ r) \subseteq (set\ xvec) \times (set(r \cdot xvec))$ **and** *distinctPerm r*
by(*rule-tac xvec*=*xvec* **and** *c*=(Ψ, P, Q, M, K, N, A_P, A_Q, Ψ_P, Ψ_Q, P', Q', C) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts*)

obtain *q*::*name prm* **where** $(q \cdot A_Q) \sharp* \Psi$ **and** $(q \cdot A_Q) \sharp* P$ **and** $(q \cdot A_Q) \sharp* Q$ **and** $(q \cdot A_Q) \sharp* K$
and $(q \cdot A_Q) \sharp* N$ **and** $(q \cdot A_Q) \sharp*$ *xvec* **and** $(q \cdot A_Q) \sharp* Q'$ **and** $(q \cdot A_Q) \sharp* P'$
and $(q \cdot A_Q) \sharp* \Psi_P$ **and** $(q \cdot A_Q) \sharp* A_P$ **and** $(q \cdot A_Q) \sharp* \Psi_Q$ **and** $(q \cdot A_Q) \sharp* (r \cdot xvec)$
and $(q \cdot A_Q) \sharp* C$ **and** *Sq*: $(set\ q) \subseteq (set\ A_Q) \times (set(q \cdot A_Q))$
by(*rule-tac xvec*=A_Q **and** *c*=(Ψ, P, Q, K, N, *xvec*, $r \cdot xvec$, Ψ_Q, A_P, Ψ_P, Q', P', C) **in** *name-list-avoiding*) *clarsimp*

obtain *p*::*name prm* **where** $(p \cdot A_P) \sharp* \Psi$ **and** $(p \cdot A_P) \sharp* P$ **and** $(p \cdot A_P) \sharp* Q$ **and** $(p \cdot A_P) \sharp* M$
and $(p \cdot A_P) \sharp* N$ **and** $(p \cdot A_P) \sharp*$ *xvec* **and** $(p \cdot A_P) \sharp* Q'$ **and** $(p \cdot A_P) \sharp* A_Q$
and $(p \cdot A_P) \sharp* P'$ **and** $(p \cdot A_P) \sharp* \Psi_P$ **and** $(p \cdot A_P) \sharp* \Psi_Q$ **and** $(p \cdot A_P) \sharp* (q \cdot A_Q)$
and $(p \cdot A_P) \sharp* C$ **and** $(p \cdot A_P) \sharp* (r \cdot xvec)$ **and** *Sp*: $(set\ p) \subseteq (set\ A_P) \times (set(p \cdot A_P))$
by(*rule-tac xvec*=A_P **and** *c*=(Ψ, P, Q, M, N, *xvec*, $r \cdot xvec$, A_Q, $q \cdot A_Q$, Ψ_Q, Ψ_P, Q', P', C) **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)

have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P \rangle$ **by** *fact*
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$ **by** *fact*

from ‹$A_P \sharp* Q$› *FrQ* ‹$A_P \sharp* A_Q$› **have** $A_P \sharp* \Psi_Q$
by(*drule-tac extractFrameFreshChain*) *auto*

from ‹A_Q ♯* P› FrP ‹A_P ♯* A_Q› **have** A_Q ♯* Ψ_P
by(*drule-tac extractFrameFreshChain*) *auto*

note ‹$cP = P \parallel Q$›
moreover from ‹$(r \cdot xvec)$ ♯* P'› ‹$(r \cdot xvec)$ ♯* Q'› **have** $(r \cdot xvec)$ ♯* $(P' \parallel Q')$
by *simp*
with ‹$cRs = \tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q')$› ‹$(r \cdot xvec)$ ♯* N› Sr
have $cRs = \tau \prec (\!|\nu{*}(r \cdot xvec)|\!)(r \cdot (P' \parallel Q'))$ **by**(*simp add*: *resChainAlpha residualInject*)
hence $cRs = \tau \prec (\!|\nu{*}(r \cdot xvec)|\!)((r \cdot P') \parallel (r \cdot Q'))$
by *simp*

moreover from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› Sr ‹$(r \cdot xvec)$ ♯* N› ‹$(r \cdot xvec)$ ♯* P'›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}(r \cdot xvec)|\!)\langle(r \cdot N)\rangle \prec (r \cdot P')$ **by**(*simp add*: *boundOutputChainAlpha''* *residualInject*)
hence $(q \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto (q \cdot M)(\!|\nu{*}(r \cdot xvec)|\!)\langle(r \cdot N)\rangle \prec (r \cdot P')$ **using** Sq ‹A_Q ♯* P› ‹$(q \cdot A_Q)$ ♯* P› ‹$(r \cdot xvec)$ ♯* M› ‹$(r \cdot xvec)$ ♯* A_Q› ‹$(q \cdot A_Q)$ ♯* $(r \cdot xvec)$›
by(*rule-tac outputPermFrameSubject*) (*assumption* | *auto*)
hence *PTrans*: $\Psi \otimes (q \cdot \Psi_Q) \rhd P \longmapsto (q \cdot M)(\!|\nu{*}(r \cdot xvec)|\!)\langle(r \cdot N)\rangle \prec (r \cdot P')$ **using** Sq ‹A_Q ♯* Ψ› ‹$(q \cdot A_Q)$ ♯* Ψ›
by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› Sp ‹$(p \cdot A_P)$ ♯* Ψ_P›
have *FrP*: *extractFrame* $P = \langle(p \cdot A_P), (p \cdot \Psi_P)\rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹*distinct* A_P› **have** *distinct*$(p \cdot A_P)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› Sr ‹*distinctPerm* r› ‹$xvec$ ♯* Q› ‹$(r \cdot xvec)$ ♯* Q›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|(r \cdot N)|\!) \prec (r \cdot Q')$ **by**(*rule inputAlpha*)
hence $(p \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto (p \cdot K)(\!|(r \cdot N)|\!) \prec (r \cdot Q')$ **using** Sp ‹A_P ♯* Q› ‹$(p \cdot A_P)$ ♯* Q›
by(*rule-tac inputPermFrameSubject*) (*assumption* | *simp*)+
hence *QTrans*: $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto (p \cdot K)(\!|(r \cdot N)|\!) \prec (r \cdot Q')$ **using** Sp ‹A_P ♯* Ψ› ‹$(p \cdot A_P)$ ♯* Ψ›
by(*simp add*: *eqvts*)

moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› Sq ‹$(q \cdot A_Q)$ ♯* Ψ_Q›
have *FrQ*: *extractFrame* $Q = \langle(q \cdot A_Q), (q \cdot \Psi_Q)\rangle$
by(*simp add*: *frameChainAlpha*)
moreover from ‹*distinct* A_Q› **have** *distinct*$(q \cdot A_Q)$ **by** *simp*

moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(p \cdot q \cdot (\Psi \otimes \Psi_P \otimes \Psi_Q)) \vdash (p \cdot q \cdot M) \leftrightarrow (p \cdot q \cdot K)$
by(*rule-tac chanEqClosed*)+
with ‹A_P ♯* Ψ› ‹$(p \cdot A_P)$ ♯* Ψ› ‹A_Q ♯* Ψ› ‹$(q \cdot A_Q)$ ♯* Ψ› ‹A_Q ♯* Ψ_P› ‹$(q \cdot$

$A_Q)$ $\sharp*$ Ψ_P›
‹A_P $\sharp*$ Ψ_Q› ‹$(p \cdot A_P)$ $\sharp*$ Ψ_Q› ‹A_P $\sharp*$ M› ‹$(p \cdot A_P)$ $\sharp*$ M› ‹$(q \cdot A_Q)$ $\sharp*$ A_P› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot A_Q)$›
‹A_Q $\sharp*$ K› ‹$(q \cdot A_Q)$ $\sharp*$ K› ‹A_P $\sharp*$ A_Q› ‹$(p \cdot A_P)$ $\sharp*$ A_Q› *Sp Sq*
have $\Psi \otimes (p \cdot \Psi_P) \otimes (q \cdot \Psi_Q) \vdash (q \cdot M) \leftrightarrow (p \cdot K)$
by(*simp add*: *eqvts freshChainSimps*)
moreover note ‹$(p \cdot A_P)$ $\sharp*$ Ψ›
moreover from ‹$(p \cdot A_P)$ $\sharp*$ A_Q› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot A_Q)$› ‹$(p \cdot A_P)$ $\sharp*$ Ψ_Q› *Sq* **have** $(p \cdot A_P)$ $\sharp*$ $(q \cdot \Psi_Q)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P)$ $\sharp*$ P›
moreover from ‹$(p \cdot A_P)$ $\sharp*$ A_Q› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot A_Q)$› ‹$(p \cdot A_P)$ $\sharp*$ M› *Sq* **have** $(p \cdot A_P)$ $\sharp*$ $(q \cdot M)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(p \cdot A_P)$ $\sharp*$ *xvec*› ‹$(p \cdot A_P)$ $\sharp*$ $(r \cdot xvec)$› ‹$(p \cdot A_P)$ $\sharp*$ N› *Sr* **have** $(p \cdot A_P)$ $\sharp*$ $(r \cdot N)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(p \cdot A_P)$ $\sharp*$ *xvec*› ‹$(p \cdot A_P)$ $\sharp*$ $(r \cdot xvec)$› ‹$(p \cdot A_P)$ $\sharp*$ P'› *Sr* **have** $(p \cdot A_P)$ $\sharp*$ $(r \cdot P')$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P)$ $\sharp*$ Q›
moreover from ‹$(p \cdot A_P)$ $\sharp*$ *xvec*› ‹$(p \cdot A_P)$ $\sharp*$ $(r \cdot xvec)$› ‹$(p \cdot A_P)$ $\sharp*$ Q'› *Sr* **have** $(p \cdot A_P)$ $\sharp*$ $(r \cdot Q')$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot A_Q)$› ‹$(p \cdot A_P)$ $\sharp*$ $(r \cdot xvec)$› ‹$(q \cdot A_Q)$ $\sharp*$ Ψ›
moreover from ‹$(q \cdot A_Q)$ $\sharp*$ A_P› ‹$(p \cdot A_P)$ $\sharp*$ A_Q› ‹$(q \cdot A_Q)$ $\sharp*$ Ψ_P› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot A_Q)$› *Sp Sq* **have** $(q \cdot A_Q)$ $\sharp*$ $(p \cdot \Psi_P)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q)$ $\sharp*$ P›
moreover from ‹$(q \cdot A_Q)$ $\sharp*$ A_P› ‹$(p \cdot A_P)$ $\sharp*$ A_Q› ‹$(q \cdot A_Q)$ $\sharp*$ K› ‹$(p \cdot A_P)$ $\sharp*$ $(q \cdot A_Q)$› *Sp Sq* **have** $(q \cdot A_Q)$ $\sharp*$ $(p \cdot K)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(q \cdot A_Q)$ $\sharp*$ *xvec*› ‹$(q \cdot A_Q)$ $\sharp*$ $(r \cdot xvec)$› ‹$(q \cdot A_Q)$ $\sharp*$ N› *Sr* **have** $(q \cdot A_Q)$ $\sharp*$ $(r \cdot N)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(q \cdot A_Q)$ $\sharp*$ *xvec*› ‹$(q \cdot A_Q)$ $\sharp*$ $(r \cdot xvec)$› ‹$(q \cdot A_Q)$ $\sharp*$ P'› *Sr* **have** $(q \cdot A_Q)$ $\sharp*$ $(r \cdot P')$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q)$ $\sharp*$ Q›
moreover from ‹$(q \cdot A_Q)$ $\sharp*$ *xvec*› ‹$(q \cdot A_Q)$ $\sharp*$ $(r \cdot xvec)$› ‹$(q \cdot A_Q)$ $\sharp*$ Q'› *Sr* **have** $(q \cdot A_Q)$ $\sharp*$ $(r \cdot Q')$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(q \cdot A_Q)$ $\sharp*$ $(r \cdot xvec)$› ‹$(r \cdot xvec)$ $\sharp*$ Ψ›
moreover from ‹$(r \cdot xvec)$ $\sharp*$ A_P› ‹$(p \cdot A_P)$ $\sharp*$ $(r \cdot xvec)$› ‹$(r \cdot xvec)$ $\sharp*$ Ψ_P› *Sp* **have** $(r \cdot xvec)$ $\sharp*$ $(p \cdot \Psi_P)$
by(*simp add*: *freshChainSimps*)
moreover from ‹$(r \cdot xvec)$ $\sharp*$ A_Q› ‹$(q \cdot A_Q)$ $\sharp*$ $(r \cdot xvec)$› ‹$(r \cdot xvec)$ $\sharp*$ Ψ_Q› *Sq* **have** $(r \cdot xvec)$ $\sharp*$ $(q \cdot \Psi_Q)$
by(*simp add*: *freshChainSimps*)

moreover note ‹$(r \cdot xvec) \sharp* P$›
moreover from ‹$(r \cdot xvec) \sharp* A_Q$› ‹$(q \cdot A_Q) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* M$› *Sq*
have $(r \cdot xvec) \sharp* (q \cdot M)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(r \cdot xvec) \sharp* Q$›
moreover from ‹$(r \cdot xvec) \sharp* A_P$› ‹$(p \cdot A_P) \sharp* (r \cdot xvec)$› ‹$(r \cdot xvec) \sharp* K$› *Sp*
have $(r \cdot xvec) \sharp* (p \cdot K)$
by(*simp add*: *freshChainSimps*)
moreover note ‹$(p \cdot A_P) \sharp* C$› ‹$(q \cdot A_Q) \sharp* C$› ‹$(r \cdot xvec) \sharp* C$›
moreover from ‹*distinct xvec*› **have** *distinct*$(r \cdot xvec)$ **by** *simp*
ultimately show *?thesis* **by**(*rule rComm2*)
next
case(*cOpen P M xvec yvec N P′ x*)
from ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$› **have** *distinct*$(xvec@yvec)$
by(*force dest*: *boundOutputDistinct*)
hence $xvec \sharp* yvec$ **by**(*induct xvec*) *auto*
obtain p **where** $(p \cdot yvec) \sharp* \Psi$ **and** $(p \cdot yvec) \sharp* P$ **and** $(p \cdot yvec) \sharp* M$
and $(p \cdot yvec) \sharp* yvec$ **and** $(p \cdot yvec) \sharp* N$ **and** $(p \cdot yvec) \sharp* P'$
and $x \sharp (p \cdot yvec)$ **and** $(p \cdot yvec) \sharp* xvec$
and $(p \cdot yvec) \sharp* C$ **and** *Sp*: $(set\ p) \subseteq (set\ yvec) \times (set(p \cdot yvec))$
by(*rule-tac xvec=yvec* **and** $c=(\Psi, P, M, xvec, yvec, N, P', x, C)$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain q **where** $(q \cdot xvec) \sharp* \Psi$ **and** $(q \cdot xvec) \sharp* P$ **and** $(q \cdot xvec) \sharp* M$
and $(q \cdot xvec) \sharp* xvec$ **and** $(q \cdot xvec) \sharp* N$ **and** $(q \cdot xvec) \sharp* P'$
and $x \sharp (q \cdot xvec)$ **and** $(q \cdot xvec) \sharp* yvec$
and $(q \cdot xvec) \sharp* p$ **and** $(q \cdot xvec) \sharp* (p \cdot yvec)$
and $(q \cdot xvec) \sharp* C$ **and** *Sq*: $(set\ q) \subseteq (set\ xvec) \times (set(q \cdot xvec))$
by(*rule-tac xvec=xvec* **and** $c=(\Psi, P, M, xvec, yvec, p \cdot yvec, N, P', x, p, C)$
in *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
obtain y::*name* **where** $y \sharp P$ **and** $y \sharp C$ **and** $y \sharp xvec$ **and** $y \sharp yvec$ **and** $y \neq x$
and $y \sharp N$
and $y \sharp (q \cdot xvec)$ **and** $y \sharp (p \cdot yvec)$ **and** $y \sharp M$ **and** $y \sharp \Psi$ **and** y
$\sharp P'$
by(*generate-fresh name*) (*auto simp add*: *freshChainSimps*)

from ‹$cP = (\!|\nu x|\!)P$› ‹$y \sharp P$› **have** $cP = (\!|\nu y|\!)([(x, y)] \cdot P)$ **by**(*simp add*: *alphaRes*)
moreover have $cRs = M(\!|\nu*((q \cdot xvec)@y\#(p \cdot yvec))|\!)\langle((q@(x, y)\#p) \cdot N)\rangle \prec$
$((q@(x, y)\#p) \cdot P')$
proof −
note ‹$cRs = M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$›
moreover have $(\!|\nu*(xvec@x\#yvec)|\!)N \prec' P' = (\!|\nu*xvec|\!)((\!|\nu x|\!)((\!|\nu*yvec|\!)N \prec'$
$P'))$ **by**(*simp add*: *boundOutputApp*)
moreover from ‹$(p \cdot yvec) \sharp* N$› ‹$(p \cdot yvec) \sharp* P'$› *Sp* **have** $\ldots = (\!|\nu*xvec|\!)((\!|\nu x|\!)((\!|\nu*(p$
$\cdot yvec)|\!)(p \cdot N) \prec' (p \cdot P')))$
by(*simp add*: *boundOutputChainAlpha″*)
moreover with ‹$y \sharp N$› ‹$y \sharp P'$› ‹$y \sharp (p \cdot yvec)$› ‹$y \sharp yvec$› ‹$x \sharp yvec$› ‹$x \sharp (p \cdot$
$yvec)$› *Sp*
have $\ldots = (\!|\nu*xvec|\!)((\!|\nu y|\!)((\!|\nu*(p \cdot yvec)|\!)(([(x, y)] \cdot p \cdot N) \prec' ([(x, y)] \cdot p \cdot$

P'))))
by(*subst alphaBoundOutput*[**where** *y*=*y*]) (*simp add*: *freshChainSimps eqvts*)+
moreover hence ... = ⦇ν*xvec⦈(⦇νy⦈(⦇ν*(p · yvec)⦈((((x, y)#p) · N) ≺′ (((x, y)#p) · P′))))
by *simp*
moreover from ‹(q · xvec) ♯* N› ‹(q · xvec) ♯* P′› ‹xvec ♯* yvec› ‹(p · yvec) ♯* xvec› ‹(q · xvec) ♯* yvec› ‹(q · xvec) ♯* (p · yvec)›
‹y ♯ xvec› ‹y ♯ (q · xvec)› ‹x ♯ xvec› ‹x ♯ (q · xvec)› *Sp Sq*
have ... = ⦇ν*(q · xvec)⦈(⦇νy⦈(⦇ν*(p · yvec)⦈((q · ((x, y)#p) · N) ≺′ (q · ((x, y)#p) · P′))))
apply(*subst boundOutputChainAlpha*[**where** *p*=*q* **and** *xvec*=*xvec* **and** *yvec*=*xvec*])
defer
apply *assumption*
apply *simp*
apply(*simp add*: *eqvts*)
apply(*simp add*: *eqvts*)
apply(*simp add*: *boundOutputFreshSet*(*4*))
apply(*rule conjI*)
apply(*simp add*: *freshChainSimps*)
apply(*simp add*: *freshChainSimps*)
done
moreover hence ... = ⦇ν*(q · xvec@y#(p · yvec))⦈((q@(x, y)#p) · N) ≺′ ((q@(x, y)#p) · P′)
by(*simp only*: *pt2*[*OF pt-name-inst*] *boundOutputApp BOresChain.simps*)
ultimately show *?thesis*
by(*simp only*: *residualInject*)
qed
moreover have Ψ ▷ ([(x, y)] · P) ⟼M⦇ν*((q · xvec)@(p · yvec))⦈⟨((q@(x, y)#p) · N)⟩ ≺ ((q@(x, y)#p) · P′)
proof −
note‹Ψ ▷ P ⟼M⦇ν*(xvec@yvec)⦈⟨N⟩ ≺ P′›
moreover from ‹(p · yvec) ♯* N› ‹(q · xvec) ♯* N› ‹xvec ♯* yvec› ‹(q · xvec) ♯* yvec› ‹(q · xvec) ♯* (p · yvec)› ‹(p · yvec) ♯* xvec› *Sp Sq*
have ((q@p) · (xvec @ yvec)) ♯* N **apply**(*simp only*: *eqvts*) **apply**(*simp only*: *pt2*[*OF pt-name-inst*])
by *simp*
moreover from ‹(p · yvec) ♯* P′› ‹(q · xvec) ♯* P′› ‹xvec ♯* yvec› ‹(q · xvec) ♯* yvec› ‹(q · xvec) ♯* (p · yvec)› ‹(p · yvec) ♯* xvec› *Sp Sq*
have ((q@p) · (xvec @ yvec)) ♯* P′ **by**(*simp del*: *freshAlphaPerm add*: *eqvts pt2*[*OF pt-name-inst*])
moreover from *Sp Sq* ‹xvec ♯* yvec› ‹(q · xvec) ♯* yvec› ‹(q · xvec) ♯* (p · yvec)› ‹(p · yvec) ♯* xvec›
have *Spq*: *set*(q@p) ⊆ *set*(xvec@yvec) × *set*((q@p) · (xvec@yvec))
by(*simp add*: *pt2*[*OF pt-name-inst*] *eqvts*) *blast*
ultimately have Ψ ▷ P ⟼M⦇ν*((q@p) · (xvec@yvec))⦈⟨((q@p) · N)⟩ ≺ ((q@p) · P′)
apply(*simp only*: *residualInject*)
by(*erule-tac rev-mp*) (*subst boundOutputChainAlpha*, *auto*)
with *Sp Sq* ‹xvec ♯* yvec› ‹(q · xvec) ♯* yvec› ‹(q · xvec) ♯* (p · yvec)› ‹(p ·

yvec) ♯∗ *xvec*›
have $\Psi \rhd P \longmapsto M(\!|\nu*((q \cdot xvec)@(p \cdot yvec))|\!)\langle((q@p) \cdot N)\rangle \prec ((q@p) \cdot P')$
by(*simp add*: *eqvts pt2*[*OF pt-name-inst*] *del*: *freshAlphaPerm*)
hence $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \longmapsto [(x, y)] \cdot (M(\!|\nu*((q \cdot xvec)@(p \cdot yvec))|\!)\langle((q@p) \cdot N)\rangle \prec ((q@p) \cdot P'))$
by(*rule semantics.eqvt*)
with ‹x ♯ Ψ› ‹y ♯ Ψ› ‹x ♯ M› ‹y ♯ M› ‹x ♯ *xvec*› ‹y ♯ *xvec*› ‹x ♯ $(q \cdot xvec)$› ‹y ♯ $(q \cdot xvec)$ › ‹x ♯ *yvec*› ‹y ♯ *yvec*› ‹x ♯ $(p \cdot yvec)$› ‹y ♯ $(p \cdot yvec)$› *Sp Sq*
show *?thesis*
apply(*simp add*: *eqvts pt2*[*OF pt-name-inst*])
by(*subst perm-compose*[*of q*], *simp*)+
qed
moreover from ‹$x \in supp\ N$› **have** $((q@(x, y)\#p) \cdot x) \in ((q@(x, y)\#p) \cdot (supp\ N))$
by(*simp add*: *pt-set-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹x ♯ *xvec*› ‹x ♯ *yvec*› ‹x ♯ $(q \cdot xvec)$› ‹x ♯ $(p \cdot yvec)$› ‹y ♯ *xvec*› ‹y ♯ $(q \cdot xvec)$› *Sp Sq*
have $y \in supp((q@(x, y)\#p) \cdot N)$ **by**(*simp add*: *pt2*[*OF pt-name-inst*] *calc-atm eqvts*)
moreover from ‹*distinct xvec*› **have** $distinct(q \cdot xvec)$ **by** *simp*
moreover from ‹*distinct yvec*› **have** $distinct(p \cdot yvec)$ **by** *simp*
moreover note ‹x ♯ $(q \cdot xvec)$› ‹x ♯ $(p \cdot yvec)$› ‹x ♯ M› ‹x ♯ Ψ›
‹$(q \cdot xvec)$ ♯∗ Ψ› ‹$(q \cdot xvec)$ ♯∗ P› ‹$(q \cdot xvec)$ ♯∗ M› ‹$(q \cdot xvec)$ ♯∗ $(p \cdot yvec)$›
‹$(p \cdot yvec)$ ♯∗ Ψ› ‹$(p \cdot yvec)$ ♯∗ P› ‹$(p \cdot yvec)$ ♯∗ M› ‹y ♯ $(q \cdot xvec)$› ‹y ♯ $(p \cdot yvec)$› ‹y ♯ M› ‹y ♯ C› ‹y ♯ Ψ›
‹$(p \cdot yvec)$ ♯∗ C› ‹$(q \cdot xvec)$ ♯∗ C›
ultimately show *Prop* **by**(*rule-tac rOpen*) (*assumption* | *simp*)+
next
case(*cScope* P α P' x)
obtain *p*::*name prm* **where** $(bn(p \cdot \alpha))$ ♯∗ Ψ **and** $(bn(p \cdot \alpha))$ ♯∗ P
and $(bn(p \cdot \alpha))$ ♯∗ α **and** $(bn(p \cdot \alpha))$ ♯∗ P' **and** x ♯ $bn(p \cdot \alpha)$
and *distinctPerm p*
and $(bn(p \cdot \alpha))$ ♯∗ C **and** *Sp*: $(set\ p) \subseteq set(bn\ \alpha) \times (set(bn(p \cdot \alpha)))$
by(*rule-tac xvec*=*bn* α **and** c=(Ψ, P, α, x, P', C) **in** *name-list-avoiding*) (*auto simp add*: *eqvts*)
obtain *y*::*name* **where** y ♯ Ψ **and** y ♯ P **and** y ♯ $(p \cdot P')$ **and** y ♯ $(p \cdot \alpha)$ **and** y ♯ C
by(*generate-fresh name*) (*auto simp add*: *freshChainSimps simp del*: *action-Fresh*)
from ‹*bn* α ♯∗ *subject* α› ‹*distinctPerm p*› **have** $bn(p \cdot \alpha)$ ♯∗ $subject(p \cdot \alpha)$
by(*subst fresh-star-bij*[*symmetric*, *of - - p*]) (*simp add*: *eqvts*)
from ‹$distinct(bn\ \alpha)$› ‹*distinctPerm p*› **have** $distinct(bn(p \cdot \alpha))$
by(*subst distinctClosed*[*symmetric*, *of - p*]) (*simp add*: *eqvts*)
from ‹x ♯ α› ‹x ♯ $(bn(p \cdot \alpha))$› ‹*distinctPerm p*› *Sp* **have** x ♯ $(p \cdot \alpha)$
by(*subst fresh-bij*[*symmetric*, *of - - p*]) (*simp add*: *eqvts freshChainSimps*)

moreover from ‹$cP = (\!|\nu x|\!)P$› ‹y ♯ P› **have** $cP = (\!|\nu y|\!)([(x, y)] \cdot P)$ **by**(*simp*

add: *alphaRes*)
moreover from ‹$cRs = \alpha \prec (\!|\nu x|\!)P'$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$(bn(p \cdot \alpha)) \sharp* \alpha$› ‹$x \sharp bn(p \cdot \alpha)$› ‹$(bn(p \cdot \alpha)) \sharp* P'$› ‹$x \sharp \alpha$› *Sp*
have $cRs = (p \cdot \alpha) \prec (\!|\nu x|\!)(p \cdot P')$
by(*force simp add*: *residualAlpha*)
with ‹$y \sharp (p \cdot P')$› **have** $cRs = (p \cdot \alpha) \prec (\!|\nu y|\!)([(x, y)] \cdot p \cdot P')$
by(*simp add*: *alphaRes*)
moreover from ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$(bn(p \cdot \alpha)) \sharp* \alpha$› ‹$(bn(p \cdot \alpha)) \sharp* P'$› *Sp*
have $\Psi \rhd P \longmapsto (p \cdot \alpha) \prec (p \cdot P')$ **by**(*force simp add*: *residualAlpha*)
hence$([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \longmapsto [(x, y)] \cdot ((p \cdot \alpha) \prec (p \cdot P'))$
by(*rule eqvts*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› ‹$y \sharp (p \cdot \alpha)$› ‹$x \sharp (p \cdot \alpha)$› *Sp* ‹*distinctPerm p*›
have $\Psi \rhd ([(x, y)] \cdot P) \longmapsto (p \cdot \alpha) \prec ([(x, y)] \cdot p \cdot P')$
by(*simp add*: *eqvts*)
moreover from ‹$bn(p \cdot \alpha) \sharp* P$› ‹$y \sharp (p \cdot \alpha)$› ‹$y \sharp P$› **have** $bn(p \cdot \alpha) \sharp* ([(x, y)] \cdot P)$
by(*auto simp add*: *fresh-star-def fresh-left calc-atm*) (*simp add*: *fresh-def name-list-supp*)
moreover from ‹$distinct(bn\ \alpha)$› **have** $distinct(p \cdot bn\ \alpha)$ **by** *simp*
hence $distinct(bn(p \cdot \alpha))$ **by**(*simp add*: *eqvts*)
ultimately show *?thesis*
using ‹$y \sharp \Psi$› ‹$y \sharp (p \cdot \alpha)$› ‹$y \sharp C$› ‹$bn(p \cdot \alpha) \sharp* \Psi$› ‹$bn(p \cdot \alpha) \sharp* subject(p \cdot \alpha)$› ‹$bn(p \cdot \alpha) \sharp* C$›
by(*rule-tac rScope*)
next
case(*Bang P*)
thus *?thesis* **by**(*rule-tac rBang*)
qed

nominal-primrec
inputLength :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psi* ⇒ *nat*
and *inputLength'* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *input* ⇒ *nat*
and *inputLength''* :: ('*a*::*fs-name*, '*b*::*fs-name*, '*c*::*fs-name*) *psiCase* ⇒ *nat*

where
$inputLength\ (\mathbf{0}) = 0$
$| \ inputLength\ (M\langle N\rangle.P) = 0$
$| \ inputLength\ (M(\!|I) = inputLength'\ I$
$| \ inputLength\ (Case\ C) = 0$
$| \ inputLength\ (P \parallel Q) = 0$
$| \ inputLength\ ((\!|\nu x|\!)P) = 0$
$| \ inputLength\ (\{\!|\Psi|\!\}) = 0$
$| \ inputLength\ (!P) = 0$

$| \ inputLength'\ (Trm\ M\ P) = 0$
$| \ inputLength'\ (\nu\ y\ I) = 1 + (inputLength'\ I)$

$| \ inputLength''\ (\bot_c) = 0$
$| \ inputLength''\ (\Box\Phi \Rightarrow P\ C) = 0$

apply(*finite-guess*)+
apply(*rule TrueI*)+
by(*fresh-guess add*: *fresh-nat*)+

nominal-primrec *boundOutputLength* :: ($'a$, $'b$, $'c$) *boundOutput* $\Rightarrow$ *nat*
where
  boundOutputLength (*BOut M P*) = *0*
| *boundOutputLength* (*BStep x B*) = (*boundOutputLength B*) + *1*
apply(*finite-guess*)+
apply(*rule TrueI*)+
by(*fresh-guess add*: *fresh-nat*)+

nominal-primrec *residualLength* :: ($'a$, $'b$, $'c$) *residual* $\Rightarrow$ *nat*
where
  residualLength (*RIn M N P*) = *0*
| *residualLength* (*ROut M B*) = *boundOutputLength B*
| *residualLength* (*RTau P*) = *0*
by(*rule TrueI*)+

lemma *inputLengthProc*[*simp*]:
  shows *inputLength*($M(\!|\lambda{*}xvec\ N|\!).P$) = *length xvec*
by(*induct xvec*) *auto*

lemma *boundOutputLengthSimp*[*simp*]:
  shows *residualLength*($M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P$) = *length xvec*
by(*induct xvec*) (*auto simp add*: *residualInject*)

lemma *boundOuputLengthSimp2*[*simp*]:
  shows *residualLength*($\alpha \prec P$) = *length*(*bn* α)
by(*nominal-induct* α *rule*: *action.strong-induct*, *auto*) (*auto simp add*: *residualInject*)

lemmas [*simp del*] = *inputLength-inputLength'-inputLength''.simps residualLength.simps boundOutputLength.simps*

lemma *constructPerm*:
  fixes *xvec* :: *name list*
  and *yvec* :: *name list*

  assumes *length xvec* = *length yvec*
  and *xvec* $\sharp{*}$ *yvec*
  and *distinct xvec*
  and *distinct yvec*

  obtains p **where** *set* $p \subseteq$ *set xvec* $\times$ *set*($p \cdot$ *xvec*) **and** *distinctPerm* p **and** *yvec* = $p \cdot$ *xvec*
proof −
  assume $\bigwedge p.$ $[\![$*set* $p \subseteq$ *set xvec* $\times$ *set* ($p \cdot$ *xvec*); *distinctPerm* p; *yvec* = $p \cdot$ *xvec*$]\!]$ $\Longrightarrow$ *thesis*

moreover obtain n **where** $n = length\ xvec$ **by** *auto*
with *assms* **have** $\exists p.\ (set\ p) \subseteq (set\ xvec) \times set\ (yvec) \wedge distinctPerm\ p \wedge\ yvec = p \cdot xvec$
proof(*induct n arbitrary*: *xvec yvec*)
case(*0 xvec yvec*)
thus *?case* **by** *simp*
next
case(*Suc n xvec yvec*)
from ‹$Suc\ n = length\ xvec$›
obtain $x\ xvec'$ **where** $xvec = x\#xvec'$ **and** $length\ xvec' = n$
by(*case-tac xvec*) *auto*
from ‹$length\ xvec = length\ yvec$› ‹$xvec = x\ \#\ xvec'$›
obtain $y\ yvec'$ **where** $length\ xvec' = length\ yvec'$ **and** $yvec = y\#yvec'$
by(*case-tac yvec*) *auto*
from ‹$xvec = x\#xvec'$› ‹$yvec{=}y\#yvec'$› ‹$xvec\ \sharp*\ yvec$›
have $x \neq y$ **and** $xvec'\ \sharp*\ yvec'$ **and** $x\ \sharp\ yvec'$ **and** $y\ \sharp\ xvec'$
by(*auto simp add*: *fresh-list-cons*)
from ‹*distinct xvec*› ‹*distinct yvec*› ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$› **have** $x\ \sharp\ xvec'$ **and** $y\ \sharp\ yvec'$ **and** $distinct\ xvec'$ **and** $distinct\ yvec'$
by *simp+*
from ‹$Suc\ n = length\ xvec$› ‹$xvec{=}x\#xvec'$› **have** $n = length\ xvec'$ **by** *simp*
with ‹$length\ xvec' = length\ yvec'$› ‹$xvec'\ \sharp*\ yvec'$› ‹*distinct xvec*› ‹*distinct yvec*›
obtain p **where** S: $set\ p \subseteq set\ xvec' \times set\ yvec'$ **and** $distinctPerm\ p$ **and** $yvec' = p \cdot xvec'$
by(*drule-tac Suc*) *auto*
from S **have** $set((x,\ y)\#p) \subseteq set(x\#xvec') \times set(y\#yvec')$ **by** *auto*
moreover from ‹$x\ \sharp\ xvec'$› ‹$x\ \sharp\ yvec'$› ‹$y\ \sharp\ xvec'$› ‹$y\ \sharp\ yvec'$› S **have** $x\ \sharp\ p$ **and** $y\ \sharp\ p$
apply(*induct p*)
by(*auto simp add*: *fresh-list-nil fresh-list-cons fresh-prod name-list-supp*) (*auto simp add*: *fresh-def*)

with S ‹$distinctPerm\ p$› ‹$x \neq y$› **have** $distinctPerm((x,\ y)\#p)$ **by** *auto*
moreover from ‹$yvec' = p \cdot xvec'$› ‹$x\ \sharp\ p$› ‹$y\ \sharp\ p$› ‹$x\ \sharp\ xvec'$› ‹$y\ \sharp\ xvec'$› **have** $(y\#yvec') = ((x,\ y)\#p) \cdot (x\#xvec')$
by(*simp add*: *calc-atm freshChainSimps*)
ultimately show *?case* **using** ‹$xvec{=}x\#xvec'$› ‹$yvec{=}y\#yvec'$›
by *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *distinctApend*[*simp*]:
fixes $xvec$ :: *name list*
and $yvec$ :: *name list*

shows $(set\ xvec \cap set\ yvec = \{\}) = xvec\ \sharp*\ yvec$
by(*auto simp add*: *fresh-star-def name-list-supp fresh-def*)

lemma *lengthAux*:
  fixes *xvec* :: *name list*
  and *y* :: *name*
  and *yvec* :: *name list*

  assumes *length xvec* = *length*(*y*#*yvec*)

  obtains *z zvec* **where** *xvec* = *z*#*zvec* **and** *length zvec* = *length yvec*
using *assms*
by(*induct xvec arbitrary*: *yvec y*) *auto*

lemma *lengthAux2*:
  fixes *xvec* :: *name list*
  and *yvec* :: *name list*
  and *zvec* :: *name list*

  assumes *length xvec* = *length*(*yvec*@*y*#*zvec*)

  obtains *xvec1 x xvec2* **where** *xvec*=*xvec1*@*x*#*xvec2* **and** *length xvec1* = *length yvec* **and** *length xvec2* = *length zvec*
proof −
  assume ⋀*xvec1 x xvec2*.
      ⟦*xvec* = *xvec1* @ *x* # *xvec2*; *length xvec1* = *length yvec*;
       length xvec2 = *length zvec*⟧
      ⟹ *thesis*
  moreover from *assms* **have** ∃*xvec1 x xvec2*. *xvec*=*xvec1*@*x*#*xvec2* ∧ *length xvec1* = *length yvec* ∧ *length xvec2* = *length zvec*
    apply(*rule-tac x*=*take* (*length yvec*) *xvec* **in** *exI*)
    apply(*rule-tac x*=*hd*(*drop* (*length yvec*) *xvec*) **in** *exI*)
    apply(*rule-tac x*=*tl*(*drop* (*length yvec*) *xvec*) **in** *exI*)
    by *auto*
  ultimately show *?thesis* **by** *blast*
qed

lemma *semanticsCases*[*consumes 11*, *case-names cInput cCase cPar1 cPar2 cComm1 cComm2 cScope cBang*]:
  fixes Ψ :: *'b*
  and *cP* :: (*'a*, *'b*, *'c*) *psi*
  and *cRs* :: (*'a*, *'b*, *'c*) *residual*
  and *C* :: *'d*::*fs-name*
  and *x1* :: *name*
  and *x2* :: *name*
  and *xvec1* :: *name list*
  and *xvec2* :: *name list*
  and *xvec3* :: *name list*
  and *xvec4* :: *name list*
  and *xvec5* :: *name list*

  assumes Ψ ▷ *cP* ⟼*cRs*

and $length\ xvec1 = inputLength\ cP$ **and** $distinct\ xvec1$
and $length\ xvec2 = residualLength\ cRs$ **and** $distinct\ xvec2$
and $length\ xvec3 = residualLength\ cRs$ **and** $distinct\ xvec3$
and $length\ xvec4 = residualLength\ cRs$ **and** $distinct\ xvec4$
and $length\ xvec5 = residualLength\ cRs$ **and** $distinct\ xvec5$
and $rInput$: $\bigwedge M\ K\ N\ Tvec\ P.\ (\llbracket xvec1 \sharp* \Psi;\ xvec1 \sharp* cP;\ xvec1 \sharp* cRs\rrbracket \Longrightarrow cP = M(\!|\lambda{*}xvec1\ N|\!).P \wedge\ cRs = K(\!|(N[xvec1::=Tvec])|\!) \prec P[xvec1::=Tvec] \wedge \Psi \vdash M \leftrightarrow K \wedge distinct\ xvec1 \wedge set\ xvec1 \subseteq supp\ N \wedge length\ xvec1{=}length\ Tvec \wedge xvec1 \sharp* Tvec \wedge xvec1 \sharp* \Psi \wedge xvec1 \sharp* M \wedge xvec1 \sharp* K) \Longrightarrow Prop$
and $rOutput$: $\bigwedge M\ K\ N\ P.\ \llbracket cP = M\langle N\rangle.P;\ cRs = K\langle N\rangle \prec P;\ \Psi \vdash M \leftrightarrow K\rrbracket \Longrightarrow Prop$
and $rCase$: $\bigwedge Cs\ P\ \varphi.\ \llbracket cP = Cases\ Cs;\ \Psi \rhd P \longmapsto cRs;\ (\varphi,\ P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P\rrbracket \Longrightarrow Prop$
and $rPar1$: $\bigwedge \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q.\ (\llbracket xvec2 \sharp* \Psi;\ xvec2 \sharp* cP;\ xvec2 \sharp* cRs\rrbracket \Longrightarrow cP = P \parallel Q \wedge cRs = \alpha \prec (P' \parallel Q) \wedge xvec2 = bn\ \alpha \wedge \Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P' \wedge extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle \wedge distinct\ A_Q \wedge A_Q \sharp* P \wedge A_Q \sharp* Q \wedge A_Q \sharp* \Psi \wedge A_Q \sharp* \alpha \wedge A_Q \sharp* P' \wedge A_Q \sharp* C) \Longrightarrow Prop$
and $rPar2$: $\bigwedge \Psi_P\ Q\ \alpha\ Q'\ P\ A_P.\ (\llbracket xvec3 \sharp* \Psi;\ xvec3 \sharp* cP;\ xvec3 \sharp* cRs\rrbracket \Longrightarrow cP = P \parallel Q \wedge cRs = \alpha \prec (P \parallel Q') \wedge xvec3 = bn\ \alpha \wedge \Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q' \wedge extractFrame\ P = \langle A_P,\ \Psi_P\rangle \wedge distinct\ A_P \wedge A_P \sharp* P \wedge A_P \sharp* Q \wedge A_P \sharp* \Psi \wedge A_P \sharp* \alpha \wedge A_P \sharp* Q' \wedge A_P \sharp* C) \Longrightarrow Prop$
and $rComm1$: $\bigwedge \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q.$
$\llbracket cP = P \parallel Q;\ cRs = \tau \prec (\!|\nu{*}xvec|\!)P' \parallel Q';$
$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ P = \langle A_P,\ \Psi_P\rangle;$
$distinct\ A_P;$
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';\ extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle;\ distinct\ A_Q;$
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;\ A_P \sharp* \Psi;\ A_P \sharp* \Psi_Q;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* N;$
$A_P \sharp* P';\ A_P \sharp* Q;\ A_P \sharp* Q';\ A_P \sharp* A_Q;\ A_P \sharp* xvec;\ A_Q \sharp* \Psi;\ A_Q \sharp* \Psi_P;$
$A_Q \sharp* P;\ A_Q \sharp* K;\ A_Q \sharp* N;\ A_Q \sharp* P';\ A_Q \sharp* Q;\ A_Q \sharp* Q';\ A_Q \sharp* xvec;$
$xvec \sharp* \Psi;\ xvec \sharp* \Psi_P;\ xvec \sharp* \Psi_Q;\ xvec \sharp* P;\ xvec \sharp* M;\ xvec \sharp* Q;$
$xvec \sharp* K;\ A_P \sharp* C;\ A_Q \sharp* C;\ xvec \sharp* C;\ distinct\ xvec\rrbracket \Longrightarrow Prop$
and $rComm2$: $\bigwedge \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q.$
$\llbracket cP = P \parallel Q;\ cRs = \tau \prec (\!|\nu{*}xvec|\!)P' \parallel Q';$
$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ extractFrame\ P = \langle A_P,$

$\Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; $A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$;
$A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp*$ *xvec*; $A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$;
$A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* Q'$; $A_Q \sharp*$ *xvec*;
xvec $\sharp* \Psi$; *xvec* $\sharp* \Psi_P$; *xvec* $\sharp* \Psi_Q$; *xvec* $\sharp* P$; *xvec* $\sharp* M$; *xvec* $\sharp* Q$;
xvec $\sharp* K$; $A_P \sharp* C$; $A_Q \sharp* C$; *xvec* $\sharp* C$; *distinct xvec*$\rrbracket \Longrightarrow$ *Prop*
and *rOpen*: $\bigwedge P$ *M xvec y yvec N P'*.
($\llbracket$*xvec4* $\sharp* \Psi$; *xvec4* $\sharp*$ *cP*; *xvec4* $\sharp*$ *cRs*; *x1* $\sharp \Psi$; *x1* $\sharp$ *cP*; *x1* $\sharp$ *cRs*; *x1* $\sharp$ *xvec4*$\rrbracket \Longrightarrow$
$cP = (\!|\nu x1|\!)P \wedge cRs = M(\!|\nu*(xvec@x1\#yvec)|\!)\langle N\rangle \prec P' \wedge$ *xvec4*=*xvec@y#yvec* $\wedge$
$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P' \wedge x1 \in$ *supp N* $\wedge$ *x1* $\sharp$ *xvec* $\wedge$ *x1* $\sharp$ *yvec* $\wedge$
distinct xvec $\wedge$ *distinct yvec* $\wedge$ *xvec* $\sharp* \Psi \wedge$ *xvec* $\sharp* P \wedge$ *xvec* $\sharp* M$ $\wedge$ *xvec* $\sharp*$ *yvec* $\wedge$
yvec $\sharp* \Psi \wedge$ *yvec* $\sharp* P \wedge$ *yvec* $\sharp* M) \Longrightarrow$ *Prop*
and *rScope*: $\bigwedge P\ \alpha\ P'$. ($\llbracket$*xvec5* $\sharp* \Psi$; *xvec5* $\sharp*$ *cP*; *xvec5* $\sharp*$ *cRs*; *x2* $\sharp \Psi$; *x2* $\sharp$ *cP*; *x2* $\sharp$ *cRs*; *x2* $\sharp$ *xvec5*$\rrbracket \Longrightarrow$
$cP = (\!|\nu x2|\!)P \wedge cRs = \alpha \prec (\!|\nu x2|\!)P' \wedge$ *xvec5* $= bn\ \alpha \wedge$
$\Psi \rhd P \longmapsto \alpha \prec P' \wedge x2 \sharp \Psi \wedge x2 \sharp \alpha \wedge bn\ \alpha \sharp*$ *subject* α $\wedge$ *distinct*(*bn* α)) $\Longrightarrow$ *Prop*
and *rBang*: $\bigwedge P$. $\llbracket cP = !P$; $\Psi \rhd P \parallel !P \longmapsto cRs$; *guarded P*$\rrbracket \Longrightarrow$ *Prop*
shows *Prop*
using ‹$\Psi \rhd cP \longmapsto cRs$›
proof(*cases rule*: *semanticsCasesAux*[**where** *C*=(*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)])
case(*cInput M K xvec N Tvec P*)
have *B*: $cP = M(\!|\lambda*xvec\ N|\!).P$ **and** *C*: $cRs = K(\!|(N[xvec::=Tvec])|\!) \prec (P[xvec::=Tvec])$

by *fact+*
from ‹*xvec* $\sharp*$ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** *xvec* $\sharp*$ *xvec1* **by** *simp*

from ‹*length xvec1* = *inputLength cP*› *B* **have** *length xvec1* = *length xvec*
by *simp*
then obtain *p* **where** *S*: *set p* $\subseteq$ *set xvec* $\times$ *set*(*p* $\cdot$ *xvec*) **and** *distinctPerm p* **and** *xvec1* = *p* $\cdot$ *xvec*
using ‹*xvec* $\sharp*$ *xvec1*› ‹*distinct xvec*› ‹*distinct xvec1*›
by(*rule-tac constructPerm*[**where** *xvec*=*xvec* **and** *yvec*=*xvec1*]) *auto*
show *?thesis*
proof(*rule rInput*[**where** *M*=*M* **and** *K*=*K* **and** *N* = *p* $\cdot$ *N* **and** *Tvec*=*Tvec* **and** *P*=*p* $\cdot$ *P*], *goal-cases*)
case *1*

from B ‹$xvec \sharp* xvec1$› ‹$xvec1 \sharp* cP$› **have** $xvec1 \sharp* N$ **and** $xvec1 \sharp* P$
by(*auto simp add*: *fresh-star-def inputChainFresh name-list-supp*) (*auto simp add*: *fresh-def*)

from ‹$cP = M(\!|\lambda*xvec\ N|\!).P$› S ‹$xvec1 \sharp* N$› ‹$xvec1 \sharp* P$› ‹$xvec1 = p \cdot xvec$›
have $cP = M(\!|\lambda*xvec1\ (p \cdot N)|\!).(p \cdot P)$
apply *simp*
by(*subst inputChainAlpha*) *auto*
moreover from ‹$cRs = K(\!|(N[xvec::=Tvec])|\!) \prec P[xvec::=Tvec]$› S ‹$xvec1 \sharp* N$› ‹$xvec1 \sharp* P$› ‹$xvec1 = p \cdot xvec$› ‹$length\ xvec = length\ Tvec$› ‹$distinctPerm\ p$›
have $cRs = K(\!|((p \cdot N)[xvec1::=Tvec])|\!) \prec (p \cdot P)[xvec1::=Tvec]$
by(*simp add*: *renaming substTerm.renaming*)
moreover note ‹$\Psi \vdash M \leftrightarrow K$›
moreover from ‹$distinct\ xvec$› ‹$xvec1 = p \cdot xvec$› **have** $distinct\ xvec1$ **by** *simp*
moreover from ‹$set\ xvec \subseteq supp\ N$› **have** $(p \cdot set\ xvec) \subseteq (p \cdot (supp\ N))$
by(*simp add*: *eqvts*)
with ‹$xvec1 = p \cdot xvec$› **have** $set\ xvec1 \subseteq supp(p \cdot N)$ **by**(*simp add*: *eqvts*)
moreover from ‹$length\ xvec = length\ Tvec$› ‹$xvec1 = p \cdot xvec$› **have** $length\ xvec1 = length\ Tvec$
by *simp*

moreover from ‹$xvec1 \sharp* cRs$› C ‹$length\ xvec = length\ Tvec$› ‹$distinct\ xvec$› ‹$set\ xvec \subseteq supp\ N$›
have $(set\ xvec1) \sharp* Tvec$
by(*rule-tac substTerm.subst3Chain*[**where** T=N]) *auto*
hence $xvec1 \sharp* Tvec$ **by** *simp*
moreover from ‹$xvec \sharp* Tvec$› **have** $(p \cdot xvec) \sharp* (p \cdot Tvec)$ **by**(*simp add*: *fresh-star-bij*)
with S ‹$xvec \sharp* Tvec$› ‹$xvec1 \sharp* Tvec$› ‹$xvec1 = p \cdot xvec$› **have** $xvec1 \sharp* Tvec$ **by** *simp*
moreover from ‹$xvec \sharp* M$› **have** $(p \cdot xvec) \sharp* (p \cdot M)$ **by**(*simp add*: *fresh-star-bij*)
with S ‹$xvec \sharp* M$› ‹$xvec1 \sharp* cP$› B ‹$xvec1 = p \cdot xvec$› **have** $xvec1 \sharp* M$ **by** *simp*
moreover from ‹$xvec \sharp* K$› **have** $(p \cdot xvec) \sharp* (p \cdot K)$ **by**(*simp add*: *fresh-star-bij*)
with S ‹$xvec \sharp* K$› ‹$xvec1 \sharp* cRs$› C ‹$xvec1 = p \cdot xvec$› **have** $xvec1 \sharp* K$ **by** *simp*
ultimately show *?case* **using** ‹$xvec1 \sharp* \Psi$› **by** *blast*
qed
next
case(*cOutput M K N P*)
thus *?thesis* **by**(*rule rOutput*)
next
case(*cCase Cs P* φ)
thus *?thesis* **by**(*rule rCase*)
next
case($cPar1\ \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q$)
have B: $cP = P \parallel Q$ **and** C: $cRs = \alpha \prec P' \parallel Q$

by *fact+*
from ‹*bn* α $\sharp*$ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** *bn* α $\sharp*$ *xvec2* **by** *simp*
from ‹A_Q $\sharp*$ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** A_Q $\sharp*$ *xvec2* **and** A_Q $\sharp*$ *C* **by** *simp+*

from ‹*length xvec2* = *residualLength cRs*› *C* **have** *length xvec2* = *length*(*bn* α)
by *simp*
then obtain *p* **where** *S*: *set p* $\subseteq$ *set*(*bn* α) $\times$ *set*(*bn*($p \cdot \alpha$)) **and** *distinctPerm p* **and** *xvec2*= *bn*($p \cdot \alpha$)
using ‹*bn* α $\sharp*$ *xvec2*› ‹*distinct*(*bn* α)› ‹*distinct xvec2*›
by(*rule-tac constructPerm*[**where** *xvec*=*bn* α **and** *yvec*=*xvec2*]) (*auto simp add*: *eqvts*)
show *?thesis*
proof(*rule rPar1*[**where** *P*=*P* **and** *Q*=*Q* **and** α=$p \cdot \alpha$ **and** P'=$p \cdot P'$ **and** A_Q=A_Q **and** Ψ_Q=Ψ_Q], *goal-cases*)
case *1*
note ‹*cP* = *P* $\parallel$ *Q*›
moreover from *B C S* ‹*bn* α $\sharp*$ *xvec2*› ‹*xvec2* $\sharp*$ *cRs*› ‹*xvec2* = *bn*($p \cdot \alpha$)› ‹*bn* α $\sharp*$ *subject* α› ‹*xvec2* $\sharp*$ *cP*› ‹*bn* α $\sharp*$ *Q*›
have *cRs* = ($p \cdot \alpha$) $\prec$ ($p \cdot P'$) $\parallel$ *Q*
apply *auto*
by(*subst residualAlpha*[**where** *p*=*p*]) *auto*
moreover note ‹*xvec2* = *bn*($p \cdot \alpha$)›
moreover from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$› *S B C S* ‹*bn* α $\sharp*$ *xvec2*› ‹*xvec2* $\sharp*$ *cRs*› ‹*xvec2* = *bn*($p \cdot \alpha$)› ‹*bn* α $\sharp*$ *subject* α› ‹*xvec2* $\sharp*$ *cP*›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto (p \cdot \alpha) \prec (p \cdot P')$
by(*subst residualAlpha*[*symmetric*]) *auto*
moreover note ‹*extractFrame Q* = $\langle A_Q, \Psi_Q \rangle$› ‹*distinct* A_Q› ‹A_Q $\sharp*$ *P*› ‹A_Q $\sharp*$ *Q*› ‹A_Q $\sharp*$ Ψ› ‹A_Q $\sharp*$ α›
moreover from ‹A_Q $\sharp*$ α› ‹A_Q $\sharp*$ *xvec2*› *S* ‹*xvec2* = *bn*($p \cdot \alpha$)› ‹*distinctPerm p*› **have** A_Q $\sharp*$ ($p \cdot \alpha$)
by(*subst fresh-star-bij*[*symmetric*, **where** *pi*=*p*]) *simp*
moreover from ‹A_Q $\sharp*$ P'› ‹A_Q $\sharp*$ α› ‹A_Q $\sharp*$ *xvec2*› *S* ‹*xvec2* = *bn*($p \cdot \alpha$)› ‹*distinctPerm p*› **have** A_Q $\sharp*$ ($p \cdot P'$)
by(*subst fresh-star-bij*[*symmetric*, **where** *pi*=*p*]) *simp*
moreover note ‹A_Q $\sharp*$ *C*›
ultimately show *?case* **by** *blast*
qed
next
case(*cPar2* Ψ_P *Q* α Q' *P* A_P)
have *B*: *cP* = *P* $\parallel$ *Q* **and** *C*: *cRs* = $\alpha \prec P \parallel Q'$
by *fact+*
from ‹*bn* α $\sharp*$ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** *bn* α $\sharp*$ *xvec3* **by** *simp*
from ‹A_P $\sharp*$ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** A_P $\sharp*$ *xvec3* **and** A_P $\sharp*$ *C* **by** *simp+*

from ‹*length xvec3* = *residualLength cRs*› *C* **have** *length xvec3* = *length*(*bn* α)

by *simp*
then obtain p **where** S: $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** *distinctPerm* p **and** $xvec3 = bn(p \cdot \alpha)$
using ‹$bn\ \alpha \sharp* xvec3$› ‹$distinct(bn\ \alpha)$› ‹$distinct\ xvec3$›
by(*rule-tac constructPerm*[**where** $xvec{=}bn\ \alpha$ **and** $yvec{=}xvec3$]) (*auto simp add*: *eqvts*)
show *?thesis*
proof(*rule rPar2*[**where** $P{=}P$ **and** $Q{=}Q$ **and** $\alpha{=}p \cdot \alpha$ **and** $Q'{=}p \cdot Q'$ **and** $A_P{=}A_P$ **and** $\Psi_P{=}\Psi_P$], *goal-cases*)
case *1*
note ‹$cP = P \parallel Q$›
moreover from $B\ C\ S$ ‹$bn\ \alpha \sharp* xvec3$› ‹$xvec3 \sharp* cRs$› ‹$xvec3 = bn(p \cdot \alpha)$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$xvec3 \sharp* cP$› ‹$bn\ \alpha \sharp* P$›
have $cRs = (p \cdot \alpha) \prec P \parallel (p \cdot Q')$
apply *auto*
by(*subst residualAlpha*[**where** $p{=}p$]) *auto*
moreover note ‹$xvec3 = bn(p \cdot \alpha)$›
moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto\alpha \prec Q'$› $S\ B\ C\ S$ ‹$bn\ \alpha \sharp* xvec3$› ‹$xvec3 \sharp* cRs$› ‹$xvec3 = bn(p \cdot \alpha)$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$xvec3 \sharp* cP$›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto(p \cdot \alpha) \prec (p \cdot Q')$
by(*subst residualAlpha*[*symmetric*]) *auto*
moreover note ‹$extractFrame\ P = \langle A_P, \Psi_P\rangle$› ‹$distinct\ A_P$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \alpha$›
moreover from ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* xvec3$› S ‹$xvec3 = bn(p \cdot \alpha)$› ‹*distinctPerm* p› **have** $A_P \sharp* (p \cdot \alpha)$
by(*subst fresh-star-bij*[*symmetric*, **where** $pi{=}p$]) *simp*
moreover from ‹$A_P \sharp* Q'$› ‹$A_P \sharp* \alpha$› ‹$A_P \sharp* xvec3$› S ‹$xvec3 = bn(p \cdot \alpha)$› ‹*distinctPerm* p› **have** $A_P \sharp* (p \cdot Q')$
by(*subst fresh-star-bij*[*symmetric*, **where** $pi{=}p$]) *simp*
moreover note ‹$A_P \sharp* C$›
ultimately show *?case* **by** *blast*
qed
next
case($cComm1\ \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q$)
thus *?thesis* **by**(*rule-tac rComm1*[**where** $P{=}P$ **and** $Q{=}Q$]) (*assumption* | *simp*)+
next
case($cComm2\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q$)
thus *?thesis* **by**(*rule-tac rComm2*[**where** $P{=}P$ **and** $Q{=}Q$]) (*assumption* | *simp*)+
next
case($cOpen\ P\ M\ xvec\ yvec\ N\ P'\ x$)
have B: $cP = (\!|\nu x|\!)P$ **and** C: $cRs = M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$
by *fact+*
from ‹$xvec \sharp* (xvec1, xvec2, xvec3, xvec4, xvec5, x1, x2, cP, cRs, C)$› **have** $xvec \sharp* xvec4$ **and** $xvec \sharp* cP$ **and** $xvec \sharp* cRs$ **and** $x1 \sharp xvec$ **by** *simp+*
from ‹$x \sharp (xvec1, xvec2, xvec3, xvec4, xvec5, x1, x2, cP, cRs, C)$› **have** $x \sharp xvec4$ **and** $x \sharp cP$ **and** $x \sharp cRs$ **and** $x \neq x1$ **by** *simp+*
from ‹$yvec \sharp* (xvec1, xvec2, xvec3, xvec4, xvec5, x1, x2, cP, cRs, C)$› **have** $yvec \sharp* xvec4$ **and** $yvec \sharp* cP$ **and** $yvec \sharp* cRs$ **and** $x1 \sharp yvec$ **by** *simp+*

from ‹$xvec$ $\sharp*$ cRs› ‹x $\sharp$ cRs› ‹$yvec$ $\sharp*$ cRs› C **have** $(xvec@x\#yvec)$ $\sharp*$ M **by** *simp*
from ‹$xvec$ $\sharp*$ Ψ› ‹x $\sharp$ Ψ› ‹$yvec$ $\sharp*$ Ψ› **have** $(xvec@x\#yvec)$ $\sharp*$ Ψ **by** *simp*
from ‹$length\ xvec4 = residualLength\ cRs$› C **obtain** $xvec'$ y $yvec'$ **where** D: $xvec4 = xvec'@y\#yvec'$ **and** $length\ xvec' = length\ xvec$ **and** $length\ yvec' = length\ yvec$
by(*rule-tac lengthAux2*) *auto*
with ‹$distinct\ xvec$› ‹$distinct\ yvec$› ‹x $\sharp$ $xvec$› ‹x $\sharp$ $yvec$› ‹$xvec$ $\sharp*$ $yvec$› ‹$xvec$ $\sharp*$ $xvec4$› ‹$yvec$ $\sharp*$ $xvec4$› ‹x $\sharp$ $xvec4$› ‹$distinct\ xvec4$›
have $distinct\ xvec'$ **and** $distinct\ yvec'$ **and** $xvec'$ $\sharp*$ $yvec'$ **and** $x \neq y$ **and** y $\sharp$ $xvec'$ **and** y $\sharp$ $yvec'$
and x $\sharp$ $xvec'$ **and** x $\sharp$ $yvec'$ **and** y $\sharp$ $xvec$ **and** y $\sharp$ $yvec$ **and** $xvec$ $\sharp*$ $xvec'$ **and** $yvec$ $\sharp*$ $yvec'$
by *auto*
from ‹$length\ xvec' = length\ xvec$› ‹$xvec$ $\sharp*$ $xvec'$› ‹$distinct\ xvec$› ‹$distinct\ xvec'$›
obtain p **where** Sp: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$ **and** $distinctPerm\ p$ **and** E: $xvec' = p \cdot xvec$
by(*metis constructPerm*)
from ‹$length\ yvec' = length\ yvec$› ‹$yvec$ $\sharp*$ $yvec'$› ‹$distinct\ yvec$› ‹$distinct\ yvec'$›
obtain q **where** Sq: $set\ q \subseteq set\ yvec \times set(q \cdot yvec)$ **and** $distinctPerm\ q$ **and** F: $yvec' = q \cdot yvec$
by(*metis constructPerm*)

show *?thesis*
proof(*rule rOpen*[**where** P=$([(x,\ x1)] \cdot P)$ **and** $xvec$=$p \cdot xvec$ **and** y=y **and** $yvec$=$q \cdot yvec$ **and** N=$(p@(x1,\ x)\#q) \cdot N$ **and** P'=$(p@(x1,\ x)\#q) \cdot P'$ **and** M=M], *goal-cases*)
case *1*
from ‹$xvec$ $\sharp*$ $xvec4$› ‹x $\sharp$ $xvec4$› ‹$x1$ $\sharp$ $xvec4$› ‹$yvec$ $\sharp*$ $xvec4$› D E F
have $x \neq y$ **and** $x1 \neq y$ **and** $x1$ $\sharp$ $p \cdot xvec$ **and** $x1$ $\sharp$ $q \cdot yvec$ **by** *simp+*
from ‹$xvec4$ $\sharp*$ cRs› ‹$x1$ $\sharp$ cRs› C **have** $xvec4$ $\sharp*$ M **and** $x1$ $\sharp$ M **by** *simp+*
from ‹$cP = (\!|\nu x|\!)P$› ‹x $\sharp$ cP› ‹$x \neq x1$› **have** $([(x,\ x1)] \cdot cP) = [(x,\ x1)] \cdot (\!|\nu x|\!)P$
by *simp*
with ‹x $\sharp$ cP› ‹$x1$ $\sharp$ cP› **have** $cP = (\!|\nu x1|\!)([(x,\ x1)] \cdot P)$ **by**(*simp add*: *eqvts calc-atm*)
moreover from C **have** $((p@(x1,\ x)\#q) \cdot cRs) = (p@(x1,\ x)\#q) \cdot (M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P')$ **by**(*simp add*: *fresh-star-bij*)
with Sp Sq ‹$xvec4$ $\sharp*$ cRs› D E F ‹$xvec$ $\sharp*$ cRs› ‹x $\sharp$ cRs› ‹$yvec$ $\sharp*$ cRs› ‹$xvec4$ $\sharp*$ M› ‹$(xvec@x\#yvec)$ $\sharp*$ M› ‹$xvec$ $\sharp*$ $xvec4$› ‹x $\sharp$ $xvec4$› ‹$yvec$ $\sharp*$ $xvec4$› ‹$xvec$ $\sharp*$ $yvec$› ‹x $\sharp$ $xvec$› ‹x $\sharp$ $yvec$› ‹y $\sharp$ $xvec'$› ‹y $\sharp$ $yvec'$› ‹$xvec'$ $\sharp*$ $yvec'$› ‹$x1$ $\sharp$ $xvec$› ‹$x1$ $\sharp$ $yvec$› ‹$x1 \neq y$› ‹$x1$ $\sharp$ $xvec4$› ‹$x1$ $\sharp$ cRs› ‹$x1$ $\sharp$ cRs› ‹$x \neq x1$› ‹$x1$ $\sharp$ M›
have $cRs = M(\!|\nu*((p \cdot xvec)@x1\#(q \cdot yvec))|\!)\langle((p@(x1,\ x)\#q) \cdot N)\rangle \prec ((p@(x1,\ x)\#q) \cdot P')$
by(*simp add*: *eqvts pt2*[*OF pt-name-inst*] *calc-atm*)
moreover from D E F **have** $xvec4 = (p \cdot xvec)@y\#(q \cdot yvec)$ **by** *simp*
moreover from ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$› **have** $((p@(x1,\ x)\#q) \cdot \Psi) \rhd ((p@(x1,\ x)\#q) \cdot P) \longmapsto ((p@(x1,\ x)\#q) \cdot (M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'))$

by(*intro eqvts*)
with *Sp Sq B C D E F* ‹*xvec4* ♯∗ Ψ› ‹(*xvec*@*x*#*yvec*) ♯∗ Ψ› ‹*xvec4* ♯∗ *cRs*› ‹*x* ♯ *xvec4*› *C D* ‹*x* ♯ *cRs*› ‹*yvec* ♯∗ *cRs*› ‹*xvec4* ♯∗ *M*› ‹(*xvec*@*x*#*yvec*) ♯∗ *M*› ‹*x* ♯ *M*› ‹*x1* ♯ *cRs*› ‹*x* ≠ *x1*› ‹*x1* ♯ *xvec*› ‹*x1* ♯ *yvec*› ‹*xvec* ♯∗ *xvec4*› ‹*yvec* ♯∗ *xvec4*› ‹*x1* ♯ *xvec4*› ‹*x* ♯ *xvec*› ‹*x* ♯ *yvec*› ‹*x1* ♯ Ψ› ‹*xvec4* ♯∗ *cP*› ‹*xvec* ♯∗ *P*› ‹*yvec* ♯∗ *P*› ‹*xvec′* ♯∗ *yvec′*› ‹*x1* ♯ *xvec4*› ‹*xvec4* ♯∗ *cP*› ‹*yvec* ♯∗ *xvec4*› ‹*xvec* ♯∗ *xvec4*› ‹*x* ≠ *x1*› ‹*xvec* ♯∗ *yvec*›
have Ψ ⊳ ([(*x*, *x1*)] · *P*) ⟼*M*⦇*ν*∗((*p* · *xvec*)@(*q* · *yvec*))⦈⟨((*p*@(*x1*, *x*)#*q*) · *N*)⟩ ≺ ((*p*@(*x1*, *x*)#*q*) · *P′*)
by(*simp add*: *eqvts pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*] *pt2*[*OF pt-name-inst*] *name-swap*)

moreover from ‹*x* ∈ *supp N*› **have** ((*p*@(*x1*, *x*)#*q*) · *x*) ∈ ((*p*@(*x1*, *x*)#*q*) · *supp N*)
by(*simp add*: *pt-set-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence *x1* ∈ *supp*((*p*@(*x1*, *x*)#*q*) · *N*)
using ‹*x* ♯ *xvec*› ‹*x* ♯ *yvec*› ‹*x1* ♯ *xvec*› ‹*x1* ♯ *yvec*› ‹*x* ♯ *xvec4*› ‹*x1* ♯ *xvec4*› ‹*xvec* ♯∗ *xvec4*› ‹*yvec* ♯∗ *xvec4*› ‹*xvec′* ♯∗ *yvec′*› *D E F Sp Sq* ‹*x* ≠ *x1*›
by(*simp add*: *eqvts pt2*[*OF pt-name-inst*] *calc-atm*)
moreover from ‹*x1* ♯ *xvec4*› *D E F* **have** *x1* ♯ (*p* · *xvec*) **and** *x1* ♯ (*q* · *yvec*) **by** *simp*+
moreover from ‹*distinct xvec′*› ‹*distinct yvec′*› *E F* **have** *distinct*(*p* · *xvec*) **and** *distinct*(*q* · *yvec*) **by** *simp*+
moreover from ‹*xvec′* ♯∗ *yvec′*› *E F* **have** (*p* · *xvec*) ♯∗ (*q* · *yvec*) **by** *auto*
moreover from ‹*xvec* ♯∗ Ψ› **have** (*p* · *xvec*) ♯∗ (*p* · Ψ) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with *Sp D E* ‹*xvec4* ♯∗ Ψ› ‹*xvec* ♯∗ Ψ› **have** (*p* · *xvec*) ♯∗ Ψ **by**(*simp add*: *eqvts*)
moreover from ‹*yvec* ♯∗ Ψ› **have** (*p* · *yvec*) ♯∗ (*p* · Ψ) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with *Sq D F* ‹*xvec4* ♯∗ Ψ› ‹*yvec* ♯∗ Ψ› **have** (*q* · *yvec*) ♯∗ Ψ **by**(*simp add*: *eqvts*)
moreover from ‹*xvec4* ♯∗ *cP*› ‹*x* ♯ *xvec4*› ‹*x1* ♯ *xvec4*› *B D E F* **have** (*p* · *xvec*) ♯∗ ([(*x*, *x1*)] · *P*) **and** (*q* · *yvec*) ♯∗ ([(*x*, *x1*)] · *P*)
by *simp*+
moreover from ‹*xvec4* ♯∗ *M*› *C D E F* **have** (*p* · *xvec*) ♯∗ *M* **and** (*q* · *yvec*) ♯∗ *M* **by** *simp*+
ultimately show *?case*
by *blast*
qed
next
case(*cScope P* α *P′ x*)
have *B*: *cP* = ⦇*ν*x⦈*P* **and** *C*: *cRs* = α ≺ ⦇*ν*x⦈*P′*
by *fact*+
from ‹*bn* α ♯∗ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** *bn* α ♯∗ *xvec5* **and** *x2* ♯ *bn* α **by** *simp*+
from ‹*x* ♯ (*xvec1*, *xvec2*, *xvec3*, *xvec4*, *xvec5*, *x1*, *x2*, *cP*, *cRs*, *C*)› **have** *x* ♯ *xvec5* **and** *x* ≠ *x2* **and** *x* ♯ *cRs* **by** *simp*+

from ‹*length xvec5* = *residualLength cRs*› *C* **have** *length xvec5* = *length*(*bn* α)
by *simp*
then obtain *p* **where** *S*: *set p* $\subseteq$ *set*(*bn* α) $\times$ *set*(*bn*(*p* $\cdot$ α)) **and** *distinctPerm p* **and** *xvec5*= *bn*(*p* $\cdot$ α)
using ‹*bn* α $\sharp*$ *xvec5*› ‹*distinct*(*bn* α)› ‹*distinct xvec5*›
by(*rule-tac constructPerm*[**where** *xvec*=*bn* α **and** *yvec*=*xvec5*]) (*auto simp add*: *eqvts*)
show *?thesis*
proof(*rule rScope*[**where** *P*=[(*x*, *x2*)] $\cdot$ *P* **and** α=[(*x*, *x2*)] $\cdot$ *p* $\cdot$ α **and** *P'*=[(*x*, *x2*)] $\cdot$ *p* $\cdot$ *P'*], *goal-cases*)
case *1*
from ‹*x2* $\sharp$ *cRs*› *C* ‹*x2* $\sharp$ *bn* α› ‹*x* $\neq$ *x2*› **have** *x2* $\sharp$ α **and** *x2* $\sharp$ *P'* **by**(*auto simp add*: *abs-fresh*)
from ‹*cP* = ⦇ν*x*⦈*P*› ‹*x2* $\sharp$ *cP*› ‹*x* $\neq$ *x2*› **have** *cP* = ⦇ν*x2*⦈([(*x*, *x2*)] $\cdot$ *P*)
by(*simp add*: *alphaRes abs-fresh*)
moreover from *B C S* ‹*bn* α $\sharp*$ *xvec5*› ‹*xvec5* $\sharp*$ *cRs*› ‹*xvec5* = *bn*(*p* $\cdot$ α)› ‹*bn* α $\sharp*$ *subject* α› ‹*xvec5* $\sharp*$ *cP*› ‹*x* $\sharp$ α› ‹*x* $\sharp$ *xvec5*›
have *cRs* = (*p* $\cdot$ α) $\prec$ ⦇ν*x*⦈(*p* $\cdot$ *P'*)
apply *auto*
by(*subst residualAlpha*[**where** *p*=*p*] *alphaRes*) (*auto simp del*: *actionFresh*)
hence ([(*x*, *x2*)] $\cdot$ *cRs*) = [(*x*, *x2*)] $\cdot$ ((*p* $\cdot$ α) $\prec$ ⦇ν*x*⦈(*p* $\cdot$ *P'*))
by *simp*
with ‹*x2* $\sharp$ *cRs*› ‹*x* $\sharp$ *cRs*› **have** *cRs* = ([(*x*, *x2*)] $\cdot$ *p* $\cdot$ α) $\prec$ ⦇ν*x2*⦈([(*x*, *x2*)] $\cdot$ *p* $\cdot$ *P'*)
by(*simp add*: *eqvts calc-atm*)
moreover from ‹*xvec5*= *bn*(*p* $\cdot$ α)› **have** ([(*x*, *x2*)] $\cdot$ *xvec5*) = ([(*x*, *x2*)] $\cdot$ *bn*(*p* $\cdot$ α))
by *simp*
with ‹*x* $\sharp$ *xvec5*› ‹*x2* $\sharp$ *xvec5*› **have** *xvec5* = *bn*([(*x*, *x2*)] $\cdot$ *p* $\cdot$ α)
by(*simp add*: *eqvts*)
moreover from ‹Ψ $\rhd$ *P* $\longmapsto$$\alpha$ $\prec$ *P'*› *S B C S* ‹*bn* α $\sharp*$ *xvec5*› ‹*xvec5* $\sharp*$ *cRs*› ‹*xvec5* = *bn*(*p* $\cdot$ α)› ‹*bn* α $\sharp*$ *subject* α› ‹*xvec5* $\sharp*$ *cP*› ‹*x* $\sharp$ *xvec5*›
have Ψ $\rhd$ *P* $\longmapsto$(*p* $\cdot$ α) $\prec$ (*p* $\cdot$ *P'*)
by(*subst residualAlpha*[*symmetric*]) *auto*
hence ([(*x*, *x2*)] $\cdot$ Ψ) $\rhd$ ([(*x*, *x2*)] $\cdot$ *P*) $\longmapsto$([(*x*, *x2*)] $\cdot$ ((*p* $\cdot$ α) $\prec$ (*p* $\cdot$ *P'*)))
by(*rule eqvt*)
with ‹*x* $\sharp$ Ψ› ‹*x2* $\sharp$ Ψ› **have** Ψ $\rhd$ ([(*x*, *x2*)] $\cdot$ *P*) $\longmapsto$([(*x*, *x2*)] $\cdot$ *p* $\cdot$ α) $\prec$ ([(*x*, *x2*)] $\cdot$ *p* $\cdot$ *P'*)
by(*simp add*: *eqvts*)
moreover note ‹*x2* $\sharp$ Ψ›
moreover from ‹*x* $\sharp$ α› ‹*x2* $\sharp$ α› ‹*x* $\sharp$ *xvec5*› ‹*x2* $\sharp$ *xvec5*› *S* ‹*x* $\neq$ *x2*› ‹*xvec5* = *bn*(*p* $\cdot$ α)› **have** *x2* $\sharp$ [(*x*, *x2*)] $\cdot$ *p* $\cdot$ α
apply(*subgoal-tac x* $\sharp$ *p* $\wedge$ *x2* $\sharp$ *p*)
apply(*simp add*: *perm-compose freshChainSimps del*: *actionFresh*)
by(*auto dest*: *freshAlphaSwap*)
moreover from ‹*bn* α $\sharp*$ *subject* α› **have** ([(*x*, *x2*)] $\cdot$ *p* $\cdot$ (*bn* α)) $\sharp*$ ([(*x*, *x2*)] $\cdot$ *p* $\cdot$ (*subject* α))
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence *bn*([(*x*, *x2*)] $\cdot$ *p* $\cdot$ α) $\sharp*$ *subject*([(*x*, *x2*)] $\cdot$ *p* $\cdot$ α)

by(*simp add*: *eqvts*)
moreover from ‹*distinct*(*bn* α)› **have** *distinct*([(*x*, *x2*)] · *p* · (*bn* α)) **by** *simp*
hence *distinct*(*bn*([(*x*, *x2*)] · *p* · α)) **by**(*simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
qed
next
case(*cBang P*)
thus *?thesis* **by**(*rule-tac rBang*) *auto*
qed

lemma *parCases*[*consumes 5*, *case-names cPar1 cPar2 cComm1 cComm2*]:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a action*
and *T* :: (*'a*, *'b*, *'c*) *psi*
and *C* :: *'d*::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto \alpha \prec T$
and *bn* α $\sharp*$ Ψ
and *bn* α $\sharp*$ *P*
and *bn* α $\sharp*$ *Q*
and *bn* α $\sharp*$ *subject* α
and *rPar1*: $\bigwedge P'\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P';$ *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$A_Q \sharp* \Psi$; $A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* \alpha$; $A_Q \sharp* P'$; $A_Q \sharp* C]\!] \Longrightarrow$ *Prop* α $(P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* \alpha$; $A_P \sharp* Q'$; $A_P \sharp* C]\!] \Longrightarrow$ *Prop* α $(P \parallel Q')$
and *rComm1*: $\bigwedge \Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K\ xvec\ Q'\ A_Q.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';$ *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C]\!] \Longrightarrow$
Prop (τ) $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
and *rComm2*: $\bigwedge \Psi_Q\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ K\ Q'\ A_Q.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q';$ *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; A_P

$\sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C]\!] \Longrightarrow$
$Prop\ (\tau)\ ((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

shows *Prop* α T
proof −
from *Trans* **have** *distinct*(*bn* α) **by**(*auto dest*: *boundOutputDistinct*)
have *length*(*bn* α) = *residualLength*($\alpha \prec T$) **by** *simp*
note *Trans*
moreover have *length* [] = *inputLength*($P \parallel Q$) **and** *distinct* []
by(*auto simp add*: *inputLength-inputLength′-inputLength″.simps*)
moreover note ‹*length*(*bn* α) = *residualLength*($\alpha \prec T$)› ‹*distinct*(*bn* α)›
moreover note ‹*length*(*bn* α) = *residualLength*($\alpha \prec T$)› ‹*distinct*(*bn* α)›
moreover note ‹*length*(*bn* α) = *residualLength*($\alpha \prec T$)› ‹*distinct*(*bn* α)›
moreover note ‹*length*(*bn* α) = *residualLength*($\alpha \prec T$)› ‹*distinct*(*bn* α)›
moreover obtain *x*::*name* **where** $x \sharp \Psi$ **and** $x \sharp P$ **and** $x \sharp Q$ **and** $x \sharp \alpha$ **and** $x \sharp T$
by(*generate-fresh name*) *auto*
ultimately show *?thesis* **using** ‹*bn* $\alpha \sharp* \Psi$› ‹*bn* $\alpha \sharp* P$› ‹*bn* $\alpha \sharp* Q$› ‹*bn* $\alpha \sharp*$ *subject* α›
apply(*cases rule*: *semanticsCases*[*of* - - - - - - - - - *C x x*])
apply(*auto simp add*: *psi.inject*)
apply(*force simp add*: *residualInject residualInject′ intro*: *rPar1*)
apply(*force simp add*: *residualInject residualInject′ intro*: *rPar2*)
apply(*fastforce simp add*: *residualInject residualInject′ intro*: *rComm1*)
by(*fastforce simp add*: *residualInject residualInject′ intro*: *rComm2*)
qed

lemma *parInputCases*[*consumes 1*, *case-names cPar1 cPar2*]:
fixes Ψ :: *′b*
and P :: (*′a*, *′b*, *′c*) *psi*
and Q :: (*′a*, *′b*, *′c*) *psi*
and M :: *′a*
and N :: *′a*
and R :: (*′a*, *′b*, *′c*) *psi*
and C :: *′d*::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto M(\!|N|\!) \prec R$
and *rPar1*: $\bigwedge P'\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$A_Q \sharp* \Psi;\ A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* M;\ A_Q \sharp* N;\ A_Q \sharp* C]\!] \Longrightarrow Prop\ (P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* M;\ A_P \sharp* N;\ A_P \sharp* C]\!] \Longrightarrow Prop\ (P \parallel Q')$

shows *Prop R*
proof −
from *Trans* **obtain** α **where** $\Psi \rhd P \parallel Q \longmapsto \alpha \prec R$ **and** $bn\ \alpha \sharp* \Psi$ **and** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp*$ *subject* α **and** $\alpha = M(\!|N|\!)$ **by** *auto*
thus *?thesis* **using** *rPar1 rPar2*
by(*induct rule*: *parCases*) (*auto simp add*: *residualInject*)
qed

lemma *parOutputCases*[*consumes 5*, *case-names cPar1 cPar2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and R :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec R$
and $xvec \sharp* \Psi$
and $xvec \sharp* P$
and $xvec \sharp* Q$
and $xvec \sharp* M$
and *rPar1*: $\bigwedge P'\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$A_Q \sharp* \Psi;\ A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* M;\ A_Q \sharp* xvec;\ A_Q \sharp* N;\ A_Q \sharp* C;\ A_Q \sharp* xvec;\ distinct\ xvec]\!] \Longrightarrow Prop\ (P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* M;\ A_P \sharp* xvec;\ A_P \sharp* N;\ A_P \sharp* C;\ A_P \sharp* xvec;\ distinct\ xvec]\!] \Longrightarrow Prop\ (P \parallel Q')$
shows *Prop R*
proof −
from *Trans* **have** *distinct xvec* **by**(*auto dest*: *boundOutputDistinct*)
obtain α **where** $\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$ **by** *simp*
with *Trans* ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$xvec \sharp* Q$› ‹$xvec \sharp* M$›
have $\Psi \rhd P \parallel Q \longmapsto \alpha \prec R$ **and** $bn\ \alpha \sharp* \Psi$ **and** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ $bn\ \alpha \sharp*$ *subject* α
by *simp*+
thus *?thesis* **using** ‹$\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$› *rPar1 rPar2* ‹*distinct xvec*›
by(*induct rule*: *parCases*[**where** $C = (xvec, C)$]) (*auto simp add*: *residualInject*)
qed

lemma *theEqvt*[*eqvt-force*]:
fixes p :: *name prm*
and α :: $'a$ *action*

assumes $\alpha \neq \tau$

shows $(p \cdot the(subject\ \alpha)) = the(p \cdot (subject\ \alpha))$
using *assms*
by(*induct rule*: *actionCases*[**where** α=α]) *auto*

lemma *theSubjectFresh*[*simp*]:
fixes α :: *'a action*
and x :: *name*

assumes $\alpha \neq \tau$

shows $x \sharp the(subject\ \alpha) = x \sharp subject\ \alpha$
using *assms*
by(*cases rule*: *actionCases*) *auto*

lemma *theSubjectFreshChain*[*simp*]:
fixes α :: *'a action*
and *xvec* :: *name list*

assumes $\alpha \neq \tau$

shows $xvec \sharp* the(subject\ \alpha) = xvec \sharp* subject\ \alpha$
using *assms*
by(*cases rule*: *actionCases*) *auto*

lemma *obtainPrefix*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a action*
and P' :: (*'a*, *'b*, *'c*) *psi*
and A_P :: *name list*
and Ψ_P :: *'b*
and B :: *name list*

assumes $\Psi \rhd P \longmapsto \alpha \prec P'$
and $extractFrame\ P = \langle A_P, \Psi_P \rangle$
and $distinct\ A_P$
and $bn\ \alpha \sharp* subject\ \alpha$
and $distinct(bn\ \alpha)$
and $\alpha \neq \tau$
and $B \sharp* P$
and $A_P \sharp* \Psi$
and $A_P \sharp* B$
and $A_P \sharp* P$
and $A_P \sharp* subject\ \alpha$

obtains M **where** $\Psi \otimes \Psi_P \vdash the(subject\ \alpha) \leftrightarrow M$ **and** $B \sharp* M$
using *assms*
proof(*nominal-induct avoiding*: *B arbitrary*: *thesis rule*: *semanticsFrameInduct'*)
case(*cAlpha* Ψ P α P' p A_P Ψ_P B)

then obtain M **where** *subjEq*: $\Psi \otimes \Psi_P \vdash$ *the*(*subject* α) $\leftrightarrow M$ **and** $B \sharp* M$
by(*rule-tac cAlpha*) *auto*
from ‹*set* $p \subseteq$ *set*(*bn* α) $\times$ *set*(*bn*($p \cdot \alpha$))› ‹*bn* $\alpha \sharp*$ *subject* α› ‹*bn*($p \cdot \alpha$) $\sharp* \alpha$› *subjEq*
have $\Psi \otimes \Psi_P \vdash$ *the*(*subject*($p \cdot \alpha$)) $\leftrightarrow M$
by(*simp add*: *subjectEqvt*[*symmetric*])
thus *?case* **using** *cAlpha* ‹$B \sharp* M$›
by *auto*
next
case(*cFrameAlpha* Ψ P A_P Ψ_P p α P' B)
then obtain M **where** *subjEq*: $\Psi \otimes \Psi_P \vdash$ *the*(*subject* α) $\leftrightarrow M$ **and** $B \sharp* M$
by(*rule-tac cFrameAlpha*) *auto*
have S: *set* $p \subseteq$ *set* $A_P \times$ *set* ($p \cdot A_P$) **by** *fact*
from *subjEq* **have** ($p \cdot (\Psi \otimes \Psi_P)$) $\vdash$ ($p \cdot$ *the*(*subject* α)) $\leftrightarrow$ ($p \cdot M$)
by(*rule chanEqClosed*)
with ‹$A_P \sharp* \Psi$› ‹($p \cdot A_P$) $\sharp* \Psi$› ‹($p \cdot A_P$) $\sharp*$ *subject* α› S ‹$\alpha \neq \tau$› ‹$A_P \sharp* \alpha$›
have $\Psi \otimes (p \cdot \Psi_P) \vdash$ *the*(*subject* α) $\leftrightarrow$ ($p \cdot M$)
by(*simp add*: *eqvts del*: *subjectEqvt*)
moreover from ‹$B \sharp* M$› **have** ($p \cdot B$) $\sharp*$ ($p \cdot M$)
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* B$› ‹($p \cdot A_P$) $\sharp* B$› S **have** $B \sharp*$ ($p \cdot M$) **by**(*simp add*: *eqvts*)
ultimately show *?case* **by**(*rule cFrameAlpha*)
next
case(*cInput* Ψ M K *xvec* N *Tvec* P B)
from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt AssertionStatEqSym*[*OF Identity*])
hence $\Psi \otimes \mathbf{1} \vdash K \leftrightarrow M$ **by**(*rule chanEqSym*)
moreover from ‹$B \sharp*$ ($M(\!|\lambda*xvec\ N|\!).P$)› **have** $B \sharp* M$ **by** *simp*
ultimately show *?case* **by**(*rule-tac cInput*) *auto*
next
case(*cOutput* Ψ M K N P B)
from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt AssertionStatEqSym*[*OF Identity*])
hence $\Psi \otimes \mathbf{1} \vdash K \leftrightarrow M$
by(*rule chanEqSym*)
moreover from ‹$B \sharp*$ ($M\langle N\rangle.P$)› **have** $B \sharp* M$ **by** *simp*
ultimately show *?case* **by**(*rule-tac cOutput*) *auto*
next
case(*cCase* Ψ P α P' φ *Cs* A_P Ψ_P B)
then obtain M **where** $\Psi \otimes \Psi_P \vdash$ *the*(*subject* α) $\leftrightarrow M$ **and** $B \sharp* M$
by(*rule-tac cCase*) (*auto dest*: *memFreshChain*)
with ‹$\Psi_P \simeq \mathbf{1}$› **show** *?case* **by**(*blast intro*: *cCase statEqEnt compositionSym Identity*)
next
case(*cPar1* Ψ Ψ_Q P α P' A_Q Q A_P Ψ_P B)
then obtain M **where** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash$ *the*(*subject* α) $\leftrightarrow M$ **and** $B \sharp* M$
apply(*rule-tac cPar1*) **by** *assumption auto*
thus *?case*
by(*metis cPar1 statEqEnt Associativity Commutativity AssertionStatEqTrans*

Composition)
next
  case(*cPar2* Ψ Ψ_P Q α Q' A_P P A_Q Ψ_Q B)
  then obtain M **where** $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash$ *the*(*subject* α) $\leftrightarrow M$ **and** $B \sharp* M$
    by(*rule-tac cPar2*) *auto*
  thus *?case* **by**(*metis cPar2 statEqEnt Associativity*)
next
  case *cComm1*
  thus *?case* **by** *simp*
next
  case *cComm2*
  thus *?case* **by** *simp*
next
  case(*cOpen* Ψ P M *xvec yvec* N P' x A_P Ψ_P B)
  then obtain K **where** $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$ **and** $B \sharp* K$
    apply(*rule-tac cOpen*) **by** *force auto*
  thus *?case* **by**(*fastforce intro*: *cOpen*)
next
  case(*cScope* Ψ P α P' x A_P Ψ_P B)
  then obtain M **where** $\Psi \otimes \Psi_P \vdash$ *the*(*subject* α) $\leftrightarrow M$ **and** $B \sharp* M$
    by(*rule-tac cScope*) *auto*
  thus *?case* **by**(*fastforce intro*: *cScope*)
next
  case(*cBang* Ψ P α P' A_P Ψ_P B)
  then obtain K **where** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash$ *the*(*subject* α) $\leftrightarrow K$ **and** $B \sharp* K$
    by(*rule-tac cBang*) *auto*
  with ‹$\Psi_P \simeq \mathbf{1}$› **show** *?case* **by**(*metis cBang statEqEnt compositionSym Identity*)
qed

lemma *inputRenameSubject*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$
  and P' :: $('a, 'b, 'c)$ *psi*
  and A_P :: *name list*
  and Ψ_P :: $'b$

  assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
  and *extractFrame* $P = \langle A_P, \Psi_P \rangle$
  and *distinct* A_P
  and $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$
  and $A_P \sharp* \Psi$
  and $A_P \sharp* P$
  and $A_P \sharp* M$
  and $A_P \sharp* K$

  shows $\Psi \rhd P \longmapsto K(\!|N|\!) \prec P'$
using *assms*

proof(*nominal-induct avoiding*: *K rule*: *inputFrameInduct*)
case(*cAlpha* Ψ P M N P' A_P Ψ_P p K)
have S: *set* $p \subseteq$ *set* $A_P \times$ *set* $(p \cdot A_P)$ **by** *fact*
from ‹$\Psi \otimes (p \cdot \Psi_P) \vdash M \leftrightarrow K$› **have** $(p \cdot (\Psi \otimes (p \cdot \Psi_P))) \vdash (p \cdot M) \leftrightarrow (p \cdot K)$
by(*rule chanEqClosed*)
with S ‹*distinctPerm p*› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K$› ‹$(p \cdot A_P) \sharp* \Psi$› ‹$(p \cdot A_P) \sharp* M$› ‹$(p \cdot A_P) \sharp* K$›
have $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$ **by**(*simp add*: *eqvts*)
with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K$›
‹$[\![\Psi \otimes \Psi_P \vdash M \leftrightarrow K;\ A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* K]\!] \Longrightarrow \Psi \rhd P \longmapsto K(\!|N|\!) \prec P'$›
show *?case* **by** *blast*
next
case(*cInput* Ψ M K *xvec* N *Tvec* P K')
from ‹$\Psi \otimes \mathbf{1} \vdash K \leftrightarrow K'$› **have** $\Psi \vdash K \leftrightarrow K'$
by(*blast intro*: *statEqEnt Identity*)
with ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \vdash M \leftrightarrow K'$
by(*rule chanEqTrans*)
thus *?case* **using** ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp* N› ‹*length xvec* = *length Tvec*›
by(*rule Input*)
next
case(*cCase* Ψ P M N P' φ *Cs* A_P Ψ_P K)
from ‹$\Psi \otimes \mathbf{1} \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity compositionSym AssertionStatEqSym*)
with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K$›
‹$\bigwedge K.\ [\![\Psi \otimes \Psi_P \vdash M \leftrightarrow K;\ A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* K]\!] \Longrightarrow \Psi \rhd P \longmapsto K(\!|N|\!) \prec P'$›
have $\Psi \rhd P \longmapsto K(\!|N|\!) \prec P'$ **by** *force*
thus *?case* **using** ‹(φ, P) *mem Cs*› ‹$\Psi \vdash \varphi$› ‹*guarded* P› **by**(*rule Case*)
next
case(*cPar1* Ψ Ψ_Q P M N P' A_Q Q A_P Ψ_P K)
from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Composition AssertionStatEqTrans Commutativity*)
with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K$›
‹$\bigwedge K.\ [\![(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K;\ A_P \sharp* (\Psi \otimes \Psi_Q);\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* K]\!] \Longrightarrow \Psi \otimes \Psi_Q \rhd P \longmapsto K(\!|N|\!) \prec P'$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto K(\!|N|\!) \prec P'$ **by** *force*
thus *?case* **using** ‹*extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* N$›
by(*rule-tac Par1*) *auto*
next
case(*cPar2* Ψ Ψ_P Q M N Q' A_P P A_Q Ψ_Q K)
from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash M \leftrightarrow K$
by(*rule statEqEnt*[*OF AssertionStatEqSym*[*OF Associativity*]])
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* M$› ‹$A_Q \sharp* K$›
‹$\bigwedge K.\ [\![(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash M \leftrightarrow K;\ A_Q \sharp* (\Psi \otimes \Psi_P);\ A_Q \sharp* Q;\ A_Q \sharp* M;\ A_Q \sharp* K]\!] \Longrightarrow \Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$›

have $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$ **by** *force*
thus *?case* **using** ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* K$› ‹$A_P \sharp* N$›
by(*rule-tac Par2*) *auto*
next
case(*cScope* Ψ P M N P' x A_P Ψ_P)
hence $\Psi \rhd P \longmapsto K(\!|N|\!) \prec P'$ **by** *force*
with ‹$x \sharp \Psi$› ‹$x \sharp K$› ‹$x \sharp N$› **show** *?case*
by(*rule-tac Scope*) *auto*
next
case(*cBang* Ψ P M N P' A_P Ψ_P K)
from ‹$\Psi \otimes \mathbf{1} \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity compositionSym AssertionStatEqSym*)
with ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K$›
‹$\bigwedge K.\ [\![\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K;\ A_P \sharp* \Psi;\ A_P \sharp* (P \parallel !P);\ A_P \sharp* M;\ A_P \sharp* K]\!] \Longrightarrow \Psi \rhd P \parallel !P \longmapsto K(\!|N|\!) \prec P'$›
have $\Psi \rhd P \parallel !P \longmapsto K(\!|N|\!) \prec P'$ **by** *force*
thus *?case* **using** ‹*guarded* P› **by**(*rule Bang*)
qed

lemma *outputRenameSubject*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$

assumes $\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$
and $A_P \sharp* \Psi$
and $A_P \sharp* P$
and $A_P \sharp* M$
and $A_P \sharp* K$

shows $\Psi \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
using *assms*
apply(*simp add*: *residualInject*)
proof(*nominal-induct avoiding*: K *rule*: *outputFrameInduct*)
case(*cAlpha* Ψ P M A_P Ψ_P p B K)
have S: *set* $p \subseteq$ *set* $A_P \times$ *set*$(p \cdot A_P)$ **by** *fact*
from ‹$\Psi \otimes (p \cdot \Psi_P) \vdash M \leftrightarrow K$› **have** $(p \cdot (\Psi \otimes (p \cdot \Psi_P))) \vdash (p \cdot M) \leftrightarrow (p \cdot K)$
by(*rule chanEqClosed*)
with S ‹*distinctPerm* p› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K$› ‹$(p \cdot A_P) \sharp* \Psi$› ‹$(p \cdot$

A_P) $\sharp*$ M› ‹(p · A_P) $\sharp*$ K›
have $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$ **by**(*simp add*: *eqvts*)
with ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ K›
show *?case* **by**(*blast intro*: *cAlpha*)
next
case(*cOutput* Ψ M K N P K')
from ‹$\Psi \otimes \mathbf{1} \vdash K \leftrightarrow K'$› **have** $\Psi \vdash K \leftrightarrow K'$
by(*blast intro*: *statEqEnt Identity*)
with ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \vdash M \leftrightarrow K'$
by(*rule chanEqTrans*)
thus *?case* **using** *Output* **by**(*force simp add*: *residualInject*)
next
case(*cCase* Ψ P M B φ Cs A_P Ψ_P K)
from ‹$\Psi \otimes \mathbf{1} \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity compositionSym AssertionStatEqSym*)
with ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ K›
‹$\bigwedge K.$ $[\![\Psi \otimes \Psi_P \vdash M \leftrightarrow K;\ A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* K]\!] \Longrightarrow \Psi \rhd P \longmapsto (ROut\ K\ B)$›
have $\Psi \rhd P \longmapsto ROut\ K\ B$ **by** *force*
thus *?case* **using** ‹(φ, P) *mem Cs*› ‹$\Psi \vdash \varphi$› ‹*guarded* P› **by**(*rule Case*)
next
case(*cPar1* Ψ Ψ_Q P M *xvec* N P' A_Q Q A_P Ψ_P K)
from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Composition AssertionStatEqTrans Commutativity*)
with ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ_Q› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ K›
‹$\bigwedge K.$ $[\![(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K;\ A_P \sharp* (\Psi \otimes \Psi_Q);\ A_P \sharp* P;\ A_P \sharp* M;\ A_P \sharp* K]\!] \Longrightarrow \Psi \otimes \Psi_Q \rhd P \longmapsto (ROut\ K\ ((\!|\nu*xvec|\!)N \prec' P'))$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **by**(*force simp add*: *residualInject*)
thus *?case* **using** ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹*xvec* $\sharp*$ Q› ‹A_Q $\sharp*$ Ψ› ‹A_Q $\sharp*$ P› ‹A_Q $\sharp*$ K› ‹A_Q $\sharp*$ *xvec*› ‹A_Q $\sharp*$ N› *Par1*[**where** α=$K(\!|\nu*xvec|\!)\langle N\rangle$]
by(*auto simp add*: *residualInject*)
next
case(*cPar2* Ψ Ψ_P Q M *xvec* N Q' A_P P A_Q Ψ_Q K)
from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash M \leftrightarrow K$
by(*rule statEqEnt*[*OF AssertionStatEqSym*[*OF Associativity*]])
with ‹A_Q $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ_P› ‹A_Q $\sharp*$ Q› ‹A_Q $\sharp*$ M› ‹A_Q $\sharp*$ K›
‹$\bigwedge K.$ $[\![(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash M \leftrightarrow K;\ A_Q \sharp* (\Psi \otimes \Psi_P);\ A_Q \sharp* Q;\ A_Q \sharp* M;\ A_Q \sharp* K]\!] \Longrightarrow \Psi \otimes \Psi_P \rhd Q \longmapsto ROut\ K\ ((\!|\nu*xvec|\!)N \prec' Q')$›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto ROut\ K\ ((\!|\nu*xvec|\!)N \prec' Q')$ **by** *force*
thus *?case* **using** ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹*xvec* $\sharp*$ P› ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ Q› ‹A_P $\sharp*$ K› ‹A_P $\sharp*$ *xvec*› ‹A_P $\sharp*$ N› *Par2*[**where** α=$K(\!|\nu*xvec|\!)\langle N\rangle$]
by(*auto simp add*: *residualInject*)
next
case(*cOpen* Ψ P M *xvec* *yvec* N P' x A_P Ψ_P)
hence $\Psi \rhd P \longmapsto K(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$ **by**(*force simp add*: *residualInject*)
with ‹$x \in supp\ N$› ‹$x \sharp \Psi$› ‹$x \sharp K$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› *Open* **show** *?case*
by(*auto simp add*: *residualInject*)
next

case(*cScope* Ψ P M *xvec* N P' x A_P Ψ_P)
hence $\Psi \rhd P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by**(*force simp add*: *residualInject*)
with ‹$x \sharp \Psi$› ‹$x \sharp K$› ‹$x \sharp xvec$› ‹$x \sharp N$› *Scope*[**where** $\alpha{=}K(\!|\nu{*}xvec|\!)\langle N\rangle$] **show** *?case*
by(*auto simp add*: *residualInject*)
next
case(*cBang* Ψ P M B A_P Ψ_P K)
from ‹$\Psi \otimes \mathbf{1} \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity compositionSym AssertionStatEqSym*)
with ‹$A_P \sharp{*} \Psi$› ‹$A_P \sharp{*} P$› ‹$A_P \sharp{*} M$› ‹$A_P \sharp{*} K$›
‹$\bigwedge K.\ [\![\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K;\ A_P \sharp{*} \Psi;\ A_P \sharp{*} (P \parallel !P);\ A_P \sharp{*} M;\ A_P \sharp{*} K]\!] \Longrightarrow \Psi \rhd P \parallel !P \longmapsto ROut\ K\ B$›
have $\Psi \rhd P \parallel !P \longmapsto ROut\ K\ B$ **by** *force*
thus *?case* **using** ‹*guarded P*› **by**(*rule Bang*)
qed

lemma *parCasesSubject*[*consumes 7*, *case-names cPar1 cPar2 cComm1 cComm2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and R :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*
and *yvec* :: *name list*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto \alpha \prec R$
and $bn\ \alpha \sharp{*} \Psi$
and $bn\ \alpha \sharp{*} P$
and $bn\ \alpha \sharp{*} Q$
and $bn\ \alpha \sharp{*}$ *subject* α
and *yvec* $\sharp{*}$ P
and *yvec* $\sharp{*}$ Q
and *rPar1*: $\bigwedge P'\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P';$ *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$A_Q \sharp{*} \Psi;\ A_Q \sharp{*} P;\ A_Q \sharp{*} \alpha;\ A_Q \sharp{*} C]\!] \Longrightarrow$ *Prop* α $(P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$A_P \sharp{*} \Psi;\ A_P \sharp{*} Q;\ A_P \sharp{*} \alpha;\ A_P \sharp{*} C]\!] \Longrightarrow$ *Prop* α $(P \parallel Q')$
and *rComm1*: $\bigwedge \Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K$ *xvec* $Q'\ A_Q$.
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';$ *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$ *yvec* $\sharp{*}$ M; *yvec* $\sharp{*}$ K; *distinct xvec*;
$A_P \sharp{*} \Psi;\ A_P \sharp{*} \Psi_Q;\ A_P \sharp{*} P;\ A_P \sharp{*} M;\ A_P \sharp{*} N;\ A_P \sharp{*} P';\ A_P \sharp{*} Q;\ A_P \sharp{*}$ *xvec*; $A_P \sharp{*} Q';\ A_P \sharp{*} A_Q;\ A_P \sharp{*} C;$
$A_Q \sharp{*} \Psi;\ A_Q \sharp{*} \Psi_P;\ A_Q \sharp{*} P;\ A_Q \sharp{*} K;\ A_Q \sharp{*} N;\ A_Q \sharp{*} P';\ A_Q \sharp{*} Q;\ A_Q \sharp{*}$ *xvec*; $A_Q \sharp{*} Q';\ A_Q \sharp{*} C;$
xvec $\sharp{*}$ Ψ; *xvec* $\sharp{*}$ Ψ_P; *xvec* $\sharp{*}$ P; *xvec* $\sharp{*}$ M; *xvec* $\sharp{*}$ K; *xvec* $\sharp{*}$ Q; *xvec* $\sharp{*}$ Ψ_Q; *xvec* $\sharp{*}$ $C]\!] \Longrightarrow$

Prop (τ) $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
and *rComm2*: $\bigwedge \Psi_Q\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ K\ Q'\ A_Q.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; $yvec \sharp* M$; $yvec \sharp* K$; *distinct* $xvec$;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C]\!] \Longrightarrow$
Prop (τ) $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

shows *Prop* α R
using *Trans* ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp*$ *subject* α›
proof(*induct rule*: *parCases*[**where** C=(C, *yvec*)])
case(*cPar1* $P'\ A_Q\ \Psi_Q$)
thus *?case* **by**(*rule-tac rPar1*) *auto*
next
case(*cPar2* $Q'\ A_P\ \Psi_P$)
thus *?case* **by**(*rule-tac rPar2*) *auto*
next
case(*cComm1* $\Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K\ xvec\ Q'\ A_Q$)
from ‹$A_P \sharp* (C, yvec)$› ‹$A_Q \sharp* (C, yvec)$› ‹$xvec \sharp* (C, yvec)$›
have $A_P \sharp* C$ **and** $A_Q \sharp* C$ **and** $xvec \sharp* C$ **and** $A_P \sharp* yvec$ **and** $A_Q \sharp* yvec$ **and** $xvec \sharp* yvec$
by *simp*+

have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
and *MeqK*: $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$ **by** *fact*+

from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› *FrP* ‹*distinct* A_P› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$yvec \sharp* P$› ‹$A_P \sharp* \Psi$›
‹$A_P \sharp* A_Q$› ‹$A_P \sharp* yvec$› ‹$A_P \sharp* xvec$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$xvec \sharp* P$› ‹$A_P \sharp* \Psi_Q$›
obtain M' **where** *MeqM'*: $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $xvec \sharp* M'$ **and** $yvec \sharp* M'$ **and** $A_Q \sharp* M'$
by(*rule-tac* B=$xvec@yvec@A_Q$ **in** *obtainPrefix*) (*assumption* | *force*)+
from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› *FrQ* ‹*distinct* A_Q› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$yvec \sharp* Q$› ‹$A_Q \sharp* \Psi$›
‹$A_P \sharp* A_Q$› ‹$A_Q \sharp* yvec$› ‹$A_Q \sharp* xvec$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$xvec \sharp* Q$› ‹$A_Q \sharp* \Psi_P$› ‹$xvec \sharp* K$› ‹*distinct xvec*›
obtain K' **where** *KeqK'*: $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow K'$ **and** $xvec \sharp* K'$ **and** $yvec \sharp* K'$ **and** $A_P \sharp* K'$
by(*rule-tac* B=$xvec@yvec@A_P$ **in** *obtainPrefix*) (*assumption* | *force* | *metis freshChainSym*)+

from *MeqK KeqK'* **have** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K'$

by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans*)
with ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› *FrP* ‹*distinct* A_P›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto K'(\!|N|\!) \prec P'$ **using** ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K'$›
by(*rule-tac inputRenameSubject*) (*assumption* | *force*)+
moreover note *FrP* ‹*distinct* A_P›
moreover from *MeqK MeqM'* **have** $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans chanEqSym*)
with ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› *FrQ* ‹*distinct* A_Q›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
by(*rule-tac outputRenameSubject*) (*assumption* | *force*)+
moreover note *FrQ* ‹*distinct* A_Q›
moreover from *MeqM' KeqK' MeqK* **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K' \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans chanEqSym*)
moreover note ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* K'$› ‹$A_P \sharp* N$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* xvec$› ‹$A_P \sharp* Q'$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* C$›
‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* M'$› ‹$A_Q \sharp* N$› ‹$A_Q \sharp* Q'$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* xvec$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* C$›
‹$xvec \sharp* \Psi$› ‹$xvec \sharp* \Psi_P$› ‹$xvec \sharp* P$› ‹$xvec \sharp* M'$› ‹$xvec \sharp* K'$› ‹$xvec \sharp* Q$› ‹$xvec \sharp* \Psi_Q$› ‹$xvec \sharp* C$› ‹$yvec \sharp* M'$› ‹$yvec \sharp* K'$› ‹*distinct xvec*›
ultimately show *?case*
by(*rule-tac rComm1*)
next
case(*cComm2* Ψ_Q *M xvec N* P' A_P Ψ_P *K* Q' A_Q)
from ‹$A_P \sharp* (C, yvec)$› ‹$A_Q \sharp* (C, yvec)$› ‹$xvec \sharp* (C, yvec)$›
have $A_P \sharp* C$ **and** $A_Q \sharp* C$ **and** $xvec \sharp* C$ **and** $A_P \sharp* yvec$ **and** $A_Q \sharp* yvec$ **and** $xvec \sharp* yvec$
by *simp*+

have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
and *MeqK*: $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$ **by** *fact*+

from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› *FrP* ‹*distinct* A_P› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$yvec \sharp* P$› ‹$A_P \sharp* \Psi$›
‹$A_P \sharp* A_Q$› ‹$A_P \sharp* yvec$› ‹$A_P \sharp* xvec$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$xvec \sharp* P$› ‹$A_P \sharp* \Psi_Q$› ‹$xvec \sharp* M$› ‹*distinct xvec*›
obtain M' **where** *MeqM'*: $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $xvec \sharp* M'$ **and** $yvec \sharp* M'$ **and** $A_Q \sharp* M'$
by(*rule-tac* B=$xvec@yvec@A_Q$ **in** *obtainPrefix*) (*assumption* | *force*)+
from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› *FrQ* ‹*distinct* A_Q› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$yvec \sharp* Q$› ‹$A_Q \sharp* \Psi$›
‹$A_P \sharp* A_Q$› ‹$A_Q \sharp* yvec$› ‹$A_Q \sharp* xvec$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$xvec \sharp* Q$› ‹$A_Q \sharp* \Psi_P$›
obtain K' **where** *KeqK'*: $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow K'$ **and** $xvec \sharp* K'$ **and** $yvec \sharp* K'$ **and** $A_P \sharp* K'$
by(*rule-tac* B=$xvec@yvec@A_P$ **in** *obtainPrefix*) (*assumption* | *force* | *metis*

freshChainSym)+

from *MeqK KeqK′* **have** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K'$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans*)
with ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› *FrP* ‹*distinct* A_P›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto K'(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **using** ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K'$›
by(*rule-tac outputRenameSubject*) (*assumption* | *force*)+
moreover note *FrP* ‹*distinct* A_P›
moreover from *MeqK MeqM′* **have** $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans chanEqSym*)
with ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› *FrQ* ‹*distinct* A_Q›
have $\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|N|\!) \prec Q'$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
by(*rule-tac inputRenameSubject*) (*assumption* | *force*)+
moreover note *FrQ* ‹*distinct* A_Q›
moreover from *MeqM′ KeqK′ MeqK* **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K' \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans chanEqSym*)
moreover note ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* K'$› ‹$A_P \sharp* N$› ‹$A_P \sharp* P'$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* xvec$› ‹$A_P \sharp* Q'$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* C$›
‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* M'$› ‹$A_Q \sharp* N$› ‹$A_Q \sharp* Q'$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* xvec$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* C$›
‹$xvec \sharp* \Psi$› ‹$xvec \sharp* \Psi_P$› ‹$xvec \sharp* P$› ‹$xvec \sharp* M'$› ‹$xvec \sharp* K'$› ‹$xvec \sharp* Q$› ‹$xvec \sharp* \Psi_Q$› ‹$xvec \sharp* C$› ‹$yvec \sharp* M'$› ‹$yvec \sharp* K'$› ‹*distinct xvec*›
ultimately show *?case*
by(*rule-tac rComm2*)
qed

lemma *inputCases*[*consumes 1*, *case-names cInput*]:
fixes Ψ :: *′b*
and M :: *′a*
and *xvec* :: *name list*
and N :: *′a*
and P :: (*′a*, *′b*, *′c*) *psi*
and α :: *′a action*
and P' :: (*′a*, *′b*, *′c*) *psi*

assumes *Trans*: $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto \alpha \prec P'$
and *rInput*: $\bigwedge K\ Tvec.\ [\![\Psi \vdash M \leftrightarrow K;\ set\ xvec \subseteq supp\ N;\ length\ xvec = length\ Tvec;\ distinct\ xvec]\!] \Longrightarrow Prop\ (K(\!|N[xvec::=Tvec]|\!))\ (P[xvec::=Tvec])$

shows *Prop* α P'
proof −
{
fix *xvec N P*
assume *Trans*: $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto \alpha \prec P'$
and $xvec \sharp* \Psi$ **and** $xvec \sharp* M$ **and** $xvec \sharp* \alpha$ **and** $xvec \sharp* P'$ **and** *distinct*

xvec
and *rInput*: $\bigwedge K\ Tvec.\ [\![\Psi \vdash M \leftrightarrow K;\ set\ xvec \subseteq supp\ N;\ length\ xvec = length\ Tvec;\ distinct\ xvec]\!] \Longrightarrow Prop\ (K(\!|N[xvec::=Tvec]|\!))\ (P[xvec::=Tvec])$

from *Trans* **have** $bn\ \alpha = []$
apply −
by(*ind-cases* $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto\alpha \prec P'$) (*auto simp add*: *residualInject*)
from *Trans* **have** $distinct(bn\ \alpha)$ **by**(*auto dest*: *boundOutputDistinct*)
have $length(bn\ \alpha) = residualLength(\alpha \prec P')$ **by** *simp*
note *Trans*
moreover have $length\ xvec = inputLength(M(\!|\lambda{*}xvec\ N|\!).P)$ **by** *auto*
moreover note ‹*distinct xvec*›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover obtain $x::name$ **where** $x \sharp \Psi$ **and** $x \sharp P$ **and** $x \sharp M$ **and** $x \sharp xvec$ **and** $x \sharp \alpha$ **and** $x \sharp P'$ **and** $x \sharp N$
by(*generate-fresh name*) *auto*
ultimately have $Prop\ \alpha\ P'$ **using** ‹$bn\ \alpha = []$› ‹$xvec \sharp{*}\ \Psi$›‹$xvec \sharp{*}\ M$› ‹$xvec \sharp{*}\ \alpha$› ‹$xvec \sharp{*}\ P'$›
apply(*cases rule*: *semanticsCases*[*of* - - - - - - - - - *C x*])
apply(*force simp add*: *residualInject psi.inject rInput*)
by(*fastforce simp add*: *residualInject psi.inject inputChainFresh*)+
}
note *Goal* = *this*
moreover obtain p :: *name prm* **where** $(p \cdot xvec) \sharp{*}\ \Psi$ **and** $(p \cdot xvec) \sharp{*}\ M$ **and** $(p \cdot xvec) \sharp{*}\ N$ **and** $(p \cdot xvec) \sharp{*}\ P$
and $(p \cdot xvec) \sharp{*}\ \alpha$ **and** $(p \cdot xvec) \sharp{*}\ P'$ **and** S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$
and *distinctPerm p*
by(*rule-tac xvec=xvec* **and** $c=(\Psi, M, N, P, \alpha, P')$ **in** *name-list-avoiding*) *auto*
from *Trans* ‹$(p \cdot xvec) \sharp{*}\ N$› ‹$(p \cdot xvec) \sharp{*}\ P$› S **have** $\Psi \rhd M(\!|\lambda{*}(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P) \longmapsto\alpha \prec P'$
by(*simp add*: *inputChainAlpha'*)
moreover {
fix *K Tvec*
assume $\Psi \vdash M \leftrightarrow K$
moreover assume $set(p \cdot xvec) \subseteq supp(p \cdot N)$
hence $(p \cdot set(p \cdot xvec)) \subseteq (p \cdot supp(p \cdot N))$ **by** *simp*
with ‹*distinctPerm p*› **have** $set\ xvec \subseteq supp\ N$ **by**(*simp add*: *eqvts*)
moreover assume $length(p \cdot xvec) = length(Tvec::'a\ list)$
hence $length\ xvec = length\ Tvec$ **by** *simp*
moreover assume *distinct xvec*
ultimately have $Prop\ (K(\!|N[xvec::=Tvec]|\!))\ (P[xvec::=Tvec])$
by(*rule rInput*)
with ‹$length\ xvec = length\ Tvec$› S ‹*distinctPerm p*› ‹$(p \cdot xvec) \sharp{*}\ N$› ‹$(p \cdot xvec) \sharp{*}\ P$›

have *Prop* $(K\langle\!|(p \cdot N)[(p \cdot xvec)::=Tvec]|\!\rangle)$ $((p \cdot P)[(p \cdot xvec)::=Tvec])$
by(*simp add*: *renaming substTerm.renaming*)
}
moreover from *Trans* **have** *distinct xvec* **by**(*rule inputDistinct*)
hence *distinct*$(p \cdot xvec)$ **by** *simp*
ultimately show *?thesis* **using** ‹$(p \cdot xvec)$ $\sharp*$ Ψ› ‹$(p \cdot xvec)$ $\sharp*$ M› ‹$(p \cdot xvec)$ $\sharp*$ α› ‹$(p \cdot xvec)$ $\sharp*$ P'› ‹*distinct xvec*›
by(*rule-tac Goal*) *assumption*+
qed

lemma *outputCases*[*consumes 1*, *case-names cOutput*]:
fixes Ψ :: $'b$
and M :: $'a$
and N :: $'a$
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd M\langle N\rangle.P \longmapsto\alpha \prec P'$
and $\bigwedge K.\ \Psi \vdash M \leftrightarrow K \Longrightarrow Prop\ (K\langle N\rangle)\ P$

shows *Prop* α P'
using *assms*
by(*cases rule*: *semantics.cases*) (*auto simp add*: *residualInject psi.inject*)

lemma *caseCases*[*consumes 1*, *case-names cCase*]:
fixes Ψ :: $'b$
and Cs :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes *Trans*: $\Psi \rhd (Cases\ Cs) \longmapsto Rs$
and *rCase*: $\bigwedge\varphi\ P.\ [\![\Psi \rhd P \longmapsto Rs;\ (\varphi, P)\ mem\ Cs;\ \Psi \vdash \varphi;\ guarded\ P]\!] \Longrightarrow Prop$

shows *Prop*
using *assms*
by(*cases rule*: *semantics.cases*) (*auto simp add*: *residualInject psi.inject*)

lemma *resCases*[*consumes 7*, *case-names cOpen cRes*]:
fixes Ψ :: $'b$
and x :: *name*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd (\!|\nu x|\!)P \longmapsto\alpha \prec P'$
and $x \sharp \Psi$

and $x \sharp \alpha$
and $x \sharp P'$
and $bn\ \alpha \sharp* \Psi$
and $bn\ \alpha \sharp* P$
and $bn\ \alpha \sharp*\ subject\ \alpha$
and $rOpen$: $\bigwedge M\ xvec\ yvec\ y\ N\ P'.\ [\![\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle([(x,\ y)] \cdot N)\rangle \prec ([(x,\ y)] \cdot P');\ y \in supp\ N;$
$x \sharp N;\ x \sharp P';\ x \neq y;\ y \sharp xvec;\ y \sharp yvec;\ y \sharp M;\ distinct\ xvec;\ distinct\ yvec;$
$xvec \sharp* \Psi;\ y \sharp \Psi;\ yvec \sharp* \Psi;\ xvec \sharp* P;\ y \sharp P;\ yvec \sharp* P;\ xvec \sharp* M;\ y \sharp M;$
$yvec \sharp* M;\ xvec \sharp* yvec]\!] \Longrightarrow$
$Prop\ (M(\!|\nu*(xvec@y\#yvec)|\!)\langle N\rangle)\ P'$
and $rScope$: $\bigwedge P'.\ [\![\Psi \rhd P \longmapsto \alpha \prec P']\!] \Longrightarrow Prop\ \alpha\ ((\!|\nu x|\!)P')$

shows $Prop\ \alpha\ P'$
proof −
from *Trans* **have** $distinct(bn\ \alpha)$ **by**(*auto dest*: *boundOutputDistinct*)
have $length(bn\ \alpha) = residualLength(\alpha \prec P')$ **by** *simp*
note *Trans*
moreover have $length\ [] = inputLength((\!|\nu x|\!)P)$ **and** $distinct\ []$
by(*auto simp add*: *inputLength-inputLength′-inputLength″.simps*)
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
ultimately show *?thesis* **using** ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*\ subject\ \alpha$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp P'$› ‹$distinct(bn\ \alpha)$›
apply(*cases rule*: *semanticsCases*[*of* - - - - - - - - - - *x x*])
apply(*auto simp add*: *psi.inject alpha abs-fresh residualInject boundOutputApp boundOutput.inject eqvts*)
apply(*subgoal-tac* $y \in supp\ Na$)
apply(*rule-tac rOpen*)
apply(*auto simp add*: *residualInject boundOutputApp*)
apply(*drule-tac pi*=$[(x,\ y)]$ **in** *pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *calc-atm eqvts*)
by(*rule rScope*)
qed

lemma *resCases′*[*consumes 7*, *case-names cOpen cRes*]:
fixes Ψ :: $'b$
and x :: *name*
and P :: $('a,\ 'b,\ 'c)\ psi$
and α :: $'a\ action$
and P' :: $('a,\ 'b,\ 'c)\ psi$
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd (\!|\nu x|\!)P \longmapsto \alpha \prec P'$
and $x \sharp \Psi$

and $x \sharp \alpha$
and $x \sharp P'$
and $bn\ \alpha \sharp* \Psi$
and $bn\ \alpha \sharp* P$
and $bn\ \alpha \sharp*\ subject\ \alpha$
and $rOpen$: $\bigwedge M\ xvec\ yvec\ y\ N\ P'.\ [\![\Psi \rhd ([(x,\ y)] \cdot P) \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P';\ y \in supp\ N;$
$x \sharp N;\ x \sharp P';\ x \neq y;\ y \sharp xvec;\ y \sharp yvec;\ y \sharp M;\ distinct\ xvec;\ distinct\ yvec;$
$xvec \sharp* \Psi;\ y \sharp \Psi;\ yvec \sharp* \Psi;\ xvec \sharp* P;\ y \sharp P;\ yvec \sharp* P;\ xvec \sharp* M;\ y \sharp M;$
$yvec \sharp* M;\ xvec \sharp* yvec]\!] \Longrightarrow$
$Prop\ (M(\!|\nu*(xvec@y\#yvec)|\!)\langle N\rangle)\ P'$
and $rScope$: $\bigwedge P'.\ [\![\Psi \rhd P \longmapsto \alpha \prec P']\!] \Longrightarrow Prop\ \alpha\ ((\!|\nu x|\!)P')$

shows $Prop\ \alpha\ P'$
proof −
from *Trans* **have** $distinct(bn\ \alpha)$ **by**(*auto dest*: *boundOutputDistinct*)
have $length(bn\ \alpha) = residualLength(\alpha \prec P')$ **by** *simp*
note *Trans*
moreover have $length\ [] = inputLength((\!|\nu x|\!)P)$ **and** $distinct\ []$
by(*auto simp add*: *inputLength-inputLength′-inputLength″.simps*)
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
ultimately show *?thesis* **using** ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*\ subject\ \alpha$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp P'$› ‹$distinct(bn\ \alpha)$›
apply(*cases rule*: *semanticsCases*[*of* - - - - - - - - - - $x\ x$])
apply(*auto simp add*: *psi.inject alpha abs-fresh residualInject boundOutputApp boundOutput.inject eqvts*)
apply(*subgoal-tac* $y \in supp\ Na$)
apply(*rule-tac rOpen*)
apply(*auto simp add*: *residualInject boundOutputApp*)
apply(*drule-tac pi*=[$(x,\ y)$] **in** *semantics.eqvt*)
apply(*simp add*: *calc-atm eqvts*)
apply(*drule-tac pi*=[$(x,\ y)$] **in** *pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *calc-atm eqvts*)
by(*rule rScope*)
qed

abbreviation
statImpJudge (‹- ↪ -› [*80*, *80*] *80*)
where $\Psi \hookrightarrow \Psi' \equiv AssertionStatImp\ \Psi\ \Psi'$

lemma *statEqTransition*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rs :: $('a,\ 'b,\ 'c)\ residual$

```
  and   Ψ' :: 'b

  assumes Ψ ▷ P ⟼ Rs
  and     Ψ ≃ Ψ'

  shows Ψ' ▷ P ⟼ Rs
using assms
proof(nominal-induct avoiding: Ψ' rule: semantics.strong-induct)
  case(cInput Ψ M K xvec N Tvec P Ψ')
  from ‹Ψ ≃ Ψ'› ‹Ψ ⊢ M ↔ K› have Ψ' ⊢ M ↔ K
    by(simp add: AssertionStatImp-def AssertionStatEq-def)
  thus ?case using ‹distinct xvec› ‹set xvec ⊆ supp N› ‹length xvec = length Tvec›
    by(rule Input)
next
  case(Output Ψ M K N P Ψ')
  from ‹Ψ ≃ Ψ'› ‹Ψ ⊢ M ↔ K› have Ψ' ⊢ M ↔ K
    by(simp add: AssertionStatImp-def AssertionStatEq-def)
  thus ?case by(rule semantics.Output)
next
  case(Case Ψ P Rs φ Cs Ψ')
  then have Ψ' ▷ P ⟼ Rs by(rule-tac Case)
  moreover note ‹(φ, P) mem Cs›
  moreover from ‹Ψ ≃ Ψ'› ‹Ψ ⊢ φ› have Ψ' ⊢ φ
    by(simp add: AssertionStatImp-def AssertionStatEq-def)
  ultimately show ?case using ‹guarded P› by(rule semantics.Case)
next
  case(cPar1 Ψ ΨQ P α P' xvec Q Ψ')
  thus ?case
    by(rule-tac Par1) (auto intro: Composition)
next
  case(cPar2 Ψ ΨP Q α Q' xvec P Ψ')
  thus ?case
    by(rule-tac Par2) (auto intro: Composition)
next
  case(cComm1 Ψ ΨQ P M N P' xvec ΨP Q K zvec Q' yvec Ψ')
  thus ?case
    by(clarsimp, rule-tac Comm1) (blast intro: Composition statEqEnt)+
next
  case(cComm2 Ψ ΨQ P M zvec N P' xvec ΨP Q K Q' yvec Ψ')
  thus ?case
    by(clarsimp, rule-tac Comm2) (blast intro: Composition statEqEnt)+
next
  case(cOpen Ψ P M xvec N P' x yvec Ψ')
  thus ?case by(force intro: Open)
next
  case(cScope Ψ P α P' x Ψ')
  thus ?case by(force intro: Scope)
next
  case(Bang Ψ P Rs Ψ')
```

thus *?case* **by**(*force intro*: *semantics.Bang*)
qed

lemma *actionPar1Dest'*:
fixes α :: ('*a*::*fs-name*) *action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
and β :: ('*a*::*fs-name*) *action*
and Q :: ('*a*, '*b*, '*c*) *psi*
and R :: ('*a*, '*b*, '*c*) *psi*

assumes $\alpha \prec P = \beta \prec (Q \parallel R)$
and *bn* $\alpha \sharp* R$
and *bn* $\beta \sharp* R$

obtains T **where** $P = T \parallel R$ **and** $\alpha \prec T = \beta \prec Q$
using *assms*
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
by(*drule-tac boundOutputPar1Dest*) *auto*

lemma *actionPar2Dest'*:
fixes α :: ('*a*::*fs-name*) *action*
and P :: ('*a*, '*b*::*fs-name*, '*c*::*fs-name*) *psi*
and β :: ('*a*::*fs-name*) *action*
and Q :: ('*a*, '*b*, '*c*) *psi*
and R :: ('*a*, '*b*, '*c*) *psi*

assumes $\alpha \prec P = \beta \prec (Q \parallel R)$
and *bn* $\alpha \sharp* Q$
and *bn* $\beta \sharp* Q$

obtains T **where** $P = Q \parallel T$ **and** $\alpha \prec T = \beta \prec R$
using *assms*
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
by(*drule-tac boundOutputPar2Dest*) *auto*

lemma *expandNonTauFrame*:
fixes Ψ :: '*b*
and P :: ('*a*, '*b*, '*c*) *psi*
and α :: '*a action*
and P' :: ('*a*, '*b*, '*c*) *psi*
and A_P :: *name list*
and Ψ_P :: '*b*
and C :: '*d*::*fs-name*
and C' :: '*e*::*fs-name*

assumes *Trans*: $\Psi \rhd P \longmapsto\alpha \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P \rangle$

and $distinct\ A_P$
and $bn\ \alpha\ \sharp*\ subject\ \alpha$
and $A_P\ \sharp*\ P$
and $A_P\ \sharp*\ \alpha$
and $A_P\ \sharp*\ C$
and $A_P\ \sharp*\ C'$
and $bn\ \alpha\ \sharp*\ P$
and $bn\ \alpha\ \sharp*\ C'$
and $\alpha \neq \tau$

obtains $p\ \Psi'\ A_P'\ \Psi_P'$ **where** $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** $(p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P'$ **and** $distinctPerm\ p$ **and**
$extractFrame\ P' = \langle A_P', \Psi_P' \rangle$ **and** $A_P'\ \sharp*\ P'$ **and** $A_P'\ \sharp*\ \alpha$ **and** $A_P'\ \sharp*\ (p \cdot \alpha)$ **and**
$A_P'\ \sharp*\ C$ **and** $(bn(p \cdot \alpha))\ \sharp*\ C'$ **and** $(bn(p \cdot \alpha))\ \sharp*\ \alpha$ **and** $(bn(p \cdot \alpha))\ \sharp*\ P'$ **and** $distinct\ A_P'$
proof −
assume A: $\bigwedge p\ \Psi'\ \Psi_P'\ A_P'$.
$[\![set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha));\ (p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P';\ distinctPerm\ p;$
$extractFrame\ P' = \langle A_P', \Psi_P' \rangle;\ A_P'\ \sharp*\ P';\ A_P'\ \sharp*\ \alpha;\ A_P'\ \sharp*\ (p \cdot \alpha);$
$A_P'\ \sharp*\ C;\ (bn(p \cdot \alpha))\ \sharp*\ C';\ (bn(p \cdot \alpha))\ \sharp*\ \alpha;\ (bn(p \cdot \alpha))\ \sharp*\ P';\ distinct\ A_P']\!]$
$\Longrightarrow thesis$

from $Trans$ **have** $distinct(bn\ \alpha)$ **by**($auto\ dest$: $boundOutputDistinct$)

with $Trans$ ‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$A_P\ \sharp*\ P$› ‹$A_P\ \sharp*\ \alpha$› **have** $A_P\ \sharp*\ P'$
by($drule\text{-}tac\ freeFreshChainDerivative$) $auto$
{
fix $X :: name\ list$
and $Y :: 'b\ list$
and $Z :: ('a, 'b, 'c)\ psi\ list$

assume $bn\ \alpha\ \sharp*\ X$ **and** $bn\ \alpha\ \sharp*\ Y$ **and** $bn\ \alpha\ \sharp*\ Z$ **and** $A_P\ \sharp*\ X$ **and** $A_P\ \sharp*\ Y$ **and** $A_P\ \sharp*\ Z$

with $assms$ **obtain** $p\ \Psi'\ A_P'\ \Psi_P'$ **where** $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** $(p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P'$ **and** $distinctPerm\ p$
and $extractFrame\ P' = \langle A_P', \Psi_P' \rangle$ **and** $A_P'\ \sharp*\ P'$ **and** $A_P'\ \sharp*\ \alpha$ **and** $A_P'\ \sharp*\ (p \cdot \alpha)$
and $A_P'\ \sharp*\ C$ **and** $(bn(p \cdot \alpha))\ \sharp*\ C'$ **and** $(bn(p \cdot \alpha))\ \sharp*\ \alpha$ **and** $(bn(p \cdot \alpha))\ \sharp*\ P'$
and $A_P'\ \sharp*\ X$ **and** $A_P'\ \sharp*\ Y$ **and** $A_P'\ \sharp*\ Z$ **and** $distinct\ A_P'$
and $(bn(p \cdot \alpha))\ \sharp*\ X$ **and** $(bn(p \cdot \alpha))\ \sharp*\ Y$ **and** $(bn(p \cdot \alpha))\ \sharp*\ Z$
using ‹$A_P\ \sharp*\ P'$› ‹$distinct(bn\ \alpha)$›
proof($nominal\text{-}induct\ \Psi\ P\ Rs{=}{=}\alpha \prec P'\ A_P\ \Psi_P\ avoiding$: $C\ C'\ \alpha\ P'\ X\ Y\ Z$

arbitrary: *thesis rule*: *semanticsFrameInduct*)
case(*cAlpha* Ψ P A_P Ψ_P p C C' α P' X Y Z)
then obtain q Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *Sq*: $set\ q \subseteq set(bn\ \alpha) \times set(bn(q \cdot \alpha))$ **and** *PeqP'*: $(q \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P{}'$ **and** *distinctPerm* q
and *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}' \rangle$ **and** $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* \alpha$ **and** $A_P{}' \sharp* (q \cdot \alpha)$
and $A_P{}' \sharp* C$ **and** $(bn(q \cdot \alpha)) \sharp* C'$ **and** $(bn(q \cdot \alpha)) \sharp* \alpha$ **and** $(bn(q \cdot \alpha)) \sharp* P'$
and $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$ **and** *distinct* $A_P{}'$
and $(bn(q \cdot \alpha)) \sharp* X$ **and** $(bn(q \cdot \alpha)) \sharp* Y$ **and** $(bn(q \cdot \alpha)) \sharp* Z$
by *metis*

have *Sp*: $set\ p \subseteq set\ A_P \times set\ (p \cdot A_P)$ **by** *fact*

from *Sq* **have** $(p \cdot set\ q) \subseteq p \cdot (set(bn\ \alpha) \times set(bn(q \cdot \alpha)))$
by(*simp add*: *subsetClosed*)
hence $set(p \cdot q) \subseteq set(bn(p \cdot \alpha)) \times set(p \cdot bn(q \cdot \alpha))$
by(*simp add*: *eqvts*)
with ‹$A_P \sharp* \alpha$› ‹$(p \cdot A_P) \sharp* \alpha$› *Sp* **have** $set(p \cdot q) \subseteq set(bn\ \alpha) \times set(bn((p \cdot q) \cdot \alpha))$
by(*simp add*: *perm-compose bnEqvt*[*symmetric*])
moreover from *PeqP'* **have** $(p \cdot (q \cdot \Psi_P) \otimes \Psi') \simeq (p \cdot \Psi_P{}')$
by(*simp add*: *AssertionStatEqClosed*)
hence $((p \cdot q) \cdot p \cdot \Psi_P) \otimes (p \cdot \Psi') \simeq (p \cdot \Psi_P{}')$
apply(*subst perm-compose*[*symmetric*])
by(*simp add*: *eqvts*)
moreover from ‹*distinctPerm* q› **have** *distinctPerm* $(p \cdot q)$
by *simp*
moreover from ‹$(bn(q \cdot \alpha)) \sharp* C'$› **have** $(p \cdot bn(q \cdot \alpha)) \sharp* (p \cdot C')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* \alpha$› ‹$(p \cdot A_P) \sharp* \alpha$› ‹$A_P \sharp* C'$› ‹$(p \cdot A_P) \sharp* C'$› *Sp* **have** $bn((p \cdot q) \cdot \alpha) \sharp* C'$
by(*simp add*: *perm-compose bnEqvt*[*symmetric*])
moreover from *FrP'* **have** $(p \cdot extractFrame\ P') = p \cdot \langle A_P{}', \Psi_P{}' \rangle$ **by** *simp*
with ‹$A_P \sharp* P'$› ‹$(p \cdot A_P) \sharp* P'$› *Sp* **have** $extractFrame\ P' = \langle p \cdot A_P{}', p \cdot \Psi_P{}' \rangle$
by(*simp add*: *eqvts*)
moreover from ‹$A_P{}' \sharp* P'$› **have** $(p \cdot A_P{}') \sharp* (p \cdot P')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* P'$› ‹$(p \cdot A_P) \sharp* P'$› *Sp* **have** $(p \cdot A_P{}') \sharp* P'$ **by** *simp*
moreover from ‹$A_P{}' \sharp* \alpha$› **have** $(p \cdot A_P{}') \sharp* (p \cdot \alpha)$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* \alpha$› ‹$(p \cdot A_P) \sharp* \alpha$› *Sp* **have** $(p \cdot A_P{}') \sharp* \alpha$ **by** *simp*
moreover from ‹$A_P{}' \sharp* C$› **have** $(p \cdot A_P{}') \sharp* (p \cdot C)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* C$› ‹$(p \cdot A_P) \sharp* C$› *Sp* **have** $(p \cdot A_P{}') \sharp* C$ **by** *simp*
moreover from ‹$(bn(q \cdot \alpha)) \sharp* \alpha$› **have** $(p \cdot bn(q \cdot \alpha)) \sharp* (p \cdot \alpha)$

by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› *Sp* **have** $bn((p \cdot q) \cdot \alpha)$ ♯* α
by(*simp add*: *perm-compose eqvts*)
moreover from ‹$(bn(q \cdot \alpha))$ ♯* P'› **have** $(p \cdot bn(q \cdot \alpha))$ ♯* $(p \cdot P')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› ‹A_P ♯* P'› ‹$(p \cdot A_P)$ ♯* P'› *Sp* **have** $bn((p \cdot q) \cdot \alpha)$ ♯* P'
by(*simp add*: *perm-compose eqvts*)
moreover from ‹$A_P{}'$ ♯* $(q \cdot \alpha)$› **have** $(p \cdot A_P{}')$ ♯* $(p \cdot q \cdot \alpha)$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with *Sp* ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› **have** $(p \cdot A_P{}')$ ♯* $((p \cdot q) \cdot \alpha)$
by(*simp add*: *perm-compose*)
moreover from ‹$A_P{}'$ ♯* X› **have** $(p \cdot A_P{}')$ ♯* $(p \cdot X)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* X› ‹$(p \cdot A_P)$ ♯* X› *Sp* **have** $(p \cdot A_P{}')$ ♯* X **by** *simp*
moreover from ‹$A_P{}'$ ♯* Y› **have** $(p \cdot A_P{}')$ ♯* $(p \cdot Y)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* Y› ‹$(p \cdot A_P)$ ♯* Y› *Sp* **have** $(p \cdot A_P{}')$ ♯* Y **by** *simp*
moreover from ‹$A_P{}'$ ♯* Z› **have** $(p \cdot A_P{}')$ ♯* $(p \cdot Z)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* Z› ‹$(p \cdot A_P)$ ♯* Z› *Sp* **have** $(p \cdot A_P{}')$ ♯* Z **by** *simp*
moreover from ‹$(bn(q \cdot \alpha))$ ♯* X› **have** $(p \cdot bn(q \cdot \alpha))$ ♯* $(p \cdot X)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› ‹A_P ♯* X› ‹$(p \cdot A_P)$ ♯* X› *Sp* **have** $bn((p \cdot q) \cdot \alpha)$ ♯* X
by(*simp add*: *perm-compose eqvts*)
moreover from ‹$(bn(q \cdot \alpha))$ ♯* Y› **have** $(p \cdot bn(q \cdot \alpha))$ ♯* $(p \cdot Y)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› ‹A_P ♯* Y› ‹$(p \cdot A_P)$ ♯* Y› *Sp* **have** $bn((p \cdot q) \cdot \alpha)$ ♯* Y
by(*simp add*: *perm-compose eqvts*)
moreover from ‹$(bn(q \cdot \alpha))$ ♯* Z› **have** $(p \cdot bn(q \cdot \alpha))$ ♯* $(p \cdot Z)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_P ♯* α› ‹$(p \cdot A_P)$ ♯* α› ‹A_P ♯* Z› ‹$(p \cdot A_P)$ ♯* Z› *Sp* **have** $bn((p \cdot q) \cdot \alpha)$ ♯* Z
by(*simp add*: *perm-compose eqvts*)
moreover from ‹*distinct* $A_P{}'$› **have** *distinct*$(p \cdot A_P{}')$ **by** *simp*
ultimately show *?case*
by(*rule-tac cAlpha*)
next
case(*cInput* Ψ M K *xvec* N *Tvec* P C C' α P' X Y Z)
moreover obtain A_P Ψ_P **where** *extractFrame*$(P[xvec::=Tvec]) = \langle A_P, \Psi_P\rangle$ **and** *distinct* A_P
and A_P ♯* $(C, P[xvec::=Tvec], \alpha, P', X, Y, Z, N)$
by(*rule freshFrame*)
moreover have $\mathbf{1} \otimes \Psi_P \simeq \Psi_P$
by(*blast intro*: *Identity Commutativity AssertionStatEqTrans*)
ultimately show *?case*

by(*rule-tac cInput*) (*assumption* | *simp add*: *residualInject*)+
next
case(*cOutput* Ψ M K N P C C' α P' X Y Z)
moreover obtain A_P Ψ_P **where** *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *distinct* A_P
and $A_P \sharp* (C, C', P, \alpha, N, P', X, Y, Z)$
by(*rule freshFrame*)
moreover have $\mathbf{1} \otimes \Psi_P \simeq \Psi_P$
by(*blast intro*: *Identity Commutativity AssertionStatEqTrans*)
ultimately show *?case* **by**(*simp add*: *residualInject*)
next
case(*cCase* Ψ P φ *Cs* A_P Ψ_P C C' α P' X Y Z)
moreover from ‹$bn\ \alpha \sharp* (Cases\ Cs)$› ‹$(\varphi, P)$ *mem Cs*› **have** $bn\ \alpha \sharp* P$ **by**(*auto dest*: *memFreshChain*)
ultimately obtain p Ψ' $A_P{}'$ $\Psi_P{}'$ **where** S: $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$
and FrP': *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$
and $PeqP'$: $(p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P{}'$ **and** *distinct* $A_P{}'$
and $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* \alpha$ **and** $A_P{}' \sharp* (p \cdot \alpha)$
and $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$
and *distinctPerm* p **and** $(bn(p \cdot \alpha)) \sharp* \alpha$ **and** $(bn(p \cdot \alpha)) \sharp* P'$
and $(bn(p \cdot \alpha)) \sharp* C'$ **and** $(bn(p \cdot \alpha)) \sharp* X$ **and** $(bn(p \cdot \alpha)) \sharp* Y$ **and** $(bn(p \cdot \alpha)) \sharp* Z$
by(*rule-tac cCase*) (*assumption* | *simp* (*no-asm-use*))+
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $(p \cdot \Psi_P) \simeq (p \cdot \mathbf{1})$
by(*simp add*: *AssertionStatEqClosed*)
hence $(p \cdot \Psi_P) \simeq \mathbf{1}$ **by**(*simp add*: *permBottom*)
with $PeqP'$ **have** $(\mathbf{1} \otimes \Psi') \simeq \Psi_P{}'$
by(*metis Identity AssertionStatEqTrans composition′ Commutativity Associativity AssertionStatEqSym*)
ultimately show *?case* **using** *cCase* ‹$bn\ \alpha \sharp* P$›
by(*rule-tac cCase*(*20*)) (*assumption* | *simp*)+
next
case(*cPar1* Ψ Ψ_Q P α P' A_Q Q A_P Ψ_P C C' α' PQ' X Y Z)
have FrP: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** FrQ: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
by *fact*+

note ‹$bn\ \alpha' \sharp* subject\ \alpha'$›
moreover from ‹$bn\ \alpha' \sharp* (P \parallel Q)$› **have** $bn\ \alpha' \sharp* P$ **and** $bn\ \alpha' \sharp* Q$ **by** *simp*+
moreover with FrP FrQ ‹$A_P \sharp* \alpha'$› ‹$A_Q \sharp* \alpha'$› **have** $bn\ \alpha' \sharp* \Psi_P$ **and** $bn\ \alpha' \sharp* \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+

moreover from ‹$bn\ \alpha' \sharp* X$› ‹$A_Q \sharp* \alpha'$› **have** $bn\ \alpha' \sharp* (X@A_Q)$ **by** *simp*
moreover from ‹$bn\ \alpha' \sharp* Y$› ‹$bn\ \alpha' \sharp* \Psi_Q$› **have** $bn\ \alpha' \sharp* (\Psi_Q\#Y)$ **by** *simp*

moreover from ‹$bn\ \alpha'\ \sharp*\ Z$› ‹$bn\ \alpha'\ \sharp*\ Q$› **have** $bn\ \alpha'\ \sharp*\ (Q\#Z)$ **by** *simp*
moreover from ‹$A_P\ \sharp*\ X$› ‹$A_P\ \sharp*\ A_Q$› **have** $A_P\ \sharp*\ (X@A_Q)$ **by** *simp*
moreover from ‹$A_P\ \sharp*\ Y$› ‹$A_P\ \sharp*\ \Psi_Q$› **have** $A_P\ \sharp*\ (\Psi_Q\#Y)$ **by** *force*
moreover from ‹$A_P\ \sharp*\ Z$› ‹$A_P\ \sharp*\ Q$› **have** $A_P\ \sharp*\ (Q\#Z)$ **by** *simp*
moreover from ‹$\alpha \prec (P' \parallel Q) = \alpha' \prec PQ'$› ‹$bn\ \alpha\ \sharp*\ Q$› ‹$bn\ \alpha'\ \sharp*\ Q$› ‹$bn\ \alpha\ \sharp*\ \alpha'$›
obtain P'' **where** A: $\alpha \prec P' = \alpha' \prec P''$ **and** $PQ' = P'' \parallel Q$
by(*metis actionPar1Dest'*)
moreover from ‹$A_P\ \sharp*\ PQ'$› ‹$PQ' = P'' \parallel Q$› **have** $A_P\ \sharp*\ P''$ **by** *simp*
ultimately obtain $p\ \Psi'\ A_P'\ \Psi_P'$ **where** S: $set\ p \subseteq set(bn\ \alpha') \times set\ (bn(p \cdot \alpha'))$ **and** $PeqP'$: $((p \cdot \Psi_P) \otimes \Psi') \simeq \Psi_P'$
and $distinctPerm\ p$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ C'$ **and** FrP': $extractFrame\ P'' = \langle A_P', \Psi_P'\rangle$
and $A_P'\ \sharp*\ P''$ **and** $A_P'\ \sharp*\ \alpha'$ $A_P'\ \sharp*\ (p \cdot \alpha')$ **and** $A_P'\ \sharp*\ C$
and $(bn(p \cdot \alpha'))\ \sharp*\ \alpha'$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ P''$ **and** $distinct\ A_P'$
and $A_P'\ \sharp*\ (X\ @\ A_Q)$ **and** $A_P'\ \sharp*\ (\Psi_Q\#Y)$
and $A_P'\ \sharp*\ (Q\#Z)$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ (X\ @\ A_Q)$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ (\Psi_Q\#Y)$
and $(bn(p \cdot \alpha'))\ \sharp*\ (Q\#Z)$ **using** *cPar1*
by(*rule-tac cPar1*)

then have $A_P'\ \sharp*\ Q$ **and** $A_P'\ \sharp*\ Z$ **and** $A_P'\ \sharp*\ A_Q$ **and** $A_P'\ \sharp*\ X$ **and** $A_P'\ \sharp*\ \Psi_Q$ **and** $A_P'\ \sharp*\ Y$
and $(bn(p \cdot \alpha'))\ \sharp*\ A_Q$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ X$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ Y$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ Z$ **and** $(bn(p \cdot \alpha'))\ \sharp*\ \Psi_Q$
and $(bn(p \cdot \alpha'))\ \sharp*\ Q$
by(*simp del*: *freshChainSimps*)+

from ‹$A_Q\ \sharp*\ PQ'$› ‹$PQ' = P'' \parallel Q$› ‹$A_P'\ \sharp*\ A_Q$› FrP' **have** $A_Q\ \sharp*\ \Psi_P'$
by(*force dest*: *extractFrameFreshChain*)
note S
moreover from $PeqP'$ **have** $((p \cdot (\Psi_P \otimes \Psi_Q)) \otimes \Psi') \simeq \Psi_P' \otimes (p \cdot \Psi_Q)$
by(*simp add*: *eqvts*) (*metis Composition Associativity AssertionStatEqTrans AssertionStatEqSym Commutativity*)
with ‹$(bn(p \cdot \alpha'))\ \sharp*\ \Psi_Q$› ‹$bn\ \alpha'\ \sharp*\ \Psi_Q$› S **have** $((p \cdot (\Psi_P \otimes \Psi_Q)) \otimes \Psi') \simeq \Psi_P' \otimes \Psi_Q$
by *simp*
moreover from ‹$PQ' = P'' \parallel Q$› ‹$A_P'\ \sharp*\ A_Q$› ‹$A_P'\ \sharp*\ \Psi_Q$› ‹$A_Q\ \sharp*\ \Psi_P'$› ‹$A_Q\ \sharp*\ PQ'$› FrP' FrQ **have** $extractFrame\ PQ' = \langle (A_P'@A_Q), \Psi_P' \otimes \Psi_Q\rangle$
by *simp*
moreover note ‹$distinctPerm\ p$› ‹$(bn(p \cdot \alpha'))\ \sharp*\ C'$›
moreover from ‹$A_P'\ \sharp*\ P''$› ‹$A_P'\ \sharp*\ Q$› ‹$PQ' = P'' \parallel Q$› **have** $A_P'\ \sharp*\ PQ'$ **by** *simp*
moreover note ‹$A_Q\ \sharp*\ PQ'$› ‹$A_P'\ \sharp*\ \alpha'$› ‹$A_Q\ \sharp*\ \alpha'$› ‹$A_P'\ \sharp*\ C$› ‹$A_Q\ \sharp*\ C$› ‹$(bn(p \cdot \alpha'))\ \sharp*\ \alpha'$›
moreover from ‹$bn\ \alpha'\ \sharp*\ Q$› **have** $(bn(p \cdot \alpha'))\ \sharp*\ (p \cdot Q)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *bnEqvt*[*symmetric*])

with ‹$bn\ \alpha\ \sharp*\ Q$› ‹$(bn(p \cdot \alpha'))\ \sharp*\ Q$› S **have** $(bn(p \cdot \alpha'))\ \sharp*\ Q$ **by** *simp*
with ‹$(bn(p \cdot \alpha'))\ \sharp*\ P''$› ‹$PQ' = P''\ \|\ Q$› **have** $(bn(p \cdot \alpha'))\ \sharp*\ PQ'$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ \alpha'$› ‹$A_Q\ \sharp*\ \alpha'$› **have** $(A_P'@A_Q)\ \sharp*\ \alpha'$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ C$› ‹$A_Q\ \sharp*\ C$› **have** $(A_P'@A_Q)\ \sharp*\ C$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ X$› ‹$A_Q\ \sharp*\ X$› **have** $(A_P'@A_Q)\ \sharp*\ X$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ Y$› ‹$A_Q\ \sharp*\ Y$› **have** $(A_P'@A_Q)\ \sharp*\ Y$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ Z$› ‹$A_Q\ \sharp*\ Z$› **have** $(A_P'@A_Q)\ \sharp*\ Z$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ PQ'$› ‹$A_Q\ \sharp*\ PQ'$› **have** $(A_P'@A_Q)\ \sharp*\ PQ'$ **by** *simp*
moreover from ‹$A_P'\ \sharp*\ \alpha'$› ‹$A_Q\ \sharp*\ \alpha'$› **have** $(A_P'@A_Q)\ \sharp*\ \alpha'$ **by** *simp*
moreover from ‹$A_Q\ \sharp*\ \alpha'$› **have** $(p \cdot A_Q)\ \sharp*\ (p \cdot \alpha')$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *bnEqvt*[*symmetric*])
with S ‹$A_Q\ \sharp*\ \alpha'$› ‹$bn(p \cdot \alpha')\ \sharp*\ A_Q$› **have** $A_Q\ \sharp*\ (p \cdot \alpha')$
by *simp*
with ‹$A_P'\ \sharp*\ (p \cdot \alpha')$› ‹$A_Q\ \sharp*\ \alpha'$› ‹$bn(p \cdot \alpha')\ \sharp*\ A_Q$› S **have** $(A_P'@A_Q)\ \sharp*$
$(p \cdot \alpha')$
by *simp*
moreover from ‹$A_P'\ \sharp*\ A_Q$› ‹*distinct* A_P'› ‹*distinct* A_Q› **have** *distinct*$(A_P'@A_Q)$
by *auto*
moreover note ‹$(bn(p \cdot \alpha'))\ \sharp*\ X$› ‹$(bn(p \cdot \alpha'))\ \sharp*\ Y$› ‹$(bn(p \cdot \alpha'))\ \sharp*\ Z$›
ultimately show *?case* **using** *cPar1*
by *metis*
next
case(*cPar2* $\Psi\ \Psi_P\ Q\ \alpha\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ C\ C'\ \alpha'\ PQ'\ X\ Y\ Z$)
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *FrQ*: *extractFrame* $Q = \langle A_Q,$
$\Psi_Q\rangle$
by *fact*+

note ‹$bn\ \alpha'\ \sharp*$ *subject* α'›
moreover from ‹$bn\ \alpha'\ \sharp*\ (P\ \|\ Q)$› **have** $bn\ \alpha'\ \sharp*\ Q$ **and** $bn\ \alpha'\ \sharp*\ P$ **by**
simp+
moreover with *FrP FrQ* ‹$A_P\ \sharp*\ \alpha'$› ‹$A_Q\ \sharp*\ \alpha'$› **have** $bn\ \alpha'\ \sharp*\ \Psi_P$ **and** bn
$\alpha'\ \sharp*\ \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+

moreover from ‹$bn\ \alpha'\ \sharp*\ X$› ‹$A_P\ \sharp*\ \alpha'$› **have** $bn\ \alpha'\ \sharp*\ (X@A_P)$ **by** *simp*
moreover from ‹$bn\ \alpha'\ \sharp*\ Y$› ‹$bn\ \alpha'\ \sharp*\ \Psi_P$› **have** $bn\ \alpha'\ \sharp*\ (\Psi_P\#Y)$ **by** *simp*
moreover from ‹$bn\ \alpha'\ \sharp*\ Z$› ‹$bn\ \alpha'\ \sharp*\ P$› **have** $bn\ \alpha'\ \sharp*\ (P\#Z)$ **by** *simp*
moreover from ‹$A_Q\ \sharp*\ X$› ‹$A_P\ \sharp*\ A_Q$› **have** $A_Q\ \sharp*\ (X@A_P)$ **by** *simp*
moreover from ‹$A_Q\ \sharp*\ Y$› ‹$A_Q\ \sharp*\ \Psi_P$› **have** $A_Q\ \sharp*\ (\Psi_P\#Y)$ **by** *force*
moreover from ‹$A_Q\ \sharp*\ Z$› ‹$A_Q\ \sharp*\ P$› **have** $A_Q\ \sharp*\ (P\#Z)$ **by** *simp*
moreover from ‹$\alpha \prec (P\ \|\ Q') = \alpha' \prec PQ'$› ‹$bn\ \alpha\ \sharp*\ P$› ‹$bn\ \alpha'\ \sharp*\ P$› ‹$bn$
$\alpha\ \sharp*\ \alpha'$›
obtain Q'' **where** A: $\alpha \prec Q' = \alpha' \prec Q''$ **and** $PQ' = P\ \|\ Q''$
by(*metis actionPar2Dest'*)
moreover from ‹$A_Q\ \sharp*\ PQ'$› ‹$PQ' = P\ \|\ Q''$› **have** $A_Q\ \sharp*\ Q''$ **by** *simp*
ultimately obtain $p\ \Psi'\ A_Q'\ \Psi_Q'$ **where** S: *set* $p \subseteq$ *set*$(bn\ \alpha') \times$ *set* $(bn(p$
$\cdot\ \alpha'))$ **and** *QeqQ'*: $((p \cdot \Psi_Q) \otimes \Psi') \simeq \Psi_Q'$
and *distinctPerm* p **and** $(bn(p \cdot \alpha'))\ \sharp*\ C'$ **and**
FrQ': *extractFrame* $Q'' = \langle A_Q', \Psi_Q'\rangle$

and $A_Q{}' \sharp* Q''$ **and** $A_Q{}' \sharp* \alpha'$ $A_Q{}' \sharp* (p \cdot \alpha')$ **and** $A_Q{}' \sharp* C$
and $(bn(p \cdot \alpha')) \sharp* \alpha'$ **and** $(bn(p \cdot \alpha')) \sharp* Q''$ **and** *distinct* $A_Q{}'$
and $A_Q{}' \sharp* (X @ A_P)$ **and** $A_Q{}' \sharp* (\Psi_P \# Y)$
and $A_Q{}' \sharp* (P \# Z)$ **and** $(bn(p \cdot \alpha')) \sharp* (X @ A_P)$ **and** $(bn(p \cdot \alpha')) \sharp* (\Psi_P \# Y)$
and $(bn(p \cdot \alpha')) \sharp* (P \# Z)$ **using** *cPar2*
by(*rule-tac cPar2*)

then have $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* Z$ **and** $A_Q{}' \sharp* A_P$ **and** $A_Q{}' \sharp* X$ **and** $A_Q{}' \sharp* \Psi_P$ **and** $A_Q{}' \sharp* Y$
and $(bn(p \cdot \alpha')) \sharp* A_P$ **and** $(bn(p \cdot \alpha')) \sharp* X$ **and** $(bn(p \cdot \alpha')) \sharp* Y$ **and** $(bn(p \cdot \alpha')) \sharp* Z$ **and** $(bn(p \cdot \alpha')) \sharp* \Psi_P$
and $(bn(p \cdot \alpha')) \sharp* P$
by(*simp del*: *freshChainSimps*)+

from ‹$A_P \sharp* PQ'$› ‹$PQ' = P \parallel Q''$› ‹$A_Q{}' \sharp* A_P$› FrQ' **have** $A_P \sharp* \Psi_Q{}'$
by(*force dest*: *extractFrameFreshChain*)
note S
moreover from $QeqQ'$ **have** $((p \cdot (\Psi_P \otimes \Psi_Q)) \otimes \Psi') \simeq (p \cdot \Psi_P) \otimes \Psi_Q{}'$
by(*simp add*: *eqvts*) (*metis Composition Associativity AssertionStatEqTrans AssertionStatEqSym Commutativity*)
with ‹$(bn(p \cdot \alpha')) \sharp* \Psi_P$› ‹$bn\ \alpha' \sharp* \Psi_P$› S **have** $((p \cdot (\Psi_P \otimes \Psi_Q)) \otimes \Psi') \simeq \Psi_P \otimes \Psi_Q{}'$
by *simp*
moreover from ‹$PQ' = P \parallel Q''$› ‹$A_Q{}' \sharp* A_P$› ‹$A_Q{}' \sharp* \Psi_P$› ‹$A_P \sharp* \Psi_Q{}'$› ‹$A_P \sharp* PQ'$› FrQ' FrP **have** $extractFrame\ PQ' = \langle (A_P @ A_Q{}'), \Psi_P \otimes \Psi_Q{}' \rangle$
by *simp*
moreover note ‹*distinctPerm p*› ‹$(bn(p \cdot \alpha')) \sharp* C'$›
moreover from ‹$A_Q{}' \sharp* Q''$› ‹$A_Q{}' \sharp* P$› ‹$PQ' = P \parallel Q''$› **have** $A_Q{}' \sharp* PQ'$ **by** *simp*
moreover note ‹$A_Q \sharp* PQ'$› ‹$A_Q{}' \sharp* \alpha'$› ‹$A_Q \sharp* \alpha'$› ‹$A_Q{}' \sharp* C$› ‹$A_Q \sharp* C$› ‹$(bn(p \cdot \alpha')) \sharp* \alpha'$›
moreover from ‹$bn\ \alpha' \sharp* Q$› **have** $(bn(p \cdot \alpha')) \sharp* (p \cdot Q)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *bnEqvt*[*symmetric*])
with ‹$bn\ \alpha \sharp* P$› ‹$(bn(p \cdot \alpha')) \sharp* P$› S **have** $(bn(p \cdot \alpha')) \sharp* P$ **by** *simp*
with ‹$(bn(p \cdot \alpha')) \sharp* Q''$› ‹$PQ' = P \parallel Q''$› **have** $(bn(p \cdot \alpha')) \sharp* PQ'$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* \alpha'$› ‹$A_P \sharp* \alpha'$› **have** $(A_P @ A_Q{}') \sharp* \alpha'$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* C$› ‹$A_P \sharp* C$› **have** $(A_P @ A_Q{}') \sharp* C$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* X$› ‹$A_P \sharp* X$› **have** $(A_P @ A_Q{}') \sharp* X$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* Y$› ‹$A_P \sharp* Y$› **have** $(A_P @ A_Q{}') \sharp* Y$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* Z$› ‹$A_P \sharp* Z$› **have** $(A_P @ A_Q{}') \sharp* Z$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* PQ'$› ‹$A_P \sharp* PQ'$› **have** $(A_P @ A_Q{}') \sharp* PQ'$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* \alpha'$› ‹$A_P \sharp* \alpha'$› **have** $(A_P @ A_Q{}') \sharp* \alpha'$ **by** *simp*
moreover from ‹$A_P \sharp* \alpha'$› **have** $(p \cdot A_P) \sharp* (p \cdot \alpha')$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *bnEqvt*[*symmetric*])
with S ‹$A_P \sharp* \alpha'$› ‹$bn(p \cdot \alpha') \sharp* A_P$› **have** $A_P \sharp* (p \cdot \alpha')$
by *simp*

with ‹$A_Q{}' \sharp* (p \cdot \alpha')$› ‹$A_P \sharp* \alpha'$› ‹$bn(p \cdot \alpha') \sharp* A_P$› S **have** $(A_P@A_Q{}') \sharp* (p \cdot \alpha')$
by *simp*
moreover note ‹$(bn(p \cdot \alpha')) \sharp* X$› ‹$(bn(p \cdot \alpha')) \sharp* Y$› ‹$(bn(p \cdot \alpha')) \sharp* Z$›
moreover from ‹$A_Q{}' \sharp* A_P$› ‹*distinct* A_P› ‹*distinct* $A_Q{}'$› **have** *distinct*$(A_P@A_Q{}')$ **by** *auto*
ultimately show *?case* **using** *cPar2*
by *metis*
next
case *cComm1*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *cComm2*
thus *?case* **by**(*simp add*: *residualInject*)
next
case(*cOpen* Ψ P M *xvec1* *xvec2* N P' x A_P Ψ_P C C' α P'' X Y Z)
from ‹$M(\!|\nu*(xvec1@x\#xvec2)|\!)\langle N\rangle \prec P' = \alpha \prec P''$› ‹$x \sharp xvec1$› ‹$x \sharp xvec2$› ‹$x \sharp \alpha$› ‹$x \sharp P''$› ‹*distinct*$(bn\ \alpha)$› ‹$A_P \sharp* \alpha$› ‹$x \sharp \alpha$›
obtain *yvec1* y *yvec2* N' **where** *yvecEq*: $bn\ \alpha = yvec1@y\#yvec2$ **and** $P'eqP''$: $(\!|\nu*(xvec1@xvec2)|\!)N \prec' P' = (\!|\nu*(yvec1@yvec2)|\!)([(x, y)] \cdot N') \prec' ([(x, y)] \cdot P'')$ **and** $A_P \sharp* N'$ **and** *Subj*: *subject* α = *Some* M **and** $x \sharp N'$ **and** αeq: $\alpha = M(\!|\nu*(yvec1@y\#yvec2)|\!)\langle N'\rangle$
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
apply(*rule boundOutputOpenDest*)
by *assumption auto*
note ‹$A_P \sharp* P$› ‹$A_P \sharp* M$›
moreover from *Subj* *yvecEq* ‹$bn\ \alpha \sharp*$ *subject* α› **have** *yvec1* $\sharp* M$ *yvec2* $\sharp* M$ **by** *simp*+
moreover from *yvecEq* ‹$A_P \sharp* \alpha$› **have** $A_P \sharp* (yvec1@yvec2)$ **by** *simp*
moreover note ‹$A_P \sharp* C$›
moreover from *yvecEq* ‹$bn\ \alpha \sharp* (\!|\nu x|\!)P$› ‹$x \sharp \alpha$› **have** $(yvec1@yvec2) \sharp* P$ **by** *simp*
moreover from *yvecEq* ‹$bn\ \alpha \sharp* C'$› ‹$bn\ \alpha \sharp* X$› ‹$bn\ \alpha \sharp* Y$› ‹$bn\ \alpha \sharp* Z$› ‹*distinct*$(bn\ \alpha)$› ‹$x \sharp \alpha$›
have $(yvec1@yvec2) \sharp* C'$ **and** $(yvec1@yvec2) \sharp* (x\#y\#X)$ **and** $(yvec1@yvec2) \sharp* Y$ **and** $(yvec1@yvec2) \sharp* Z$
by *simp*+
moreover from ‹$A_P \sharp* X$› ‹$x \sharp A_P$› ‹$A_P \sharp* \alpha$› *yvecEq* **have** $A_P \sharp* (x\#y\#X)$ **by** *simp*
moreover note ‹$A_P \sharp* Y$› ‹$A_P \sharp* Z$›
moreover from ‹$A_P \sharp* N'$› ‹$A_P \sharp* P''$› ‹$x \sharp A_P$› ‹$A_P \sharp* \alpha$› *yvecEq* **have** $A_P \sharp* ([(x, y)] \cdot N')$ **and** $A_P \sharp* ([(x, y)] \cdot P'')$
by *simp*+
moreover from *yvecEq* ‹*distinct*$(bn\ \alpha)$› **have** *distinct*$(yvec1@yvec2)$ **by** *simp*
moreover from $P'eqP''$ **have** $M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec P' = M(\!|\nu*(yvec1@yvec2)|\!)\langle([(x, y)] \cdot N')\rangle \prec ([(x, y)] \cdot P'')$
by(*simp add*: *residualInject*)
ultimately obtain p Ψ' $A_P{}'$ $\Psi_P{}'$ **where** S: *set* $p \subseteq$ *set* $(yvec1@yvec2) \times$ *set*

$(p \cdot (yvec1@yvec2))$ **and** $PeqP'$: $((p \cdot \Psi_P) \otimes \Psi') \simeq \Psi_P{}'$
and $distinctPerm\ p$ **and** $(p \cdot (yvec1@yvec2)) \sharp*$ C' **and** FrP': $extractFrame([(x, y)] \cdot P'') = \langle A_P{}', \Psi_P{}'\rangle$
and $A_P{}' \sharp* ([(x, y)] \cdot P'')$ **and** $A_P{}' \sharp* ([(x, y)] \cdot N')$ **and** $A_P{}' \sharp* C$ **and** $(p \cdot (yvec1@yvec2)) \sharp* ([(x, y)] \cdot N')$ **and** $A_P{}' \sharp* M$ **and** $(p \cdot (yvec1@yvec2)) \sharp* (yvec1@yvec2)$ **and** $(p \cdot (yvec1@yvec2)) \sharp* M$ **and** $distinct\ A_P{}'$
and $(p \cdot (yvec1@yvec2)) \sharp* ([(x, y)] \cdot P'')$ **and** $(yvec1@yvec2) \sharp* A_P{}'$ **and** $(p \cdot (yvec1@yvec2)) \sharp* A_P{}'$
and $A_P{}' \sharp* (x\#y\#X)$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$ **and** $(p \cdot (yvec1@yvec2)) \sharp* (x\#y\#X)$
and $(p \cdot (yvec1@yvec2)) \sharp* Y$ **and** $(p \cdot (yvec1@yvec2)) \sharp* Z$ **using** ‹$A_P \sharp* C'$›
by(*rule-tac cOpen*(*4*)[**where** *bd*=$x\#y\#X$]) (*assumption* | *simp-all*)+

from ‹$A_P{}' \sharp* (x\#y\#X)$› **have** $x \sharp A_P{}'$ **and** $y \sharp A_P{}'$ **and** $A_P{}' \sharp* X$ **by** *simp*+
from ‹$(p \cdot (yvec1@yvec2)) \sharp* (x\#y\#X)$› **have** $x \sharp (p \cdot (yvec1@yvec2))$ **and** $y \sharp (p \cdot (yvec1@yvec2))$ **and** $(p \cdot (yvec1@yvec2)) \sharp* X$ **by** *simp*+

from ‹$x \sharp \alpha$› *yvecEq* **have** $x \sharp yvec1$ **and** $x \neq y$ **and** $x \sharp yvec2$ **by** *simp*+
from ‹$distinct(bn\ \alpha)$› *yvecEq* **have** $yvec1 \sharp* yvec2$ **and** $y \sharp yvec1$ **and** $y \sharp yvec2$ **by** *simp*+
from ‹$bn\ \alpha \sharp* C'$› *yvecEq* **have** $yvec1 \sharp* C'$ **and** $y \sharp C'$ **and** $yvec2 \sharp* C'$ **by** *simp*+

from S ‹$x \sharp \alpha$› ‹$x \sharp p \cdot (yvec1@yvec2)$› *yvecEq* **have** $x \sharp p$ **by**(*rule-tac freshAlphaSwap*) (*assumption* | *simp*)+
from S ‹$distinct(bn\ \alpha)$› ‹$y \sharp p \cdot (yvec1@yvec2)$› *yvecEq* **have** $y \sharp p$ **by**(*rule-tac freshAlphaSwap*) (*assumption* | *simp*)+

from *yvecEq* S ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p \cdot (yvec1@yvec2)$› ‹$y \sharp p \cdot (yvec1@yvec2)$›
have $set\ ((y, x)\#p) \subseteq set(bn\ \alpha) \times set(bn(((y, x)\#p) \cdot \alpha))$
apply(*simp add*: *bnEqvt*[*symmetric*])
by(*auto simp add*: *eqvts calc-atm*)

moreover from $PeqP'$ **have** $([(y, x)] \cdot ((p \cdot \Psi_P) \otimes \Psi')) \simeq [(y, x)] \cdot \Psi_P{}'$
by(*simp add*: *AssertionStatEqClosed*)
hence $(((y, x)\#p) \cdot \Psi_P) \otimes ([(y, x)] \cdot \Psi') \simeq ([(y, x)] \cdot \Psi_P{}')$
by(*simp add*: *eqvts*)
moreover from ‹$distinctPerm\ p$› S ‹$x \neq y$› ‹$x \sharp p$› ‹$y \sharp p$› **have** $distinctPerm((y, x)\#p)$
by *simp*
moreover from FrP' **have** $([(x, y)] \cdot (extractFrame([(x, y)] \cdot P''))) = ([(x, y)] \cdot \langle A_P{}', \Psi_P{}'\rangle)$
by *simp*
with ‹$x \sharp A_P{}'$› ‹$y \sharp A_P{}'$› **have** $extractFrame\ P'' = \langle A_P{}', ([(y, x)] \cdot \Psi_P{}')\rangle$
by(*simp add*: *eqvts name-swap*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp N'$› ‹$(p \cdot (yvec1@yvec2)) \sharp* ([(x, y)] \cdot$

N'› **have** $([(y, x)] \cdot p \cdot (yvec1@yvec2)) \sharp* ([(y, x)] \cdot [(x, y)] \cdot N')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence $(((y, x)\#p) \cdot (yvec1@yvec2)) \sharp* N'$ **by**(*simp add*: *name-swap*)
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$x \sharp C$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp N'$› *yvecEq*
have $bn(((y, x)\#p) \cdot \alpha) \sharp* N'$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *eqvts perm-compose calc-atm freshChainSimps*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp N'$› ‹$(p \cdot (yvec1@yvec2)) \sharp* ([(x, y)] \cdot P'')$› **have** $([(y, x)] \cdot p \cdot (yvec1@yvec2)) \sharp* ([(y, x)] \cdot [(x, y)] \cdot P'')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence $(((y, x)\#p) \cdot (yvec1@yvec2)) \sharp* P''$ **by**(*simp add*: *name-swap*)
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp P''$› *yvecEq*
have $bn(((y, x)\#p) \cdot \alpha) \sharp* P''$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *perm-compose calc-atm eqvts freshChainSimps*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp A_P{}'$› ‹$(p \cdot (yvec1@yvec2)) \sharp* A_P{}'$› **have** $(p \cdot (yvec1@x\#yvec2)) \sharp* A_P{}'$
by(*simp add*: *eqvts freshChainSimps*)
hence $([(y, x)] \cdot p \cdot (yvec1@x\#yvec2)) \sharp* ([(y, x)] \cdot A_P{}')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$x \sharp A_P{}'$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$y \sharp A_P{}'$› *yvecEq*
have $bn(((y, x)\#p) \cdot \alpha) \sharp* A_P{}'$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *perm-compose calc-atm eqvts freshChainSimps*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp C'$› ‹$(p \cdot (yvec1@yvec2)) \sharp* C'$› **have** $(p \cdot (yvec1@x\#yvec2)) \sharp* C'$
by(*simp add*: *eqvts freshChainSimps*)
hence $([(y, x)] \cdot p \cdot (yvec1@x\#yvec2)) \sharp* ([(y, x)] \cdot C')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$x \sharp C'$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$y \sharp C'$› *yvecEq*
have $bn(((y, x)\#p) \cdot \alpha) \sharp* C'$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *perm-compose calc-atm eqvts freshChainSimps*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp X$› ‹$(p \cdot (yvec1@yvec2)) \sharp* X$› **have** $(p \cdot (yvec1@x\#yvec2)) \sharp* X$
by(*simp add*: *eqvts freshChainSimps*)
hence $([(y, x)] \cdot p \cdot (yvec1@x\#yvec2)) \sharp* ([(y, x)] \cdot X)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$x \sharp X$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$bn\ \alpha \sharp* X$› *yvecEq*
have $bn(((y, x)\#p) \cdot \alpha) \sharp* X$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *perm-compose calc-atm eqvts freshChainSimps*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp Y$› ‹$(p \cdot (yvec1@yvec2)) \sharp* Y$› **have** $(p \cdot (yvec1@x\#yvec2)) \sharp* Y$
by(*simp add*: *eqvts freshChainSimps*)
hence $([(y, x)] \cdot p \cdot (yvec1@x\#yvec2)) \sharp* ([(y, x)] \cdot Y)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$x \sharp Y$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$bn\ \alpha \sharp* Y$› *yvecEq*

have $bn(((y, x)\#p) \cdot \alpha) \sharp* Y$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *perm-compose calc-atm eqvts freshChainSimps*)
moreover from ‹$x \sharp p$› ‹$y \sharp p$› ‹$x \sharp Z$› ‹$(p \cdot (yvec1@yvec2)) \sharp* Z$› **have** $(p \cdot (yvec1@x\#yvec2)) \sharp* Z$
by(*simp add*: *eqvts freshChainSimps*)
hence $([(y, x)] \cdot p \cdot (yvec1@x\#yvec2)) \sharp* ([(y, x)] \cdot Z)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \neq y$› ‹$x \sharp Z$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$› ‹$x \sharp p$› ‹$y \sharp p$› ‹$bn\ \alpha \sharp* Z$› *yvecEq*
have $bn(((y, x)\#p) \cdot \alpha) \sharp* Z$ **by**(*simp add*: *bnEqvt*[*symmetric*]) (*simp add*: *perm-compose calc-atm eqvts freshChainSimps*)
moreover from ‹$(yvec1@yvec2) \sharp* A_P{}'$› ‹$y \sharp A_P{}'$› *yvecEq* **have** $bn\ \alpha \sharp* A_P{}'$
by *simp*
moreover from ‹$A_P{}' \sharp* ([(x, y)] \cdot N')$› **have** $([(x, y)] \cdot A_P{}') \sharp* ([(x, y)] \cdot [(x, y)] \cdot N')$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_P{}'$› ‹$y \sharp A_P{}'$› **have** $A_P{}' \sharp* N'$ **by** *simp*
with ‹$A_P{}' \sharp* M$› ‹$(yvec1@yvec2) \sharp* A_P{}'$› ‹$y \sharp A_P{}'$› αeq **have** $A_P{}' \sharp* \alpha$ **by** *simp*
moreover hence $(((y, x)\#p) \cdot A_P{}') \sharp* (((y, x)\#p) \cdot \alpha)$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_P{}'$› ‹$y \sharp A_P{}'$› S ‹$(yvec1@yvec2) \sharp* A_P{}'$› ‹$(p \cdot (yvec1@yvec2)) \sharp* A_P{}'$›
have $A_P{}' \sharp* (((y, x)\#p) \cdot \alpha)$ **by**(*simp add*: *eqvts*)
moreover from ‹$A_P{}' \sharp* ([(x, y)] \cdot P'')$› **have** $([(x, y)] \cdot A_P{}') \sharp* ([(x, y)] \cdot [(x, y)] \cdot P'')$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_P{}'$› ‹$y \sharp A_P{}'$› **have** $A_P{}' \sharp* P''$ **by** *simp*
moreover from *yvecEq* αeq ‹$(p \cdot (yvec1@yvec2)) \sharp* (yvec1@yvec2)$› ‹$y \sharp p$› ‹$x \sharp \alpha$› S ‹$(p \cdot (yvec1@yvec2)) \sharp* M$›‹$(p \cdot (yvec1@yvec2)) \sharp* ([(x, y)] \cdot N')$› ‹$y \sharp yvec1$›‹$y \sharp yvec2$› ‹$x \sharp p$›
have $bn(((y, x)\#p) \cdot \alpha) \sharp* \alpha$
apply(*simp add*: *eqvts del*: *set-append*)
apply(*intro conjI*)
apply(*simp add*: *perm-compose eqvts del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*simp add*: *perm-compose eqvts del*: *set-append*)
apply(*simp add*: *perm-compose eqvts swapStarFresh del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *swapStarFresh calc-atm eqvts del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*subst pt-fresh-star-bij*[*symmetric*, *OF pt-name-inst*, *OF at-name-inst*, **where** *pi*=[(*x*, *y*)]])
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
apply(*subst pt-fresh-star-bij*[*symmetric*, *OF pt-name-inst*, *OF at-name-inst*,

where *pi*=[(*x*, *y*)]])
by(*simp add*: *perm-compose freshChainSimps*(*6*) *calc-atm eqvts del*: *set-append*)
moreover note ‹$A_P{}' \sharp* C$› ‹$A_P{}' \sharp* X$› ‹$A_P{}' \sharp* Y$› ‹$A_P{}' \sharp* Z$› ‹*distinct* $A_P{}'$›

ultimately show *?case*
by(*rule-tac cOpen*)
next
case(*cScope* Ψ P α P' x A_P Ψ_P C C' α' P'' X Y Z)
from ‹$\alpha \prec (\!|\nu x|\!)P' = \alpha' \prec P''$› ‹$x \sharp \alpha$› ‹$x \sharp \alpha'$›
obtain P''' **where** $\alpha \prec P' = \alpha' \prec P'''$ **and** $P'' = (\!|\nu x|\!)P'''$
apply(*cases rule*: *actionCases*[**where** α=α])
apply(*auto simp add*: *residualInject*)
by(*metis boundOutputScopeDest*)
then obtain p Ψ' $A_P{}'$ $\Psi_P{}'$ **where** S: *set* $p \subseteq$ *set*(*bn* α') $\times$ *set*(*bn*($p \cdot \alpha'$)) **and** *PeqP'*: $((p \cdot \Psi_P) \otimes \Psi') \simeq \Psi_P{}'$
and *distinctPerm* p **and** *bn*($p \cdot \alpha'$) $\sharp* C'$ **and** *FrP'*: *extractFrame* $P''' = \langle A_P{}', \Psi_P{}'\rangle$
and $A_P{}' \sharp* P'''$ **and** $A_P{}' \sharp* \alpha'$ **and** $A_P{}' \sharp* (p \cdot \alpha')$ **and** $A_P{}' \sharp* C$ **and** *distinct* $A_P{}'$
and *bn*($p \cdot \alpha'$) $\sharp* P'''$ **and** $A_P{}' \sharp* (x\#X)$ **and** $A_P{}' \sharp* Y$ **and** *bn*($p \cdot \alpha'$) $\sharp* \alpha'$
and $A_P{}' \sharp* Z$ **and** *bn*($p \cdot \alpha'$) $\sharp* (x\#X)$ **and** *bn*($p \cdot \alpha'$) $\sharp* Y$
and *bn*($p \cdot \alpha'$) $\sharp* Z$ **using** *cScope*
by(*rule-tac cScope*) (*assumption* | *simp*)+
from ‹$A_P{}' \sharp* (x\#X)$› **have** $x \sharp A_P{}'$ **and** $A_P{}' \sharp* X$ **by** *simp*+
from ‹*bn*($p \cdot \alpha'$) $\sharp* (x\#X)$› **have** $x \sharp$ *bn*($p \cdot \alpha'$) **and** *bn*($p \cdot \alpha'$) $\sharp* X$ **by** *simp*+

note S *PeqP'* ‹*distinctPerm* p› ‹*bn*($p \cdot \alpha'$) $\sharp* C'$›
moreover from *FrP'* ‹$P'' = (\!|\nu x|\!)P'''$› **have** *extractFrame* $P'' = \langle (x\#A_P{}'), \Psi_P{}'\rangle$ **by** *simp*
moreover from ‹$A_P{}' \sharp* P'''$› ‹$P'' = (\!|\nu x|\!)P'''$› ‹$x \sharp A_P{}'$› **have** $(x\#A_P{}') \sharp* P''$ **by**(*simp add*: *abs-fresh*)
moreover from ‹$A_P{}' \sharp* \alpha'$› ‹$A_P{}' \sharp* C$› ‹$x \sharp \alpha'$› ‹$x \sharp C$› **have** $(x\#A_P{}') \sharp* \alpha'$ **and** $(x\#A_P{}') \sharp* C$ **by** *simp*+
moreover note ‹*bn*($p \cdot \alpha'$) $\sharp* \alpha'$›
moreover from ‹*bn*($p \cdot \alpha'$) $\sharp* P'''$› ‹$P'' = (\!|\nu x|\!)P'''$› ‹$x \sharp$ *bn*($p \cdot \alpha'$)› **have** *bn*($p \cdot \alpha'$) $\sharp* P''$ **by** *simp*
moreover from ‹$A_P{}' \sharp* \alpha'$› ‹$x \sharp \alpha'$› **have** $(x\#A_P{}') \sharp* \alpha'$ **by** *simp*
moreover from ‹$A_P{}' \sharp* (p \cdot \alpha')$› ‹$x \sharp \alpha'$› S ‹$x \sharp$ *bn*($p \cdot \alpha'$)› **have** $(x\#A_P{}') \sharp* (p \cdot \alpha')$
by(*simp add*: *subjectEqvt*[*symmetric*] *bnEqvt*[*symmetric*] *objectEqvt*[*symmetric*] *freshChainSimps*)
moreover from ‹$A_P{}' \sharp* X$› ‹$x \sharp X$› **have** $(x\#A_P{}') \sharp* X$ **by** *simp*+
moreover from ‹$A_P{}' \sharp* Y$› ‹$x \sharp Y$› **have** $(x\#A_P{}') \sharp* Y$ **by** *simp*+
moreover from ‹$A_P{}' \sharp* Z$› ‹$x \sharp Z$› **have** $(x\#A_P{}') \sharp* Z$ **by** *simp*+
moreover note ‹*bn*($p \cdot \alpha'$) $\sharp* X$› ‹*bn*($p \cdot \alpha'$) $\sharp* Y$› ‹*bn*($p \cdot \alpha'$) $\sharp* Z$›
moreover from ‹*distinct* $A_P{}'$› ‹$x \sharp A_P{}'$› **have** *distinct*$(x\#A_P{}')$ **by** *simp*

ultimately show *?case* **by**(*rule-tac cScope*)
next
case(*cBang* Ψ P A_P Ψ_P C C' α P' X Y Z)
then obtain p Ψ' $A_P{}'$ $\Psi_P{}'$ **where** S: *set* $p \subseteq$ *set*(*bn* α) $\times$ *set*(*bn*($p \cdot \alpha$))
and *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$
and *PeqP'*: $(p \cdot (\Psi_P \otimes \mathbf{1})) \otimes \Psi' \simeq \Psi_P{}'$
and $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* \alpha$ **and** $A_P{}' \sharp* (p \cdot \alpha)$
and $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$ **and** *distinct* $A_P{}'$
and *distinctPerm* p **and** (*bn*($p \cdot \alpha$)) $\sharp* \alpha$ **and** (*bn*($p \cdot \alpha$)) $\sharp* P'$
and (*bn*($p \cdot \alpha$)) $\sharp* C'$ **and** (*bn*($p \cdot \alpha$)) $\sharp* X$ **and** (*bn*($p \cdot \alpha$)) $\sharp* Y$ **and** (*bn*($p \cdot \alpha$)) $\sharp* Z$
by(*rule-tac cBang*) (*assumption* | *simp* (*no-asm-use*))+
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $(p \cdot \Psi_P) \simeq (p \cdot \mathbf{1})$
by(*simp add*: *AssertionStatEqClosed*)
hence $(p \cdot \Psi_P) \simeq \mathbf{1}$ **by**(*simp add*: *permBottom*)
with *PeqP'* **have** $(\mathbf{1} \otimes \Psi') \simeq \Psi_P{}'$
by(*simp add*: *eqvts permBottom*) (*metis Identity AssertionStatEqTrans composition' Commutativity Associativity AssertionStatEqSym*)
ultimately show *?case* **using** *cBang*
apply(*rule-tac cBang*(*18*)) **by**(*assumption* | *simp*)+
qed

with A **have** *?thesis* **by** *blast*
}
moreover have *bn* α $\sharp*$ ([]::*name list*) **and** *bn* α $\sharp*$ ([]::*'b list*) **and** *bn* α $\sharp*$ ([]::(*'a*, *'b*, *'c*) *psi list*)
and A_P $\sharp*$ ([]::*name list*) **and** A_P $\sharp*$ ([]::*'b list*) **and** A_P $\sharp*$ ([]::(*'a*, *'b*, *'c*) *psi list*)
by *simp*+
ultimately show *?thesis* **by** *blast*
qed

lemma *expandTauFrame*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*,*'c*) *psi*
and P' :: (*'a*, *'b*, *'c*) *psi*
and A_P :: *name list*
and Ψ_P :: *'b*
and C :: *'d*::*fs-name*

assumes $\Psi \rhd P \longmapsto\tau \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and $A_P \sharp* P$
and $A_P \sharp* C$

obtains Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$ **and** $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* P'$ **and** *distinct* $A_P{}'$
proof −
assume A: $\bigwedge A_P{}'\ \Psi_P{}'\ \Psi'.$
$[\![$*extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$; $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$; $A_P{}' \sharp* C$; $A_P{}' \sharp* P'$; *distinct* $A_P{}'$$]\!]$
$\Longrightarrow$ *thesis*

from ‹$\Psi \rhd P \longmapsto\tau \prec P'$› ‹$A_P \sharp* P$› **have** $A_P \sharp* P'$ **by**(*rule tauFreshChainDerivative*)

{
fix X :: *name list*
and Y :: $'b$ *list*
and Z :: $('a, 'b, 'c)$ *psi list*

assume $A_P \sharp* X$
and $A_P \sharp* Y$
and $A_P \sharp* Z$

with *assms* ‹$A_P \sharp* P'$› **obtain** Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$ **and** $A_P{}' \sharp* C$
and $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$ **and** *distinct* $A_P{}'$
proof(*nominal-induct avoiding*: $C\ X\ Y\ Z$ *arbitrary*: *thesis rule*: *tauFrameInduct*)
case(*cAlpha* $\Psi\ P\ P'\ A_P\ \Psi_P\ p\ C\ X\ Y\ Z$)
then obtain Ψ' $A_P{}'$ $\Psi_P{}'$ **where** FrP': *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$ **and** *distinct* $A_P{}'$
and $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$
by *metis*

have S: *set* $p \subseteq$ *set* $A_P \times$ *set*$(p \cdot A_P)$ **by** *fact*

from FrP' **have** $(p \cdot$ *extractFrame* $P') = p \cdot \langle A_P{}', \Psi_P{}'\rangle$ **by** *simp*
with ‹$A_P \sharp* P'$› ‹$(p \cdot A_P) \sharp* P'$› S **have** *extractFrame* $P' = \langle (p \cdot A_P{}'), (p \cdot \Psi_P{}')\rangle$ **by**(*simp add*: *eqvts*)
moreover from ‹$\Psi_P \otimes \Psi' \simeq \Psi_P{}'$› **have** $(p \cdot (\Psi_P \otimes \Psi')) \simeq (p \cdot \Psi_P{}')$ **by**(*rule AssertionStatEqClosed*)
hence $(p \cdot \Psi_P) \otimes (p \cdot \Psi') \simeq (p \cdot \Psi_P{}')$ **by**(*simp add*: *eqvts*)
moreover from ‹$A_P{}' \sharp* C$› **have** $(p \cdot A_P{}') \sharp* (p \cdot C)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* C$› ‹$(p \cdot A_P) \sharp* C$› S **have** $(p \cdot A_P{}') \sharp* C$ **by** *simp*
moreover from ‹$A_P{}' \sharp* P'$› **have** $(p \cdot A_P{}') \sharp* (p \cdot P')$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* P'$› ‹$(p \cdot A_P) \sharp* P'$› S **have** $(p \cdot A_P{}') \sharp* P'$ **by** *simp*
moreover from ‹$A_P{}' \sharp* X$› **have** $(p \cdot A_P{}') \sharp* (p \cdot X)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* X$› ‹$(p \cdot A_P) \sharp* X$› S **have** $(p \cdot A_P{}') \sharp* X$ **by** *simp*

moreover from ‹$A_P{}' \sharp* Y$› **have** $(p \cdot A_P{}') \sharp* (p \cdot Y)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* Y$› ‹$(p \cdot A_P) \sharp* Y$› S **have** $(p \cdot A_P{}') \sharp* Y$ **by** *simp*
moreover from ‹$A_P{}' \sharp* Z$› **have** $(p \cdot A_P{}') \sharp* (p \cdot Z)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* Z$› ‹$(p \cdot A_P) \sharp* Z$› S **have** $(p \cdot A_P{}') \sharp* Z$ **by** *simp*
moreover from ‹*distinct* $A_P{}'$› **have** *distinct*$(p \cdot A_P{}')$ **by** *simp*
ultimately show *?case* **by**(*rule cAlpha*)
next
case(*cCase* Ψ P P' φ *Cs* A_P Ψ_P C B Y Z *thesis*)
then obtain Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$
and $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$ **and** *distinct* $A_P{}'$
and $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* P'$
and $A_P{}' \sharp* B$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* Z$
by(*rule-tac cCase*) (*assumption* | *simp* (*no-asm-use*))+
with ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi_P \otimes \Psi' \simeq \Psi_P{}'$› **have** $\mathbf{1} \otimes \Psi' \simeq \Psi_P{}'$
by(*metis Identity AssertionStatEqTrans composition' Commutativity Associativity AssertionStatEqSym*)
thus *?case* **using** *FrP'* ‹$A_P{}' \sharp* P'$› ‹$A_P{}' \sharp* C$› ‹$A_P{}' \sharp* B$› ‹$A_P{}' \sharp* Y$› ‹$A_P{}' \sharp* Z$› ‹*distinct* $A_P{}'$› **using** *cCase*
by *force*
next
case(*cPar1* Ψ Ψ_Q P P' A_Q Q A_P Ψ_P C X Y Z)
moreover from ‹$A_P \sharp* X$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* Y$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* Z$›
have $A_P \sharp* (X@A_Q)$ **and** $A_P \sharp* (\Psi_Q\#Y)$ **and** $A_P \sharp* (Q\#Z)$
by *simp*+
ultimately obtain Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** *distinct* $A_P{}'$
and $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$ **and** $A_P \sharp* P$ **and** $A_P \sharp* C$ **and** $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* P'$
and $A_P{}' \sharp* (X@A_Q)$ **and** $A_P{}' \sharp* (\Psi_Q\#Y)$ **and** $A_P{}' \sharp* (Q\#Z)$
by *metis*

hence $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* A_Q$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* \Psi_Q$ **and** $A_P{}' \sharp* Q$ **and** $A_P{}' \sharp* Z$
by *simp*+

from ‹$A_P{}' \sharp* A_Q$› ‹$A_Q \sharp* P'$› *FrP'* **have** $A_Q \sharp* \Psi_P{}'$ **by**(*force dest*: *extractFrameFreshChain*)
with ‹$A_P{}' \sharp* \Psi_Q$› ‹$A_P{}' \sharp* A_Q$› ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› *FrP'*
have *extractFrame*$(P' \parallel Q) = \langle (A_P{}'@A_Q), \Psi_P{}' \otimes \Psi_Q\rangle$ **by** *simp*

moreover from ‹$\Psi_P \otimes \Psi' \simeq \Psi_P{}'$›**have** $(\Psi_P \otimes \Psi_Q) \otimes \Psi' \simeq \Psi_P{}' \otimes \Psi_Q$
by(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)

moreover from ‹$A_P{}' \sharp* C$› ‹$A_Q \sharp* C$› **have** $(A_P{}'@A_Q) \sharp* C$ **by** *simp*

moreover from ‹$A_P{}' \sharp* P'$› ‹$A_Q \sharp* P'$› ‹$A_P{}' \sharp* Q$› ‹$A_Q \sharp* Q$› **have** $(A_P{}'@A_Q) \sharp* (P' \parallel Q)$ **by** *simp*
moreover from ‹$A_P{}' \sharp* X$› ‹$A_Q \sharp* X$› **have** $(A_P{}'@A_Q) \sharp* X$ **by** *simp*
moreover from ‹$A_P{}' \sharp* Y$› ‹$A_Q \sharp* Y$› **have** $(A_P{}'@A_Q) \sharp* Y$ **by** *simp*
moreover from ‹$A_P{}' \sharp* Z$› ‹$A_Q \sharp* Z$› **have** $(A_P{}'@A_Q) \sharp* Z$ **by** *simp*
moreover from ‹$A_P{}' \sharp* A_Q$› ‹*distinct* $A_P{}'$› ‹*distinct* A_Q› **have** *distinct*$(A_P{}'@A_Q)$ **by** *simp*
ultimately show *?case* **by**(*rule cPar1*)
next
case(*cPar2* Ψ Ψ_P Q Q' A_P P A_Q Ψ_Q C X Y Z)
moreover from ‹$A_Q \sharp* X$› ‹$A_P \sharp* A_Q$› ‹$A_Q \sharp* Y$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Z$›
have $A_Q \sharp* (X@A_P)$ **and** $A_Q \sharp* (\Psi_P\#Y)$ **and** $A_Q \sharp* (P\#Z)$
by(*simp add*: *freshChainSimps*)+
ultimately obtain Ψ' $A_Q{}'$ $\Psi_Q{}'$ **where** *FrQ'*: *extractFrame* $Q' = \langle A_Q{}', \Psi_Q{}'\rangle$ **and** *distinct* $A_Q{}'$
and $\Psi_Q \otimes \Psi' \simeq \Psi_Q{}'$**and** $A_Q{}' \sharp* C$ **and** $A_Q{}' \sharp* Q'$
and $A_Q{}' \sharp* (X@A_P)$ **and** $A_Q{}' \sharp* (\Psi_P\#Y)$ **and** $A_Q{}' \sharp* (P\#Z)$
by *metis*

hence $A_Q{}' \sharp* X$ **and** $A_Q{}' \sharp* A_P$ **and** $A_Q{}' \sharp* Y$ **and** $A_Q{}' \sharp* \Psi_P$ **and** $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* Z$
by *simp*+

from ‹$A_Q{}' \sharp* A_P$› ‹$A_P \sharp* Q'$› *FrQ'* **have** $A_P \sharp* \Psi_Q{}'$ **by**(*force dest*: *extractFrameFreshChain*)
with ‹$A_Q{}' \sharp* \Psi_P$› ‹$A_Q{}' \sharp* A_P$› ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› *FrQ'*
have *extractFrame*$(P \parallel Q') = \langle (A_P@A_Q{}'), \Psi_P \otimes \Psi_Q{}'\rangle$ **by** *simp*

moreover from ‹$\Psi_Q \otimes \Psi' \simeq \Psi_Q{}'$›**have** $(\Psi_P \otimes \Psi_Q) \otimes \Psi' \simeq \Psi_P \otimes \Psi_Q{}'$
by(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)
moreover from ‹$A_P \sharp* C$› ‹$A_Q{}' \sharp* C$› **have** $(A_P@A_Q{}') \sharp* C$ **by** *simp*
moreover from ‹$A_P \sharp* P$› ‹$A_Q{}' \sharp* P$› ‹$A_P \sharp* Q'$› ‹$A_Q{}' \sharp* Q'$› **have** $(A_P@A_Q{}') \sharp* (P \parallel Q')$ **by** *simp*
moreover from ‹$A_P \sharp* X$› ‹$A_Q{}' \sharp* X$› **have** $(A_P@A_Q{}') \sharp* X$ **by** *simp*
moreover from ‹$A_P \sharp* Y$› ‹$A_Q{}' \sharp* Y$› **have** $(A_P@A_Q{}') \sharp* Y$ **by** *simp*
moreover from ‹$A_P \sharp* Z$› ‹$A_Q{}' \sharp* Z$› **have** $(A_P@A_Q{}') \sharp* Z$ **by** *simp*
moreover from ‹$A_Q{}' \sharp* A_P$› ‹*distinct* A_P› ‹*distinct* $A_Q{}'$› **have** *distinct*$(A_P@A_Q{}')$ **by** *simp*
ultimately show *?case* **by**(*rule cPar2*)
next
case(*cComm1* Ψ Ψ_Q P M N P' A_P Ψ_P Q K *xvec* Q' A_Q C X Y Z)
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$ **and** *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **by** *fact*+
from *PTrans* ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹*distinct* A_P› ‹$A_P \sharp* P$› ‹$A_P \sharp* N$› ‹$A_P \sharp* M$›
‹$A_P \sharp* Q'$› ‹$A_P \sharp* C$› ‹$A_P \sharp* X$› ‹$A_P \sharp* Y$› ‹$A_P \sharp* Z$› ‹$A_P \sharp* A_Q$› ‹A_P

$\sharp*$ *xvec*›

obtain Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** *PeqP'*: $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$

and $A_P{}' \sharp* Q'$ **and** $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** *distinct* $A_P{}'$

and $A_P{}' \sharp* Z$ **and** $A_P{}' \sharp* A_Q$ **and** $A_P{}' \sharp*$ *xvec* **and** $A_P{}' \sharp* P'$

by(*rule-tac* C=(Q', C, X, Y, Z, A_Q, *xvec*) **and** C'=(Q', C, X, Y, Z, A_Q, *xvec*) **in** *expandNonTauFrame*) *auto*

moreover from *QTrans* **have** *distinct xvec* **by**(*auto dest*: *boundOutputDistinct*)

from *QTrans* ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹*distinct* A_Q› ‹$A_Q \sharp* P$› ‹$A_Q \sharp*$ *xvec*› ‹$A_Q \sharp* K$› ‹*xvec* $\sharp* K$› ‹*distinct xvec*›

‹$A_P{}' \sharp* A_Q$› ‹$A_P{}' \sharp*$ *xvec*› ‹$A_Q \sharp* Q$› ‹*xvec* $\sharp* Q$› ‹$A_Q \sharp* \Psi_P$› ‹*xvec* $\sharp* \Psi_P$› ‹$A_Q \sharp* N$›

‹$A_Q \sharp* C$› ‹$A_Q \sharp* X$› ‹$A_Q \sharp* Y$› ‹$A_Q \sharp* Z$› ‹*xvec* $\sharp* P$› ‹*xvec* $\sharp* C$› ‹*xvec* $\sharp* X$› ‹*xvec* $\sharp* Y$› ‹*xvec* $\sharp* Z$›

obtain p Ψ'' $A_Q{}'$ $\Psi_Q{}'$ **where** S: *set* $p \subseteq$ *set xvec* $\times$ *set*($p \cdot$ *xvec*) **and** *QeqQ'*: $(p \cdot \Psi_Q) \otimes \Psi'' \simeq \Psi_Q{}'$ **and** *FrQ'*: *extractFrame* $Q' = \langle A_Q{}', \Psi_Q{}'\rangle$

and $A_Q{}' \sharp* A_P{}'$ **and** $A_Q{}' \sharp* C$ **and** $A_Q{}' \sharp* X$ **and** $A_Q{}' \sharp* Y$ **and** $A_Q{}' \sharp* Z$ **and** $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* N$ **and** *distinct* $A_Q{}'$

and $(p \cdot$ *xvec*$) \sharp* A_P{}'$ **and** $(p \cdot$ *xvec*$) \sharp* C$ **and** $(p \cdot$ *xvec*$)$ $\sharp* X$ **and** $(p \cdot$ *xvec*$) \sharp* Y$ **and** $(p \cdot$ *xvec*$) \sharp* P$

and $(p \cdot$ *xvec*$) \sharp* Z$ **and** $(p \cdot$ *xvec*$) \sharp* N$ **and** $(p \cdot$ *xvec*$) \sharp* \Psi_P$ **and** $(p \cdot$ *xvec*$) \sharp* A_Q{}'$**and** $(p \cdot$ *xvec*$) \sharp* Q'$

and *distinctPerm* p **and** $A_Q{}' \sharp*$ *xvec* **and** $A_Q{}' \sharp* Q'$

by(*rule-tac* C=(P, C, X, Y, Z, $A_P{}'$, Ψ_P) **and** C'=(P, C, X, Y, Z, $A_P{}'$, Ψ_P) **in** *expandNonTauFrame*) (*assumption* | *simp*)+

from *PTrans* ‹$A_Q{}' \sharp* P$› ‹$A_Q{}' \sharp* N$› ‹$(p \cdot$ *xvec*$) \sharp* P$› ‹$(p \cdot$ *xvec*$) \sharp* N$›

have $A_Q{}' \sharp* P'$ **and** $(p \cdot$ *xvec*$) \sharp* P'$ **by**(*force dest*: *inputFreshChainDerivative*)+

with *FrP'* ‹$A_Q{}' \sharp* A_P{}'$› ‹$(p \cdot$ *xvec*$) \sharp* A_P{}'$› **have** $A_Q{}' \sharp* \Psi_P{}'$ **and** $(p \cdot$ *xvec*$)$ $\sharp* \Psi_P{}'$ **by**(*force dest*: *extractFrameFreshChain*)+

from *FrQ'* ‹$A_Q{}' \sharp* A_P{}'$› ‹$A_P{}' \sharp* Q'$› ‹$(p \cdot$ *xvec*$) \sharp* A_Q{}'$› ‹$(p \cdot$ *xvec*$) \sharp* Q'$› **have** $A_P{}' \sharp* \Psi_Q{}'$ **and** $(p \cdot$ *xvec*$) \sharp* \Psi_Q{}'$

by(*force dest*: *extractFrameFreshChain*)+

have *extractFrame*$((\!|\nu*xvec|\!)(P' \parallel Q')) = \langle((p \cdot$ *xvec*$)@A_P{}'@A_Q{}'), (p \cdot \Psi_P{}') \otimes (p \cdot \Psi_Q{}')\rangle$

proof −

from *FrP'* *FrQ'* ‹$A_P{}' \sharp* \Psi_Q{}'$› ‹$A_Q{}' \sharp* A_P{}'$› ‹$A_Q{}' \sharp* \Psi_P{}'$› **have** *extractFrame*$(P' \parallel Q') = \langle(A_P{}'@A_Q{}'), \Psi_P{}' \otimes \Psi_Q{}'\rangle$

by *simp*

hence *extractFrame*$((\!|\nu*xvec|\!)(P' \parallel Q')) = \langle(xvec@A_P{}'@A_Q{}'), \Psi_P{}' \otimes \Psi_Q{}'\rangle$

by(*induct xvec*) *auto*

moreover from ‹$(p \cdot$ *xvec*$) \sharp* \Psi_P{}'$› ‹$(p \cdot$ *xvec*$) \sharp* \Psi_Q{}'$› S

have $(\!|\nu*xvec|\!)((\!|\nu*(A_P{}'@A_Q{}')|\!)(FAssert\ (\Psi_P{}' \otimes \Psi_Q{}'))) = (\!|\nu*(p \cdot$ *xvec*$)|\!)(p \cdot (\!|\nu*(A_P{}'@A_Q{}')|\!)(FAssert(\Psi_P{}' \otimes \Psi_Q{}')))$

by(*rule-tac frameChainAlpha*) (*auto simp add*: *fresh-star-def frameResChainFresh*)
hence $(\!|\nu*xvec|\!)((\!|\nu*(A_P'@A_Q')|\!)(FAssert\ (\Psi_P' \otimes \Psi_Q'))) = (\!|\nu*(p \cdot xvec)|\!)((\!|\nu*(A_P'@A_Q')|\!)(FAssert((p \cdot \Psi_P') \otimes (p \cdot \Psi_Q'))))$
using ‹$A_P' \sharp* xvec$› ‹$(p \cdot xvec) \sharp* A_P'$› ‹$A_Q' \sharp* xvec$› ‹$(p \cdot xvec) \sharp* A_Q'$› S
by(*fastforce simp add*: *eqvts*)
ultimately show *?thesis*
by(*simp add*: *frameChainAppend*)
qed

moreover have $(\Psi_P \otimes \Psi_Q) \otimes ((p \cdot \Psi') \otimes (p \cdot \Psi'')) \simeq (p \cdot \Psi_P') \otimes (p \cdot \Psi_Q')$
proof −
have $(\Psi_P \otimes (p \cdot \Psi_Q)) \otimes (\Psi' \otimes \Psi'') \simeq (\Psi_P \otimes \Psi') \otimes ((p \cdot \Psi_Q) \otimes \Psi'')$
by(*metis Associativity Commutativity Composition AssertionStatEqTrans*)
moreover from *PeqP' QeqQ'* **have** $(\Psi_P \otimes \Psi') \otimes ((p \cdot \Psi_Q) \otimes \Psi'') \simeq \Psi_P' \otimes \Psi_Q'$
by(*metis Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately have $(\Psi_P \otimes (p \cdot \Psi_Q)) \otimes (\Psi' \otimes \Psi'') \simeq \Psi_P' \otimes \Psi_Q'$
by(*metis AssertionStatEqTrans*)
hence $(p \cdot ((\Psi_P \otimes (p \cdot \Psi_Q)) \otimes (\Psi' \otimes \Psi''))) \simeq (p \cdot (\Psi_P' \otimes \Psi_Q'))$
by(*rule AssertionStatEqClosed*)
with ‹$xvec \sharp* \Psi_P$› ‹$(p \cdot xvec) \sharp* \Psi_P$› S ‹*distinctPerm p*› **show** *?thesis*
by(*simp add*: *eqvts*)
qed

moreover from ‹$(p \cdot xvec) \sharp* C$› ‹$A_P' \sharp* C$› ‹$A_Q' \sharp* C$› **have** $((p \cdot xvec)@A_P'@A_Q') \sharp* C$ **by** *simp*
moreover from ‹$(p \cdot xvec) \sharp* X$› ‹$A_P' \sharp* X$› ‹$A_Q' \sharp* X$› **have** $((p \cdot xvec)@A_P'@A_Q') \sharp* X$ **by** *simp*
moreover from ‹$(p \cdot xvec) \sharp* Y$› ‹$A_P' \sharp* Y$› ‹$A_Q' \sharp* Y$› **have** $((p \cdot xvec)@A_P'@A_Q') \sharp* Y$ **by** *simp*
moreover from ‹$(p \cdot xvec) \sharp* Z$› ‹$A_P' \sharp* Z$› ‹$A_Q' \sharp* Z$› **have** $((p \cdot xvec)@A_P'@A_Q') \sharp* Z$ **by** *simp*
moreover from ‹$(p \cdot xvec) \sharp* P'$› ‹$(p \cdot xvec) \sharp* Q'$› ‹$A_P' \sharp* P'$› ‹$A_P' \sharp* Q'$› ‹$A_Q' \sharp* P'$› ‹$A_Q' \sharp* Q'$›
have $((p \cdot xvec)@A_P'@A_Q') \sharp* ((\!|\nu*xvec|\!)(P' \parallel Q'))$ **by**(*auto simp add*: *resChainFresh fresh-star-def*)
moreover from ‹$(p \cdot xvec) \sharp* A_P'$› ‹$(p \cdot xvec) \sharp* A_Q'$› ‹$A_Q' \sharp* A_P'$› ‹*distinct xvec*› ‹*distinct* A_P'› ‹*distinct* A_Q'›
have $distinct((p \cdot xvec)@A_P'@A_Q')$
by *auto* (*simp add*: *name-list-supp fresh-star-def fresh-def*)+

ultimately show *?case* **using** *cComm1*
by *metis*
next
case(*cComm2* $\Psi\ \Psi_Q\ P\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ Q\ K\ Q'\ A_Q\ C\ X\ Y\ Z$)
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **and** *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$ **by** *fact*+
from *PTrans* **have** *distinct xvec* **by**(*auto dest*: *boundOutputDistinct*)

from *PTrans* ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› ‹*distinct* A_P› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹*xvec* $\sharp* Q$› ‹*distinct xvec*› ‹*xvec* $\sharp* M$›
‹$A_P \sharp* C$› ‹$A_P \sharp* X$› ‹$A_P \sharp* Y$› ‹$A_P \sharp* Z$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp*$ *xvec*› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* M$› ‹$A_P \sharp* N$›
‹*xvec* $\sharp* P$› ‹*xvec* $\sharp* C$› ‹*xvec* $\sharp* X$› ‹*xvec* $\sharp* Y$› ‹*xvec* $\sharp* Z$› ‹$A_Q \sharp*$ *xvec*› ‹*xvec* $\sharp* \Psi_Q$›
obtain p Ψ' $A_P{}'$ $\Psi_P{}'$ **where** S: *set* $p \subseteq$ *set xvec* $\times$ *set*$(p \cdot xvec)$
and *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}' \rangle$ **and** *PeqP'*: $(p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P{}'$ **and** *distinct* $A_P{}'$
and $A_P{}' \sharp* C$ **and** $A_P{}' \sharp* X$ **and** $A_P{}' \sharp* Y$ **and** $A_P{}' \sharp* N$ **and** $A_P{}' \sharp* Q$ **and** $(p \cdot xvec) \sharp* Q$
and $A_P{}' \sharp* Z$ **and** $A_P{}' \sharp* A_Q$ **and** $A_P{}' \sharp*$ *xvec* **and** $A_P{}' \sharp* P'$ **and** $(p \cdot xvec) \sharp* N$ **and** $(p \cdot xvec) \sharp* \Psi_Q$
and $(p \cdot xvec) \sharp* A_P{}'$ **and** $(p \cdot xvec) \sharp* C$ **and** $(p \cdot xvec) \sharp* X$ **and** $(p \cdot xvec) \sharp* A_Q$
and $(p \cdot xvec) \sharp* Y$ **and** $(p \cdot xvec) \sharp* Z$ **and** $(p \cdot xvec) \sharp* P'$ **and** *distinctPerm p*
by(*rule-tac* C=(C, X, Y, Z, A_Q, Q, Ψ_Q) **and** C'=(C, X, Y, Z, A_Q, Q, Ψ_Q) **in** *expandNonTauFrame*) (*assumption* | *simp*)+

from *QTrans* ‹*extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$› ‹*distinct* A_Q› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp*$ *xvec*› ‹$A_Q \sharp* P'$› ‹$(p \cdot xvec) \sharp* A_Q$›
‹$A_P{}' \sharp* A_Q$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* C$› ‹$A_Q \sharp* X$› ‹$A_Q \sharp* Y$› ‹$A_Q \sharp* Z$› ‹$A_Q \sharp* N$› ‹$A_Q \sharp* K$›
obtain Ψ'' $A_Q{}'$ $\Psi_Q{}'$ **where** *QeqQ'*: $\Psi_Q \otimes \Psi'' \simeq \Psi_Q{}'$ **and** *FrQ'*: *extractFrame* $Q' = \langle A_Q{}', \Psi_Q{}' \rangle$ **and** *distinct* $A_Q{}'$
and $A_Q{}' \sharp*$ *xvec* **and** $A_Q{}' \sharp* Q'$ **and** $A_Q{}' \sharp*$ *xvec* **and** $A_Q{}' \sharp* P'$ **and** $A_Q{}' \sharp* (p \cdot xvec)$
and $A_Q{}' \sharp* A_P{}'$ **and** $A_Q{}' \sharp* C$ **and** $A_Q{}' \sharp* X$ **and** $A_Q{}' \sharp* Y$ **and** $A_Q{}' \sharp* Z$ **and** $A_Q{}' \sharp* P$
by(*rule-tac* C=(P, C, P', X, Y, Z, $A_P{}'$, *xvec*, $(p \cdot xvec)$, Ψ_P) **and** C'=(P, C, P', X, Y, Z, $A_P{}'$, *xvec*, $(p \cdot xvec)$, Ψ_P) **in** *expandNonTauFrame*) (*assumption* | *simp*)+

from *QTrans* ‹$A_P{}' \sharp* Q$› ‹$A_P{}' \sharp* N$› ‹$(p \cdot xvec) \sharp* Q$› ‹$(p \cdot xvec) \sharp* N$›
have $A_P{}' \sharp* Q'$ **and** $(p \cdot xvec) \sharp* Q'$ **by**(*force dest*: *inputFreshChainDerivative*)+
with *FrQ'* ‹$A_Q{}' \sharp* A_P{}'$› ‹$A_Q{}' \sharp* (p \cdot xvec)$› **have** $A_P{}' \sharp* \Psi_Q{}'$ **and** $(p \cdot xvec) \sharp* \Psi_Q{}'$ **by**(*force dest*: *extractFrameFreshChain*)+
from *FrP'* ‹$A_Q{}' \sharp* A_P{}'$› ‹$A_Q{}' \sharp* P'$› ‹$(p \cdot xvec) \sharp* A_P{}'$› ‹$(p \cdot xvec) \sharp* P'$› **have** $A_Q{}' \sharp* \Psi_P{}'$ **and** $(p \cdot xvec) \sharp* \Psi_P{}'$
by(*force dest*: *extractFrameFreshChain*)+

have *extractFrame*$((\!|\nu{*}xvec|\!)(P' \parallel Q')) = \langle ((p \cdot xvec)@A_P{}'@A_Q{}'), (p \cdot \Psi_P{}') \otimes (p \cdot \Psi_Q{}') \rangle$
proof −
from *FrP'* *FrQ'* ‹$A_P{}' \sharp* \Psi_Q{}'$› ‹$A_Q{}' \sharp* A_P{}'$› ‹$A_Q{}' \sharp* \Psi_P{}'$› **have** *extractFrame*$(P' \parallel Q') = \langle (A_P{}'@A_Q{}'), \Psi_P{}' \otimes \Psi_Q{}' \rangle$
by *simp*

hence *extractFrame*($\llparenthesis\nu*xvec\rrparenthesis$($P'$ ‖ Q')) = ⟨($xvec$@A_P'@A_Q'), $\Psi_P' \otimes \Psi_Q'$⟩
by(*induct xvec*) *auto*
moreover from ‹(p · *xvec*) ♯* Ψ_P'› ‹(p · *xvec*) ♯* Ψ_Q'› S
have $\llparenthesis\nu*xvec\rrparenthesis$($\llparenthesis\nu*(A_P'@A_Q')\rrparenthesis$(*FAssert* ($\Psi_P' \otimes \Psi_Q'$))) = $\llparenthesis\nu*(p \cdot xvec)\rrparenthesis$($p$ · $\llparenthesis\nu*(A_P'@A_Q')\rrparenthesis$(*FAssert*($\Psi_P' \otimes \Psi_Q'$)))
by(*rule-tac frameChainAlpha*) (*auto simp add*: *fresh-star-def frameResChainFresh*)
hence $\llparenthesis\nu*xvec\rrparenthesis$($\llparenthesis\nu*(A_P'@A_Q')\rrparenthesis$(*FAssert* ($\Psi_P' \otimes \Psi_Q'$))) = $\llparenthesis\nu*(p \cdot xvec)\rrparenthesis$($\llparenthesis\nu*(A_P'@A_Q')\rrparenthesis$(*FAssert*(($p$ · Ψ_P') $\otimes$ (p · Ψ_Q'))))
using ‹A_P' ♯* *xvec*› ‹(p · *xvec*) ♯* A_P'› ‹A_Q' ♯* *xvec*› ‹A_Q' ♯* (p · *xvec*)› S
by(*fastforce simp add*: *eqvts*)
ultimately show *?thesis*
by(*simp add*: *frameChainAppend*)
qed

moreover have ($\Psi_P \otimes \Psi_Q$) $\otimes$ ((p · Ψ') $\otimes$ (p · Ψ'')) $\simeq$ (p · Ψ_P') $\otimes$ (p · Ψ_Q')
proof −
have ((p · Ψ_P) $\otimes$ Ψ_Q) $\otimes$ ($\Psi' \otimes \Psi''$) $\simeq$ ((p · Ψ_P) $\otimes$ Ψ') $\otimes$ ($\Psi_Q \otimes \Psi''$)
by(*metis Associativity Commutativity Composition AssertionStatEqTrans*)
moreover from *PeqP' QeqQ'* **have** ((p · Ψ_P) $\otimes$ Ψ') $\otimes$ ($\Psi_Q \otimes \Psi''$) $\simeq$ Ψ_P' $\otimes$ Ψ_Q'
by(*metis Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately have ((p · Ψ_P) $\otimes$ Ψ_Q) $\otimes$ ($\Psi' \otimes \Psi''$) $\simeq$ $\Psi_P' \otimes \Psi_Q'$
by(*metis AssertionStatEqTrans*)
hence (p · ((p · Ψ_P) $\otimes$ Ψ_Q) $\otimes$ ($\Psi' \otimes \Psi''$)) $\simeq$ (p · ($\Psi_P' \otimes \Psi_Q'$))
by(*rule AssertionStatEqClosed*)
with ‹*xvec* ♯* Ψ_Q› ‹(p · *xvec*) ♯* Ψ_Q› S ‹*distinctPerm p*› **show** *?thesis*
by(*simp add*: *eqvts*)
qed

moreover from ‹(p · *xvec*) ♯* C› ‹A_P' ♯* C› ‹A_Q' ♯* C› **have** ((p · *xvec*)@A_P'@A_Q') ♯* C **by** *simp*
moreover from ‹(p · *xvec*) ♯* X› ‹A_P' ♯* X› ‹A_Q' ♯* X› **have** ((p · *xvec*)@A_P'@A_Q') ♯* X **by** *simp*
moreover from ‹(p · *xvec*) ♯* Y› ‹A_P' ♯* Y› ‹A_Q' ♯* Y› **have** ((p · *xvec*)@A_P'@A_Q') ♯* Y **by** *simp*
moreover from ‹(p · *xvec*) ♯* Z› ‹A_P' ♯* Z› ‹A_Q' ♯* Z› **have** ((p · *xvec*)@A_P'@A_Q') ♯* Z **by** *simp*
moreover from ‹(p · *xvec*) ♯* P'› ‹(p · *xvec*) ♯* Q'› ‹A_P' ♯* P'› ‹A_P' ♯* Q'› ‹A_Q' ♯* P'› ‹A_Q' ♯* Q'›
have ((p · *xvec*)@A_P'@A_Q') ♯* ($\llparenthesis\nu*xvec\rrparenthesis$($P'$ ‖ Q')) **by**(*auto simp add*: *resChainFresh fresh-star-def*)
moreover from ‹(p · *xvec*) ♯* A_P'› ‹ A_Q' ♯* (p · *xvec*)› ‹A_Q' ♯* A_P'› ‹*distinct xvec*› ‹*distinct* A_P'› ‹*distinct* A_Q'›
have *distinct*((p · *xvec*)@A_P'@A_Q')
by *auto* (*simp add*: *name-list-supp fresh-star-def fresh-def*)+

ultimately show *?case* **using** *cComm2* **by** *metis*
next

case(*cScope* Ψ P P' x A_P Ψ_P C X Y Z)
then obtain Ψ' A_P' Ψ_P' **where** *FrP'*: *extractFrame* $P' = \langle A_P', \Psi_P' \rangle$ **and** *distinct* A_P'
and $\Psi_P \otimes \Psi' \simeq \Psi_P'$ **and** $A_P' \sharp* C$ **and** $A_P' \sharp* P'$
and $A_P' \sharp* (x\#X)$ **and** $A_P' \sharp* Y$ **and** $A_P' \sharp* Z$
by(*rule-tac cScope*(*4*)[**where** *ba*=$x\#X$]) *auto*
from ‹$A_P' \sharp* (x\#X)$› **have** $x \sharp A_P'$ **and** $A_P' \sharp* X$ **by** *simp*+
moreover from *FrP'* **have** *extractFrame*($(\!|\nu x|\!)P'$) = $\langle (x\#A_P'), \Psi_P' \rangle$ **by** *simp*
moreover note ‹$\Psi_P \otimes \Psi' \simeq \Psi_P'$›
moreover from ‹$x \sharp C$› ‹$A_P' \sharp* C$› **have** $(x\#A_P') \sharp* C$ **by** *simp*
moreover from ‹$A_P' \sharp* P'$› **have** $(x\#A_P') \sharp* ((\!|\nu x|\!)P')$ **by**(*simp add*: *abs-fresh fresh-star-def*)
moreover from ‹$x \sharp X$› ‹$A_P' \sharp* X$› **have** $(x\#A_P') \sharp* X$ **by** *simp*
moreover from ‹$x \sharp Y$› ‹$A_P' \sharp* Y$› **have** $(x\#A_P') \sharp* Y$ **by** *simp*
moreover from ‹$x \sharp Z$› ‹$A_P' \sharp* Z$› **have** $(x\#A_P') \sharp* Z$ **by** *simp*
moreover from ‹$x \sharp A_P'$› ‹*distinct* A_P'› **have** *distinct*($x\#A_P'$) **by** *simp*
ultimately show *?case* **by**(*rule-tac cScope*)
next
case(*cBang* Ψ P P' A_P Ψ_P C B Y Z)
then obtain Ψ' A_P' Ψ_P' **where** *FrP'*: *extractFrame* $P' = \langle A_P', \Psi_P' \rangle$
and $(\Psi_P \otimes \mathbf{1}) \otimes \Psi' \simeq \Psi_P'$
and $A_P' \sharp* C$ **and** $A_P' \sharp* P'$ **and** *distinct* A_P'
and $A_P' \sharp* B$ **and** $A_P' \sharp* Y$ **and** $A_P' \sharp* Z$
by(*rule-tac cBang*) (*assumption* | *simp*)+
with ‹$\Psi_P \simeq \mathbf{1}$› ‹$(\Psi_P \otimes \mathbf{1}) \otimes \Psi' \simeq \Psi_P'$› **have** $\mathbf{1} \otimes \Psi' \simeq \Psi_P'$
by(*metis Identity AssertionStatEqTrans composition' Commutativity Associativity AssertionStatEqSym*)
thus *?case* **using** *FrP'* ‹$A_P' \sharp* P'$› ‹$A_P' \sharp* C$› ‹$A_P' \sharp* B$› ‹$A_P' \sharp* Y$› ‹$A_P' \sharp* Z$› ‹*distinct* A_P'›
by(*rule-tac cBang*)
qed
with *A* **have** *?thesis* **by** *simp*
}
moreover have $A_P \sharp* ([]::name\ list)$ **and** $A_P \sharp* ([]::'b\ list)$ **and** $A_P \sharp* ([]::('a, 'b, 'c)\ psi\ list)$ **by** *simp*+
ultimately show *?thesis* **by** *blast*
qed

lemma *expandFrame*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and α :: $'a\ action$
and P' :: $('a, 'b, 'c)\ psi$
and A_P :: *name list*
and Ψ_P :: $'b$
and C :: $'d::fs\text{-}name$
and C' :: $'e::fs\text{-}name$

assumes *Trans*: $\Psi \rhd P \longmapsto \alpha \prec P'$

and $extractFrame\ P = \langle A_P, \Psi_P \rangle$
and $distinct\ A_P$
and $bn\ \alpha \sharp* subject\ \alpha$
and $distinct(bn\ \alpha)$
and $A_P \sharp* \alpha$
and $A_P \sharp* P$
and $A_P \sharp* C$
and $A_P \sharp* C'$
and $bn\ \alpha \sharp* P$
and $bn\ \alpha \sharp* C'$

obtains $p\ \Psi'\ A_P'\ \Psi_P'$ **where** $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** $(p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_P'$ **and** $distinctPerm\ p$ **and**
$extractFrame\ P' = \langle A_P', \Psi_P' \rangle$ **and** $A_P' \sharp* P'$ **and** $A_P' \sharp* \alpha$ **and** $A_P' \sharp* (p \cdot \alpha)$ **and**
$A_P' \sharp* C$ **and** $(bn(p \cdot \alpha)) \sharp* C'$ **and** $(bn(p \cdot \alpha)) \sharp* \alpha$ **and** $(bn(p \cdot \alpha)) \sharp* P'$ **and** $distinct\ A_P'$
using *assms*
apply(*cases* $\alpha=\tau$)
by(*auto intro*: *expandTauFrame*[**where** $C=C$] *expandNonTauFrame*[**where** $C=C$ **and** $C'=C'$])

abbreviation
frameImpJudge (‹- $\hookrightarrow_F$ -› [*80*, *80*] *80*)
where $F \hookrightarrow_F G \equiv FrameStatImp\ F\ G$

lemma *FrameStatEqImpCompose*:
fixes F :: *'b frame*
and G :: *'b frame*
and H :: *'b frame*
and I :: *'b frame*

assumes $F \simeq_F G$
and $G \hookrightarrow_F H$
and $H \simeq_F I$

shows $F \hookrightarrow_F I$
using *assms*
by(*auto simp add*: *FrameStatEq-def*) (*blast intro*: *FrameStatImpTrans*)

lemma *transferNonTauFrame*:
fixes Ψ_F :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a action*
and P' :: (*'a*, *'b*, *'c*) *psi*
and A_F :: *name list*
and A_G :: *name list*
and Ψ_G :: *'b*

assumes $\Psi_F \rhd P \longmapsto\alpha \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and *distinct*(*bn* α)
and $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
and $A_F \sharp* P$
and $A_G \sharp* P$
and $A_F \sharp*$ *subject* α
and $A_G \sharp*$ *subject* α
and $A_P \sharp* A_F$
and $A_P \sharp* A_G$
and $A_P \sharp* \Psi_F$
and $A_P \sharp* \Psi_G$
and $\alpha \neq \tau$

shows $\Psi_G \rhd P \longmapsto\alpha \prec P'$
using *assms*
proof(*nominal-induct* Ψ_F P *Rs*==$\alpha \prec P'$ A_P Ψ_P *avoiding*: α P' Ψ_G A_F A_G *rule*: *semanticsFrameInduct*)
case(*cAlpha* Ψ_F P A_P Ψ_P p α P' Ψ_G A_F A_G)
from ‹$\langle A_F, \Psi_F \otimes (p \cdot \Psi_P)\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes (p \cdot \Psi_P)\rangle$›
have $(p \cdot (\langle A_F, \Psi_F \otimes (p \cdot \Psi_P)\rangle)) \hookrightarrow_F (p \cdot (\langle A_G, \Psi_G \otimes (p \cdot \Psi_P)\rangle))$
by(*rule FrameStatImpClosed*)
with ‹$A_P \sharp* A_F$› ‹$(p \cdot A_P) \sharp* A_F$› ‹$A_P \sharp* \Psi_F$› ‹$(p \cdot A_P) \sharp* \Psi_F$› ‹$A_P \sharp* A_G$› ‹$(p \cdot A_P) \sharp* A_G$› ‹$A_P \sharp* \Psi_G$› ‹$(p \cdot A_P) \sharp* \Psi_G$›
‹*distinctPerm p*› ‹*set p* $\subseteq$ *set* $A_P \times$ *set* $(p \cdot A_P)$› **have** $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
by(*simp add*: *eqvts*)
with *cAlpha* **show** *?case* **by** *force*
next
case(*cInput* Ψ_F M K *xvec* N *Tvec* P α P' Ψ_G A_F A_G)
from *cInput* **have** $A_F \sharp* K$ **and** $A_G \sharp* K$ **by**(*auto simp add*: *residualInject*)

from ‹$A_F \sharp* (M(\!|\lambda*xvec\ N|\!).P)$› ‹$A_G \sharp* (M(\!|\lambda*xvec\ N|\!).P)$› **have** $A_F \sharp* M$ **and** $A_G \sharp* M$ **by** *simp+*
from ‹$\Psi_F \vdash M \leftrightarrow K$›
have $\Psi_F \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
with ‹$A_F \sharp* M$› ‹$A_F \sharp* K$›
have $(\langle A_F, \Psi_F \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow K$
by(*force intro*: *frameImpI*)
with ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$›
have $(\langle A_G, \Psi_G \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow K$
by(*simp add*: *FrameStatEq-def FrameStatImp-def*)
with ‹$A_G \sharp* M$› ‹$A_G \sharp* K$›
have $\Psi_G \otimes \mathbf{1} \vdash M \leftrightarrow K$ **by**(*force dest*: *frameImpE*)
hence $\Psi_G \vdash M \leftrightarrow K$ **by**(*blast intro*: *statEqEnt Identity*)
thus *?case* **using** ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*› **using** *cInput Input*

by(*force simp add*: *residualInject*)
next
case(*cOutput* Ψ_F M K N P α P' Ψ_G A_F A_G)
from *cOutput* **have** $A_F \sharp* K$ **and** $A_G \sharp* K$ **by**(*auto simp add*: *residualInject*)

from ‹$A_F \sharp* (M\langle N\rangle.P)$› ‹$A_G \sharp* (M\langle N\rangle.P)$› **have** $A_F \sharp* M$ **and** $A_G \sharp* M$ **by** *simp+*
from ‹$\Psi_F \vdash M \leftrightarrow K$›
have $\Psi_F \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
with ‹$A_F \sharp* M$› ‹$A_F \sharp* K$›
have $(\langle A_F, \Psi_F \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow K$
by(*force intro*: *frameImpI*)
with ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$›
have $(\langle A_G, \Psi_G \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow K$
by(*simp add*: *FrameStatImp-def*)
with ‹$A_G \sharp* M$› ‹$A_G \sharp* K$›
have $\Psi_G \otimes \mathbf{1} \vdash M \leftrightarrow K$ **by**(*force dest*: *frameImpE*)
hence $\Psi_G \vdash M \leftrightarrow K$ **by**(*blast intro*: *statEqEnt Identity*)
thus *?case* **using** *cOutput Output* **by**(*force simp add*: *residualInject*)
next
case(*cCase* Ψ_F P φ *Cs* A_P Ψ_P α P' Ψ_G A_F A_G)
from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_F, \Psi_F \otimes \Psi_P\rangle \simeq_F \langle A_F, \Psi_F \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans*)
moreover note ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_G, \Psi_G \otimes \mathbf{1}\rangle \simeq_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans AssertionStatEqSym*)
ultimately have $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
by(*rule FrameStatEqImpCompose*)
with *cCase* **have** $\Psi_G \rhd P \longmapsto \alpha \prec P'$ **by**(*fastforce dest*: *memFreshChain*)
moreover note ‹(φ, P) *mem Cs*›
moreover from ‹$A_F \sharp* (Cases\ Cs)$›‹$A_G \sharp* (Cases\ Cs)$› ‹(φ, P) *mem Cs*› **have** $A_F \sharp* \varphi$ **and** $A_G \sharp* \varphi$
by(*auto dest*: *memFreshChain*)
from ‹$\Psi_F \vdash \varphi$› **have** $\Psi_F \otimes \mathbf{1} \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
with ‹$A_F \sharp* \varphi$› **have** $(\langle A_F, \Psi_F \otimes \mathbf{1}\rangle) \vdash_F \varphi$ **by**(*force intro*: *frameImpI*)
with ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$› **have** $(\langle A_G, \Psi_G \otimes \mathbf{1}\rangle) \vdash_F \varphi$
by(*simp add*: *FrameStatImp-def*)
with ‹$A_G \sharp* \varphi$› **have** $\Psi_G \otimes \mathbf{1} \vdash \varphi$ **by**(*force dest*: *frameImpE*)
hence $\Psi_G \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity*)
ultimately show *?case* **using** ‹*guarded P*› *cCase Case* **by**(*force intro*: *residualInject*)
next
case(*cPar1* Ψ_F Ψ_Q P α P' A_Q Q A_P Ψ_P α' PQ' Ψ_G A_F A_G)
from ‹$A_F \sharp* (P \parallel Q)$› ‹$A_G \sharp* (P \parallel Q)$› **have** $A_F \sharp* P$ **and** $A_G \sharp* P$ **and** $A_F \sharp* Q$ **and** $A_G \sharp* Q$
by *simp+*

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
moreover note ‹$\langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle$›
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
ultimately have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
by(*rule FrameStatEqImpCompose*)
moreover from ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› *FrQ* ‹$A_Q \sharp* A_F$› ‹$A_Q \sharp* A_G$› **have** $A_F \sharp* \Psi_Q$ **and** $A_G \sharp* \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+
moreover note ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› ‹$A_F \sharp*$ *subject* α'› ‹$A_G \sharp*$ *subject* α'› ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* A_G$› ‹$A_P \sharp* \Psi_G$›
moreover from ‹$\alpha \prec P' \parallel Q = \alpha' \prec PQ'$› ‹$bn\ \alpha \sharp* \alpha'$›
obtain $p\ P''$ **where** $\alpha \prec P' = \alpha' \prec P''$ **and** $set\ p \subseteq set(bn\ \alpha') \times set(bn\ \alpha)$ **and** $PQ' = P'' \parallel (p \cdot Q)$
apply(*drule-tac sym*)
by(*rule-tac actionPar1Dest*) (*assumption* | *simp* | *blast dest*: *sym*)+
ultimately have $\Psi_G \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$ **using** ‹$\alpha' \neq \tau$› **by**(*force intro*: *cPar1*)
thus *?case* **using** *FrQ* ‹$(bn\ \alpha) \sharp* Q$› ‹$A_Q \sharp* \Psi_G$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$› **using** *cPar1*
by(*metis Par1*)
next
case(*cPar2* $\Psi_F\ \Psi_P\ Q\ \alpha\ Q'\ A_P\ P\ A_Q\ \Psi_Q\ \alpha'\ PQ'\ \Psi_G\ A_F\ A_G$)
from ‹$A_F \sharp* (P \parallel Q)$› ‹$A_G \sharp* (P \parallel Q)$› **have** $A_F \sharp* P$ **and** $A_G \sharp* P$ **and** $A_F \sharp* Q$ **and** $A_G \sharp* Q$
by *simp*+
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity frameResChainPres frameNilStatEq*)
moreover note ‹$\langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle$›
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
by(*metis Associativity AssertionStatEqSym frameResChainPres frameNilStatEq*)
ultimately have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
by(*rule FrameStatEqImpCompose*)
moreover from ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› *FrP* ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› **have** $A_F \sharp* \Psi_P$ **and** $A_G \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)+
moreover note ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› ‹$A_F \sharp*$ *subject* α'› ‹$A_G \sharp*$ *subject* α'› ‹$A_Q \sharp* A_F$› ‹$A_Q \sharp* A_G$› ‹$A_Q \sharp* \Psi_F$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* A_G$› ‹$A_Q \sharp* \Psi_G$›
moreover from ‹$\alpha \prec P \parallel Q' = \alpha' \prec PQ'$› ‹$bn\ \alpha \sharp* \alpha'$›
obtain $p\ Q''$ **where** $\alpha \prec Q' = \alpha' \prec Q''$ **and** $set\ p \subseteq set(bn\ \alpha') \times set(bn\ \alpha)$ **and** $PQ' = (p \cdot P) \parallel Q''$
apply(*drule-tac sym*)
by(*rule-tac actionPar2Dest*) (*assumption* | *simp* | *blast dest*: *sym*)+
ultimately have $\Psi_G \otimes \Psi_P \rhd Q \longmapsto\alpha \prec Q'$ **using** ‹$\alpha' \neq \tau$› **by**(*force intro*:

cPar2)
thus *?case* **using** *FrP* ‹$(bn\ \alpha)\ \sharp*\ P$› ‹$A_P\ \sharp*\ \Psi_G$› ‹$A_P\ \sharp*\ Q$› ‹$A_P\ \sharp*\ \alpha$› **using** *cPar2*
by(*metis Par2*)
next
case *cComm1*
thus *?case* **by**(*simp add*: *residualInject*)
next
case *cComm2*
thus *?case* **by**(*simp add*: *residualInject*)
next
case(*cOpen* $\Psi_F\ P\ M\ xvec\ yvec\ N\ P'\ x\ A_P\ \Psi_P\ \alpha\ P''\ \Psi_G\ A_F\ A_G$)
from ‹$M(\!|\nu*(xvec\ @\ x\ \#\ yvec)|\!)\langle N\rangle \prec P' = \alpha \prec P''$› ‹$x\ \sharp\ xvec$› ‹$x\ \sharp\ yvec$› ‹$x\ \sharp\ \alpha$› ‹$x\ \sharp\ P''$› ‹$distinct(bn\ \alpha)$›
obtain $xvec'\ x'\ yvec'\ N'$ **where** $M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P' = M(\!|\nu*(xvec'@yvec')|\!)\langle([(x, x')] \cdot N')\rangle \prec ([(x, x')] \cdot P'')$
and $\alpha = M(\!|\nu*(xvec'@x'\#yvec')|\!)\langle N'\rangle$
apply(*cases rule*: *actionCases*[**where** $\alpha=\alpha$])
apply(*auto simp add*: *residualInject*)
apply(*rule boundOutputOpenDest*) **by** *assumption auto*

then have $\Psi_G \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$ **using** *cOpen*
by(*rule-tac cOpen*) (*assumption* | *simp*)+
with ‹$x \in supp\ N$› ‹$x\ \sharp\ \Psi_G$› ‹$x\ \sharp\ M$› ‹$x\ \sharp\ xvec$› ‹$x\ \sharp\ yvec$›
have $\Psi_G \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$
by(*rule-tac Open*)
thus *?case* **using** ‹$\alpha = M(\!|\nu*(xvec'@x'\#yvec')|\!)\langle N'\rangle$› ‹$M(\!|\nu*(xvec\ @\ x\ \#\ yvec)|\!)\langle N\rangle \prec P' = \alpha \prec P''$›
by *simp*
next
case(*cScope* $\Psi_F\ P\ \alpha\ P'\ x\ A_P\ \Psi_P\ \alpha'\ xP\ \Psi_G\ A_F\ A_G$)
from ‹$\alpha \prec (\!|\nu x|\!)P' = \alpha' \prec xP$› ‹$x\ \sharp\ \alpha$› ‹$x\ \sharp\ \alpha'$› **obtain** P'' **where** $xP = (\!|\nu x|\!)P''$ **and** $\alpha \prec P' = \alpha' \prec P''$
by(*drule-tac sym*) (*force intro*: *actionScopeDest*)
then have $\Psi_G \rhd P \longmapsto \alpha \prec P'$ **using** *cScope* **by** *auto*
with ‹$x\ \sharp\ \Psi_G$› ‹$x\ \sharp\ \alpha'$› ‹$\alpha \prec P' = \alpha' \prec P''$› ‹$xP = (\!|\nu x|\!)P''$› **show** *?case*
by(*metis Scope*)
next
case(*cBang* $\Psi_F\ P\ A_P\ \Psi_P\ \alpha\ P'\ \Psi_G\ A_F\ A_G$)
from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_F, \Psi_F \otimes \Psi_P \otimes \mathbf{1}\rangle \simeq_F \langle A_F, \Psi_F \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans*)
moreover note ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_G, \Psi_G \otimes \mathbf{1}\rangle \simeq_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans AssertionStatEqSym*)
ultimately have $\langle A_F, \Psi_F \otimes \Psi_P \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \mathbf{1}\rangle$
by(*rule FrameStatEqImpCompose*)
with *cBang* **have** $\Psi_G \rhd P \parallel !P \longmapsto \alpha \prec P'$ **by** *force*
thus *?case* **using** ‹*guarded P*› **using** *cBang* **by**(*metis Bang*)

qed

lemma *transferTauFrame*:
fixes Ψ_F :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and A_F :: *name list*
and A_G :: *name list*
and Ψ_G :: $'b$

assumes $\Psi_F \rhd P \longmapsto\tau \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
and $A_F \sharp* P$
and $A_G \sharp* P$
and $A_P \sharp* A_F$
and $A_P \sharp* A_G$
and $A_P \sharp* \Psi_F$
and $A_P \sharp* \Psi_G$

shows $\Psi_G \rhd P \longmapsto\tau \prec P'$
using *assms*
proof(*nominal-induct avoiding*: Ψ_G A_F A_G *rule*: *tauFrameInduct*)
case(*cAlpha* Ψ_F P P' A_P Ψ_P p Ψ_G A_F A_G)
from ‹$\langle A_F, \Psi_F \otimes (p \cdot \Psi_P)\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes (p \cdot \Psi_P)\rangle$›
have $(p \cdot (\langle A_F, \Psi_F \otimes (p \cdot \Psi_P)\rangle)) \hookrightarrow_F (p \cdot (\langle A_G, \Psi_G \otimes (p \cdot \Psi_P)\rangle))$
by(*rule FrameStatImpClosed*)
with ‹$A_P \sharp* A_F$› ‹$(p \cdot A_P) \sharp* A_F$› ‹$A_P \sharp* \Psi_F$› ‹$(p \cdot A_P) \sharp* \Psi_F$› ‹$A_P \sharp* A_G$› ‹$(p \cdot A_P) \sharp* A_G$› ‹$A_P \sharp* \Psi_G$› ‹$(p \cdot A_P) \sharp* \Psi_G$›
‹*distinctPerm* p› ‹*set* $p \subseteq$ *set* $A_P \times$ *set* $(p \cdot A_P)$› **have** $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
by(*simp add*: *eqvts*)
with *cAlpha* **show** *?case* **by** *blast*
next
case(*cCase* Ψ_F P P' φ *Cs* A_P Ψ_P Ψ_G A_F A_G)
from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_F, \Psi_F \otimes \Psi_P\rangle \simeq_F \langle A_F, \Psi_F \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans*)
moreover note ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_G, \Psi_G \otimes \mathbf{1}\rangle \simeq_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans AssertionStatEqSym*)
ultimately have $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
by(*rule FrameStatEqImpCompose*)
with *cCase* **have** $\Psi_G \rhd P \longmapsto\tau \prec P'$ **by**(*fastforce dest*: *memFreshChain*)
moreover note ‹(φ, P) *mem Cs*›
moreover from ‹$A_F \sharp*$ (*Cases Cs*)›‹$A_G \sharp*$ (*Cases Cs*)› ‹(φ, P) *mem Cs*› **have** $A_F \sharp* \varphi$ **and** $A_G \sharp* \varphi$
by(*auto dest*: *memFreshChain*)

from ‹$\Psi_F \vdash \varphi$› **have** $\Psi_F \otimes \mathbf{1} \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
with ‹$A_F \sharp\!* \varphi$› **have** $(\langle A_F, \Psi_F \otimes \mathbf{1}\rangle) \vdash_F \varphi$ **by**(*force intro*: *frameImpI*)
with ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$› **have** $(\langle A_G, \Psi_G \otimes \mathbf{1}\rangle) \vdash_F \varphi$
by(*simp add*: *FrameStatImp-def*)
with ‹$A_G \sharp\!* \varphi$› **have** $\Psi_G \otimes \mathbf{1} \vdash \varphi$ **by**(*force dest*: *frameImpE*)
hence $\Psi_G \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity*)
ultimately show *?case* **using** ‹*guarded P*› **by**(*rule Case*)
next
case(*cPar1* Ψ_F Ψ_Q P P' A_Q Q A_P Ψ_P Ψ_G A_F A_G)
from ‹$A_F \sharp\!* (P \parallel Q)$› ‹$A_G \sharp\!* (P \parallel Q)$› **have** $A_F \sharp\!* P$ **and** $A_G \sharp\!* P$ **and** $A_F \sharp\!* Q$ **and** $A_G \sharp\!* Q$
by *simp+*
have *IH*: $\bigwedge\Psi\ A_F\ A_G.\ [\![\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi \otimes \Psi_P\rangle;\ A_F \sharp\!* P;\ A_G \sharp\!* P;$
$A_P \sharp\!* A_F;\ A_P \sharp\!* A_G;\ A_P \sharp\!* (\Psi_F \otimes \Psi_Q);\ A_P \sharp\!* \Psi]\!] \Longrightarrow \Psi \rhd P \longmapsto\tau \prec P'$
by *fact*
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
moreover note ‹$\langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle$›
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
ultimately have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
by(*rule FrameStatEqImpCompose*)
moreover from ‹$A_F \sharp\!* Q$› ‹$A_G \sharp\!* Q$› *FrQ* ‹$A_Q \sharp\!* A_F$› ‹$A_Q \sharp\!* A_G$› **have** $A_F \sharp\!* \Psi_Q$ **and** $A_G \sharp\!* \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+
moreover note ‹$A_F \sharp\!* P$› ‹$A_G \sharp\!* P$› ‹$A_P \sharp\!* A_F$› ‹$A_P \sharp\!* A_G$› ‹$A_P \sharp\!* \Psi_F$› ‹$A_P \sharp\!* \Psi_Q$› ‹$A_P \sharp\!* A_G$› ‹$A_P \sharp\!* \Psi_G$›
ultimately have $\Psi_G \otimes \Psi_Q \rhd P \longmapsto\tau \prec P'$ **by**(*rule-tac IH*) (*assumption* | *auto*)+
thus *?case* **using** *FrQ* ‹$A_Q \sharp\!* \Psi_G$› ‹$A_Q \sharp\!* P$›
by(*rule-tac Par1*) *auto*
next
case(*cPar2* Ψ_F Ψ_P Q Q' A_P P A_Q Ψ_Q Ψ_G A_F A_G)
from ‹$A_F \sharp\!* (P \parallel Q)$› ‹$A_G \sharp\!* (P \parallel Q)$› **have** $A_F \sharp\!* P$ **and** $A_G \sharp\!* P$ **and** $A_F \sharp\!* Q$ **and** $A_G \sharp\!* Q$
by *simp+*
have *IH*: $\bigwedge\Psi\ A_F\ A_G.\ [\![\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi \otimes \Psi_Q\rangle;\ A_F \sharp\!* Q;\ A_G \sharp\!* Q;$
$A_Q \sharp\!* A_F;\ A_Q \sharp\!* A_G;\ A_Q \sharp\!* (\Psi_F \otimes \Psi_P);\ A_Q \sharp\!* \Psi]\!] \Longrightarrow \Psi \rhd Q \longmapsto\tau \prec Q'$
by *fact*
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$

by(*metis Associativity frameResChainPres frameNilStatEq*)
moreover note ‹$\langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle$›
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
by(*metis Associativity AssertionStatEqSym frameResChainPres frameNilStatEq*)
ultimately have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
by(*rule FrameStatEqImpCompose*)
moreover from ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› *FrP* ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› **have** $A_F \sharp* \Psi_P$ **and** $A_G \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)+
moreover note ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› ‹$A_Q \sharp* A_F$› ‹$A_Q \sharp* A_G$› ‹$A_Q \sharp* \Psi_F$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* A_G$› ‹$A_Q \sharp* \Psi_G$›
ultimately have $\Psi_G \otimes \Psi_P \rhd Q \longmapsto\tau \prec Q'$ **by**(*rule-tac IH*) (*assumption* | *auto*)+
thus *?case* **using** *FrP* ‹$A_P \sharp* \Psi_G$› ‹$A_P \sharp* Q$›
by(*rule-tac Par2*) *auto*
next
case(*cComm1* Ψ_F Ψ_Q P M N P' A_P Ψ_P Q K *xvec* Q' A_Q Ψ_G A_F A_G)
have *FimpG*: $\langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle$ **by** *fact*
from ‹$A_F \sharp* (P \parallel Q)$› ‹$A_G \sharp* (P \parallel Q)$› **have** $A_F \sharp* P$ **and** $A_G \sharp* P$ **and** $A_F \sharp* Q$ **and** $A_G \sharp* Q$
by *simp*+
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› *FrP* ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› **have** $A_F \sharp* \Psi_P$ **and** $A_G \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)+
from ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› *FrQ* ‹$A_Q \sharp* A_F$› ‹$A_Q \sharp* A_G$› **have** $A_F \sharp* \Psi_Q$ **and** $A_G \sharp* \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+
from ‹$\Psi_F \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› *FrQ* ‹*distinct* A_Q›
obtain K' **where** *KeqK'*: $(\Psi_F \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow K'$ **and** $A_P \sharp* K'$ **and** $A_F \sharp* K'$ **and** $A_G \sharp* K'$
using ‹$A_P \sharp* Q$› ‹$A_Q \sharp* A_F$› ‹$A_Q \sharp* A_G$› ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› ‹$A_Q \sharp* \Psi_F$› ‹$A_Q \sharp* \Psi_P$› ‹$A_P \sharp* A_Q$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹*xvec* $\sharp* K$› ‹*distinct xvec*›
by(*rule-tac* $B{=}A_P@A_F@A_G$ **in** *obtainPrefix*) *force*+
have $\Psi_G \otimes \Psi_Q \rhd P \longmapsto K'(\!|N|\!) \prec P'$
proof −
from *KeqK'* **have** $\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash K \leftrightarrow K'$ **by**(*rule statEqEnt*[*OF Associativity*])
with ‹$\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K$› **have** $\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K'$
by(*rule chanEqTrans*)
hence $(\Psi_F \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K'$
by(*metis statEqEnt AssertionStatEqSym Associativity AssertionStatEqTrans compositionSym Commutativity*)
with ‹$\Psi_F \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› *FrP* ‹*distinct* A_P›
have $\Psi_F \otimes \Psi_Q \rhd P \longmapsto K'(\!|N|\!) \prec P'$ **using** ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K'$›
by(*force intro*: *inputRenameSubject*)
moreover have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$

proof –
have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
ultimately show *?thesis* **using** *FimpG*
by(*rule-tac FrameStatEqImpCompose*)
qed
ultimately show *?thesis* **using** ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› ‹$A_F \sharp* K'$› ‹$A_G \sharp* K'$› ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_G$› ‹$A_P \sharp* \Psi_Q$› *FrP* ‹*distinct* A_P›
by(*auto intro*: *transferNonTauFrame*)
qed

moreover from *FrP* ‹$\Psi_F \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› ‹*distinct* A_P›
obtain M' **where** *MeqM'*: $(\Psi_F \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $A_Q \sharp* M'$ **and** $A_F \sharp* M'$ **and** $A_G \sharp* M'$
using ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$›
by(*rule-tac* $B{=}A_Q@A_F@A_G$ **in** *obtainPrefix*) *force*+

have $\Psi_G \otimes \Psi_P \rhd Q \longmapsto M'(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
proof –
from *MeqM'* **have** $\Psi_F \otimes (\Psi_Q \otimes \Psi_P) \vdash M \leftrightarrow M'$ **by**(*rule statEqEnt*[*OF Associativity*])
with ‹$\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K$› **have** $\Psi_F \otimes (\Psi_Q \otimes \Psi_P) \vdash K \leftrightarrow M'$
by(*blast intro*: *chanEqTrans chanEqSym compositionSym Commutativity statEqEnt*)
hence $(\Psi_F \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*blast intro*: *statEqEnt AssertionStatEqSym Associativity AssertionStatEqTrans compositionSym Commutativity*)
with ‹$\Psi_F \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› *FrQ* ‹*distinct* A_Q›
have $\Psi_F \otimes \Psi_P \rhd Q \longmapsto M'(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$ **using** ‹$A_Q \sharp* \Psi_F$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
by(*force intro*: *outputRenameSubject*)
moreover have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
proof –
have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity frameResChainPres frameNilStatEq*)
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
by(*metis Associativity AssertionStatEqSym frameResChainPres frameNilStatEq*)
ultimately show *?thesis* **using** *FimpG*
by(*rule-tac FrameStatEqImpCompose*)
qed

ultimately show *?thesis* **using** ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› ‹$A_F \sharp* M'$› ‹$A_G \sharp* M'$›

‹A_Q ♯* A_F› ‹A_Q ♯* A_G› ‹A_Q ♯* Ψ_F› ‹A_Q ♯* Ψ_G› ‹A_Q ♯* Ψ_P› *FrQ* ‹*distinct* A_Q› ‹*distinct xvec*›
by(*auto intro*: *transferNonTauFrame*)
qed

moreover have $\Psi_G \otimes \Psi_P \otimes \Psi_Q \vdash K' \leftrightarrow M'$
proof −
from *MeqM'* **have** $\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash M' \leftrightarrow M$
by(*blast intro*: *chanEqSym Associativity statEqEnt Commutativity compositionSym*)
moreover from *KeqK'* **have** $\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow K'$
by(*blast intro*: *chanEqSym Associativity statEqEnt Commutativity compositionSym*)
ultimately have $\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash K' \leftrightarrow M'$ **using** ‹$\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$›
by(*blast intro*: *chanEqSym chanEqTrans*)
thus *?thesis* **using** ‹A_F ♯* M'› ‹A_F ♯* K'› ‹A_G ♯* M'› ‹A_G ♯* K'› *FimpG*
apply(*auto simp add*: *FrameStatImp-def*)
apply(*erule-tac x=SChanEq' K' M'* **in** *allE*)
by(*force intro*: *frameImpI dest*: *frameImpE*)
qed

ultimately show *?case* **using** ‹A_P ♯* Ψ_G› ‹A_P ♯* P› ‹A_P ♯* Q› ‹A_P ♯* A_Q› ‹A_P ♯* K'› ‹A_Q ♯* Ψ_G› ‹A_Q ♯* P› ‹A_Q ♯* Q› ‹A_Q ♯* M'› ‹*xvec* ♯* P› *FrP FrQ*
by(*rule-tac Comm1*) (*assumption* | *simp*)+
next
case(*cComm2* Ψ_F Ψ_Q P M *xvec* N P' A_P Ψ_P Q K Q' A_Q Ψ_G A_F A_G)
have *FimpG*: $\langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle$ **by** *fact*
from ‹A_F ♯* $(P \parallel Q)$› ‹A_G ♯* $(P \parallel Q)$› **have** A_F ♯* P **and** A_G ♯* P **and** A_F ♯* Q **and** A_G ♯* Q
by *simp*+
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹A_F ♯* P› ‹A_G ♯* P› *FrP* ‹A_P ♯* A_F› ‹A_P ♯* A_G› **have** A_F ♯* Ψ_P **and** A_G ♯* Ψ_P
by(*force dest*: *extractFrameFreshChain*)+
from ‹A_F ♯* Q› ‹A_G ♯* Q› *FrQ* ‹A_Q ♯* A_F› ‹A_Q ♯* A_G› **have** A_F ♯* Ψ_Q **and** A_G ♯* Ψ_Q
by(*force dest*: *extractFrameFreshChain*)+
from ‹$\Psi_F \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› *FrQ* ‹*distinct* A_Q›
obtain K' **where** *KeqK'*: $(\Psi_F \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow K'$ **and** A_P ♯* K' **and** A_F ♯* K' **and** A_G ♯* K'
using ‹A_P ♯* Q› ‹A_Q ♯* A_F› ‹A_Q ♯* A_G› ‹A_F ♯* Q› ‹A_G ♯* Q› ‹A_Q ♯* Ψ_F› ‹A_Q ♯* Ψ_P› ‹A_P ♯* A_Q› ‹A_Q ♯* Q› ‹A_Q ♯* K›
by(*rule-tac* $B=A_P@A_F@A_G$ **in** *obtainPrefix*) *force*+
have $\Psi_G \otimes \Psi_Q \rhd P \longmapsto K'(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
proof −
from *KeqK'* **have** $\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash K \leftrightarrow K'$ **by**(*rule statEqEnt*[*OF Associativity*])

with ‹$\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K$› **have** $\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K'$
by(*rule chanEqTrans*)
hence $(\Psi_F \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K'$
by(*metis statEqEnt AssertionStatEqSym Associativity AssertionStatEqTrans compositionSym Commutativity*)
with ‹$\Psi_F \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› *FrP* ‹*distinct* A_P›
have $\Psi_F \otimes \Psi_Q \rhd P \longmapsto K'(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **using** ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K'$›
by(*force intro*: *outputRenameSubject*)
moreover have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
proof −
have $\langle A_F, (\Psi_F \otimes \Psi_Q) \otimes \Psi_P\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_Q) \otimes \Psi_P\rangle$
by(*metis Associativity Composition AssertionStatEqSym AssertionStatEqTrans Commutativity frameResChainPres frameNilStatEq*)
ultimately show *?thesis* **using** *FimpG*
by(*rule-tac FrameStatEqImpCompose*)
qed
ultimately show *?thesis* **using** ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› ‹$A_F \sharp* K'$› ‹$A_G \sharp* K'$› ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_G$› ‹$A_P \sharp* \Psi_Q$› *FrP* ‹*distinct* A_P› ‹*distinct xvec*›
by(*auto intro*: *transferNonTauFrame*)
qed

moreover from *FrP* ‹$\Psi_F \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹*distinct* A_P›
obtain M' **where** *MeqM'*: $(\Psi_F \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $A_Q \sharp* M'$ **and** $A_F \sharp* M'$ **and** $A_G \sharp* M'$
using ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_F$› ‹$A_P \sharp* A_G$› ‹$A_F \sharp* P$› ‹$A_G \sharp* P$› ‹$A_P \sharp* \Psi_F$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹*xvec* $\sharp* M$› ‹*distinct xvec*›
by(*rule-tac* $B{=}A_Q@A_F@A_G$ **in** *obtainPrefix*) *force*+

have $\Psi_G \otimes \Psi_P \rhd Q \longmapsto M'(\!|N|\!) \prec Q'$
proof −
from *MeqM'* **have** $\Psi_F \otimes (\Psi_Q \otimes \Psi_P) \vdash M \leftrightarrow M'$ **by**(*rule statEqEnt*[*OF Associativity*])
with ‹$\Psi_F \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K$› **have** $\Psi_F \otimes (\Psi_Q \otimes \Psi_P) \vdash K \leftrightarrow M'$
by(*blast intro*: *chanEqTrans chanEqSym compositionSym Commutativity statEqEnt*)
hence $(\Psi_F \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*blast intro*: *statEqEnt AssertionStatEqSym Associativity AssertionStatEqTrans compositionSym Commutativity*)
with ‹$\Psi_F \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› *FrQ* ‹*distinct* A_Q›
have $\Psi_F \otimes \Psi_P \rhd Q \longmapsto M'(\!|N|\!) \prec Q'$ **using** ‹$A_Q \sharp* \Psi_F$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
by(*force intro*: *inputRenameSubject*)
moreover have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$

proof −
have $\langle A_F, (\Psi_F \otimes \Psi_P) \otimes \Psi_Q\rangle \simeq_F \langle A_F, \Psi_F \otimes \Psi_P \otimes \Psi_Q\rangle$
by(*metis Associativity frameResChainPres frameNilStatEq*)
moreover have $\langle A_G, \Psi_G \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle A_G, (\Psi_G \otimes \Psi_P) \otimes \Psi_Q\rangle$
by(*metis Associativity AssertionStatEqSym frameResChainPres frameNilStatEq*)
ultimately show *?thesis* **using** *FimpG*
by(*rule-tac FrameStatEqImpCompose*)
qed

ultimately show *?thesis* **using** ‹$A_F \sharp* Q$› ‹$A_G \sharp* Q$› ‹$A_F \sharp* M'$› ‹$A_G \sharp* M'$› ‹$A_Q \sharp* A_F$› ‹$A_Q \sharp* A_G$› ‹$A_Q \sharp* \Psi_F$› ‹$A_Q \sharp* \Psi_G$› ‹$A_Q \sharp* \Psi_P$› *FrQ* ‹*distinct* A_Q› ‹*distinct xvec*›
by(*auto intro*: *transferNonTauFrame*)
qed

moreover have $\Psi_G \otimes \Psi_P \otimes \Psi_Q \vdash K' \leftrightarrow M'$
proof −
from *MeqM'* **have** $\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash M' \leftrightarrow M$
by(*blast intro*: *chanEqSym Associativity statEqEnt Commutativity compositionSym*)
moreover from *KeqK'* **have** $\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow K'$
by(*blast intro*: *chanEqSym Associativity statEqEnt Commutativity compositionSym*)
ultimately have $\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash K' \leftrightarrow M'$ **using** ‹$\Psi_F \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$›
by(*blast intro*: *chanEqSym chanEqTrans*)
thus *?thesis* **using** ‹$A_F \sharp* M'$› ‹$A_F \sharp* K'$› ‹$A_G \sharp* M'$› ‹$A_G \sharp* K'$› *FimpG*
apply(*auto simp add*: *FrameStatImp-def*)
apply(*erule-tac x=SChanEq' K' M'* **in** *allE*)
by(*force intro*: *frameImpI dest*: *frameImpE*)
qed

ultimately show *?case* **using** ‹$A_P \sharp* \Psi_G$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* K'$› ‹$A_Q \sharp* \Psi_G$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* M'$› ‹$xvec \sharp* Q$› *FrP FrQ*
by(*rule-tac Comm2*) (*assumption* | *simp*)+
next
case(*cScope* Ψ_F P P' x A_P Ψ_P Ψ_G A_F A_G)
then have $\Psi_G \rhd P \longmapsto\tau \prec P'$ **by** *auto*
with ‹$x \sharp \Psi_G$› **show** *?case*
by(*rule-tac Scope*) *auto*
next
case(*cBang* Ψ_F P P' A_P Ψ_P Ψ_G A_F A_G)
from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_F, \Psi_F \otimes \Psi_P \otimes \mathbf{1}\rangle \simeq_F \langle A_F, \Psi_F \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans*)
moreover note ‹$\langle A_F, \Psi_F \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_G, \Psi_G \otimes \mathbf{1}\rangle \simeq_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym Identity AssertionStatEqTrans AssertionStatEqSym*)

ultimately have $\langle A_F, \Psi_F \otimes \Psi_P \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \otimes \mathbf{1}\rangle$
by(*rule FrameStatEqImpCompose*)
with *cBang* **have** $\Psi_G \rhd P \parallel !P \longmapsto\tau \prec P'$ **by** *force*
thus *?case* **using** ‹*guarded P*› **by**(*rule Bang*)
qed

lemma *transferFrame*:
fixes Ψ_F :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and A_F :: *name list*
and A_G :: *name list*
and Ψ_G :: $'b$

assumes $\Psi_F \rhd P \longmapsto\alpha \prec P'$
and *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *distinct* A_P
and $\langle A_F, \Psi_F \otimes \Psi_P\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P\rangle$
and $A_F \sharp* P$
and $A_G \sharp* P$
and $A_F \sharp*$ *subject* α
and $A_G \sharp*$ *subject* α
and $A_P \sharp* A_F$
and $A_P \sharp* A_G$
and $A_P \sharp* \Psi_F$
and $A_P \sharp* \Psi_G$

shows $\Psi_G \rhd P \longmapsto\alpha \prec P'$
using *assms*
proof −
from ‹$\Psi_F \rhd P \longmapsto\alpha \prec P'$› **have** *distinct*(*bn* α) **by**(*auto dest*: *boundOutputDistinct*)
thus *?thesis* **using** *assms*
by(*cases* $\alpha = \tau$) (*auto intro*: *transferNonTauFrame transferTauFrame*)
qed

lemma *parCasesInputFrame*[*consumes 7*, *case-names cPar1 cPar2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and T :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto M(\!|N|\!) \prec T$
and *extractFrame*$(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$

and *distinct* A_{PQ}
and $A_{PQ} \sharp* \Psi$
and $A_{PQ} \sharp* P$
and $A_{PQ} \sharp* Q$
and $A_{PQ} \sharp* M$
and *rPar1*: $\bigwedge P'\ A_P\ \Psi_P\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$; *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$; *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$;
distinct A_P; *distinct* A_Q; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* M$; $A_Q \sharp* \Psi$; $A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* M$;
$A_P \sharp* \Psi_Q$; $A_Q \sharp* \Psi_P$; $A_P \sharp* A_Q$; $A_{PQ} = A_P@A_Q$; $\Psi_{PQ} = \Psi_P \otimes \Psi_Q]\!] \Longrightarrow$ *Prop* $(P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q'$; *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$; *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$;
distinct A_P; *distinct* A_Q; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* M$; $A_Q \sharp* \Psi$; $A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* M$;
$A_P \sharp* \Psi_Q$; $A_Q \sharp* \Psi_P$; $A_P \sharp* A_Q$; $A_{PQ} = A_P@A_Q$; $\Psi_{PQ} = \Psi_P \otimes \Psi_Q]\!] \Longrightarrow$ *Prop* $(P \parallel Q')$
shows *Prop T*
using *Trans*
proof(*induct rule*: *parInputCases*[*of* - - - - - - $(A_{PQ},\ \Psi_{PQ})$])
case(*cPar1* $P'\ A_Q\ \Psi_Q$)
from ‹$A_Q \sharp* (A_{PQ},\ \Psi_{PQ})$› **have** $A_Q \sharp* A_{PQ}$ **and** $A_Q \sharp* \Psi_{PQ}$ **by** *simp*+
obtain $A_P\ \Psi_P$ **where** *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$ **and** *distinct* A_P
$A_P \sharp* (P,\ Q,\ \Psi,\ M,\ A_Q,\ A_{PQ},\ \Psi_Q)$
by(*rule freshFrame*)
hence $A_P \sharp* P$ **and** $A_P \sharp* Q$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* M$ **and** $A_P \sharp* A_Q$ **and** $A_P \sharp* A_{PQ}$ **and** $A_P \sharp* \Psi_Q$
by *simp*+

have *FrQ*: *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$ **by** *fact*

from ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_Q$› *FrP* **have** $A_Q \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)

from ‹*extractFrame*$(P \parallel Q) = \langle A_{PQ},\ \Psi_{PQ}\rangle$› *FrP FrQ* ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›
have $\langle (A_P@A_Q),\ \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ},\ \Psi_{PQ}\rangle$ **by** *simp*
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› ‹$A_P \sharp* A_Q$› **have** *distinct*$(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
ultimately obtain p **where** S: *set* $p \subseteq$ *set*$(A_P@A_Q) \times$ *set*$((p \cdot A_P)@(p \cdot A_Q))$ **and** *distinctPerm* p
and Ψeq: $\Psi_{PQ} = (p \cdot \Psi_P) \otimes (p \cdot \Psi_Q)$ **and** *Aeq*: $A_{PQ} = (p \cdot A_P)@(p \cdot A_Q)$
using ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$› ‹*distinct* A_{PQ}›
by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$› S ‹$A_{PQ} \sharp* P$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_{PQ} \sharp* M$› ‹$A_P \sharp* M$› ‹$A_Q \sharp* M$› *Aeq*
have $(p \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto M(\!|N|\!) \prec P'$

by(*rule-tac inputPermFrame*) *auto*
with S ‹A_{PQ} $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ› *Aeq* **have** $\Psi \otimes (p \cdot \Psi_Q) \rhd P \longmapsto M(\!|N|\!) \prec P'$
by(*simp add*: *eqvts*)
moreover from *FrP* **have** $(p \cdot \mathit{extractFrame}\ P) = p \cdot \langle A_P, \Psi_P\rangle$ **by** *simp*
with S ‹A_{PQ} $\sharp*$ P› ‹A_P $\sharp*$ P› ‹A_Q $\sharp*$ P› *Aeq* **have** $\mathit{extractFrame}\ P = \langle (p \cdot A_P), p \cdot \Psi_P\rangle$
by(*simp add*: *eqvts*)
moreover from *FrQ* **have** $(p \cdot \mathit{extractFrame}\ Q) = p \cdot \langle A_Q, \Psi_Q\rangle$ **by** *simp*
with S ‹A_{PQ} $\sharp*$ Q› ‹A_P $\sharp*$ Q› ‹A_Q $\sharp*$ Q› *Aeq* **have** $\mathit{extractFrame}\ Q = \langle (p \cdot A_Q), p \cdot \Psi_Q\rangle$
by(*simp add*: *eqvts*)
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› **have** $\mathit{distinct}(p \cdot A_P)$ **and** $\mathit{distinct}(p \cdot A_Q)$
by *simp+*
moreover from ‹A_P $\sharp*$ A_Q› **have** $(p \cdot A_P)$ $\sharp*$ $(p \cdot A_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹A_P $\sharp*$ Ψ_Q› **have** $(p \cdot A_P)$ $\sharp*$ $(p \cdot \Psi_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹A_Q $\sharp*$ Ψ_P› **have** $(p \cdot A_Q)$ $\sharp*$ $(p \cdot \Psi_P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case* **using** ‹A_{PQ} $\sharp*$ Ψ› ‹A_{PQ} $\sharp*$ P› ‹A_{PQ} $\sharp*$ Q› ‹A_{PQ} $\sharp*$ M› *Aeq* Ψeq
by(*rule-tac rPar1*) (*assumption* | *simp*)+
next
case(*cPar2* Q' A_P Ψ_P)
from ‹A_P $\sharp*$ (A_{PQ}, Ψ_{PQ})› **have** A_P $\sharp*$ A_{PQ} **and** A_P $\sharp*$ Ψ_{PQ} **by** *simp+*
obtain A_Q Ψ_Q **where** *FrQ*: $\mathit{extractFrame}\ Q = \langle A_Q, \Psi_Q\rangle$ **and** $\mathit{distinct}\ A_Q$
A_Q $\sharp*$ $(P, Q, \Psi, M, A_P, A_{PQ}, \Psi_P)$
by(*rule freshFrame*)
hence A_Q $\sharp*$ P **and** A_Q $\sharp*$ Q **and** A_Q $\sharp*$ Ψ **and** A_Q $\sharp*$ M **and** A_Q $\sharp*$ A_P **and** A_Q $\sharp*$ A_{PQ} **and** A_Q $\sharp*$ Ψ_P
by *simp+*

have *FrP*: $\mathit{extractFrame}\ P = \langle A_P, \Psi_P\rangle$ **by** *fact*

from ‹A_P $\sharp*$ Q› ‹A_Q $\sharp*$ A_P› *FrQ* **have** A_P $\sharp*$ Ψ_Q
by(*force dest*: *extractFrameFreshChain*)

from ‹$\mathit{extractFrame}(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› *FrP FrQ* ‹$A_Q$ $\sharp*$ A_P› ‹A_P $\sharp*$ Ψ_Q› ‹A_Q $\sharp*$ Ψ_P›
have $\langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$ **by** *simp*
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› ‹A_Q $\sharp*$ A_P› **have** $\mathit{distinct}(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
ultimately obtain p **where** S: $\mathit{set}\ p \subseteq \mathit{set}(A_P@A_Q) \times \mathit{set}((p \cdot A_P)@(p \cdot A_Q))$ **and** *distinctPerm p*
and Ψeq: $\Psi_{PQ} = (p \cdot \Psi_P) \otimes (p \cdot \Psi_Q)$ **and** *Aeq*: $A_{PQ} = (p \cdot A_P)@(p \cdot A_Q)$
using ‹A_P $\sharp*$ A_{PQ}› ‹A_Q $\sharp*$ A_{PQ}› ‹*distinct* A_{PQ}›

by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q'$› S ‹$A_{PQ} \sharp* Q$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_{PQ} \sharp* M$› ‹$A_P \sharp* M$› ‹$A_Q \sharp* M$› *Aeq*
have $(p \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto M(\!|N|\!) \prec Q'$
by(*rule-tac inputPermFrame*) *auto*
with S ‹$A_{PQ} \sharp* \Psi$› ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› *Aeq* **have** $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto M(\!|N|\!) \prec Q'$
by(*simp add*: *eqvts*)
moreover from *FrP* **have** $(p \cdot \mathit{extractFrame}\ P) = p \cdot \langle A_P, \Psi_P\rangle$ **by** *simp*
with S ‹$A_{PQ} \sharp* P$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› *Aeq* **have** $\mathit{extractFrame}\ P = \langle (p \cdot A_P), p \cdot \Psi_P\rangle$
by(*simp add*: *eqvts*)
moreover from *FrQ* **have** $(p \cdot \mathit{extractFrame}\ Q) = p \cdot \langle A_Q, \Psi_Q\rangle$ **by** *simp*
with S ‹$A_{PQ} \sharp* Q$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› *Aeq* **have** $\mathit{extractFrame}\ Q = \langle (p \cdot A_Q), p \cdot \Psi_Q\rangle$
by(*simp add*: *eqvts*)
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› **have** $\mathit{distinct}(p \cdot A_P)$ **and** $\mathit{distinct}(p \cdot A_Q)$
by *simp*+
moreover from ‹$A_Q \sharp* A_P$› **have** $(p \cdot A_P) \sharp* (p \cdot A_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹$A_P \sharp* \Psi_Q$› **have** $(p \cdot A_P) \sharp* (p \cdot \Psi_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹$A_Q \sharp* \Psi_P$› **have** $(p \cdot A_Q) \sharp* (p \cdot \Psi_P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case* **using** ‹$A_{PQ} \sharp* \Psi$› ‹$A_{PQ} \sharp* P$› ‹$A_{PQ} \sharp* Q$› ‹$A_{PQ} \sharp* M$› *Aeq* Ψ*eq*
by(*rule-tac rPar2*) (*assumption* | *simp*)+
qed

lemma *parCasesOutputFrame*[*consumes 11*, *case-names cPar1 cPar2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and T :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec T$
and $xvec \sharp* \Psi$
and $xvec \sharp* P$
and $xvec \sharp* Q$
and $xvec \sharp* M$
and $\mathit{extractFrame}(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$
and $\mathit{distinct}\ A_{PQ}$
and $A_{PQ} \sharp* \Psi$

and $A_{PQ} \sharp* P$
and $A_{PQ} \sharp* Q$
and $A_{PQ} \sharp* M$
and *rPar1*: $\bigwedge P'\ A_P\ \Psi_P\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$;
distinct A_P; *distinct* A_Q; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* M$; $A_Q \sharp* \Psi$; $A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* M$;
$A_P \sharp* \Psi_Q$; $A_Q \sharp* \Psi_P$; $A_P \sharp* A_Q$; $A_{PQ} = A_P@A_Q$; $\Psi_{PQ} = \Psi_P \otimes \Psi_Q]\!] \Longrightarrow$ *Prop* $(P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';$ *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$;
distinct A_P; *distinct* A_Q; $A_P \sharp* \Psi$; $A_P \sharp* P$; $A_P \sharp* Q$; $A_P \sharp* M$; $A_Q \sharp* \Psi$; $A_Q \sharp* P$; $A_Q \sharp* Q$; $A_Q \sharp* M$;
$A_P \sharp* \Psi_Q$; $A_Q \sharp* \Psi_P$; $A_P \sharp* A_Q$; $A_{PQ} = A_P@A_Q$; $\Psi_{PQ} = \Psi_P \otimes \Psi_Q]\!] \Longrightarrow$ *Prop* $(P \parallel Q')$
shows *Prop T*
using *Trans* ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ P› ‹*xvec* $\sharp*$ Q› ‹*xvec* $\sharp*$ M›
proof(*induct rule*: *parOutputCases*[*of* - - - - - - - (A_{PQ}, Ψ_{PQ})])
case(*cPar1* P' A_Q Ψ_Q)
from ‹$A_Q \sharp* (A_{PQ}, \Psi_{PQ})$› **have** $A_Q \sharp* A_{PQ}$ **and** $A_Q \sharp* \Psi_{PQ}$ **by** *simp*+
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *distinct* A_P
$A_P \sharp* (P, Q, \Psi, M, A_Q, A_{PQ}, \Psi_Q)$
by(*rule freshFrame*)
hence $A_P \sharp* P$ **and** $A_P \sharp* Q$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* M$ **and** $A_P \sharp* A_Q$ **and** $A_P \sharp* A_{PQ}$ **and** $A_P \sharp* \Psi_Q$
by *simp*+

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*

from ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_Q$› *FrP* **have** $A_Q \sharp* \Psi_P$
by(*force dest*: *extractFrameFreshChain*)

from ‹*extractFrame*$(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› *FrP FrQ* ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›
have $\langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$ **by** *simp*
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› ‹$A_P \sharp* A_Q$› **have** *distinct*$(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
ultimately obtain p **where** S: *set* $p \subseteq$ *set*$(A_P@A_Q) \times$ *set*$((p \cdot A_P)@(p \cdot A_Q))$ **and** *distinctPerm* p
and Ψeq: $\Psi_{PQ} = (p \cdot \Psi_P) \otimes (p \cdot \Psi_Q)$ **and** Aeq: $A_{PQ} = (p \cdot A_P)@(p \cdot A_Q)$
using ‹$A_P \sharp* A_{PQ}$› ‹$A_Q \sharp* A_{PQ}$› ‹*distinct* A_{PQ}›
by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› S ‹$A_{PQ} \sharp* P$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_{PQ} \sharp* M$› ‹$A_P \sharp* M$› ‹$A_Q \sharp* M$› Aeq
have $(p \cdot (\Psi \otimes \Psi_Q)) \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule-tac outputPermFrame*) (*assumption* | *simp*)+

with S ‹A_{PQ} $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ› Aeq **have** $\Psi \otimes (p \cdot \Psi_Q) \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*simp add*: *eqvts*)
moreover from FrP **have** $(p \cdot extractFrame\ P) = p \cdot \langle A_P, \Psi_P\rangle$ **by** *simp*
with S ‹A_{PQ} $\sharp*$ P› ‹A_P $\sharp*$ P› ‹A_Q $\sharp*$ P› Aeq **have** $extractFrame\ P = \langle(p \cdot A_P), p \cdot \Psi_P\rangle$
by(*simp add*: *eqvts*)
moreover from FrQ **have** $(p \cdot extractFrame\ Q) = p \cdot \langle A_Q, \Psi_Q\rangle$ **by** *simp*
with S ‹A_{PQ} $\sharp*$ Q› ‹A_P $\sharp*$ Q› ‹A_Q $\sharp*$ Q› Aeq **have** $extractFrame\ Q = \langle(p \cdot A_Q), p \cdot \Psi_Q\rangle$
by(*simp add*: *eqvts*)
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› **have** $distinct(p \cdot A_P)$ **and** $distinct(p \cdot A_Q)$
by *simp*+
moreover from ‹A_P $\sharp*$ A_Q› **have** $(p \cdot A_P)$ $\sharp*$ $(p \cdot A_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹A_P $\sharp*$ Ψ_Q› **have** $(p \cdot A_P)$ $\sharp*$ $(p \cdot \Psi_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹A_Q $\sharp*$ Ψ_P› **have** $(p \cdot A_Q)$ $\sharp*$ $(p \cdot \Psi_P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case* **using** ‹A_{PQ} $\sharp*$ Ψ› ‹A_{PQ} $\sharp*$ P› ‹A_{PQ} $\sharp*$ Q› ‹A_{PQ} $\sharp*$ M› Aeq Ψeq
by(*rule-tac rPar1*) (*assumption* | *simp*)+
next
case(*cPar2* Q' A_P Ψ_P)
from ‹A_P $\sharp*$ (A_{PQ}, Ψ_{PQ})› **have** A_P $\sharp*$ A_{PQ} **and** A_P $\sharp*$ Ψ_{PQ} **by** *simp*+
obtain A_Q Ψ_Q **where** FrQ: $extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$ **and** $distinct\ A_Q$
A_Q $\sharp*$ $(P, Q, \Psi, M, A_P, A_{PQ}, \Psi_P)$
by(*rule freshFrame*)
hence A_Q $\sharp*$ P **and** A_Q $\sharp*$ Q **and** A_Q $\sharp*$ Ψ **and** A_Q $\sharp*$ M **and** A_Q $\sharp*$ A_P **and** A_Q $\sharp*$ A_{PQ} **and** A_Q $\sharp*$ Ψ_P
by *simp*+

have FrP: $extractFrame\ P = \langle A_P, \Psi_P\rangle$ **by** *fact*

from ‹A_P $\sharp*$ Q› ‹A_Q $\sharp*$ A_P› FrQ **have** A_P $\sharp*$ Ψ_Q
by(*force dest*: *extractFrameFreshChain*)

from ‹$extractFrame(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ}\rangle$› FrP FrQ ‹A_Q $\sharp*$ A_P› ‹A_P $\sharp*$ Ψ_Q› ‹A_Q $\sharp*$ Ψ_P›
have $\langle(A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle = \langle A_{PQ}, \Psi_{PQ}\rangle$ **by** *simp*
moreover from ‹*distinct* A_P› ‹*distinct* A_Q› ‹A_Q $\sharp*$ A_P› **have** $distinct(A_P@A_Q)$
by(*auto simp add*: *fresh-star-def fresh-def name-list-supp*)
ultimately obtain p **where** S: $set\ p \subseteq set(A_P@A_Q) \times set((p \cdot A_P)@(p \cdot A_Q))$ **and** $distinctPerm\ p$
and Ψeq: $\Psi_{PQ} = (p \cdot \Psi_P) \otimes (p \cdot \Psi_Q)$ **and** Aeq: $A_{PQ} = (p \cdot A_P)@(p \cdot A_Q)$
using ‹A_P $\sharp*$ A_{PQ}› ‹A_Q $\sharp*$ A_{PQ}› ‹*distinct* A_{PQ}›
by(*rule-tac frameChainEq'*) (*assumption* | *simp add*: *eqvts*)+

from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› S ‹$A_{PQ} \sharp{*}\ Q$› ‹$A_P \sharp{*}\ Q$› ‹$A_Q \sharp{*}\ Q$› ‹$A_{PQ} \sharp{*}\ M$› ‹$A_P \sharp{*}\ M$› ‹$A_Q \sharp{*}\ M$› Aeq
have $(p \cdot (\Psi \otimes \Psi_P)) \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
by(*rule-tac outputPermFrame*) (*assumption* | *simp*)+
with S ‹$A_{PQ} \sharp{*}\ \Psi$› ‹$A_P \sharp{*}\ \Psi$› ‹$A_Q \sharp{*}\ \Psi$› Aeq **have** $\Psi \otimes (p \cdot \Psi_P) \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
by(*simp add*: *eqvts*)
moreover from FrP **have** $(p \cdot extractFrame\ P) = p \cdot \langle A_P, \Psi_P\rangle$ **by** *simp*
with S ‹$A_{PQ} \sharp{*}\ P$› ‹$A_P \sharp{*}\ P$› ‹$A_Q \sharp{*}\ P$› Aeq **have** $extractFrame\ P = \langle (p \cdot A_P), p \cdot \Psi_P\rangle$
by(*simp add*: *eqvts*)
moreover from FrQ **have** $(p \cdot extractFrame\ Q) = p \cdot \langle A_Q, \Psi_Q\rangle$ **by** *simp*
with S ‹$A_{PQ} \sharp{*}\ Q$› ‹$A_P \sharp{*}\ Q$› ‹$A_Q \sharp{*}\ Q$› Aeq **have** $extractFrame\ Q = \langle (p \cdot A_Q), p \cdot \Psi_Q\rangle$
by(*simp add*: *eqvts*)
moreover from ‹$distinct\ A_P$› ‹$distinct\ A_Q$› **have** $distinct(p \cdot A_P)$ **and** $distinct(p \cdot A_Q)$
by *simp*+
moreover from ‹$A_Q \sharp{*}\ A_P$› **have** $(p \cdot A_P) \sharp{*} (p \cdot A_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹$A_P \sharp{*}\ \Psi_Q$› **have** $(p \cdot A_P) \sharp{*} (p \cdot \Psi_Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹$A_Q \sharp{*}\ \Psi_P$› **have** $(p \cdot A_Q) \sharp{*} (p \cdot \Psi_P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case* **using** ‹$A_{PQ} \sharp{*}\ \Psi$› ‹$A_{PQ} \sharp{*}\ P$› ‹$A_{PQ} \sharp{*}\ Q$› ‹$A_{PQ} \sharp{*}\ M$› Aeq Ψeq
by(*rule-tac rPar2*) (*assumption* | *simp*)+
qed

inductive $bangPred :: ('a, 'b, 'c)\ psi \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow bool$
where
aux1: $bangPred\ P\ (!P)$
| *aux2*: $bangPred\ P\ (P \parallel !P)$

lemma *bangInduct*[*consumes 1*, *case-names cPar1 cPar2 cComm1 cComm2 cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and Rs :: $('a, 'b, 'c)\ residual$
and $Prop :: 'd{::}fs\text{-}name \Rightarrow 'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow ('a, 'b, 'c)\ residual \Rightarrow bool$
and C :: $'d$

assumes $\Psi \rhd !P \longmapsto Rs$
and *rPar1*: $\bigwedge\alpha\ P'\ C.\ [\![\Psi \rhd P \longmapsto \alpha \prec P';\ bn\ \alpha \sharp{*} \Psi;\ bn\ \alpha \sharp{*} P;\ bn\ \alpha \sharp{*} subject\ \alpha;\ bn\ \alpha \sharp{*} C;\ distinct(bn\ \alpha)]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\alpha \prec (P' \parallel !P))$
and *rPar2*: $\bigwedge\alpha\ P'\ C.\ [\![\Psi \rhd !P \longmapsto \alpha \prec P';\ bn\ \alpha \sharp{*} \Psi;\ bn\ \alpha \sharp{*} P;\ bn\ \alpha \sharp{*} subject\ \alpha;\ bn\ \alpha \sharp{*} C;\ distinct(bn\ \alpha);$
$\bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (\alpha \prec P')]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\alpha \prec (P \parallel P'))$

and *rComm1*: $\bigwedge M\ N\ P'\ K\ xvec\ P''\ C.\ [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ \Psi \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'');\ \Psi \vdash M \leftrightarrow K;$
$xvec \sharp{*}\ \Psi;\ xvec \sharp{*}\ P;\ xvec \sharp{*}\ M;\ xvec \sharp{*}\ K;\ xvec \sharp{*}\ C;\ distinct\ xvec]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P''))$
and *rComm2*: $\bigwedge M\ xvec\ N\ P'\ K\ P''\ C.\ [\![\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ \Psi \rhd !P \longmapsto K(\!|N|\!) \prec P'';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (K(\!|N|\!) \prec P'');\ \Psi \vdash M \leftrightarrow K;$
$xvec \sharp{*}\ \Psi;\ xvec \sharp{*}\ P;\ xvec \sharp{*}\ M;\ xvec \sharp{*}\ K;\ xvec \sharp{*}\ C;\ distinct\ xvec]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P''))$
and *rBang*: $\bigwedge Rs\ C.\ [\![\Psi \rhd P \parallel !P \longmapsto Rs;\ \bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ Rs;\ guarded\ P]\!] \Longrightarrow Prop\ C\ \Psi\ (!P)\ Rs$
shows $Prop\ C\ \Psi\ (!P)\ Rs$
proof −
from ‹$\Psi \rhd !P \longmapsto Rs$› **have** *guarded P*
by(*nominal-induct* Ψ *P*==!*P Rs rule*: *semantics.strong-induct*) (*auto simp add*: *psi.inject*)
{
fix Q :: ('*a*, '*b*, '*c*) *psi*
and Ψ' :: '*b*

assume $\Psi' \rhd Q \longmapsto Rs$
and *guarded Q*
and *bangPred P Q*
and $\Psi \simeq \Psi'$

hence *Prop C* Ψ *Q Rs* **using** *rPar1 rPar1 rPar2 rPar2 rComm1 rComm2 rBang*
proof(*nominal-induct avoiding*: Ψ *C rule*: *semantics.strong-induct*)
case(*cInput* Ψ' *M K xvec N Tvec Q* Ψ *C*)
thus *?case* **by** − (*ind-cases bangPred P* ($M(\!|\lambda{*}xvec\ N|\!).Q$))
next
case(*Output* Ψ' *M K N Q* Ψ *C*)
thus *?case* **by** − (*ind-cases bangPred P* ($M\langle N\rangle.Q$))
next
case(*Case* Ψ' *Q Rs* φ *Cs* Ψ *C*)
thus *?case* **by** − (*ind-cases bangPred P* (*Cases Cs*))
next
case(*cPar1* $\Psi'\ \Psi_R\ Q\ \alpha\ P'\ R\ A_R\ \Psi\ C$)
have *rPar1*: $\bigwedge \alpha\ P'\ C.\ [\![\Psi \rhd P \longmapsto \alpha \prec P';\ bn\ \alpha \sharp{*}\ \Psi;\ bn\ \alpha \sharp{*}\ P;\ bn\ \alpha \sharp{*}\ subject\ \alpha;\ bn\ \alpha \sharp{*}\ C;\ distinct(bn\ \alpha)]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\alpha \prec (P' \parallel !P))$
by *fact*
from ‹*bangPred P* ($Q \parallel R$)› **have** $Q = P$ **and** $R = !P$
by − (*ind-cases bangPred P* ($Q \parallel R$), *auto simp add*: *psi.inject*)+
from ‹$R = !P$› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** $A_R = []$ **and** $\Psi_R = \mathbf{1}$ **by** *auto*
from ‹$\Psi' \otimes \Psi_R \rhd Q \longmapsto \alpha \prec P'$› ‹$Q = P$› ‹$\Psi \simeq \Psi'$› ‹$\Psi_R = \mathbf{1}$› **have** $\Psi \rhd P \longmapsto \alpha \prec P'$
by(*metis statEqTransition Identity AssertionStatEqSym*)
hence *Prop C* Ψ ($P \parallel !P$) ($\alpha \prec (P' \parallel !P)$) **using** ‹$bn\ \alpha \sharp{*}\ \Psi$› ‹$bn\ \alpha \sharp{*}\ Q$›

‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$bn\ \alpha\ \sharp*\ C$› ‹$Q = P$› ‹$distinct(bn\ \alpha)$›
by(*rule-tac rPar1*) *auto*
with ‹$R = !P$› ‹$Q = P$› **show** *?case* **by** *simp*
next
case(*cPar2* $\Psi'\ \Psi_P\ R\ \alpha\ P'\ Q\ A_P\ \Psi\ C$)
have *rPar2*: $\bigwedge\alpha\ P'\ C.\ [\![\Psi \rhd !P \longmapsto\alpha \prec P';\ bn\ \alpha\ \sharp*\ \Psi;\ bn\ \alpha\ \sharp*\ P;\ bn\ \alpha\ \sharp*\ subject\ \alpha;\ bn\ \alpha\ \sharp*\ C;\ distinct(bn\ \alpha);$
$\bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (\alpha \prec P')]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\alpha \prec (P \parallel P'))$
by *fact*
from ‹$bangPred\ P\ (Q \parallel R)$› **have** $Q = P$ **and** $R = !P$
by − (*ind-cases bangPred* $P\ (Q \parallel R)$, *auto simp add*: *psi.inject*)+
from ‹$Q = P$› ‹$extractFrame\ Q = \langle A_P, \Psi_P\rangle$› ‹$guarded\ P$› **have** $\Psi_P \simeq \mathbf{1}$ **and** $supp\ \Psi_P = (\{\}::name\ set)$
by(*blast dest*: *guardedStatEq*)+
from ‹$\Psi' \otimes \Psi_P \rhd R \longmapsto\alpha \prec P'$› ‹$R = !P$› ‹$\Psi \simeq \Psi'$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \rhd !P \longmapsto\alpha \prec P'$
by(*metis statEqTransition Identity Composition Commutativity AssertionStatEqSym*)
moreover
{
fix C
have $bangPred\ P\ (!P)$ **by**(*rule aux1*)
moreover from ‹$\Psi \simeq \Psi'$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \simeq \Psi' \otimes \Psi_P$ **by**(*metis Composition Identity Commutativity AssertionStatEqSym AssertionStatEqTrans*)
ultimately have $\bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (\alpha \prec P')$ **using** *cPar2* ‹$R = !P$› ‹$guarded\ P$› **by** *simp*
}
ultimately have $Prop\ C\ \Psi\ (P \parallel !P)\ (\alpha \prec (P \parallel P'))$ **using** ‹$bn\ \alpha\ \sharp*\ \Psi$› ‹$bn\ \alpha\ \sharp*\ Q$› ‹$bn\ \alpha\ \sharp*\ subject\ \alpha$› ‹$bn\ \alpha\ \sharp*\ C$› ‹$Q = P$› ‹$distinct(bn\ \alpha)$›
by(*rule-tac rPar2*) *auto*
with ‹$R = !P$› ‹$Q = P$› **show** *?case* **by** *simp*
next
case(*cComm1* $\Psi'\ \Psi_R\ Q\ M\ N\ P'\ A_P\ \Psi_P\ R\ K\ xvec\ P''\ A_R\ \Psi\ C$)
have *rComm1*: $\bigwedge M\ N\ P'\ K\ xvec\ P''\ C.\ [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ \Psi \rhd !P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (K(\!|\nu*xvec|\!)\langle N\rangle \prec P'');\ \Psi \vdash M \leftrightarrow K;$
$xvec\ \sharp*\ \Psi;\ xvec\ \sharp*\ P;\ xvec\ \sharp*\ M;\ xvec\ \sharp*\ K;\ xvec\ \sharp*\ C;\ distinct\ xvec]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\tau \prec (\!|\nu*xvec|\!)(P' \parallel P''))$
by *fact*
from ‹$bangPred\ P\ (Q \parallel R)$› **have** $Q = P$ **and** $R = !P$
by − (*ind-cases bangPred* $P\ (Q \parallel R)$, *auto simp add*: *psi.inject*)+
from ‹$R = !P$› ‹$extractFrame\ R = \langle A_R, \Psi_R\rangle$› **have** $A_R = []$ **and** $\Psi_R = \mathbf{1}$ **by** *auto*
from ‹$\Psi' \otimes \Psi_R \rhd Q \longmapsto M(\!|N|\!) \prec P'$› ‹$Q = P$› ‹$\Psi \simeq \Psi'$› ‹$\Psi_R = \mathbf{1}$› **have** $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
by(*metis statEqTransition Identity AssertionStatEqSym*)
moreover from ‹$Q = P$› ‹$extractFrame\ Q = \langle A_P, \Psi_P\rangle$› ‹$guarded\ P$› **have** $\Psi_P \simeq \mathbf{1}$ **and** $supp\ \Psi_P = (\{\}::name\ set)$

by(*blast dest*: *guardedStatEq*)+
moreover from ‹$\Psi' \otimes \Psi_P \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$› ‹$R = !P$› ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi \simeq \Psi'$› **have** $\Psi \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$
by(*metis statEqTransition Identity Composition Commutativity AssertionStatEqSym*)
moreover
{
fix C
have *bangPred* P $(!P)$ **by**(*rule aux1*)
moreover from ‹$\Psi \simeq \Psi'$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \simeq \Psi' \otimes \Psi_P$ **by**(*metis Composition Identity Commutativity AssertionStatEqSym AssertionStatEqTrans*)
ultimately have $\bigwedge C.$ *Prop* C Ψ $(!P)$ $(K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'')$ **using** *cComm1* ‹$R = !P$› ‹*guarded* P› **by** *simp*
}
moreover from ‹$\Psi' \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi \simeq \Psi'$› ‹$\Psi_R = \mathbf{1}$› **have** $\Psi \vdash M \leftrightarrow K$
by(*metis statEqEnt Identity Composition Commutativity AssertionStatEqSym*)
ultimately have *Prop* C Ψ $(P \parallel !P)$ $(\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P''))$ **using** ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ Q› ‹*xvec* $\sharp*$ M› ‹*xvec* $\sharp*$ K› ‹*xvec* $\sharp*$ C› ‹$Q = P$› ‹*distinct xvec*›
by(*rule-tac rComm1*) (*assumption* | *auto*)+
with ‹$R = !P$› ‹$Q = P$› **show** *?case* **by** *simp*
next
case(*cComm2* Ψ' Ψ_R Q M *xvec* N P' A_P Ψ_P R K P'' A_R Ψ C)
have *rComm2*: $\bigwedge M$ *xvec* N P' K P'' $C.$ $[\![\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; $\Psi \rhd !P \longmapsto K(\!|N|\!) \prec P''$; $\bigwedge C.$ *Prop* C Ψ $(!P)$ $(K(\!|N|\!) \prec P'')$; $\Psi \vdash M \leftrightarrow K$;
xvec $\sharp*$ Ψ; *xvec* $\sharp*$ P; *xvec* $\sharp*$ M; *xvec* $\sharp*$ K; *xvec* $\sharp*$ C; *distinct xvec*$]\!] \Longrightarrow$ *Prop* C Ψ $(P \parallel !P)$ $(\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P''))$
by *fact*
from ‹*bangPred* P $(Q \parallel R)$› **have** $Q = P$ **and** $R = !P$
by − (*ind-cases bangPred* P $(Q \parallel R)$, *auto simp add*: *psi.inject*)+
from ‹$R = !P$› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** $A_R = []$ **and** $\Psi_R = \mathbf{1}$ **by** *auto*
from ‹$\Psi' \otimes \Psi_R \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹$Q = P$› ‹$\Psi \simeq \Psi'$› ‹$\Psi_R = \mathbf{1}$› **have** $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*metis statEqTransition Identity AssertionStatEqSym*)
moreover from ‹$Q = P$› ‹*extractFrame* $Q = \langle A_P, \Psi_P\rangle$› ‹*guarded* P› **have** $\Psi_P \simeq \mathbf{1}$ **and** *supp* $\Psi_P = (\{\}::$*name set*$)$
by(*blast dest*: *guardedStatEq*)+
moreover from ‹$\Psi' \otimes \Psi_P \rhd R \longmapsto K(\!|N|\!) \prec P''$› ‹$R = !P$› ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi \simeq \Psi'$› **have** $\Psi \rhd !P \longmapsto K(\!|N|\!) \prec P''$
by(*metis statEqTransition Identity Composition Commutativity AssertionStatEqSym*)
moreover
{
fix C
have *bangPred* P $(!P)$ **by**(*rule aux1*)
moreover from ‹$\Psi \simeq \Psi'$› ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \simeq \Psi' \otimes \Psi_P$ **by**(*metis Composition Identity Commutativity AssertionStatEqSym AssertionStatEqTrans*)

ultimately have $\bigwedge C.$ *Prop* C Ψ $(!P)$ $(K\langle\!|N|\!\rangle \prec P'')$ **using** *cCom2* ‹$R = !P$› ‹*guarded* P› **by** *simp*
}
moreover from ‹$\Psi' \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi \simeq \Psi'$› ‹$\Psi_R = \mathbf{1}$› **have** $\Psi \vdash M \leftrightarrow K$
by(*metis statEqEnt Identity Composition Commutativity AssertionStatEqSym*)
ultimately have *Prop* C Ψ $(P \parallel !P)$ $(\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P''))$ **using** ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ Q› ‹*xvec* $\sharp*$ M› ‹*xvec* $\sharp*$ K› ‹*xvec* $\sharp*$ C› ‹$Q = P$› ‹*distinct xvec*›
by(*rule-tac rComm2*) (*assumption* | *auto*)+
with ‹$R = !P$› ‹$Q = P$› **show** *?case* **by** *simp*
next
case(*cOpen* Ψ Q M *xvec yvec* N P' x C)
thus *?case* **by** − (*ind-cases bangPred* P $((\!|\nu x|\!)Q)$)
next
case(*cScope* Ψ Q α P' x C)
thus *?case* **by** − (*ind-cases bangPred* P $((\!|\nu x|\!)Q)$)
next
case(*Bang* Ψ' Q *Rs* Ψ C)
have *rBang*: $\bigwedge Rs\ C.\ [\![\Psi \rhd P \parallel !P \longmapsto Rs;\ \bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ Rs;\ guarded\ P]\!] \Longrightarrow Prop\ C\ \Psi\ (!P)\ Rs$
by *fact*
from ‹*bangPred* P $(!Q)$› **have** $P = Q$
by − (*ind-cases bangPred* P $(!Q)$, *auto simp add*: *psi.inject*)
with ‹$\Psi' \rhd Q \parallel !Q \longmapsto Rs$› ‹$\Psi \simeq \Psi'$› **have** $\Psi \rhd P \parallel !P \longmapsto Rs$ **by**(*metis statEqTransition AssertionStatEqSym*)
moreover
{
fix C
have *bangPred* P $(P \parallel !P)$ **by**(*rule aux2*)
with *Bang* ‹$P = Q$› **have** $\bigwedge C.$ *Prop* C Ψ $(P \parallel !P)$ *Rs* **by** *simp*
}
moreover from ‹*guarded* Q› ‹$P = Q$› **have** *guarded* P **by** *simp*
ultimately have *Prop* C Ψ $(!P)$ *Rs* **by**(*rule rBang*)
with ‹$P = Q$› **show** *?case* **by** *simp*
qed
}
with ‹*guarded* P› ‹$\Psi \rhd !P \longmapsto Rs$›
show *?thesis* **by**(*force intro*: *aux1*)
qed

lemma *bangInputInduct*[*consumes 1*, *case-names cPar1 cPar2 cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and *Prop* :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow 'a \Rightarrow 'a \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool*

assumes $\Psi \rhd !P \longmapsto M(\!|N|\!) \prec P'$
and *rPar1*: $\bigwedge P'.\ \Psi \rhd P \longmapsto M(\!|N|\!) \prec P' \Longrightarrow Prop\ \Psi\ (P \parallel !P)\ M\ N\ (P' \parallel !P)$
and *rPar2*: $\bigwedge P'.\ [\![\Psi \rhd !P \longmapsto M(\!|N|\!) \prec P';\ Prop\ \Psi\ (!P)\ M\ N\ P']\!] \Longrightarrow Prop\ \Psi\ (P \parallel !P)\ M\ N\ (P \parallel P')$
and *rBang*: $\bigwedge P'.\ [\![\Psi \rhd P \parallel !P \longmapsto M(\!|N|\!) \prec P';\ Prop\ \Psi\ (P \parallel !P)\ M\ N\ P';\ guarded\ P]\!] \Longrightarrow Prop\ \Psi\ (!P)\ M\ N\ P'$
shows $Prop\ \Psi\ (!P)\ M\ N\ P'$
using ‹$\Psi \rhd !P \longmapsto M(\!|N|\!) \prec P'$›
by(*nominal-induct* Ψ P *Rs*==$M(\!|N|\!) \prec P'$ *arbitrary*: P' *rule*: *bangInduct*)
(*auto simp add*: *residualInject intro*: *rPar1 rPar2 rBang*)

lemma *bangOutputInduct*[*consumes 1*, *case-names cPar1 cPar2 cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and *Prop* :: $'d$::*fs-name* $\Rightarrow$ $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ $'a \Rightarrow ('a, 'b, 'c)$ *boundOutput* $\Rightarrow$ *bool*
and C :: $'d$

assumes $\Psi \rhd !P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and *rPar1*: $\bigwedge xvec\ N\ P'\ C.\ [\![\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ xvec \sharp{*} \Psi;\ xvec \sharp{*} P;\ xvec \sharp{*} M;\ xvec \sharp{*} C;\ distinct\ xvec]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ M\ ((\!|\nu{*}xvec|\!)N \prec' (P' \parallel !P))$
and *rPar2*: $\bigwedge xvec\ N\ P'\ C.\ [\![\Psi \rhd !P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ M\ ((\!|\nu{*}xvec|\!)N \prec' P');\ xvec \sharp{*} \Psi;\ xvec \sharp{*} P;\ xvec \sharp{*} M;\ xvec \sharp{*} C;\ distinct\ xvec]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel !P)\ M\ ((\!|\nu{*}xvec|\!)N \prec' (P \parallel P'))$
and *rBang*: $\bigwedge B\ C.\ [\![\Psi \rhd P \parallel !P \longmapsto (ROut\ M\ B);\ \bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ M\ B;\ guarded\ P]\!] \Longrightarrow Prop\ C\ \Psi\ (!P)\ M\ B$

shows $Prop\ C\ \Psi\ (!P)\ M\ ((\!|\nu{*}xvec|\!)N \prec' P')$
using ‹$\Psi \rhd !P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$›
apply(*auto simp add*: *residualInject*)
by(*nominal-induct* Ψ P *Rs*==$ROut\ M\ ((\!|\nu{*}xvec|\!)N \prec' P')$ *avoiding*: C *arbitrary*: $xvec\ N\ P'$ *rule*: *bangInduct*)
(*force simp add*: *residualInject intro*: *rPar1 rPar2 rBang*)+

lemma *bangTauInduct*[*consumes 1*, *case-names cPar1 cPar2 cComm1 cComm2 cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and *Prop* :: $'d$::*fs-name* $\Rightarrow$ $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool*
and C :: $'d$

assumes $\Psi \rhd !P \longmapsto\tau \prec P'$
and *rPar1*: $\bigwedge P'\ C.\ \Psi \rhd P \longmapsto\tau \prec P' \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (P' \parallel !P)$
and *rPar2*: $\bigwedge P'\ C.\ [\![\Psi \rhd !P \longmapsto\tau \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ P']\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (P \parallel P')$
and *rComm1*: $\bigwedge M\ N\ P'\ K\ xvec\ P''\ C.\ [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ \Psi \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'';\ \Psi \vdash M \leftrightarrow K;$
$xvec \sharp* \Psi;\ xvec \sharp* P;\ xvec \sharp* M;\ xvec \sharp* K;\ xvec \sharp* C]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ ((\!|\nu{*}xvec|\!)(P' \parallel P''))$
and *rComm2*: $\bigwedge M\ N\ P'\ K\ xvec\ P''\ C.\ [\![\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P';\ \Psi \rhd !P \longmapsto K(\!|N|\!) \prec P'';\ \Psi \vdash M \leftrightarrow K;$
$xvec \sharp* \Psi;\ xvec \sharp* P;\ xvec \sharp* M;\ xvec \sharp* K;\ xvec \sharp* C]\!] \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ ((\!|\nu{*}xvec|\!)(P' \parallel P''))$
and *rBang*: $\bigwedge P'\ C.\ [\![\Psi \rhd P \parallel !P \longmapsto\tau \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ P';\ guarded\ P]\!] \Longrightarrow Prop\ C\ \Psi\ (!P)\ P'$

shows $Prop\ C\ \Psi\ (!P)\ P'$
using ‹$\Psi \rhd !P \longmapsto\tau \prec P'$›
by(*nominal-induct* $\Psi\ P\ Rs{==}\tau \prec P'$ *avoiding*: C *arbitrary*: P' *rule*: *bangInduct*)
(*auto simp add*: *residualInject intro*: *rPar1 rPar2 rComm1 rComm2 rBang*)

lemma *bangInduct'*[*consumes 2*, *case-names cAlpha cPar1 cPar2 cComm1 cComm2 cBang*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and α :: $'a\ action$
and P' :: $('a, 'b, 'c)\ psi$
and $Prop$ :: $'d{::}fs\text{-}name \Rightarrow 'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow 'a\ action \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow bool$
and C :: $'d{::}fs\text{-}name$

assumes $\Psi \rhd !P \longmapsto\alpha \prec P'$
and $bn\ \alpha \sharp* subject\ \alpha$
and *rAlpha*: $\bigwedge \alpha\ P'\ p\ C.\ [\![bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* C;$
$set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha));\ distinctPerm\ p;$
$bn(p \cdot \alpha) \sharp* \alpha;\ bn(p \cdot \alpha) \sharp* P';\ Prop\ C\ \Psi\ (P \parallel !P)\ \alpha\ P']\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel !P)\ (p \cdot \alpha)\ (p \cdot P')$
and *rPar1*: $\bigwedge \alpha\ P'\ C.$
$[\![\Psi \rhd P \longmapsto\alpha \prec P';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* C;\ distinct(bn\ \alpha)]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel !P)\ \alpha\ (P' \parallel !P)$
and *rPar2*: $\bigwedge \alpha\ P'\ C.$
$[\![\Psi \rhd !P \longmapsto\alpha \prec P';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ \alpha\ P';$
$bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* C;\ distinct(bn\ \alpha)]\!] \Longrightarrow$
$Prop\ C\ \Psi\ (P \parallel !P)\ \alpha\ (P \parallel P')$
and *rComm1*: $\bigwedge M\ N\ P'\ K\ xvec\ P''\ C.\ [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P';\ \Psi \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'';\ \bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (K(\!|\nu{*}xvec|\!)\langle N\rangle)\ P'';\ \Psi \vdash M \leftrightarrow K;$

$xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* C$; $distinct\ xvec\rrbracket \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\tau)\ ((\!|\nu*xvec|\!)(P' \parallel P''))$
and *rComm2*: $\bigwedge M\ xvec\ N\ P'\ K\ P''\ C.\ \llbracket \Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$; $\Psi \rhd !P \longmapsto K(\!|N|\!) \prec P''$; $\bigwedge C.\ Prop\ C\ \Psi\ (!P)\ (K(\!|N|\!))\ P''$; $\Psi \vdash M \leftrightarrow K$; $xvec \sharp* \Psi$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* C$; $distinct\ xvec\rrbracket \Longrightarrow Prop\ C\ \Psi\ (P \parallel !P)\ (\tau)\ ((\!|\nu*xvec|\!)(P' \parallel P''))$
and *rBang*: $\bigwedge \alpha\ P'\ C.$
$\llbracket \Psi \rhd P \parallel !P \longmapsto \alpha \prec P'$; $guarded\ P$; $\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ \alpha\ P'$; $guarded\ P$; $distinct(bn\ \alpha)\rrbracket \Longrightarrow$
$Prop\ C\ \Psi\ (!P)\ \alpha\ P'$
shows $Prop\ C\ \Psi\ (!P)\ \alpha\ P'$
proof −
from ‹$\Psi \rhd !P \longmapsto \alpha \prec P'$› **have** $distinct(bn\ \alpha)$ **by**(*rule boundOutputDistinct*)
with ‹$\Psi \rhd !P \longmapsto \alpha \prec P'$› ‹$bn\ \alpha \sharp* subject\ \alpha$› **show** *?thesis*
proof(*nominal-induct* $\Psi\ P\ Rs==\alpha \prec P'$ *avoiding*: $C\ \alpha\ P'$ *rule*: *bangInduct*)
case(*cPar1* $\alpha\ P'\ C\ \alpha'\ P''$)
note ‹$\alpha \prec (P' \parallel !P) = \alpha' \prec P''$›
moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* bn\ \alpha'$ **by** *simp*
moreover note ‹$distinct(bn\ \alpha)$› ‹$distinct(bn\ \alpha')$›
moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› **have** $bn\ \alpha \sharp* (\alpha \prec P' \parallel !P)$ **by** *simp*
moreover from ‹$bn\ \alpha' \sharp* subject\ \alpha'$› **have** $bn\ \alpha' \sharp* (\alpha' \prec P'')$ **by** *simp*
ultimately obtain p **where** S: $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** $distinctPerm\ p$ **and** $\alpha' = p \cdot \alpha$
and $P'eq$: $P'' = p \cdot (P' \parallel !P)$ **and** $bn(p \cdot \alpha) \sharp* \alpha$ **and** $bn(p \cdot \alpha) \sharp* (P' \parallel !P)$
by(*rule-tac residualEq*)

from ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* C$› ‹$distinct(bn\ \alpha)$›
have $Prop\ C\ \Psi\ (P \parallel !P)\ \alpha\ (P' \parallel !P)$
by(*rule rPar1*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* C$› S ‹$distinctPerm\ p$› ‹$bn\ \alpha \sharp* \alpha'$› ‹$bn\ \alpha \sharp* P''$› ‹$\alpha' = (p \cdot \alpha)$› $P'eq$ ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn(p \cdot \alpha) \sharp* (P' \parallel !P)$›
have $Prop\ C\ \Psi\ (P \parallel !P)\ (p \cdot \alpha)\ (p \cdot (P' \parallel !P))$
by(*rule-tac rAlpha*)
with $P'eq$ ‹$\alpha' = p \cdot \alpha$› ‹$distinctPerm\ p$› **show** *?case* **by** *simp*
next
case(*cPar2* $\alpha\ P'\ C\ \alpha'\ P''$)
note ‹$\alpha \prec (P \parallel P') = \alpha' \prec P''$›
moreover from ‹$bn\ \alpha \sharp* \alpha'$› **have** $bn\ \alpha \sharp* bn\ \alpha'$ **by** *simp*
moreover note ‹$distinct(bn\ \alpha)$› ‹$distinct(bn\ \alpha')$›
moreover from ‹$bn\ \alpha \sharp* subject\ \alpha$› **have** $bn\ \alpha \sharp* (\alpha \prec P \parallel P')$ **by** *simp*
moreover from ‹$bn\ \alpha' \sharp* subject\ \alpha'$› **have** $bn\ \alpha' \sharp* (\alpha' \prec P'')$ **by** *simp*
ultimately obtain p **where** S: $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **and** $distinctPerm\ p$ **and** $\alpha' = p \cdot \alpha$
and $P'eq$: $P'' = p \cdot (P \parallel P')$ **and** $bn(p \cdot \alpha) \sharp* \alpha$ **and** $bn(p \cdot \alpha) \sharp* (P \parallel P')$

by(*rule-tac residualEq*)

note ‹$\Psi \rhd !P \longmapsto\alpha \prec P'$›
moreover from ‹*bn* α $\sharp*$ *subject* α› ‹*distinct*(*bn* α)› **have** $\bigwedge C$. *Prop* C Ψ $(!P)$ α P' **by**(*rule-tac cPar2*) *auto*
moreover note ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ P› ‹*bn* α $\sharp*$ *subject* α› ‹*bn* α $\sharp*$ C› ‹*distinct*(*bn* α)›
ultimately have *Prop* C Ψ $(P \parallel !P)$ α $(P \parallel P')$
by(*rule rPar2*)
with ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ P› ‹*bn* α $\sharp*$ *subject* α› ‹*bn* α $\sharp*$ C› S ‹*distinctPerm* p› ‹*bn* α $\sharp*$ α'› ‹*bn* α $\sharp*$ P''› ‹$\alpha' = (p \cdot \alpha)$› $P'eq$ ‹*bn*$(p \cdot \alpha)$ $\sharp*$ α› ‹*bn*$(p \cdot \alpha)$ $\sharp*$ $(P \parallel P')$›
have *Prop* C Ψ $(P \parallel !P)$ $(p \cdot \alpha)$ $(p \cdot (P \parallel P'))$
by(*rule-tac rAlpha*)
with $P'eq$ ‹$\alpha' = p \cdot \alpha$› **show** *?case* **by** *simp*
next
case(*cComm1 M N P′ K xvec P″ C α P‴*)
hence *Prop* C Ψ $(P \parallel !P)$ (τ) $((\!|\nu{*}xvec|\!)(P' \parallel P''))$
by(*rule-tac rComm1*) (*assumption* | *simp*)+
thus *?case* **using** ‹$\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P'') = \alpha \prec P'''$›
by(*simp add*: *residualInject*)
next
case(*cComm2 M xvec N P′ K P″ C α P‴*)
hence *Prop* C Ψ $(P \parallel !P)$ (τ) $((\!|\nu{*}xvec|\!)(P' \parallel P''))$
by(*rule-tac rComm2*) (*assumption* | *simp*)+
thus *?case* **using** ‹$\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel P'') = \alpha \prec P'''$›
by(*simp add*: *residualInject*)
next
case(*cBang C α P′*)
thus *?case* **by**(*auto intro*: *rBang*)
qed
qed

lemma *comm1Aux*:
fixes Ψ :: $'b$
and Ψ_Q :: $'b$
and R :: $('a, 'b, 'c)$ *psi*
and K :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and R' :: $('a, 'b, 'c)$ *psi*
and A_R :: *name list*
and Ψ_R :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and L :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$

and A_Q :: *name list*

assumes *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$
and *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$
and *PTrans*: $\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|L|\!) \prec P'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$
and *PeqQ*: $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
and *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
and *distinct* A_P
and *distinct* A_R
and $A_R \sharp* A_P$
and $A_R \sharp* A_Q$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$
and $A_R \sharp* R$
and $A_R \sharp* K$
and $A_P \sharp* \Psi$
and $A_P \sharp* R$
and $A_P \sharp* P$
and $A_P \sharp* M$
and $A_Q \sharp* R$
and $A_Q \sharp* M$

obtains K' **where** $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$ **and** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $A_R \sharp* K'$
proof −
from ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* A_P$› ‹$A_R \sharp* A_Q$› *FrP FrQ* **have** $A_R \sharp* \Psi_P$ **and** $A_R \sharp* \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+
assume *Assumptions*: $\bigwedge K'.\ [\![\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'; \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'; A_R \sharp* K']\!] \Longrightarrow$ *thesis*
{
fix Ψ' ::*'b*
and *zvec* ::*name list*

assume *A*: $\Psi \otimes \Psi_Q \simeq \Psi'$
assume $\Psi' \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$
hence $\Psi' \rhd R \longmapsto ROut\ K\ ((\!|\nu{*}xvec|\!)N \prec' R')$ **by**(*simp add*: *residualInject*)
moreover note *FrR* ‹*distinct* A_R› *PTrans*
moreover from ‹$\Psi' \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$› **have** *distinct xvec* **by**(*auto dest*: *boundOutputDistinct*)
moreover assume $\Psi' \otimes \Psi_R \vdash M \leftrightarrow K$ **and** $\langle A_Q, \Psi' \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
and $A_R \sharp* zvec$ **and** $A_P \sharp* zvec$ **and** $zvec \sharp* R$ **and** $zvec \sharp* P$
and $A_R \sharp* \Psi'$
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R') \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge zvec \sharp* K' \wedge A_R \sharp* K'$

using *FrP* ‹A_P ♯* R› ‹A_Q ♯* R› ‹A_P ♯* Ψ› ‹A_P ♯* P› ‹*distinct* A_P› ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* R›
‹A_P ♯* M› ‹A_Q ♯* M› ‹A_R ♯* K› ‹A_R ♯* A_P› ‹A_R ♯* A_Q› ‹A_R ♯* Ψ_P›
proof(*nominal-induct* Ψ' R K B==⦇ν*xvec⦈N $\prec'$ R' A_R Ψ_R *avoiding*: Ψ P A_P Ψ_P A_Q *zvec xvec* N R' *arbitrary*: M *rule*: *outputFrameInduct*)
case(*cAlpha* Ψ' R K A_R Ψ_R p Ψ P A_P Ψ_P A_Q *zvec xvec* N R' M)
have S: *set* $p \subseteq$ *set* $A_R \times$ *set* $(p \cdot A_R)$ **by** *fact*
from ‹$\Psi' \otimes (p \cdot \Psi_R) \vdash M \leftrightarrow K$› **have** $(p \cdot (\Psi' \otimes (p \cdot \Psi_R))) \vdash (p \cdot M) \leftrightarrow (p \cdot K)$
by(*rule chanEqClosed*)
with ‹A_R ♯* Ψ'› ‹$(p \cdot A_R)$ ♯* Ψ'› ‹A_R ♯* K› ‹$(p \cdot A_R)$ ♯* K› S ‹*distinctPerm* p›
have $\Psi' \otimes \Psi_R \vdash (p \cdot M) \leftrightarrow K$ **by**(*simp add*: *eqvts*)
moreover from ‹$\Psi \otimes (p \cdot \Psi_R) \rhd P \longmapsto M$⦇$L$⦈ $\prec P'$› S ‹A_R ♯* P› ‹$(p \cdot A_R)$ ♯* P› **have** $(p \cdot (\Psi \otimes (p \cdot \Psi_R))) \rhd P \longmapsto (p \cdot M)$⦇$L$⦈ $\prec P'$
by(*rule-tac inputPermFrameSubject*) *auto*
with ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› S ‹*distinctPerm* p› **have** $\Psi \otimes \Psi_R \rhd P \longmapsto (p \cdot M)$⦇$L$⦈ $\prec P'$
by(*simp add*: *eqvts*)
moreover from ‹A_P ♯* M› **have** $(p \cdot A_P)$ ♯* $(p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_R ♯* A_P› ‹$(p \cdot A_R)$ ♯* A_P› S **have** A_P ♯* $(p \cdot M)$ **by** *simp*
moreover from ‹A_Q ♯* M› **have** $(p \cdot A_Q)$ ♯* $(p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_R ♯* A_Q› ‹$(p \cdot A_R)$ ♯* A_Q› S **have** A_Q ♯* $(p \cdot M)$ **by** *simp*
moreover from ‹$\langle A_Q, \Psi' \otimes (p \cdot \Psi_R)\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes (p \cdot \Psi_R)\rangle$ ›
have $(p \cdot \langle A_Q, \Psi' \otimes (p \cdot \Psi_R)\rangle) \hookrightarrow_F (p \cdot \langle A_P, (\Psi \otimes \Psi_P) \otimes (p \cdot \Psi_R)\rangle)$
by(*rule FrameStatImpClosed*)
with ‹A_R ♯* A_P› ‹$(p \cdot A_R)$ ♯* A_P› ‹A_R ♯* Ψ'› ‹$(p \cdot A_R)$ ♯* Ψ'› ‹A_R ♯* Ψ_P› ‹$(p \cdot A_R)$ ♯* Ψ_P› ‹A_R ♯* A_Q›
‹$(p \cdot A_R)$ ♯* A_Q› ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› S ‹*distinctPerm* p›
have $\langle A_Q, \Psi' \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$ **by**(*simp add*: *eqvts*)
ultimately obtain K' **where** $\Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'$ (⦇ν*xvec⦈N $\prec'$ R') **and** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash (p \cdot M) \leftrightarrow K'$ **and** *zvec* ♯* K' **and** A_R ♯* K'
using *cAlpha*
by *metis*
from ‹$\Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'$ (⦇ν*xvec⦈N $\prec'$ R')› S ‹A_R ♯* R› ‹$(p \cdot A_R)$ ♯* R›
have $(p \cdot (\Psi \otimes \Psi_P)) \rhd R \longmapsto (ROut\ (p \cdot K')$ (⦇ν*xvec⦈N $\prec'$ R')) **using** *outputPermFrameSubject*
by(*auto simp add*: *residualInject*)
with S ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› ‹A_R ♯* Ψ_P› ‹$(p \cdot A_R)$ ♯* Ψ_P› **have** $\Psi \otimes \Psi_P \rhd R \longmapsto (ROut\ (p \cdot K')$ (⦇ν*xvec⦈N $\prec'$ R'))
by(*simp add*: *eqvts*)
moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_R \vdash (p \cdot M) \leftrightarrow K'$› **have** $(p \cdot (\Psi \otimes \Psi_P \otimes \Psi_R)) \vdash (p \cdot p \cdot M) \leftrightarrow (p \cdot K')$
by(*rule chanEqClosed*)
with S ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› ‹A_R ♯* Ψ_P› ‹$(p \cdot A_R)$ ♯* Ψ_P› ‹*distinctPerm* p› **have** $\Psi \otimes \Psi_P \otimes (p \cdot \Psi_R) \vdash M \leftrightarrow (p \cdot K')$

by(*simp add*: *eqvts*)
moreover from ‹*zvec* $\sharp*$ K'› **have** $(p \cdot zvec)$ $\sharp*$ $(p \cdot K')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_R $\sharp*$ *zvec*› ‹$(p \cdot A_R)$ $\sharp*$ *zvec*› S **have** *zvec* $\sharp*$ $(p \cdot K')$ **by** *simp*
moreover from ‹A_R $\sharp*$ K'› **have** $(p \cdot A_R)$ $\sharp*$ $(p \cdot K')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case* **by** *blast*
next
case(*cOutput* Ψ' M' K N R Ψ P A_P Ψ_P A_Q *zvec xvec* N' R' M)
from ‹A_P $\sharp*$ $(M'\langle N\rangle.R)$› ‹A_Q $\sharp*$ $(M'\langle N\rangle.R)$› ‹*zvec* $\sharp*$ $(M'\langle N\rangle.R)$›
have A_P $\sharp*$ M' **and** A_Q $\sharp*$ M' **and** *zvec* $\sharp*$ M' **by** *simp*+

from ‹$\Psi' \vdash M' \leftrightarrow K$› **have** $\Psi' \otimes \mathbf{1} \vdash M' \leftrightarrow K$ **by**(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
hence $\Psi' \otimes \mathbf{1} \vdash M' \leftrightarrow M'$ **by**(*blast intro*: *chanEqSym chanEqTrans*)
with ‹A_Q $\sharp*$ M'› **have** $(\langle A_Q, \Psi' \otimes \mathbf{1}\rangle) \vdash_F M' \leftrightarrow M'$ **by**(*force intro*: *frameImpI*)

with ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$› **have** $(\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle) \vdash_F M' \leftrightarrow M'$
by(*simp add*: *FrameStatImp-def*)
with ‹A_P $\sharp*$ M'› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \vdash M' \leftrightarrow M'$ **by**(*force dest*: *frameImpE*)
hence $\Psi \otimes \Psi_P \vdash M' \leftrightarrow M'$ **by**(*blast intro*: *statEqEnt Identity*) **hence** $\Psi \otimes \Psi_P \rhd M'\langle N\rangle.R \longmapsto M'\langle N\rangle \prec R$
by(*rule Output*)

moreover from ‹$\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow K$› ‹$\Psi' \otimes \mathbf{1} \vdash M' \leftrightarrow K$›
have $\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow M'$ **by**(*metis chanEqSym chanEqTrans*)
with ‹A_Q $\sharp*$ M› ‹A_Q $\sharp*$ M'›
have $(\langle A_Q, \Psi' \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow M'$
by(*force intro*: *frameImpI*)
with ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$›
have $(\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow M'$
by(*simp add*: *FrameStatImp-def*)
with ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ M'› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \vdash M \leftrightarrow M'$
by(*force dest*: *frameImpE*)
hence $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity*)
ultimately show *?case* **using** *cOutput* **by**(*auto simp add*: *residualInject*)
next
case(*cCase* Ψ' R M' φ *Cs* A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec xvec* N R' M)
from ‹*guarded* R› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** $\Psi_R \simeq \mathbf{1}$
by(*metis guardedStatEq*)
with ‹$\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow M'$› **have** $\Psi' \otimes \Psi_R \vdash M \leftrightarrow M'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition*)
moreover have $\langle A_Q, \Psi' \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
proof −
from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_Q, \Psi' \otimes \Psi_R\rangle \simeq_F \langle A_Q, \Psi' \otimes \mathbf{1}\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChain-*

Pres frameNilStatEq AssertionStatEqTrans)
moreover note ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle \simeq_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \mathbf{1} \rhd P \longmapsto M(\!|L|\!) \prec P'$› ‹$\Psi_R \simeq \mathbf{1}$›
have $\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|L|\!) \prec P'$ **by**(*metis statEqTransition Identity Commutativity AssertionStatEqSym Composition*)
moreover from ‹*zvec* $\sharp*$ (*Cases Cs*)› ‹$A_P \sharp*$ (*Cases Cs*)› ‹$A_Q \sharp*$ (*Cases Cs*)› ‹(φ, *R*) *mem Cs*›
have $A_P \sharp* R$ **and** $A_Q \sharp* R$ **and** *zvec* $\sharp* R$ **and** $A_P \sharp* \varphi$ **and** $A_Q \sharp* \varphi$
by(*auto dest*: *memFreshChain*)
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R') \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge zvec \sharp* K' \wedge A_R \sharp* K'$ **using** *cCase*
by(*rule-tac cCase*) (*assumption* | *simp*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R')$
and *MeqK'*: $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** *zvec* $\sharp* K'$ **and** $A_R \sharp* K'$
by *metis*
note *RTrans* ‹(φ, *R*) *mem Cs*›
moreover from ‹$\Psi' \vdash \varphi$› **have** $\Psi' \otimes \mathbf{1} \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
with ‹$A_Q \sharp* \varphi$› **have** $(\langle A_Q, \Psi' \otimes \mathbf{1}\rangle) \vdash_F \varphi$ **by**(*force intro*: *frameImpI*)
with ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$› **have** $(\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle) \vdash_F \varphi$
by(*simp add*: *FrameStatImp-def*)
with ‹$A_P \sharp* \varphi$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \vdash \varphi$ **by**(*force dest*: *frameImpE*)
hence $\Psi \otimes \Psi_P \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity*)
ultimately have $\Psi \otimes \Psi_P \rhd Cases\ Cs \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R')$ **using** ‹*guarded R*› **by**(*rule Case*)
moreover from *MeqK'* ‹$\Psi_R \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition AssertionStatEqTrans*)
ultimately show *?case* **using** ‹*zvec* $\sharp* K'$›
by *fastforce*
next
case(*cPar1* Ψ' Ψ_{R2} R_1 M' *xvec* N' $R_1{}'$ A_{R2} R_2 A_{R1} Ψ_{R1} Ψ P A_P Ψ_P A_Q *zvec yvec N R' M*)
have *FrR2*: *extractFrame* $R_2 = \langle A_{R2}, \Psi_{R2}\rangle$ **by** *fact*
from ‹$\Psi' \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow M'$› **have** $(\Psi' \otimes \Psi_{R2}) \otimes \Psi_{R1} \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Composition Commutativity*)
moreover have $\langle A_Q, (\Psi' \otimes \Psi_{R2}) \otimes \Psi_{R1}\rangle \hookrightarrow_F \langle A_P, ((\Psi \otimes \Psi_{R2}) \otimes \Psi_P) \otimes \Psi_{R1}\rangle$
proof −
have $\langle A_Q, (\Psi' \otimes \Psi_{R2}) \otimes \Psi_{R1}\rangle \simeq_F \langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$

by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
moreover note ‹$\langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$›
moreover have $\langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \simeq_F \langle A_P, ((\Psi \otimes \Psi_{R2}) \otimes \Psi_P) \otimes \Psi_{R1}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \Psi_{R1} \otimes \Psi_{R2} \rhd P \longmapsto M(\!|L|\!) \prec P'$› **have** $(\Psi \otimes \Psi_{R2}) \otimes \Psi_{R1} \rhd P \longmapsto M(\!|L|\!) \prec P'$
by(*metis statEqTransition Associativity Composition Commutativity*)
moreover from ‹$A_{R1} \sharp* A_P$› ‹$A_{R2} \sharp* A_P$› ‹$A_P \sharp* (R_1 \parallel R_2)$› ‹*extractFrame* $R_1 = \langle A_{R1}, \Psi_{R1}\rangle$› *FrR2* **have** $A_P \sharp* \Psi_{R1}$ **and** $A_P \sharp* \Psi_{R2}$
by(*force dest*: *extractFrameFreshChain*)+

moreover from ‹$(\!|\nu*xvec|\!)N' \prec' (R_1{}' \parallel R_2) = (\!|\nu*yvec|\!)N \prec' R'$› ‹*xvec* $\sharp*$ *yvec*›
obtain p T **where** $(\!|\nu*xvec|\!)N' \prec' R_1{}' = (\!|\nu*yvec|\!)N \prec' T$ **and** $R' = T \parallel (p \cdot R_2)$ **and** *set* $p \subseteq$ *set yvec* $\times$ *set xvec*
apply(*drule-tac sym*)
by(*rule-tac boundOutputPar1Dest'*) (*assumption* | *simp* | *blast dest*: *sym*)+
ultimately have $\exists K'. (\Psi \otimes \Psi_{R2}) \otimes \Psi_P \rhd R_1 \longmapsto ROut\ K' ((\!|\nu*yvec|\!)N \prec' T) \wedge (\Psi \otimes \Psi_{R2}) \otimes \Psi_P \otimes \Psi_{R1} \vdash M \leftrightarrow K' \wedge (A_{R2}@zvec) \sharp* K' \wedge A_{R1} \sharp* K'$ **using** *cPar1*
apply(*rule-tac cPar1*(*6*)) **by**(*assumption* | *simp* | *fastforce*)+
then obtain K' **where** *RTrans*: $(\Psi \otimes \Psi_{R2}) \otimes \Psi_P \rhd R_1 \longmapsto K'(\!|\nu*xvec|\!)\langle N'\rangle \prec R_1{}'$
and *MeqK'*: $(\Psi \otimes \Psi_{R2}) \otimes \Psi_P \otimes \Psi_{R1} \vdash M \leftrightarrow K'$ **and** $A_{R2} \sharp* K'$ **and** $A_{R1} \sharp* K'$ **and** *zvec* $\sharp*$ K'
using ‹$(\!|\nu*xvec|\!)N' \prec' R_1{}' = (\!|\nu*yvec|\!)N \prec' T$› **by**(*auto simp add*: *residualInject*)

from *RTrans* **have** $(\Psi \otimes \Psi_P) \otimes \Psi_{R2} \rhd R_1 \longmapsto K'(\!|\nu*xvec|\!)\langle N'\rangle \prec R_1{}'$
by(*metis statEqTransition Associativity Composition Commutativity*)
hence $\Psi \otimes \Psi_P \rhd (R_1 \parallel R_2) \longmapsto K'(\!|\nu*xvec|\!)\langle N'\rangle \prec (R_1{}' \parallel R_2)$ **using** *FrR2* ‹*xvec* $\sharp*$ R_2› ‹$A_{R2} \sharp* \Psi$› ‹$A_{R2} \sharp* \Psi_P$› ‹$A_{R2} \sharp* K'$› ‹$A_{R2} \sharp* R_1$› ‹$A_{R2} \sharp*$ *xvec*› ‹$A_{R2} \sharp* N'$›
by(*force intro*: *Par1*)
moreover from *MeqK'* **have** $\Psi \otimes \Psi_P \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow K'$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹*zvec* $\sharp*$ K'› ‹$A_{R1} \sharp* K'$› ‹$A_{R2} \sharp* K'$› ‹$(\!|\nu*xvec|\!)N' \prec' (R_1{}' \parallel R_2) = (\!|\nu*yvec|\!)N \prec' R'$›
by(*auto simp add*: *residualInject*)
next
case(*cPar2* Ψ' Ψ_{R1} R_2 M' *xvec* N' $R_2{}'$ A_{R1} R_1 A_{R2} Ψ_{R2} Ψ P A_P Ψ_P A_Q *zvec yvec* N R' M)
have *FrR1*: *extractFrame* $R_1 = \langle A_{R1}, \Psi_{R1}\rangle$ **by** *fact*

from ‹$\Psi' \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow M'$› **have** $(\Psi' \otimes \Psi_{R1}) \otimes \Psi_{R2} \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Composition Commutativity*)
moreover have $\langle A_Q, (\Psi' \otimes \Psi_{R1}) \otimes \Psi_{R2}\rangle \hookrightarrow_F \langle A_P, ((\Psi \otimes \Psi_{R1}) \otimes \Psi_P) \otimes \Psi_{R2}\rangle$
proof −
have $\langle A_Q, (\Psi' \otimes \Psi_{R1}) \otimes \Psi_{R2}\rangle \simeq_F \langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
moreover note ‹$\langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$›
moreover have $\langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \simeq_F \langle A_P, ((\Psi \otimes \Psi_{R1}) \otimes \Psi_P) \otimes \Psi_{R2}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \Psi_{R1} \otimes \Psi_{R2} \rhd P \longmapsto M(\!|L|\!) \prec P'$› **have** $(\Psi \otimes \Psi_{R1}) \otimes \Psi_{R2} \rhd P \longmapsto M(\!|L|\!) \prec P'$
by(*metis statEqTransition Associativity Composition Commutativity*)
moreover from ‹$A_{R1} \sharp* A_P$› ‹$A_{R2} \sharp* A_P$› ‹$A_P \sharp* (R_1 \parallel R_2)$› *FrR1* ‹*extractFrame* $R_2 = \langle A_{R2}, \Psi_{R2}\rangle$› **have** $A_P \sharp* \Psi_{R1}$ **and** $A_P \sharp* \Psi_{R2}$
by(*force dest*: *extractFrameFreshChain*)+
moreover from ‹$(\!|\nu*xvec|\!)N' \prec' (R_1 \parallel R_2{}') = (\!|\nu*yvec|\!)N \prec' R'$› ‹$xvec \sharp* yvec$›
obtain p T **where** $(\!|\nu*xvec|\!)N' \prec' R_2{}' = (\!|\nu*yvec|\!)N \prec' T$ **and** $R' = (p \cdot R_1) \parallel T$ **and** $set\ p \subseteq set\ yvec \times set\ xvec$
apply(*drule-tac sym*)
by(*rule-tac boundOutputPar2Dest'*) (*assumption* | *simp* | *blast dest*: *sym*)+
ultimately have $\exists K'.\ (\Psi \otimes \Psi_{R1}) \otimes \Psi_P \rhd R_2 \longmapsto ROut\ K'\ ((\!|\nu*yvec|\!)N \prec' T) \wedge (\Psi \otimes \Psi_{R1}) \otimes \Psi_P \otimes \Psi_{R2} \vdash M \leftrightarrow K' \wedge (A_{R1}@zvec) \sharp* K' \wedge A_{R2} \sharp* K'$ **using** *cPar2*
by(*rule-tac cPar2*(*6*)) (*assumption* | *simp* | *fastforce*)+
then obtain K' **where** *RTrans*: $(\Psi \otimes \Psi_{R1}) \otimes \Psi_P \rhd R_2 \longmapsto K'(\!|\nu*xvec|\!)\langle N'\rangle \prec R_2{}'$
and *MeqK'*: $(\Psi \otimes \Psi_{R1}) \otimes \Psi_P \otimes \Psi_{R2} \vdash M \leftrightarrow K'$ **and** $A_{R1} \sharp* K'$ **and** $zvec \sharp* K'$ **and** $A_{R2} \sharp* K'$
using ‹$(\!|\nu*xvec|\!)N' \prec' R_2{}' = (\!|\nu*yvec|\!)N \prec' T$› **by**(*auto simp add*: *residualInject*)

from *RTrans* **have** $(\Psi \otimes \Psi_P) \otimes \Psi_{R1} \rhd R_2 \longmapsto K'(\!|\nu*xvec|\!)\langle N'\rangle \prec R_2{}'$
by(*metis statEqTransition Associativity Composition Commutativity*)
hence $\Psi \otimes \Psi_P \rhd (R_1 \parallel R_2) \longmapsto K'(\!|\nu*xvec|\!)\langle N'\rangle \prec (R_1 \parallel R_2{}')$ **using** *FrR1* ‹$xvec \sharp* R_1$› ‹$A_{R1} \sharp* \Psi$› ‹$A_{R1} \sharp* \Psi_P$› ‹$A_{R1} \sharp* K'$›‹$A_{R1} \sharp* xvec$› ‹$A_{R1} \sharp* N'$› ‹$A_{R1} \sharp* R_2$›
by(*force intro*: *Par2*)
moreover from *MeqK'* **have** $\Psi \otimes \Psi_P \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow K'$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹$zvec \sharp* K'$› ‹$A_{R1} \sharp* K'$› ‹$A_{R2} \sharp* K'$› ‹$(\!|\nu*xvec|\!)N' \prec' (R_1 \parallel R_2{}') = (\!|\nu*yvec|\!)N \prec' R'$›

by(*auto simp add*: *residualInject*)
next
case(*cOpen* Ψ' R M' *xvec yvec* N' R' x A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec zvec2* N R'' M)
from ‹$(\!|\nu*(xvec\ @\ x\ \#\ yvec)|\!)N' \prec' R' = (\!|\nu*zvec2|\!)N \prec' R''$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› ‹$x \sharp zvec2$› ‹$x \sharp R''$› ‹$x \sharp N$› ‹*distinct zvec2*›
obtain *xvec'* x' *yvec'* **where** A: $(\!|\nu*(xvec@yvec)|\!)N' \prec' R' = (\!|\nu*(xvec'@yvec')|\!)([(x, x')] \cdot N) \prec' ([(x, x')] \cdot R'')$
and B: $zvec2 = (xvec'@x'\#yvec')$
by(*rule-tac boundOutputOpenDest*) *auto*
then have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'\ ((\!|\nu*(xvec'@yvec')|\!)([(x, x')] \cdot N) \prec' ([(x, x')] \cdot R'')) \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge (zvec) \sharp* K' \wedge A_R \sharp* K'$ **using** *cOpen*
by(*rule-tac cOpen*(*4*)) (*assumption* | *simp*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|\nu*(xvec@yvec)|\!)\langle N'\rangle \prec R'$
and *MeqK'*: $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $zvec \sharp* K'$ **and** $A_R \sharp* K'$
using A **by**(*auto simp add*: *residualInject*)
from ‹$A_R \sharp* A_P$› ‹$A_P \sharp* ((\!|\nu x|\!)R)$› ‹$x \sharp A_P$› ‹$extractFrame\ R = \langle A_R, \Psi_R\rangle$› **have** $A_P \sharp* \Psi_R$
by(*force dest*: *extractFrameFreshChain*)+

from ‹$\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|L|\!) \prec P'$› ‹$extractFrame\ P = \langle A_P, \Psi_P\rangle$› ‹*distinct* A_P› ‹$zvec \sharp* P$› ‹$A_P \sharp* \Psi_R$› ‹$x \sharp A_P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* P$› ‹$A_P \sharp* zvec$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* zvec$› ‹$A_R \sharp* P$› ‹$A_R \sharp* A_P$› ‹$x \sharp A_P$› ‹$x \sharp P$›

obtain K'' **where** *MeqK''*: $(\Psi \otimes \Psi_R) \otimes \Psi_P \vdash M \leftrightarrow K''$ **and** $A_R \sharp* K''$ **and** $zvec \sharp* K''$ **and** $x \sharp K''$
by(*rule-tac* B=($x\#A_R@zvec$) **in** *obtainPrefix*) (*assumption* | *simp* | *force*)+

from *MeqK''* *MeqK'* **have** *KeqK''*: $(\Psi \otimes \Psi_P) \otimes \Psi_R \vdash K' \leftrightarrow K''$
by(*metis statEqEnt Associativity Composition Commutativity chanEqSym chanEqTrans*)
with *RTrans* ‹$extractFrame\ R = \langle A_R, \Psi_R\rangle$› ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P$› ‹$A_R \sharp* K'$› ‹$A_R \sharp* K''$› ‹$A_R \sharp* R$›
have $\Psi \otimes \Psi_P \rhd R \longmapsto K''(\!|\nu*(xvec@yvec)|\!)\langle N'\rangle \prec R'$
by(*rule-tac outputRenameSubject*) (*assumption* | *force*)+
hence $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)R \longmapsto K''(\!|\nu*(xvec@x\#yvec)|\!)\langle N'\rangle \prec R'$
using ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp K''$› ‹$x \sharp xvec$› ‹ $x \sharp yvec$› ‹$x \in supp\ N'$› ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* \Psi_P$› ‹$xvec \sharp* R$› ‹$x \sharp K''$›
by(*rule-tac Open*) (*assumption* | *force*)+
moreover from *MeqK''* **have** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K''$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹$zvec \sharp* K''$› ‹$x \sharp K''$› ‹$A_R \sharp* K''$› B ‹$(\!|\nu*(xvec\ @\ x\ \#\ yvec)|\!)N' \prec' R' = (\!|\nu*zvec2|\!)N \prec' R''$›
by(*auto simp add*: *residualInject*)
next
case(*cScope* Ψ' R M' *xvec* N' R' x A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec yvec* N R''

M)
from ‹$(\!|\nu{*}xvec|\!)N' \prec' (\!|\nu x|\!)R' = (\!|\nu{*}yvec|\!)N \prec' R''$› ‹$x \sharp xvec$› ‹$x \sharp yvec$›
obtain R''' **where** $R'' = (\!|\nu x|\!)R'''$ **and** $(\!|\nu{*}xvec|\!)N' \prec' R' = (\!|\nu{*}yvec|\!)N \prec' R'''$
apply(*drule-tac sym*)
by(*rule boundOutputScopeDest*) (*assumption* | *auto*)+
then have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'\ ((\!|\nu{*}yvec|\!)N \prec' R''') \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge zvec \sharp{*} K' \wedge A_R \sharp{*} K'$ **using** *cScope*
by(*rule-tac cScope*(*4*)) (*assumption* | *simp*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|\nu{*}xvec|\!)\langle N'\rangle \prec R'$
and *MeqK'*: $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $zvec \sharp{*} K'$ **and** $A_R \sharp{*} K'$
using ‹$(\!|\nu{*}xvec|\!)N' \prec' R' = (\!|\nu{*}yvec|\!)N \prec' R'''$›
by(*auto simp add*: *residualInject*)
from ‹$A_R \sharp{*} A_P$› ‹$A_P \sharp{*} ((\!|\nu x|\!)R)$› ‹$x \sharp A_P$› ‹$extractFrame\ R = \langle A_R, \Psi_R\rangle$› **have** $A_P \sharp{*} \Psi_R$
by(*force dest*: *extractFrameFreshChain*)+
from ‹$\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|L|\!) \prec P'$› ‹$extractFrame\ P = \langle A_P, \Psi_P\rangle$› ‹$distinct\ A_P$› ‹$x \sharp P$› ‹$zvec \sharp{*} P$› ‹$A_P \sharp{*} \Psi_R$› ‹$x \sharp A_P$› ‹$A_P \sharp{*} M$› ‹$A_P \sharp{*} P$› ‹$A_P \sharp{*} zvec$› ‹$A_P \sharp{*} \Psi$› ‹$A_P \sharp{*} zvec$› ‹$A_R \sharp{*} P$› ‹$A_R \sharp{*} A_P$›
obtain K'' **where** *MeqK''*: $(\Psi \otimes \Psi_R) \otimes \Psi_P \vdash M \leftrightarrow K''$ **and** $x \sharp K''$ **and** $A_R \sharp{*} K''$ **and** $zvec \sharp{*} K''$
by(*rule-tac B*=($x\#A_R$@*zvec*) **in** *obtainPrefix*) (*assumption* | *force*)+

from *MeqK''* *MeqK'* **have** *KeqK''*: $(\Psi \otimes \Psi_P) \otimes \Psi_R \vdash K' \leftrightarrow K''$
by(*metis statEqEnt Associativity Composition Commutativity chanEqSym chanEqTrans*)
with *RTrans* ‹$extractFrame\ R = \langle A_R, \Psi_R\rangle$› ‹$distinct\ A_R$› ‹$A_R \sharp{*} \Psi$› ‹$A_R \sharp{*} \Psi_P$› ‹$A_R \sharp{*} K'$› ‹$A_R \sharp{*} K''$› ‹$A_R \sharp{*} R$›
have $\Psi \otimes \Psi_P \rhd R \longmapsto K''(\!|\nu{*}xvec|\!)\langle N'\rangle \prec R'$
by(*rule-tac outputRenameSubject*) (*assumption* | *force*)+
hence $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)R \longmapsto K''(\!|\nu{*}xvec|\!)\langle N'\rangle \prec (\!|\nu x|\!)R'$ **using** ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp K''$› ‹$x \sharp xvec$› ‹$x \sharp N'$› ‹$xvec \sharp{*} \Psi$› ‹$xvec \sharp{*} \Psi_P$› ‹$xvec \sharp{*} R$› ‹$x \sharp K''$›
by(*rule-tac Scope*) (*assumption* | *force*)+
moreover from *MeqK''* **have** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K''$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹$zvec \sharp{*} K''$› ‹$x \sharp K''$› ‹$A_R \sharp{*} K''$› ‹$(\!|\nu{*}xvec|\!)N' \prec' (\!|\nu x|\!)R' = (\!|\nu{*}yvec|\!)N \prec' R''$›
by(*auto simp add*: *residualInject*)
next
case(*cBang* Ψ' R M' A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec xvec* N R' M)
from ‹*guarded R*› ‹$extractFrame\ R = \langle A_R, \Psi_R\rangle$› **have** $\Psi_R \simeq \mathbf{1}$
by(*metis guardedStatEq*)
with ‹$\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow M'$› **have** $\Psi' \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow M'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition*)
moreover have $\langle A_Q, \Psi' \otimes \Psi_R \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R \otimes \mathbf{1}\rangle$
proof −
from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_Q,\ \Psi' \otimes \Psi_R \otimes \mathbf{1}\rangle \simeq_F \langle A_Q, \Psi' \otimes \mathbf{1}\rangle$

by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
moreover note ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle \simeq_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R \otimes \mathbf{1}\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \mathbf{1} \rhd P \longmapsto M(\!|L|\!) \prec P'$› ‹$\Psi_R \simeq \mathbf{1}$›
have $\Psi \otimes \Psi_R \otimes \mathbf{1} \rhd P \longmapsto M(\!|L|\!) \prec P'$ **by**(*metis statEqTransition Identity Commutativity AssertionStatEqSym Composition*)
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \parallel !R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R') \wedge \Psi \otimes \Psi_P \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K' \wedge zvec \sharp* K' \wedge A_R \sharp* K'$ **using** *cBang*
by(*rule-tac cBang(5)*) (*assumption* |*simp*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \parallel !R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R')$
and *MeqK'*: $\Psi \otimes \Psi_P \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K'$ **and** $zvec \sharp* K'$ **and** $A_R \sharp* K'$
by *metis*
from *RTrans* ‹*guarded R*› **have** $\Psi \otimes \Psi_P \rhd !R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R')$ **by**(*rule Bang*)
moreover from *MeqK'* ‹$\Psi_R \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition AssertionStatEqTrans*)
ultimately show *?case* **using** ‹$zvec \sharp* K'$›
by *force*
qed
}
note *Goal* = *this*
have $\Psi \otimes \Psi_Q \simeq \Psi \otimes \Psi_Q$ **by** *simp*
moreover note *RTrans*
moreover from *MeqK* **have** $(\Psi \otimes \Psi_Q) \otimes \Psi_R \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Commutativity*)
moreover note *PeqQ* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_Q$›
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto ROut\ K'\ ((\!|\nu{*}xvec|\!)N \prec' R') \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge ([]::name\ list) \sharp* K' \wedge A_R \sharp* K'$
by(*rule-tac Goal*) (*assumption* | *force simp add*: *residualInject*)+
with *Assumptions* **show** *?thesis*
by(*force simp add*: *residualInject*)
qed

lemma *comm2Aux*:
fixes Ψ :: *'b*
and Ψ_Q :: *'b*
and R :: (*'a*, *'b*, *'c*) *psi*
and K :: *'a*
and N :: *'a*
and R' :: (*'a*, *'b*, *'c*) *psi*

and A_R :: *name list*
and Ψ_R :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and P' :: $('a,\ 'b,\ 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$
and A_Q :: *name list*

assumes *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$
and *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$
and *PTrans*: $\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$
and *QimpP*: $\langle A_Q,\ (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P,\ (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
and *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$
and *FrQ*: *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$
and *distinct* A_P
and *distinct* A_R
and $A_R \sharp{*}\ A_P$
and $A_R \sharp{*}\ A_Q$
and $A_R \sharp{*}\ \Psi$
and $A_R \sharp{*}\ P$
and $A_R \sharp{*}\ Q$
and $A_R \sharp{*}\ R$
and $A_R \sharp{*}\ K$
and $A_P \sharp{*}\ \Psi$
and $A_P \sharp{*}\ R$
and $A_P \sharp{*}\ P$
and $A_P \sharp{*}\ M$
and $A_Q \sharp{*}\ R$
and $A_Q \sharp{*}\ M$
and $A_R \sharp{*}\ xvec$
and $xvec \sharp{*}\ M$

obtains K' **where** $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$ **and** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $A_R \sharp{*}\ K'$
proof −
from ‹$A_R \sharp{*}\ P$› ‹$A_R \sharp{*}\ Q$› ‹$A_R \sharp{*}\ A_P$› ‹$A_R \sharp{*}\ A_Q$› *FrP FrQ* **have** $A_R \sharp{*}\ \Psi_P$ **and** $A_R \sharp{*}\ \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+
assume *Assumptions*: $\bigwedge K'.\ [\![\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R';\ \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K';\ A_R \sharp{*}\ K']\!] \Longrightarrow thesis$
{
fix $\Psi'::'b$
fix *zvec*::*name list*
assume $A_R \sharp{*}\ \Psi'$
assume $A_R \sharp{*}\ zvec$
assume $A_P \sharp{*}\ zvec$

assume *zvec* $\sharp*$ R
assume *zvec* $\sharp*$ P

assume A: $\Psi \otimes \Psi_Q \simeq \Psi'$
with *RTrans* **have** $\Psi' \rhd R \longmapsto K(\!|N|\!) \prec R'$
by(*rule statEqTransition*)
moreover note *FrR* ‹*distinct* A_R›
moreover from ‹$\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_Q) \otimes \Psi_R \vdash M \leftrightarrow K$
by(*blast intro*: *statEqEnt Associativity AssertionStatEqSym*)
with A **have** $\Psi' \otimes \Psi_R \vdash M \leftrightarrow K$ **by**(*rule statEqEnt*[*OF Composition*])
moreover have $\langle A_Q, \Psi' \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle$ **using** A
by(*blast dest*: *frameIntComposition FrameStatEqTrans FrameStatEqSym*)
with *QimpP* **have** $\langle A_Q, \Psi' \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
by(*force intro*: *FrameStatEqImpCompose*)
moreover from *PTrans* **have** *distinct xvec* **by**(*auto dest*: *boundOutputDistinct*)
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R' \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge$ *zvec* $\sharp*$ $K' \wedge A_R \sharp* K'$
using *PTrans FrP* ‹$A_R \sharp* K$› ‹$A_R \sharp* \Psi'$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* R$› ‹$A_R \sharp* \Psi_P$› ‹$A_P \sharp* R$› ‹$A_Q \sharp* R$› ‹$A_P \sharp* \Psi$›
‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp*$ *zvec*› ‹$A_Q \sharp* M$› ‹$A_R \sharp*$ *zvec*› ‹*zvec* $\sharp*$ R› ‹*zvec* $\sharp*$ P› ‹*distinct* A_P›
‹$A_R \sharp* A_P$› ‹$A_R \sharp* A_Q$› ‹$A_R \sharp*$ *xvec*› ‹*xvec* $\sharp*$ M›
proof(*nominal-induct avoiding*: Ψ P A_P Ψ_P A_Q *zvec xvec arbitrary*: M *rule*: *inputFrameInduct*)
case(*cAlpha* Ψ' R K N R' A_R Ψ_R p Ψ P A_P Ψ_P A_Q *zvec xvec* M)
have S: *set* $p \subseteq$ *set* $A_R \times$ *set* $(p \cdot A_R)$ **by** *fact*
from ‹$\Psi' \otimes (p \cdot \Psi_R) \vdash M \leftrightarrow K$› **have** $(p \cdot (\Psi' \otimes (p \cdot \Psi_R))) \vdash (p \cdot M) \leftrightarrow (p \cdot K)$
by(*rule chanEqClosed*)
with ‹$A_R \sharp* \Psi'$› ‹$(p \cdot A_R) \sharp* \Psi'$› ‹$A_R \sharp* K$› ‹$(p \cdot A_R) \sharp* K$› S ‹*distinctPerm* p›
have $\Psi' \otimes \Psi_R \vdash (p \cdot M) \leftrightarrow K$ **by**(*simp add*: *eqvts*)
moreover from ‹$\Psi \otimes (p \cdot \Psi_R) \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› S ‹$A_R \sharp* P$› ‹$(p \cdot A_R) \sharp* P$› ‹$A_R \sharp*$ *xvec*› ‹$(p \cdot A_R) \sharp*$ *xvec*› ‹*xvec* $\sharp*$ M›
have $(p \cdot (\Psi \otimes (p \cdot \Psi_R))) \rhd P \longmapsto (p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
using *outputPermFrameSubject* **by**(*auto simp add*: *residualInject*)
with ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* \Psi$› S ‹*distinctPerm* p› **have** $\Psi \otimes \Psi_R \rhd P \longmapsto (p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*simp add*: *eqvts*)
moreover from ‹$A_P \sharp* M$› **have** $(p \cdot A_P) \sharp* (p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* A_P$› ‹$(p \cdot A_R) \sharp* A_P$› S **have** $A_P \sharp* (p \cdot M)$ **by** *simp*
moreover from ‹$A_Q \sharp* M$› **have** $(p \cdot A_Q) \sharp* (p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* A_Q$› ‹$(p \cdot A_R) \sharp* A_Q$› S **have** $A_Q \sharp* (p \cdot M)$ **by** *simp*

moreover from ‹$\langle A_Q, \Psi' \otimes (p \cdot \Psi_R)\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes (p \cdot \Psi_R)\rangle$›
have $(p \cdot \langle A_Q, \Psi' \otimes (p \cdot \Psi_R)\rangle) \hookrightarrow_F (p \cdot \langle A_P, (\Psi \otimes \Psi_P) \otimes (p \cdot \Psi_R)\rangle)$
by(*rule FrameStatImpClosed*)

with ‹A_R ♯* A_P› ‹$(p \cdot A_R)$ ♯* A_P› ‹A_R ♯* Ψ'› ‹$(p \cdot A_R)$ ♯* Ψ'› ‹A_R ♯* Ψ_P› ‹$(p \cdot A_R)$ ♯* Ψ_P› ‹A_R ♯* A_Q›
‹$(p \cdot A_R)$ ♯* A_Q› ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› S ‹*distinctPerm p*›
have $\langle A_Q, \Psi' \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$ **by**(*simp add*: *eqvts*)
moreover from ‹*xvec* ♯* M› **have** $(p \cdot xvec)$ ♯* $(p \cdot M)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with S ‹A_R ♯* *xvec*› ‹$(p \cdot A_R)$ ♯* *xvec*› **have** *xvec* ♯* $(p \cdot M)$ **by** *simp*
ultimately obtain K' **where** $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$ **and** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash (p \cdot M) \leftrightarrow K'$ **and** *zvec* ♯* K' **and** A_R ♯* K'
using *cAlpha*
by(*fastforce simp del*: *freshChainSimps*)
from ‹$\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$› S ‹A_R ♯* R› ‹$(p \cdot A_R)$ ♯* R› **have** $(p \cdot (\Psi \otimes \Psi_P)) \rhd R \longmapsto (p \cdot K')(\!|N|\!) \prec R'$
by(*rule-tac inputPermFrameSubject*) *auto*
with S ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› ‹A_R ♯* Ψ_P› ‹$(p \cdot A_R)$ ♯* Ψ_P› **have** $\Psi \otimes \Psi_P \rhd R \longmapsto (p \cdot K')(\!|N|\!) \prec R'$
by(*simp add*: *eqvts*)
moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_R \vdash (p \cdot M) \leftrightarrow K'$› **have** $(p \cdot (\Psi \otimes \Psi_P \otimes \Psi_R)) \vdash (p \cdot p \cdot M) \leftrightarrow (p \cdot K')$
by(*rule chanEqClosed*)
with S ‹A_R ♯* Ψ› ‹$(p \cdot A_R)$ ♯* Ψ› ‹A_R ♯* Ψ_P› ‹$(p \cdot A_R)$ ♯* Ψ_P› ‹*distinctPerm p*› **have** $\Psi \otimes \Psi_P \otimes (p \cdot \Psi_R) \vdash M \leftrightarrow (p \cdot K')$
by(*simp add*: *eqvts*)
moreover from ‹*zvec* ♯* K'› **have** $(p \cdot zvec)$ ♯* $(p \cdot K')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹A_R ♯* *zvec*› ‹$(p \cdot A_R)$ ♯* *zvec*› S **have** *zvec* ♯* $(p \cdot K')$ **by** *simp*
moreover from ‹A_R ♯* K'› **have** $(p \cdot A_R)$ ♯* $(p \cdot K')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case* **by** *blast*
next
case(*cInput* Ψ' M' K *xvec* N *Tvec* R Ψ P A_P Ψ_P A_Q *zvec yvec* M)
from ‹A_P ♯* $(M'(\!|\lambda{*}xvec\ N|\!).R)$› ‹$A_Q$ ♯* $(M'(\!|\lambda{*}xvec\ N|\!).R)$› ‹*zvec* ♯* $(M'(\!|\lambda{*}xvec\ N|\!).R)$›
have A_P ♯* M' **and** A_Q ♯* M' **and** *zvec* ♯* M' **by** *simp*+

from ‹$\Psi' \vdash M' \leftrightarrow K$›
have $\Psi' \otimes \mathbf{1} \vdash M' \leftrightarrow K$
by(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
hence $\Psi' \otimes \mathbf{1} \vdash M' \leftrightarrow M'$
by(*blast intro*: *chanEqSym chanEqTrans*)
with ‹A_Q ♯* M'›
have $(\langle A_Q, \Psi' \otimes \mathbf{1}\rangle) \vdash_F M' \leftrightarrow M'$
by(*force intro*: *frameImpI*)

with ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$›
have $(\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle) \vdash_F M' \leftrightarrow M'$
by(*simp add*: *FrameStatImp-def*)
with ‹A_P ♯* M'› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \vdash M' \leftrightarrow M'$
by(*force dest*: *frameImpE*)

hence $\Psi \otimes \Psi_P \vdash M' \leftrightarrow M'$ **by**(*blast intro*: *statEqEnt Identity*)
hence $\Psi \otimes \Psi_P \rhd M'(\!|\lambda{*}xvec\ N|\!).R \longmapsto M'(\!|(N[xvec::=Tvec])|\!) \prec R[xvec::=Tvec]$
using ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*›
by(*rule Input*)

moreover from ‹$\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow K$› ‹$\Psi' \otimes \mathbf{1} \vdash M' \leftrightarrow K$›
have $\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow M'$ **by**(*metis chanEqSym chanEqTrans*)
with ‹$A_Q \sharp{*} M$› ‹$A_Q \sharp{*} M'$›
have $(\langle A_Q, \Psi' \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow M'$
by(*force intro*: *frameImpI*)
with ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$ ›
have $(\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle) \vdash_F M \leftrightarrow M'$
by(*simp add*: *FrameStatImp-def*)
with ‹$A_P \sharp{*} M$› ‹$A_P \sharp{*} M'$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \vdash M \leftrightarrow M'$
by(*force dest*: *frameImpE*)
hence $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity*)
ultimately show *?case* **using** ‹*zvec* $\sharp{*}$ M'›
by *force*
next
case(*cCase* Ψ' R M' N R' φ *Cs* A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec xvec* M)
from ‹*guarded R*› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** $\Psi_R \simeq \mathbf{1}$
by(*metis guardedStatEq*)
with ‹$\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow M'$› **have** $\Psi' \otimes \Psi_R \vdash M \leftrightarrow M'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition*)
moreover have $\langle A_Q, \Psi' \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
proof −
from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_Q, \Psi' \otimes \Psi_R\rangle \simeq_F \langle A_Q, \Psi' \otimes \mathbf{1}\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
moreover note ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle \simeq_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \mathbf{1} \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹$\Psi_R \simeq \mathbf{1}$›
have $\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by**(*metis statEqTransition Identity Commutativity AssertionStatEqSym Composition*)
moreover from ‹*zvec* $\sharp{*}$ (*Cases Cs*)› ‹$A_P \sharp{*}$ (*Cases Cs*)› ‹$A_Q \sharp{*}$ (*Cases Cs*)› ‹(φ, R) *mem Cs*›
have $A_P \sharp{*} R$ **and** $A_Q \sharp{*} R$ **and** *zvec* $\sharp{*}$ R **and** $A_P \sharp{*} \varphi$ **and** $A_Q \sharp{*} \varphi$
by(*auto dest*: *memFreshChain*)
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R' \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge$ *zvec* $\sharp{*}$ $K' \wedge A_R \sharp{*} K'$ **using** *cCase*
by(*rule-tac cCase*) (*assumption* |*simp*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$

and *MeqK′*: $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** *zvec* $\sharp*$ K' **and** $A_R \sharp* K'$
by *metis*
note *RTrans* ‹(φ, R) *mem Cs*›
moreover from ‹$\Psi' \vdash \varphi$› **have** $\Psi' \otimes \mathbf{1} \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity AssertionStatEqSym*)
with ‹$A_Q \sharp* \varphi$› **have** $(\langle A_Q, \Psi' \otimes \mathbf{1}\rangle) \vdash_F \varphi$ **by**(*force intro*: *frameImpI*)
with ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$› **have** $(\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle) \vdash_F \varphi$
by(*simp add*: *FrameStatImp-def*)
with ‹$A_P \sharp* \varphi$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \vdash \varphi$ **by**(*force dest*: *frameImpE*)
hence $\Psi \otimes \Psi_P \vdash \varphi$ **by**(*blast intro*: *statEqEnt Identity*)
ultimately have $\Psi \otimes \Psi_P \rhd$ *Cases Cs* $\longmapsto K'(\!|N|\!) \prec R'$ **using** ‹*guarded R*› **by**(*rule Case*)
moreover from *MeqK′* ‹$\Psi_R \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition AssertionStatEqTrans*)
ultimately show *?case* **using** ‹*zvec* $\sharp*$ K'›
by *force*
next
case(*cPar1* Ψ' Ψ_{R2} R_1 M' N $R_1{}'$ A_{R2} R_2 A_{R1} Ψ_{R1} Ψ P A_P Ψ_P A_Q *zvec xvec* M)
have *FrR2*: *extractFrame* $R_2 = \langle A_{R2}, \Psi_{R2}\rangle$ **by** *fact*
from ‹$\Psi' \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow M'$› **have** $(\Psi' \otimes \Psi_{R2}) \otimes \Psi_{R1} \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Composition Commutativity*)
moreover have $\langle A_Q, (\Psi' \otimes \Psi_{R2}) \otimes \Psi_{R1}\rangle \hookrightarrow_F \langle A_P, ((\Psi \otimes \Psi_{R2}) \otimes \Psi_P) \otimes \Psi_{R1}\rangle$
proof −
have $\langle A_Q, (\Psi' \otimes \Psi_{R2}) \otimes \Psi_{R1}\rangle \simeq_F \langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
moreover note ‹$\langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$›
moreover have $\langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \simeq_F \langle A_P, ((\Psi \otimes \Psi_{R2}) \otimes \Psi_P) \otimes \Psi_{R1}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \Psi_{R1} \otimes \Psi_{R2} \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› **have** $(\Psi \otimes \Psi_{R2}) \otimes \Psi_{R1} \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*metis statEqTransition Associativity Composition Commutativity*)
moreover from ‹$A_{R1} \sharp* A_P$› ‹$A_{R2} \sharp* A_P$› ‹$A_P \sharp* (R_1 \parallel R_2)$› ‹*extractFrame* $R_1 = \langle A_{R1}, \Psi_{R1}\rangle$› *FrR2* **have** $A_P \sharp* \Psi_{R1}$ **and** $A_P \sharp* \Psi_{R2}$
by(*force dest*: *extractFrameFreshChain*)+
moreover note ‹*distinct xvec*›

ultimately have $\exists K'.\ (\Psi \otimes \Psi_{R2}) \otimes \Psi_P \rhd R_1 \longmapsto K'(\!|N|\!) \prec R_1{}' \wedge (\Psi \otimes \Psi_{R2}) \otimes \Psi_P \otimes \Psi_{R1} \vdash M \leftrightarrow K' \wedge (A_{R2}@zvec) \sharp* K' \wedge A_{R1} \sharp* K'$ **using** *cPar1*

by(*rule-tac cPar1*(*6*)[**where** *bf*=*xvec*]) (*assumption* | *simp* | *fastforce*)+
then obtain K' **where** *RTrans*: $(\Psi \otimes \Psi_{R2}) \otimes \Psi_P \rhd R_1 \longmapsto K'(\!|N|\!) \prec R_1{}'$
and *MeqK'*: $(\Psi \otimes \Psi_{R2}) \otimes \Psi_P \otimes \Psi_{R1} \vdash M \leftrightarrow K'$ **and** $A_{R2} \sharp* K'$ **and** $zvec \sharp* K'$ **and** $A_{R1} \sharp* K'$
by *force*

from *RTrans* **have** $(\Psi \otimes \Psi_P) \otimes \Psi_{R2} \rhd R_1 \longmapsto K'(\!|N|\!) \prec R_1{}'$
by(*metis statEqTransition Associativity Composition Commutativity*)
hence $\Psi \otimes \Psi_P \rhd (R_1 \parallel R_2) \longmapsto K'(\!|N|\!) \prec (R_1{}' \parallel R_2)$ **using** *FrR2* ‹$A_{R2} \sharp* \Psi$› ‹$A_{R2} \sharp* \Psi_P$› ‹$A_{R2} \sharp* K'$› ‹$A_{R2} \sharp* R_1$› ‹$A_{R2} \sharp* N$›
by(*force intro*: *Par1*)
moreover from *MeqK'* **have** $\Psi \otimes \Psi_P \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow K'$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹$zvec \sharp* K'$› ‹$A_{R1} \sharp* K'$› ‹$A_{R2} \sharp* K'$›
by *force*
next
case(*cPar2* Ψ' Ψ_{R1} R_2 M' N $R_2{}'$ A_{R1} R_1 A_{R2} Ψ_{R2} Ψ P A_P Ψ_P A_Q *zvec* *xvec* M)
have *FrR1*: *extractFrame* $R_1 = \langle A_{R1}, \Psi_{R1}\rangle$ **by** *fact*
from ‹$\Psi' \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow M'$› **have** $(\Psi' \otimes \Psi_{R1}) \otimes \Psi_{R2} \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Composition Commutativity*)
moreover have $\langle A_Q, (\Psi' \otimes \Psi_{R1}) \otimes \Psi_{R2}\rangle \hookrightarrow_F \langle A_P, ((\Psi \otimes \Psi_{R1}) \otimes \Psi_P) \otimes \Psi_{R2}\rangle$
proof −
have $\langle A_Q, (\Psi' \otimes \Psi_{R1}) \otimes \Psi_{R2}\rangle \simeq_F \langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
moreover note ‹$\langle A_Q, \Psi' \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle$›
moreover have $\langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_{R1} \otimes \Psi_{R2}\rangle \simeq_F \langle A_P, ((\Psi \otimes \Psi_{R1}) \otimes \Psi_P) \otimes \Psi_{R2}\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \Psi_{R1} \otimes \Psi_{R2} \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› **have** $(\Psi \otimes \Psi_{R1}) \otimes \Psi_{R2} \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*metis statEqTransition Associativity Composition Commutativity*)
moreover from ‹$A_{R1} \sharp* A_P$› ‹$A_{R2} \sharp* A_P$› ‹$A_P \sharp* (R_1 \parallel R_2)$› *FrR1* ‹*extractFrame* $R_2 = \langle A_{R2}, \Psi_{R2}\rangle$› **have** $A_P \sharp* \Psi_{R1}$ **and** $A_P \sharp* \Psi_{R2}$
by(*force dest*: *extractFrameFreshChain*)+
ultimately have $\exists K'.\ (\Psi \otimes \Psi_{R1}) \otimes \Psi_P \rhd R_2 \longmapsto K'(\!|N|\!) \prec R_2{}' \wedge (\Psi \otimes \Psi_{R1}) \otimes \Psi_P \otimes \Psi_{R2} \vdash M \leftrightarrow K' \wedge (A_{R1}@zvec) \sharp* K' \wedge A_{R2} \sharp* K'$ **using** ‹*distinct xvec*› *cPar2*
by(*rule-tac cPar2*(*6*)) (*assumption* | *simp* | *fastforce*)+
then obtain K' **where** *RTrans*: $(\Psi \otimes \Psi_{R1}) \otimes \Psi_P \rhd R_2 \longmapsto K'(\!|N|\!) \prec R_2{}'$
and *MeqK'*: $(\Psi \otimes \Psi_{R1}) \otimes \Psi_P \otimes \Psi_{R2} \vdash M \leftrightarrow K'$ **and** $A_{R1} \sharp* K'$ **and** $zvec \sharp* K'$ **and** $A_{R2} \sharp* K'$
by *force*

from *RTrans* **have** $(\Psi \otimes \Psi_P) \otimes \Psi_{R1} \rhd R_2 \longmapsto K'(\!|N|\!) \prec R_2{}'$
by(*metis statEqTransition Associativity Composition Commutativity*)
hence $\Psi \otimes \Psi_P \rhd (R_1 \parallel R_2) \longmapsto K'(\!|N|\!) \prec (R_1 \parallel R_2{}')$ **using** *FrR1* ‹$A_{R1} \sharp* \Psi$› ‹$A_{R1} \sharp* \Psi_P$› ‹$A_{R1} \sharp* K'$› ‹$A_{R1} \sharp* R_2$› ‹$A_{R1} \sharp* N$›
by(*force intro*: *Par2*)
moreover from *MeqK'* **have** $\Psi \otimes \Psi_P \otimes \Psi_{R1} \otimes \Psi_{R2} \vdash M \leftrightarrow K'$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹*zvec* $\sharp* K'$› ‹$A_{R1} \sharp* K'$› ‹$A_{R2} \sharp* K'$›
by *force*
next
case(*cScope* Ψ' R M' N R' x A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec xvec* M)
then have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R' \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge$ *zvec* $\sharp* K' \wedge A_R \sharp* K'$
by(*rule-tac cScope*(*4*)) (*assumption* | *simp del*: *freshChainSimps*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$
and *MeqK'*: $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** *zvec* $\sharp* K'$ **and** $A_R \sharp* K'$
by *metis*
from ‹$A_R \sharp* A_P$› ‹$A_P \sharp* ((\!|\nu x|\!)R)$› ‹$x \sharp A_P$› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** $A_P \sharp* \Psi_R$
by(*force dest*: *extractFrameFreshChain*)+
from ‹$\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹*distinct* A_P› ‹$x \sharp P$› ‹*zvec* $\sharp* P$› ‹$A_P \sharp* \Psi_R$› ‹$x \sharp A_P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* P$› ‹$A_P \sharp*$ *zvec*› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp*$ *zvec*› ‹$A_R \sharp* P$› ‹$A_R \sharp* A_P$› ‹*xvec* $\sharp* M$› ‹*distinct xvec*›
obtain K'' **where** *MeqK''*: $(\Psi \otimes \Psi_R) \otimes \Psi_P \vdash M \leftrightarrow K''$ **and** $x \sharp K''$ **and** $A_R \sharp* K''$ **and** *zvec* $\sharp* K''$
by(*rule-tac* B=(x#A_R@*zvec*) **in** *obtainPrefix*) (*assumption* | *simp* | *force* | *metis freshChainSym*)+

from *MeqK''* *MeqK'* **have** *KeqK''*: $(\Psi \otimes \Psi_P) \otimes \Psi_R \vdash K' \leftrightarrow K''$
by(*metis statEqEnt Associativity Composition Commutativity chanEqSym chanEqTrans*)
with *RTrans* ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P$› ‹$A_R \sharp* K'$› ‹$A_R \sharp* K''$› ‹$A_R \sharp* R$›
have $\Psi \otimes \Psi_P \rhd R \longmapsto K''(\!|N|\!) \prec R'$
by(*rule-tac inputRenameSubject*) (*assumption* | *force*)+
hence $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)R \longmapsto K''(\!|N|\!) \prec (\!|\nu x|\!)R'$ **using** ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp K''$› ‹$x \sharp N$›
by(*rule-tac Scope*) (*assumption* | *force*)+
moreover from *MeqK''* **have** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K''$
by(*metis statEqEnt Associativity Composition Commutativity*)
ultimately show *?case* **using** ‹*zvec* $\sharp* K''$› ‹$x \sharp K''$› ‹$A_R \sharp* K''$›
by *force*
next
case(*cBang* Ψ' R M' N R' A_R Ψ_R Ψ P A_P Ψ_P A_Q *zvec xvec* M)
from ‹*guarded* R› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** $\Psi_R \simeq \mathbf{1}$
by(*metis guardedStatEq*)

with ‹$\Psi' \otimes \mathbf{1} \vdash M \leftrightarrow M'$› **have** $\Psi' \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow M'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition*)
moreover have $\langle A_Q, \Psi' \otimes \Psi_R \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R \otimes \mathbf{1}\rangle$
proof −
from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_Q, \Psi' \otimes \Psi_R \otimes \mathbf{1}\rangle \simeq_F \langle A_Q, \Psi' \otimes \mathbf{1}\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
moreover note ‹$\langle A_Q, \Psi' \otimes \mathbf{1}\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle$›
moreover from ‹$\Psi_R \simeq \mathbf{1}$› **have** $\langle A_P, (\Psi \otimes \Psi_P) \otimes \mathbf{1}\rangle \simeq_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R \otimes \mathbf{1}\rangle$
by(*metis Identity Commutativity AssertionStatEqSym Composition frameResChainPres frameNilStatEq AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
moreover from ‹$\Psi \otimes \mathbf{1} \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› ‹$\Psi_R \simeq \mathbf{1}$›
have $\Psi \otimes \Psi_R \otimes \mathbf{1} \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by**(*metis statEqTransition Identity Commutativity AssertionStatEqSym Composition*)
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \parallel !R \longmapsto K'(\!|N|\!) \prec R' \wedge \Psi \otimes \Psi_P \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K' \wedge zvec \sharp* K' \wedge A_R \sharp* K'$ **using** *cBang*
by(*rule-tac cBang*(*5*)) (*assumption* | *simp del*: *freshChainSimps*)+
then obtain K' **where** *RTrans*: $\Psi \otimes \Psi_P \rhd R \parallel !R \longmapsto K'(\!|N|\!) \prec R'$
and *MeqK'*: $\Psi \otimes \Psi_P \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K'$ **and** $zvec \sharp* K'$ **and** $A_R \sharp* K'$
by *metis*
from *RTrans* ‹*guarded R*› **have** $\Psi \otimes \Psi_P \rhd !R \longmapsto K'(\!|N|\!) \prec R'$ **by**(*rule Bang*)
moreover from *MeqK'* ‹$\Psi_R \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K'$
by(*metis Identity Commutativity statEqEnt AssertionStatEqSym Composition AssertionStatEqTrans*)
ultimately show *?case* **using** ‹$zvec \sharp* K'$›
by *force*
qed
}
note *Goal* = *this*
have $\Psi \otimes \Psi_Q \simeq \Psi \otimes \Psi_Q$ **by**(*simp add*: *AssertionStatEqRefl*)
moreover from ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_Q$› **have** $A_R \sharp* (\Psi \otimes \Psi_Q)$ **by** *force*
ultimately have $\exists K'.\ \Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R' \wedge \Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K' \wedge ([]::name\ list) \sharp* K' \wedge A_R \sharp* K'$
by(*rule-tac Goal*) (*assumption* | *force*)+
with *Assumptions* **show** *?thesis*
by *blast*
qed

end

end

theory *Simulation*
imports *Semantics*

begin

context *env* **begin**

definition
simulation :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$
$('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set* $\Rightarrow$
$('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- ▷ - ⇝[-] -› [*80*, *80*, *80*, *80*] *80*)
where
$\Psi \triangleright P \rightsquigarrow[Rel]\ Q \equiv \forall \alpha\ Q'.\ \Psi \triangleright Q \longmapsto \alpha \prec Q' \longrightarrow bn\ \alpha \sharp* \Psi \longrightarrow bn\ \alpha \sharp* P$
$\longrightarrow (\exists P'.\ \Psi \triangleright P \longmapsto \alpha \prec P' \wedge (\Psi, P', Q') \in Rel)$

abbreviation
simulationNilJudge (‹- ⇝[-] -› [*80*, *80*, *80*] *80*) **where** $P \rightsquigarrow[Rel]\ Q \equiv SBottom'$
$\triangleright P \rightsquigarrow[Rel]\ Q$

lemma *simI*[*consumes 1*, *case-names cSim*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Eqvt*: *eqvt Rel*
and *Sim*: $\bigwedge \alpha\ Q'.\ [\![\Psi \triangleright Q \longmapsto \alpha \prec Q';\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* Q;\ bn\ \alpha \sharp* \Psi;$
$distinct(bn\ \alpha);$
$bn\ \alpha \sharp* (subject\ \alpha);\ bn\ \alpha \sharp* C]\!] \Longrightarrow \exists P'.\ \Psi \triangleright P \longmapsto \alpha \prec P'$
$\wedge (\Psi, P', Q') \in Rel$

shows $\Psi \triangleright P \rightsquigarrow[Rel]\ Q$
proof(*auto simp add*: *simulation-def*)
fix $\alpha\ Q'$
assume $\Psi \triangleright Q \longmapsto \alpha \prec Q'$ **and** $bn\ \alpha \sharp* \Psi$ **and** $bn\ \alpha \sharp* P$
thus $\exists P'.\ \Psi \triangleright P \longmapsto \alpha \prec P' \wedge (\Psi, P', Q') \in Rel$
proof(*nominal-induct* α *rule*: *action.strong-induct*)
case(*In M N*)
thus *?case* **by**(*rule-tac Sim*) *auto*
next
case(*Out M xvec N*)
moreover {
fix *M xvec N Q'*
assume (*xvec*::*name list*) $\sharp*$ Ψ **and** *xvec* $\sharp*$ P
obtain p **where** *xvecFreshPsi*: $((p$::*name prm*$) \cdot ($*xvec*::*name list*$)) \sharp* \Psi$
and *xvecFreshM*: $(p \cdot xvec) \sharp* (M::'a)$
and *xvecFreshN*: $(p \cdot xvec) \sharp* (N::'a)$
and *xvecFreshP*: $(p \cdot xvec) \sharp* P$
and *xvecFreshQ*: $(p \cdot xvec) \sharp* Q$
and *xvecFreshQ'*: $(p \cdot xvec) \sharp* (Q'::('a, 'b, 'c)\ psi)$
and *xvecFreshC*: $(p \cdot xvec) \sharp* C$

and *xvecFreshxvec*: $(p \cdot xvec) \sharp* xvec$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$
and *dpr*: *distinctPerm p*
by(*rule-tac xvec=xvec* **and** $c=(\Psi, M, Q, N, P, Q', xvec, C)$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)

from ‹$(p \cdot xvec) \sharp* M$› ‹*distinctPerm p*› **have** $xvec \sharp* (p \cdot M)$
by(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, **where** *pi=p*, *symmetric*]) *simp*

assume *Trans*: $\Psi \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$
with *xvecFreshN xvecFreshQ' S*
have $\Psi \rhd Q \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot Q')$
by(*simp add*: *boundOutputChainAlpha'' residualInject*)
moreover hence $distinct(p \cdot xvec)$ **by**(*auto dest*: *boundOutputDistinct*)

moreover note *xvecFreshPsi xvecFreshP xvecFreshQ xvecFreshM xvecFreshC*
ultimately obtain P' **where** *PTrans*: $\Psi \rhd P \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec P'$
and $P'RelQ'$: $(\Psi, P', p \cdot Q') \in Rel$
by(*drule-tac Sim*) *auto*
hence $(p \cdot \Psi) \rhd (p \cdot P) \longmapsto (p \cdot (M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec P'))$
by(*rule-tac semantics.eqvt*)
with ‹$xvec \sharp* \Psi$› *xvecFreshPsi* ‹$xvec \sharp* P$› *xvecFreshP S dpr*
have $\Psi \rhd P \longmapsto (p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec (p \cdot P')$
by(*simp add*: *eqvts name-set-fresh-fresh*)
with ‹$xvec \sharp* \Psi$› *xvecFreshPsi* ‹$xvec \sharp* P$› *xvecFreshP S* ‹$xvec \sharp* (p \cdot M)$›
have $\Psi \rhd P \longmapsto (p \cdot p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec (p \cdot P')$
by(*rule-tac outputPermSubject*)
(*simp add*: *fresh-star-def* | *assumption*)+

with *dpr* **have** $\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec (p \cdot P')$
by *simp*

moreover from $P'RelQ'$ *Eqvt* **have** $(p \cdot \Psi, p \cdot P', p \cdot p \cdot Q') \in Rel$
apply(*simp add*: *eqvt-def eqvts*)
apply(*erule-tac* $x=(\Psi, P', p \cdot Q')$ **in** *ballE*)
apply(*erule-tac x=p* **in** *allE*)
by(*auto simp add*: *eqvts*)

with ‹$xvec \sharp* \Psi$› *xvecFreshPsi S dpr* **have** $(\Psi, p \cdot P', Q') \in Rel$ **by** *simp*
ultimately have $\exists P'.\ \Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P' \wedge (\Psi, P', Q') \in Rel$
by *blast*
}
ultimately show *?case* **by** *force*
next
case *Tau*
thus *?case* **by**(*rule-tac Sim*) *auto*

qed
qed

lemma *simI2*[*case-names cSim*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Sim*: $\bigwedge\alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q'; bn\ \alpha \sharp* P; bn\ \alpha \sharp* \Psi; distinct(bn\ \alpha)]\!] \Longrightarrow \exists P'.\ \Psi \rhd P \longmapsto\alpha \prec P' \wedge (\Psi, P', Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
using *assms*
by(*auto simp add*: *simulation-def dest*: *boundOutputDistinct*)

lemma *simIChainFresh*[*consumes 4*, *case-names cSim*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*
and C :: $'d$::*fs-name*

assumes *Eqvt*: *eqvt Rel*
and $xvec \sharp* \Psi$
and $xvec \sharp* P$
and $xvec \sharp* Q$
and *Sim*: $\bigwedge\alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q'; bn\ \alpha \sharp* P; bn\ \alpha \sharp* Q; bn\ \alpha \sharp* \Psi; bn\ \alpha \sharp* subject\ \alpha; distinct(bn\ \alpha); bn\ \alpha \sharp* C; xvec \sharp* \alpha; xvec \sharp* Q']\!] \Longrightarrow \exists P'.\ \Psi \rhd P \longmapsto\alpha \prec P' \wedge (\Psi, P', Q') \in Rel$
shows $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
using ‹*eqvt Rel*›
proof(*induct rule*: *simI*[**where** C=($xvec$, C)])
case(*cSim* $\alpha\ Q'$)
from ‹$bn\ \alpha \sharp* (xvec, C)$› **have** $bn\ \alpha \sharp* xvec$ **and** $bn\ \alpha \sharp* C$ **by**(*simp add*: *freshChainSym*)+
obtain p::*name prm* **where** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$
and $(p \cdot xvec) \sharp* \alpha$ **and** S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$
and *distinctPerm p*
by(*rule-tac* c=(Ψ, P, Q, α) **and** $xvec$=$xvec$ **in** *name-list-avoiding*) *auto*
show *?case*
proof(*cases rule*: *actionCases*[**where** α=α])
case(*cInput M N*)
from ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹$\alpha$=$M(\!|N|\!)$› **have** $(p \cdot \Psi) \rhd (p \cdot Q) \longmapsto(p \cdot (M(\!|N|\!) \prec Q'))$

by(*fastforce intro*: *semantics.eqvt*)
with ‹*xvec* ♯∗ Ψ› ‹(*p* · *xvec*) ♯∗ Ψ› ‹*xvec* ♯∗ *Q*› ‹(*p* · *xvec*) ♯∗ *Q*› *S*
have *QTrans*: Ψ ⊳ *Q* ⟼(*p* · *M*)⦇(*p* · *N*)⦈ ≺ (*p* · *Q*′)
by(*simp add*: *eqvts*)
moreover from ‹(*p* · *xvec*) ♯∗ α› **have** (*p* · (*p* · *xvec*)) ♯∗ (*p* · α)
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› ‹α=*M*⦇*N*⦈› **have** *xvec* ♯∗ (*p* · *M*) **and** *xvec* ♯∗ (*p* · *N*) **by** *simp*+
moreover with *QTrans* ‹*xvec* ♯∗ *Q*› **have** *xvec* ♯∗ (*p* · *Q*′) **by**(*rule-tac input-FreshChainDerivative*)
ultimately have ∃ *P*′. Ψ ⊳ *P* ⟼ (*p* · *M*)⦇(*p* · *N*)⦈ ≺ *P*′ ∧ (Ψ, *P*′, (*p* · *Q*′)) ∈ *Rel*
by(*rule-tac Sim*) (*assumption* | *simp*)+
then obtain *P*′ **where** *PTrans*: Ψ ⊳ *P* ⟼ (*p* · *M*)⦇(*p* · *N*)⦈ ≺ *P*′ **and** *P*′*RelQ*′: (Ψ, *P*′, (*p* · *Q*′)) ∈ *Rel*
by *blast*
from *PTrans* **have** (*p* · Ψ) ⊳ (*p* · *P*) ⟼ (*p* · ((*p* · *M*)⦇(*p* · *N*)⦈ ≺ *P*′))
by(*rule semantics.eqvt*)
with ‹*xvec* ♯∗ Ψ› ‹(*p* · *xvec*) ♯∗ Ψ› ‹*xvec* ♯∗ *P*› ‹(*p* · *xvec*) ♯∗ *P*› *S* ‹*distinctPerm p*›
have Ψ ⊳ *P* ⟼ *M*⦇*N*⦈ ≺ (*p* · *P*′) **by**(*simp add*: *eqvts*)
moreover from *P*′*RelQ*′ ‹*eqvt Rel*› **have** (*p* · Ψ, *p* · *P*′, *p* · *p* · *Q*′) ∈ *Rel*
by(*auto simp add*: *eqvt-def*)
with ‹*xvec* ♯∗ Ψ› ‹(*p* · *xvec*) ♯∗ Ψ› *S* ‹*distinctPerm p*›
have (Ψ, *p* · *P*′, *Q*′) ∈ *Rel* **by** *simp*
ultimately show *?thesis* **using** ‹α=*M*⦇*N*⦈› **by** *blast*
next
case(*cOutput M yvec N*)
from ‹*distinct*(*bn* α)› ‹*bn* α ♯∗ *subject* α› ‹α=*M*⦇ν∗*yvec*⦈⟨*N*⟩› **have** *distinct yvec* **and** *yvec* ♯∗ *M* **by** *simp*+
from ‹Ψ ⊳ *Q* ⟼α ≺ *Q*′› ‹α=*M*⦇ν∗*yvec*⦈⟨*N*⟩› **have** (*p* · Ψ) ⊳ (*p* · *Q*) ⟼(*p* · (*M*⦇ν∗*yvec*⦈⟨*N*⟩ ≺ *Q*′))
by(*fastforce intro*: *semantics.eqvt*)
with ‹*xvec* ♯∗ Ψ› ‹(*p* · *xvec*) ♯∗ Ψ› ‹*xvec* ♯∗ *Q*› ‹(*p* · *xvec*) ♯∗ *Q*› *S*
have *QTrans*: Ψ ⊳ *Q* ⟼(*p* · *M*)⦇ν∗(*p* · *yvec*)⦈⟨(*p* · *N*)⟩ ≺ (*p* · *Q*′)
by(*simp add*: *eqvts*)
with *S* ‹*bn* α ♯∗ *xvec*› ‹(*p* · *xvec*) ♯∗ α› ‹α=*M*⦇ν∗*yvec*⦈⟨*N*⟩› **have** Ψ ⊳ *Q* ⟼(*p* · *M*)⦇ν∗*yvec*⦈⟨(*p* · *N*)⟩ ≺ (*p* · *Q*′)
by *simp*
moreover from ‹(*p* · *xvec*) ♯∗ α› **have** (*p* · (*p* · *xvec*)) ♯∗ (*p* · α)
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› ‹α=*M*⦇ν∗*yvec*⦈⟨*N*⟩› **have** *xvec* ♯∗ (*p* · *M*) **and** *xvec* ♯∗ (*p* · *N*) **and** *xvec* ♯∗ (*p* · *yvec*) **by** *simp*+
moreover with *QTrans* ‹*xvec* ♯∗ *Q*› ‹*distinct yvec*› ‹*yvec* ♯∗ *M*› **have** *xvec* ♯∗ (*p* · *Q*′)
by(*drule-tac outputFreshChainDerivative*(*2*)) (*auto simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹*yvec* ♯∗ *M*› *S* ‹*bn* α ♯∗ *xvec*› ‹(*p* · *xvec*) ♯∗ α› ‹α=*M*⦇ν∗*yvec*⦈⟨*N*⟩› ‹*distinctPerm p*›

have *yvec* $\sharp*$ $(p \cdot M)$ **by**(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, *symmetric*, **where** *pi*=*p*]) *simp*
ultimately have $\exists P'.\ \Psi \rhd P \longmapsto (p \cdot M)(\!|\nu{*}yvec|\!)\langle(p \cdot N)\rangle \prec P' \wedge (\Psi,\ P',\ (p \cdot Q')) \in Rel$
using ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ P› ‹*bn* α $\sharp*$ Q›‹*bn* α $\sharp*$ *xvec*› ‹*bn* α $\sharp*$ C› ‹*yvec* $\sharp*$ M› ‹α=$M(\!|\nu{*}yvec|\!)\langle N\rangle$› ‹*distinct yvec*›
by(*rule-tac Sim*) *auto*
then obtain P' **where** *PTrans*: $\Psi \rhd P \longmapsto (p \cdot M)(\!|\nu{*}yvec|\!)\langle(p \cdot N)\rangle \prec P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ (p \cdot Q')) \in Rel$
by *blast*
from *PTrans* **have** $(p \cdot \Psi) \rhd (p \cdot P) \longmapsto (p \cdot ((p \cdot M)(\!|\nu{*}yvec|\!)\langle(p \cdot N)\rangle \prec P'))$
by(*rule semantics.eqvt*)
with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec)$ $\sharp*$ Ψ› ‹*xvec* $\sharp*$ P› ‹$(p \cdot xvec)$ $\sharp*$ P› S ‹*distinctPerm p*› ‹*bn* α $\sharp*$ *xvec*› ‹$(p \cdot xvec)$ $\sharp*$ α› ‹α=$M(\!|\nu{*}yvec|\!)\langle N\rangle$›
have $\Psi \rhd P \longmapsto M(\!|\nu{*}yvec|\!)\langle N\rangle \prec (p \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from $P'RelQ'$ ‹*eqvt Rel*› **have** $(p \cdot \Psi,\ p \cdot P',\ p \cdot p \cdot Q') \in Rel$
by(*auto simp add*: *eqvt-def*)
with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec)$ $\sharp*$ Ψ› S ‹*distinctPerm p*›
have $(\Psi,\ p \cdot P',\ Q') \in Rel$ **by** *simp*
ultimately show *?thesis* **using** ‹α=$M(\!|\nu{*}yvec|\!)\langle N\rangle$› **by** *blast*
next
case *cTau*
from ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹$\alpha = \tau$› ‹*xvec* $\sharp*$ Q› **have** *xvec* $\sharp*$ Q'
by(*blast dest*: *tauFreshChainDerivative*)
with ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹$\alpha = \tau$›
show *?thesis* **by**(*drule-tac Sim*) *auto*
qed
qed

lemma *simIFresh*[*consumes 4*, *case-names cSim*]:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and Q :: (*'a*, *'b*, *'c*) *psi*
and x :: *name*
and C :: *'d*::*fs-name*

assumes *Eqvt*: *eqvt Rel*
and $x \sharp \Psi$
and $x \sharp P$
and $x \sharp Q$
and $\bigwedge\alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* Q;\ bn\ \alpha \sharp* \Psi;$
$bn\ \alpha \sharp* subject\ \alpha;\ distinct(bn\ \alpha);\ bn\ \alpha \sharp* C;\ x \sharp \alpha;\ x \sharp Q']\!] \Longrightarrow$
$\exists P'.\ \Psi \rhd P \longmapsto\alpha \prec P' \wedge (\Psi,\ P',\ Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
using *assms*
by(*rule-tac xvec*=[*x*] **and** *C*=*C* **in** *simIChainFresh*) *auto*

lemma *simE*:
  fixes F :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*

  assumes $\Psi \rhd P \rightsquigarrow[Rel]\ Q$

  shows $\bigwedge\alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P]\!] \Longrightarrow \exists P'.\ \Psi \rhd P \longmapsto\alpha \prec P' \wedge (\Psi, P', Q') \in Rel$
using *assms*
by(*auto simp add*: *simulation-def*)

lemma *simClosedAux*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*
  and p :: *name prm*

  assumes *EqvtRel*: *eqvt Rel*
  and *PSimQ*: $\Psi \rhd P \rightsquigarrow[Rel]\ Q$

  shows $(p \cdot \Psi) \rhd (p \cdot P) \rightsquigarrow[Rel]\ (p \cdot Q)$
using *EqvtRel*
proof(*induct rule*: *simI*[*of* - - - - (Ψ, P, p)])
  case(*cSim* $\alpha\ Q'$)
  from ‹$p \cdot \Psi \rhd p \cdot Q \longmapsto\alpha \prec Q'$›
  have $(rev\ p \cdot p \cdot \Psi) \rhd (rev\ p \cdot p \cdot Q) \longmapsto(rev\ p \cdot (\alpha \prec Q'))$
    by(*blast dest*: *semantics.eqvt*)
  hence $\Psi \rhd Q \longmapsto(rev\ p \cdot \alpha) \prec (rev\ p \cdot Q')$
    by(*simp add*: *eqvts*)
  moreover with ‹$bn\ \alpha \sharp* (\Psi, P, p)$› **have** $bn\ \alpha \sharp* \Psi$ **and** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* p$ **by** *simp*+
  ultimately obtain P' **where** *PTrans*: $\Psi \rhd P \longmapsto(rev\ p \cdot \alpha) \prec P'$
                and $P'RelQ'$: $(\Psi, P', rev\ p \cdot Q') \in Rel$
    using *PSimQ*
    by(*force dest*: *simE freshChainPermSimp simp add*: *eqvts*)
  from *PTrans* **have** $(p \cdot \Psi) \rhd (p \cdot P) \longmapsto(p \cdot ((rev\ p \cdot \alpha) \prec P'))$
    by(*rule semantics.eqvt*)
  with ‹$bn\ \alpha \sharp* p$› **have** $(p \cdot \Psi) \rhd (p \cdot P) \longmapsto\alpha \prec (p \cdot P')$
    by(*simp add*: *eqvts freshChainPermSimp*)
  moreover from $P'RelQ'$ *EqvtRel* **have** $(p \cdot (\Psi, P', rev\ p \cdot Q')) \in Rel$
    by(*simp only*: *eqvt-def*)
  hence $(p \cdot \Psi, p \cdot P', Q') \in Rel$ **by** *simp*
  ultimately show *?case* **by** *blast*
qed

lemma *simClosed*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*
  and p :: *name prm*

  assumes *EqvtRel*: *eqvt Rel*

  shows $\Psi \rhd P \rightsquigarrow[Rel]\ Q \Longrightarrow (p \cdot \Psi) \rhd (p \cdot P) \rightsquigarrow[Rel]\ (p \cdot Q)$
  and $P \rightsquigarrow[Rel]\ Q \Longrightarrow (p \cdot P) \rightsquigarrow[Rel]\ (p \cdot Q)$
using *EqvtRel*
by(*force dest*: *simClosedAux simp add*: *permBottom*)+

lemma *reflexive*:
  fixes *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*

  assumes $\{(\Psi, P, P) \mid \Psi\ P.\ True\} \subseteq Rel$

  shows $\Psi \rhd P \rightsquigarrow[Rel]\ P$
using *assms*
by(*auto simp add*: *simulation-def*)

lemma *transitive*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*
  and *Rel'* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and R :: $('a, 'b, 'c)$ *psi*
  and *Rel''* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*

  assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
  and *QSimR*: $\Psi \rhd Q \rightsquigarrow[Rel']\ R$
  and *Eqvt*: *eqvt Rel''*
  and *Set*: $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi, P, Q) \in Rel \wedge (\Psi, Q, R) \in Rel'\} \subseteq$
Rel''

  shows $\Psi \rhd P \rightsquigarrow[Rel'']\ R$
using ‹*eqvt Rel''*›
proof(*induct rule*: *simI*[**where** $C{=}Q$])
  case(*cSim* α R')
  from *QSimR* ‹$\Psi \rhd R \longmapsto\alpha \prec R'$› ‹$(bn\ \alpha) \sharp* \Psi$› ‹$(bn\ \alpha) \sharp* Q$›
  obtain Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto\alpha \prec Q'$ **and** $Q'Rel'R'$: $(\Psi, Q', R') \in Rel'$
    by(*blast dest*: *simE*)
  from *PSimQ QTrans* ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$›
  obtain P' **where** *PTrans*: $\Psi \rhd P \longmapsto\alpha \prec P'$ **and** $P'RelQ'$: $(\Psi, P', Q') \in Rel$

by(*blast dest*: *simE*)
with *PTrans Q'Rel'R' P'RelQ' Set*
show *?case* **by** *blast*
qed

lemma *statEqSim*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and Ψ' :: *'b*

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
and *eqvt Rel'*
and $\Psi \simeq \Psi'$
and *C1*: $\bigwedge \Psi''\ R\ S\ \Psi'''.\ [\![(\Psi'', R, S) \in Rel;\ \Psi'' \simeq \Psi''']\!] \Longrightarrow (\Psi''', R, S) \in Rel'$

shows $\Psi' \rhd P \rightsquigarrow[Rel']\ Q$
using ‹*eqvt Rel'*›
proof(*induct rule*: *simI*[*of* - - - - Ψ])
case(*cSim* α *Q'*)
from ‹$\Psi' \rhd Q \longmapsto \alpha \prec Q'$› ‹$\Psi \simeq \Psi'$›
have $\Psi \rhd Q \longmapsto \alpha \prec Q'$ **by**(*metis statEqTransition AssertionStatEqSym*)
with *PSimQ* ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ *P*›
obtain *P'* **where** $\Psi \rhd P \longmapsto \alpha \prec P'$ **and** $(\Psi, P', Q') \in Rel$
by(*blast dest*: *simE*)

from ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› ‹$\Psi \simeq \Psi'$› **have** $\Psi' \rhd P \longmapsto \alpha \prec P'$
by(*rule statEqTransition*)
moreover from ‹$(\Psi, P', Q') \in Rel$› ‹$\Psi \simeq \Psi'$› **have** $(\Psi', P', Q') \in Rel'$
by(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *monotonic*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *A* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *B* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*

assumes $\Psi \rhd P \rightsquigarrow[A]\ Q$
and $A \subseteq B$

shows $\Psi \rhd P \rightsquigarrow[B]\ Q$
using *assms*
by(*simp* (*no-asm*) *add*: *simulation-def*) (*auto dest*: *simE*)

end

end

theory *Tau-Chain*
  imports *Semantics*
begin

context *env* **begin**

abbreviation *tauChain* :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- $\triangleright$ - $\Longrightarrow\hat{}_\tau$ -› [*80*, *80*, *80*] *80*)
where $\Psi \triangleright P \Longrightarrow\hat{}_\tau P' \equiv (P, P') \in \{(P, P').\ \Psi \triangleright P \longmapsto\tau \prec P'\}\hat{}*$

abbreviation *tauStepChain* :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- $\triangleright$ - $\Longrightarrow_\tau$ -› [*80*, *80*, *80*] *80*)
where $\Psi \triangleright P \Longrightarrow_\tau P' \equiv (P, P') \in \{(P, P').\ \Psi \triangleright P \longmapsto\tau \prec P'\}\hat{}+$

abbreviation *tauContextChain* :: $('a, 'b, 'c)$ *psi* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- $\Longrightarrow\hat{}_\tau$ -› [*80*, *80*] *80*)
where $P \Longrightarrow\hat{}_\tau P' \equiv \mathbf{1} \triangleright P \Longrightarrow\hat{}_\tau P'$
abbreviation *tauContextStepChain* :: $('a, 'b, 'c)$ *psi* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- $\Longrightarrow_\tau$ -› [*80*, *80*] *80*)
where $P \Longrightarrow_\tau P' \equiv \mathbf{1} \triangleright P \Longrightarrow_\tau P'$

lemmas *tauChainInduct*[*consumes 1*, *case-names TauBase TauStep*] = *rtrancl.induct*[*of* - - $\{(P, P').\ \Psi \triangleright P \longmapsto\tau \prec P'\}$, *simplified*] **for** Ψ
lemmas *tauStepChainInduct*[*consumes 1*, *case-names TauBase TauStep*] = *trancl.induct*[*of* - - $\{(P, P').\ \Psi \triangleright P \longmapsto\tau \prec P'\}$, *simplified*] **for** Ψ

lemma *tauActTauStepChain*:
  fixes $\Psi :: 'b$
  and $P :: ('a, 'b, 'c)$ *psi*
  and $P' :: ('a, 'b, 'c)$ *psi*

  assumes $\Psi \triangleright P \longmapsto\tau \prec P'$

  shows $\Psi \triangleright P \Longrightarrow_\tau P'$
using *assms* **by** *auto*

lemma *tauActTauChain*:
  fixes $\Psi :: 'b$
  and $P :: ('a, 'b, 'c)$ *psi*
  and $P' :: ('a, 'b, 'c)$ *psi*

  assumes $\Psi \triangleright P \longmapsto\tau \prec P'$

  shows $\Psi \triangleright P \Longrightarrow\hat{}_\tau P'$

using *assms* **by**(*auto simp add*: *rtrancl-eq-or-trancl*)

lemma *tauStepChainEqvt*[*eqvt*]:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*
  and p :: *name prm*

  assumes $\Psi \rhd P \Longrightarrow_\tau P'$

  shows $(p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow_\tau (p \cdot P')$
using *assms*
proof(*induct rule*: *tauStepChainInduct*)
  case(*TauBase P P'*)
  hence $\Psi \rhd P \longmapsto\tau \prec P'$ **by** *simp*
  thus *?case* **by**(*force dest*: *semantics.eqvt simp add*: *eqvts*)
next
  case(*TauStep P P' P''*)
  hence $\Psi \rhd P' \longmapsto\tau \prec P''$ **by** *simp*
  hence $(p \cdot \Psi) \rhd (p \cdot P') \longmapsto\tau \prec (p \cdot P'')$ **by**(*force dest*: *semantics.eqvt simp add*: *eqvts*)
  with ‹$(p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow_\tau (p \cdot P')$› **show** *?case*
    by(*subst trancl.trancl-into-trancl*) *auto*
qed

lemma *tauChainEqvt*[*eqvt*]:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*
  and p :: *name prm*

  assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$

  shows $(p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow\hat{}_\tau (p \cdot P')$
using *assms*
by(*auto simp add*: *rtrancl-eq-or-trancl eqvts*)

lemma *tauStepChainEqvt'*[*eqvt*]:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*
  and p :: *name prm*

  shows $(p \cdot (\Psi \rhd P \Longrightarrow_\tau P')) = (p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow_\tau (p \cdot P')$
apply(*auto simp add*: *eqvts perm-set-def pt-bij*[*OF pt-name-inst*, *OF at-name-inst*])
by(*drule-tac p=rev p* **in** *tauStepChainEqvt*) *auto*

lemma *tauChainEqvt'*[*eqvt*]:
  fixes Ψ :: $'b$

and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*

shows $(p \cdot (\Psi \rhd P \Longrightarrow\hat{}_\tau P')) = (p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow\hat{}_\tau (p \cdot P')$
apply(*auto simp add*: *eqvts perm-set-def pt-bij*[*OF pt-name-inst*, *OF at-name-inst*] *rtrancl-eq-or-trancl*)
by(*drule-tac p=rev p* **in** *tauStepChainEqvt*) *auto*

lemma *tauStepChainFresh*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes $\Psi \rhd P \Longrightarrow_\tau P'$
and $x \sharp P$

shows $x \sharp P'$
using *assms*
by(*induct rule*: *trancl.induct*) (*auto dest*: *tauFreshDerivative*)

lemma *tauChainFresh*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
and $x \sharp P$

shows $x \sharp P'$
using *assms*
by(*auto simp add*: *rtrancl-eq-or-trancl intro*: *tauStepChainFresh*)

lemma *tauStepChainFreshChain*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \Longrightarrow_\tau P'$
and $xvec \sharp* P$

shows $xvec \sharp* P'$
using *assms*
by(*induct xvec*) (*auto intro*: *tauStepChainFresh*)

lemma *tauChainFreshChain*:

fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \Longrightarrow\hat{}_{\tau} P'$
and *xvec* $\sharp*$ P

shows *xvec* $\sharp*$ P'
using *assms*
by(*induct xvec*) (*auto intro*: *tauChainFresh*)

lemma *tauStepChainCase*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and φ :: $'c$
and *Cs* :: $('c \times ('a, 'b, 'c)$ *psi*$)$ *list*

assumes $\Psi \rhd P \Longrightarrow_{\tau} P'$
and (φ, P) *mem Cs*
and $\Psi \vdash \varphi$
and *guarded P*

shows $\Psi \rhd$ (*Cases Cs*) $\Longrightarrow_{\tau} P'$
using *assms*
by(*induct rule*: *trancl.induct*) (*auto intro*: *Case trancl-into-trancl*)

lemma *tauStepChainResPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes $\Psi \rhd P \Longrightarrow_{\tau} P'$
and $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)P \Longrightarrow_{\tau} (\!|\nu x|\!)P'$
using *assms*
by(*induct rule*: *trancl.induct*) (*auto dest*: *Scope trancl-into-trancl*)

lemma *tauChainResPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes $\Psi \rhd P \Longrightarrow\hat{}_{\tau} P'$
and $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)P \Longrightarrow\hat{}_{\tau} (\!|\nu x|\!)P'$
using *assms*
by(*auto simp add*: *rtrancl-eq-or-trancl intro*: *tauStepChainResPres*)

lemma *tauStepChainResChainPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \Longrightarrow_{\tau} P'$
and *xvec* $\sharp*$ Ψ

shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \Longrightarrow_{\tau} (\!|\nu{*}xvec|\!)P'$
using *assms*
by(*induct xvec*) (*auto intro*: *tauStepChainResPres*)

lemma *tauChainResChainPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \Longrightarrow\hat{}_{\tau} P'$
and *xvec* $\sharp*$ Ψ

shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \Longrightarrow\hat{}_{\tau} (\!|\nu{*}xvec|\!)P'$
using *assms*
by(*induct xvec*) (*auto intro*: *tauChainResPres*)

lemma *tauStepChainPar1*:
fixes Ψ :: $'b$
and Ψ_Q :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and A_Q :: *name list*

assumes $\Psi \otimes \Psi_Q \rhd P \Longrightarrow_{\tau} P'$
and *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$

shows $\Psi \rhd P \parallel Q \Longrightarrow_{\tau} P' \parallel Q$
using *assms*
by(*induct rule*: *trancl.induct*) (*auto dest*: *Par1 tauStepChainFreshChain trancl-into-trancl*)

lemma *tauChainPar1*:

fixes Ψ :: $'b$
and Ψ_Q :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and A_Q :: *name list*

assumes $\Psi \otimes \Psi_Q \rhd P \Longrightarrow\hat{}_\tau P'$
and *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$

shows $\Psi \rhd P \parallel Q \Longrightarrow\hat{}_\tau P' \parallel Q$
using *assms*
by(*auto simp add*: *rtrancl-eq-or-trancl intro*: *tauStepChainPar1*)

lemma *tauStepChainPar2*:
fixes Ψ :: $'b$
and Ψ_P :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and Q' :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*

assumes $\Psi \otimes \Psi_P \rhd Q \Longrightarrow_\tau Q'$
and *extractFrame* $P = \langle A_P, \Psi_P \rangle$
and $A_P \sharp* \Psi$
and $A_P \sharp* Q$

shows $\Psi \rhd P \parallel Q \Longrightarrow_\tau P \parallel Q'$
using *assms*
by(*induct rule*: *trancl.induct*) (*auto dest*: *Par2 trancl-into-trancl tauStepChain-FreshChain*)

lemma *tauChainPar2*:
fixes Ψ :: $'b$
and Ψ_P :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and Q' :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*

assumes $\Psi \otimes \Psi_P \rhd Q \Longrightarrow\hat{}_\tau Q'$
and *extractFrame* $P = \langle A_P, \Psi_P \rangle$
and $A_P \sharp* \Psi$
and $A_P \sharp* Q$

shows $\Psi \rhd P \parallel Q \Longrightarrow\hat{}_\tau P \parallel Q'$
using *assms*

by(*auto simp add*: *rtrancl-eq-or-trancl intro*: *tauStepChainPar2*)

lemma *tauStepChainBang*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \parallel !P \Longrightarrow_\tau P'$
and *guarded* P

shows $\Psi \rhd !P \Longrightarrow_\tau P'$
using *assms*
by(*induct x1*==$P \parallel !P\ P'$ *rule*: *trancl.induct*) (*auto intro*: *Bang dest*: *Bang trancl-into-trancl*)

lemma *tauStepChainStatEq*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes $\Psi \rhd P \Longrightarrow_\tau P'$
and $\Psi \simeq \Psi'$

shows $\Psi' \rhd P \Longrightarrow_\tau P'$
using *assms*
by(*induct rule*: *trancl.induct*) (*auto dest*: *statEqTransition trancl-into-trancl*)

lemma *tauChainStatEq*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
and $\Psi \simeq \Psi'$

shows $\Psi' \rhd P \Longrightarrow\hat{}_\tau P'$
using *assms*
by(*auto simp add*: *rtrancl-eq-or-trancl intro*: *tauStepChainStatEq*)

definition *weakTransition* :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ $('a, 'b, 'c)$ *psi* $\Rightarrow$ $'a$ *action* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- : - ▷ - ⟹- ≺ -› [*80*, *80*, *80*, *80*, *80*] *80*)
where
$\Psi : Q \rhd P \Longrightarrow\alpha \prec P' \equiv \exists P''.\ \Psi \rhd P \Longrightarrow\hat{}_\tau P'' \wedge$ (*insertAssertion* (*extractFrame* Q) Ψ) $\hookrightarrow_F$ (*insertAssertion* (*extractFrame* P'') Ψ) $\wedge$
$\Psi \rhd P'' \longmapsto\alpha \prec P'$

lemma *weakTransitionI*:
fixes Ψ :: $'b$

and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and P'' :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$
and *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and $\Psi \rhd P'' \longmapsto\alpha \prec P'$

shows $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$
using *assms*
by(*auto simp add*: *weakTransition-def*)

lemma *weakTransitionE*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$

obtains P'' **where** $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **and** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and $\Psi \rhd P'' \longmapsto\alpha \prec P'$
using *assms*
by(*auto simp add*: *weakTransition-def*)

lemma *weakTransitionClosed*[*eqvt*]:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*

assumes $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$

shows $(p \cdot \Psi) : (p \cdot Q) \rhd (p \cdot P) \Longrightarrow(p \cdot \alpha)\prec (p \cdot P')$
proof −
from *assms* **obtain** P'' **where** $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **and** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and $\Psi \rhd P'' \longmapsto\alpha \prec P'$
by(*rule weakTransitionE*)

from ‹$\Psi \rhd P \Longrightarrow\hat{}_\tau P''$› **have** $(p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow\hat{}_\tau (p \cdot P'')$
by(*rule tauChainEqvt*)

moreover from ‹*insertAssertion* (*extractFrame Q*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame P''*) Ψ›
have $(p \cdot (\textit{insertAssertion}\ (\textit{extractFrame}\ Q)\ \Psi)) \hookrightarrow_F (p \cdot (\textit{insertAssertion}\ (\textit{extractFrame}\ P'')\ \Psi))$
by(*rule FrameStatImpClosed*)
hence *insertAssertion* (*extractFrame*$(p \cdot Q)$) $(p \cdot \Psi) \hookrightarrow_F$ *insertAssertion* (*extractFrame*$(p \cdot P'')$) $(p \cdot \Psi)$ **by**(*simp add*: *eqvts*)
moreover from ‹$\Psi \rhd P'' \longmapsto \alpha \prec P'$› **have** $(p \cdot \Psi) \rhd (p \cdot P'') \longmapsto (p \cdot (\alpha \prec P'))$
by(*rule semantics.eqvt*)
hence $(p \cdot \Psi) \rhd (p \cdot P'') \longmapsto (p \cdot \alpha) \prec (p \cdot P')$ **by**(*simp add*: *eqvts*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakOutputAlpha*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*
and *yvec* :: *name list*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec P'$
and S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$
and *distinctPerm p*
and $xvec \sharp* P$
and $xvec \sharp* (p \cdot xvec)$
and $(p \cdot xvec) \sharp* M$
and *distinct xvec*

shows $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*xvec|\!)\langle N\rangle \prec (p \cdot P')$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **and** *QeqP''*: *insertAssertion* (*extractFrame Q*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame P''*) Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec P'$
by(*rule weakTransitionE*)

note *PChain QeqP''*
moreover from *PChain* ‹$xvec \sharp* P$› **have** $xvec \sharp* P''$ **by**(*rule tauChainFreshChain*)
with *P''Trans* ‹$xvec \sharp* (p \cdot xvec)$› ‹*distinct xvec*› ‹$(p \cdot xvec) \sharp* M$› **have** $xvec \sharp* (p \cdot N)$ **and** $xvec \sharp* P'$
by(*force intro*: *outputFreshChainDerivative*)+
hence $(p \cdot xvec) \sharp* (p \cdot p \cdot N)$ **and** $(p \cdot xvec) \sharp* (p \cdot P')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])+
with ‹*distinctPerm p*› **have** $(p \cdot xvec) \sharp* N$ **and** $(p \cdot xvec) \sharp* (p \cdot P')$ **by** *simp*+

with $P''Trans\ S$ ‹*distinctPerm p*› **have** $\Psi \rhd P'' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec (p \cdot P')$
apply(*simp add*: *residualInject*)
by(*subst boundOutputChainAlpha*) *auto*

ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakFreshDerivative*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P'$
and $x \sharp P$
and $x \sharp \alpha$
and $bn\ \alpha \sharp{*}\ subject\ \alpha$
and $distinct(bn\ \alpha)$

shows $x \sharp P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_{\tau}\ P''$ **and** $P''Trans$: $\Psi \rhd P'' \longmapsto \alpha \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$x \sharp P$› **have** $x \sharp P''$ **by**(*rule tauChainFresh*)
with $P''Trans$ **show** $x \sharp P'$ **using** ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp{*}\ subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
by(*force intro*: *freeFreshDerivative*)
qed

lemma *weakFreshChainDerivative*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and *yvec* :: *name list*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P'$
and $yvec \sharp{*}\ P$
and $yvec \sharp{*}\ \alpha$
and $bn\ \alpha \sharp{*}\ subject\ \alpha$
and $distinct(bn\ \alpha)$

shows $yvec \sharp{*}\ P'$
using *assms*
by(*induct yvec*) (*auto intro*: *weakFreshDerivative*)

lemma *weakInputFreshDerivative*:
  fixes Ψ :: $'b$
  and Q :: $('a, 'b, 'c)$ *psi*
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$
  and P' :: $('a, 'b, 'c)$ *psi*
  and x :: *name*

  assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|N|\!) \prec P'$
  and $x \sharp P$
  and $x \sharp N$

  shows $x \sharp P'$
proof −
  from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_{\tau} P''$ **and** P''*Trans*: $\Psi \rhd P'' \longmapsto M(\!|N|\!) \prec P'$
    by(*rule weakTransitionE*)

  from *PChain* ‹$x \sharp P$› **have** $x \sharp P''$ **by**(*rule tauChainFresh*)
  with P''*Trans* **show** $x \sharp P'$ **using** ‹$x \sharp N$›
    by(*force intro*: *inputFreshDerivative*)
qed

lemma *weakInputFreshChainDerivative*:
  fixes Ψ :: $'b$
  and Q :: $('a, 'b, 'c)$ *psi*
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$
  and P' :: $('a, 'b, 'c)$ *psi*
  and *xvec* :: *name list*

  assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|N|\!) \prec P'$
  and *xvec* $\sharp*$ P
  and *xvec* $\sharp*$ N

  shows *xvec* $\sharp*$ P'
using *assms*
by(*induct xvec*) (*auto intro*: *weakInputFreshDerivative*)

lemma *weakOutputFreshDerivative*:
  fixes Ψ :: $'b$
  and Q :: $('a, 'b, 'c)$ *psi*
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and *xvec* :: *name list*
  and N :: $'a$

and P' :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $x \sharp P$
and $x \sharp xvec$
and $xvec \sharp{*} M$
and *distinct xvec*

shows $x \sharp N$
and $x \sharp P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **and** P''*Trans*: $\Psi \rhd P'' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$x \sharp P$› **have** $x \sharp P''$ **by**(*rule tauChainFresh*)
with P''*Trans* **show** $x \sharp N$ **and** $x \sharp P'$ **using** ‹$x \sharp xvec$› ‹$xvec \sharp{*} M$› ‹*distinct xvec*›
by(*force intro*: *outputFreshDerivative*)+
qed

lemma *weakOutputFreshChainDerivative*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and *yvec* :: *name list*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $yvec \sharp{*} P$
and $xvec \sharp{*} yvec$
and $xvec \sharp{*} M$
and *distinct xvec*

shows $yvec \sharp{*} N$
and $yvec \sharp{*} P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **and** P''*Trans*: $\Psi \rhd P'' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$yvec \sharp{*} P$› **have** $yvec \sharp{*} P''$ **by**(*rule tauChainFreshChain*)
with P''*Trans* **show** $yvec \sharp{*} N$ **and** $yvec \sharp{*} P'$ **using** ‹$xvec \sharp{*} yvec$› ‹$xvec \sharp{*} M$› ‹*distinct xvec*›
by(*force intro*: *outputFreshChainDerivative*)+

qed

lemma *weakOutputPermSubject*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*
and *yvec* :: *name list*
and *zvec* :: *name list*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and *S*: *set p* $\subseteq$ *set yvec* $\times$ *set zvec*
and *yvec* $\sharp*$ Ψ
and *zvec* $\sharp*$ Ψ
and *yvec* $\sharp*$ P
and *zvec* $\sharp*$ P

shows $\Psi : Q \rhd P \Longrightarrow (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_{\tau}\ P''$ **and** *QeqP''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹*yvec* $\sharp*$ P› ‹*zvec* $\sharp*$ P› **have** *yvec* $\sharp*$ P'' **and** *zvec* $\sharp*$ P''
by(*force intro*: *tauChainFreshChain*)+

note *PChain QeqP''*
moreover from *P''Trans S* ‹*yvec* $\sharp*$ Ψ› ‹*zvec* $\sharp*$ Ψ› ‹*yvec* $\sharp*$ P''› ‹*zvec* $\sharp*$ P''›
have $\Psi \rhd P'' \longmapsto (p \cdot M)(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule-tac outputPermSubject*) (*assumption* | *auto*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakInputPermSubject*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*
and *yvec* :: *name list*
and *zvec* :: *name list*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|N|\!) \prec P'$

and S: *set* $p \subseteq$ *set yvec* $\times$ *set zvec*
and *yvec* $\sharp*$ Ψ
and *zvec* $\sharp*$ Ψ
and *yvec* $\sharp*$ P
and *zvec* $\sharp*$ P

shows $\Psi : Q \rhd P \Longrightarrow (p \cdot M)(\!|N|\!) \prec P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_{\tau}\ P''$ **and** *QeqP''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto M(\!|N|\!) \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹*yvec* $\sharp*$ P› ‹*zvec* $\sharp*$ P› **have** *yvec* $\sharp*$ P'' **and** *zvec* $\sharp*$ P''
by(*force intro*: *tauChainFreshChain*)+

note *PChain QeqP''*
moreover from *P''Trans S* ‹*yvec* $\sharp*$ Ψ› ‹*zvec* $\sharp*$ Ψ› ‹*yvec* $\sharp*$ P''› ‹*zvec* $\sharp*$ P''› **have** $\Psi \rhd P'' \longmapsto (p \cdot M)(\!|N|\!) \prec P'$
by(*rule-tac inputPermSubject*) *auto*
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakInput*:
fixes Ψ :: *'b*
and Q :: (*'a*, *'b*, *'c*) *psi*
and M :: *'a*
and K :: *'a*
and *xvec* :: *name list*
and N :: *'a*
and *Tvec* :: *'a list*
and P :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \vdash M \leftrightarrow K$
and *distinct xvec*
and *set xvec* $\subseteq$ *supp N*
and *length xvec* = *length Tvec*
and *QeqΨ*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F \langle \varepsilon, \Psi \otimes \mathbf{1} \rangle$

shows $\Psi : Q \rhd M(\!|\lambda*xvec\ N|\!).P \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec P[xvec::=Tvec]$
proof −
have $\Psi \rhd M(\!|\lambda*xvec\ N|\!).P \Longrightarrow\hat{}_{\tau}\ M(\!|\lambda*xvec\ N|\!).P$ **by** *simp*
moreover from *QeqΨ* **have** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M(\!|\lambda*xvec\ N|\!).P$)) Ψ
by *auto*
moreover from *assms* **have** $\Psi \rhd M(\!|\lambda*xvec\ N|\!).P \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec P[xvec::=Tvec]$
by(*rule-tac Input*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)

qed

lemma *weakOutput*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and K :: $'a$
and N :: $'a$
and P :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \vdash M \leftrightarrow K$
and *QeqΨ*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$

shows $\Psi : Q \rhd M\langle N\rangle.P \Longrightarrow K\langle N\rangle \prec P$
proof −
have $\Psi \rhd M\langle N\rangle.P \Longrightarrow\hat{}_\tau M\langle N\rangle.P$ **by** *simp*
moreover from *QeqΨ* **have** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M\langle N\rangle.P$)) Ψ
by *auto*
moreover have *insertAssertion* (*extractFrame*($M\langle N\rangle.P$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M\langle N\rangle.P$)) Ψ **by** *simp*
moreover from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \rhd M\langle N\rangle.P \longmapsto K\langle N\rangle \prec P$
by(*rule Output*)
ultimately show *?thesis* **by**(*rule-tac weakTransitionI*) *auto*
qed

lemma *insertGuardedAssertion*:
fixes P :: $('a, 'b, 'c)$ *psi*

assumes *guarded P*

shows *insertAssertion*(*extractFrame* P) $\Psi \simeq_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$
proof −
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **by**(*rule freshFrame*)
from ‹*guarded P*› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **and** *supp* $\Psi_P = (\{\}::name\ set)$
by(*blast dest*: *guardedStatEq*)+

from *FrP* ‹$A_P \sharp* \Psi$› ‹$\Psi_P \simeq \mathbf{1}$› **have** *insertAssertion*(*extractFrame* P) $\Psi \simeq_F \langle A_P, \Psi \otimes \mathbf{1}\rangle$
by *simp* (*metis frameIntCompositionSym*)
moreover from ‹$A_P \sharp* \Psi$› **have** $\langle A_P, \Psi \otimes \mathbf{1}\rangle \simeq_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$
by(*rule-tac frameResFreshChain*) *auto*
ultimately show *?thesis* **by**(*rule FrameStatEqTrans*)
qed

lemma *weakCase*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*

and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$
and (φ, P) *mem CsP*
and $\Psi \vdash \varphi$
and *guarded P*
and *RImpQ*: *insertAssertion* (*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame Q*) Ψ
and *ImpR*: *insertAssertion* (*extractFrame R*) $\Psi \hookrightarrow_F \langle\varepsilon, \Psi\rangle$

shows $\Psi : R \rhd$ *Cases CsP* $\Longrightarrow\alpha \prec P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$
and *QeqP''*: *insertAssertion* (*extractFrame Q*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame P''*) Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto\alpha \prec P'$
by(*rule weakTransitionE*)
show *?thesis*
proof(*case-tac* $P = P''$)
assume $P = P''$
have $\Psi \rhd$ *Cases CsP* $\Longrightarrow\hat{}_\tau$ *Cases CsP* **by** *simp*
moreover from *ImpR AssertionStatEq-def* **have** *insertAssertion*(*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame*(*Cases CsP*)) Ψ
by(*rule-tac FrameStatImpTrans*) (*auto intro*: *Identity*)+

moreover from *P''Trans* ‹(φ, P) *mem CsP*› ‹$\Psi \vdash \varphi$› ‹*guarded P*› ‹$P = P''$› **have** $\Psi \rhd$ *Cases CsP* $\longmapsto\alpha \prec P'$
by(*blast intro*: *Case*)
ultimately show *?thesis*
by(*rule weakTransitionI*)
next
assume $P \neq P''$
with *PChain* **have** $\Psi \rhd P \Longrightarrow_\tau P''$ **by**(*simp add*: *rtrancl-eq-or-trancl*)
hence $\Psi \rhd$ *Cases CsP* $\Longrightarrow_\tau P''$ **using** ‹(φ, P) *mem CsP*› ‹$\Psi \vdash \varphi$› ‹*guarded P*›
by(*rule tauStepChainCase*)
hence $\Psi \rhd$ *Cases CsP* $\Longrightarrow\hat{}_\tau P''$ **by** *simp*
moreover from *RImpQ QeqP''* **have** *insertAssertion*(*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame P''*) Ψ
by(*rule FrameStatImpTrans*)
ultimately show *?thesis* **using** *P''Trans* **by**(*rule weakTransitionI*)
qed
qed

lemma *weakOpen*:
fixes Ψ :: $'b$

and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and *yvec* :: *name list*
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$
and $x \in supp\ N$
and $x \sharp \Psi$
and $x \sharp M$
and $x \sharp xvec$
and $x \sharp yvec$

shows $\Psi : (\!|\nu x|\!)Q \rhd (\!|\nu x|\!)P \Longrightarrow M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P''$ **and** *QeqP''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$x \sharp \Psi$› **have** $\Psi \rhd (\!|\nu x|\!)P \Longrightarrow\hat{}_\tau\ (\!|\nu x|\!)P''$ **by**(*rule tauChainResPres*)
moreover from *QeqP''* ‹$x \sharp \Psi$› **have** *insertAssertion* (*extractFrame*$((\!|\nu x|\!)Q)$) Ψ $\hookrightarrow_F$ *insertAssertion* (*extractFrame*$((\!|\nu x|\!)P'')$) Ψ **by**(*force intro*: *frameImpResPres*)
moreover from *P''Trans* ‹$x \in supp\ N$› ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp xvec$› ‹$x \sharp yvec$› **have** $\Psi \rhd (\!|\nu x|\!)P'' \longmapsto M(\!|\nu*(xvec@x\#yvec)|\!)\langle N\rangle \prec P'$
by(*rule Open*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakScope*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P'$
and $x \sharp \Psi$
and $x \sharp \alpha$

shows $\Psi : (\!|\nu x|\!)Q \rhd (\!|\nu x|\!)P \Longrightarrow \alpha \prec (\!|\nu x|\!)P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P''$ **and** *QeqP''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto \alpha \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$x \sharp \Psi$› **have** $\Psi \rhd (\!|\nu x|\!) P \Longrightarrow\hat{}_\tau (\!|\nu x|\!) P''$ **by**(*rule tauChainResPres*)
moreover from *QeqP''* ‹$x \sharp \Psi$› **have** *insertAssertion* (*extractFrame*$((\!|\nu x|\!) Q)$) Ψ $\hookrightarrow_F$ *insertAssertion* (*extractFrame*$((\!|\nu x|\!) P'')$) Ψ **by**(*force intro*: *frameImpResPres*)
moreover from *P''Trans* ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› **have** $\Psi \rhd (\!|\nu x|\!) P'' \longmapsto \alpha \prec (\!|\nu x|\!) P'$
by(*rule Scope*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakPar1*:
fixes Ψ :: $'b$
and R :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and A_Q :: *name list*
and Ψ_Q :: $'b$

assumes *PTrans*: $\Psi \otimes \Psi_Q : R \rhd P \Longrightarrow \alpha \prec P'$
and *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$
and *bn* $\alpha \sharp* Q$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* \alpha$
and $A_Q \sharp* R$

shows $\Psi : R \parallel Q \rhd P \parallel Q \Longrightarrow \alpha \prec P' \parallel Q$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_Q \rhd P \Longrightarrow\hat{}_\tau P''$
and *ReqP''*: *insertAssertion* (*extractFrame* R) $(\Psi \otimes \Psi_Q)$ $\hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') $(\Psi \otimes \Psi_Q)$
and *P''Trans*: $\Psi \otimes \Psi_Q \rhd P'' \longmapsto \alpha \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$A_Q \sharp* P$› **have** $A_Q \sharp* P''$ **by**(*rule tauChainFreshChain*)
from *PChain FrQ* ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› **have** $\Psi \rhd P \parallel Q \Longrightarrow\hat{}_\tau P'' \parallel Q$ **by**(*rule tauChainPar1*)
moreover have *insertAssertion* (*extractFrame*$(R \parallel Q)$) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*$(P'' \parallel Q)$) Ψ
proof −
obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R \rangle$ **and** $A_R \sharp* A_Q$ **and** $A_R \sharp* \Psi_Q$ **and** $A_R \sharp* \Psi$
by(*rule-tac* C=(A_Q, Ψ_Q, Ψ) **in** *freshFrame*) *auto*
obtain A_P'' Ψ_P'' **where** *FrP''*: *extractFrame* $P'' = \langle A_P'', \Psi_P'' \rangle$ **and** $A_P'' \sharp* A_Q$ **and** $A_P'' \sharp* \Psi_Q$ **and** $A_P'' \sharp* \Psi$
by(*rule-tac* C=(A_Q, Ψ_Q, Ψ) **in** *freshFrame*) *auto*

from *FrR FrP''* ‹$A_Q \sharp* R$› ‹$A_Q \sharp* P''$› ‹$A_R \sharp* A_Q$› ‹$A_P'' \sharp* A_Q$› **have** $A_Q \sharp* \Psi_R$ **and** $A_Q \sharp* \Psi_P''$

by(*force dest*: *extractFrameFreshChain*)+
have $\langle A_R, \Psi \otimes \Psi_R \otimes \Psi_Q\rangle \simeq_F \langle A_R, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle$
by(*metis frameNilStatEq frameResChainPres Associativity Commutativity Composition AssertionStatEqTrans*)
moreover from *ReqP'' FrR FrP''* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_Q$› ‹$A_P'' \sharp* \Psi$› ‹$A_P'' \sharp* \Psi_Q$›
have $\langle A_R, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P'', (\Psi \otimes \Psi_Q) \otimes \Psi_P''\rangle$ **using** *freshCompChain* **by** *auto*
moreover have $\langle A_P'', (\Psi \otimes \Psi_Q) \otimes \Psi_P''\rangle \simeq_F \langle A_P'', \Psi \otimes \Psi_P'' \otimes \Psi_Q\rangle$
by(*metis frameNilStatEq frameResChainPres Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately have $\langle A_R, \Psi \otimes \Psi_R \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P'', \Psi \otimes \Psi_P'' \otimes \Psi_Q\rangle$
by(*force dest*: *FrameStatImpTrans simp add*: *FrameStatEq-def*)

hence $\langle (A_R@A_Q), \Psi \otimes \Psi_R \otimes \Psi_Q\rangle \hookrightarrow_F \langle (A_P''@A_Q), \Psi \otimes \Psi_P'' \otimes \Psi_Q\rangle$
apply(*simp add*: *frameChainAppend*)
apply(*drule-tac xvec=*A_Q **in** *frameImpResChainPres*)
by(*metis frameImpChainComm FrameStatImpTrans*)
with *FrR FrQ FrP''* ‹$A_R \sharp* A_Q$› ‹$A_R \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_R$› ‹$A_P'' \sharp* A_Q$› ‹$A_P'' \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P''$› ‹$A_R \sharp* \Psi$› ‹$A_P'' \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› *ReqP''*
show *?thesis* **by** *simp*
qed
moreover from *P''Trans FrQ* ‹$bn\ \alpha \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P''$› ‹$A_Q \sharp* \alpha$› **have** $\Psi \rhd P'' \parallel Q \longmapsto\alpha \prec (P' \parallel Q)$
by(*rule Par1*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *weakPar2*:
fixes Ψ :: $'b$
and R :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and Q' :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and A_P :: *name list*
and Ψ_P :: $'b$

assumes *QTrans*: $\Psi \otimes \Psi_P : R \rhd Q \Longrightarrow\alpha \prec Q'$
and *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$
and $bn\ \alpha \sharp* P$
and $A_P \sharp* \Psi$
and $A_P \sharp* Q$
and $A_P \sharp* \alpha$
and $A_P \sharp* R$

shows $\Psi : P \parallel R \rhd P \parallel Q \Longrightarrow\alpha \prec P \parallel Q'$

proof −
from *QTrans* **obtain** Q'' **where** *QChain*: $\Psi \otimes \Psi_P \rhd Q \Longrightarrow\hat{}_\tau Q''$
and *ReqQ''*: *insertAssertion* (*extractFrame* R) $(\Psi \otimes \Psi_P)$ $\hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'') $(\Psi \otimes \Psi_P)$
and Q''*Trans*: $\Psi \otimes \Psi_P \rhd Q'' \longmapsto\alpha \prec Q'$
by(*rule weakTransitionE*)

from *QChain* ‹$A_P \sharp* Q$› **have** $A_P \sharp* Q''$ **by**(*rule tauChainFreshChain*)

from *QChain FrP* ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q$› **have** $\Psi \rhd P \parallel Q \Longrightarrow\hat{}_\tau P \parallel Q''$ **by**(*rule tauChainPar2*)
moreover have *insertAssertion* (*extractFrame*$(P \parallel R)$) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*$(P \parallel Q'')$) Ψ
proof −
obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **and** $A_R \sharp* A_P$ **and** $A_R \sharp* \Psi_P$ **and** $A_R \sharp* \Psi$
by(*rule-tac* C=(A_P, Ψ_P, Ψ) **in** *freshFrame*) *auto*
obtain $A_Q{}''$ $\Psi_Q{}''$ **where** *FrQ''*: *extractFrame* $Q'' = \langle A_Q{}'', \Psi_Q{}''\rangle$ **and** $A_Q{}'' \sharp* A_P$ **and** $A_Q{}'' \sharp* \Psi_P$ **and** $A_Q{}'' \sharp* \Psi$
by(*rule-tac* C=(A_P, Ψ_P, Ψ) **in** *freshFrame*) *auto*

from *FrR FrQ''* ‹$A_P \sharp* R$› ‹$A_P \sharp* Q''$› ‹$A_R \sharp* A_P$› ‹$A_Q{}'' \sharp* A_P$› **have** $A_P \sharp* \Psi_R$ **and** $A_P \sharp* \Psi_Q{}''$
by(*force dest*: *extractFrameFreshChain*)+
have $\langle A_R, \Psi \otimes \Psi_P \otimes \Psi_R\rangle \simeq_F \langle A_R, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
by(*metis frameNilStatEq frameResChainPres Associativity Commutativity Composition AssertionStatEqTrans*)

moreover from *ReqQ'' FrR FrQ''* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P$› ‹$A_Q{}'' \sharp* \Psi$› ‹$A_Q{}'' \sharp* \Psi_P$›
have $\langle A_R, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_Q{}'', (\Psi \otimes \Psi_P) \otimes \Psi_Q{}''\rangle$ **using** *freshCompChain* **by** *simp*
moreover have $\langle A_Q{}'', (\Psi \otimes \Psi_P) \otimes \Psi_Q{}''\rangle \simeq_F \langle A_Q{}'', \Psi \otimes \Psi_P \otimes \Psi_Q{}''\rangle$
by(*metis frameNilStatEq frameResChainPres Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately have $\langle A_R, \Psi \otimes \Psi_P \otimes \Psi_R\rangle \hookrightarrow_F \langle A_Q{}'', \Psi \otimes \Psi_P \otimes \Psi_Q{}''\rangle$
by(*force dest*: *FrameStatImpTrans simp add*: *FrameStatEq-def*)
hence $\langle (A_P@A_R), \Psi \otimes \Psi_P \otimes \Psi_R\rangle \hookrightarrow_F \langle (A_P@A_Q{}''), \Psi \otimes \Psi_P \otimes \Psi_Q{}''\rangle$
apply(*simp add*: *frameChainAppend*)
apply(*drule-tac xvec*=A_P **in** *frameImpResChainPres*)
by(*metis frameImpChainComm FrameStatImpTrans*)
with *FrR FrP FrQ''* ‹$A_R \sharp* A_P$› ‹$A_R \sharp* \Psi_P$› ‹$A_P \sharp* \Psi_R$› ‹$A_Q{}'' \sharp* A_P$› ‹$A_Q{}'' \sharp* \Psi_P$› ‹$A_P \sharp* \Psi_Q{}''$› ‹$A_R \sharp* \Psi$› ‹$A_Q{}'' \sharp* \Psi$› ‹$A_P \sharp* \Psi$› *ReqQ''*
show *?thesis* **by** *simp*
qed
moreover from Q''*Trans FrP* ‹*bn* $\alpha \sharp* P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q''$› ‹$A_P \sharp* \alpha$› **have** $\Psi \rhd P \parallel Q'' \longmapsto\alpha \prec (P \parallel Q')$
by(*rule-tac Par2*) *auto*
ultimately show *?thesis* **by**(*rule weakTransitionI*)

qed

lemma *weakComm1*:
fixes Ψ :: $'b$
and R :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and A_Q :: *name list*
and Ψ_Q :: $'b$

assumes *PTrans*: $\Psi \otimes \Psi_Q : R \rhd P \Longrightarrow M(\!|N|\!) \prec P'$
and *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$
and *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
and *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
and *MeqK*: $\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$
and $A_R \sharp* R$
and $A_R \sharp* M$
and $A_R \sharp* A_Q$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* Q$
and $A_Q \sharp* R$
and $A_Q \sharp* K$
and $xvec \sharp* P$

shows $\Psi \rhd P \parallel Q \Longrightarrow_\tau ((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
proof −
from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* R$› ‹$A_Q \sharp* K$› ‹$A_R \sharp* A_Q$›
obtain $A_Q{}'$ **where** *FrQ'*: *extractFrame* $Q = \langle A_Q{}', \Psi_Q\rangle$ **and** *distinct* $A_Q{}'$ **and** $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* P$
and $A_Q{}' \sharp* Q$ **and** $A_Q{}' \sharp* R$ **and** $A_Q{}' \sharp* K$ **and** $A_R \sharp* A_Q{}'$
by(*rule-tac* C=(Ψ, P, Q, R, K, A_R) **in** *distinctFrame*) *auto*

from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_Q \rhd P \Longrightarrow\hat{}_\tau P''$
and *RimpP''*: *insertAssertion* (*extractFrame* R) $(\Psi \otimes \Psi_Q)$ $\hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') $(\Psi \otimes \Psi_Q)$
and *P''Trans*: $\Psi \otimes \Psi_Q \rhd P'' \longmapsto M(\!|N|\!) \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$A_Q{}' \sharp* P$› **have** $A_Q{}' \sharp* P''$ **by**(*rule tauChainFreshChain*)
obtain $A_P{}''$ $\Psi_P{}''$ **where** *FrP''*: *extractFrame* $P'' = \langle A_P{}'', \Psi_P{}''\rangle$ **and** $A_P{}'' \sharp*$ $(\Psi, A_Q{}', \Psi_Q, A_R, \Psi_R, M, N, K, R, Q, P'', xvec)$ **and** *distinct* $A_P{}''$
by(*rule freshFrame*)

hence $A_P{}'' \sharp* \Psi$ **and** $A_P{}'' \sharp* A_Q{}'$ **and** $A_P{}'' \sharp* \Psi_Q$ **and** $A_P{}'' \sharp* M$ **and** $A_P{}''$ $\sharp* R$ **and** $A_P{}'' \sharp* Q$
and $A_P{}'' \sharp* N$ **and** $A_P{}'' \sharp* K$ **and** $A_P{}'' \sharp* A_R$ **and** $A_P{}'' \sharp* P''$ **and** $A_P{}'' \sharp*$ *xvec* **and** $A_P{}'' \sharp* \Psi_R$
by *simp+*
from *FrR* ‹$A_R \sharp* A_Q{}'$› ‹$A_Q{}' \sharp* R$› **have** $A_Q{}' \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from *FrQ'* ‹$A_R \sharp* A_Q{}'$› ‹$A_R \sharp* Q$› **have** $A_R \sharp* \Psi_Q$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from *PChain* ‹*xvec* $\sharp* P$› **have** *xvec* $\sharp* P''$ **by**(*force intro*: *tauChainFreshChain*)+

have $\langle A_R, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \simeq_F \langle A_R, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover with *RimpP'' FrP'' FrR* ‹$A_P{}'' \sharp* \Psi$› ‹$A_R \sharp* \Psi$› ‹$A_P{}'' \sharp* \Psi_Q$› ‹A_R $\sharp* \Psi_Q$›
have $\langle A_R, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_Q) \otimes \Psi_P{}''\rangle$ **using** *freshCompChain*
by(*simp add*: *freshChainSimps*)
moreover have $\langle A_P{}'', (\Psi \otimes \Psi_Q) \otimes \Psi_P{}''\rangle \simeq_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'') \otimes \Psi_Q\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
ultimately have *RImpP''*: $\langle A_R, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'') \otimes$ $\Psi_Q\rangle$
by(*rule FrameStatEqImpCompose*)

from *PChain FrQ'* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P$› **have** $\Psi \rhd P \parallel Q \Longrightarrow\hat{}_\tau P'' \parallel Q$ **by**(*rule tauChainPar1*)
moreover from *QTrans FrR P''Trans MeqK RImpP'' FrP'' FrQ'* ‹*distinct* $A_P{}''$› ‹*distinct* $A_Q{}'$› ‹$A_P{}'' \sharp* A_Q{}'$› ‹$A_R \sharp* A_Q{}'$›
‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P''$› ‹$A_Q{}' \sharp* Q$› ‹$A_Q{}' \sharp* R$› ‹$A_Q{}' \sharp* K$› ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* R$› ‹$A_P{}'' \sharp* Q$›
‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* M$› ‹$A_Q \sharp* R$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* M$›
obtain K' **where** $\Psi \otimes \Psi_P{}'' \rhd Q \longmapsto K'(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **and** $\Psi \otimes \Psi_P{}'' \otimes \Psi_Q$ $\vdash M \leftrightarrow K'$ **and** $A_Q{}' \sharp* K'$
by(*rule-tac comm1Aux*) (*assumption* | *simp*)+
with *P''Trans FrP''* **have** $\Psi \rhd P'' \parallel Q \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$ **using** *FrQ'* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P''$› ‹$A_Q{}' \sharp* Q$›
‹*xvec* $\sharp* P''$› ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* Q$› ‹$A_P{}'' \sharp* M$› ‹$A_P{}'' \sharp* A_Q{}'$›
by(*rule-tac Comm1*)
ultimately show *?thesis*
by(*drule-tac tauActTauStepChain*) *auto*
qed

lemma *weakComm2*:
fixes Ψ :: *'b*
and R :: (*'a*, *'b*, *'c*) *psi*
and P :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a action*

and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and A_Q :: *name list*
and Ψ_Q :: $'b$

assumes *PTrans*: $\Psi \otimes \Psi_Q : R \rhd P \Longrightarrow M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$
and *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto K(\!|N|\!) \prec Q'$
and *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$
and *MeqK*: $\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$
and $A_R \sharp* R$
and $A_R \sharp* M$
and $A_R \sharp* A_Q$
and $A_Q \sharp* \Psi$
and $A_Q \sharp* P$
and $A_Q \sharp* Q$
and $A_Q \sharp* R$
and $A_Q \sharp* K$
and $xvec \sharp* Q$
and $xvec \sharp* M$
and $xvec \sharp* A_Q$
and $xvec \sharp* A_R$

shows $\Psi \rhd P \parallel Q \Longrightarrow_\tau ((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
proof −
from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* R$› ‹$A_Q \sharp* K$› ‹$A_R \sharp* A_Q$› ‹$xvec \sharp* A_Q$›
obtain $A_Q{}'$ **where** *FrQ'*: *extractFrame* $Q = \langle A_Q{}', \Psi_Q\rangle$ **and** *distinct* $A_Q{}'$ **and** $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* P$
and $A_Q{}' \sharp* Q$ **and** $A_Q{}' \sharp* R$ **and** $A_Q{}' \sharp* K$ **and** $A_R \sharp* A_Q{}'$ **and** $A_Q{}' \sharp* xvec$
by(*rule-tac* C=(Ψ, P, Q, R, K, A_R, *xvec*) **in** *distinctFrame*) *auto*

from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_Q \rhd P \Longrightarrow\hat{}_\tau P''$
and *RimpP''*: *insertAssertion* (*extractFrame* R) ($\Psi \otimes \Psi_Q$) $\hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') ($\Psi \otimes \Psi_Q$)
and *P''Trans*: $\Psi \otimes \Psi_Q \rhd P'' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$A_Q{}' \sharp* P$› **have** $A_Q{}' \sharp* P''$ **by**(*rule tauChainFreshChain*)
obtain $A_P{}''$ $\Psi_P{}''$ **where** *FrP''*: *extractFrame* $P'' = \langle A_P{}'', \Psi_P{}''\rangle$ **and** $A_P{}'' \sharp*$ (Ψ, $A_Q{}'$, Ψ_Q, A_R, Ψ_R, M, N, K, R, Q, P'', *xvec*) **and** *distinct* $A_P{}''$
by(*rule freshFrame*)
hence $A_P{}'' \sharp* \Psi$ **and** $A_P{}'' \sharp* A_Q{}'$ **and** $A_P{}'' \sharp* \Psi_Q$ **and** $A_P{}'' \sharp* M$ **and** $A_P{}'' \sharp* R$ **and** $A_P{}'' \sharp* Q$
and $A_P{}'' \sharp* N$ **and** $A_P{}'' \sharp* K$ **and** $A_P{}'' \sharp* A_R$ **and** $A_P{}'' \sharp* P''$ **and** $A_P{}'' \sharp*$

xvec **and** $A_P{}'' \sharp* \Psi_R$
by *simp+*
from *FrR* ‹$A_R \sharp* A_Q{}'$› ‹$A_Q{}' \sharp* R$› **have** $A_Q{}' \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from *FrQ'* ‹$A_R \sharp* A_Q{}'$› ‹$A_R \sharp* Q$› **have** $A_R \sharp* \Psi_Q$ **by**(*drule-tac extractFrameFreshChain*) *auto*

have $\langle A_R, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \simeq_F \langle A_R, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover with *RimpP'' FrP'' FrR* ‹$A_P{}'' \sharp* \Psi$› ‹$A_R \sharp* \Psi$› ‹$A_P{}'' \sharp* \Psi_Q$› ‹$A_R \sharp* \Psi_Q$›
have $\langle A_R, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_Q) \otimes \Psi_P{}''\rangle$ **using** *freshCompChain*
by(*simp add*: *freshChainSimps*)
moreover have $\langle A_P{}'', (\Psi \otimes \Psi_Q) \otimes \Psi_P{}''\rangle \simeq_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'') \otimes \Psi_Q\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
ultimately have *RImpP''*: $\langle A_R, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'') \otimes \Psi_Q\rangle$
by(*rule FrameStatEqImpCompose*)

from *PChain FrQ'* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P$› **have** $\Psi \rhd P \parallel Q \Longrightarrow\hat{}_\tau P'' \parallel Q$ **by**(*rule tauChainPar1*)
moreover from *QTrans FrR P''Trans MeqK RImpP'' FrP'' FrQ'* ‹*distinct* $A_P{}''$› ‹*distinct* $A_Q{}'$› ‹$A_P{}'' \sharp* A_Q{}'$› ‹$A_R \sharp* A_Q{}'$›
‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P''$› ‹$A_Q{}' \sharp* Q$› ‹$A_Q{}' \sharp* R$› ‹$A_Q{}' \sharp* K$› ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* R$› ‹$A_P{}'' \sharp* Q$›
‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* M$› ‹$A_Q \sharp* R$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* M$› ‹*xvec* $\sharp* A_R$› ‹*xvec* $\sharp* M$› ‹$A_Q{}' \sharp*$ *xvec*›
obtain K' **where** $\Psi \otimes \Psi_P{}'' \rhd Q \longmapsto K'(\!|N|\!) \prec Q'$ **and** $\Psi \otimes \Psi_P{}'' \otimes \Psi_Q \vdash M \leftrightarrow K'$ **and** $A_Q{}' \sharp* K'$
by(*rule-tac comm2Aux*) (*assumption* | *simp*)+
with *P''Trans FrP''* **have** $\Psi \rhd P'' \parallel Q \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$ **using** *FrQ'* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P''$› ‹$A_Q{}' \sharp* Q$›
‹*xvec* $\sharp* Q$› ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* Q$› ‹$A_P{}'' \sharp* M$› ‹$A_P{}'' \sharp* A_Q{}'$›
by(*rule-tac Comm2*)
ultimately show *?thesis*
by(*drule-tac tauActTauStepChain*) *auto*
qed

lemma *frameImpIntComposition*:
fixes Ψ :: *'b*
and Ψ' :: *'b*
and A_F :: *name list*
and Ψ_F :: *'b*

assumes $\Psi \simeq \Psi'$

shows $\langle A_F,\ \Psi \otimes \Psi_F\rangle \hookrightarrow_F \langle A_F,\ \Psi' \otimes \Psi_F\rangle$
proof –
from *assms* **have** $\langle A_F,\ \Psi \otimes \Psi_F\rangle \simeq_F \langle A_F,\ \Psi' \otimes \Psi_F\rangle$ **by**(*rule frameIntComposition*)
thus *?thesis* **by**(*simp add*: *FrameStatEq-def*)
qed

lemma *insertAssertionStatImp*:
fixes F :: *'b frame*
and Ψ :: *'b*
and G :: *'b frame*
and Ψ' :: *'b*

assumes *FeqG*: *insertAssertion* $F\ \Psi \hookrightarrow_F$ *insertAssertion* $G\ \Psi$
and $\Psi \simeq \Psi'$

shows *insertAssertion* $F\ \Psi' \hookrightarrow_F$ *insertAssertion* $G\ \Psi'$
proof –
obtain $A_F\ \Psi_F$ **where** *FrF*: $F = \langle A_F,\ \Psi_F\rangle$ **and** $A_F \sharp\!* \Psi$ **and** $A_F \sharp\!* \Psi'$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*
obtain $A_G\ \Psi_G$ **where** *FrG*: $G = \langle A_G,\ \Psi_G\rangle$ **and** $A_G \sharp\!* \Psi$ **and** $A_G \sharp\!* \Psi'$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*

from ‹$\Psi \simeq \Psi'$› **have** $\langle A_F,\ \Psi' \otimes \Psi_F\rangle \simeq_F \langle A_F,\ \Psi \otimes \Psi_F\rangle$ **by** (*metis frameIntComposition FrameStatEqSym*)
moreover from ‹$\Psi \simeq \Psi'$› **have** $\langle A_G,\ \Psi \otimes \Psi_G\rangle \simeq_F \langle A_G,\ \Psi' \otimes \Psi_G\rangle$ **by**(*rule frameIntComposition*)
ultimately have $\langle A_F,\ \Psi' \otimes \Psi_F\rangle \hookrightarrow_F \langle A_G,\ \Psi' \otimes \Psi_G\rangle$ **using** *FeqG FrF FrG* ‹$A_F \sharp\!* \Psi$› ‹$A_G \sharp\!* \Psi$› ‹$\Psi \simeq \Psi'$›
by(*force simp add*: *FrameStatEq-def dest*: *FrameStatImpTrans*)
with *FrF FrG* ‹$A_F \sharp\!* \Psi'$› ‹$A_G \sharp\!* \Psi'$› **show** *?thesis* **by** *simp*
qed

lemma *insertAssertionStatEq*:
fixes F :: *'b frame*
and Ψ :: *'b*
and G :: *'b frame*
and Ψ' :: *'b*

assumes *FeqG*: *insertAssertion* $F\ \Psi \simeq_F$ *insertAssertion* $G\ \Psi$
and $\Psi \simeq \Psi'$

shows *insertAssertion* $F\ \Psi' \simeq_F$ *insertAssertion* $G\ \Psi'$
proof –
obtain $A_F\ \Psi_F$ **where** *FrF*: $F = \langle A_F,\ \Psi_F\rangle$ **and** $A_F \sharp\!* \Psi$ **and** $A_F \sharp\!* \Psi'$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*
obtain $A_G\ \Psi_G$ **where** *FrG*: $G = \langle A_G,\ \Psi_G\rangle$ **and** $A_G \sharp\!* \Psi$ **and** $A_G \sharp\!* \Psi'$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*

from *FeqG FrF FrG* ‹A_F ♯* Ψ› ‹A_G ♯* Ψ› ‹$\Psi \simeq \Psi'$›
have $\langle A_F, \Psi' \otimes \Psi_F\rangle \simeq_F \langle A_G, \Psi' \otimes \Psi_G\rangle$
by *simp* (*metis frameIntComposition FrameStatEqTrans FrameStatEqSym*)
with *FrF FrG* ‹A_F ♯* Ψ'› ‹A_G ♯* Ψ'› **show** *?thesis* **by** *simp*
qed

lemma *weakTransitionStatEq*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$
and $\Psi \simeq \Psi'$

shows $\Psi' : Q \rhd P \Longrightarrow\alpha \prec P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$
and *QeqP''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto\alpha \prec P'$
by(*rule weakTransitionE*)

from *PChain* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \rhd P \Longrightarrow\hat{}_\tau P''$ **by**(*rule tauChainStatEq*)
moreover from *QeqP''* ‹$\Psi \simeq \Psi'$› **have** *insertAssertion* (*extractFrame* Q) Ψ' $\hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ'
by(*rule insertAssertionStatImp*)
moreover from *P''Trans* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \rhd P'' \longmapsto\alpha \prec P'$
by(*rule statEqTransition*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed

lemma *transitionWeakTransition*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \longmapsto\alpha \prec P'$
and *insertAssertion*(*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P) Ψ

shows $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$
using *assms*
by(*fastforce intro*: *weakTransitionI*)

lemma *weakPar1Guarded*:
fixes Ψ :: $'b$
and R :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes *PTrans*: $\Psi : R \rhd P \Longrightarrow \alpha \prec P'$
and *bn* $\alpha \sharp* Q$
and *guarded* Q

shows $\Psi : (R \parallel Q) \rhd P \parallel Q \Longrightarrow \alpha \prec P' \parallel Q$
proof −
obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* P$ **and** $A_Q \sharp* \alpha$ **and** $A_Q \sharp* R$
by(*rule-tac* C=(Ψ, P, α, R) **in** *freshFrame*) *auto*
from ‹*guarded* Q› *FrQ* **have** $\Psi_Q \simeq \mathbf{1}$ **by**(*blast dest*: *guardedStatEq*)
with *PTrans* **have** $\Psi \otimes \Psi_Q : R \rhd P \Longrightarrow \alpha \prec P'$ **by**(*metis weakTransitionStatEq Identity AssertionStatEqSym compositionSym*)
thus *?thesis* **using** *FrQ* ‹*bn* $\alpha \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$› ‹$A_Q \sharp* R$›

by(*rule weakPar1*)
qed

lemma *weakBang*:
fixes Ψ :: $'b$
and R :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes *PTrans*: $\Psi : R \rhd P \parallel !P \Longrightarrow \alpha \prec P'$
and *guarded* P

shows $\Psi : R \rhd !P \Longrightarrow \alpha \prec P'$
proof −
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \parallel !P \Longrightarrow\hat{}_\tau P''$
and *RImpP''*: *insertAssertion*(*extractFrame* R) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P'') Ψ
and P''*Trans*: $\Psi \rhd P'' \longmapsto \alpha \prec P'$
by(*rule weakTransitionE*)
moreover obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P \rangle$ **and** $A_P \sharp* \Psi$ **by**(*rule freshFrame*)
moreover from ‹*guarded* P› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **by**(*blast dest*: *guardedStatEq*)
ultimately show *?thesis*
proof(*auto simp add*: *rtrancl-eq-or-trancl*)
have $\Psi \rhd !P \Longrightarrow\hat{}_\tau !P$ **by** *simp*

moreover assume *RimpP*: *insertAssertion*(*extractFrame R*) $\Psi \hookrightarrow_F \langle A_P, \Psi \otimes \Psi_P \otimes \mathbf{1}\rangle$
have *insertAssertion*(*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame*(!*P*)) Ψ
proof −
from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle A_P, \Psi \otimes \Psi_P \otimes \mathbf{1}\rangle \simeq_F \langle A_P, \Psi \otimes \mathbf{1}\rangle$
by(*metis frameIntCompositionSym frameIntAssociativity frameIntCommutativity frameIntIdentity FrameStatEqTrans FrameStatEqSym*)
moreover from ‹$A_P \sharp* \Psi$› **have** $\langle A_P, \Psi \otimes \mathbf{1}\rangle \simeq_F \langle \varepsilon, \Psi \otimes \mathbf{1}\rangle$
by(*force intro*: *frameResFreshChain*)
ultimately show *?thesis* **using** *RimpP* **by**(*auto simp add*: *FrameStatEq-def dest*: *FrameStatImpTrans*)
qed
moreover assume $\Psi \rhd P \parallel !P \longmapsto\alpha \prec P'$
hence $\Psi \rhd !P \longmapsto\alpha \prec P'$ **using** ‹*guarded P*› **by**(*rule Bang*)
ultimately show *?thesis* **by**(*rule weakTransitionI*)
next
fix P'''
assume $\Psi \rhd P \parallel !P \Longrightarrow_\tau P''$
hence $\Psi \rhd !P \Longrightarrow_\tau P''$ **using** ‹*guarded P*› **by**(*rule tauStepChainBang*)
hence $\Psi \rhd !P \Longrightarrow\hat{}_\tau P''$ **by** *simp*
moreover assume *insertAssertion*(*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame P''*) Ψ
and $\Psi \rhd P'' \longmapsto\alpha \prec P'$
ultimately show *?thesis* **by**(*rule weakTransitionI*)
qed
qed

lemma *weakTransitionFrameImp*:
fixes Ψ :: *'b*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *M* :: *'a*
and *xvec* :: *name list*
and *N* :: *'a*
and *P'* :: (*'a*, *'b*, *'c*) *psi*
and *R* :: (*'a*, *'b*, *'c*) *psi*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$
and *insertAssertion*(*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame Q*) Ψ

shows $\Psi : R \rhd P \Longrightarrow\alpha \prec P'$
using *assms*
by(*auto simp add*: *weakTransition-def intro*: *FrameStatImpTrans*)

lemma *guardedFrameStatEq*:
fixes *P* :: (*'a*, *'b*, *'c*) *psi*

assumes *guarded P*

shows *extractFrame P* $\simeq_F$ $\langle\varepsilon, \mathbf{1}\rangle$
proof −
obtain A_P Ψ_P **where** *FrP*: *extractFrame P* = $\langle A_P, \Psi_P\rangle$ **by**(*rule freshFrame*)
from ‹*guarded P*› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **by**(*blast dest*: *guardedStatEq*)
hence $\langle A_P, \Psi_P\rangle \simeq_F \langle A_P, \mathbf{1}\rangle$ **by**(*rule-tac frameResChainPres*) *auto*
moreover have $\langle A_P, \mathbf{1}\rangle \simeq_F \langle\varepsilon, \mathbf{1}\rangle$ **by**(*rule-tac frameResFreshChain*) *auto*
ultimately show *?thesis* **using** *FrP* **by**(*force intro*: *FrameStatEqTrans*)
qed

lemma *weakGuardedTransition*:
fixes Ψ :: *'b*
and Q :: (*'a*, *'b*, *'c*) *psi*
and P :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a action*
and P' :: (*'a*, *'b*, *'c*) *psi*
and R :: (*'a*, *'b*, *'c*) *psi*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P'$
and *guarded Q*

shows $\Psi : \mathbf{0} \rhd P \Longrightarrow \alpha \prec P'$
proof −
obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame Q* = $\langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **by**(*rule freshFrame*)
moreover from ‹*guarded Q*› *FrQ* **have** $\Psi_Q \simeq \mathbf{1}$ **by**(*blast dest*: *guardedStatEq*)
hence $\langle A_Q, \Psi \otimes \Psi_Q\rangle \simeq_F \langle A_Q, \Psi \otimes \mathbf{1}\rangle$ **by**(*metis frameIntCompositionSym*)
moreover from ‹$A_Q \sharp* \Psi$› **have** $\langle A_Q, \Psi \otimes \mathbf{1}\rangle \simeq_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$ **by**(*rule-tac frameResFreshChain*) *auto*
ultimately have *insertAssertion*(*extractFrame Q*) $\Psi \simeq_F$ *insertAssertion* (*extractFrame* (**0**)) Ψ
using *FrQ* ‹$A_Q \sharp* \Psi$› **by** *simp* (*blast intro*: *FrameStatEqTrans*)
with *PTrans* **show** *?thesis* **by**(*rule-tac weakTransitionFrameImp*) (*auto simp add*: *FrameStatEq-def*)
qed

lemma *expandTauChainFrame*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and P' :: (*'a*, *'b*, *'c*) *psi*
and A_P :: *name list*
and Ψ_P :: *'b*
and C :: *'d::fs-name*

assumes *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
and *FrP*: *extractFrame P* = $\langle A_P, \Psi_P\rangle$
and *distinct* A_P
and $A_P \sharp* P$

and $A_P \sharp* C$

obtains Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$ **and** $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* C$ **and** *distinct* $A_P{}'$
using *PChain FrP* ‹$A_P \sharp* P$›
proof(*induct arbitrary*: *thesis rule*: *tauChainInduct*)
case(*TauBase P*)
have $\Psi_P \otimes SBottom' \simeq \Psi_P$ **by**(*rule Identity*)
with ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› **show** *?case* **using** ‹$A_P \sharp* P$› ‹$A_P \sharp* C$› ‹*distinct* A_P› **by**(*rule TauBase*)
next
case(*TauStep P P′ P″*)
from ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹$A_P \sharp* P$›
obtain Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *FrP′*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$
and $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* C$ **and** *distinct* $A_P{}'$
by(*rule-tac TauStep*)
from ‹$\Psi \rhd P' \longmapsto\tau \prec P''$› *FrP′* ‹*distinct* $A_P{}'$› ‹$A_P{}' \sharp* P'$› ‹$A_P{}' \sharp* C$›
obtain Ψ'' $A_P{}''$ $\Psi_P{}''$ **where** *FrP″*: *extractFrame* $P'' = \langle A_P{}'', \Psi_P{}''\rangle$ **and** $\Psi_P{}' \otimes \Psi'' \simeq \Psi_P{}''$
and $A_P{}'' \sharp* P''$ **and** $A_P{}'' \sharp* C$ **and** *distinct* $A_P{}''$
by(*rule expandTauFrame*)
from ‹$\Psi_P \otimes \Psi' \simeq \Psi_P{}'$› **have** $(\Psi_P \otimes \Psi') \otimes \Psi'' \simeq \Psi_P{}' \otimes \Psi''$ **by**(*rule Composition*)
with ‹$\Psi_P{}' \otimes \Psi'' \simeq \Psi_P{}''$› **have** $\Psi_P \otimes \Psi' \otimes \Psi'' \simeq \Psi_P{}''$
by(*metis AssertionStatEqTrans Associativity Commutativity*)
with *FrP″* **show** *?case* **using** ‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* C$› ‹*distinct* $A_P{}''$›
by(*rule TauStep*)
qed

lemma *frameIntImpComposition*:
fixes Ψ :: *′b*
and Ψ' :: *′b*
and A_F :: *name list*
and Ψ_F :: *′b*

assumes $\Psi \simeq \Psi'$

shows $\langle A_F, \Psi \otimes \Psi_F\rangle \hookrightarrow_F \langle A_F, \Psi' \otimes \Psi_F\rangle$
using *assms frameIntComposition*
by(*simp add*: *FrameStatEq-def*)

lemma *tauChainInduct2*[*consumes 1*, *case-names TauBase TauStep*]:
fixes Ψ :: *′b*
and P :: (*′a*, *′b*, *′c*) *psi*
and P' :: (*′a*, *′b*, *′c*) *psi*

assumes *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
and *cBase*: $\bigwedge P.$ *Prop P P*

and *cStep*: $\bigwedge P\ P'\ P''.\ [\![\Psi \rhd P' \longmapsto\tau \prec P''; \Psi \rhd P \Longrightarrow\hat{}_\tau P'; Prop\ P\ P']\!] \Longrightarrow Prop\ P\ P''$

shows $Prop\ P\ P'$
using *assms*
by(*rule tauChainInduct*)

lemma *tauStepChainInduct2*[*consumes 1*, *case-names TauBase TauStep*]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $P' :: ('a, 'b, 'c)\ psi$

assumes *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$
and *cBase*: $\bigwedge P\ P'.\ \Psi \rhd P \longmapsto\tau \prec P' \Longrightarrow Prop\ P\ P'$
and *cStep*: $\bigwedge P\ P'\ P''.\ [\![\Psi \rhd P' \longmapsto\tau \prec P''; \Psi \rhd P \Longrightarrow_\tau P'; Prop\ P\ P']\!] \Longrightarrow Prop\ P\ P''$

shows $Prop\ P\ P'$
using *assms*
by(*rule tauStepChainInduct*)

lemma *weakTransferTauChainFrame*:
fixes $\Psi_F :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $P' :: ('a, 'b, 'c)\ psi$
and $A_P :: name\ list$
and $\Psi_P :: 'b$
and $A_F :: name\ list$
and $A_G :: name\ list$
and $\Psi_G :: 'b$

assumes *PChain*: $\Psi_F \rhd P \Longrightarrow\hat{}_\tau P'$
and *FrP*: $extractFrame\ P = \langle A_P, \Psi_P\rangle$
and $distinct\ A_P$
and *FeqG*: $\bigwedge\Psi.\ insertAssertion\ (\langle A_F, \Psi_F \otimes \Psi_P\rangle)\ \Psi \hookrightarrow_F insertAssertion\ (\langle A_G, \Psi_G \otimes \Psi_P\rangle)\ \Psi$
and $A_F \sharp* \Psi_G$
and $A_G \sharp* \Psi$
and $A_G \sharp* \Psi_F$
and $A_F \sharp* A_G$
and $A_F \sharp* P$
and $A_G \sharp* P$
and $A_P \sharp* A_F$
and $A_P \sharp* A_G$
and $A_P \sharp* \Psi_F$
and $A_P \sharp* \Psi_G$
and $A_P \sharp* P$

shows $\Psi_G \rhd P \Longrightarrow\hat{}_\tau P'$

using *PChain FrP* ‹A_F ♯* P› ‹A_G ♯* P› ‹A_P ♯* P›
proof(*induct rule*: *tauChainInduct2*)
  case *TauBase*
  thus *?case* **by** *simp*
next
  case(*TauStep P P′ P″*)
  have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
  then have *PChain*: $\Psi_G \rhd P \Longrightarrow\hat{}_\tau P'$ **using** ‹A_F ♯* P› ‹A_G ♯* P› ‹A_P ♯* P› **by**(*rule TauStep*)
  then obtain $A_P{}'$ $\Psi_P{}'$ Ψ' **where** *FrP′*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$
                and $A_P{}'$ ♯* A_F **and** $A_P{}'$ ♯* A_G **and** $A_P{}'$ ♯* Ψ_F **and** $A_P{}'$ ♯* Ψ_G
                and *distinct* $A_P{}'$

    using *FrP* ‹*distinct* A_P› ‹A_P ♯* P› ‹A_P ♯* A_F› ‹A_P ♯* A_G› ‹A_P ♯* Ψ_F› ‹A_P ♯* Ψ_G›
    by(*rule-tac* C=(A_F, A_G, Ψ_F, Ψ_G) **in** *expandTauChainFrame*) *auto*

  from *PChain* ‹A_F ♯* P› ‹A_G ♯* P› **have** A_F ♯* P' **and** A_G ♯* P' **by**(*blast dest*: *tauChainFreshChain*)+

  with ‹A_F ♯* P› ‹A_G ♯* P› ‹A_P ♯* A_F› ‹A_P ♯* A_G›‹$A_P{}'$ ♯* A_F› ‹$A_P{}'$ ♯* A_G› *FrP FrP′*
  have A_F ♯* Ψ_P **and** A_G ♯* Ψ_P **and** A_F ♯* $\Psi_P{}'$ **and** A_G ♯* $\Psi_P{}'$
    by(*auto dest*: *extractFrameFreshChain*)

  from *FeqG* **have** *FeqG*: *insertAssertion* $(\langle A_F, \Psi_F \otimes \Psi_P\rangle)$ $\Psi' \hookrightarrow_F$ *insertAssertion* $(\langle A_G, \Psi_G \otimes \Psi_P\rangle)$ Ψ'
    by *blast*
  obtain p::*name prm* **where** $(p \cdot A_F)$ ♯* Ψ_F **and** $(p \cdot A_F)$ ♯* Ψ_P **and** $(p \cdot A_F)$ ♯* $\Psi_P{}'$ **and** $(p \cdot A_F)$ ♯* Ψ'
                and *Sp*: (*set* p) $\subseteq$ *set* $A_F \times$ *set*$(p \cdot A_F)$ **and** *distinctPerm* p
      by(*rule-tac xvec*=A_F **and** c=(Ψ_F, Ψ_P, Ψ', $\Psi_P{}'$) **in** *name-list-avoiding*) *auto*
  obtain q::*name prm* **where** $(q \cdot A_G)$ ♯* Ψ_G **and** $(q \cdot A_G)$ ♯* Ψ_P **and** $(q \cdot A_G)$ ♯* $\Psi_P{}'$ **and** $(q \cdot A_G)$ ♯* Ψ'
                and *Sq*: (*set* q) $\subseteq$ *set* $A_G \times$ *set*$(q \cdot A_G)$ **and** *distinctPerm* q
    by(*rule-tac xvec*=A_G **and** c=(Ψ_G, Ψ_P, Ψ', $\Psi_P{}'$) **in** *name-list-avoiding*) *auto*
  from ‹$\Psi_P \otimes \Psi' \simeq \Psi_P{}'$› **have** $\langle(p \cdot A_F), ((p \cdot \Psi_F) \otimes \Psi_P{}')\rangle \simeq_F \langle(p \cdot A_F), (p \cdot \Psi_F) \otimes (\Psi_P \otimes \Psi')\rangle$
    by(*rule frameIntCompositionSym*[*OF AssertionStatEqSym*])
  hence $\langle(p \cdot A_F), (p \cdot \Psi_F) \otimes \Psi_P{}'\rangle \simeq_F \langle(p \cdot A_F), \Psi' \otimes ((p \cdot \Psi_F) \otimes \Psi_P)\rangle$
    by(*metis frameIntAssociativity FrameStatEqTrans frameIntCommutativity FrameStatEqSym*)
  moreover from *FeqG* ‹A_F ♯* Ψ_P› ‹$(p \cdot A_F)$ ♯* Ψ_P› ‹$(p \cdot A_F)$ ♯* Ψ_F› ‹$(p \cdot A_F)$ ♯* Ψ'› *Sp*
  have $\langle(p \cdot A_F), \Psi' \otimes ((p \cdot \Psi_F) \otimes \Psi_P)\rangle \hookrightarrow_F$ *insertAssertion* $(\langle A_G, \Psi_G \otimes \Psi_P\rangle)$ Ψ'
    apply(*erule-tac rev-mp*) **by**(*subst frameChainAlpha*) (*auto simp add*: *eqvts*)

hence $\langle(p \cdot A_F), \Psi' \otimes ((p \cdot \Psi_F) \otimes \Psi_P)\rangle \hookrightarrow_F \ (\langle(q \cdot A_G), \Psi' \otimes (q \cdot \Psi_G) \otimes \Psi_P\rangle)$
using ‹$A_G \sharp* \Psi_P$› ‹$(q \cdot A_G) \sharp* \Psi_P$› ‹$(q \cdot A_G) \sharp* \Psi_G$› ‹$(q \cdot A_G) \sharp* \Psi'$› *Sq*
apply(*erule-tac rev-mp*) **by**(*subst frameChainAlpha*) (*auto simp add*: *eqvts*)
moreover have $\langle(q \cdot A_G), \Psi' \otimes ((q \cdot \Psi_G) \otimes \Psi_P)\rangle \simeq_F \langle(q \cdot A_G), (q \cdot \Psi_G) \otimes (\Psi_P \otimes \Psi')\rangle$
by(*metis frameIntAssociativity FrameStatEqTrans frameIntCommutativity FrameStatEqSym*)
hence $\langle(q \cdot A_G), \Psi' \otimes ((q \cdot \Psi_G) \otimes \Psi_P)\rangle \simeq_F \langle(q \cdot A_G), (q \cdot \Psi_G) \otimes \Psi_P'\rangle$ **using** ‹$\Psi_P \otimes \Psi' \simeq \Psi_P'$›
by(*blast intro*: *FrameStatEqTrans frameIntCompositionSym*)
ultimately have $\langle(p \cdot A_F), (p \cdot \Psi_F) \otimes \Psi_P'\rangle \hookrightarrow_F \langle(q \cdot A_G), (q \cdot \Psi_G) \otimes \Psi_P'\rangle$
by(*rule FrameStatEqImpCompose*)
with ‹$A_F \sharp* \Psi_P'$› ‹$(p \cdot A_F) \sharp* \Psi_P'$› ‹$(p \cdot A_F) \sharp* \Psi_F$› *Sp* **have** $\langle A_F, \Psi_F \otimes \Psi_P'\rangle \hookrightarrow_F \langle(q \cdot A_G), (q \cdot \Psi_G) \otimes \Psi_P'\rangle$
by(*subst frameChainAlpha*) (*auto simp add*: *eqvts*)
with ‹$A_G \sharp* \Psi_P'$› ‹$(q \cdot A_G) \sharp* \Psi_P'$› ‹$(q \cdot A_G) \sharp* \Psi_G$› *Sq* **have** $\langle A_F, \Psi_F \otimes \Psi_P'\rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P'\rangle$
by(*subst frameChainAlpha*) (*auto simp add*: *eqvts*)

with ‹$\Psi_F \rhd P' \longmapsto\tau \prec P''$› *FrP'* ‹*distinct* A_P'›
‹$A_F \sharp* P'$› ‹$A_G \sharp* P'$› ‹$A_F \sharp* \Psi_G$› ‹$A_G \sharp* \Psi_F$› ‹$A_P' \sharp* A_F$› ‹$A_P' \sharp* A_G$› ‹$A_P' \sharp* \Psi_F$› ‹$A_P' \sharp* \Psi_G$›
have $\Psi_G \rhd P' \longmapsto\tau \prec P''$ **by**(*rule-tac transferTauFrame*)
with *PChain* **show** *?case* **by**(*simp add*: *r-into-rtrancl rtrancl-into-rtrancl*)
qed

coinductive *quiet* :: $('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool*
where $[\![\forall \Psi.\ (\mathit{insertAssertion}\ (\mathit{extractFrame}\ P)\ \Psi \simeq_F \langle\varepsilon, \Psi\rangle \wedge (\forall Rs.\ \Psi \rhd P \longmapsto Rs \longrightarrow (\exists P'.\ Rs = \tau \prec P' \wedge \mathit{quiet}\ P')))]\!] \Longrightarrow \mathit{quiet}\ P$

lemma *quietFrame*:
fixes $\Psi :: 'b$
and $P \ :: ('a, 'b, 'c)$ *psi*

assumes *quiet P*

shows $\mathit{insertAssertion}\ (\mathit{extractFrame}\ P)\ \Psi \simeq_F \langle\varepsilon, \Psi\rangle$
using *assms*
by(*erule-tac quiet.cases*) *force*

lemma *quietTransition*:
fixes $\Psi :: 'b$
and $P \ :: ('a, 'b, 'c)$ *psi*
and $Rs :: ('a, 'b, 'c)$ *residual*

assumes *quiet P*
and $\Psi \rhd P \longmapsto Rs$

obtains P' **where** $Rs = \tau \prec P'$ **and** *quiet* P'
using *assms*
by(*erule-tac quiet.cases*) *force*

lemma *quietEqvt*:
fixes P :: ($'a$, $'b$, $'c$) *psi*
and p :: *name prm*

assumes *quiet* P

shows *quiet*($p \cdot P$)
proof −
let $?X = \lambda P.\ \exists\, p{::}name\ prm.\ quiet(p \cdot P)$
from *assms* **have** $?X\ (p \cdot P)$ **by**(*rule-tac x=rev p* **in** *exI*) *auto*
thus *?thesis*
apply *coinduct*
apply(*clarify*)
apply(*rule-tac x=x* **in** *exI*)
apply *auto*
apply(*drule-tac* $\Psi{=}p \cdot \Psi$ **in** *quietFrame*)
apply(*drule-tac p=rev p* **in** *FrameStatEqClosed*)
apply(*simp add*: *eqvts*)
apply(*drule-tac pi=p* **in** *semantics.eqvt*)
apply(*erule-tac quietTransition*)
apply *assumption*
apply(*rule-tac x=rev* $p \cdot P'$ **in** *exI*)
apply *auto*
apply(*drule-tac pi=rev p* **in** *pt-bij3*)
apply(*simp add*: *eqvts*)
apply(*rule-tac x=p* **in** *exI*)
by *simp*
qed

lemma *quietOutput*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P' :: ($'a$, $'b$, $'c$) *psi*

assumes $\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and *quiet* P

shows *False*
using *assms*
apply(*erule-tac quiet.cases*)

by(*force simp add*: *residualInject*)

lemma *quietInput*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$
  and P' :: $('a, 'b, 'c)$ *psi*

  assumes $\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$
  and *quiet* P

  shows *False*
using *assms*
by(*erule-tac quiet.cases*) (*force simp add*: *residualInject*)

lemma *quietTau*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*

  assumes $\Psi \rhd P \longmapsto \tau \prec P'$
  and *insertAssertion* (*extractFrame* P) $\Psi \simeq_F \langle \varepsilon, \Psi \rangle$
  and *quiet* P

  shows *quiet* P'
using *assms*
by(*erule-tac quiet.cases*) (*force simp add*: *residualInject*)

lemma *tauChainCases*[*consumes 1*, *case-names TauBase TauStep*]:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*

  assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
  and $P = P' \Longrightarrow Prop$
  and $\Psi \rhd P \Longrightarrow_\tau P' \Longrightarrow Prop$

  shows *Prop*
using *assms*
by(*blast elim*: *rtranclE dest*: *rtrancl-into-trancl1*)

end

end

theory *Weak-Simulation*
  imports *Simulation Tau-Chain*
begin

context *env* **begin**

definition
weakSimulation :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$
$('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set* $\Rightarrow$
$('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- ⊳ - ⇝<-> -› [80, 80, 80, 80] 80)
where
$\Psi \rhd P \rightsquigarrow$<*Rel*> $Q \equiv (\forall \Psi' \ \alpha \ Q'.\ \Psi \rhd Q \longmapsto \alpha \prec Q' \longrightarrow bn\ \alpha \ \sharp* \ \Psi \longrightarrow bn\ \alpha \ \sharp* \ P \longrightarrow \alpha \neq \tau \longrightarrow$
$(\exists P''.\ \Psi : Q \rhd P \Longrightarrow \alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi \otimes \Psi', P', Q') \in Rel))) \wedge$
$(\forall Q'.\ \Psi \rhd Q \longmapsto \tau \prec Q' \longrightarrow (\exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi, P', Q') \in Rel))$

abbreviation
weakSimulationNilJudge (‹- ⇝<-> -› [80, 80, 80] 80) **where** $P \rightsquigarrow$<*Rel*> $Q \equiv$ *SBottom'* $\rhd P \rightsquigarrow$<*Rel*> Q

lemma *weakSimI*[*consumes 1*, *case-names cAct cTau*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Eqvt*: *eqvt Rel*
and *rAct*: $\bigwedge \Psi' \ \alpha \ Q'.\ [\![\Psi \rhd Q \longmapsto \alpha \prec Q';\ bn\ \alpha \ \sharp* \ \Psi;\ bn\ \alpha \ \sharp* \ P;\ bn\ \alpha \ \sharp* \ Q;$
$bn\ \alpha \ \sharp* \ subject\ \alpha;\ bn\ \alpha \ \sharp* \ C;\ distinct(bn\ \alpha);\ \alpha \neq \tau]\!] \Longrightarrow$

$\exists P''.\ \Psi : Q \rhd P \Longrightarrow \alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi \otimes \Psi', P', Q') \in Rel)$
and *rTau*: $\bigwedge Q'.\ \Psi \rhd Q \longmapsto \tau \prec Q' \Longrightarrow \exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi, P', Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow$<*Rel*> Q
proof(*auto simp add*: *weakSimulation-def*)
fix $\Psi' \ \alpha \ Q'$
assume $\Psi \rhd Q \longmapsto \alpha \prec Q'$ **and** $bn\ \alpha \ \sharp* \ \Psi$ **and** $bn\ \alpha \ \sharp* \ P$ **and** $\alpha \neq \tau$
thus $\exists P''.\ \Psi : Q \rhd P \Longrightarrow \alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi \otimes \Psi', P', Q') \in Rel)$
proof(*nominal-induct* α *rule*: *action.strong-induct*)
case(*In M N*)
thus *?case* **by**(*rule-tac rAct*) *auto*
next
case(*Out M xvec N*)
from ‹$bn(M(\!|\nu*xvec|\!)\langle N\rangle) \ \sharp* \ \Psi$› ‹$bn(M(\!|\nu*xvec|\!)\langle N\rangle) \ \sharp* \ P$› **have** $xvec \ \sharp* \ \Psi$ **and** $xvec \ \sharp* \ P$ **by** *simp+*
from ‹$\Psi \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› **have** *distinct xvec* **by**(*force dest*:

boundOutputDistinct)
obtain p **where** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* M$ **and** $(p \cdot xvec) \sharp* xvec$
and$(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* Q'$ **and** $(p \cdot xvec) \sharp* \Psi'$
and $(p \cdot xvec) \sharp* C$ **and** $(p \cdot xvec) \sharp* xvec$ **and** $(p \cdot xvec) \sharp* N$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm* p
by(*rule-tac xvec=xvec* **and** $c=(\Psi, M, Q, N, P, Q', xvec, C, \Psi')$ **in** *name-list-avoiding*)
(*auto simp add*: *eqvts fresh-star-prod*)
from ‹*distinct xvec*› **have** *distinct*$(p \cdot xvec)$ **by** *simp*
from ‹$\Psi \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* Q'$› S
have $\Psi \rhd Q \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot Q')$ **by**(*simp add*: *boundOutputChainAlpha''* *residualInject*)

then obtain P'' P' **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec P''$
and *P''Chain*: $\Psi \otimes (p \cdot \Psi') \rhd P'' \Longrightarrow\hat{}_\tau P'$
and *P'RelQ'*: $(\Psi \otimes (p \cdot \Psi'), P', p \cdot Q') \in Rel$
using ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› ‹$(p \cdot xvec) \sharp* M$› ‹$(p \cdot xvec) \sharp* C$› ‹*distinct*$(p \cdot xvec)$›
by(*drule-tac* $\Psi'=p \cdot \Psi'$ **in** *rAct*) *auto*

from *PTrans S* ‹*distinctPerm p*› ‹$xvec \sharp* P$› ‹$(p \cdot xvec) \sharp* xvec$› ‹$(p \cdot xvec) \sharp* M$› ‹*distinct xvec*›
have $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*xvec|\!)\langle N\rangle \prec (p \cdot P'')$
by(*rule-tac weakOutputAlpha*) *auto*
moreover from *P''Chain* **have** $(p \cdot (\Psi \otimes (p \cdot \Psi'))) \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*rule eqvts*)
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi'$› S ‹*distinctPerm p*›
have $\Psi \otimes \Psi' \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from *P'RelQ' Eqvt S* ‹*distinctPerm p*› **have** $(p \cdot (\Psi \otimes (p \cdot \Psi')), p \cdot P', p \cdot p \cdot Q') \in Rel$
apply(*simp add*: *eqvt-def eqvts*)
apply(*erule-tac* $x=(\Psi \otimes (p \cdot \Psi'), P', p \cdot Q')$ **in** *ballE*)
apply(*erule-tac* $x=p$ **in** *allE*)
by(*auto simp add*: *eqvts*)
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S ‹*distinctPerm p*›
have $(\Psi \otimes \Psi', p \cdot P', Q') \in Rel$ **by**(*simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
case *Tau*
thus *?case* **by** *simp*
qed
next
fix Q'
assume $\Psi \rhd Q \longmapsto \tau \prec Q'$
thus $\exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau P' \wedge (\Psi, P', Q') \in Rel$
by(*rule rTau*)
qed

lemma *weakSimI2*[*case-names cAct cTau*]:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *C* :: *'d*::*fs-name*

assumes *rOutput*: $\bigwedge\Psi'\ \alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ \alpha \neq \tau]\!] \Longrightarrow$
$\exists P''.\ \Psi : Q \rhd P \Longrightarrow\alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi \otimes \Psi',\ P',\ Q') \in Rel)$
and *rTau*: $\bigwedge Q'.\ \Psi \rhd Q \longmapsto\tau \prec Q' \Longrightarrow \exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi,\ P',\ Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow$<*Rel*> *Q*
using *assms* **by**(*simp add*: *weakSimulation-def*)

lemma *weakSimIChainFresh*[*consumes 4*, *case-names cOutput cInput*]:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *yvec* :: *name list*
and *C* :: *'d*::*fs-name*

assumes *Eqvt*: *eqvt Rel*
and $yvec \sharp* \Psi$
and $yvec \sharp* P$
and $yvec \sharp* Q$
and *rAct*: $\bigwedge\Psi'\ \alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* Q;\ \alpha \neq \tau;$
$bn\ \alpha \sharp* subject\ \alpha;\ bn\ \alpha \sharp* C;\ yvec \sharp* \alpha;\ yvec \sharp* Q';\ yvec \sharp* \Psi']\!] \Longrightarrow$
$\exists P''.\ \Psi : Q \rhd P \Longrightarrow\alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi \otimes \Psi',\ P',\ Q') \in Rel)$
and *rTau*: $\bigwedge Q'.\ [\![\Psi \rhd Q \longmapsto\tau \prec Q';\ yvec \sharp* Q']\!] \Longrightarrow \exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi,\ P',\ Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow$<*Rel*> *Q*
using ‹*eqvt Rel*›
proof(*induct rule*: *weakSimI*[*of* - - - - (*yvec*, *C*)])
case(*cAct* $\Psi'\ \alpha\ Q'$)
obtain *p*::*name prm* **where** $(p \cdot yvec) \sharp* \Psi$ **and** $(p \cdot yvec) \sharp* P$ **and** $(p \cdot yvec) \sharp* Q$
and $(p \cdot yvec) \sharp* \alpha$ **and** $(p \cdot yvec) \sharp* \Psi'$
and $(p \cdot yvec) \sharp* Q'$ **and** *S*: (*set p*) ⊆ *set yvec* × *set*($p \cdot yvec$)
and *distinctPerm p*
by(*rule-tac c*=(Ψ, *P*, *Q*, α, Q', Ψ') **and** *xvec*=*yvec* **in** *name-list-avoiding*) *auto*
from ‹$bn\ \alpha \sharp* (yvec,\ C)$› **have** $bn\ \alpha \sharp* yvec$ **and** $bn\ \alpha \sharp* C$ **by**(*simp add*:

freshChainSym)+
show *?case*
proof(*cases rule*: *actionCases*[**where** $\alpha = \alpha$])
case(*cInput M N*)
from ‹$(p \cdot yvec) \sharp* \alpha$› ‹$\alpha = M(\!|N|\!)$› **have** $(p \cdot yvec) \sharp* M$ **and** $(p \cdot yvec) \sharp* N$ **by** *simp*+
from ‹$\Psi \rhd Q \longmapsto \alpha \prec Q'$› ‹$\alpha = M(\!|N|\!)$› ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› ‹$yvec \sharp* Q$› ‹$(p \cdot yvec) \sharp* Q$› S
have $\Psi \rhd Q \longmapsto (p \cdot M)(\!|(p \cdot N)|\!) \prec (p \cdot Q')$
by(*drule-tac pi=p* **in** *semantics.eqvt*) (*simp add*: *eqvts*)
moreover from ‹$(p \cdot yvec) \sharp* M$› **have** $(p \cdot (p \cdot yvec)) \sharp* (p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $yvec \sharp* (p \cdot M)$ **by** *simp*
moreover from ‹$(p \cdot yvec) \sharp* N$› **have** $(p \cdot p \cdot yvec) \sharp* (p \cdot N)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $yvec \sharp* (p \cdot N)$ **by** *simp*
moreover from ‹$(p \cdot yvec) \sharp* Q'$› **have** $(p \cdot p \cdot yvec) \sharp* (p \cdot Q')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $yvec \sharp* (p \cdot Q')$ **by** *simp*
moreover from ‹$(p \cdot yvec) \sharp* \Psi'$› **have** $(p \cdot p \cdot yvec) \sharp* (p \cdot \Psi')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $yvec \sharp* (p \cdot \Psi')$ **by** *simp*
ultimately obtain P' P'' **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow (p \cdot M)(\!|(p \cdot N)|\!) \prec P''$ **and** $P''Chain$: $\Psi \otimes (p \cdot \Psi') \rhd P'' \Longrightarrow\hat{}_\tau P'$
and $P'RelQ'$: $(\Psi \otimes (p \cdot \Psi'), P', (p \cdot Q')) \in Rel$
by(*auto dest*: *rAct*)
from *PTrans* **have** $(p \cdot \Psi) : (p \cdot Q) \rhd (p \cdot P) \Longrightarrow p \cdot ((p \cdot M)(\!|(p \cdot N)|\!)) \prec (p \cdot P'')$
by(*rule weakTransitionClosed*)
with S ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› ‹$yvec \sharp* Q$› ‹$(p \cdot yvec) \sharp* Q$› ‹$yvec \sharp* P$› ‹$(p \cdot yvec) \sharp* P$› ‹*distinctPerm p*›
have $\Psi : Q \rhd P \Longrightarrow M(\!|N|\!) \prec (p \cdot P'')$ **by**(*simp add*: *eqvts*)
moreover from $P''Chain$ **have** $(p \cdot (\Psi \otimes (p \cdot \Psi'))) \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*rule eqvts*)
with ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› S ‹*distinctPerm p*›
have $\Psi \otimes \Psi' \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from $P'RelQ'$ *Eqvt* **have** $(p \cdot (\Psi \otimes (p \cdot \Psi')), p \cdot P', p \cdot p \cdot Q') \in Rel$
apply(*simp add*: *eqvt-def eqvts*)
apply(*erule-tac* $x=(\Psi \otimes (p \cdot \Psi'), P', p \cdot Q')$ **in** *ballE*)
apply(*erule-tac x=p* **in** *allE*)
by(*auto simp add*: *eqvts*)
with ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› S ‹*distinctPerm p*› **have** $(\Psi \otimes \Psi', p \cdot P', Q') \in Rel$ **by**(*simp add*: *eqvts*)
ultimately show *?thesis* **using** ‹$\alpha=M(\!|N|\!)$› **by** *blast*
next
case(*cOutput M xvec N*)
from ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$\alpha=M(\!|\nu*xvec|\!)\langle N\rangle$› ‹$bn\ \alpha \sharp* yvec$›

‹$(p \cdot yvec) \sharp* \alpha$› ‹$bn\ \alpha \sharp* C$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
have $xvec \sharp* \Psi$ **and** $xvec \sharp* P$ **and** $xvec \sharp* Q$ **and** $xvec \sharp* M$ **and** $yvec \sharp* xvec$
and $(p \cdot yvec) \sharp* M$ **and** $(p \cdot yvec) \sharp* xvec$ **and** $xvec \sharp* C$ **and** $xvec \sharp* M$ **and** $(p \cdot yvec) \sharp* N$
and *distinct xvec* **by** *simp+*
from ‹$\Psi \rhd Q \longmapsto \alpha \prec Q'$› ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› ‹$yvec \sharp* Q$› ‹$(p \cdot yvec) \sharp* Q$› S ‹$\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$›
have $\Psi \rhd Q \longmapsto (p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$
by(*rule-tac outputPermSubject*) (*assumption* | *simp add*: *fresh-star-def*)+
moreover note ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$›
moreover note ‹$xvec \sharp* Q$›
moreover from ‹$xvec \sharp* M$› **have** $(p \cdot xvec) \sharp* (p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$yvec \sharp* xvec$› ‹$(p \cdot yvec) \sharp* xvec$› S **have** $xvec \sharp* (p \cdot M)$
by *simp*
moreover note ‹$xvec \sharp* C$›
moreover from ‹$(p \cdot yvec) \sharp* M$› **have** $(p \cdot (p \cdot yvec)) \sharp* (p \cdot M)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $yvec \sharp* (p \cdot M)$ **by** *simp*
moreover note ‹$yvec \sharp* xvec$›
moreover from ‹$\Psi \rhd Q \longmapsto \alpha \prec Q'$› ‹$yvec \sharp* Q$› ‹$yvec \sharp* xvec$› ‹$\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$› ‹$xvec \sharp* M$› ‹*distinct xvec*›
have $yvec \sharp* N$ **and** $yvec \sharp* Q'$ **by**(*force dest*: *outputFreshChainDerivative*)+
moreover from ‹$(p \cdot yvec) \sharp* \Psi'$› **have** $(p \cdot p \cdot yvec) \sharp* (p \cdot \Psi')$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $yvec \sharp* (p \cdot \Psi')$ **by** *simp*
ultimately obtain P'' P' **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow (p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle \prec P''$
and *PChain*: $\Psi \otimes (p \cdot \Psi') \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
and $P'RelQ'$: $(\Psi \otimes (p \cdot \Psi'), P', Q') \in Rel$
by(*drule-tac rAct*) *auto*
from *PTrans* **have** $(p \cdot \Psi) : (p \cdot Q) \rhd (p \cdot P) \Longrightarrow (p \cdot ((p \cdot M)(\!|\nu*xvec|\!)\langle N\rangle)) \prec (p \cdot P'')$
by(*rule eqvts*)
with S ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› ‹$yvec \sharp* P$› ‹$(p \cdot yvec) \sharp* P$› ‹$yvec \sharp* Q$› ‹$(p \cdot yvec) \sharp* Q$› ‹$yvec \sharp* xvec$› ‹$(p \cdot yvec) \sharp* xvec$›
‹$yvec \sharp* N$› ‹$(p \cdot yvec) \sharp* N$› ‹*distinctPerm p*› **have** $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*xvec|\!)\langle N\rangle \prec (p \cdot P'')$
by(*simp add*: *eqvts*)
moreover from *PChain* **have** $(p \cdot (\Psi \otimes (p \cdot \Psi'))) \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau\ (p \cdot P')$ **by**(*rule eqvts*)
with S ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$› ‹*distinctPerm p*› **have** $\Psi \otimes \Psi' \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau\ (p \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from $P'RelQ'$ ‹*eqvt Rel*› **have** $p \cdot (\Psi \otimes (p \cdot \Psi'), P', Q') \in Rel$
by(*simp add*: *eqvt-def*) *auto*
with ‹$yvec \sharp* \Psi$› ‹$(p \cdot yvec) \sharp* \Psi$›‹$yvec \sharp* Q'$› ‹$(p \cdot yvec) \sharp* Q'$› S ‹*distinctPerm p*›
have $(\Psi \otimes \Psi', p \cdot P', Q') \in Rel$ **by**(*simp add*: *eqvts*)
ultimately show *?thesis* **using** ‹$\alpha = M(\!|\nu*xvec|\!)\langle N\rangle$› **by** *blast*

next
case *cTau*
from ‹$\alpha = \tau$› ‹$\alpha \neq \tau$› **have** *False* **by** *simp*
thus *?thesis* **by** *simp*
qed
next
case(*cTau* Q')
from ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$› ‹*yvec* $\sharp*$ Q› **have** *yvec* $\sharp*$ Q'
by(*force dest*: *tauFreshChainDerivative*)
with ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$› **show** *?case*
by(*rule rTau*)
qed

lemma *weakSimIFresh*[*consumes 4*, *case-names cAct cTau*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and x :: *name*
and C :: $'d$::*fs-name*

assumes *Eqvt*: *eqvt Rel*
and $x \sharp \Psi$
and $x \sharp P$
and $x \sharp Q$
and $\bigwedge\alpha\ Q'\ \Psi'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ bn\ \alpha \sharp* Q;\ \alpha \neq \tau;$
$bn\ \alpha \sharp*$ *subject* α; $bn\ \alpha \sharp* C$; $x \sharp \alpha$; $x \sharp Q'$; $x \sharp \Psi']\!] \Longrightarrow$
$\exists P''.\ \Psi : Q \rhd P \Longrightarrow\alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
$\wedge\ (\Psi \otimes \Psi',\ P',\ Q') \in Rel)$
and $\bigwedge Q'.\ [\![\Psi \rhd Q \longmapsto\tau \prec Q';\ x \sharp Q']\!] \Longrightarrow \exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi,\ P',$
$Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow$<*Rel*> Q
using *assms*
by(*rule-tac yvec*=[x] **and** C=C **in** *weakSimIChainFresh*) *auto*

lemma *weakSimE*:
fixes F :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \rightsquigarrow$<*Rel*> Q

shows $\bigwedge\Psi'\ \alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P;\ \alpha \neq \tau]\!] \Longrightarrow$
$\exists P''.\ \Psi : Q \rhd P \Longrightarrow\alpha \prec P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau$
$P' \wedge (\Psi \otimes \Psi',\ P',\ Q') \in Rel)$
and $\bigwedge Q'.\ \Psi \rhd Q \longmapsto\tau \prec Q' \Longrightarrow \exists P'.\ \Psi \rhd P \Longrightarrow\hat{}_\tau\ P' \wedge (\Psi,\ P',\ Q') \in Rel$
using *assms*

by(*auto simp add*: *weakSimulation-def*)

lemma *weakSimClosedAux*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*

assumes *EqvtRel*: *eqvt Rel*
and *PSimQ*: $\Psi \rhd P \rightsquigarrow$<*Rel*> Q

shows $(p \cdot \Psi) \rhd (p \cdot P) \rightsquigarrow$<*Rel*> $(p \cdot Q)$
using *EqvtRel*
proof(*induct rule*: *weakSimI*[*of* - - - - (Ψ, P, p)])
case(*cAct* Ψ' α Q')
from ‹$p \cdot \Psi \rhd p \cdot Q \longmapsto \alpha \prec Q'$›
have $(rev\ p \cdot p \cdot \Psi) \rhd (rev\ p \cdot p \cdot Q) \longmapsto (rev\ p \cdot (\alpha \prec Q'))$
by(*blast dest*: *semantics.eqvt*)
hence $\Psi \rhd Q \longmapsto (rev\ p \cdot \alpha) \prec (rev\ p \cdot Q')$
by(*simp add*: *eqvts*)
moreover with ‹$bn\ \alpha \sharp* (\Psi, P, p)$› **have** $bn\ \alpha \sharp* \Psi$ **and** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* p$ **by** *simp+*
moreover from ‹$\alpha \neq \tau$› **have** $(rev\ p \cdot \alpha) \neq \tau$
by(*cases rule*: *actionCases*[**where** α=α]) *auto*
ultimately obtain P'' P' **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow (rev\ p \cdot \alpha) \prec P''$
and P''*Chain*: $\Psi \otimes (rev\ p \cdot \Psi') \rhd P'' \Longrightarrow\hat{}_\tau P'$ **and** P'*RelQ'*: $(\Psi \otimes (rev\ p \cdot \Psi'), P', rev\ p \cdot Q') \in Rel$
using ‹$\alpha \neq \tau$› *PSimQ*
by(*drule-tac* Ψ'=$rev\ p \cdot \Psi'$ **in** *weakSimE*(*1*)) (*auto simp add*: *freshChainPermSimp bnEqvt*[*symmetric*])

from *PTrans* **have** $(p \cdot \Psi) : (p \cdot Q) \rhd (p \cdot P) \Longrightarrow (p \cdot (rev\ p \cdot \alpha)) \prec (p \cdot P'')$
by(*rule eqvts*)
hence $(p \cdot \Psi) : (p \cdot Q) \rhd (p \cdot P) \Longrightarrow \alpha \prec (p \cdot P'')$ **by**(*simp add*: *eqvts freshChainPermSimp*)
moreover from P''*Chain* **have** $(p \cdot (\Psi \otimes (rev\ p \cdot \Psi'))) \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*rule eqvts*)
hence $(p \cdot \Psi) \otimes \Psi' \rhd (p \cdot P'') \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from P'*RelQ'* *EqvtRel* **have** $(p \cdot ((\Psi \otimes (rev\ p \cdot \Psi')), P', rev\ p \cdot Q')) \in Rel$
by(*simp only*: *eqvt-def*)
hence $((p \cdot \Psi) \otimes \Psi', p \cdot P', Q') \in Rel$ **by**(*simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
case(*cTau* Q')
from ‹$p \cdot \Psi \rhd p \cdot Q \longmapsto \tau \prec Q'$›
have $(rev\ p \cdot p \cdot \Psi) \rhd (rev\ p \cdot p \cdot Q) \longmapsto (rev\ p \cdot (\tau \prec Q'))$
by(*blast dest*: *semantics.eqvt*)

hence $\Psi \rhd Q \longmapsto\tau \prec (rev\ p \cdot Q')$ **by**(*simp add*: *eqvts*)
with *PSimQ* **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ rev\ p \cdot Q') \in Rel$
by(*blast dest*: *weakSimE*)
from *PChain* **have** $(p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow\hat{}_\tau (p \cdot P')$ **by**(*rule tauChainEqvt*)
moreover from $P'RelQ'$ *EqvtRel* **have** $(p \cdot (\Psi,\ P',\ rev\ p \cdot Q')) \in Rel$
by(*simp only*: *eqvt-def*)
hence $(p \cdot \Psi,\ p \cdot P',\ Q') \in Rel$ **by**(*simp add*: *eqvts*)
ultimately show *?case*
by *blast*
qed

lemma *weakSimClosed*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and p :: $name\ prm$

assumes *EqvtRel*: $eqvt\ Rel$

shows $\Psi \rhd P \rightsquigarrow\!<Rel>\ Q \Longrightarrow (p \cdot \Psi) \rhd (p \cdot P) \rightsquigarrow\!<Rel>\ (p \cdot Q)$
and $P \rightsquigarrow\!<Rel>\ Q \Longrightarrow (p \cdot P) \rightsquigarrow\!<Rel>\ (p \cdot Q)$
using *EqvtRel*
by(*force dest*: *weakSimClosedAux simp add*: *permBottom*)+

lemma *weakSimReflexive*:
fixes Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$

assumes $\{(\Psi,\ P,\ P) \mid \Psi\ P.\ True\} \subseteq Rel$

shows $\Psi \rhd P \rightsquigarrow\!<Rel>\ P$
using *assms*
by(*auto simp add*: *weakSimulation-def weakTransition-def dest*: *rtrancl-into-rtrancl*) *force*+

lemma *weakSimTauChain*:
fixes Ψ :: $'b$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and Q' :: $('a,\ 'b,\ 'c)\ psi$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$
and $(\Psi,\ P,\ Q) \in Rel$
and Sim: $\bigwedge\Psi'\ R\ S.\ (\Psi',\ R,\ S) \in Rel \Longrightarrow \Psi' \rhd R \rightsquigarrow\!<Rel>\ S$

obtains P' **where** $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **and** $(\Psi, P', Q') \in Rel$
proof −
assume A: $\bigwedge P'. [\![\Psi \rhd P \Longrightarrow\hat{}_\tau P'; (\Psi, P', Q') \in Rel]\!] \Longrightarrow thesis$
from ‹$\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$› ‹$(\Psi, P, Q) \in Rel$› A **show** *?thesis*
proof(*induct arbitrary*: P *thesis rule*: *tauChainInduct*)
case(*TauBase* Q P)
moreover have $\Psi \rhd P \Longrightarrow\hat{}_\tau P$ **by** *simp*
ultimately show *?case* **by** *blast*
next
case(*TauStep* Q Q' Q'' P)
from ‹$(\Psi, P, Q) \in Rel$› **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **and** $(\Psi, P', Q') \in Rel$
by(*rule TauStep*)
from ‹$(\Psi, P', Q') \in Rel$› **have** $\Psi \rhd P' \rightsquigarrow$<*Rel*> Q' **by**(*rule Sim*)
then obtain P'' **where** *P'Chain*: $\Psi \rhd P' \Longrightarrow\hat{}_\tau P''$ **and** $(\Psi, P'', Q'') \in Rel$
using ‹$\Psi \rhd Q' \longmapsto\tau \prec Q''$› **by**(*blast dest*: *weakSimE*)
from *PChain P'Chain* **have** $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **by** *simp*
thus *?case* **using** ‹$(\Psi, P'', Q'') \in Rel$› **by**(*rule TauStep*)
qed
qed

lemma *weakSimE2*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)$ *psi*

assumes *PRelQ*: $(\Psi, P, Q) \in Rel$
and *Sim*: $\bigwedge \Psi'\ R\ S. (\Psi', R, S) \in Rel \Longrightarrow \Psi' \rhd R \rightsquigarrow$<*Rel*> S
and *QTrans*: $\Psi : R \rhd Q \Longrightarrow \alpha \prec Q'$
and $bn\ \alpha \sharp* \Psi$
and $bn\ \alpha \sharp* P$
and $\alpha \neq \tau$

obtains $P''\ P'$ **where** $\Psi : R \rhd P \Longrightarrow \alpha \prec P''$ **and** $\Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **and** $(\Psi \otimes \Psi', P', Q') \in Rel$
proof −
assume A: $\bigwedge P''\ P'. [\![\Psi : R \rhd P \Longrightarrow \alpha \prec P''; \Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'; (\Psi \otimes \Psi', P', Q') \in Rel]\!] \Longrightarrow thesis$
from *QTrans* **obtain** Q''
where *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q''$
and *ReqQ''*: *insertAssertion* (*extractFrame* R) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'') Ψ
and *Q''Trans*: $\Psi \rhd Q'' \longmapsto \alpha \prec Q'$
by(*rule weakTransitionE*)

from *QChain PRelQ Sim*
obtain P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$ **and** *P''RelQ''*: $(\Psi, P'', Q'') \in Rel$
by(*rule weakSimTauChain*)

from *PChain* ‹*bn* α $\sharp*$ *P*› **have** *bn* α $\sharp*$ *P''* **by**(*rule tauChainFreshChain*)
from *P''RelQ''* **have** $\Psi \rhd P'' \rightsquigarrow$<*Rel*> *Q''* **by**(*rule Sim*)
with *Q''Trans* ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ *P''*› ‹$\alpha \neq \tau$›
obtain *P''' P'* **where** *P''Trans*: $\Psi : Q'' \rhd P'' \Longrightarrow\alpha \prec P'''$ **and** *P'''Chain*: $\Psi \otimes \Psi' \rhd P''' \Longrightarrow\hat{}_\tau P'$
and *P'RelQ'*: $(\Psi \otimes \Psi', P', Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *P''Trans* **obtain** *P''''* **where** *P''Chain*: $\Psi \rhd P'' \Longrightarrow\hat{}_\tau P''''$
and *Q''eqP''''*: *insertAssertion* (*extractFrame Q''*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame P''''*) Ψ
and *P''''Trans*: $\Psi \rhd P'''' \longmapsto\alpha \prec P'''$
by(*rule weakTransitionE*)
from *PChain P''Chain* **have** $\Psi \rhd P \Longrightarrow\hat{}_\tau P''''$ **by** *simp*
moreover from *ReqQ'' Q''eqP''''* **have** *insertAssertion* (*extractFrame R*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame P''''*) Ψ
by(*rule FrameStatImpTrans*)
ultimately have $\Psi : R \rhd P \Longrightarrow\alpha \prec P'''$ **using** *P''''Trans* **by**(*rule weakTransitionI*)
with *P'''Chain P'RelQ' A* **show** *?thesis* **by** *blast*
qed

lemma *weakSimTransitive*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *Rel'* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *T* :: (*'a*, *'b*, *'c*) *psi*
and *Rel''* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*

assumes *PRelQ*: $(\Psi, P, Q) \in Rel$
and *QSimR*: $\Psi \rhd Q \rightsquigarrow$<*Rel'*> *R*
and *Eqvt*: *eqvt Rel''*
and *Set*: $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi, P, Q) \in Rel \wedge (\Psi, Q, R) \in Rel'\} \subseteq Rel''$
and *Sim*: $\bigwedge\Psi'\ R\ S.\ (\Psi', R, S) \in Rel \Longrightarrow \Psi' \rhd R \rightsquigarrow$<*Rel*> *S*

shows $\Psi \rhd P \rightsquigarrow$<*Rel''*> *R*
using ‹*eqvt Rel''*›
proof(*induct rule*: *weakSimI*[*of* - - - - *Q*])
case(*cAct* Ψ' α *R'*)
from *QSimR* ‹$\Psi \rhd R \longmapsto\alpha \prec R'$› ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ *Q*› ‹$\alpha \neq \tau$›
obtain *Q'' Q'* **where** *QTrans*: $\Psi : R \rhd Q \Longrightarrow\alpha \prec Q''$ **and** *Q''Chain*: $\Psi \otimes \Psi' \rhd Q'' \Longrightarrow\hat{}_\tau Q'$
and *Q'RelR'*: $(\Psi \otimes \Psi', Q', R') \in Rel'$
by(*blast dest*: *weakSimE*)
with *PRelQ Sim QTrans* ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ *P*› ‹$\alpha \neq \tau$›

obtain P''' P'' **where** *PTrans*: $\Psi : R \triangleright P \Longrightarrow\alpha \prec P'''$
and P'''*Chain*: $\Psi \otimes \Psi' \triangleright P''' \Longrightarrow\hat{}_\tau P''$ **and** P''*RelQ''*: $(\Psi \otimes \Psi', P'', Q'') \in Rel$
by(*drule-tac weakSimE2*) *auto*

note *PTrans*
moreover from *Q''Chain P''RelQ'' Sim* **obtain** P' **where** P''*Chain*: $\Psi \otimes \Psi' \triangleright P'' \Longrightarrow\hat{}_\tau P'$ **and** P'*RelQ'*: $(\Psi \otimes \Psi', P', Q') \in Rel$
by(*rule weakSimTauChain*)
from P'''*Chain* P''*Chain* **have** $\Psi \otimes \Psi' \triangleright P''' \Longrightarrow\hat{}_\tau P'$ **by** *simp*
moreover from *P'RelQ' Q'RelR' Set* **have** $(\Psi \otimes \Psi', P', R') \in Rel''$ **by** *blast*
ultimately show *?case* **by** *blast*
next
case(*cTau R'*)
from *QSimR* ‹$\Psi \triangleright R \longmapsto\tau \prec R'$› **obtain** Q' **where** *QChain*: $\Psi \triangleright Q \Longrightarrow\hat{}_\tau Q'$ **and** Q'*RelR'*: $(\Psi, Q', R') \in Rel'$
by(*blast dest*: *weakSimE*)
from *QChain PRelQ Sim* **obtain** P' **where** *PChain*: $\Psi \triangleright P \Longrightarrow\hat{}_\tau P'$ **and** P'*RelQ'*: $(\Psi, P', Q') \in Rel$
by(*rule weakSimTauChain*)
note *PChain*
moreover from *P'RelQ' Q'RelR' Set* **have** $(\Psi, P', R') \in Rel''$ **by** *blast*
ultimately show *?case* **by** *blast*
qed

lemma *weakSimStatEq*:
fixes Ψ :: *'b*
and P :: *('a, 'b, 'c) psi*
and *Rel* :: *('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set*
and Q :: *('a, 'b, 'c) psi*
and Ψ' :: *'b*

assumes *PSimQ*: $\Psi \triangleright P \rightsquigarrow$<*Rel*> Q
and *eqvt Rel'*
and $\Psi \simeq \Psi'$
and *C1*: $\bigwedge\Psi'\ R\ S\ \Psi''.\ [\![(\Psi', R, S) \in Rel; \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'', R, S) \in Rel'$

shows $\Psi' \triangleright P \rightsquigarrow$<*Rel'*> Q
using ‹*eqvt Rel'*›
proof(*induct rule*: *weakSimI*[*of* - - - - Ψ])
case(*cAct* Ψ'' α Q')
from ‹$\Psi' \triangleright Q \longmapsto\alpha \prec Q'$› ‹$\Psi \simeq \Psi'$›
have $\Psi \triangleright Q \longmapsto\alpha \prec Q'$ **by**(*metis statEqTransition AssertionStatEqSym*)
with *PSimQ* ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ P› ‹$\alpha \neq \tau$›
obtain P'' P' **where** *PTrans*: $\Psi : Q \triangleright P \Longrightarrow\alpha \prec P''$ **and** P''*Chain*: $\Psi \otimes \Psi'' \triangleright P'' \Longrightarrow\hat{}_\tau P'$
and P'*RelQ'*: $(\Psi \otimes \Psi'', P', Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *PTrans* ‹$\Psi \simeq \Psi'$› **have** $\Psi' : Q \rhd P \Longrightarrow\alpha \prec P''$ **by**(*rule weakTransition-StatEq*)
moreover from *P''Chain* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \otimes \Psi'' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **by**(*metis tauChainStatEq Composition*)
moreover from *P'RelQ'* ‹$\Psi \simeq \Psi'$› **have** $(\Psi' \otimes \Psi'', P', Q') \in Rel'$
by(*metis C1 Composition*)
ultimately show *?case*
by *blast*
next
case(*cTau Q'*)
from ‹$\Psi' \rhd Q \longmapsto\tau \prec Q'$› ‹$\Psi \simeq \Psi'$›
have $\Psi \rhd Q \longmapsto\tau \prec Q'$ **by**(*metis statEqTransition AssertionStatEqSym*)
with *PSimQ* **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **and** *P'RelQ'*: $(\Psi, P', Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *PChain* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \rhd P \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauChainStatEq*)
moreover from ‹$(\Psi, P', Q') \in Rel$› ‹$\Psi \simeq \Psi'$› **have** $(\Psi', P', Q') \in Rel'$
by(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *weakSimMonotonic*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and A :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and Q :: (*'a*, *'b*, *'c*) *psi*
and B :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*

assumes $\Psi \rhd P \rightsquigarrow<A> Q$
and $A \subseteq B$

shows $\Psi \rhd P \rightsquigarrow<B> Q$
using *assms*
by(*simp* (*no-asm*) *add*: *weakSimulation-def*) (*blast dest*: *weakSimE*)+

lemma *strongSimWeakSim*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*

assumes *PRelQ*: $(\Psi, P, Q) \in Rel$
and *StatImp*: $\bigwedge\Psi'\ R\ S.\ (\Psi', R, S) \in Rel \Longrightarrow insertAssertion(extractFrame\ S)\ \Psi' \hookrightarrow_F insertAssertion(extractFrame\ R)\ \Psi'$
and *Sim*: $\bigwedge\Psi'\ R\ S.\ (\Psi', R, S) \in Rel \Longrightarrow \Psi' \rhd R \rightsquigarrow[Rel]\ S$
and *Ext*: $\bigwedge\Psi'\ R\ S\ \Psi''.\ (\Psi', R, S) \in Rel \Longrightarrow (\Psi' \otimes \Psi'', R, S) \in Rel$

shows $\Psi \rhd P \rightsquigarrow<Rel> Q$

proof(*induct rule*: *weakSimI2*)
  case(*cAct* Ψ' α Q')
  from *PRelQ* **have** $\Psi \rhd P \rightsquigarrow[Rel]\ Q$ **by**(*rule Sim*)
  with ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$›
  obtain P' **where** *PTrans*: $\Psi \rhd P \longmapsto\alpha \prec P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ Q') \in Rel$
    by(*blast dest*: *simE*)

 from *PRelQ* **have** *insertAssertion*(*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P) Ψ **by**(*rule StatImp*)
  with *PTrans* **have** $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$ **by**(*rule transitionWeakTransition*)
  moreover from $P'RelQ'$ **have** $\forall \Psi'.\ \exists P''.\ \Psi \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau\ P'' \wedge (\Psi \otimes \Psi',\ P'',\ Q') \in Rel$
    by(*force intro*: *Ext*)
  ultimately show *?case* **by** *blast*
next
  case(*cTau* Q')
  from *PRelQ* **have** $\Psi \rhd P \rightsquigarrow[Rel]\ Q$ **by**(*rule Sim*)
  with ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$› **obtain** P' **where** *PTrans*: $\Psi \rhd P \longmapsto\tau \prec P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ Q') \in Rel$
    by(*force dest*: *simE*)
  with *PTrans* **have** $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'$ **by** *auto*
  thus *?case* **using** $P'RelQ'$ **by** *blast*
qed

lemma *strongAppend*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)\ psi$
  and Q :: $('a,\ 'b,\ 'c)\ psi$
  and R :: $('a,\ 'b,\ 'c)\ psi$
  and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
  and Rel' :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
  and Rel'' :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

  assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow<Rel>\ Q$
  and *QSimR*: $\Psi \rhd Q \rightsquigarrow[Rel']\ R$
  and *Eqvt''*: *eqvt Rel''*
 and *RimpQ*: *insertAssertion* (*extractFrame* R) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q) Ψ
  and *Set*: $\{(\Psi,\ P,\ R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi,\ P,\ Q) \in Rel \wedge (\Psi,\ Q,\ R) \in Rel'\} \subseteq Rel''$
  and *C1*: $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in Rel' \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in Rel'$

  shows $\Psi \rhd P \rightsquigarrow<Rel''>\ R$
proof −
  from *Eqvt''* **show** *?thesis*
  proof(*induct rule*: *weakSimI*[*of* - - - - Q])
    case(*cAct* Ψ' α R')
    from ‹$\Psi \rhd Q \rightsquigarrow[Rel']\ R$› ‹$\Psi \rhd R \longmapsto\alpha \prec R'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* Q$›
    obtain Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto\alpha \prec Q'$ **and** $(\Psi,\ Q',\ R') \in Rel'$

by(*blast dest*: *simE*)

from ‹$(\Psi, Q', R') \in Rel'$› **have** $Q'RelR'$: $(\Psi \otimes \Psi', Q', R') \in Rel'$ **by**(*rule C1*)

from ‹$\Psi \rhd P \rightsquigarrow$<*Rel*> Q› *QTrans* ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$\alpha \neq \tau$›
obtain $P''\ P'$ **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P''$ **and** $P''Chain$: $\Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
and $P'RelQ'$: $(\Psi \otimes \Psi', P', Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *PTrans RimpQ* **have** $\Psi : R \rhd P \Longrightarrow \alpha \prec P''$ **by**(*rule weakTransitionFrameImp*)
moreover note $P''Chain$
moreover from $P'RelQ'$ $Q'RelR'$ *Set* **have** $(\Psi \otimes \Psi', P', R') \in Rel''$ **by** *blast*
ultimately show *?case* **by** *blast*
next
case(*cTau* R')
from ‹$\Psi \rhd Q \rightsquigarrow[Rel']\ R$› ‹$\Psi \rhd R \longmapsto \tau \prec R'$›
obtain Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto \tau \prec Q'$ **and** $Q'RelR'$: $(\Psi, Q', R') \in Rel'$
by(*force dest*: *simE*)

from ‹$\Psi \rhd P \rightsquigarrow$<*Rel*> Q› *QTrans*
obtain P' **where** *PTrans*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'$ **and** $P'RelQ'$: $(\Psi, P', Q') \in Rel$
by(*blast dest*: *weakSimE*)

note *PTrans*
moreover from $P'RelQ'$ $Q'RelR'$ *Set* **have** $(\Psi, P', R') \in Rel''$ **by** *blast*
ultimately show *?case* **by** *blast*
qed
qed

lemma *quietSim*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

assumes *quiet P*
and *eqvt Rel*
and *cQuiet*: $\bigwedge P.$ *quiet* $P \Longrightarrow (\Psi, \mathbf{0}, P) \in Rel$

shows $\Psi \rhd \mathbf{0} \rightsquigarrow$<*Rel*> P
using ‹*eqvt Rel*›
proof(*induct rule*: *weakSimI*[*of* - - - - ()])
case(*cAct* $\Psi'\ \alpha\ P'$)
from ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› ‹$\alpha \neq \tau$› **have** *False* **using** ‹*quiet P*›
by(*cases rule*: *actionCases*[**where** α=α]) (*auto intro*: *quietOutput quietInput*)
thus *?case* **by** *simp*
next
case(*cTau* P')
from ‹$\Psi \rhd P \longmapsto \tau \prec P'$› ‹*quiet P*› **have** *quiet* P'

by(*erule-tac quiet.cases*) (*force simp add*: *residualInject*)
have $\Psi \rhd P \Longrightarrow\hat{}_\tau P$ **by** *simp*
moreover from ‹*quiet P'*› **have** $(\Psi, \mathbf{0}, P') \in Rel$ **by**(*rule cQuiet*)
ultimately show *?case* **by** *blast*
qed

end

end

theory *Weak-Stat-Imp*
imports *Tau-Chain*
begin

context *env* **begin**

definition
weakStatImp :: $'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow$
$('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set \Rightarrow$
$('a, 'b, 'c)\ psi \Rightarrow bool$ (‹- ⊳ - ≲<-> -› [80, 80, 80, 80] 80)
where $\Psi \rhd P \lesssim\!\!\approx\!<Rel>\ Q \equiv \forall \Psi'.\ \exists Q'\ Q''.\ \Psi \rhd Q \Longrightarrow\hat{}_\tau Q' \wedge insertAssertion(extractFrame\ P)\ \Psi \hookrightarrow_F insertAssertion(extractFrame\ Q')\ \Psi \wedge \Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau Q'' \wedge (\Psi \otimes \Psi', P, Q'') \in Rel$

lemma *weakStatImpMonotonic*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $A :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $Q :: ('a, 'b, 'c)\ psi$
and $B :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $\Psi \rhd P \lesssim\!\!\approx\!<A>\ Q$
and $A \subseteq B$

shows $\Psi \rhd P \lesssim\!\!\approx\!<B>\ Q$
using *assms*
by(*auto simp add*: *weakStatImp-def*) *blast*

lemma *weakStatImpI*[*case-names cStatImp*]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $Q :: ('a, 'b, 'c)\ psi$
and $\Psi' :: 'b$

assumes $\bigwedge \Psi'.\ \exists Q'\ Q''.\ \Psi \rhd Q \Longrightarrow\hat{}_\tau Q' \wedge insertAssertion(extractFrame\ P)\ \Psi \hookrightarrow_F insertAssertion(extractFrame\ Q')\ \Psi \wedge \Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau Q'' \wedge (\Psi \otimes \Psi', P, Q'') \in Rel$

shows $\Psi \rhd P \lessapprox <Rel> Q$
using *assms*
by(*auto simp add*: *weakStatImp-def*)

lemma *weakStatImpE*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes $\Psi \rhd P \lessapprox <Rel> Q$

obtains Q' Q'' **where** $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$ **and** *insertAssertion*(*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* Q') Ψ **and** $\Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau Q''$ **and** $(\Psi \otimes \Psi', P, Q'') \in Rel$

using *assms*
by(*auto simp add*: *weakStatImp-def*) *blast*

lemma *weakStatImpClosed*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*

assumes *EqvtRel*: *eqvt Rel*
and *PStatImpQ*: $\Psi \rhd P \lessapprox <Rel> Q$

shows $(p \cdot \Psi) \rhd (p \cdot P) \lessapprox <Rel> (p \cdot Q)$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
from *PStatImpQ* **obtain** Q' Q'' **where** *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$
and *PimpQ'*: *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q') Ψ
and *Q'Chain*: $\Psi \otimes (rev(p::name\ prm) \cdot \Psi') \rhd Q' \Longrightarrow\hat{}_\tau Q''$ **and** $(\Psi \otimes (rev\ p \cdot \Psi'), P, Q'') \in Rel$
by(*rule weakStatImpE*)
from *QChain* **have** $(p \cdot \Psi) \rhd (p \cdot Q) \Longrightarrow\hat{}_\tau (p \cdot Q')$ **by**(*rule tauChainEqvt*)
moreover from *PimpQ'* **have** *insertAssertion* (*extractFrame* $(p \cdot P)$) $(p \cdot \Psi) \hookrightarrow_F$ *insertAssertion* (*extractFrame*$(p \cdot Q')$) $(p \cdot \Psi)$
by(*drule-tac p=p* **in** *FrameStatImpClosed*) (*simp add*: *eqvts*)
moreover from *Q'Chain* **have** $(p \cdot \Psi) \otimes \Psi' \rhd (p \cdot Q') \Longrightarrow\hat{}_\tau (p \cdot Q'')$ **by**(*drule-tac p=p* **in** *tauChainEqvt*) (*simp add*: *eqvts*)
moreover from ‹$(\Psi \otimes (rev\ p \cdot \Psi'), P, Q'') \in Rel$› *EqvtRel* **have** $((p \cdot \Psi) \otimes \Psi', (p \cdot P), (p \cdot Q'')) \in Rel$
by(*drule-tac p=p* **in** *eqvtI*) (*auto simp add*: *eqvts*)

ultimately show *?case*
by *blast*
qed

lemma *weakStatImpReflexive*:
fixes $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

assumes $\{(\Psi, P, P) \mid \Psi\ P.\ True\} \subseteq Rel$

shows $\Psi \rhd P \lessapprox <Rel> P$
using *assms*
by(*auto simp add*: *weakStatImp-def weakTransition-def dest*: *rtrancl-into-rtrancl*) *force+*

lemma *weakStatImpTransitive*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $Q :: ('a, 'b, 'c)\ psi$
and $Rel' :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $R :: ('a, 'b, 'c)\ psi$
and $Rel'' :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes *PStatImpQ*: $\Psi \rhd P \lessapprox <Rel> Q$
and *QRelR*: $(\Psi, Q, R) \in Rel'$
and *Set*: $\{(\Psi', S, U) \mid \Psi'\ S\ U.\ \exists T.\ (\Psi', S, T) \in Rel \wedge (\Psi', T, U) \in Rel'\} \subseteq Rel''$
and *C1*: $\bigwedge \Psi'\ S\ T.\ (\Psi', S, T) \in Rel' \Longrightarrow \Psi' \rhd S \lessapprox <Rel'> T$
and *C2*: $\bigwedge \Psi'\ S\ T\ S'.\ [\![(\Psi', S, T) \in Rel'; \Psi' \rhd S \Longrightarrow\hat{}_\tau\ S']\!] \Longrightarrow \exists T'.\ \Psi' \rhd T \Longrightarrow\hat{}_\tau\ T' \wedge (\Psi', S', T') \in Rel'$

shows $\Psi \rhd P \lessapprox <Rel''> R$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
from ‹$\Psi \rhd P \lessapprox <Rel> Q$› **obtain** $Q'\ Q''$ **where** *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'$
and *PimpQ'*: $insertAssertion\ (extractFrame\ P)\ \Psi \hookrightarrow_F insertAssertion\ (extractFrame\ Q')\ \Psi$
and *Q'Chain*: $\Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$ **and** $(\Psi \otimes \Psi', P, Q'') \in Rel$
by(*rule weakStatImpE*)
from *QChain* ‹$(\Psi, Q, R) \in Rel'$› **obtain** R' **where** *RChain*: $\Psi \rhd R \Longrightarrow\hat{}_\tau\ R'$ **and** $(\Psi, Q', R') \in Rel'$
by(*metis C2*)
from ‹$(\Psi, Q', R') \in Rel'$› **obtain** $R''\ R'''$ **where** *R'Chain*: $\Psi \rhd R' \Longrightarrow\hat{}_\tau\ R''$
and *Q'impR''*: $insertAssertion\ (extractFrame\ Q')\ \Psi \hookrightarrow_F insertAssertion\ (extractFrame\ R'')\ \Psi$
and *R''Chain*: $\Psi \otimes \Psi' \rhd R'' \Longrightarrow\hat{}_\tau\ R'''$ **and**

$(\Psi \otimes \Psi', Q', R''') \in Rel'$
by(*blast dest*: *C1 weakStatImpE*)
from *RChain R'Chain* **have** $\Psi \rhd R \Longrightarrow\hat{}_\tau R''$ **by** *auto*
moreover from *PimpQ' Q'impR''* **have** *insertAssertion* (*extractFrame P*) Ψ $\hookrightarrow_F$ *insertAssertion* (*extractFrame R''*) Ψ
by(*rule FrameStatImpTrans*)
moreover from *Q'Chain* ‹$(\Psi \otimes \Psi', Q', R''') \in Rel'$› **obtain** R'''' **where** *R'''Chain*: $\Psi \otimes \Psi' \rhd R''' \Longrightarrow\hat{}_\tau R''''$ **and** $(\Psi \otimes \Psi', Q'', R'''') \in Rel'$
by(*metis C2*)
from *R''Chain R'''Chain* **have** $\Psi \otimes \Psi' \rhd R'' \Longrightarrow\hat{}_\tau R''''$ **by** *auto*
moreover from ‹$(\Psi \otimes \Psi', P, Q'') \in Rel$› ‹$(\Psi \otimes \Psi', Q'', R'''') \in Rel'$› *Set* **have** $(\Psi \otimes \Psi', P, R'''') \in Rel''$ **by** *blast*
ultimately show *?case*
by *blast*
qed

lemma *weakStatImpStatEq*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and Ψ' :: *'b*

assumes *PSimQ*: $\Psi \rhd P \lessapprox <Rel> Q$
and $\Psi \simeq \Psi'$
and *C1*: $\bigwedge \Psi'\ R\ S\ \Psi''.\ [\![(\Psi', R, S) \in Rel; \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'', R, S) \in Rel'$

shows $\Psi' \rhd P \lessapprox <Rel'> Q$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ'')
from ‹$\Psi \rhd P \lessapprox <Rel> Q$› **obtain** $Q'\ Q''$ **where** *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$
and *PimpQ*: *insertAssertion* (*extractFrame P*) Ψ $\hookrightarrow_F$ *insertAssertion* (*extractFrame Q'*) Ψ
and *Q'Chain*: $\Psi \otimes \Psi'' \rhd Q' \Longrightarrow\hat{}_\tau Q''$ **and** $(\Psi \otimes \Psi'', P, Q'') \in Rel$
by(*rule weakStatImpE*)
from *QChain* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \rhd Q \Longrightarrow\hat{}_\tau Q'$ **by**(*rule tauChainStatEq*)
moreover from *PimpQ* ‹$\Psi \simeq \Psi'$› **have** *insertAssertion* (*extractFrame P*) Ψ' $\hookrightarrow_F$ *insertAssertion* (*extractFrame Q'*) Ψ'
by(*rule insertAssertionStatImp*)
moreover from *Q'Chain* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \otimes \Psi'' \rhd Q' \Longrightarrow\hat{}_\tau Q''$ **by**(*metis tauChainStatEq Composition*)
moreover from ‹$(\Psi \otimes \Psi'', P, Q'') \in Rel$› ‹$\Psi \simeq \Psi'$› **have** $(\Psi' \otimes \Psi'', P, Q'') \in Rel'$ **by**(*blast intro*: *Composition C1*)
ultimately show *?case*
by *blast*
qed

lemma *statImpWeakStatImp*:

fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *PImpQ*: $insertAssertion(extractFrame\ P)\ \Psi \hookrightarrow_F insertAssertion(extractFrame\ Q)\ \Psi$
and *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in Rel$

shows $\Psi \rhd P \lessapprox <Rel> Q$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q$ **by** *simp*
moreover note *PImpQ*
moreover have $\Psi \otimes \Psi' \rhd Q \Longrightarrow\hat{}_\tau\ Q$ **by** *simp*
moreover have $(\Psi \otimes \Psi',\ P,\ Q) \in Rel$ **by**(*rule C1*)
ultimately show *?case*
by *blast*
qed

end

end

theory *Bisimulation*
imports *Simulation*
begin

context *env* **begin**

lemma *monoCoinduct*: $\bigwedge x\ y\ xa\ xb\ xc\ P\ Q\ \Psi.$
$x \leq y \Longrightarrow$
$(\Psi \rhd Q \rightsquigarrow[\{(xc,\ xb,\ xa).\ x\ xc\ xb\ xa\}]\ P) \longrightarrow$
$(\Psi \rhd Q \rightsquigarrow[\{(xb,\ xa,\ xc).\ y\ xb\ xa\ xc\}]\ P)$
apply *auto*
apply(*rule monotonic*)
by(*auto dest*: *le-funE*)

coinductive-set *bisim* :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
where
step: $[\![(insertAssertion\ (extractFrame\ P))\ \Psi \simeq_F (insertAssertion\ (extractFrame\ Q)\ \Psi);$
$\Psi \rhd P \rightsquigarrow[bisim]\ Q;$
$\forall\Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in bisim;\ (\Psi,\ Q,\ P) \in bisim]\!] \Longrightarrow (\Psi,\ P,\ Q) \in bisim$
monos *monoCoinduct*

abbreviation
bisimJudge (‹- ▷ - ∼ -› [*70*, *70*, *70*] *65*) **where** $\Psi \rhd P \sim Q \equiv (\Psi,\ P,\ Q) \in bisim$

abbreviation
bisimNilJudge (‹- ∼ -› [70, 70] 65) **where** $P \sim Q \equiv SBottom' \rhd P \sim Q$

lemma *bisimCoinductAux*[*consumes 1*]:
fixes $F :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $X :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow insertAssertion\ (extractFrame\ P)\ \Psi \simeq_F insertAssertion\ (extractFrame\ Q)\ \Psi \wedge$
$(\Psi \rhd P \rightsquigarrow [(X \cup bisim)]\ Q) \wedge$
$(\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in X \vee (\Psi \otimes \Psi', P, Q) \in bisim) \wedge$
$((\Psi, Q, P) \in X \vee (\Psi, Q, P) \in bisim)$

shows $(\Psi, P, Q) \in bisim$
proof −
have $X \cup bisim = \{(\Psi, P, Q).\ (\Psi, P, Q) \in X \vee (\Psi, P, Q) \in bisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct simp*
qed

lemma *bisimCoinduct*[*consumes 1*, *case-names cStatEq cSim cExt cSym*]:
fixes $F :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $X :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi'\ R\ S.\ (\Psi', R, S) \in X \Longrightarrow insertAssertion\ (extractFrame\ R)\ \Psi' \simeq_F insertAssertion\ (extractFrame\ S)\ \Psi'$
and $\bigwedge \Psi'\ R\ S.\ (\Psi', R, S) \in X \Longrightarrow \Psi' \rhd R \rightsquigarrow [(X \cup bisim)]\ S$
and $\bigwedge \Psi'\ R\ S\ \Psi''.\ (\Psi', R, S) \in X \Longrightarrow (\Psi' \otimes \Psi'', R, S) \in X \vee (\Psi' \otimes \Psi'', R, S) \in bisim$
and $\bigwedge \Psi'\ R\ S.\ (\Psi', R, S) \in X \Longrightarrow (\Psi', S, R) \in X \vee (\Psi', S, R) \in bisim$

shows $(\Psi, P, Q) \in bisim$
proof −
have $X \cup bisim = \{(\Psi, P, Q).\ (\Psi, P, Q) \in X \vee (\Psi, P, Q) \in bisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct simp*
qed

lemma *bisimWeakCoinductAux*[*consumes 1*]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$

and $X :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow insertAssertion\ (extractFrame\ P)\ \Psi \simeq_F insertAssertion\ (extractFrame\ Q)\ \Psi \wedge$
$\Psi \rhd P \rightsquigarrow [X]\ Q \wedge$
$(\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in X) \wedge (\Psi, Q, P) \in X$

shows $(\Psi, P, Q) \in bisim$
using *assms*
by(*coinduct rule*: *bisimCoinductAux*) (*blast intro*: *monotonic*)

lemma *bisimWeakCoinduct*[*consumes 1*, *case-names cStatEq cSim cExt cSym*]:
fixes $F :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $X :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow insertAssertion\ (extractFrame\ P)\ \Psi \simeq_F insertAssertion\ (extractFrame\ Q)\ \Psi$
and $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow [X]\ Q$
and $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi \otimes \Psi', P, Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi, Q, P) \in X$

shows $(\Psi, P, Q) \in bisim$
proof −
have $X \cup bisim = \{(\Psi, P, Q).\ (\Psi, P, Q) \in X \vee (\Psi, P, Q) \in bisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by(*coinduct rule*: *bisimCoinduct*) (*blast intro*: *monotonic*)+
qed

lemma *bisimE*:
fixes $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $\Psi :: 'b$
and $\Psi' :: 'b$

assumes $(\Psi, P, Q) \in bisim$

shows $insertAssertion\ (extractFrame\ P)\ \Psi \simeq_F insertAssertion\ (extractFrame\ Q)\ \Psi$
and $\Psi \rhd P \rightsquigarrow [bisim]\ Q$
and $(\Psi \otimes \Psi', P, Q) \in bisim$
and $(\Psi, Q, P) \in bisim$
using *assms*
by(*auto simp add*: *intro*: *bisim.cases*)

lemma *bisimI*:

fixes P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and Ψ :: $'b$

assumes *insertAssertion* (*extractFrame* P) $\Psi \simeq_F$ *insertAssertion* (*extractFrame* Q) Ψ
and $\Psi \rhd P \rightsquigarrow[bisim]\ Q$
and $\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in bisim$
and $(\Psi, Q, P) \in bisim$

shows $(\Psi, P, Q) \in bisim$
using *assms*
by(*auto intro*: *bisim.step*)

lemma *bisimReflexive*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

shows $\Psi \rhd P \sim P$
proof −
let $?X = \{(\Psi, P, P) \mid \Psi\ P.\ True\}$
have $(\Psi, P, P) \in ?X$ **by** *simp*
thus *?thesis*
by(*coinduct rule*: *bisimWeakCoinduct*, *auto intro*: *reflexive*)
qed

lemma *bisimClosed*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*

assumes *PBisimQ*: $\Psi \rhd P \sim Q$

shows $(p \cdot \Psi) \rhd\ (p \cdot P) \sim (p \cdot Q)$
proof −
let $?X = \{(p \cdot \Psi, p \cdot P, p \cdot Q) \mid (p{::}name\ prm)\ \Psi\ \ P\ Q.\ \Psi \rhd P \sim Q\}$
from *PBisimQ* **have** $(p \cdot \Psi, p \cdot P, p \cdot Q) \in ?X$ **by** *blast*
thus *?thesis*
proof(*coinduct rule*: *bisimWeakCoinduct*)
case(*cStatEq* Ψ P Q)
have $\bigwedge \Psi\ P\ Q\ (p{::}name\ prm).$ *insertAssertion* (*extractFrame* P) $\Psi \simeq_F$ *insertAssertion* (*extractFrame* Q) $\Psi \Longrightarrow$
insertAssertion (*extractFrame*$(p \cdot P)$) $(p \cdot \Psi) \simeq_F$ *insertAssertion* (*extractFrame*$(p \cdot Q)$) $(p \cdot \Psi)$
by(*drule-tac* $p = p$ **in** *FrameStatEqClosed*) (*simp add*: *eqvts*)

with ‹$(\Psi, P, Q) \in ?X$› **show** *?case* **by**(*blast dest*: *bisimE*)

```
  next
    case(cSim Ψ P Q)
    {
      fix p :: name prm
      fix Ψ P Q
      have eqvt ?X
        apply(auto simp add: eqvt-def)
        apply(rule-tac x=pa@p in exI)
        by(auto simp add: pt2[OF pt-name-inst])
      moreover assume Ψ ▷ P ⇝[bisim] Q
      hence Ψ ▷ P ⇝[?X] Q
        apply(rule-tac A=bisim in monotonic, auto)
        by(rule-tac x=[]::name prm in exI) auto
      ultimately have ((p::name prm) • Ψ) ▷ (p • P) ⇝[?X] (p • Q)
        by(rule-tac simClosed)
    }
    with ‹(Ψ, P, Q) ∈ ?X› show ?case
      by(blast dest: bisimE)
  next
    case(cExt Ψ P Q Ψ')
    {
      fix p :: name prm
      fix Ψ P Q Ψ'
      assume ∀Ψ'. (Ψ ⊗ Ψ', P, Q) ∈ bisim
      hence ((p • Ψ) ⊗ Ψ', p • P, p • Q) ∈ ?X
        apply(auto, rule-tac x=p in exI)
        apply(rule-tac x=Ψ ⊗ (rev p • Ψ') in exI)
        by(auto simp add: eqvts)
    }
    with ‹(Ψ, P, Q) ∈ ?X› show ?case
      by(blast dest: bisimE)
  next
    case(cSym Ψ P Q)
    thus ?case
      by(blast dest: bisimE)
  qed
qed

lemma bisimEqvt[simp]:
  shows eqvt bisim
by(auto simp add: eqvt-def bisimClosed)

lemma statEqBisim:
  fixes Ψ  :: 'b
  and   P  :: ('a, 'b, 'c) psi
  and   Q  :: ('a, 'b, 'c) psi
  and   Ψ' :: 'b

  assumes Ψ ▷ P ∼ Q
```

and $\Psi \simeq \Psi'$

shows $\Psi' \rhd P \sim Q$
proof −
let $?X = \{(\Psi', P, Q) \mid \Psi\ P\ Q\ \Psi'.\ \Psi \rhd P \sim Q \wedge \Psi \simeq \Psi'\}$
from ‹$\Psi \rhd P \sim Q$› ‹$\Psi \simeq \Psi'$› **have** $(\Psi', P, Q) \in ?X$ **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* $\Psi'\ P\ Q$)
from ‹$(\Psi', P, Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \sim Q$ **and** $\Psi \simeq \Psi'$
by *auto*
from ‹$\Psi \rhd P \sim Q$› **have** *PeqQ*: *insertAssertion* (*extractFrame* P) $\Psi \simeq_F$ *insertAssertion* (*extractFrame* Q) Ψ
by(*rule bisimE*)

obtain $A_P\ \Psi_P$ **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* \Psi'$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*
obtain $A_Q\ \Psi_Q$ **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* \Psi'$
by(*rule-tac* C=(Ψ, Ψ') **in** *freshFrame*) *auto*

from *PeqQ FrP FrQ* ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$\Psi \simeq \Psi'$›
have $\langle A_P, \Psi' \otimes \Psi_P\rangle \simeq_F \langle A_Q, \Psi' \otimes \Psi_Q\rangle$
by *simp* (*metis frameIntComposition FrameStatEqTrans FrameStatEqSym*)
with *FrP FrQ* ‹$A_P \sharp* \Psi'$› ‹$A_Q \sharp* \Psi'$› **show** *?case* **by** *simp*
next
case(*cSim* $\Psi'\ P\ Q$)
from ‹$(\Psi', P, Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \sim Q$ **and** $\Psi \simeq \Psi'$
by *auto*
from ‹$\Psi \rhd P \sim Q$› **have** $\Psi \rhd P \rightsquigarrow[bisim]\ Q$ **by**(*blast dest*: *bisimE*)
moreover have *eqvt ?X*
by(*auto simp add*: *eqvt-def*) (*metis bisimClosed AssertionStatEqClosed*)
hence *eqvt*(*?X* ∪ *bisim*) **by** *auto*
moreover note ‹$\Psi \simeq \Psi'$›
moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ [\![\Psi \rhd P \sim Q;\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi', P, Q) \in ?X \cup bisim$
by *auto*
ultimately show *?case*
by(*rule statEqSim*)
next
case(*cExt* $\Psi'\ P\ Q\ \Psi''$)
from ‹$(\Psi', P, Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \sim Q$ **and** $\Psi \simeq \Psi'$
by *auto*
from ‹$\Psi \rhd P \sim Q$› **have** $\Psi \otimes \Psi'' \rhd P \sim Q$ **by**(*rule bisimE*)
moreover from ‹$\Psi \simeq \Psi'$› **have** $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$ **by**(*rule Composition*)
ultimately show *?case* **by** *blast*
next
case(*cSym* $\Psi'\ P\ Q$)

```
    from ‹(Ψ′, P, Q) ∈ ?X› obtain Ψ where Ψ ▷ P ∼ Q and Ψ ≃ Ψ′
      by auto
    from ‹Ψ ▷ P ∼ Q› have Ψ ▷ Q ∼ P by(rule bisimE)
    thus ?case using ‹Ψ ≃ Ψ′› by auto
  qed
qed

lemma bisimTransitive:
  fixes Ψ :: ′b
  and   P :: (′a, ′b, ′c) psi
  and   Q :: (′a, ′b, ′c) psi
  and   R :: (′a, ′b, ′c) psi

  assumes PQ: Ψ ▷ P ∼ Q
  and     QR: Ψ ▷ Q ∼ R

  shows Ψ ▷ P ∼ R
proof −
  let ?X = {(Ψ, P, R) | Ψ P Q R. Ψ ▷ P ∼ Q ∧ Ψ ▷ Q ∼ R}
  from PQ QR have (Ψ, P, R) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: bisimCoinduct)
    case(cStatEq Ψ P R)
    thus ?case by(blast dest: bisimE FrameStatEqTrans)
  next
    case(cSim Ψ P R)
    {
      fix Ψ P Q R
      assume Ψ ▷ P ⇝[bisim] Q and Ψ ▷ Q ⇝[bisim] R
      moreover have eqvt ?X
        by(force simp add: eqvt-def dest: bisimClosed)
      with bisimEqvt have eqvt (?X ∪ bisim) by blast
      moreover have ?X ⊆ ?X ∪ bisim by auto
      ultimately have Ψ ▷ P ⇝[(?X ∪ bisim)] R
        by(force intro: transitive)
    }
    with ‹(Ψ, P, R) ∈ ?X› show ?case
      by(blast dest: bisimE)
  next
    case(cExt Ψ P R Ψ′)
    thus ?case by(blast dest: bisimE)
  next
    case(cSym Ψ P R)
    thus ?case by(blast dest: bisimE)
  qed
qed

lemma weakTransitiveCoinduct[case-names cStatEq cSim cExt cSym, case-conclusion
bisim step, consumes 2]:
```

fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes p: $(\Psi,\ P,\ Q) \in X$
and $Eqvt$: $eqvt\ X$
and $rStatEq$: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow insertAssertion\ (extractFrame\ P)\ \Psi \simeq_F insertAssertion\ (extractFrame\ Q)\ \Psi$
and $rSim$: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow[(\{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge$
$(\Psi,\ P',\ Q') \in X \wedge$
$\Psi \rhd Q' \sim Q\})]\ Q$
and $rExt$: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X$
and $rSym$: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi,\ Q,\ P) \in X$

shows $\Psi \rhd P \sim Q$
proof −
let $?X = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \sim Q\}$
from p **have** $(\Psi,\ P,\ Q) \in ?X$
by(*blast intro*: *bisimReflexive*)
thus *?thesis*
proof(*coinduct rule*: *bisimWeakCoinduct*)
case(*cStatEq* $\Psi\ P\ Q$)
thus *?case*
by(*blast dest*: *rStatEq bisimE FrameStatEqTrans*)
next
case(*cSim* $\Psi\ P\ Q$)
{
fix $\Psi\ P\ P'\ Q'\ Q$
assume $\Psi \rhd P \rightsquigarrow[bisim]\ P'$
moreover assume $P'RelQ'$: $(\Psi,\ P',\ Q') \in X$
hence $\Psi \rhd P' \rightsquigarrow[?X]\ Q'$ **by**(*rule rSim*)
moreover from ‹*eqvt X*› $P'RelQ'$ **have** *eqvt ?X*
apply(*auto simp add*: *eqvt-def*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*rule-tac x=p* · $P'a$ **in** *exI*, *simp*)
by(*rule-tac x=p* · $Q'a$ **in** *exI*, *auto*)
ultimately have $\Psi \rhd P \rightsquigarrow[?X]\ Q'$
by(*force intro*: *transitive dest*: *bisimTransitive*)
moreover assume $\Psi \rhd Q' \rightsquigarrow[bisim]\ Q$
ultimately have $\Psi \rhd P \rightsquigarrow[?X]\ Q$ **using** ‹*eqvt ?X*›
by(*force intro*: *transitive dest*: *bisimTransitive*)
}
with ‹$(\Psi,\ P,\ Q) \in ?X$› **show** *?case*
by(*blast dest*: *bisimE*)
next

case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*blast dest*: *bisimE intro*: *rExt*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *bisimE intro*: *rSym*)
qed
qed

lemma *weakTransitiveCoinduct'*[*case-names cStatEq cSim cExt cSym*, *case-conclusion bisim step*, *consumes 2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and X :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*) *set*

assumes *p*: $(\Psi, P, Q) \in X$
and *Eqvt*: *eqvt* X
and *rStatEq*: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow$ *insertAssertion* (*extractFrame* P) Ψ $\simeq_F$ *insertAssertion* (*extractFrame* Q) Ψ
and *rSim*: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow[(\{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge$
$(\Psi, P', Q') \in X \wedge$
$\Psi \rhd Q' \sim Q\})]\ Q$
and *rExt*: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi \otimes \Psi', P, Q) \in X$
and *rSym*: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow$
$(\Psi, Q, P) \in \{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \rhd Q' \sim Q\}$

shows $\Psi \rhd P \sim Q$
proof −
let *?X* = $\{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \rhd Q' \sim Q\}$
from *p* **have** $(\Psi, P, Q) \in$ *?X*
by(*blast intro*: *bisimReflexive*)
thus *?thesis*
proof(*coinduct rule*: *bisimWeakCoinduct*)
case(*cStatEq* Ψ P Q)
thus *?case*
by(*blast dest*: *rStatEq bisimE FrameStatEqTrans*)
next
case(*cSim* Ψ P Q)
{
fix Ψ P P' Q' Q
assume $\Psi \rhd P \rightsquigarrow[bisim]\ P'$
moreover assume *P'RelQ'*: $(\Psi, P', Q') \in X$
hence $\Psi \rhd P' \rightsquigarrow[?X]\ Q'$ **by**(*rule rSim*)
moreover from ‹*eqvt X*› *P'RelQ'* **have** *eqvt ?X*
apply(*auto simp add*: *eqvt-def*)
apply(*drule-tac p=p* **in** *bisimClosed*)

apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*rule-tac x=p* · *P'a* **in** *exI*, *simp*)
by(*rule-tac x=p* · *Q'a* **in** *exI*, *auto*)
ultimately have $\Psi \triangleright P \rightsquigarrow[?X]\ Q'$
by(*force intro*: *transitive dest*: *bisimTransitive*)
moreover assume $\Psi \triangleright Q' \rightsquigarrow[bisim]\ Q$
ultimately have $\Psi \triangleright P \rightsquigarrow[?X]\ Q$ **using** ‹*eqvt ?X*›
by(*force intro*: *transitive dest*: *bisimTransitive*)
}
with ‹$(\Psi, P, Q) \in ?X$› **show** *?case*
by(*blast dest*: *bisimE*)
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*blast dest*: *bisimE intro*: *rExt*)
next
case(*cSym* Ψ P Q)
thus *?case*
apply *auto*
apply(*drule rSym*)
apply *auto*
by(*metis bisimTransitive bisimE*(*4*))
qed
qed

lemma *weakTransitiveCoinduct''*[*case-names cStatEq cSim cExt cSym*, *case-conclusion bisim step*, *consumes 2*]:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and X :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*

assumes *p*: $(\Psi, P, Q) \in X$
and *Eqvt*: *eqvt X*
and *rStatEq*: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow$ *insertAssertion* (*extractFrame* P) Ψ $\simeq_F$ *insertAssertion* (*extractFrame* Q) Ψ
and *rSim*: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \triangleright P \rightsquigarrow[(\{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\})]\ Q$
and *rExt*: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in \{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\} \Longrightarrow (\Psi \otimes \Psi', P, Q) \in \{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\}$
and *rSym*: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in \{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\} \Longrightarrow (\Psi, Q, P) \in \{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\}$

shows $\Psi \triangleright P \sim Q$

```
proof −
  let ?X = {(Ψ, P, Q) | Ψ P P′ Q′ Q. Ψ ▷ P ∼ P′ ∧ (Ψ, P′, Q′) ∈ X ∧ Ψ ▷ Q′
∼ Q}
  from p have (Ψ, P, Q) ∈ ?X
    by(blast intro: bisimReflexive)
  thus ?thesis
  proof(coinduct rule: bisimWeakCoinduct)
    case(cStatEq Ψ P Q)
    thus ?case
      by(blast dest: rStatEq bisimE FrameStatEqTrans)
  next
    case(cSim Ψ P Q)
    {
      fix Ψ P P′ Q′ Q
      assume Ψ ▷ P ⇝[bisim] P′
      moreover assume P′RelQ′: (Ψ, P′, Q′) ∈ X
      hence Ψ ▷ P′ ⇝[?X] Q′ by(rule rSim)
      moreover from ‹eqvt X› P′RelQ′ have eqvt ?X
        apply(auto simp add: eqvt-def)
        apply(drule-tac p=p in bisimClosed)
        apply(drule-tac p=p in bisimClosed)
        apply(rule-tac x=p • P′a in exI, simp)
        by(rule-tac x=p • Q′a in exI, auto)
      ultimately have Ψ ▷ P ⇝[?X] Q′
        by(force intro: transitive dest: bisimTransitive)
      moreover assume Ψ ▷ Q′ ⇝[bisim] Q
      ultimately have Ψ ▷ P ⇝[?X] Q using ‹eqvt ?X›
        by(force intro: transitive dest: bisimTransitive)
    }
    with ‹(Ψ, P, Q) ∈ ?X› show ?case
      by(blast dest: bisimE)
  next
    case(cExt Ψ P Q Ψ′)
    thus ?case by(rule-tac rExt)
  next
    case(cSym Ψ P Q)
    thus ?case by(rule-tac rSym)
  qed
qed

lemma transitiveCoinduct[case-names cStatEq cSim cExt cSym, case-conclusion
bisim step, consumes 2]:
  fixes Ψ :: ′b
  and   P :: (′a, ′b, ′c) psi
  and   Q :: (′a, ′b, ′c) psi
  and   X :: (′b × (′a, ′b, ′c) psi × (′a, ′b, ′c) psi) set

  assumes p: (Ψ, P, Q) ∈ X
  and Eqvt: eqvt X
```

and *rStatEq*: $\bigwedge\Psi'\ R\ S.\ (\Psi',\ R,\ S) \in X \Longrightarrow$ *insertAssertion* (*extractFrame* R) Ψ' $\simeq_F$ *insertAssertion* (*extractFrame* S) Ψ'
and *rSim*: $\bigwedge\Psi'\ R\ S.\ (\Psi',\ R,\ S) \in X \Longrightarrow \Psi' \rhd R \rightsquigarrow[(\{(\Psi',\ R,\ S) \mid \Psi'\ R\ R'\ S'\ S.\ \Psi' \rhd R \sim R' \wedge$
$((\Psi',\ R',\ S') \in X \vee \Psi' \rhd R' \sim S') \wedge$
$\Psi' \rhd S' \sim S\})]\ S$
and *rExt*: $\bigwedge\Psi'\ R\ S\ \Psi''.\ (\Psi',\ R,\ S) \in X \Longrightarrow (\Psi' \otimes \Psi'',\ R,\ S) \in X \vee \Psi' \otimes \Psi'' \rhd R \sim S$
and *rSym*: $\bigwedge\Psi'\ R\ S.\ (\Psi',\ R,\ S) \in X \Longrightarrow (\Psi',\ S,\ R) \in X \vee \Psi' \rhd S \sim R$

shows $\Psi \rhd P \sim Q$
proof −
from p **have** $(\Psi,\ P,\ Q) \in (X \cup bisim)$
by *blast*
moreover from ‹*eqvt X*› *bisimEqvt* **have** *eqvt* $(X \cup bisim)$
by *auto*
ultimately show *?thesis*
proof(*coinduct rule*: *weakTransitiveCoinduct′*)
case(*cStatEq* Ψ P Q)
thus *?case*
by(*blast intro*: *rStatEq dest*: *bisimE*)
next
case(*cSim* Ψ P Q)
thus *?case*
apply *auto*
apply(*blast intro*: *rSim*)
apply(*drule bisimE*(*2*))
apply(*rule-tac A=bisim* **in** *monotonic*, *simp*)
by(*force intro*: *bisimReflexive*)
next
case(*cExt* Ψ P Q Ψ')
thus *?case*
by(*blast dest*: *bisimE rExt*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *bisimE rSym intro*: *bisimReflexive*)
qed
qed

lemma *transitiveCoinduct′*[*case-names cStatEq cSim cExt cSym*, *case-conclusion bisim step*, *consumes 2*]:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes p: $(\Psi,\ P,\ Q) \in X$

and *Eqvt*: *eqvt X*
and *rStatEq*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow$ *insertAssertion* (*extractFrame P*) Ψ $\simeq_F$ *insertAssertion* (*extractFrame Q*) Ψ
and *rSim*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow[(\{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge$
$(\Psi,\ P',\ Q') \in (X \cup bisim) \wedge$
$\Psi \rhd Q' \sim Q\})]\ Q$
and *rExt*: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X \vee \Psi \otimes \Psi' \rhd P \sim Q$
and *rSym*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow$
$(\Psi,\ Q,\ P) \in \{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge ((\Psi,\ P',\ Q') \in (X \cup bisim)) \wedge \Psi \rhd Q' \sim Q\}$

shows $\Psi \rhd P \sim Q$
proof −
from *p* **have** $(\Psi,\ P,\ Q) \in (X \cup bisim)$
by *blast*
moreover from ‹*eqvt X*› *bisimEqvt* **have** *eqvt* $(X \cup bisim)$
by *auto*
ultimately show *?thesis*
proof(*coinduct rule*: *weakTransitiveCoinduct′*)
case(*cStatEq* Ψ *P Q*)
thus *?case*
by(*blast intro*: *rStatEq dest*: *bisimE*)
next
case(*cSim* Ψ *P Q*)
thus *?case*
apply −
apply(*case-tac* $(\Psi,\ P,\ Q) \in X$ **for** *X*)
apply(*rule-tac rSim*)
apply *simp*
apply(*clarify*)
apply(*drule bisimE*(*2*))
apply(*rule-tac A=bisim* **in** *monotonic*, *simp*)
by(*force intro*: *bisimReflexive*)
next
case(*cExt* Ψ *P Q* Ψ')
thus *?case*
by(*blast dest*: *bisimE rExt*)
next
case(*cSym* Ψ *P Q*)
thus *?case*
apply *auto*
apply(*drule rSym*)
apply *auto*
apply(*rule-tac x=Q* **in** *exI*)
apply(*auto intro*: *bisimReflexive*)
apply(*rule-tac x=P* **in** *exI*)
by(*auto intro*: *bisimReflexive dest*: *bisimE*(*4*))

qed
qed

lemma *bisimSymmetric*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd P \sim Q$

shows $\Psi \rhd Q \sim P$
using *assms*
by(*rule bisimE*)

lemma *eqvtTrans*[*intro*]:
assumes *eqvt X*

shows *eqvt* $\{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge ((\Psi,\ P',\ Q') \in X \vee \Psi \rhd P' \sim Q') \wedge \Psi \rhd Q' \sim Q\}$
using *assms*
apply(*auto simp add*: *eqvt-def eqvts*)
apply(*erule-tac x*=(a, P', Q') **in** *ballE*, *auto*)
by(*blast dest*: *bisimClosed*)+

lemma *eqvtWeakTrans*[*intro*]:
assumes *eqvt X*

shows *eqvt* $\{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \sim Q\}$
using *assms*
apply(*auto simp add*: *eqvt-def eqvts*)
apply(*erule-tac x*=(a, P', Q') **in** *ballE*, *auto*)
by(*blast dest*: *bisimClosed*)+

end

end

theory *Sim-Pres*
imports *Simulation*
begin

context *env* **begin**

lemma *inputPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and *Rel* :: $('b \times ('a,\ 'b,\ 'c)$ *psi* $\times\ ('a,\ 'b,\ 'c)$ *psi*$)$ *set*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

and M :: $'a$
and $xvec$ :: $name\ list$
and N :: $'a$

assumes $PRelQ$: $\bigwedge Tvec.\ length\ xvec = length\ Tvec \Longrightarrow (\Psi,\ P[xvec::=Tvec],\ Q[xvec::=Tvec]) \in Rel$

shows $\Psi \rhd M(\!|\lambda*xvec\ N|\!).P \rightsquigarrow[Rel]\ M(\!|\lambda*xvec\ N|\!).Q$
proof(*auto simp add*: *simulation-def residual.inject psi.inject*)
fix $\alpha\ Q'$
assume $\Psi \rhd M(\!|\lambda*xvec\ N|\!).Q \longmapsto\alpha \prec Q'$
thus $\exists P'.\ \Psi \rhd M(\!|\lambda*xvec\ N|\!).P \longmapsto\alpha \prec P' \wedge (\Psi,\ P',\ Q') \in Rel$
by(*induct rule*: *inputCases*) (*auto intro*: *Input PRelQ*)
qed

lemma *outputPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and M :: $'a$
and N :: $'a$

assumes $PRelQ$: $(\Psi,\ P,\ Q) \in Rel$

shows $\Psi \rhd M\langle N\rangle.P \rightsquigarrow[Rel]\ M\langle N\rangle.Q$
proof(*auto simp add*: *simulation-def residual.inject psi.inject*)
fix $\alpha\ Q'$
assume $\Psi \rhd M\langle N\rangle.Q \longmapsto\alpha \prec Q'$
thus $\exists P'.\ \Psi \rhd M\langle N\rangle.P \longmapsto\alpha \prec P' \wedge (\Psi,\ P',\ Q') \in Rel$
by(*induct rule*: *outputCases*) (*auto intro*: *Output PRelQ*)
qed

lemma *casePres*:
fixes Ψ :: $'b$
and CsP :: $('c \times ('a,\ 'b,\ 'c)\ psi)\ list$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and CsQ :: $('c \times ('a,\ 'b,\ 'c)\ psi)\ list$
and M :: $'a$
and N :: $'a$

assumes $PRelQ$: $\bigwedge\varphi\ Q.\ (\varphi,\ Q)\ mem\ CsQ \Longrightarrow \exists P.\ (\varphi,\ P)\ mem\ CsP \wedge guarded\ P \wedge (\Psi,\ P,\ Q) \in Rel$
and Sim: $\bigwedge\Psi'\ R\ S.\ (\Psi',\ R,\ S) \in Rel \Longrightarrow \Psi' \rhd R \rightsquigarrow[Rel]\ S$
and $Rel \subseteq Rel'$

shows $\Psi \rhd Cases\ CsP \rightsquigarrow[Rel']\ Cases\ CsQ$
proof(*auto simp add*: *simulation-def residual.inject psi.inject*)
fix $\alpha\ Q'$

assume $\Psi \rhd$ *Cases CsQ* $\longmapsto\alpha \prec Q'$ **and** *bn* $\alpha \sharp* CsP$ **and** *bn* $\alpha \sharp* \Psi$
thus $\exists P'.\ \Psi \rhd$ *Cases CsP* $\longmapsto\alpha \prec P' \wedge (\Psi,\ P',\ Q') \in Rel'$
proof(*induct rule*: *caseCases*)
case(*cCase* φ Q)
from ‹$(\varphi,\ Q)$ *mem CsQ*› **obtain** P **where** $(\varphi,\ P)$ *mem CsP* **and** *guarded P* **and** $(\Psi,\ P,\ Q) \in Rel$
by(*metis PRelQ*)
from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $\Psi \rhd P \rightsquigarrow[Rel]\ Q$ **by**(*rule Sim*)
moreover from ‹*bn* $\alpha \sharp* CsP$› ‹$(\varphi,\ P)$ *mem CsP*› **have** *bn* $\alpha \sharp* P$ **by**(*auto dest*: *memFreshChain*)
moreover note ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹*bn* $\alpha \sharp* \Psi$›
ultimately obtain P' **where** *PTrans*: $\Psi \rhd P \longmapsto\alpha \prec P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ Q') \in Rel$
by(*blast dest*: *simE*)
from *PTrans* ‹$(\varphi,\ P)$ *mem CsP*› ‹$\Psi \vdash \varphi$› ‹*guarded P*› **have** $\Psi \rhd$ *Cases CsP* $\longmapsto\alpha \prec P'$
by(*rule Case*)
moreover from $P'RelQ'$ ‹$Rel \subseteq Rel'$› **have** $(\Psi,\ P',\ Q') \in Rel'$ **by** *blast*
ultimately show *?case* **by** *blast*
qed
qed

lemma *resPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and x :: *name*
and Rel' :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
and *eqvt* Rel'
and $x \sharp \Psi$
and $Rel \subseteq Rel'$
and *C1*: $\bigwedge\Psi'\ R\ S\ y.\ [\![(\Psi',\ R,\ S) \in Rel;\ y \sharp \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu y|\!)R,\ (\!|\nu y|\!)S) \in Rel'$

shows $\Psi \rhd (\!|\nu x|\!)P \rightsquigarrow[Rel']\ (\!|\nu x|\!)Q$
proof −
note ‹*eqvt* Rel'› ‹$x \sharp \Psi$›
moreover have $x \sharp (\!|\nu x|\!)P$ **and** $x \sharp (\!|\nu x|\!)Q$ **by**(*simp add*: *abs-fresh*)+
ultimately show *?thesis*
proof(*induct rule*: *simIFresh*[**where** C=()])
case(*cSim* α Q')
from ‹*bn* $\alpha \sharp* (\!|\nu x|\!)P$› ‹*bn* $\alpha \sharp* (\!|\nu x|\!)Q$› ‹$x \sharp \alpha$› **have** *bn* $\alpha \sharp* P$ **and** *bn* $\alpha \sharp* Q$ **by** *simp*+
from ‹$\Psi \rhd (\!|\nu x|\!)Q \longmapsto\alpha \prec Q'$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp Q'$› ‹*bn* $\alpha \sharp* \Psi$› ‹*bn* $\alpha \sharp* Q$› ‹*bn* $\alpha \sharp*$ *subject* α›
‹*bn* $\alpha \sharp* \Psi$› ‹*bn* $\alpha \sharp* P$› ‹$x \sharp \alpha$›

show *?case*
proof(*induct rule*: *resCases*)
case(*cOpen M xvec1 xvec2 y N Q′*)
from ‹*bn* (*M*⦇*ν*∗(*xvec1*@*y*#*xvec2*)⦈⟨*N*⟩) ♯∗ Ψ› **have** *xvec1* ♯∗ Ψ **and** *y* ♯ Ψ **and** *xvec2* ♯∗ Ψ **by** *simp*+
from ‹*bn* (*M*⦇*ν*∗(*xvec1*@*y*#*xvec2*)⦈⟨*N*⟩) ♯∗ *P*› **have** *xvec1* ♯∗ *P* **and** *y* ♯ *P* **and** *xvec2* ♯∗ *P* **by** *simp*+
from ‹*x* ♯ (*M*⦇*ν*∗(*xvec1*@*y*#*xvec2*)⦈⟨*N*⟩)› **have** *x* ♯ *xvec1* **and** *x* ≠ *y* **and** *x* ♯ *xvec2* **and** *x* ♯ *M* **by** *simp*+
from *PSimQ* ‹Ψ ⊳ *Q* ⟼*M*⦇*ν*∗(*xvec1*@*xvec2*)⦈⟨([(*x*, *y*)] ∙ *N*)⟩ ≺ ([(*x*, *y*)] ∙ *Q′*)›
‹*xvec1* ♯∗ Ψ› ‹*xvec2* ♯∗ Ψ› ‹*xvec1* ♯∗ *P*› ‹*xvec2* ♯∗ *P*›
obtain *P′* **where** *PTrans*: Ψ ⊳ *P* ⟼*M*⦇*ν*∗(*xvec1*@*xvec2*)⦈⟨([(*x*, *y*)] ∙ *N*)⟩ ≺ *P′* **and** *P′RelQ′*: (Ψ, *P′*, ([(*x*, *y*)] ∙ *Q′*)) ∈ *Rel*
by(*force dest*: *simE*)
from ‹*y* ∈ *supp N*› ‹*x* ≠ *y*› **have** *x* ∈ *supp*([(*x*, *y*)] ∙ *N*)
by(*drule-tac pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*, **where** *pi*=[(*x*, *y*)]]) (*simp add*: *eqvts calc-atm*)
with *PTrans* ‹*x* ♯ Ψ› ‹*x* ♯ *M*› ‹*x* ♯ *xvec1*› ‹*x* ♯ *xvec2*›
have Ψ ⊳ ⦇*νx*⦈*P* ⟼*M*⦇*ν*∗(*xvec1*@*x*#*xvec2*)⦈⟨([(*x*, *y*)] ∙ *N*)⟩ ≺ *P′*
by(*rule-tac Open*)
hence ([(*x*, *y*)] ∙ Ψ) ⊳ ([(*x*, *y*)] ∙ ⦇*νx*⦈*P*) ⟼([(*x*, *y*)] ∙ (*M*⦇*ν*∗(*xvec1*@*x*#*xvec2*)⦈⟨([(*x*, *y*)] ∙ *N*)⟩ ≺ *P′*))
by(*rule eqvts*)
with ‹*x* ♯ Ψ› ‹*y* ♯ Ψ› ‹*y* ♯ *P*› ‹*x* ♯ *M*› ‹*y* ♯ *M*› ‹*x* ♯ *xvec1*› ‹*y* ♯ *xvec1*› ‹*x* ♯ *xvec2*› ‹*y* ♯ *xvec2*› ‹*x* ≠ *y*›
have Ψ ⊳ ⦇*νx*⦈*P* ⟼*M*⦇*ν*∗(*xvec1*@*y*#*xvec2*)⦈⟨*N*⟩ ≺ ([(*x*, *y*)] ∙ *P′*) **by**(*simp add*: *eqvts calc-atm alphaRes*)
moreover from *P′RelQ′* ‹*Rel* ⊆ *Rel′*› ‹*eqvt Rel′*› **have** ([(*y*, *x*)] ∙ Ψ, [(*y*, *x*)] ∙ *P′*, [(*y*, *x*)] ∙ [(*x*, *y*)] ∙ *Q′*) ∈ *Rel′*
by(*force simp add*: *eqvt-def*)
with ‹*x* ♯ Ψ› ‹*y* ♯ Ψ› **have** (Ψ, [(*x*, *y*)] ∙ *P′*, *Q′*) ∈ *Rel′* **by**(*simp add*: *name-swap*)
ultimately show *?case* **by** *blast*
next
case(*cRes Q′*)
from *PSimQ* ‹Ψ ⊳ *Q* ⟼*α* ≺ *Q′*› ‹*bn α* ♯∗ Ψ› ‹*bn α* ♯∗ *P*›
obtain *P′* **where** *PTrans*: Ψ ⊳ *P* ⟼*α* ≺ *P′* **and** *P′RelQ′*: (Ψ, *P′*, *Q′*) ∈ *Rel*
by(*blast dest*: *simE*)
from *PTrans* ‹*x* ♯ Ψ› ‹*x* ♯ *α*› **have** Ψ ⊳ ⦇*νx*⦈*P* ⟼*α* ≺ ⦇*νx*⦈*P′*
by(*rule Scope*)
moreover from *P′RelQ′* ‹*x* ♯ Ψ› **have** (Ψ, ⦇*νx*⦈*P′*, ⦇*νx*⦈*Q′*) ∈ *Rel′* **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
qed
qed

lemma *resChainPres*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and *Rel* :: $('b \times ('a,\ 'b,\ 'c)$ *psi* $\times ('a,\ 'b,\ 'c)$ *psi*$)$ *set*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*
  and *xvec* :: *name list*

  assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow[Rel]\ Q$
  and *eqvt Rel*
  and *xvec* $\sharp*\ \Psi$
  and *C1*: $\bigwedge \Psi'\ R\ S\ y.\ [\![(\Psi',\ R,\ S) \in Rel;\ y \sharp \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu y|\!)R,\ (\!|\nu y|\!)S) \in$ *Rel*

  shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow[Rel]\ (\!|\nu{*}xvec|\!)Q$
using ‹*xvec* $\sharp*\ \Psi$›
proof(*induct xvec*)
  case *Nil*
  from *PSimQ* **show** *?case* **by** *simp*
next
  case(*Cons x xvec*)
  from ‹(x#*xvec*) $\sharp*\ \Psi$› **have** $x \sharp \Psi$ **and** *xvec* $\sharp*\ \Psi$ **by** *simp*+
  from ‹*xvec* $\sharp*\ \Psi$› **have** $\Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow[Rel]\ (\!|\nu{*}xvec|\!)Q$ **by**(*rule Cons*)
  moreover note ‹*eqvt Rel*› ‹$x \sharp \Psi$›
  moreover have $Rel \subseteq Rel$ **by** *simp*
  ultimately have $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec|\!)P) \rightsquigarrow[Rel]\ (\!|\nu x|\!)((\!|\nu{*}xvec|\!)Q)$ **using** *C1*
    by(*rule resPres*)
  thus *?case* **by** *simp*
qed

lemma *parPres*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and *Rel* :: $('b \times ('a,\ 'b,\ 'c)$ *psi* $\times ('a,\ 'b,\ 'c)$ *psi*$)$ *set*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*
  and R :: $('a,\ 'b,\ 'c)$ *psi*
  and *Rel'* :: $('b \times ('a,\ 'b,\ 'c)$ *psi* $\times ('a,\ 'b,\ 'c)$ *psi*$)$ *set*

  assumes *PRelQ*: $\bigwedge A_R\ \Psi_R.\ [\![extractFrame\ R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp*\ \Psi;\ A_R \sharp*\ P;$ $A_R \sharp*\ Q]\!] \Longrightarrow (\Psi \otimes \Psi_R,\ P,\ Q) \in Rel$
  and *Eqvt*: *eqvt Rel*
  and *Eqvt'*: *eqvt Rel'*

  and *StatImp*: $\bigwedge \Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow insertAssertion\ (extractFrame$ $T)\ \Psi' \hookrightarrow_F insertAssertion\ (extractFrame\ S)\ \Psi'$
  and *Sim*: $\bigwedge \Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow \Psi' \rhd S \rightsquigarrow[Rel]\ T$
  and *Ext*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel]\!] \Longrightarrow (\Psi' \otimes \Psi'',\ S,\ T) \in Rel$

  and *C1*: $\bigwedge \Psi'\ S\ T\ A_U\ \Psi_U\ U.\ [\![(\Psi' \otimes \Psi_U,\ S,\ T) \in Rel;\ extractFrame\ U =$

$\langle A_U, \Psi_U\rangle$; $A_U \sharp* \Psi'$; $A_U \sharp* S$; $A_U \sharp* T]\!] \Longrightarrow (\Psi', S \parallel U, T \parallel U) \in Rel'$
and *C2*: $\bigwedge \Psi'\ S\ T\ xvec.\ [\![(\Psi', S, T) \in Rel'; xvec \sharp* \Psi']\!] \Longrightarrow (\Psi', (\!|\nu*xvec|\!) S, (\!|\nu*xvec|\!) T) \in Rel'$
and *C3*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi', S, T) \in Rel; \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'', S, T) \in Rel$

shows $\Psi \rhd P \parallel R \rightsquigarrow[Rel'] Q \parallel R$
using *Eqvt'*
proof(*induct rule*: *simI*[*of* - - - - ()])
case(*cSim* α *QR*)
from ‹$bn\ \alpha \sharp* (P \parallel R)$› ‹$bn\ \alpha \sharp* (Q \parallel R)$›
have $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp* R$
by *simp+*
from ‹$\Psi \rhd Q \parallel R \longmapsto \alpha \prec QR$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* R$› ‹$bn\ \alpha \sharp* subject\ \alpha$›
show *?case*
proof(*induct rule*: *parCases*[**where** $C = (P, R)$])
case(*cPar1* $Q'\ A_R\ \Psi_R$)
from ‹$A_R \sharp* (P, R)$› **have** $A_R \sharp* P$ **by** *simp*
have *FrR*: $extractFrame\ R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹$A_R \sharp* \alpha$› ‹$bn\ \alpha \sharp* R$› *FrR*
have $bn\ \alpha \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› **have** $\Psi \otimes \Psi_R \rhd P \rightsquigarrow[Rel]\ Q$
by(*blast intro*: *Sim PRelQ*)
moreover have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto \alpha \prec Q'$ **by** *fact*
ultimately obtain P' **where** *PTrans*: $\Psi \otimes \Psi_R \rhd P \longmapsto \alpha \prec P'$
and $P'RelQ'$: $(\Psi \otimes \Psi_R, P', Q') \in Rel$
using ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_R$› ‹$bn\ \alpha \sharp* P$›
by(*force dest*: *simE*)
from *PTrans QTrans* ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* \alpha$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$› **have** $A_R \sharp* P'$ **and** $A_R \sharp* Q'$
by(*blast dest*: *freeFreshChainDerivative*)+
from *PTrans* ‹$bn\ \alpha \sharp* R$› *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* \alpha$› **have** $\Psi \rhd P \parallel R \longmapsto \alpha \prec (P' \parallel R)$
by(*rule-tac Par1*)
moreover from $P'RelQ'$ *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P'$› ‹$A_R \sharp* Q'$› **have** $(\Psi, P' \parallel R, Q' \parallel R) \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* $R'\ A_Q\ \Psi_Q$)
from ‹$A_Q \sharp* (P, R)$› **have** $A_Q \sharp* P$ **and** $A_Q \sharp* R$ **by** *simp+*
obtain $A_P\ \Psi_P$ **where** *FrP*: $extractFrame\ P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* (\Psi, A_Q, \Psi_Q, \alpha, R)$
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* A_Q$ **and** $A_P \sharp* \Psi_Q$ **and** $A_P \sharp* \alpha$ **and** $A_P \sharp* R$
by *simp+*

have *FrQ*: $extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹$A_Q \sharp* P$› *FrP* ‹$A_P \sharp* A_Q$› **have** $A_Q \sharp* \Psi_P$
by(*drule-tac extractFrameFreshChain*) *auto*

from *FrP FrQ* ‹*bn* α $\sharp*$ P› ‹*bn* α $\sharp*$ Q› ‹A_P $\sharp*$ α› ‹A_Q $\sharp*$ α›
have *bn* α $\sharp*$ Ψ_P **and** *bn* α $\sharp*$ Ψ_Q
by(*force dest*: *extractFrameFreshChain*)+

obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **and** A_R $\sharp*$ (Ψ, P, Q, A_Q, A_P, Ψ_Q, Ψ_P, α, R) **and** *distinct* A_R
by(*rule freshFrame*)
then have A_R $\sharp*$ Ψ **and** A_R $\sharp*$ P **and** A_R $\sharp*$ Q **and** A_R $\sharp*$ A_Q **and** A_R $\sharp*$ A_P **and** A_R $\sharp*$ Ψ_Q **and** A_R $\sharp*$ Ψ_P **and** A_R $\sharp*$ α **and** A_R $\sharp*$ R
by *simp*+

from ‹A_Q $\sharp*$ R› *FrR* ‹A_R $\sharp*$ A_Q› **have** A_Q $\sharp*$ Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹A_P $\sharp*$ R› ‹A_R $\sharp*$ A_P› *FrR* **have** A_P $\sharp*$ Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*

have *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto\alpha \prec R'$ **by** *fact*
moreover have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
proof −
have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym FrameStatEqSym*)
moreover from *FrR* ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ P› ‹A_R $\sharp*$ Q›
have (*insertAssertion* (*extractFrame* Q) ($\Psi \otimes \Psi_R$)) $\hookrightarrow_F$ (*insertAssertion* (*extractFrame* P) ($\Psi \otimes \Psi_R$))
by(*blast intro*: *PRelQ StatImp*)
with *FrP FrQ* ‹A_P $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ_R› ‹A_Q $\sharp*$ Ψ_R›
have $\langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes \Psi_R) \otimes \Psi_P\rangle$ **using** *freshCompChain* **by** *auto*
moreover have $\langle A_P, (\Psi \otimes \Psi_R) \otimes \Psi_P\rangle \simeq_F \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym frameIntAssociativity*[*THEN FrameStatEqSym*])
ultimately show *?thesis*
by(*rule FrameStatEqImpCompose*)
qed

ultimately have $\Psi \otimes \Psi_P \rhd R \longmapsto\alpha \prec R'$
using ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ_Q› ‹A_Q $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ_P› ‹A_P $\sharp*$ R› ‹A_Q $\sharp*$ R› ‹A_P $\sharp*$ α› ‹A_Q $\sharp*$ α›
‹A_R $\sharp*$ A_P› ‹A_R $\sharp*$ A_Q› ‹A_R $\sharp*$ Ψ_P› ‹A_R $\sharp*$ Ψ_Q› ‹A_R $\sharp*$ Ψ› *FrR* ‹*distinct* A_R›
by(*force intro*: *transferFrame*)
with ‹*bn* α $\sharp*$ P› ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ R› ‹A_P $\sharp*$ α› *FrP* **have** $\Psi \rhd P \parallel R \longmapsto\alpha \prec (P \parallel R')$
by(*force intro*: *Par2*)
moreover obtain $A_R{}'$ $\Psi_R{}'$ **where** *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$ **and** $A_R{}'$ $\sharp*$ Ψ **and** $A_R{}'$ $\sharp*$ P **and** $A_R{}'$ $\sharp*$ Q

by(*rule-tac freshFrame*[**where** C=(Ψ, P, Q)]) *auto*

moreover from *RTrans FrR* ‹*distinct* A_R› ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ P› ‹A_R $\sharp*$ Q› ‹A_R $\sharp*$ R› ‹A_R $\sharp*$ α› ‹*bn* α $\sharp*$ Ψ› ‹*bn* α $\sharp*$ P› ‹*bn* α $\sharp*$ Q› ‹*bn* α $\sharp*$ R› ‹*bn* α $\sharp*$ *subject* α› ‹*distinct*(*bn* α)›
obtain p Ψ' $A_R{}'$ $\Psi_R{}'$ **where** S: *set* $p \subseteq$ *set*(*bn* α) $\times$ *set*(*bn*($p \cdot \alpha$)) **and** ($p \cdot \Psi_R$) $\otimes$ $\Psi' \simeq \Psi_R{}'$ **and** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$
and *bn*($p \cdot \alpha$) $\sharp*$ R **and** *bn*($p \cdot \alpha$) $\sharp*$ Ψ **and** *bn*($p \cdot \alpha$) $\sharp*$ P **and** *bn*($p \cdot \alpha$) $\sharp*$ Q **and** *bn*($p \cdot \alpha$) $\sharp*$ R
and $A_R{}'$ $\sharp*$ Ψ **and** $A_R{}'$ $\sharp*$ P **and** $A_R{}'$ $\sharp*$ Q
by(*rule-tac* C=(Ψ, P, Q, R) **and** C'=(Ψ, P, Q, R) **in** *expandFrame*) (*assumption* | *simp*)+

from ‹A_R $\sharp*$ Ψ› **have** ($p \cdot A_R$) $\sharp*$ ($p \cdot \Psi$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*bn* α $\sharp*$ Ψ› ‹*bn*($p \cdot \alpha$) $\sharp*$ Ψ› S **have** ($p \cdot A_R$) $\sharp*$ Ψ **by** *simp*
from ‹A_R $\sharp*$ P› **have** ($p \cdot A_R$) $\sharp*$ ($p \cdot P$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*bn* α $\sharp*$ P› ‹*bn*($p \cdot \alpha$) $\sharp*$ P› S **have** ($p \cdot A_R$) $\sharp*$ P **by** *simp*
from ‹A_R $\sharp*$ Q› **have** ($p \cdot A_R$) $\sharp*$ ($p \cdot Q$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*bn* α $\sharp*$ Q› ‹*bn*($p \cdot \alpha$) $\sharp*$ Q› S **have** ($p \cdot A_R$) $\sharp*$ Q **by** *simp*

from *FrR* **have** ($p \cdot$ *extractFrame* R) = $p \cdot \langle A_R, \Psi_R\rangle$ **by** *simp*
with ‹*bn* α $\sharp*$ R› ‹*bn*($p \cdot \alpha$) $\sharp*$ R› S **have** *extractFrame* $R = \langle(p \cdot A_R), (p \cdot \Psi_R)\rangle$
by(*simp add*: *eqvts*)

with ‹($p \cdot A_R$) $\sharp*$ Ψ› ‹($p \cdot A_R$) $\sharp*$ P› ‹($p \cdot A_R$) $\sharp*$ Q› **have** ($\Psi \otimes (p \cdot \Psi_R)$, P, Q) $\in$ *Rel* **by**(*rule-tac PRelQ*)

hence (($\Psi \otimes (p \cdot \Psi_R)$) $\otimes$ Ψ', P, Q) $\in$ *Rel* **by**(*rule Ext*)
with ‹($p \cdot \Psi_R$) $\otimes$ $\Psi' \simeq \Psi_R{}'$› **have** ($\Psi \otimes \Psi_R{}'$, P, Q) $\in$ *Rel* **by**(*blast intro*: *C3 Associativity compositionSym*)
with *FrR'* ‹$A_R{}'$ $\sharp*$ Ψ› ‹$A_R{}'$ $\sharp*$ P› ‹$A_R{}'$ $\sharp*$ Q› **have** (Ψ, $P \parallel R'$, $Q \parallel R'$) $\in$ *Rel'* **by**(*rule-tac C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_R M N Q' A_Q Ψ_Q K *xvec* R' A_R)
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹A_Q $\sharp*$ (P, R)› **have** A_Q $\sharp*$ P **and** A_Q $\sharp*$ R **by** *simp*+

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹A_R $\sharp*$ (P, R)› **have** A_R $\sharp*$ P **and** A_R $\sharp*$ R **by** *simp*+

from ‹*xvec* $\sharp*$ (P, R)› **have** *xvec* $\sharp*$ P **and** *xvec* $\sharp*$ R **by** *simp*+

obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** A_P $\sharp*$ (Ψ, A_Q, Ψ_Q, A_R, M, N, K, R, P, *xvec*) **and** *distinct* A_P

by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* A_Q$ **and** $A_P \sharp* \Psi_Q$ **and** $A_P \sharp* M$ **and** $A_P \sharp* R$ **and** $A_P \sharp* N$ **and** $A_P \sharp* K$ **and** $A_P \sharp* A_R$ **and** $A_P \sharp* P$ **and** $A_P \sharp* xvec$
by *simp*+

have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto M(\!|N|\!) \prec Q'$ **and** *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$ **by** *fact*+

from *FrP FrR* ‹$A_Q \sharp* P$› ‹$A_P \sharp* R$› ‹$A_R \sharp* P$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* A_R$› ‹$A_P \sharp* xvec$› ‹$xvec \sharp* P$›
have $A_P \sharp* \Psi_R$ **and** $A_Q \sharp* \Psi_P$ **and** $A_R \sharp* \Psi_P$ **and** $xvec \sharp* \Psi_P$
by(*fastforce dest*!: *extractFrameFreshChain*)+

from *RTrans FrR* ‹*distinct* A_R› ‹$A_R \sharp* R$› ‹$A_R \sharp* xvec$› ‹$xvec \sharp* R$› ‹$xvec \sharp* Q$› ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* \Psi_Q$› ‹$A_R \sharp* Q$›
‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_Q$› ‹$xvec \sharp* K$› ‹$A_R \sharp* K$› ‹$A_R \sharp* N$› ‹$A_R \sharp* R$› ‹$xvec \sharp* R$› ‹$A_R \sharp* P$› ‹$xvec \sharp* P$› ‹$A_P \sharp* A_R$› ‹$A_P \sharp* xvec$›
‹$A_Q \sharp* A_R$› ‹$A_Q \sharp* xvec$› ‹$A_R \sharp* \Psi_P$› ‹$xvec \sharp* \Psi_P$› ‹*distinct xvec*› ‹$xvec \sharp* M$›
obtain p Ψ' $A_R{}'$ $\Psi_R{}'$ **where** S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$ **and** FrR': $extractFrame\ R' = \langle A_R{}', \Psi_R{}'\rangle$
and $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}' \sharp* Q$ **and** $A_R{}' \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* \Psi$
and $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi_Q$ **and** $(p \cdot xvec) \sharp* K$ **and** $(p \cdot xvec) \sharp* R$
and $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* A_P$ **and** $(p \cdot xvec) \sharp* A_Q$ **and** $(p \cdot xvec) \sharp* \Psi_P$
and $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* N$
by(*rule-tac* C=(Ψ, Q, Ψ_Q, K, R, P, A_P, A_Q, Ψ_P) **and** C'=(Ψ, Q, Ψ_Q, K, R, P, A_P, A_Q, Ψ_P) **in** *expandFrame*)
(*assumption* | *simp*)+

from ‹$A_R \sharp* \Psi$› **have** $(p \cdot A_R) \sharp* (p \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S **have** $(p \cdot A_R) \sharp* \Psi$ **by** *simp*
from ‹$A_R \sharp* P$› **have** $(p \cdot A_R) \sharp* (p \cdot P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* P$› ‹$(p \cdot xvec) \sharp* P$› S **have** $(p \cdot A_R) \sharp* P$ **by** *simp*
from ‹$A_R \sharp* Q$› **have** $(p \cdot A_R) \sharp* (p \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* Q$› ‹$(p \cdot xvec) \sharp* Q$› S **have** $(p \cdot A_R) \sharp* Q$ **by** *simp*
from ‹$A_R \sharp* R$› **have** $(p \cdot A_R) \sharp* (p \cdot R)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* R$› ‹$(p \cdot xvec) \sharp* R$› S **have** $(p \cdot A_R) \sharp* R$ **by** *simp*
from ‹$A_R \sharp* K$› **have** $(p \cdot A_R) \sharp* (p \cdot K)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* K$› ‹$(p \cdot xvec) \sharp* K$› S **have** $(p \cdot A_R) \sharp* K$ **by** *simp*

from ‹A_P ♯* *xvec*› ‹(p · *xvec*) ♯* A_P› ‹A_P ♯* M› S **have** A_P ♯* (p · M) **by**(*simp add*: *freshChainSimps*)
from ‹A_Q ♯* *xvec*› ‹(p · *xvec*) ♯* A_Q› ‹A_Q ♯* M› S **have** A_Q ♯* (p · M) **by**(*simp add*: *freshChainSimps*)
from ‹A_P ♯* *xvec*› ‹(p · *xvec*) ♯* A_P› ‹A_P ♯* A_R› S **have** (p · A_R) ♯* A_P **by**(*simp add*: *freshChainSimps*)
from ‹A_Q ♯* *xvec*› ‹(p · *xvec*) ♯* A_Q› ‹A_Q ♯* A_R› S **have** (p · A_R) ♯* A_Q **by**(*simp add*: *freshChainSimps*)

from *QTrans* S ‹*xvec* ♯* Q› ‹(p · *xvec*) ♯* Q› **have** (p · (Ψ ⊗ Ψ_R)) ⊳ Q ⟼ (p · M)⦇N⦈ ≺ Q'
by(*rule-tac inputPermFrameSubject*) (*assumption* | *auto simp add*: *fresh-star-def*)+
with ‹*xvec* ♯* Ψ› ‹(p · *xvec*) ♯* Ψ› S **have** *QTrans*: (Ψ ⊗ (p · Ψ_R)) ⊳ Q ⟼ (p · M)⦇N⦈ ≺ Q'
by(*simp add*: *eqvts*)

from *FrR* **have** (p · *extractFrame* R) = p · ⟨A_R, Ψ_R⟩ **by** *simp*
with ‹*xvec* ♯* R› ‹(p · *xvec*) ♯* R› S **have** *FrR*: *extractFrame* R = ⟨(p · A_R), (p · Ψ_R)⟩
by(*simp add*: *eqvts*)

note *RTrans FrR*
moreover from *FrR* ‹(p · A_R) ♯* Ψ› ‹(p · A_R) ♯* P› ‹(p · A_R) ♯* Q› **have** Ψ ⊗ (p · Ψ_R) ⊳ P ⇝[*Rel*] Q
by(*metis Sim PRelQ*)
with *QTrans* **obtain** P' **where** *PTrans*: Ψ ⊗ (p · Ψ_R) ⊳ P ⟼(p · M)⦇N⦈ ≺ P' **and** $P'RelQ'$: (Ψ ⊗ (p · Ψ_R), P', Q') ∈ *Rel*
by(*force dest*: *simE*)
from *PTrans QTrans* ‹A_R' ♯* P› ‹A_R' ♯* Q› ‹A_R' ♯* N› **have** A_R' ♯* P' **and** A_R' ♯* Q'
by(*blast dest*: *inputFreshChainDerivative*)+

note *PTrans*
moreover from *MeqK* **have** (p · (Ψ ⊗ Ψ_Q ⊗ Ψ_R)) ⊢ (p · M) ↔ (p · K) **by**(*rule chanEqClosed*)
with ‹*xvec* ♯* Ψ› ‹(p · *xvec*) ♯* Ψ› ‹*xvec* ♯* Ψ_Q› ‹(p · *xvec*) ♯* Ψ_Q› ‹*xvec* ♯* K› ‹(p · *xvec*) ♯* K› S
have *MeqK*: Ψ ⊗ Ψ_Q ⊗ (p · Ψ_R) ⊢ (p · M) ↔ K **by**(*simp add*: *eqvts*)

moreover have ⟨A_Q, (Ψ ⊗ Ψ_Q) ⊗ (p · Ψ_R)⟩ ↪$_F$ ⟨A_P, (Ψ ⊗ Ψ_P) ⊗ (p · Ψ_R)⟩
proof −
have ⟨A_P, (Ψ ⊗ (p · Ψ_R)) ⊗ Ψ_P⟩ ≃$_F$ ⟨A_P, (Ψ ⊗ Ψ_P) ⊗ (p · Ψ_R)⟩
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover from *FrR* ‹(p · A_R) ♯* Ψ› ‹(p · A_R) ♯* P› ‹(p · A_R) ♯* Q›
have (*insertAssertion* (*extractFrame* Q) (Ψ ⊗ (p · Ψ_R))) ↪$_F$ (*insertAssertion* (*extractFrame* P) (Ψ ⊗ (p · Ψ_R)))
by(*metis PRelQ StatImp*)
with *FrP FrQ* ‹A_P ♯* Ψ› ‹A_Q ♯* Ψ› ‹A_P ♯* Ψ_R› ‹A_Q ♯* Ψ_R› ‹A_P ♯* *xvec*› ‹(p

$\cdot$ *xvec*) $\sharp*$ A_P› ‹A_Q $\sharp*$ *xvec*› ‹(p $\cdot$ *xvec*) $\sharp*$ A_Q› S
have $\langle A_Q, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_P\rangle$ **using** *freshCompChain*
by(*simp add*: *freshChainSimps*)
moreover have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes (p \cdot \Psi_R)\rangle \simeq_F \langle A_Q, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_Q\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
ultimately show *?thesis* **by**(*rule-tac FrameStatEqImpCompose*)
qed
moreover note *FrP FrQ* ‹*distinct* A_P›
moreover from ‹*distinct* A_R› **have** *distinct*(p $\cdot$ A_R) **by** *simp*
moreover note ‹(p $\cdot$ A_R) $\sharp*$ A_P› ‹(p $\cdot$ A_R) $\sharp*$ A_Q› ‹(p $\cdot$ A_R) $\sharp*$ Ψ› ‹(p $\cdot$ A_R) $\sharp*$ P› ‹(p $\cdot$ A_R) $\sharp*$ Q› ‹(p $\cdot$ A_R) $\sharp*$ R› ‹(p $\cdot$ A_R) $\sharp*$ K›
‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ R› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ (p $\cdot$ M)› ‹A_Q $\sharp*$ R› ‹A_Q $\sharp*$ (p $\cdot$ M)› ‹A_P $\sharp*$ *xvec*› ‹*xvec* $\sharp*$ P› ‹A_P $\sharp*$ R›
ultimately obtain K' **where** $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$ **and** $\Psi \otimes \Psi_P \otimes (p \cdot \Psi_R) \vdash (p \cdot M) \leftrightarrow K'$ **and** $(p \cdot A_R) \sharp* K'$
by(*rule-tac comm1Aux*)

with *PTrans FrP* **have** $\Psi \rhd P \parallel R \longmapsto\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel R')$ **using** *FrR* ‹(p $\cdot$ A_R) $\sharp*$ Ψ› ‹(p $\cdot$ A_R) $\sharp*$ P› ‹(p $\cdot$ A_R) $\sharp*$ R›
‹*xvec* $\sharp*$ P› ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ R› ‹A_P $\sharp*$ (p $\cdot$ M)› ‹(p $\cdot$ A_R) $\sharp*$ K'› ‹(p $\cdot$ A_R) $\sharp*$ A_P›
by(*rule-tac Comm1*) (*assumption* | *simp*)+

moreover from $P'RelQ'$ **have** $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P', Q') \in Rel$ **by**(*rule Ext*)
with ‹(p $\cdot$ Ψ_R) $\otimes$ $\Psi' \simeq \Psi_R{}'$› **have** $(\Psi \otimes \Psi_R{}', P', Q') \in Rel$ **by**(*metis C3 Associativity compositionSym*)
with FrR' ‹$A_R{}'$ $\sharp*$ P'› ‹$A_R{}'$ $\sharp*$ Q'› ‹$A_R{}'$ $\sharp*$ Ψ› **have** $(\Psi, P' \parallel R', Q' \parallel R') \in Rel'$ **by**(*rule-tac C1*)
with ‹*xvec* $\sharp*$ Ψ› **have** $(\Psi, (\!|\nu{*}xvec|\!)(P' \parallel R'), (\!|\nu{*}xvec|\!)(Q' \parallel R')) \in Rel'$ **by**(*rule-tac C2*)
ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_R M *xvec* N Q' A_Q Ψ_Q K R' A_R)
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹A_Q $\sharp*$ (P, R)› **have** A_Q $\sharp*$ P **and** A_Q $\sharp*$ R **by** *simp*+

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹A_R $\sharp*$ (P, R)› **have** A_R $\sharp*$ P **and** A_R $\sharp*$ R **by** *simp*+

from ‹*xvec* $\sharp*$ (P, R)› **have** *xvec* $\sharp*$ P **and** *xvec* $\sharp*$ R **by** *simp*+

obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** A_P $\sharp*$ (Ψ, A_Q, Ψ_Q, A_R, M, N, K, R, P, *xvec*) **and** *distinct* A_P
by(*rule freshFrame*)
hence A_P $\sharp*$ Ψ **and** A_P $\sharp*$ A_Q **and** A_P $\sharp*$ Ψ_Q **and** A_P $\sharp*$ M **and** A_P $\sharp*$ R **and** A_P $\sharp*$ N A_P $\sharp*$ K **and** A_P $\sharp*$ A_R **and** A_P $\sharp*$ P **and** A_P $\sharp*$ *xvec*

by *simp+*

from *FrP FrR* ‹A_Q ♯* P› ‹A_P ♯* R› ‹A_R ♯* P› ‹A_P ♯* A_Q› ‹A_P ♯* A_R› ‹A_P ♯* *xvec*› ‹*xvec* ♯* P›
have A_P ♯* Ψ_R **and** A_Q ♯* Ψ_P **and** A_R ♯* Ψ_P **and** *xvec* ♯* Ψ_P
by(*fastforce dest*!: *extractFrameFreshChain*)+

have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$ **by** *fact*

note ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$› *FrR* ‹$\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$›
moreover from *FrR* ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* Q› **have** $\Psi \otimes \Psi_R \rhd P \rightsquigarrow[Rel]\ Q$ **by**(*metis PRelQ Sim*)
with *QTrans* **obtain** P' **where** *PTrans*: $\Psi \otimes \Psi_R \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **and** $P'RelQ'$: $(\Psi \otimes \Psi_R,\ P',\ Q') \in Rel$
using ‹*xvec* ♯* Ψ› ‹*xvec* ♯* Ψ_R› ‹*xvec* ♯* P›
by(*force dest*: *simE*)
from *PTrans QTrans* ‹A_R ♯* P› ‹A_R ♯* Q› ‹A_R ♯* *xvec*› ‹*xvec* ♯* M› ‹*distinct xvec*› **have** A_R ♯* P' **and** A_R ♯* Q'
by(*blast dest*: *outputFreshChainDerivative*)+
note *PTrans* ‹$\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$›
moreover have $\langle A_Q,\ (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P,\ (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
proof −
have $\langle A_P,\ (\Psi \otimes \Psi_R) \otimes \Psi_P\rangle \simeq_F \langle A_P,\ (\Psi \otimes \Psi_P) \otimes \Psi_R\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover from *FrR* ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* Q›
have (*insertAssertion* (*extractFrame* Q) $(\Psi \otimes \Psi_R)$) $\hookrightarrow_F$ (*insertAssertion* (*extractFrame* P) $(\Psi \otimes \Psi_R)$)
by(*metis PRelQ StatImp*)
with *FrP FrQ* ‹A_P ♯* Ψ› ‹A_Q ♯* Ψ› ‹A_P ♯* Ψ_R› ‹A_Q ♯* Ψ_R›
have $\langle A_Q,\ (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P,\ (\Psi \otimes \Psi_R) \otimes \Psi_P\rangle$ **using** *freshCompChain* **by** *simp*
moreover have $\langle A_Q,\ (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q,\ (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
ultimately show *?thesis* **by**(*rule-tac FrameStatEqImpCompose*)
qed
moreover note *FrP FrQ* ‹*distinct* A_P› ‹*distinct* A_R›
moreover from ‹A_P ♯* A_R› ‹A_Q ♯* A_R› **have** A_R ♯* A_P **and** A_R ♯* A_Q **by** *simp+*
moreover note ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* Q› ‹A_R ♯* R› ‹A_R ♯* K› ‹A_P ♯* Ψ› ‹A_P ♯* P›
‹A_P ♯* R› ‹A_P ♯* M› ‹A_Q ♯* R› ‹A_Q ♯* M› ‹A_R ♯* *xvec*› ‹*xvec* ♯* M›
ultimately obtain K' **where** $\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$ **and** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** A_R ♯* K'
by(*rule-tac comm2Aux*) *assumption+*

with *PTrans FrP* **have** $\Psi \rhd P \parallel R \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(P' \parallel R')$ **using** *FrR* ‹A_R

$\sharp*\ \Psi$› ‹$A_R\ \sharp*\ P$› ‹$A_R\ \sharp*\ R$›
‹$A_R\ \sharp*\ \Psi$› ‹$A_R\ \sharp*\ P$› ‹$A_R\ \sharp*\ R$› **and** ‹$xvec\ \sharp*\ R$› ‹$A_P\ \sharp*\ \Psi$› ‹$A_P\ \sharp*\ P$› ‹$A_P\ \sharp*\ R$› ‹$A_P\ \sharp*\ A_R$› ‹$A_P\ \sharp*\ M$› ‹$A_R\ \sharp*\ K'$›
by(*force intro*: *Comm2*)

moreover from ‹$\Psi \otimes \Psi_P \rhd R \longmapsto K'(\!|N|\!) \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R\ \sharp*\ \Psi$› ‹$A_R\ \sharp*\ R$› ‹$A_R\ \sharp*\ P'$› ‹$A_R\ \sharp*\ Q'$› ‹$A_R\ \sharp*\ N$› ‹$A_R\ \sharp*\ K'$›
obtain $\Psi'\ A_R'\ \Psi_R'$ **where** *ReqR'*: $\Psi_R \otimes \Psi' \simeq \Psi_R'$ **and** *FrR'*: *extractFrame* $R' = \langle A_R', \Psi_R' \rangle$
and $A_R'\ \sharp*\ \Psi$ **and** $A_R'\ \sharp*\ P'$ **and** $A_R'\ \sharp*\ Q'$
by(*rule-tac* C=(Ψ, P', Q') **and** C'=Ψ **in** *expandFrame*) *auto*

from *P'RelQ'* **have** $((\Psi \otimes \Psi_R) \otimes \Psi', P', Q') \in Rel$ **by**(*rule Ext*)
with *ReqR'* **have** $(\Psi \otimes \Psi_R', P', Q') \in Rel$ **by**(*metis C3 Associativity compositionSym*)
with *FrR'* ‹$A_R'\ \sharp*\ P'$› ‹$A_R'\ \sharp*\ Q'$› ‹$A_R'\ \sharp*\ \Psi$› **have** $(\Psi, P' \parallel R', Q' \parallel R') \in Rel'$
by(*rule-tac C1*)
with ‹$xvec\ \sharp*\ \Psi$› **have** $(\Psi, (\!|\nu{*}xvec|\!)(P' \parallel R'), (\!|\nu{*}xvec|\!)(Q' \parallel R')) \in Rel'$ **by**(*rule-tac C2*)
ultimately show *?case* **by** *blast*
qed
qed
unbundle *no relcomp-syntax*
lemma *bangPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and *Rel'* :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in Rel$
and *eqvt Rel'*
and *guarded P*
and *guarded Q*
and *cSim*: $\bigwedge \Psi'\ S\ T.\ (\Psi', S, T) \in Rel \Longrightarrow \Psi' \rhd S \rightsquigarrow [Rel]\ T$
and *cExt*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ (\Psi', S, T) \in Rel \Longrightarrow (\Psi' \otimes \Psi'', S, T) \in Rel$
and *cSym*: $\bigwedge \Psi'\ S\ T.\ (\Psi', S, T) \in Rel \Longrightarrow (\Psi', T, S) \in Rel$
and *StatEq*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi', S, T) \in Rel; \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'', S, T) \in Rel$
and *Closed*: $\bigwedge \Psi'\ S\ T\ p.\ (\Psi', S, T) \in Rel \Longrightarrow ((p{::}name\ prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel$
and *Assoc*: $\bigwedge \Psi'\ S\ T\ U.\ (\Psi', S \parallel (T \parallel U), (S \parallel T) \parallel U) \in Rel$
and *ParPres*: $\bigwedge \Psi'\ S\ T\ U.\ (\Psi', S, T) \in Rel \Longrightarrow (\Psi', S \parallel U, T \parallel U) \in Rel$
and *FrameParPres*: $\bigwedge \Psi'\ \Psi_U\ S\ T\ U\ A_U.\ [\![(\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame\ U = \langle A_U, \Psi_U \rangle; A_U\ \sharp*\ \Psi'; A_U\ \sharp*\ S; A_U\ \sharp*\ T]\!] \Longrightarrow$
$(\Psi', U \parallel S, U \parallel T) \in Rel$
and *ResPres*: $\bigwedge \Psi'\ S\ T\ xvec.\ [\![(\Psi', S, T) \in Rel; xvec\ \sharp*\ \Psi']\!] \Longrightarrow (\Psi', (\!|\nu{*}xvec|\!)S,$

$(\!|\nu{*}xvec|\!)\, T) \in Rel$
and *ScopeExt*: $\bigwedge xvec\ \Psi'\ S\ T.\ [\![xvec \sharp{*}\ \Psi';\ xvec \sharp{*}\ T]\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)(S \parallel T),\ ((\!|\nu{*}xvec|\!)S) \parallel T) \in Rel$
and *Trans*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi',\ S,\ T) \in Rel;\ (\Psi',\ T,\ U) \in Rel]\!] \Longrightarrow (\Psi',\ S,\ U) \in Rel$
and *Compose*: $\bigwedge \Psi'\ S\ T\ U\ O.\ [\![(\Psi',\ S,\ T) \in Rel;\ (\Psi',\ T,\ U) \in Rel';\ (\Psi',\ U,\ O) \in Rel]\!] \Longrightarrow (\Psi',\ S,\ O) \in Rel'$
and *C1*: $\bigwedge \Psi\ S\ T\ U.\ [\![(\Psi,\ S,\ T) \in Rel;\ guarded\ S;\ guarded\ T]\!] \Longrightarrow (\Psi,\ U \parallel !S,\ U \parallel !T) \in Rel'$
and *Der*: $\bigwedge \Psi'\ S\ \alpha\ S'\ T.\ [\![\Psi' \rhd !S \longmapsto \alpha \prec S';\ (\Psi',\ S,\ T) \in Rel;\ bn\ \alpha \sharp{*}\ \Psi';\ bn\ \alpha \sharp{*}\ S;\ bn\ \alpha \sharp{*}\ T;\ guarded\ T;\ bn\ \alpha \sharp{*}\ subject\ \alpha]\!] \Longrightarrow$
$\exists\, T'\ U\ O.\ \ \Psi' \rhd !T \longmapsto \alpha \prec T' \wedge (\Psi',\ S',\ U \parallel !S) \in Rel \wedge (\Psi',\ T',\ O \parallel !T) \in Rel \wedge$
$(\Psi',\ U,\ O) \in Rel \wedge ((supp\ U)::name\ set) \subseteq supp\ S' \wedge$
$((supp\ O)::name\ set) \subseteq supp\ T'$

shows $\Psi \rhd R \parallel !P \rightsquigarrow [Rel']\ R \parallel !Q$
using ‹*eqvt Rel'*›
proof(*induct rule*: *simI*[*of* - - - - ()])
case(*cSim* α *RQ'*)
from ‹$bn\ \alpha \sharp{*}\ (R \parallel !P)$› ‹$bn\ \alpha \sharp{*}\ (R \parallel !Q)$› **have** $bn\ \alpha \sharp{*}\ P$ **and** $bn\ \alpha \sharp{*}\ (!Q)$ **and** $bn\ \alpha \sharp{*}\ Q$ **and** $bn\ \alpha \sharp{*}\ R$ **by** *simp+*
from ‹$\Psi \rhd R \parallel !Q \longmapsto \alpha \prec RQ'$› ‹$bn\ \alpha \sharp{*}\ \Psi$› ‹$bn\ \alpha \sharp{*}\ R$› ‹$bn\ \alpha \sharp{*}\ !Q$› ‹$bn\ \alpha \sharp{*}\ subject\ \alpha$› **show** *?case*
proof(*induct rule*: *parCases*[**where** *C=P*])
case(*cPar1* $R'\ A_Q\ \Psi_Q$)
from ‹$extractFrame\ (!Q) = \langle A_Q,\ \Psi_Q\rangle$› **have** $A_Q = []$ **and** $\Psi_Q = SBottom'$ **by** *simp+*
with ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto \alpha \prec R'$› ‹$bn\ \alpha \sharp{*}\ P$› **have** $\Psi \rhd R \parallel !P \longmapsto \alpha \prec (R' \parallel !P)$
by(*rule-tac Par1*) (*assumption* | *simp*)+
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› ‹*guarded P*› ‹*guarded Q*› **have** $(\Psi,\ R' \parallel !P,\ R' \parallel !Q) \in Rel'$
by(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* $Q'\ A_R\ \Psi_R$)
have *QTrans*: $\Psi \otimes \Psi_R \rhd !Q \longmapsto \alpha \prec Q'$ **and** *FrR*: $extractFrame\ R = \langle A_R,\ \Psi_R\rangle$ **by** *fact+*
with ‹$bn\ \alpha \sharp{*}\ R$› ‹$A_R \sharp{*}\ \alpha$› **have** $bn\ \alpha \sharp{*}\ \Psi_R$ **by**(*force dest*: *extractFrameFreshChain*)
with *QTrans* ‹$(\Psi,\ P,\ Q) \in Rel$› ‹$bn\ \alpha \sharp{*}\ \Psi$›‹$bn\ \alpha \sharp{*}\ P$› ‹$bn\ \alpha \sharp{*}\ Q$› ‹*guarded P*› ‹$bn\ \alpha \sharp{*}\ subject\ \alpha$›
obtain $P'\ S\ T$ **where** *PTrans*: $\Psi \otimes \Psi_R \rhd !P \longmapsto \alpha \prec P'$ **and** $(\Psi \otimes \Psi_R,\ P',\ T \parallel !P) \in Rel$
and $(\Psi \otimes \Psi_R,\ Q',\ S \parallel !Q) \in Rel$ **and** $(\Psi \otimes \Psi_R,\ S,\ T) \in Rel$
and *suppT*: $((supp\ T)::name\ set) \subseteq supp\ P'$ **and** *suppS*: $((supp\ S)::name\ set) \subseteq supp\ Q'$

by(*drule-tac cSym*) (*auto dest*: *Der cExt*)
from *PTrans FrR* ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* α› ‹*bn* α ♯* R› **have** $\Psi \rhd R \parallel !P \longmapsto \alpha \prec (R \parallel P')$
by(*rule-tac Par2*) *auto*
moreover
{
from ‹A_R ♯* P› ‹A_R ♯* $(!Q)$› ‹A_R ♯* α› *PTrans QTrans* ‹*bn* α ♯* *subject* α› ‹*distinct*(*bn* α)› **have** A_R ♯* P' **and** A_R ♯* Q'
by(*force dest*: *freeFreshChainDerivative*)+
from ‹$(\Psi \otimes \Psi_R, P', T \parallel !P) \in Rel$› *FrR* ‹$A_R$ ♯* Ψ› ‹A_R ♯* P'› ‹A_R ♯* P› *suppT* **have** $(\Psi, R \parallel P', R \parallel (T \parallel !P)) \in Rel$
by(*rule-tac FrameParPres*) (*auto simp add*: *fresh-star-def fresh-def psi.supp*)
hence $(\Psi, R \parallel P', (R \parallel T) \parallel !P) \in Rel$ **by**(*blast intro*: *Assoc Trans*)
moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded* P› ‹*guarded* Q› **have** $(\Psi, (R \parallel T) \parallel !P, (R \parallel T) \parallel !Q) \in Rel'$
by(*rule C1*)
moreover from ‹$(\Psi \otimes \Psi_R, Q', S \parallel !Q) \in Rel$› ‹$(\Psi \otimes \Psi_R, S, T) \in Rel$› **have** $(\Psi \otimes \Psi_R, Q', T \parallel !Q) \in Rel$
by(*blast intro*: *ParPres Trans*)
with *FrR* ‹A_R ♯* Ψ› ‹A_R ♯* P'› ‹A_R ♯* Q'› ‹A_R ♯* $(!Q)$› *suppT suppS* **have** $(\Psi, R \parallel Q', R \parallel (T \parallel !Q)) \in Rel$
by(*rule-tac FrameParPres*) (*auto simp add*: *fresh-star-def fresh-def psi.supp*)
hence $(\Psi, R \parallel Q', (R \parallel T) \parallel !Q) \in Rel$ **by**(*blast intro*: *Assoc Trans*)
ultimately have $(\Psi, R \parallel P', R \parallel Q') \in Rel'$ **by**(*blast intro*: *cSym Compose*)
}
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_Q M N R' A_R Ψ_R K *xvec* Q' A_Q)
from ‹*extractFrame* $(!Q) = \langle A_Q, \Psi_Q \rangle$› **have** $A_Q = []$ **and** $\Psi_Q = SBottom'$ **by** *simp*+
have *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$ **and** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R \rangle$ **by** *fact*+
moreover have *QTrans*: $\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N \rangle \prec Q'$ **by** *fact*
from *FrR* ‹*xvec* ♯* R› ‹A_R ♯* *xvec*› **have** *xvec* ♯* Ψ_R **by**(*force dest*: *extractFrameFreshChain*)
with *QTrans* ‹$(\Psi, P, Q) \in Rel$› ‹*xvec* ♯* Ψ›‹*xvec* ♯* P› ‹*xvec* ♯* $(!Q)$› ‹*guarded* P› ‹*xvec* ♯* K›
obtain P' S T **where** *PTrans*: $\Psi \otimes \Psi_R \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N \rangle \prec P'$ **and** $(\Psi \otimes \Psi_R, P', T \parallel !P) \in Rel$
and $(\Psi \otimes \Psi_R, Q', S \parallel !Q) \in Rel$ **and** $(\Psi \otimes \Psi_R, S, T) \in Rel$
and *suppT*: ((*supp* T)::*name set*) $\subseteq$ *supp* P' **and** *suppS*: ((*supp* S)::*name set*) $\subseteq$ *supp* Q'
by(*drule-tac cSym*) (*fastforce dest*: *Der intro*: *cExt*)
note ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$›
ultimately have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(R' \parallel P')$
using *PTrans* ‹$\Psi_Q = SBottom'$› ‹*xvec* ♯* R› ‹A_R ♯* Ψ› ‹A_R ♯* R› ‹A_R ♯* M› ‹A_R ♯* P›
by(*rule-tac Comm1*) (*assumption* | *simp*)+

moreover from ‹A_R ♯* P› ‹A_R ♯* $(!Q)$› ‹A_R ♯* *xvec*› *PTrans QTrans* ‹*xvec* ♯* K› ‹*distinct xvec*›
have A_R ♯* P' **and** A_R ♯* Q' **by**(*force dest*: *outputFreshChainDerivative*)+
moreover with *RTrans FrR* ‹*distinct* A_R› ‹A_R ♯* R› ‹A_R ♯* N› ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* $(!Q)$› ‹A_R ♯* M›
obtain Ψ' A_R' Ψ_R' **where** *FrR'*: *extractFrame* $R' = \langle A_R', \Psi_R' \rangle$ **and** $\Psi_R \otimes \Psi' \simeq \Psi_R'$ **and** A_R' ♯* Ψ
and A_R' ♯* P' **and** A_R' ♯* Q' **and** A_R' ♯* P **and** A_R' ♯* Q
by(*rule-tac* C=(Ψ, P, P', Q, Q') **and** C'=Ψ **in** *expandFrame*) *auto*

moreover
{
from ‹$(\Psi \otimes \Psi_R, P', T \parallel !P) \in Rel$› **have** $((\Psi \otimes \Psi_R) \otimes \Psi', P', T \parallel !P) \in Rel$ **by**(*rule cExt*)
with ‹$\Psi_R \otimes \Psi' \simeq \Psi_R'$› **have** $(\Psi \otimes \Psi_R', P', T \parallel !P) \in Rel$
by(*metis Associativity StatEq compositionSym*)
with *FrR'* ‹A_R' ♯* Ψ› ‹A_R' ♯* P'› ‹A_R' ♯* P› *suppT* **have** $(\Psi, R' \parallel P', R' \parallel (T \parallel !P)) \in Rel$
by(*rule-tac FrameParPres*) (*auto simp add*: *fresh-star-def fresh-def psi.supp*)
hence $(\Psi, R' \parallel P', (R' \parallel T) \parallel !P) \in Rel$ **by**(*blast intro*: *Assoc Trans*)
with ‹*xvec* ♯* Ψ› ‹*xvec* ♯* P› **have** $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel P'), ((\!|\nu{*}xvec|\!)(R' \parallel T)) \parallel !P) \in Rel$
by(*metis ResPres psiFreshVec ScopeExt Trans*)
moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded* P› ‹*guarded* Q› **have** $(\Psi, ((\!|\nu{*}xvec|\!)(R' \parallel T)) \parallel !P, ((\!|\nu{*}xvec|\!)(R' \parallel T)) \parallel !Q) \in Rel'$
by(*rule C1*)
moreover from ‹$(\Psi \otimes \Psi_R, Q', S \parallel !Q) \in Rel$› ‹$(\Psi \otimes \Psi_R, S, T) \in Rel$› **have** $(\Psi \otimes \Psi_R, Q', T \parallel !Q) \in Rel$
by(*blast intro*: *ParPres Trans*)
hence $((\Psi \otimes \Psi_R) \otimes \Psi', Q', T \parallel !Q) \in Rel$ **by**(*rule cExt*)
with ‹$\Psi_R \otimes \Psi' \simeq \Psi_R'$› **have** $(\Psi \otimes \Psi_R', Q', T \parallel !Q) \in Rel$
by(*metis Associativity StatEq compositionSym*)
with *FrR'* ‹A_R' ♯* Ψ› ‹A_R' ♯* P'› ‹A_R' ♯* Q'› ‹A_R' ♯* Q› *suppT suppS* **have** $(\Psi, R' \parallel Q', R' \parallel (T \parallel !Q)) \in Rel$
by(*rule-tac FrameParPres*) (*auto simp add*: *fresh-star-def fresh-def psi.supp*)
hence $(\Psi, R' \parallel Q', (R' \parallel T) \parallel !Q) \in Rel$ **by**(*blast intro*: *Assoc Trans*)
with ‹*xvec* ♯* Ψ› ‹*xvec* ♯* $(!Q)$› **have** $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel Q'), ((\!|\nu{*}xvec|\!)(R' \parallel T)) \parallel !Q) \in Rel$
by(*metis ResPres psiFreshVec ScopeExt Trans*)
ultimately have $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel P'), (\!|\nu{*}xvec|\!)(R' \parallel Q')) \in Rel'$ **by**(*blast intro*: *cSym Compose*)
}
ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_Q M *xvec* N R' A_R Ψ_R K Q' A_Q)
from ‹*extractFrame* $(!Q) = \langle A_Q, \Psi_Q \rangle$› **have** $A_Q = []$ **and** $\Psi_Q = SBottom'$ **by** *simp*+
have *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu{*}xvec|\!)\langle N \rangle \prec R'$ **and** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R \rangle$ **by** *fact*+

then obtain p Ψ' $A_R{}'$ $\Psi_R{}'$ **where** S: *set* $p \subseteq$ *set xvec* $\times$ *set*$(p \cdot xvec)$
and FrR': *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$ **and** $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}' \sharp* \Psi$
and $A_R{}' \sharp* N$ **and** $A_R{}' \sharp* R'$ **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi$
and $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $xvec \sharp* A_R{}'$ **and** $(p \cdot xvec) \sharp* A_R{}'$
and *distinctPerm* p **and** $(p \cdot xvec) \sharp* R'$ **and** $(p \cdot xvec) \sharp* N$
using ‹*distinct* A_R› ‹$A_R \sharp* R$› ‹$A_R \sharp* M$› ‹$A_R \sharp* xvec$› ‹$A_R \sharp* N$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* (!Q)$›
‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$xvec \sharp* (!Q)$› ‹$xvec \sharp* R$› ‹$xvec \sharp* M$› ‹*distinct xvec*›
by(*rule-tac* C=(Ψ, P, Q) **and** C'=(Ψ, P, Q) **in** *expandFrame*) (*assumption* | *simp*)+

from *RTrans* S ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* R'$› **have** $\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$
apply(*simp add*: *residualInject*)
by(*subst boundOutputChainAlpha''*[*symmetric*]) *auto*

moreover have *QTrans*: $\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|N|\!) \prec Q'$ **by** *fact*
with *QTrans* S ‹$(p \cdot xvec) \sharp* N$› **have** $\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|(p \cdot N)|\!) \prec (p \cdot Q')$ **using** ‹*distinctPerm* p› ‹$xvec \sharp* (!Q)$› ‹$(p \cdot xvec) \sharp* Q$›
by(*rule-tac inputAlpha*) *auto*
with ‹$(\Psi, P, Q) \in Rel$› ‹*guarded* P›
obtain P' S T **where** *PTrans*: $\Psi \otimes \Psi_R \rhd !P \longmapsto K(\!|(p \cdot N)|\!) \prec P'$ **and** $(\Psi \otimes \Psi_R, P', T \parallel !P) \in Rel$
and $(\Psi \otimes \Psi_R, (p \cdot Q'), S \parallel !Q) \in Rel$ **and** $(\Psi \otimes \Psi_R, S, T) \in Rel$
and *suppT*: ((*supp* T)::*name set*) $\subseteq$ *supp* P' **and** *suppS*: ((*supp* S)::*name set*) $\subseteq$ *supp*$(p \cdot Q')$
by(*drule-tac cSym*) (*auto dest*: *Der cExt*)
note ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$›
ultimately have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel P')$
using *PTrans FrR* ‹$\Psi_Q = SBottom'$› ‹$(p \cdot xvec) \sharp* P$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* R$› ‹$A_R \sharp* M$› ‹$A_R \sharp* P$›
by(*rule-tac Comm2*) (*assumption* | *simp*)+

moreover from ‹$A_R{}' \sharp* P$› ‹$A_R{}' \sharp* Q$› ‹$A_R{}' \sharp* N$› S ‹$xvec \sharp* A_R{}'$› ‹$(p \cdot xvec) \sharp* A_R{}'$› *PTrans QTrans* ‹*distinctPerm* p› **have** $A_R{}' \sharp* P'$ **and** $A_R{}' \sharp* Q'$
apply −
apply(*drule-tac inputFreshChainDerivative*, *auto*)
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, *symmetric*, *of* - - *p*], *simp*)
by(*force dest*: *inputFreshChainDerivative*)+
from ‹$xvec \sharp* P$› ‹$(p \cdot xvec) \sharp* N$› *PTrans* ‹*distinctPerm* p› **have** $(p \cdot xvec) \sharp* (p \cdot P')$
apply(*drule-tac inputFreshChainDerivative*, *simp*)
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, *symmetric*,

of - - *p*], *simp*)
by(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, *symmetric*, *of* - - *p*], *simp*)

{
from ‹$(\Psi \otimes \Psi_R, P', T \parallel !P) \in Rel$› **have** $(p \cdot (\Psi \otimes \Psi_R), (p \cdot P'), p \cdot (T \parallel !P)) \in Rel$
by(*rule Closed*)
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$(p \cdot xvec) \sharp* P$› S **have** $(\Psi \otimes (p \cdot \Psi_R), p \cdot P', (p \cdot T) \parallel !P) \in Rel$
by(*simp add*: *eqvts*)
hence $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', p \cdot P', (p \cdot T) \parallel !P) \in Rel$ **by**(*rule cExt*)
with ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$› **have** $(\Psi \otimes {\Psi_R}', (p \cdot P'), (p \cdot T) \parallel !P) \in Rel$
by(*metis Associativity StatEq compositionSym*)
with FrR' ‹${A_R}' \sharp* \Psi$› ‹${A_R}' \sharp* P'$› ‹${A_R}' \sharp* P$› ‹$xvec \sharp* {A_R}'$› ‹$(p \cdot xvec) \sharp* {A_R}'$› S ‹*distinctPerm p*› *suppT*
have $(\Psi, R' \parallel (p \cdot P'), R' \parallel ((p \cdot T) \parallel !P)) \in Rel$
apply(*rule-tac FrameParPres*)
apply(*assumption* | *simp add*: *freshChainSimps*)+
by(*auto simp add*: *fresh-star-def fresh-def*)
hence $(\Psi, R' \parallel (p \cdot P'), (R' \parallel (p \cdot T)) \parallel !P) \in Rel$ **by**(*blast intro*: *Assoc Trans*)
with ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› **have** $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel (p \cdot P')), ((\!|\nu{*}xvec|\!)(R' \parallel (p \cdot T))) \parallel !P) \in Rel$
by(*metis ResPres psiFreshVec ScopeExt Trans*)
hence $(\Psi, (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot R') \parallel P'), ((\!|\nu{*}xvec|\!)(R' \parallel (p \cdot T))) \parallel !P) \in Rel$
using ‹$(p \cdot xvec) \sharp* R'$› ‹$(p \cdot xvec) \sharp* (p \cdot P')$› S ‹*distinctPerm p*›
apply(*erule-tac rev-mp*) **by**(*subst resChainAlpha*[*of p*]) *auto*
moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded P*› ‹*guarded Q*› **have** $(\Psi, ((\!|\nu{*}xvec|\!)(R' \parallel (p \cdot T))) \parallel !P, ((\!|\nu{*}xvec|\!)(R' \parallel (p \cdot T))) \parallel !Q) \in Rel'$
by(*rule C1*)
moreover from ‹$(\Psi \otimes \Psi_R, (p \cdot Q'), S \parallel !Q) \in Rel$› ‹$(\Psi \otimes \Psi_R, S, T) \in Rel$› **have** $(\Psi \otimes \Psi_R, (p \cdot Q'), T \parallel !Q) \in Rel$
by(*blast intro*: *ParPres Trans*)
hence $(p \cdot (\Psi \otimes \Psi_R), p \cdot p \cdot Q', p \cdot (T \parallel !Q)) \in Rel$ **by**(*rule Closed*)
with S ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› ‹$xvec \sharp* (!Q)$› ‹$(p \cdot xvec) \sharp* Q$› ‹*distinctPerm p*›
have $(\Psi \otimes (p \cdot \Psi_R), Q', (p \cdot T) \parallel !Q) \in Rel$ **by**(*simp add*: *eqvts*)
hence $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', Q', (p \cdot T) \parallel !Q) \in Rel$ **by**(*rule cExt*)
with ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$› **have** $(\Psi \otimes {\Psi_R}', Q', (p \cdot T) \parallel !Q) \in Rel$
by(*metis Associativity StatEq compositionSym*)
with FrR' ‹${A_R}' \sharp* \Psi$› ‹${A_R}' \sharp* P'$› ‹${A_R}' \sharp* Q'$› ‹${A_R}' \sharp* Q$› *suppT suppS* ‹$xvec \sharp* {A_R}'$› ‹$(p \cdot xvec) \sharp* {A_R}'$› S ‹*distinctPerm p*›
have $(\Psi, R' \parallel Q', R' \parallel ((p \cdot T) \parallel !Q)) \in Rel$
apply(*rule-tac FrameParPres*)
apply(*assumption* | *simp*)+
apply(*simp add*: *freshChainSimps*)
by(*auto simp add*: *fresh-star-def fresh-def*)
hence $(\Psi, R' \parallel Q', (R' \parallel (p \cdot T)) \parallel !Q) \in Rel$ **by**(*blast intro*: *Assoc Trans*)

with ‹*xvec* ♯* Ψ› ‹*xvec* ♯* (!*Q*)› **have** (Ψ, ⦇ν*xvec⦈(*R*′ ‖ *Q*′), (⦇ν*xvec⦈(*R*′ ‖ (*p* · *T*))) ‖ !*Q*) ∈ *Rel*
by(*metis ResPres psiFreshVec ScopeExt Trans*)
ultimately have (Ψ, ⦇ν*(*p* · *xvec*)⦈((*p* · *R*′) ‖ *P*′), ⦇ν*xvec⦈(*R*′ ‖ *Q*′)) ∈ *Rel*′
by(*blast intro*: *cSym Compose*)
}
ultimately show *?case* **by** *blast*
qed
qed
unbundle *relcomp-syntax*
end

end

theory *Bisim-Pres*
imports *Bisimulation Sim-Pres*
begin

context *env* **begin**

lemma *bisimInputPres*:
fixes Ψ :: ′*b*
and *P* :: (′*a*, ′*b*, ′*c*) *psi*
and *Q* :: (′*a*, ′*b*, ′*c*) *psi*
and *M* :: ′*a*
and *xvec* :: *name list*
and *N* :: ′*a*

assumes ⋀*Tvec*. *length xvec* = *length Tvec* ⟹ Ψ ▷ *P*[*xvec*::=*Tvec*] ∼ *Q*[*xvec*::=*Tvec*]

shows Ψ ▷ *M*⦇λ*xvec *N*⦈.*P* ∼ *M*⦇λ*xvec *N*⦈.*Q*
proof −
let *?X* = {(Ψ, *M*⦇λ*xvec *N*⦈.*P*, *M*⦇λ*xvec *N*⦈.*Q*) | Ψ *M xvec N P Q*. ∀ *Tvec*. *length xvec* = *length Tvec* ⟶ Ψ ▷ *P*[*xvec*::=*Tvec*] ∼ *Q*[*xvec*::=*Tvec*]}

from *assms* **have** (Ψ, *M*⦇λ*xvec *N*⦈.*P*, *M*⦇λ*xvec *N*⦈.*Q*) ∈ *?X* **by** *blast*
thus *?thesis*
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* Ψ *P Q*)
thus *?case* **by** *auto*
next
case(*cSim* Ψ *P Q*)
thus *?case* **by**(*blast intro*: *inputPres*)
next
case(*cExt* Ψ *P Q* Ψ′)
thus *?case* **by**(*blast dest*: *bisimE*)
next
case(*cSym* Ψ *P Q*)
thus *?case* **by**(*blast dest*: *bisimE*)

qed
qed

lemma *bisimOutputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$

assumes $\Psi \rhd P \sim Q$

shows $\Psi \rhd M\langle N\rangle.P \sim M\langle N\rangle.Q$
proof −
let $?X = \{(\Psi, M\langle N\rangle.P, M\langle N\rangle.Q) \mid \Psi\ M\ N\ P\ Q.\ \Psi \rhd P \sim Q\}$
from ‹$\Psi \rhd P \sim Q$› **have** $(\Psi, M\langle N\rangle.P, M\langle N\rangle.Q) \in ?X$ **by** *auto*
thus *?thesis*
by(*coinduct rule*: *bisimCoinduct*, *auto*) (*blast intro*: *outputPres dest*: *bisimE*)+
qed

lemma *bisimCasePres*:
fixes Ψ :: $'b$
and CsP :: $('c \times ('a, 'b, 'c)$ *psi*$)$ *list*
and CsQ :: $('c \times ('a, 'b, 'c)$ *psi*$)$ *list*

assumes $\bigwedge\varphi\ P.\ (\varphi, P)$ *mem* $CsP \Longrightarrow \exists\, Q.\ (\varphi, Q)$ *mem* $CsQ \wedge$ *guarded* $Q \wedge \Psi \rhd P \sim Q$
and $\bigwedge\varphi\ Q.\ (\varphi, Q)$ *mem* $CsQ \Longrightarrow \exists\, P.\ (\varphi, P)$ *mem* $CsP \wedge$ *guarded* $P \wedge \Psi \rhd P \sim Q$

shows $\Psi \rhd$ *Cases* $CsP \sim$ *Cases* CsQ
proof −
let $?X = \{(\Psi,$ *Cases* $CsP,$ *Cases* $CsQ) \mid \Psi\ CsP\ CsQ.\ (\forall\varphi\ P.\ (\varphi, P)$ *mem* $CsP \longrightarrow (\exists\, Q.\ (\varphi, Q)$ *mem* $CsQ \wedge$ *guarded* $Q \wedge \Psi \rhd P \sim Q)) \wedge$
$(\forall\varphi\ Q.\ (\varphi, Q)$ *mem* $CsQ \longrightarrow (\exists\, P.\ (\varphi, P)$ *mem* $CsP \wedge$ *guarded* $P \wedge \Psi \rhd P \sim Q))\}$
from *assms* **have** $(\Psi,$ *Cases* $CsP,$ *Cases* $CsQ) \in ?X$ **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* $\Psi\ P\ Q$)
thus *?case* **by** *auto*
next
case(*cSim* Ψ *CasesP CasesQ*)
then obtain *CsP CsQ* **where** *C1*: $\bigwedge\varphi\ Q.\ (\varphi, Q)$ *mem* $CsQ \Longrightarrow \exists\, P.\ (\varphi, P)$ *mem* $CsP \wedge$ *guarded* $P \wedge \Psi \rhd P \sim Q$
and *A*: *CasesP* = *Cases CsP* **and** *B*: *CasesQ* = *Cases CsQ*
by *auto*
note *C1*
moreover have $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow \Psi \rhd P \rightsquigarrow[bisim]\ Q$ **by**(*rule bisimE*)

moreover have *bisim* $\subseteq$ *?X* $\cup$ *bisim* **by** *blast*
ultimately have $\Psi \rhd$ *Cases CsP* $\rightsquigarrow$[(*?X* $\cup$ *bisim*)] *Cases CsQ*
by(*rule casePres*)
thus *?case* **using** *A B* **by** *blast*
next
case(*cExt* Ψ *P Q*)
thus *?case* **by**(*blast dest*: *bisimE*)
next
case(*cSym* Ψ *P Q*)
thus *?case* **by**(*blast dest*: *bisimE*)
qed
qed

lemma *bisimResPres*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *x* :: *name*

assumes $\Psi \rhd P \sim Q$
and $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)P \sim (\!|\nu x|\!)Q$
proof −
let *?X* = $\{(\Psi, (\!|\nu x|\!)P, (\!|\nu x|\!)Q) \mid \Psi\ x\ P\ Q.\ \Psi \rhd P \sim Q \wedge x \sharp \Psi\}$

from *assms* **have** $(\Psi, (\!|\nu x|\!)P, (\!|\nu x|\!)Q) \in$ *?X* **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* Ψ *xP xQ*)
from ‹(Ψ, *xP*, *xQ*) $\in$ *?X*› **obtain** *x P Q* **where** $\Psi \rhd P \sim Q$ **and** $x \sharp \Psi$ **and** $xP = (\!|\nu x|\!)P$ **and** $xQ = (\!|\nu x|\!)Q$
by *auto*
moreover from ‹$\Psi \rhd P \sim Q$› **have** *PeqQ*: *insertAssertion*(*extractFrame P*) $\Psi \simeq_F$ *insertAssertion*(*extractFrame Q*) Ψ
by(*rule bisimE*)
ultimately show *?case* **by**(*auto intro*: *frameResPres*)
next
case(*cSim* Ψ *xP xQ*)
from ‹(Ψ, *xP*, *xQ*) $\in$ *?X*› **obtain** *x P Q* **where** $\Psi \rhd P \sim Q$ **and** $x \sharp \Psi$ **and** $xP = (\!|\nu x|\!)P$ **and** $xQ = (\!|\nu x|\!)Q$
by *auto*
from ‹$\Psi \rhd P \sim Q$› **have** $\Psi \rhd P \rightsquigarrow$[*bisim*] *Q* **by**(*rule bisimE*)
moreover have *eqvt ?X*
by(*force simp add*: *eqvt-def bisimClosed pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence *eqvt*(*?X* $\cup$ *bisim*) **by** *auto*
moreover note ‹$x \sharp \Psi$›
moreover have *bisim* $\subseteq$ *?X* $\cup$ *bisim* **by** *auto*

moreover have $\bigwedge\Psi\ P\ Q\ x.\ [\![(\Psi,\ P,\ Q) \in bisim;\ x \sharp \Psi]\!] \Longrightarrow (\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X \cup bisim$
by *auto*
ultimately have $\Psi \rhd (\!|\nu x|\!)P \rightsquigarrow[(?X \cup bisim)]\ (\!|\nu x|\!)Q$
by(*rule resPres*)
with ‹$xP = (\!|\nu x|\!)P$› ‹$xQ = (\!|\nu x|\!)Q$› **show** *?case*
by *simp*
next
case(*cExt* Ψ xP xQ Ψ')
from ‹$(\Psi,\ xP,\ xQ) \in ?X$› **obtain** $x\ P\ Q$ **where** $\Psi \rhd P \sim Q$ **and** $x \sharp \Psi$ **and** $xP = (\!|\nu x|\!)P$ **and** $xQ = (\!|\nu x|\!)Q$
by *auto*
obtain y::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \Psi$ **and** $y \sharp \Psi'$
by(*generate-fresh name*, *auto simp add*: *fresh-prod*)
from ‹$\Psi \rhd P \sim Q$› **have** $\Psi \otimes ([(x,\ y)] \cdot \Psi') \rhd P \sim Q$
by(*rule bisimE*)
hence $([(x,\ y)] \cdot (\Psi \otimes ([(x,\ y)] \cdot \Psi'))) \rhd ([(x,\ y)] \cdot P) \sim ([(x,\ y)] \cdot Q)$
by(*rule bisimClosed*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \otimes \Psi' \rhd ([(x,\ y)] \cdot P) \sim ([(x,\ y)] \cdot Q)$
by(*simp add*: *eqvts*)
with ‹$y \sharp \Psi$› ‹$y \sharp \Psi'$› **have** $(\Psi \otimes \Psi',\ (\!|\nu y|\!)([(x,\ y)] \cdot P),\ (\!|\nu y|\!)([(x,\ y)] \cdot Q)) \in ?X$
by *auto*
moreover from ‹$y \sharp P$› ‹$y \sharp Q$› **have** $(\!|\nu x|\!)P = (\!|\nu y|\!)([(x,\ y)] \cdot P)$ **and** $(\!|\nu x|\!)Q = (\!|\nu y|\!)([(x,\ y)] \cdot Q)$
by(*simp add*: *alphaRes*)+
ultimately show *?case* **using** ‹$xP = (\!|\nu x|\!)P$› ‹$xQ = (\!|\nu x|\!)Q$› **by** *simp*
next
case(*cSym* Ψ P Q)
thus *?case*
by(*blast dest*: *bisimE*)
qed
qed

lemma *bisimResChainPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \sim Q$
and $xvec \sharp* \Psi$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \sim (\!|\nu{*}xvec|\!)Q$
using *assms*
by(*induct xvec*) (*auto intro*: *bisimResPres*)

lemma *bisimParPresAux*:
fixes Ψ :: $'b$

and Ψ_R :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*
and A_R :: *name list*

assumes $\Psi \otimes \Psi_R \rhd P \sim Q$
and *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$

shows $\Psi \rhd P \parallel R \sim Q \parallel R$
proof −
let $?X = \{(\Psi,\ (\!|\nu{*}xvec|\!)(P \parallel R),\ (\!|\nu{*}xvec|\!)(Q \parallel R)) \mid xvec\ \Psi\ P\ Q\ R.\ xvec \sharp* \Psi \wedge (\forall A_R\ \Psi_R.\ (extractFrame\ R = \langle A_R,\ \Psi_R\rangle \wedge A_R \sharp* \Psi \wedge A_R \sharp* P \wedge A_R \sharp* Q) \longrightarrow \Psi \otimes \Psi_R \rhd P \sim Q)\}$
{
fix *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

assume $xvec \sharp* \Psi$
and $\bigwedge A_R\ \Psi_R.\ [\![extractFrame\ R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q]\!] \Longrightarrow \Psi \otimes \Psi_R \rhd P \sim Q$

hence $(\Psi,\ (\!|\nu{*}xvec|\!)(P \parallel R),\ (\!|\nu{*}xvec|\!)(Q \parallel R)) \in ?X$
apply *auto*
by *blast*
}

note $XI = this$
{
fix *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*
and C :: $'d$::*fs-name*

assume $xvec \sharp* \Psi$
and A: $\bigwedge A_R\ \Psi_R.\ [\![extractFrame\ R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q;\ A_R \sharp* C]\!] \Longrightarrow \Psi \otimes \Psi_R \rhd P \sim Q$

from ‹$xvec \sharp* \Psi$› **have** $(\Psi,\ (\!|\nu{*}xvec|\!)(P \parallel R),\ (\!|\nu{*}xvec|\!)(Q \parallel R)) \in ?X$
proof(*rule XI*)

fix A_R Ψ_R
assume *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$
obtain *p*::*name prm* **where** $(p \cdot A_R) \sharp* \Psi$ **and** $(p \cdot A_R) \sharp* P$ **and** $(p \cdot A_R) \sharp* Q$ **and** $(p \cdot A_R) \sharp* R$ **and** $(p \cdot A_R) \sharp* C$
and $(p \cdot A_R) \sharp* \Psi_R$ **and** *S*: $(set\ p) \subseteq (set\ A_R) \times (set(p \cdot A_R))$ **and** *distinctPerm p*
by(*rule-tac c*=(Ψ, *P*, *Q*, *R*, Ψ_R, *C*) **in** *name-list-avoiding*) *auto*
from *FrR* ‹$(p \cdot A_R) \sharp* \Psi_R$› *S* **have** *extractFrame* $R = \langle (p \cdot A_R), p \cdot \Psi_R\rangle$ **by**(*simp add*: *frameChainAlpha′*)

moreover assume $A_R \sharp* \Psi$
hence $(p \cdot A_R) \sharp* (p \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* \Psi$› *S* **have** $(p \cdot A_R) \sharp* \Psi$ **by** *simp*
moreover assume $A_R \sharp* P$
hence $(p \cdot A_R) \sharp* (p \cdot P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* P$› ‹$(p \cdot A_R) \sharp* P$› *S* **have** $(p \cdot A_R) \sharp* P$ **by** *simp*
moreover assume $A_R \sharp* Q$
hence $(p \cdot A_R) \sharp* (p \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* Q$› ‹$(p \cdot A_R) \sharp* Q$› *S* **have** $(p \cdot A_R) \sharp* Q$ **by** *simp*
ultimately have $\Psi \otimes (p \cdot \Psi_R) \rhd P \sim Q$ **using** ‹$(p \cdot A_R) \sharp* C$› *A* **by** *blast*
hence $(p \cdot (\Psi \otimes (p \cdot \Psi_R))) \rhd (p \cdot P) \sim (p \cdot Q)$ **by**(*rule bisimClosed*)
with ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$(p \cdot A_R) \sharp* P$› ‹$A_R \sharp* Q$› ‹$(p \cdot A_R) \sharp* Q$› *S* ‹*distinctPerm p*›
show $\Psi \otimes \Psi_R \rhd P \sim Q$ **by**(*simp add*: *eqvts*)
qed
}
note *XI′* = *this*

have *eqvt ?X*
apply(*auto simp add*: *eqvt-def*)
apply(*rule-tac x*=$p \cdot$ *xvec* **in** *exI*)
apply(*rule-tac x*=$p \cdot P$ **in** *exI*)
apply(*rule-tac x*=$p \cdot Q$ **in** *exI*)
apply(*rule-tac x*=$p \cdot R$ **in** *exI*)
apply(*simp add*: *eqvts*)
apply(*simp add*: *fresh-star-bij*)
apply(*clarify*)
apply(*erule-tac x*=(*rev p*) $\cdot A_R$ **in** *allE*)
apply(*erule-tac x*=(*rev p*) $\cdot \Psi_R$ **in** *allE*)
apply(*drule mp*)
apply(*rule conjI*)
apply(*rule-tac pi*=*p* **in** *pt-bij4*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *eqvts*)
defer
apply(*drule-tac p*=*p* **in** *bisimClosed*)
apply(*simp add*: *eqvts*)

apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of p*, *THEN sym*])
apply *simp*
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of p*, *THEN sym*])
apply *simp*
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of p*, *THEN sym*])
by *simp*

moreover have *Res*: $\bigwedge \Psi\ P\ Q\ x.\ [\![(\Psi,\ P,\ Q) \in ?X \cup bisim;\ x \sharp \Psi]\!] \Longrightarrow (\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X \cup bisim$
proof −
fix $\Psi\ P\ Q\ x$
assume $(\Psi,\ P,\ Q) \in ?X \cup bisim$ **and** $(x::name) \sharp \Psi$
show $(\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X \cup bisim$
proof(*case-tac* $(\Psi,\ P,\ Q) \in ?X$)
assume $(\Psi,\ P,\ Q) \in ?X$
with ‹$x \sharp \Psi$› **have** $(\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X$
apply *auto*
by(*rule-tac x=x#xvec* **in** *exI*) *auto*
thus *?thesis* **by** *simp*
next
assume $\neg(\Psi,\ P,\ Q) \in ?X$
with ‹$(\Psi,\ P,\ Q) \in ?X \cup bisim$› **have** $\Psi \rhd P \sim Q$
by *blast*
hence $\Psi \rhd (\!|\nu x|\!)P \sim (\!|\nu x|\!)Q$ **using** ‹$x \sharp \Psi$›
by(*rule bisimResPres*)
thus *?thesis*
by *simp*
qed
qed

have $(\Psi,\ P \parallel R,\ Q \parallel R) \in ?X$
proof −
{
fix $A_R{}'$ :: *name list*
and $\Psi_R{}'$ :: *'b*

assume *FrR'*: *extractFrame* $R = \langle A_R{}',\ \Psi_R{}'\rangle$
and $A_R{}' \sharp* \Psi$
and $A_R{}' \sharp* P$
and $A_R{}' \sharp* Q$

obtain p **where** $(p \cdot A_R{}') \sharp* A_R$ **and** $(p \cdot A_R{}') \sharp* \Psi_R{}'$ **and** $(p \cdot A_R{}') \sharp* \Psi$ **and** $(p \cdot A_R{}') \sharp* P$ **and** $(p \cdot A_R{}') \sharp* Q$
and *Sp*: $(set\ p) \subseteq (set\ A_R{}') \times (set(p \cdot A_R{}'))$ **and** *distinctPerm p*
by(*rule-tac* $c=(A_R,\ \Psi,\ \Psi_R{}',\ P,\ Q)$ **in** *name-list-avoiding*) *auto*

from FrR' ‹$(p \cdot A_R{}') \sharp* \Psi_R{}'$› Sp **have** $extractFrame\ R = \langle (p \cdot A_R{}'),\ p \cdot \Psi_R{}' \rangle$
by(*simp add*: *frameChainAlpha eqvts*)
with FrR ‹$(p \cdot A_R{}') \sharp* A_R$› **obtain** q::*name prm*
where Sq: $set\ q \subseteq set(p \cdot A_R{}') \times set\ A_R$ **and** $distinctPerm\ q$ **and** $\Psi_R = q \cdot p \cdot \Psi_R{}'$
by(*force elim*: *frameChainEq*)

from ‹$\Psi \otimes \Psi_R \rhd P \sim Q$› ‹$\Psi_R = q \cdot p \cdot \Psi_R{}'$› **have** $\Psi \otimes (q \cdot p \cdot \Psi_R{}') \rhd P \sim Q$ **by** *simp*
hence $(q \cdot (\Psi \otimes (q \cdot p \cdot \Psi_R{}))) \rhd (q \cdot P) \sim (q \cdot Q)$ **by**(*rule bisimClosed*)
with Sq ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$A_R \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* Q$› ‹$distinctPerm\ q$›
have $\Psi \otimes (p \cdot \Psi_R{}') \rhd P \sim Q$ **by**(*simp add*: *eqvts*)
hence $(p \cdot (\Psi \otimes (p \cdot \Psi_R{}))) \rhd (p \cdot P) \sim (p \cdot Q)$ **by**(*rule bisimClosed*)
with Sp ‹$A_R{}' \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$A_R{}' \sharp* P$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$A_R{}' \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* Q$› ‹$distinctPerm\ p$›
have $\Psi \otimes \Psi_R{}' \rhd P \sim Q$ **by**(*simp add*: *eqvts*)
}
thus *?thesis*
apply *auto*
apply(*rule-tac* x=[] **in** *exI*)
by *auto blast*
qed
thus *?thesis*
proof(*coinduct rule*: *bisimCoinduct*)
case($cStatEq\ \Psi\ PR\ QR$)
from ‹$(\Psi,\ PR,\ QR) \in ?X$›
obtain $xvec\ P\ Q\ R\ A_R\ \Psi_R$ **where** $PFrR$: $PR = (\!|\nu{*}xvec|\!)(P \parallel R)$ **and** $QFrR$: $QR = (\!|\nu{*}xvec|\!)(Q \parallel R)$
and $xvec \sharp* \Psi$ **and** FrR: $extractFrame\ R = \langle A_R,\ \Psi_R \rangle$ **and** $PSimQ$: $\Psi \otimes \Psi_R \rhd P \sim Q$
and $A_R \sharp* xvec$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* R$
apply *auto*
apply(*subgoal-tac* $\exists A_R\ \Psi_R.\ extractFrame\ R = \langle A_R,\ \Psi_R \rangle \wedge A_R \sharp* (xvec,\ \Psi,\ P,\ Q,\ R)$)
apply *auto*
apply(*rule-tac* F=$extractFrame\ R$ **and** C=$(xvec,\ \Psi,\ P,\ Q,\ R)$ **in** *freshFrame*)
by *auto*

obtain $A_P\ \Psi_P$ **where** FrP: $extractFrame\ P = \langle A_P,\ \Psi_P \rangle$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* A_R$ **and** $A_P \sharp* \Psi_R$
by(*rule-tac* C=$(\Psi,\ A_R,\ \Psi_R)$ **in** *freshFrame*) *auto*
obtain $A_Q\ \Psi_Q$ **where** FrQ: $extractFrame\ Q = \langle A_Q,\ \Psi_Q \rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* A_R$ **and** $A_Q \sharp* \Psi_R$
by(*rule-tac* C=$(\Psi,\ A_R,\ \Psi_R)$ **in** *freshFrame*) *auto*
from ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_P \sharp* A_R$› ‹$A_Q \sharp* A_R$› $FrP\ FrQ$ **have** $A_R \sharp* \Psi_P$ **and** $A_R \sharp* \Psi_Q$

by(*force dest*: *extractFrameFreshChain*)+

have $\langle(A_P@A_R),\ \Psi \otimes \Psi_P \otimes \Psi_R\rangle \simeq_F \langle(A_Q@A_R),\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle$
proof −
have $\langle A_P,\ \Psi \otimes \Psi_P \otimes \Psi_R\rangle \simeq_F \langle A_P,\ (\Psi \otimes \Psi_R) \otimes \Psi_P\rangle$
by(*metis frameResChainPres frameNilStatEq Associativity Commutativity AssertionStatEqTrans Composition*)
moreover from *PSimQ* **have** *insertAssertion*(*extractFrame P*) $(\Psi \otimes \Psi_R) \simeq_F$ *insertAssertion*(*extractFrame Q*) $(\Psi \otimes \Psi_R)$
by(*rule bisimE*)
with *FrP FrQ freshCompChain* ‹$A_P \sharp\!* \Psi$› ‹$A_P \sharp\!* \Psi_R$› ‹$A_Q \sharp\!* \Psi$› ‹$A_Q \sharp\!* \Psi_R$› **have** $\langle A_P,\ (\Psi \otimes \Psi_R) \otimes \Psi_P\rangle \simeq_F \langle A_Q,\ (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by *auto*
moreover have $\langle A_Q,\ (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \simeq_F \langle A_Q,\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle$
by(*metis frameResChainPres frameNilStatEq Associativity Commutativity AssertionStatEqTrans Composition*)
ultimately have $\langle A_P,\ \Psi \otimes \Psi_P \otimes \Psi_R\rangle \simeq_F \langle A_Q,\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle$
by(*blast intro*: *FrameStatEqTrans*)
hence $\langle(A_R@A_P),\ \Psi \otimes \Psi_P \otimes \Psi_R\rangle \simeq_F \langle(A_R@A_Q),\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle$
by(*drule-tac frameResChainPres*) (*simp add*: *frameChainAppend*)
thus *?thesis*
apply(*simp add*: *frameChainAppend*)
by(*metis frameResChainComm FrameStatEqTrans*)
qed
moreover from ‹$A_P \sharp\!* \Psi$› ‹$A_R \sharp\!* \Psi$› **have** $(A_P@A_R) \sharp\!* \Psi$ **by** *simp*
moreover from ‹$A_Q \sharp\!* \Psi$› ‹$A_R \sharp\!* \Psi$› **have** $(A_Q@A_R) \sharp\!* \Psi$ **by** *simp*
ultimately have *PFrRQR*: *insertAssertion*(*extractFrame*($P \parallel R$)) $\Psi \simeq_F$ *insertAssertion*(*extractFrame*($Q \parallel R$)) Ψ
using *FrP FrQ FrR* ‹$A_P \sharp\!* A_R$› ‹$A_P \sharp\!* \Psi_R$› ‹$A_Q \sharp\!* A_R$› ‹$A_Q \sharp\!* \Psi_R$› ‹$A_R \sharp\!* \Psi_P$› ‹$A_R \sharp\!* \Psi_Q$›
by *simp*

from ‹*xvec* $\sharp\!* \Psi$› **have** *insertAssertion* (*extractFrame*($(\!|\nu*xvec|\!)P \parallel R$)) $\Psi \simeq_F (\!|\nu*xvec|\!)$(*insertAssertion* (*extractFrame*($P \parallel R$)) Ψ)
by(*rule insertAssertionExtractFrameFresh*)
moreover from *PFrRQR* **have** $(\!|\nu*xvec|\!)$(*insertAssertion* (*extractFrame*($P \parallel R$)) Ψ) $\simeq_F (\!|\nu*xvec|\!)$(*insertAssertion* (*extractFrame*($Q \parallel R$)) Ψ)
by(*induct xvec*) (*auto intro*: *frameResPres*)
moreover from ‹*xvec* $\sharp\!* \Psi$› **have** $(\!|\nu*xvec|\!)$(*insertAssertion* (*extractFrame*($Q \parallel R$)) Ψ) $\simeq_F$ *insertAssertion* (*extractFrame*($(\!|\nu*xvec|\!)Q \parallel R$)) Ψ
by(*rule FrameStatEqSym*[*OF insertAssertionExtractFrameFresh*])
ultimately show *?case* **using** *PFrR QFrR*
by(*blast intro*: *FrameStatEqTrans*)
next
case(*cSim* Ψ *PR QR*)
{
fix Ψ *P Q R xvec*
assume $\bigwedge A_R\ \Psi_R.\ [\![$*extractFrame* $R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp\!* \Psi;\ A_R \sharp\!* P;\ A_R \sharp\!* Q]\!] \Longrightarrow \Psi \otimes \Psi_R \rhd P \sim Q$

moreover have *eqvt bisim* **by** *simp*
moreover from ‹*eqvt ?X*› **have** *eqvt*(*?X* ∪ *bisim*) **by** *auto*
moreover from *bisimE*(*1*) **have** $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow$ *insertAssertion* (*extractFrame Q*) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame P*) Ψ **by**(*simp add*: *FrameStatEq-def*)
moreover note *bisimE*(*2*) *bisimE*(*3*)
moreover
{
fix $\Psi\ P\ Q\ A_R\ \Psi_R\ R$
assume *PSimQ*: $\Psi \otimes \Psi_R \rhd P \sim Q$
and *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$
hence $(\Psi, P \parallel R, Q \parallel R) \in ?X$
proof −
have $P \parallel R = (\!|\nu*[]|\!)(P \parallel R)$ **by** *simp*
moreover have $Q \parallel R = (\!|\nu*[]|\!)(Q \parallel R)$ **by** *simp*
moreover have ([]::*name list*) $\sharp*\ \Psi$ **by** *simp*
moreover
{
fix $A_R'\ \Psi_R'$

assume *FrR'*: *extractFrame* $R = \langle A_R', \Psi_R'\rangle$
and $A_R' \sharp* \Psi$
and $A_R' \sharp* P$
and $A_R' \sharp* Q$
obtain p **where** $(p \cdot A_R') \sharp* A_R$
and $(p \cdot A_R') \sharp* \Psi_R'$
and $(p \cdot A_R') \sharp* \Psi$
and $(p \cdot A_R') \sharp* P$
and $(p \cdot A_R') \sharp* Q$
and *S*: (*set p*) ⊆ (*set* A_R') × (*set*($p \cdot A_R'$)) **and** *distinctPerm p*
by(*rule-tac c*=(A_R, Ψ, Ψ_R', P, Q) **in** *name-list-avoiding*) *auto*

from ‹$(p \cdot A_R') \sharp* \Psi_R'$› *S* **have** $\langle A_R', \Psi_R'\rangle = \langle p \cdot A_R', p \cdot \Psi_R'\rangle$
by(*simp add*: *frameChainAlpha*)

with *FrR'* **have** *FrR''*: *extractFrame* $R = \langle p \cdot A_R', p \cdot \Psi_R'\rangle$ **by** *simp*
with *FrR* ‹$(p \cdot A_R') \sharp* A_R$›
obtain q **where** $p \cdot \Psi_R' = (q$::*name prm*$) \cdot \Psi_R$ **and** *S'*: *set* $q \subseteq$ (*set* A_R) × *set*($p \cdot A_R'$) **and** *distinctPerm q*
apply *auto*
apply(*drule-tac sym*) **apply** *simp*
by(*drule-tac frameChainEq*) *auto*
from *PSimQ* **have** $(q \cdot (\Psi \otimes \Psi_R)) \rhd (q \cdot P) \sim (q \cdot Q)$
by(*rule bisimClosed*)
with ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$(p \cdot A_R') \sharp* \Psi$› ‹$(p \cdot A_R') \sharp* P$›

‹$(p \cdot A_R{}') \sharp* Q$› S'
have $\Psi \otimes (q \cdot \Psi_R) \rhd P \sim Q$ **by**(*simp add*: *eqvts*)
hence $(p \cdot (\Psi \otimes (q \cdot \Psi_R))) \rhd (p \cdot P) \sim (p \cdot Q)$ **by**(*rule bisimClosed*)
with ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* P$› ‹$A_R{}' \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$(p \cdot A_R{}') \sharp* Q$› S ‹*distinctPerm p*› ‹$(p \cdot \Psi_R{}') = q \cdot \Psi_R$›
have $\Psi \otimes \Psi_R{}' \rhd P \sim Q$
by(*drule-tac sym*) (*simp add*: *eqvts*)
}
ultimately show *?thesis*
by *blast*
qed
hence $(\Psi, P \parallel R, Q \parallel R) \in ?X \cup bisim$
by *simp*
}
moreover have $\bigwedge \Psi\ P\ Q\ xvec.\ [\![(\Psi, P, Q) \in ?X \cup bisim; (xvec::name\ list) \sharp* \Psi]\!] \Longrightarrow (\Psi, (\!|\nu*xvec|\!)P, (\!|\nu*xvec|\!)Q) \in ?X \cup bisim$
proof −
fix $\Psi\ P\ Q\ xvec$
assume $(\Psi, P, Q) \in ?X \cup bisim$
assume $(xvec::name\ list) \sharp* \Psi$
thus $(\Psi, (\!|\nu*xvec|\!)P, (\!|\nu*xvec|\!)Q) \in ?X \cup bisim$
proof(*induct xvec*)
case *Nil*
thus *?case* **using** ‹$(\Psi, P, Q) \in ?X \cup bisim$› **by** *simp*
next
case(*Cons x xvec*)
thus *?case* **by**(*simp only*: *resChain.simps*) (*rule-tac Res*, *auto*)
qed
qed
ultimately have $\Psi \rhd P \parallel R \rightsquigarrow[(?X \cup bisim)]\ Q \parallel R$ **using** *statEqBisim*
by(*rule parPres*)
moreover assume $(xvec::name\ list) \sharp* \Psi$
moreover from ‹*eqvt ?X*› **have** *eqvt*$(?X \cup bisim)$ **by** *auto*
ultimately have $\Psi \rhd (\!|\nu*xvec|\!)(P \parallel R) \rightsquigarrow[(?X \cup bisim)]\ (\!|\nu*xvec|\!)(Q \parallel R)$ **using** *Res*
by(*rule-tac resChainPres*)
}
with ‹$(\Psi, PR, QR) \in ?X$› **show** *?case* **by** *blast*
next
case(*cEx*t $\Psi\ PR\ QR\ \Psi'$)

from ‹$(\Psi, PR, QR) \in ?X$›
obtain $xvec\ P\ Q\ R\ A_R\ \Psi_R$ **where** *PFrR*: $PR = (\!|\nu*xvec|\!)(P \parallel R)$ **and** *QFrR*: $QR = (\!|\nu*xvec|\!)(Q \parallel R)$
and $xvec \sharp* \Psi$ **and** A: $\forall A_R\ \Psi_R.\ (extractFrame\ R = \langle A_R, \Psi_R\rangle \wedge A_R \sharp* \Psi \wedge A_R \sharp* P \wedge A_R \sharp* Q) \longrightarrow \Psi \otimes \Psi_R \rhd P \sim Q$
by *auto*

obtain p **where** $(p \cdot xvec) \sharp* \Psi$

and $(p \cdot xvec) \sharp* P$
and $(p \cdot xvec) \sharp* Q$
and $(p \cdot xvec) \sharp* R$
and $(p \cdot xvec) \sharp* \Psi'$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm p*
by(*rule-tac c*=(Ψ, P, Q, R, Ψ') **in** *name-list-avoiding*) *auto*

from ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* R$› S **have** $(\!|\nu*xvec|\!)(P \parallel R) = (\!|\nu*(p \cdot xvec)|\!)(p \cdot (P \parallel R))$
by(*subst resChainAlpha*) *auto*
hence *PRAlpha*: $(\!|\nu*xvec|\!)(P \parallel R) = (\!|\nu*(p \cdot xvec)|\!)((p \cdot P) \parallel (p \cdot R))$
by(*simp add*: *eqvts*)

from ‹$(p \cdot xvec) \sharp* Q$› ‹$(p \cdot xvec) \sharp* R$› S **have** $(\!|\nu*xvec|\!)(Q \parallel R) = (\!|\nu*(p \cdot xvec)|\!)(p \cdot (Q \parallel R))$
by(*subst resChainAlpha*) *auto*
hence *QRAlpha*: $(\!|\nu*xvec|\!)(Q \parallel R) = (\!|\nu*(p \cdot xvec)|\!)((p \cdot Q) \parallel (p \cdot R))$
by(*simp add*: *eqvts*)

from ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu*(p \cdot xvec)|\!)((p \cdot P) \parallel (p \cdot R)), (\!|\nu*(p \cdot xvec)|\!)((p \cdot Q) \parallel (p \cdot R))) \in ?X$
proof(*rule-tac C2*=$(\Psi, (p \cdot P), (p \cdot Q), R, \Psi', xvec, p \cdot xvec)$ **in** *XI'*, *auto*)
fix $A_R\ \Psi_R$
assume *FrR*: *extractFrame* $(p \cdot R) = \langle A_R, \Psi_R\rangle$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* \Psi'$ **and** $A_R \sharp* (p \cdot P)$ **and** $A_R \sharp* (p \cdot Q)$
from *FrR* **have** $(p \cdot (extractFrame\ (p \cdot R))) = (p \cdot \langle A_R, \Psi_R\rangle)$ **by** *simp*
with ‹*distinctPerm p*› **have** $extractFrame\ R = \langle p \cdot A_R, p \cdot \Psi_R\rangle$ **by**(*simp add*: *eqvts*)
moreover from ‹$A_R \sharp* \Psi$› **have** $(p \cdot A_R) \sharp* (p \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S **have** $(p \cdot A_R) \sharp* \Psi$ **by** *simp*
moreover from ‹$A_R \sharp* (p \cdot P)$› **have** $(p \cdot A_R) \sharp* (p \cdot p \cdot P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $(p \cdot A_R) \sharp* P$ **by** *simp*
moreover from ‹$A_R \sharp* (p \cdot Q)$› **have** $(p \cdot A_R) \sharp* (p \cdot p \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $(p \cdot A_R) \sharp* Q$ **by** *simp*
ultimately have $\Psi \otimes (p \cdot \Psi_R) \rhd P \sim Q$ **using** A **by** *blast*
hence $(\Psi \otimes (p \cdot \Psi_R)) \otimes (p \cdot \Psi') \rhd P \sim Q$ **by**(*rule bisimE*)
moreover have $(\Psi \otimes (p \cdot \Psi_R)) \otimes (p \cdot \Psi') \simeq (\Psi \otimes (p \cdot \Psi')) \otimes (p \cdot \Psi_R)$
by(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)
ultimately have $(\Psi \otimes (p \cdot \Psi')) \otimes (p \cdot \Psi_R) \rhd P \sim Q$
by(*rule statEqBisim*)
hence $(p \cdot ((\Psi \otimes (p \cdot \Psi')) \otimes (p \cdot \Psi_R))) \rhd (p \cdot P) \sim (p \cdot Q)$
by(*rule bisimClosed*)
with ‹*distinctPerm p*› ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S **show** $(\Psi \otimes \Psi') \otimes \Psi_R \rhd (p \cdot P) \sim (p \cdot Q)$
by(*simp add*: *eqvts*)

```
    qed
    with PFrR QFrR PRAlpha QRAlpha show ?case by simp
  next
    case(cSym Ψ PR QR)
    thus ?case by(blast dest: bisimE)
  qed
qed

lemma bisimParPres:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   R :: ('a, 'b, 'c) psi

  assumes Ψ ⊳ P ∼ Q

  shows Ψ ⊳ P ∥ R ∼ Q ∥ R
proof −
  obtain A_R Ψ_R where extractFrame R = ⟨A_R, Ψ_R⟩ and A_R ♯* Ψ and A_R ♯* P
and A_R ♯* Q
    by(rule-tac C=(Ψ, P, Q) in freshFrame) auto
  moreover from ‹Ψ ⊳ P ∼ Q› have Ψ ⊗ Ψ_R ⊳ P ∼ Q by(rule bisimE)
  ultimately show ?thesis by(rule-tac bisimParPresAux)
qed

end

end

theory Sim-Struct-Cong
  imports Simulation
begin

lemma partitionListLeft:
  assumes xs@ys=xs'@y#ys'
  and     y mem xs
  and     distinct(xs@ys)

  obtains zs where xs = xs'@y#zs and ys'=zs@ys
using assms
by(auto simp add: append-eq-append-conv2 append-eq-Cons-conv)

lemma partitionListRight:
  assumes xs@ys=xs'@y#ys'
  and     y mem ys
  and     distinct(xs@ys)

  obtains zs where xs' = xs@zs and ys=zs@y#ys'
using assms
```

by(*force simp add*: *append-eq-append-conv2 append-eq-Cons-conv*)

context *env* **begin**

lemma *resComm*:
  fixes Ψ :: $'b$
  and x :: *name*
  and y :: *name*
  and *Rel* :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
  and P :: $('a, 'b, 'c)\ psi$

  assumes $x \sharp \Psi$
  and $y \sharp \Psi$
  and *eqvt Rel*
  and *R1*: $\bigwedge \Psi'\ Q.\ (\Psi', Q, Q) \in Rel$
  and *R2*: $\bigwedge \Psi'\ a\ b\ Q.\ [\![a \sharp \Psi'; b \sharp \Psi']\!] \Longrightarrow (\Psi', (\!|\nu a|\!)((\!|\nu b|\!)Q), (\!|\nu b|\!)((\!|\nu a|\!)Q)) \in Rel$

  shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \rightsquigarrow[Rel]\ (\!|\nu y|\!)((\!|\nu x|\!)P)$
proof(*case-tac* $x{=}y$)
  assume $x = y$
  thus *?thesis* **using** *R1*
    by(*force intro*: *reflexive*)
next
  assume $x \neq y$
  note ‹*eqvt Rel*›
  moreover from ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $[x, y] \sharp* \Psi$ **by**(*simp add*: *fresh-star-def*)
  moreover have $[x, y] \sharp* (\!|\nu x|\!)((\!|\nu y|\!)P)$ **by**(*simp add*: *abs-fresh*)
  moreover have $[x, y] \sharp* (\!|\nu y|\!)((\!|\nu x|\!)P)$ **by**(*simp add*: *abs-fresh*)
  ultimately show *?thesis*
  proof(*induct rule*: *simIChainFresh*[**where** $C{=}(x, y)$])
    case(*cSim* $\alpha\ P'$)
    from ‹$bn\ \alpha \sharp* (x, y)$› ‹$bn\ \alpha \sharp* ((\!|\nu x|\!)((\!|\nu y|\!)P))$› **have** $x \sharp bn\ \alpha$ **and** $y \sharp bn\ \alpha$ **and** $bn\ \alpha \sharp* P$ **by** *simp+*
    from ‹$[x, y] \sharp* \alpha$› **have** $x \sharp \alpha$ **and** $y \sharp \alpha$ **by** *simp+*
    from ‹$[x, y] \sharp* P'$› **have** $x \sharp P'$ **and** $y \sharp P'$ **by** *simp+*
    from ‹$bn\ \alpha \sharp* P$› ‹$x \sharp \alpha$› **have** $bn\ \alpha \sharp* (\!|\nu x|\!)P$ **by**(*simp add*: *abs-fresh*)
    with ‹$\Psi \rhd (\!|\nu y|\!)((\!|\nu x|\!)P) \longmapsto \alpha \prec P'$› ‹$y \sharp \Psi$› ‹$y \sharp \alpha$› ‹$y \sharp P'$› ‹$bn\ \alpha \sharp* \Psi$›
    show *?case* **using** ‹$bn\ \alpha \sharp*$ *subject* α› ‹$x \sharp \alpha$› ‹$x \sharp P'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹$y \sharp \alpha$›
    proof(*induct rule*: *resCases'*)
      case(*cOpen M yvec1 yvec2 y' N P'*)
    from ‹*yvec1* $\sharp*$ *yvec2*› ‹*distinct yvec1*› ‹*distinct yvec2*› **have** *distinct*(*yvec1*@*yvec2*) **by** *auto*
      from ‹$x \sharp M(\!|\nu*(yvec1\ @\ y'\ \#\ yvec2)|\!)\langle N\rangle$› **have** $x \sharp M$ **and** $x \sharp yvec1$ **and** $x \neq y'$ **and** $x \sharp yvec2$ **and** $x \sharp N$
        by *simp+*
      from ‹$y \sharp M(\!|\nu*(yvec1\ @\ y'\ \#\ yvec2)|\!)\langle N\rangle$› **have** $y \sharp M$ **and** $y \sharp yvec1$ **and** $y \sharp yvec2$

by *simp+*
from ‹$\Psi \rhd ([(y, y')] \cdot (\!|\nu x|\!)P) \longmapsto M(\!|\nu*(yvec1@yvec2)|\!)\langle N\rangle \prec P'$› ‹$x \neq y$› ‹$x \neq y'$›
have $\Psi \rhd (\!|\nu x|\!)([(y, y')] \cdot P) \longmapsto M(\!|\nu*(yvec1@yvec2)|\!)\langle N\rangle \prec P'$ **by**(*simp add*: *eqvts*)
moreover note ‹$x \sharp \Psi$›
moreover from ‹$x \sharp N$› ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$x \sharp M$› **have** $x \sharp M(\!|\nu*(yvec1@yvec2)|\!)\langle N\rangle$ **by** *simp*
moreover note ‹$x \sharp P'$›
moreover from ‹$yvec1 \sharp* \Psi$› ‹$yvec2 \sharp* \Psi$› **have** $bn(M(\!|\nu*(yvec1@yvec2)|\!)\langle N\rangle) \sharp* \Psi$ **by** *simp*
moreover from ‹$yvec1 \sharp* (\!|\nu x|\!)P$› ‹$yvec2 \sharp* (\!|\nu x|\!)P$› ‹$y \sharp yvec1$› ‹$y' \sharp yvec1$› ‹$y \sharp yvec2$› ‹$y' \sharp yvec2$› ‹$x \sharp yvec1$› ‹$x \sharp yvec2$›
have $bn(M(\!|\nu*(yvec1@yvec2)|\!)\langle N\rangle) \sharp* ([(y, y')] \cdot P)$ **by** *simp*
moreover from ‹$yvec1 \sharp* M$› ‹$yvec2 \sharp* M$› **have** $bn(M(\!|\nu*(yvec1 @ yvec2)|\!)\langle N\rangle) \sharp* subject(M(\!|\nu*(yvec1 @ yvec2)|\!)\langle N\rangle)$
by *simp*
moreover have $bn(M(\!|\nu*(yvec1 @ yvec2)|\!)\langle N\rangle) = yvec1@yvec2$ **by** *simp*
moreover have $subject(M(\!|\nu*(yvec1 @ yvec2)|\!)\langle N\rangle) = Some\ M$ **by** *simp*
moreover have $object(M(\!|\nu*(yvec1 @ yvec2)|\!)\langle N\rangle) = Some\ N$ **by** *simp*
ultimately show *?case*
proof(*induct rule*: *resCases'*)
case(*cOpen* M' *xvec1 xvec2* x' N' P')
from ‹$bn(M'(\!|\nu*(xvec1 @ x' \# xvec2)|\!)\langle N'\rangle) = yvec1 @ yvec2$› **have** $yvec1@yvec2 = xvec1@x'\#xvec2$ **by** *simp*
from ‹$subject(M'(\!|\nu*(xvec1 @ x' \# xvec2)|\!)\langle N'\rangle) = Some\ M$› **have** $M = M'$ **by** *simp*
from ‹$object(M'(\!|\nu*(xvec1 @ x' \# xvec2)|\!)\langle N'\rangle) = Some\ N$› **have** $N = N'$ **by** *simp*
from ‹$x \sharp yvec1$› ‹$x \sharp yvec2$› ‹$y' \sharp yvec1$› ‹$y' \sharp yvec2$› ‹$y \sharp yvec1$› ‹$y \sharp yvec2$›
have $x \sharp (yvec1@yvec2)$ **and** $y \sharp (yvec1@yvec2)$ **and** $y' \sharp (yvec1@yvec2)$ **by** *simp+*
with ‹$yvec1@yvec2 = xvec1@x'\#xvec2$›
have $x \sharp xvec1$ **and** $x \neq x'$ **and** $x \sharp xvec2$ **and** $y \sharp xvec1$ **and** $y \neq x'$ **and** $y \sharp xvec2$
and $y' \sharp xvec1$ **and** $x' \neq y'$ **and** $y' \sharp xvec2$
by *auto*

show *?case*
proof(*case-tac* x' *mem yvec1*)
assume x' *mem yvec1*

with ‹$yvec1@yvec2 = xvec1@x'\#xvec2$› ‹$distinct\ (yvec1@yvec2)$›
obtain $xvec2'$ **where** *Eq1*: $yvec1=xvec1@x'\#xvec2'$
and *Eq*: $xvec2=xvec2'@yvec2$
by(*rule-tac partitionListLeft*)
from ‹$\Psi \rhd ([(x, x')] \cdot [(y, y')] \cdot P) \longmapsto M'(\!|\nu*(xvec1@xvec2)|\!)\langle N'\rangle \prec P'$› ‹$y' \in supp\ N$› ‹$y' \sharp \Psi$› ‹$y' \sharp M$› ‹$y' \sharp xvec1$› ‹$y' \sharp xvec2$› *Eq* ‹$M=M'$› ‹$N = N'$›
have $\Psi \rhd (\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot P) \longmapsto M'(\!|\nu*((xvec1@xvec2')@y'\#yvec2)|\!)\langle N'\rangle$

$\prec P'$
by(*rule-tac Open*) *auto*
then have $\Psi \rhd (\!|\nu x'|\!)((\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot P)) \longmapsto M(\!|\nu*(xvec1@x'\#xvec2'@y'\#yvec2)|\!)\langle N\rangle$
$\prec P'$
using ‹$x' \in supp\ N'$› ‹$x' \sharp \Psi$› ‹$x' \sharp M'$› ‹$x' \sharp xvec1$› ‹$x' \sharp xvec2$› ‹$x' \neq y'$› *Eq* ‹$M{=}M'$› ‹$N{=}N'$›
by(*rule-tac Open*) *auto*
with ‹$x' \neq y'$› ‹$x \neq y'$› ‹$x' \sharp [(y, y')] \cdot P$›
have $\Psi \rhd (\!|\nu x|\!)((\!|\nu y'|\!)([(y, y')] \cdot P)) \longmapsto M(\!|\nu*(xvec1@x'\#xvec2'@y'\#yvec2)|\!)\langle N\rangle$
$\prec P'$
by(*subst alphaRes*[**where** $y{=}x'$]) (*simp add*: *calc-atm eqvts abs-fresh*)+
with *Eq1* ‹$y' \sharp (\!|\nu x|\!)P$› ‹$x \neq y'$› *R1* **show** *?case*
by(*fastforce simp add*: *alphaRes abs-fresh*)
next
assume $\neg x'$ *mem yvec1*
hence $x' \sharp yvec1$ **by**(*simp add*: *fresh-def*)
from ‹$\neg x'$ *mem yvec1*› ‹$yvec1@yvec2 = xvec1@x'\#xvec2$›
have x' *mem yvec2*
by(*fastforce simp add*: *append-eq-append-conv2 append-eq-Cons-conv*
fresh-list-append fresh-list-cons)
with ‹$yvec1@yvec2 = xvec1@x'\#xvec2$› ‹*distinct* $(yvec1@yvec2)$›
obtain $xvec2'$ **where** *Eq*: $xvec1{=}yvec1@xvec2'$
and *Eq1*: $yvec2{=}xvec2'@x'\#xvec2$
by(*rule-tac partitionListRight*)
from ‹$\Psi \rhd ([(x, x')] \cdot [(y, y')] \cdot P) \longmapsto M'(\!|\nu*(xvec1@xvec2)|\!)\langle N'\rangle \prec P'$› ‹$y' \in supp\ N$› ‹$y' \sharp \Psi$› ‹$y' \sharp M$› ‹$y' \sharp xvec1$› ‹$y' \sharp xvec2$› *Eq* ‹$M{=}M'$› ‹$N = N'$›
have $\Psi \rhd (\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot P) \longmapsto M'(\!|\nu*(yvec1@y'\#xvec2'@xvec2)|\!)\langle N'\rangle$
$\prec P'$
by(*rule-tac Open*) (*assumption* | *simp*)+
then have $\Psi \rhd (\!|\nu x'|\!)((\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot P)) \longmapsto M(\!|\nu*((yvec1@y'\#xvec2')@x'\#xvec2)|\!)\langle N\rangle$
$\prec P'$
using ‹$x' \in supp\ N'$› ‹$x' \sharp \Psi$› ‹$x' \sharp M'$› ‹$x' \sharp xvec1$› ‹$x' \sharp xvec2$› ‹$x' \neq y'$› *Eq* ‹$M{=}M'$› ‹$N{=}N'$›
by(*rule-tac Open*) *auto*
with ‹$x' \neq y'$› ‹$x \neq y'$› ‹$x' \sharp [(y, y')] \cdot P$›
have $\Psi \rhd (\!|\nu x|\!)((\!|\nu y'|\!)([(y, y')] \cdot P)) \longmapsto M(\!|\nu*((yvec1@y'\#xvec2')@x'\#xvec2)|\!)\langle N\rangle$
$\prec P'$
by(*subst alphaRes*[**where** $y{=}x'$]) (*simp add*: *calc-atm eqvts abs-fresh*)+
with *Eq1* ‹$y' \sharp (\!|\nu x|\!)P$› ‹$x \neq y'$› *R1* **show** *?case*
by(*fastforce simp add*: *alphaRes abs-fresh*)
qed
next
case(*cRes* P')
from ‹$\Psi \rhd ([(y, y')] \cdot P) \longmapsto M(\!|\nu*(yvec1@yvec2)|\!)\langle N\rangle \prec P'$› ‹$y' \in supp\ N$› ‹$y' \sharp \Psi$› ‹$y' \sharp M$› ‹$y' \sharp yvec1$› ‹$y' \sharp yvec2$›
have $\Psi \rhd (\!|\nu y'|\!)([(y, y')] \cdot P) \longmapsto M(\!|\nu*(yvec1@y'\#yvec2)|\!)\langle N\rangle \prec P'$ **by**(*rule Open*)
with ‹$y' \sharp (\!|\nu x|\!)P$› ‹$x \neq y'$›**have** $\Psi \rhd (\!|\nu y|\!)P \longmapsto M(\!|\nu*(yvec1@y'\#yvec2)|\!)\langle N\rangle$
$\prec P'$ **by**(*simp add*: *alphaRes abs-fresh*)

hence $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \longmapsto M(\!|\nu*(yvec1@y'\#yvec2)|\!)\langle N\rangle \prec (\!|\nu x|\!)P'$ **using** ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$x \sharp yvec1$› ‹$x \neq y'$› ‹$x \sharp yvec2$› ‹$x \sharp N$›
by(*rule-tac Scope*) *auto*
moreover have $(\Psi, (\!|\nu x|\!)P', (\!|\nu x|\!)P') \in Rel$ **by**(*rule R1*)
ultimately show *?case* **by** *blast*
qed
next
case(*cRes P'*)
from ‹$x \sharp (\!|\nu y|\!)P'$› ‹$x \neq y$› **have** $x \sharp P'$ **by**(*simp add*: *abs-fresh*)
with ‹$\Psi \rhd (\!|\nu x|\!)P \longmapsto\alpha \prec P'$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$›
show *?case* **using** ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$y \sharp \alpha$›
proof(*induct rule*: *resCases'*)
case(*cOpen M xvec1 xvec2 x' N P'*)
from ‹$y \sharp M(\!|\nu*(xvec1@x'\#xvec2)|\!)\langle N\rangle$› **have** $y \neq x'$ **and** $y \sharp M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle$ **by** *simp+*
from ‹$\Psi \rhd ([(x, x')] \cdot P) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec P'$› ‹$y \sharp \Psi$› ‹$y \sharp M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle$›
have $\Psi \rhd (\!|\nu y|\!)([(x, x')] \cdot P) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec (\!|\nu y|\!)P'$
by(*rule Scope*)
hence $\Psi \rhd (\!|\nu x'|\!)((\!|\nu y|\!)([(x, x')] \cdot P)) \longmapsto M(\!|\nu*(xvec1@x'\#xvec2)|\!)\langle N\rangle \prec (\!|\nu y|\!)P'$
using ‹$x' \in supp\ N$› ‹$x' \sharp \Psi$› ‹$x' \sharp M$› ‹$x' \sharp xvec1$› ‹$x' \sharp xvec2$›
by(*rule Open*)
with ‹$y \neq x'$› ‹$x \neq y$› ‹$x' \sharp P$› **have** $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \longmapsto M(\!|\nu*(xvec1@x'\#xvec2)|\!)\langle N\rangle \prec (\!|\nu y|\!)P'$
by(*subst alphaRes*[**where** *y=x'*]) (*simp add*: *abs-fresh eqvts calc-atm*)+
moreover have $(\Psi, (\!|\nu y|\!)P', (\!|\nu y|\!)P') \in Rel$ **by**(*rule R1*)
ultimately show *?case* **by** *blast*
next
case(*cRes P'*)
from ‹$\Psi \rhd P \longmapsto\alpha \prec P'$› ‹$y \sharp \Psi$› ‹$y \sharp \alpha$›
have $\Psi \rhd (\!|\nu y|\!)P \longmapsto\alpha \prec (\!|\nu y|\!)P'$ **by**(*rule Scope*)
hence $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \longmapsto\alpha \prec (\!|\nu x|\!)((\!|\nu y|\!)P')$ **using** ‹$x \sharp \Psi$› ‹$x \sharp \alpha$›
by(*rule Scope*)
moreover from ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $(\Psi, (\!|\nu x|\!)((\!|\nu y|\!)P'), (\!|\nu y|\!)((\!|\nu x|\!)P')) \in Rel$
by(*rule R2*)
ultimately show *?case* **by** *blast*
qed
qed
qed
qed

lemma *parAssocLeft*:
fixes Ψ :: *'b*
and P :: *('a, 'b, 'c) psi*
and Q :: *('a, 'b, 'c) psi*
and R :: *('a, 'b, 'c) psi*
and Rel :: *('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set*

assumes *eqvt Rel*
and *C1*: $\bigwedge\Psi'\ S\ T\ U.\ (\Psi,\ (S \parallel T) \parallel U,\ S \parallel (T \parallel U)) \in Rel$
and *C2*: $\bigwedge xvec\ \Psi'\ S\ T\ U.\ [\![xvec \sharp* \Psi';\ xvec \sharp* S]\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)((S \parallel T) \parallel U),\ S \parallel ((\!|\nu{*}xvec|\!)(T \parallel U))) \in Rel$
and *C3*: $\bigwedge xvec\ \Psi'\ S\ T\ U.\ [\![xvec \sharp* \Psi';\ xvec \sharp* U]\!] \Longrightarrow (\Psi',\ ((\!|\nu{*}xvec|\!)(S \parallel T)) \parallel U,\ (\!|\nu{*}xvec|\!)(S \parallel (T \parallel U))) \in Rel$
and *C4*: $\bigwedge\Psi'\ S\ T\ xvec.\ [\![(\Psi',\ S,\ T) \in Rel;\ xvec \sharp* \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)S,\ (\!|\nu{*}xvec|\!)T) \in Rel$

shows $\Psi \rhd (P \parallel Q) \parallel R \rightsquigarrow[Rel]\ P \parallel (Q \parallel R)$
using ‹*eqvt Rel*›
proof(*induct rule*: *simI*[*of* - - - - ()])
case(*cSim* α *PQR*)
from ‹$bn\ \alpha \sharp* (P \parallel Q \parallel R)$› **have** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp* R$ **by** *simp+*
hence $bn\ \alpha \sharp* (Q \parallel R)$ **by** *simp*
with ‹$\Psi \rhd P \parallel (Q \parallel R) \longmapsto\alpha \prec PQR$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$›
show *?case* **using** ‹$bn\ \alpha \sharp* subject\ \alpha$›
proof(*induct rule*: *parCases*[**where** $C = (\Psi,\ P,\ Q,\ R,\ \alpha)$])
case(*cPar1* $P'\ A_{QR}\ \Psi_{QR}$)
from ‹$A_{QR} \sharp* (\Psi,\ P,\ Q,\ R,\ \alpha)$› **have** $A_{QR} \sharp* Q$ **and** $A_{QR} \sharp* R$
by *simp+*
with ‹$extractFrame(Q \parallel R) = \langle A_{QR},\ \Psi_{QR}\rangle$› ‹$distinct\ A_{QR}$›
obtain $A_Q\ \Psi_Q\ A_R\ \Psi_R$ **where** $A_{QR} = A_Q@A_R$ **and** $\Psi_{QR} = \Psi_Q \otimes \Psi_R$ **and** *FrQ*: $extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$ **and** *FrR*: $extractFrame\ R = \langle A_R,\ \Psi_R\rangle$
and $A_Q \sharp* \Psi_R$ **and** $A_R \sharp* \Psi_Q$
by(*rule-tac mergeFrameE*) (*auto dest*: *extractFrameFreshChain*)

from ‹$A_{QR} = A_Q@A_R$› ‹$A_{QR} \sharp* \Psi$› ‹$A_{QR} \sharp* P$› ‹$A_{QR} \sharp* Q$› ‹$A_{QR} \sharp* \alpha$›
have $A_Q \sharp* \Psi$ **and** $A_R \sharp* \Psi$ **and** $A_Q \sharp* P$ **and** $A_R \sharp* P$ **and** $A_Q \sharp* Q$ **and** $A_R \sharp* Q$ **and** $A_Q \sharp* \alpha$ **and** $A_R \sharp* \alpha$
by *simp+*

from ‹$\Psi \otimes \Psi_{QR} \rhd P \longmapsto\alpha \prec P'$› ‹$\Psi_{QR} = \Psi_Q \otimes \Psi_R$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$
by(*metis statEqTransition Associativity Commutativity Composition*)
hence $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto\alpha \prec (P' \parallel Q)$ **using** *FrQ* ‹$bn\ \alpha \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_R$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$›
by(*rule-tac Par1*) *auto*
hence $\Psi \rhd (P \parallel Q) \parallel R \longmapsto\alpha \prec ((P' \parallel Q) \parallel R)$ **using** *FrR* ‹$bn\ \alpha \sharp* R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* \alpha$›
by(*rule-tac Par1*) *auto*
moreover have $(\Psi,\ (P' \parallel Q) \parallel R,\ P' \parallel (Q \parallel R)) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 QR* $A_P\ \Psi_P$)
from ‹$A_P \sharp* (\Psi,\ P,\ Q,\ R,\ \alpha)$› **have** $A_P \sharp* Q$ **and** $A_P \sharp* R$ **and** $A_P \sharp* \alpha$
by *simp+*

have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P \rangle$ **by** *fact*
with ‹*bn* α $\sharp*$ P› ‹A_P $\sharp*$ α› **have** *bn* α $\sharp*$ Ψ_P **by**(*auto dest*: *extractFrameFreshChain*)
with ‹*bn* α $\sharp*$ Ψ› **have** *bn* α $\sharp*$ $(\Psi \otimes \Psi_P)$ **by** *force*
with ‹$\Psi \otimes \Psi_P \rhd Q \parallel R \longmapsto\alpha \prec QR$›
show *?case* **using** ‹*bn* α $\sharp*$ Q› ‹*bn* α $\sharp*$ R› ‹*bn* α $\sharp*$ *subject* α› ‹A_P $\sharp*$ Q› ‹A_P $\sharp*$ R›
proof(*induct rule*: *parCasesSubject*[**where** $C = (A_P, \Psi_P, P, Q, R, \Psi)$])
case(*cPar1* Q' A_R Ψ_R)
from ‹A_R $\sharp*$ $(A_P, \Psi_P, P, Q, R, \Psi)$› **have** A_R $\sharp*$ A_P **and** A_R $\sharp*$ P **and** A_R $\sharp*$ Q **and** A_R $\sharp*$ Ψ_P **and** A_R $\sharp*$ Ψ
by *simp*+
from ‹A_P $\sharp*$ R› ‹*extractFrame* $R = \langle A_R, \Psi_R \rangle$› ‹$A_R$ $\sharp*$ A_P› **have** A_P $\sharp*$ Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_R \rhd Q \longmapsto\alpha \prec Q'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \rhd Q \longmapsto\alpha \prec Q'$
by(*metis statEqTransition Associativity Commutativity Composition*)
hence $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto\alpha \prec (P \parallel Q')$
using *FrP* ‹*bn* α $\sharp*$ P› ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ_R› ‹A_P $\sharp*$ Q› ‹A_P $\sharp*$ α›
by(*rule-tac Par2*) (*assumption* | *force*)+
hence $\Psi \rhd (P \parallel Q) \parallel R \longmapsto\alpha \prec ((P \parallel Q') \parallel R)$
using ‹*extractFrame* $R = \langle A_R, \Psi_R \rangle$› ‹*bn* α $\sharp*$ R› ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ P› ‹A_R $\sharp*$ Q› ‹A_R $\sharp*$ α›
by(*rule-tac Par1*) (*assumption* | *simp*)+
moreover have $(\Psi, (P \parallel Q') \parallel R, P \parallel (Q' \parallel R)) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* R' A_Q Ψ_Q)
from ‹A_Q $\sharp*$ $(A_P, \Psi_P, P, Q, R, \Psi)$› **have** A_Q $\sharp*$ A_P **and** A_Q $\sharp*$ R **and** A_Q $\sharp*$ Ψ_P **and** A_Q $\sharp*$ Ψ
by *simp*+
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$ **by** *fact*
from ‹A_P $\sharp*$ Q› *FrQ* ‹A_Q $\sharp*$ A_P› **have** A_P $\sharp*$ Ψ_Q
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_Q \rhd R \longmapsto\alpha \prec R'$›
have $\Psi \otimes (\Psi_P \otimes \Psi_Q) \rhd R \longmapsto\alpha \prec R'$
by(*blast intro*: *statEqTransition Associativity*)
moreover from *FrP FrQ* ‹A_Q $\sharp*$ A_P› ‹A_P $\sharp*$ Ψ_Q› ‹A_Q $\sharp*$ Ψ_P›
have *extractFrame*$(P \parallel Q) = \langle (A_P @ A_Q), \Psi_P \otimes \Psi_Q \rangle$ **by** *simp*
moreover from ‹*bn* α $\sharp*$ P› ‹*bn* α $\sharp*$ Q› **have** *bn* α $\sharp*$ $(P \parallel Q)$ **by** *simp*
moreover from ‹A_P $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ› **have** $(A_P @ A_Q)$ $\sharp*$ Ψ **by** *simp*
moreover from ‹A_P $\sharp*$ R› ‹A_Q $\sharp*$ R› **have** $(A_P @ A_Q)$ $\sharp*$ R **by** *simp*
moreover from ‹A_P $\sharp*$ α› ‹A_Q $\sharp*$ α› **have** $(A_P @ A_Q)$ $\sharp*$ α **by** *simp*
ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto\alpha \prec ((P \parallel Q) \parallel R')$
by(*rule Par2*)
moreover have $(\Psi, (P \parallel Q) \parallel R', P \parallel (Q \parallel R')) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_R M N Q' A_Q Ψ_Q K *xvec* R' A_R)

from ‹A_Q ♯* (A_P, Ψ_P, P, Q, R, Ψ)›
have A_Q ♯* P **and** A_Q ♯* Q **and** A_Q ♯* R **and** A_Q ♯* A_P **and** A_Q ♯* Ψ_P **and** A_Q ♯* Ψ **by** *simp+*
from ‹A_R ♯* (A_P, Ψ_P, P, Q, R, Ψ)› **have** A_R ♯* P **and** A_R ♯* Q **and** A_R ♯* R **and** A_R ♯* A_P **and** A_R ♯* Ψ **by** *simp+*
from ‹*xvec* ♯* (A_P, Ψ_P, P, Q, R, Ψ)› **have** *xvec* ♯* A_P **and** *xvec* ♯* P **and** *xvec* ♯* Q **and** *xvec* ♯* Ψ **by** *simp+*

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
with ‹A_P ♯* Q› ‹A_Q ♯* A_P› **have** A_P ♯* Ψ_Q
by(*drule-tac extractFrameFreshChain*) *auto*
have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
with ‹A_P ♯* R› ‹A_R ♯* A_P› **have** A_P ♯* Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*
from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$› ‹$A_P$ ♯* R› ‹*xvec* ♯* A_P› ‹*xvec* ♯* K› ‹*distinct xvec*› **have** A_P ♯* N
by(*rule-tac outputFreshChainDerivative*) *auto*

from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_R \rhd Q \longmapsto M(\!|N|\!) \prec Q'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q'$
by(*metis statEqTransition Associativity Commutativity Composition*)
hence $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto M(\!|N|\!) \prec (P \parallel Q')$ **using** *FrP* ‹A_P ♯* Ψ› ‹A_P ♯* Ψ_R› ‹A_P ♯* Q› ‹A_P ♯* M› ‹A_P ♯* N›
by(*rule-tac Par2*) *auto*
moreover from *FrP FrQ* ‹A_P ♯* Ψ_Q› ‹A_Q ♯* A_P› ‹A_Q ♯* Ψ_P› **have** *extractFrame*$(P \parallel Q) = \langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle$
by *simp*
moreover from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$› **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$
by(*metis statEqTransition Associativity*)
moreover note ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$›
moreover from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$› **have** $\Psi \otimes (\Psi_P \otimes \Psi_Q) \otimes \Psi_R \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Commutativity Composition*)
ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto \tau \prec (\!|\nu{*}xvec|\!)((P \parallel Q') \parallel R')$
using ‹A_P ♯* Ψ› ‹A_Q ♯* Ψ› ‹A_R ♯* Ψ› ‹A_P ♯* P› ‹A_Q ♯* P› ‹A_R ♯* P› ‹A_P ♯* Q› ‹A_Q ♯* Q› ‹A_R ♯* Q› ‹A_P ♯* R› ‹A_Q ♯* R› ‹A_R ♯* R›
‹A_P ♯* M› ‹A_Q ♯* M› ‹A_R ♯* K› ‹A_R ♯* A_P› ‹A_Q ♯* A_R› ‹*xvec* ♯* P› ‹*xvec* ♯* Q›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
moreover from ‹*xvec* ♯* Ψ› ‹*xvec* ♯* P› **have** $(\Psi, (\!|\nu{*}xvec|\!)((P \parallel Q') \parallel R'), P \parallel ((\!|\nu{*}xvec|\!)(Q' \parallel R'))) \in Rel$
by(*rule C2*)
ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_R M *xvec* N Q' A_Q Ψ_Q K R' A_R)
from ‹A_Q ♯* (A_P, Ψ_P, P, Q, R, Ψ)›
have A_Q ♯* P **and** A_Q ♯* Q **and** A_Q ♯* R **and** A_Q ♯* A_P **and** A_Q ♯* Ψ **and** A_Q ♯* Ψ_P **by** *simp+*

from ‹A_R ♯* (A_P, Ψ_P, P, Q, R, Ψ)› **have** A_R ♯* P **and** A_R ♯* Q **and** A_R ♯* R **and** A_R ♯* A_P**and** A_R ♯* Ψ **by** *simp+*
from ‹*xvec* ♯* (A_P, Ψ_P, P, Q, R, Ψ)› **have** *xvec* ♯* A_P **and** *xvec* ♯* P **and** *xvec* ♯* Q **and** *xvec* ♯* Ψ **by** *simp+*

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
with ‹A_P ♯* Q› ‹A_Q ♯* A_P› **have** A_P ♯* Ψ_Q
by(*drule-tac extractFrameFreshChain*) *auto*
have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
with ‹A_P ♯* R› ‹A_R ♯* A_P› **have** A_P ♯* Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*

from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_R \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› ‹$A_P$ ♯* Q› ‹*xvec* ♯* A_P› ‹*xvec* ♯* M› ‹*distinct xvec*› **have** A_P ♯* N
by(*rule-tac outputFreshChainDerivative*) *auto*

from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_R \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
by(*metis statEqTransition Associativity Commutativity Composition*)
hence $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec (P \parallel Q')$ **using** *FrP* ‹A_P ♯* Ψ› ‹A_P ♯* Ψ_R› ‹A_P ♯* Q› ‹A_P ♯* M› ‹A_P ♯* N› ‹*xvec* ♯* P› ‹*xvec* ♯* A_P›
by(*rule-tac Par2*) *auto*
moreover from *FrP FrQ* ‹A_P ♯* Ψ_Q› ‹A_Q ♯* A_P› ‹A_Q ♯* Ψ_P› **have** *extractFrame*$(P \parallel Q) = \langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle$
by *simp+*
moreover from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$› **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$
by(*metis statEqTransition Associativity*)
moreover note ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$›
moreover from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$› **have** $\Psi \otimes (\Psi_P \otimes \Psi_Q) \otimes \Psi_R \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Commutativity Composition*)
ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto \tau \prec (\!|\nu{*}xvec|\!)((P \parallel Q') \parallel R')$
using ‹A_P ♯* Ψ› ‹A_Q ♯* Ψ› ‹A_R ♯* Ψ› ‹A_P ♯* P› ‹A_Q ♯* P› ‹A_R ♯* P› ‹A_P ♯* Q› ‹A_Q ♯* Q› ‹A_R ♯* Q› ‹A_P ♯* R› ‹A_Q ♯* R› ‹A_R ♯* R›
‹A_P ♯* M› ‹A_Q ♯* M› ‹A_R ♯* K› ‹A_R ♯* A_P› ‹A_Q ♯* A_R› ‹*xvec* ♯* R›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
moreover from ‹*xvec* ♯* Ψ› ‹*xvec* ♯* P› **have** $(\Psi, (\!|\nu{*}xvec|\!)((P \parallel Q') \parallel R'), P \parallel ((\!|\nu{*}xvec|\!)(Q' \parallel R'))) \in Rel$
by(*rule C2*)
ultimately show *?case* **by** *blast*
qed
next
case(*cComm1* Ψ_{QR} M N P' A_P Ψ_P K *xvec* QR' A_{QR})
from ‹*xvec* ♯* (Ψ, P, Q, R, α)› **have** *xvec* ♯* Q **and** *xvec* ♯* R **by** *simp+*
from ‹A_{QR} ♯* (Ψ, P, Q, R, α)› **have** A_{QR} ♯* Q **and** A_{QR} ♯* R **and** A_{QR} ♯* Ψ **by** *simp+*
from ‹A_P ♯* $(Q \parallel R)$› **have** A_P ♯* Q **and** A_P ♯* R **by** *simp+*
have *PTrans*: $\Psi \otimes \Psi_{QR} \rhd P \longmapsto M(\!|N|\!) \prec P'$ **and** *FrP*: *extractFrame* $P =$

$\langle A_P, \Psi_P\rangle$ **and** *MeqK*: $\Psi \otimes \Psi_P \otimes \Psi_{QR} \vdash M \leftrightarrow K$ **by** *fact+*
note ‹$\Psi \otimes \Psi_P \rhd Q \parallel R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec QR'$›
moreover from ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ Ψ_P› **have** *xvec* $\sharp*$ $(\Psi \otimes \Psi_P)$ **by** *force*
moreover note ‹*xvec* $\sharp*$ Q›‹*xvec* $\sharp*$ R› ‹*xvec* $\sharp*$ K›
‹*extractFrame*$(Q \parallel R) = \langle A_{QR}, \Psi_{QR}\rangle$› ‹*distinct* A_{QR}›
moreover from ‹A_{QR} $\sharp*$ Ψ› ‹A_{QR} $\sharp*$ Ψ_P› **have** A_{QR} $\sharp*$ $(\Psi \otimes \Psi_P)$ **by** *force*
ultimately show *?case* **using** ‹A_{QR} $\sharp*$ Q› ‹A_{QR} $\sharp*$ R› ‹A_{QR} $\sharp*$ K›
proof(*induct rule*: *parCasesOutputFrame*)
case(*cPar1* Q' A_Q Ψ_Q A_R Ψ_R)
have *Aeq*: $A_{QR} = A_Q@A_R$ **and** Ψ*eq*: $\Psi_{QR} = \Psi_Q \otimes \Psi_R$ **by** *fact+*
from *PTrans* Ψ*eq* **have** $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$
by(*metis statEqTransition Associativity Commutativity Composition*)
moreover note *FrP*
moreover from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_R \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$
by(*metis statEqTransition Associativity Commutativity Composition*)
moreover note ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$›
moreover from *MeqK* Ψ*eq* **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Commutativity Composition*)
moreover from ‹A_P $\sharp*$ R› ‹A_P $\sharp*$ A_{QR}› *Aeq* ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› **have** A_P $\sharp*$ A_Q **and** A_P $\sharp*$ Ψ_R
by(*auto dest*: *extractFrameFreshChain*)
moreover from ‹A_{QR} $\sharp*$ P› ‹A_{QR} $\sharp*$ Ψ› *Aeq* **have** A_Q $\sharp*$ P **and** A_Q $\sharp*$ Ψ **by** *simp+*
ultimately have $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q')$
using ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ P› ‹A_P $\sharp*$ Q› ‹A_P $\sharp*$ M› ‹A_P $\sharp*$ A_Q› ‹A_Q $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ_R› ‹A_Q $\sharp*$ P› ‹A_Q $\sharp*$ Q› ‹A_Q $\sharp*$ K› ‹*xvec* $\sharp*$ P›
by(*rule-tac Comm1*) (*assumption* | *force*)+
moreover from ‹A_{QR} $\sharp*$ Ψ› *Aeq* **have** A_R $\sharp*$ Ψ **by** *simp*
moreover from ‹A_{QR} $\sharp*$ P› *Aeq* **have** A_R $\sharp*$ P **by** *simp*
ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto \tau \prec ((\!|\nu{*}xvec|\!)(P' \parallel Q')) \parallel R$ **using** ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› ‹$A_R$ $\sharp*$ Q›
by(*rule-tac Par1*) (*assumption* | *simp*)+
moreover from ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ R› **have** $(\Psi, ((\!|\nu{*}xvec|\!)(P' \parallel Q')) \parallel R, (\!|\nu{*}xvec|\!)(P' \parallel (Q' \parallel R))) \in Rel$
by(*rule C3*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* R' A_Q Ψ_Q A_R Ψ_R)
have *Aeq*: $A_{QR} = A_Q@A_R$ **and** Ψ*eq*: $\Psi_{QR} = \Psi_Q \otimes \Psi_R$ **by** *fact+*
from ‹A_{QR} $\sharp*$ Ψ› ‹A_{QR} $\sharp*$ P› ‹A_{QR} $\sharp*$ Ψ_P› ‹A_P $\sharp*$ A_{QR}› *Aeq* **have** A_R $\sharp*$ Ψ **and** A_R $\sharp*$ Ψ_P **and** A_P $\sharp*$ A_R **and** A_P $\sharp*$ A_Q **and** A_R $\sharp*$ P **by** *simp+*
from ‹A_{QR} $\sharp*$ Ψ› *Aeq* **have** A_Q $\sharp*$ Ψ **by** *simp*
from ‹A_{QR} $\sharp*$ P› ‹A_P $\sharp*$ A_{QR}› *Aeq FrP* **have** A_Q $\sharp*$ Ψ_P **by**(*auto dest*: *extractFrameFreshChain*)
from ‹A_P $\sharp*$ A_{QR}› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› *Aeq* ‹$A_P$ $\sharp*$ Q› ‹A_P $\sharp*$ R› **have** A_P $\sharp*$ Ψ_Q **and** A_P $\sharp*$ Ψ_R **by**(*auto dest*: *extractFrameFreshChain*)
have *RTrans*: $(\Psi \otimes \Psi_P) \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$ **and** *FrR*:

extractFrame $R = \langle A_R, \Psi_R\rangle$ **by** *fact+*

then obtain K' **where** *KeqK'*: $((\Psi \otimes \Psi_P) \otimes \Psi_Q) \otimes \Psi_R \vdash K \leftrightarrow K'$ **and** $A_P \sharp* K'$ **and** $A_Q \sharp* K'$

using ‹$A_P \sharp* R$› ‹$A_Q \sharp* R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P$› ‹$A_R \sharp* \Psi_Q$› ‹$A_Q \sharp* A_R$› ‹$A_P \sharp* A_R$› ‹$A_R \sharp* R$› ‹$A_R \sharp* R$› ‹$A_R \sharp* K$› ‹*distinct* A_R› ‹*xvec* $\sharp* K$› ‹*distinct xvec*›

by(*rule-tac* $B{=}A_P@A_Q$ **in** *obtainPrefix*) (*assumption* | *force*)+

from *PTrans* Ψ*eq* **have** $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$

by(*metis statEqTransition Associativity Commutativity Composition*)

moreover from *MeqK KeqK'* Ψ*eq* **have** *MeqK'*: $((\Psi \otimes \Psi_R) \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow K'$

by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans*)

ultimately have $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto K'(\!|N|\!) \prec P'$ **using** *FrP* ‹*distinct* A_P› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K'$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* \Psi_R$›

by(*rule-tac inputRenameSubject*) *auto*

moreover from ‹$A_{QR} \sharp* P$› ‹$A_{QR} \sharp* N$› *Aeq* **have** $A_Q \sharp* P$ **and** $A_Q \sharp* N$ **by** *simp+*

ultimately have $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto K'(\!|N|\!) \prec P' \parallel Q$ **using** ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp* \Psi_R$› ‹$A_Q \sharp* K'$› ‹$A_Q \sharp* \Psi$›

by(*rule-tac Par1*) (*assumption* | *force*)+

moreover from *FrP* ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›

have *extractFrame*$(P \parallel Q) = \langle (A_P@A_Q), \Psi_P \otimes \Psi_Q\rangle$ **by** *simp+*

moreover from *RTrans* **have** $\Psi \otimes (\Psi_P \otimes \Psi_Q) \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$ **by**(*metis Associativity statEqTransition*)

moreover note *FrR*

moreover from *MeqK' KeqK'* **have** $\Psi \otimes (\Psi_P \otimes \Psi_Q) \otimes \Psi_R \vdash K' \leftrightarrow K$

by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans chanEqSym*)

ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto \tau \prec (\!|\nu{*}xvec|\!)((P' \parallel Q) \parallel R')$

using ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_P \sharp* R$› ‹$A_Q \sharp* R$› ‹$A_P \sharp* K'$› ‹$A_Q \sharp* K'$› ‹$A_P \sharp* A_R$› ‹$A_Q \sharp* A_R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* K$› ‹*xvec* $\sharp* P$› ‹*xvec* $\sharp* Q$›

by(*rule-tac Comm1*) (*assumption* | *simp*)+

moreover from ‹*xvec* $\sharp* \Psi$› **have** $(\Psi, (\!|\nu{*}xvec|\!)((P' \parallel Q) \parallel R'), (\!|\nu{*}xvec|\!)(P' \parallel (Q \parallel R'))) \in$ *Rel*

by(*metis C1 C4*)

ultimately show *?case* **by** *blast*

qed

next

case(*cComm2* Ψ_{QR} *M xvec N* P' A_P Ψ_P *K* QR' A_{QR})

from ‹$A_{QR} \sharp* (\Psi, P, Q, R, \alpha)$› **have** $A_{QR} \sharp* Q$ **and** $A_{QR} \sharp* R$ **and** $A_{QR} \sharp* \Psi$ **by** *simp+*

from ‹$A_P \sharp* (Q \parallel R)$› ‹*xvec* $\sharp* (Q \parallel R)$› **have** $A_P \sharp* Q$ **and** $A_P \sharp* R$ **and** *xvec* $\sharp* Q$ **and** *xvec* $\sharp* R$ **by** *simp+*

have *PTrans*: $\Psi \otimes \Psi_{QR} \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **and** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** *MeqK*: $\Psi \otimes \Psi_P \otimes \Psi_{QR} \vdash M \leftrightarrow K$ **by** *fact+*

note ‹$\Psi \otimes \Psi_P \rhd Q \parallel R \longmapsto K(\!|N|\!) \prec QR'$› ‹*extractFrame*$(Q \parallel R) = \langle A_{QR},$

$\Psi_{QR}\rangle$› ‹*distinct* A_{QR}›
moreover from ‹$A_{QR} \sharp* \Psi$› ‹$A_{QR} \sharp* \Psi_P$› **have** $A_{QR} \sharp* (\Psi \otimes \Psi_P)$ **by** *force*
ultimately show *?case* **using** ‹$A_{QR} \sharp* Q$› ‹$A_{QR} \sharp* R$› ‹$A_{QR} \sharp* K$›
proof(*induct rule*: *parCasesInputFrame*)
case(*cPar1* Q' A_Q Ψ_Q A_R Ψ_R)
have *Aeq*: $A_{QR} = A_Q@A_R$ **and** Ψ*eq*: $\Psi_{QR} = \Psi_Q \otimes \Psi_R$ **by** *fact+*
from *PTrans* Ψ*eq* **have** $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*metis statEqTransition Associativity Commutativity Composition*)
moreover note *FrP*
moreover from ‹$(\Psi \otimes \Psi_P) \otimes \Psi_R \rhd Q \longmapsto K(\!|N|\!) \prec Q'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$
by(*metis statEqTransition Associativity Commutativity Composition*)
moreover note ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$›
moreover from *MeqK* Ψ*eq* **have** $(\Psi \otimes \Psi_R) \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$
by(*metis statEqEnt Associativity Commutativity Composition*)
moreover from ‹$A_P \sharp* Q$› ‹$A_P \sharp* R$› ‹$A_P \sharp* A_{QR}$› *Aeq* ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$›
have $A_P \sharp* A_Q$ **and** $A_P \sharp* \Psi_R$ **by**(*auto dest*: *extractFrameFreshChain*)
moreover from ‹$A_{QR} \sharp* \Psi$› ‹$A_{QR} \sharp* P$› *Aeq* **have** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* P$ **by** *simp+*
ultimately have $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto \tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$
using ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* M$› ‹$A_P \sharp* A_Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_R$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹*xvec* $\sharp* Q$›
by(*rule-tac Comm2*) (*assumption* | *force*)+
moreover from ‹$A_{QR} \sharp* \Psi$› ‹$A_{QR} \sharp* P$› *Aeq* **have** $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **by** *simp+*
ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto \tau \prec ((\!|\nu*xvec|\!)(P' \parallel Q')) \parallel R$ **using** ‹*extractFrame* $R = \langle A_R, \Psi_R\rangle$› ‹$A_R \sharp* Q$›
by(*rule-tac Par1*) (*assumption* | *simp*)+
moreover from ‹*xvec* $\sharp* \Psi$› ‹*xvec* $\sharp* R$› **have** $(\Psi, ((\!|\nu*xvec|\!)(P' \parallel Q')) \parallel R, (\!|\nu*xvec|\!)(P' \parallel (Q' \parallel R))) \in Rel$
by(*rule C3*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* R' A_Q Ψ_Q A_R Ψ_R)
have *Aeq*: $A_{QR} = A_Q@A_R$ **and** Ψ*eq*: $\Psi_{QR} = \Psi_Q \otimes \Psi_R$ **by** *fact+*
from ‹$A_{QR} \sharp* \Psi$› ‹$A_{QR} \sharp* P$› ‹$A_{QR} \sharp* \Psi_P$› ‹$A_P \sharp* A_{QR}$› *Aeq*
have $A_R \sharp* \Psi$ **and** $A_R \sharp* \Psi_P$ **and** $A_P \sharp* A_R$ **and** $A_P \sharp* A_Q$ **and** $A_R \sharp* P$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* \Psi_P$ **by** *simp+*
have *RTrans*: $(\Psi \otimes \Psi_P) \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$ **and** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact+*
then obtain K' **where** *KeqK'*: $((\Psi \otimes \Psi_P) \otimes \Psi_Q) \otimes \Psi_R \vdash K \leftrightarrow K'$ **and** $A_P \sharp* K'$ **and** $A_Q \sharp* K'$
using ‹$A_P \sharp* R$› ‹$A_Q \sharp* R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P$› ‹$A_R \sharp* \Psi_Q$› ‹$A_Q \sharp* A_R$› ‹$A_P \sharp* A_R$› ‹$A_R \sharp* R$› ‹$A_R \sharp* R$› ‹$A_R \sharp* K$› ‹*distinct* A_R›
by(*rule-tac* B=$A_P@A_Q$ **in** *obtainPrefix*) (*assumption* | *force*)+
from *PTrans* Ψ*eq* **have** $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*metis statEqTransition Associativity Commutativity Composition*)
moreover from *MeqK KeqK'* Ψ*eq* **have** *MeqK'*: $((\Psi \otimes \Psi_R) \otimes \Psi_Q) \otimes \Psi_P \vdash$

$M \leftrightarrow K'$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans*)
moreover from ‹$A_P \sharp* R$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* A_{QR}$› *FrR* ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› *Aeq* **have** $A_P \sharp* \Psi_Q$ **and** $A_P \sharp* \Psi_R$
by(*auto dest*: *extractFrameFreshChain*)
ultimately have $(\Psi \otimes \Psi_R) \otimes \Psi_Q \rhd P \longmapsto K'(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **using** *FrP* ‹*distinct* A_P› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* K'$›
by(*rule-tac outputRenameSubject*) *auto*
moreover from ‹$A_{QR} \sharp* P$› ‹$A_{QR} \sharp* N$› ‹$A_{QR} \sharp* xvec$› *Aeq* **have** $A_Q \sharp* P$ **and** $A_Q \sharp* N$ **and** $A_Q \sharp* xvec$ **by** *simp+*
ultimately have $\Psi \otimes \Psi_R \rhd P \parallel Q \longmapsto K'(\!|\nu*xvec|\!)\langle N\rangle \prec (P' \parallel Q)$ **using** ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_Q \sharp* \Psi_R$› ‹$A_Q \sharp* K'$› ‹$xvec \sharp* Q$› ‹$A_Q \sharp* \Psi$›
by(*rule-tac Par1*) (*assumption* | *force*)+
moreover from *FrP* ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_P$›
have *extractFrame*$(P \parallel Q) = \langle (A_P @ A_Q), \Psi_P \otimes \Psi_Q\rangle$ **by** *simp+*
moreover from *RTrans* **have** $\Psi \otimes (\Psi_P \otimes \Psi_Q) \rhd R \longmapsto K(\!|N|\!) \prec R'$ **by**(*metis Associativity statEqTransition*)
moreover note *FrR*
moreover from *MeqK′ KeqK′* **have** $\Psi \otimes (\Psi_P \otimes \Psi_Q) \otimes \Psi_R \vdash K' \leftrightarrow K$
by(*metis statEqEnt Associativity Commutativity Composition chanEqTrans chanEqSym*)
ultimately have $\Psi \rhd (P \parallel Q) \parallel R \longmapsto \tau \prec (\!|\nu*xvec|\!)((P' \parallel Q) \parallel R')$
using ‹$A_P \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* Q$› ‹$A_P \sharp* R$› ‹$A_Q \sharp* R$› ‹$A_P \sharp* K'$› ‹$A_Q \sharp* K'$› ‹$A_P \sharp* A_R$› ‹$A_Q \sharp* A_R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* K$› ‹$xvec \sharp* R$›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
moreover from ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu*xvec|\!)((P' \parallel Q) \parallel R'), (\!|\nu*xvec|\!)(P' \parallel (Q \parallel R'))) \in Rel$
by(*metis C1 C4*)
ultimately show *?case* **by** *blast*
qed
qed
qed

lemma *parNilLeft*:
fixes Ψ :: *′b*
and P :: (*′a*, *′b*, *′c*) *psi*
and *Rel* :: (*′b* × (*′a*, *′b*, *′c*) *psi* × (*′a*, *′b*, *′c*) *psi*) *set*

assumes *eqvt Rel*
and *C1*: $\bigwedge Q.\ (\Psi, Q \parallel \mathbf{0}, Q) \in Rel$

shows $\Psi \rhd (P \parallel \mathbf{0}) \rightsquigarrow[Rel]\ P$
using ‹*eqvt Rel*›
proof(*induct rule*: *simI*[*of* - - - - ()])
case(*cSim* α P')
from ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› **have** $\Psi \otimes SBottom' \rhd P \longmapsto \alpha \prec P'$
by(*metis statEqTransition Identity AssertionStatEqSym*)

hence $\Psi \rhd (P \parallel \mathbf{0}) \longmapsto\alpha \prec (P' \parallel \mathbf{0})$
by(*rule-tac Par1*) *auto*
moreover have $(\Psi,\ P' \parallel \mathbf{0},\ P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *parNilRight*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *eqvt Rel*
and *C1*: $\bigwedge Q.\ (\Psi,\ Q,\ (Q \parallel \mathbf{0})) \in Rel$

shows $\Psi \rhd P \rightsquigarrow[Rel]\ (P \parallel \mathbf{0})$
using ‹*eqvt Rel*›
proof(*induct rule*: *simI*[*of* - - - - ()])
case(*cSim* α P')
note ‹$\Psi \rhd P \parallel \mathbf{0} \longmapsto\alpha \prec P'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$›
moreover have $bn\ \alpha \sharp* \mathbf{0}$ **by** *simp*
ultimately show *?case* **using** ‹$bn\ \alpha \sharp*\ subject\ \alpha$›
proof(*induct rule*: *parCases*[**where** C=()])
case(*cPar1* $P'\ A_Q\ \Psi_Q$)
from ‹$extractFrame(\mathbf{0}) = \langle A_Q,\ \Psi_Q\rangle$› **have** $\Psi_Q = SBottom'$ **by** *auto*
with ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$› **have** $\Psi \rhd P \longmapsto\alpha \prec P'$
by(*metis statEqTransition Identity*)
moreover have $(\Psi,\ P',\ P' \parallel \mathbf{0}) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* $Q'\ A_P\ \Psi_P$)
from ‹$\Psi \otimes \Psi_P \rhd \mathbf{0} \longmapsto\alpha \prec Q'$› **have** *False*
by *auto*
thus *?case* **by** *simp*
next
case(*cComm1* $\Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K\ xvec\ Q'\ A_Q$)
from ‹$\Psi \otimes \Psi_P \rhd \mathbf{0} \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› **have** *False* **by** *auto*
thus *?case* **by** *simp*
next
case(*cComm2* $\Psi_Q\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ K\ Q'\ A_Q$)
from ‹$\Psi \otimes \Psi_P \rhd \mathbf{0} \longmapsto K(\!|N|\!) \prec Q'$› **have** *False*
by *auto*
thus *?case* **by** *simp*
qed
qed

lemma *resNilLeft*:
fixes x :: *name*
and Ψ :: $'b$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

shows $\Psi \rhd (\!|\nu x|\!)\mathbf{0} \rightsquigarrow[Rel]\ \mathbf{0}$
by(*auto simp add*: *simulation-def*)

lemma *resNilRight*:
fixes x :: *name*
and Ψ :: $'b$
and *Rel* :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

shows $\Psi \rhd \mathbf{0} \rightsquigarrow[Rel]\ (\!|\nu x|\!)\mathbf{0}$
apply(*auto simp add*: *simulation-def*)
by(*cases rule*: *semantics.cases*) (*auto simp add*: *psi.inject alpha'*)

lemma *inputPushResLeft*:
fixes Ψ :: $'b$
and x :: *name*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)\ psi$

assumes *eqvt Rel*
and $x \sharp \Psi$
and $x \sharp M$
and $x \sharp xvec$
and $x \sharp N$
and *C1*: $\bigwedge Q.\ (\Psi,\ Q,\ Q) \in Rel$

shows $\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \rightsquigarrow[Rel]\ M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
proof −
note ‹*eqvt Rel*› ‹$x \sharp \Psi$›
moreover have $x \sharp (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P)$ **by**(*simp add*: *abs-fresh*)
moreover from ‹$x \sharp M$› ‹$x \sharp N$› **have** $x \sharp M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
by(*auto simp add*: *inputChainFresh abs-fresh*)
ultimately show *?thesis*
proof(*induct rule*: *simIFresh*[*of* - - - - - ()])
case(*cSim* α P')
from ‹$\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P \longmapsto\alpha \prec P'$› ‹$x \sharp \alpha$› **show** *?case*
proof(*induct rule*: *inputCases*)
case(*cInput K Tvec*)
from ‹$x \sharp K(\!|N[xvec::=Tvec]|\!)$› **have** $x \sharp K$ **and** $x \sharp N[xvec::=Tvec]$ **by** *simp*+
from ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*›
have $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec P[xvec::=Tvec]$
by(*rule Input*)
hence $\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec (\!|\nu x|\!)(P[xvec::=Tvec])$
using ‹$x \sharp \Psi$› ‹$x \sharp K$› ‹$x \sharp N[xvec::=Tvec]$›
by(*rule-tac Scope*) *auto*
moreover from ‹*length xvec* = *length Tvec*› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp*

N› ‹$x \sharp N[xvec::=Tvec]$› **have** $x \sharp Tvec$
by(*rule substTerm.subst3*)
with ‹$x \sharp xvec$› **have** $(\Psi, (\!|\nu x|\!)(P[xvec::=Tvec]), ((\!|\nu x|\!)P)[xvec::=Tvec]) \in Rel$
by(*force intro*: *C1*)
ultimately show *?case* **by** *blast*
qed
qed
qed

lemma *inputPushResRight*:
fixes Ψ :: $'b$
and x :: *name*
and M :: $'a$
and $xvec$ :: *name list*
and N :: $'a$
and P :: $('a, 'b, 'c)$ *psi*

assumes *eqvt Rel*
and $x \sharp \Psi$
and $x \sharp M$
and $x \sharp xvec$
and $x \sharp N$
and *C1*: $\bigwedge Q.\ (\Psi, Q, Q) \in Rel$

shows $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P \rightsquigarrow[Rel]\ (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P)$
proof −
note ‹*eqvt Rel*› ‹$x \sharp \Psi$›
moreover from ‹$x \sharp M$› ‹$x \sharp N$› **have** $x \sharp M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
by(*auto simp add*: *inputChainFresh abs-fresh*)
moreover have $x \sharp (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P)$ **by**(*simp add*: *abs-fresh*)
ultimately show *?thesis*
proof(*induct rule*: *simIFresh*[*of* - - - - - ()])
case(*cSim* α P')
note ‹$\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \longmapsto\alpha \prec P'$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp P'$› ‹$bn\ \alpha \sharp{*} \Psi$›
moreover from ‹$bn\ \alpha \sharp{*} ((\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P))$› ‹$x \sharp \alpha$› **have** $bn\ \alpha \sharp{*} (M(\!|\lambda{*}xvec\ N|\!).P)$
by *simp*
ultimately show *?case* **using** ‹$bn\ \alpha \sharp{*}$ *subject* α›
proof(*induct rule*: *resCases*)
case(*cRes* P')
from ‹$\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto\alpha \prec P'$› ‹$x \sharp \alpha$› **show** *?case*
proof(*induct rule*: *inputCases*)
case(*cInput* K $Tvec$)
from ‹$x \sharp K(\!|N[xvec::=Tvec]|\!)$› **have** $x \sharp K$ **and** $x \sharp N[xvec::=Tvec]$ **by** *simp*+
from ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*›
have $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).((\!|\nu x|\!)P) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec ((\!|\nu x|\!)P)[xvec::=Tvec]$

```
        by(rule Input)
      moreover from ‹length xvec = length Tvec› ‹distinct xvec› ‹set xvec ⊆ supp
N› ‹x ♯ N[xvec::=Tvec]› have x ♯ Tvec
        by(rule substTerm.subst3)
       with ‹x ♯ xvec› have (Ψ, ((⦇νx⦈P)[xvec::=Tvec], ⦇νx⦈(P[xvec::=Tvec])) ∈
Rel
        by(force intro: C1)
      ultimately show ?case by blast
    qed
  next
    case cOpen
    then have False by auto
    thus ?case by simp
  qed
 qed
qed

lemma outputPushResLeft:
 fixes Ψ   :: 'b
 and   x   :: name
 and   M   :: 'a
 and   N   :: 'a
 and   P   :: ('a, 'b, 'c) psi

 assumes eqvt Rel
 and     x ♯ Ψ
 and     x ♯ M
 and     x ♯ N
 and     C1: ⋀Q. (Ψ, Q, Q) ∈ Rel

 shows Ψ ▷ ⦇νx⦈(M⟨N⟩.P) ⇝[Rel] M⟨N⟩.⦇νx⦈P
proof −
 note ‹eqvt Rel› ‹x ♯ Ψ›
 moreover have x ♯ ⦇νx⦈(M⟨N⟩.P) by(simp add: abs-fresh)
 moreover from ‹x ♯ M› ‹x ♯ N› have x ♯ M⟨N⟩.⦇νx⦈P
   by(auto simp add: abs-fresh)
 ultimately show ?thesis
 proof(induct rule: simIFresh[of - - - - - ()])
   case(cSim α P')
   from ‹Ψ ▷ M⟨N⟩.⦇νx⦈P ⟼α ≺ P'› ‹x ♯ α›
   show ?case
   proof(induct rule: outputCases)
     case(cOutput K)
     from ‹Ψ ⊢ M ↔ K› have Ψ ▷ M⟨N⟩.P ⟼K⟨N⟩ ≺ P
       by(rule Output)
     hence Ψ ▷ ⦇νx⦈(M⟨N⟩.P) ⟼K⟨N⟩ ≺ ⦇νx⦈P using ‹x ♯ Ψ› ‹x ♯ K⟨N⟩›
       by(rule Scope)
     moreover have (Ψ, ⦇νx⦈P, ⦇νx⦈P) ∈ Rel by(rule C1)
     ultimately show ?case by blast
```

```
    qed
  qed
qed

lemma outputPushResRight:
  fixes Ψ   :: 'b
  and   x   :: name
  and   M   :: 'a
  and   N   :: 'a
  and   P   :: ('a, 'b, 'c) psi

  assumes eqvt Rel
  and     x ♯ Ψ
  and     x ♯ M
  and     x ♯ N
  and     C1: ⋀Q. (Ψ, Q, Q) ∈ Rel

  shows Ψ ▷ M⟨N⟩.⦇νx⦈P ⇝[Rel] ⦇νx⦈(M⟨N⟩.P)
proof −
  note ‹eqvt Rel› ‹x ♯ Ψ›
  moreover from ‹x ♯ M› ‹x ♯ N› have x ♯ M⟨N⟩.⦇νx⦈P
    by(auto simp add: abs-fresh)
  moreover have x ♯ ⦇νx⦈(M⟨N⟩.P) by(simp add: abs-fresh)
  ultimately show ?thesis
  proof(induct rule: simIFresh[of - - - - - (M, N)])
    case(cSim α P')
    note ‹Ψ ▷ ⦇νx⦈(M⟨N⟩.P) ⟼α ≺ P'› ‹x ♯ Ψ› ‹x ♯ α› ‹x ♯ P'› ‹bn α ♯* Ψ›
    moreover from ‹bn α ♯* (⦇νx⦈(M⟨N⟩.P))› ‹x ♯ α› have bn α ♯* (M⟨N⟩.P)
by simp
    ultimately show ?case using ‹bn α ♯* subject α› ‹bn α ♯* (M, N)› ‹x ♯ α›
    proof(induct rule: resCases)
      case(cOpen K xvec1 xvec2 y N' P')
      from ‹bn(K⦇ν*(xvec1@y#xvec2)⦈⟨N'⟩) ♯* (M, N)› have y ♯ N by simp+
       from ‹Ψ ▷ M⟨N⟩.P ⟼K⦇ν*(xvec1@xvec2)⦈⟨([(x, y)] • N')⟩ ≺ ([(x, y)] •
P')›
      have N = ([(x, y)] • N')
        apply −
        by(ind-cases Ψ ▷ M⟨N⟩.P ⟼K⦇ν*(xvec1@xvec2)⦈⟨([(x, y)] • N')⟩ ≺ ([(x,
y)] • P'))
          (auto simp add: residualInject psi.inject)
      with ‹x ♯ N› ‹y ♯ N› ‹x ≠ y› have N = N'
      by(subst pt-bij[OF pt-name-inst, OF at-name-inst, symmetric, where pi=[(x,
y)]])
          (simp add: fresh-left calc-atm)
      with ‹y ∈ supp N'› ‹y ♯ N› have False by(simp add: fresh-def)
      thus ?case by simp
    next
      case(cRes P')
      from ‹Ψ ▷ M⟨N⟩.P ⟼α ≺ P'› show ?case
```

```
      proof(induct rule: outputCases)
        case(cOutput K)
        from ‹Ψ ⊢ M ↔ K› have Ψ ▷ M⟨N⟩.(⦇νx⦈P) ⟼K⟨N⟩ ≺ ⦇νx⦈P
          by(rule Output)
        moreover have (Ψ, ⦇νx⦈P, ⦇νx⦈P) ∈ Rel by(rule C1)
        ultimately show ?case by force
      qed
    qed
  qed
qed

lemma casePushResLeft:
  fixes Ψ  :: 'b
  and   x  :: name
  and   Cs :: ('c × ('a, 'b, 'c) psi) list

  assumes eqvt Rel
  and     x ♯ Ψ
  and     x ♯ map fst Cs
  and     C1: ⋀Q. (Ψ, Q, Q) ∈ Rel

  shows Ψ ▷ ⦇νx⦈(Cases Cs) ⇝[Rel] Cases (map (λ(φ, P). (φ, ⦇νx⦈P)) Cs)
proof −
  note ‹eqvt Rel› ‹x ♯ Ψ›
  moreover have x ♯ ⦇νx⦈(Cases Cs) by(simp add: abs-fresh)
  moreover from ‹x ♯ map fst Cs› have x ♯ Cases (map (λ(φ, P). (φ, ⦇νx⦈P))
Cs)
    by(induct Cs) (auto simp add: abs-fresh)
  ultimately show ?thesis
  proof(induct rule: simIFresh[of - - - - - Cs])
    case(cSim α P'')
    from ‹Ψ ▷ Cases (map (λ(φ, P). (φ, ⦇νx⦈P)) Cs) ⟼α ≺ P''›
    show ?case
    proof(induct rule: caseCases)
      case(cCase φ P')
      from ‹(φ, P') mem (map (λ(φ, P). (φ, ⦇νx⦈P)) Cs)›
      obtain P where (φ, P) mem Cs and P' = ⦇νx⦈P
        by(induct Cs) auto
      from ‹guarded P'› ‹P' = ⦇νx⦈P› have guarded P by simp
      from ‹Ψ ▷ P' ⟼α ≺ P''› ‹P' = ⦇νx⦈P› have Ψ ▷ ⦇νx⦈P ⟼α ≺ P''
        by simp
      moreover note ‹x ♯ Ψ› ‹x ♯ α› ‹x ♯ P''› ‹bn α ♯* Ψ›
      moreover from ‹bn α ♯* Cs› ‹(φ, P) mem Cs›
      have bn α ♯* P by(auto dest: memFreshChain)
      ultimately show ?case using ‹bn α ♯* subject α› ‹x ♯ α› ‹bn α ♯* Cs›
      proof(induct rule: resCases)
        case(cOpen M xvec1 xvec2 y N P')
        from ‹x ♯ M⦇ν*(xvec1@y#xvec2)⦈⟨N⟩› have x ♯ xvec1 and x ♯ xvec2 and
x ♯ M by simp+
```

from ‹$bn(M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle)\ \sharp*\ Cs$› **have** $y\ \sharp\ Cs$ **by** *simp*
from ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x,\ y)]\cdot N)\rangle \prec ([(x,\ y)]\cdot P')$› ‹$(\varphi,\ P)$ *mem Cs*› ‹$\Psi \vdash \varphi$› ‹*guarded P*›
have $\Psi \rhd Cases\ Cs \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x,\ y)]\cdot N)\rangle \prec ([(x,\ y)]\cdot P')$ **by**(*rule Case*)
hence $([(x,\ y)]\cdot\Psi) \rhd ([(x,\ y)]\cdot(Cases\ Cs)) \longmapsto([(x,\ y)]\cdot(M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x,\ y)]\cdot N)\rangle \prec ([(x,\ y)]\cdot P')))$
by(*rule semantics.eqvt*)
with ‹$x\ \sharp\ \Psi$› ‹$x\ \sharp\ M$› ‹$y\ \sharp\ xvec1$› ‹$y\ \sharp\ xvec2$› ‹$y\ \sharp\ \Psi$› ‹$y\ \sharp\ M$› ‹$x\ \sharp\ xvec1$› ‹$x\ \sharp\ xvec2$›
have $\Psi \rhd ([(x,\ y)]\cdot(Cases\ Cs)) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec P'$ **by**(*simp add*: *eqvts*)
hence $\Psi \rhd (\!|\nu y|\!)([(x,\ y)]\cdot(Cases\ Cs)) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec P'$ **using** ‹$y \in supp\ N$› ‹$y\ \sharp\ \Psi$› ‹$y\ \sharp\ M$› ‹$y\ \sharp\ xvec1$› ‹$y\ \sharp\ xvec2$›
by(*rule Open*)
hence $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec P'$ **using** ‹$y\ \sharp\ Cs$›
by(*simp add*: *alphaRes*)
moreover have $(\Psi,\ P',\ P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cRes P'*)
from ‹$\Psi \rhd P \longmapsto \alpha \prec P'$› ‹$(\varphi,\ P)$ *mem Cs*› ‹$\Psi \vdash \varphi$› ‹*guarded P*›
have $\Psi \rhd Cases\ Cs \longmapsto \alpha \prec P'$ **by**(*rule Case*)
hence $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \longmapsto \alpha \prec (\!|\nu x|\!)P'$ **using** ‹$x\ \sharp\ \Psi$› ‹$x\ \sharp\ \alpha$›
by(*rule Scope*)
moreover have $(\Psi,\ (\!|\nu x|\!)P',\ (\!|\nu x|\!)P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
qed
qed
qed

lemma *casePushResRight*:
fixes Ψ :: $'b$
and x :: *name*
and Cs :: $('c \times ('a,\ 'b,\ 'c)\ psi)\ list$

assumes *eqvt Rel*
and $x\ \sharp\ \Psi$
and $x\ \sharp\ map\ fst\ Cs$
and $C1$: $\bigwedge Q.\ (\Psi,\ Q,\ Q) \in Rel$

shows $\Psi \rhd Cases\ (map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs) \rightsquigarrow[Rel]\ (\!|\nu x|\!)(Cases\ Cs)$
proof −
note ‹*eqvt Rel*› ‹$x\ \sharp\ \Psi$›
moreover from ‹$x\ \sharp\ map\ fst\ Cs$› **have** $x\ \sharp\ Cases\ (map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs)$
by(*induct Cs*) (*auto simp add*: *abs-fresh*)

moreover have $x \sharp (\!|\nu x|\!)(Cases\ Cs)$ **by**(*simp add*: *abs-fresh*)
ultimately show *?thesis*
proof(*induct rule*: *simIFresh*[*of* - - - - - *Cs*])
case(*cSim* α P'')
note ‹$\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \longmapsto \alpha \prec P''$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp P''$› ‹$bn\ \alpha \sharp* \Psi$›
moreover from ‹$bn\ \alpha \sharp* (\!|\nu x|\!)(Cases\ Cs)$› ‹$x \sharp \alpha$› **have** $bn\ \alpha \sharp* (Cases\ Cs)$ **by** *simp*
ultimately show *?case* **using** ‹$bn\ \alpha \sharp*\ subject\ \alpha$› ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp*\ Cs$›
proof(*induct rule*: *resCases*)
case(*cOpen M xvec1 xvec2 y N P'*)
from ‹$x \sharp M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle$› **have** $x \sharp xvec1$ **and** $x \sharp xvec2$ **and** $x \sharp M$ **by** *simp+*
from ‹$bn(M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle) \sharp* Cs$› **have** $y \sharp Cs$ **by** *simp*
from ‹$\Psi \rhd Cases\ Cs \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot P')$›
show *?case*
proof(*induct rule*: *caseCases*)
case(*cCase* φ P)
from ‹$\Psi \rhd P \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot P')$›
have $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \longmapsto([(x, y)] \cdot (M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot P')))$
by(*rule semantics.eqvt*)
with ‹$x \sharp \Psi$› ‹$x \sharp M$› ‹$y \sharp xvec1$› ‹$y \sharp xvec2$› ‹$y \sharp \Psi$› ‹$y \sharp M$› ‹$x \sharp xvec1$› ‹$x \sharp xvec2$›
have $\Psi \rhd ([(x, y)] \cdot P) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec P'$ **by**(*simp add*: *eqvts*)
hence $\Psi \rhd (\!|\nu y|\!)([(x, y)] \cdot P) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec P'$ **using** ‹$y \in supp\ N$› ‹$y \sharp \Psi$› ‹$y \sharp M$› ‹$y \sharp xvec1$› ‹$y \sharp xvec2$›
by(*rule Open*)
hence $\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec P'$ **using** ‹$y \sharp Cs$› ‹$(\varphi, P)\ mem\ Cs$›
by(*subst alphaRes*, *auto dest*: *memFresh*)
moreover from ‹$(\varphi, P)\ mem\ Cs$› **have** $(\varphi, (\!|\nu x|\!)P)\ mem\ (map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs)$
by(*induct Cs*) *auto*
moreover note ‹$\Psi \vdash \varphi$›
moreover from ‹$guarded\ P$› **have** $guarded((\!|\nu x|\!)P)$ **by** *simp*
ultimately have $\Psi \rhd (Cases\ (map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs)) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec P'$
by(*rule Case*)
moreover have $(\Psi, P', P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
next
case(*cRes P'*)
from ‹$\Psi \rhd Cases\ Cs \longmapsto \alpha \prec P'$›
show *?case*
proof(*induct rule*: *caseCases*)
case(*cCase* φ P)

from ‹$\Psi \rhd P \longmapsto\alpha \prec P'$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$›
have $\Psi \rhd (\!|\nu x|\!)P \longmapsto\alpha \prec (\!|\nu x|\!)P'$ **by**(*rule Scope*)
moreover from ‹(φ, P) *mem Cs*› **have** $(\varphi, (\!|\nu x|\!)P)$ *mem* (*map* $(\lambda(\varphi, P).$ $(\varphi, (\!|\nu x|\!)P))$ *Cs*)
by(*induct Cs*) *auto*
moreover note ‹$\Psi \vdash \varphi$›
moreover from ‹*guarded P*› **have** *guarded*$((\!|\nu x|\!)P)$ **by** *simp*
ultimately have $\Psi \rhd$ (*Cases* (*map* $(\lambda(\varphi, P).$ $(\varphi, (\!|\nu x|\!)P))$ *Cs*)) $\longmapsto\alpha \prec$ $(\!|\nu x|\!)P'$
by(*rule Case*)
moreover have $(\Psi, (\!|\nu x|\!)P', (\!|\nu x|\!)P') \in$ *Rel* **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
qed
qed
qed

lemma *resInputCases*[*consumes 5*, *case-names cRes*]:
fixes Ψ :: $'b$
and x :: *name*
and P :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$
and P' :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd (\!|\nu x|\!)P \longmapsto M(\!|N|\!) \prec P'$
and $x \sharp \Psi$
and $x \sharp M$
and $x \sharp N$
and $x \sharp P'$
and *rScope*: $\bigwedge P'. [\![\Psi \rhd P \longmapsto M(\!|N|\!) \prec P']\!] \Longrightarrow$ *Prop* $((\!|\nu x|\!)P')$

shows *Prop* P'
proof −
note *Trans* ‹$x \sharp \Psi$›
moreover from ‹$x \sharp M$› ‹$x \sharp N$› **have** $x \sharp (M(\!|N|\!))$ **by** *simp*
moreover note ‹$x \sharp P'$›
moreover have $bn(M(\!|N|\!)) \sharp* \Psi$ **and** $bn(M(\!|N|\!)) \sharp* P$ **and** $bn(M(\!|N|\!)) \sharp*$ *subject*$(M(\!|N|\!))$ **and** $bn(M(\!|N|\!)) = []$ **by** *simp+*
ultimately show *?thesis*
by(*induct rule*: *resCases*) (*auto intro*: *rScope*)
qed

lemma *scopeExtLeft*:
fixes x :: *name*
and P :: $('a, 'b, 'c)$ *psi*
and Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*

and $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $x \sharp P$
and $x \sharp \Psi$
and $eqvt\ Rel$
and $C1$: $\bigwedge \Psi'\ R.\ (\Psi', R, R) \in Rel$
and $C2$: $\bigwedge y\ \Psi'\ R\ S\ zvec.\ [\![y \sharp \Psi'; y \sharp R; zvec \sharp* \Psi']\!] \Longrightarrow (\Psi', (\!|\nu y|\!)((\!|\nu{*}zvec|\!)(R \parallel S)), (\!|\nu{*}zvec|\!)(R \parallel (\!|\nu y|\!)S)) \in Rel$
and $C3$: $\bigwedge \Psi'\ zvec\ R\ y.\ [\![y \sharp \Psi'; zvec \sharp* \Psi']\!] \Longrightarrow (\Psi', (\!|\nu y|\!)((\!|\nu{*}zvec|\!)R), (\!|\nu{*}zvec|\!)((\!|\nu y|\!)R)) \in Rel$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \rightsquigarrow[Rel]\ P \parallel (\!|\nu x|\!)Q$
proof −
note ‹$eqvt\ Rel$› ‹$x \sharp \Psi$›
moreover have $x \sharp (\!|\nu x|\!)(P \parallel Q)$ **by**(*simp add*: *abs-fresh*)
moreover from ‹$x \sharp P$› **have** $x \sharp P \parallel (\!|\nu x|\!)Q$ **by**(*simp add*: *abs-fresh*)
ultimately show *?thesis*
proof(*induct rule*: *simIFresh*[*of* - - - - - x])
case(*cSim* α PQ)
from ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp* (P \parallel (\!|\nu x|\!)Q)$› **have** $bn\ \alpha \sharp* Q$ **by** *simp*
note ‹$\Psi \rhd P \parallel (\!|\nu x|\!)Q \longmapsto \alpha \prec PQ$› ‹$bn\ \alpha \sharp* \Psi$›
moreover from ‹$bn\ \alpha \sharp* (P \parallel (\!|\nu x|\!)Q)$› **have** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* (\!|\nu x|\!)Q$ **by** *simp*+
ultimately show *?case* **using** ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$x \sharp PQ$›
proof(*induct rule*: *parCases*[**where** $C{=}x$])
case(*cPar1* $P'\ A_Q\ \Psi_Q$)
from ‹$x \sharp P' \parallel (\!|\nu x|\!)Q$› **have** $x \sharp P'$ **by** *simp*
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$ **by** *fact*
from ‹$extractFrame((\!|\nu x|\!)Q) = \langle A_Q, \Psi_Q\rangle$› **have** *FrxQ*: $(\!|\nu x|\!)(extractFrame\ Q) = \langle A_Q, \Psi_Q\rangle$ **by** *simp*
then obtain $y\ A_Q{}'$ **where** A: $A_Q = y\#A_Q{}'$ **by**(*case-tac* A_Q) *auto*
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$›
have $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* \alpha$
and $y \sharp \Psi$ **and** $y \sharp P$ **and** $y \sharp \alpha$
by *simp*+
from *PTrans* ‹$y \sharp P$› ‹$y \sharp \alpha$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$› **have** $y \sharp P'$
by(*auto intro*: *freeFreshDerivative*)
note *PTrans*
moreover from A ‹$A_Q \sharp* x$› *FrxQ* **have** $extractFrame([(y, x)] \cdot Q) = \langle A_Q{}', \Psi_Q\rangle$
by(*simp add*: *frame.inject alpha′ fresh-list-cons eqvts*)
moreover from ‹$bn\ \alpha \sharp* Q$› **have** $([(y, x)] \cdot (bn\ \alpha)) \sharp* ([(y, x)] \cdot Q)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp \alpha$› ‹$A_Q \sharp* \alpha$› A **have** $bn\ \alpha \sharp* ([(y, x)] \cdot Q)$ **by** *simp*
ultimately have $\Psi \rhd P \parallel ([(y, x)] \cdot Q) \longmapsto \alpha \prec (P' \parallel ([(y, x)] \cdot Q))$
using ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* P$› ‹$A_Q{}' \sharp* \alpha$›
by(*rule Par1*)
hence $\Psi \rhd (\!|\nu y|\!)(P \parallel ([(y, x)] \cdot Q)) \longmapsto \alpha \prec (\!|\nu y|\!)(P' \parallel ([(y, x)] \cdot Q))$
using ‹$y \sharp \Psi$› ‹$y \sharp \alpha$›

by(*rule Scope*)
hence $([(y, x)] \cdot \Psi) \rhd ([(y, x)] \cdot ((\!|\nu y|\!)(P \parallel ([(y, x)] \cdot Q)))) \longmapsto ([(y, x)] \cdot (\alpha \prec (\!|\nu y|\!)(P' \parallel ([(y, x)] \cdot Q))))$
by(*rule semantics.eqvt*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp P$› ‹$y \sharp P$› ‹$x \sharp \alpha$› ‹$y \sharp \alpha$› ‹$x \sharp P'$› ‹$y \sharp P'$›
have $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto \alpha \prec (\!|\nu x|\!)(P' \parallel Q)$
by(*simp add*: *eqvts calc-atm*)
moreover from ‹$x \sharp \Psi$› ‹$x \sharp P'$› **have** $(\Psi, (\!|\nu x|\!)((\!|\nu *[]|\!)(P' \parallel Q)), (\!|\nu *[]|\!)(P' \parallel (\!|\nu x|\!)Q)) \in Rel$
by(*rule-tac C2*) *auto*
ultimately show *?case*
by *force*
next
case(*cPar2 xQ' A_P Ψ_P*)
from ‹$A_P \sharp* (\!|\nu x|\!)Q$› ‹$A_P \sharp* x$› **have** $A_P \sharp* Q$ **by** *simp*
note ‹$\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto \alpha \prec xQ'$›
moreover have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
with ‹$x \sharp P$› ‹$A_P \sharp* x$› **have** $x \sharp \Psi_P$ **and** $x \sharp A_P$
by(*force dest*: *extractFrameFresh*)+
with ‹$x \sharp \Psi$› **have** $x \sharp \Psi \otimes \Psi_P$ **by** *force*
moreover note ‹$x \sharp \alpha$›
moreover from ‹$x \sharp P \parallel xQ'$› **have** $x \sharp xQ'$ **by** *simp*
moreover from *FrP* ‹$bn\ \alpha \sharp* P$› ‹$A_P \sharp* \alpha$› **have** $bn\ \alpha \sharp* \Psi_P$
by(*drule-tac extractFrameFreshChain*) *auto*
with ‹$bn\ \alpha \sharp* \Psi$› **have** $bn\ \alpha \sharp* (\Psi \otimes \Psi_P)$ **by** *force*
ultimately show *?case* **using** ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp* P$› ‹$A_P \sharp* \alpha$› ‹$bn\ \alpha \sharp* \Psi$›
proof(*induct rule*: *resCases'*)
case(*cOpen M xvec1 xvec2 y N Q'*)
from ‹$x \sharp M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle$› **have** $x \sharp xvec1$ **and** $x \neq y$ **and** $x \sharp xvec2$ **by** *simp*+
from ‹$bn(M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle) \sharp* \Psi$› **have** $y \sharp \Psi$ **by** *simp*
note ‹$\Psi \otimes \Psi_P \rhd ([(x, y)] \cdot Q) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec Q'$› *FrP*
moreover from ‹$bn(M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle) \sharp* P$› **have** $(xvec1@xvec2) \sharp* P$ **and** $y \sharp P$ **by** *simp*+
moreover from ‹$A_P \sharp* (M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle)$› **have** $A_P \sharp* (M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle)$ **and** $y \sharp A_P$ **by** *simp*+
moreover from ‹$A_P \sharp* Q$› ‹$x \sharp A_P$› ‹$y \sharp A_P$› **have** $A_P \sharp* ([(x, y)] \cdot Q)$ **by** *simp*
ultimately have $\Psi \rhd P \parallel ([(x, y)] \cdot Q) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec (P \parallel Q')$
using ‹$A_P \sharp* \Psi$›
by(*rule-tac Par2*) (*assumption* | *simp*)+
hence $\Psi \rhd (\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec (P \parallel Q')$
using ‹$y \in supp\ N$› ‹$y \sharp \Psi$› ‹$y \sharp M$› ‹$y \sharp xvec1$› ‹$y \sharp xvec2$›
by(*rule Open*)
with ‹$x \sharp P$› ‹$y \sharp P$› ‹$y \sharp Q$› **have** $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec (P \parallel Q')$

by(*subst alphaRes*[**where** *y*=*y*]) (*simp add*: *fresh-left calc-atm eqvts*)+
moreover have $(\Psi, P \parallel Q', P \parallel Q') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cRes Q'*)
from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto\alpha \prec Q'$› *FrP* ‹*bn* $\alpha \sharp* P$›
have $\Psi \rhd P \parallel Q \longmapsto\alpha \prec (P \parallel Q')$ **using** ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \alpha$›
by(*rule Par2*)
hence $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto\alpha \prec (\!|\nu x|\!)(P \parallel Q')$ **using** ‹$x \sharp \Psi$› ‹$x \sharp \alpha$›
by(*rule Scope*)
moreover from ‹$x \sharp \Psi$› ‹$x \sharp P$› **have** $(\Psi, (\!|\nu x|\!)((\!|\nu*[]|\!)(P \parallel Q')), (\!|\nu*[]|\!)(P \parallel (\!|\nu x|\!)Q')) \in Rel$
by(*rule-tac C2*) *auto*
ultimately show *?case*
by *force*
qed
next
case(*cComm1* Ψ_Q *M N P'* A_P Ψ_P *K xvec xQ'* A_Q)
have *QTrans*: $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec xQ'$ **and** *FrQ*: *extractFrame*$((\!|\nu x|\!)Q) = \langle A_Q, \Psi_Q\rangle$ **by** *fact*+
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$ **and** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*+
have $x \sharp (\!|\nu x|\!)Q$ **by**(*simp add*: *abs-fresh*)
with *QTrans* **have** $x \sharp N$ **and** $x \sharp xQ'$ **using** ‹*xvec* $\sharp* x$› ‹*xvec* $\sharp* K$› ‹*distinct xvec*›
by(*force intro*: *outputFreshDerivative*)+
from *PTrans* ‹$x \sharp P$› ‹$x \sharp N$› **have** $x \sharp P'$ **by**(*rule inputFreshDerivative*)
from ‹$x \sharp (\!|\nu x|\!)Q$› *FrQ* ‹$A_Q \sharp* x$› **have** $x \sharp \Psi_Q$
by(*drule-tac extractFrameFresh*) *auto*
from ‹$x \sharp P$› *FrP* ‹$A_P \sharp* x$› **have** $x \sharp \Psi_P$
by(*drule-tac extractFrameFresh*) *auto*
from ‹$A_P \sharp* (\!|\nu x|\!)Q$› ‹$A_P \sharp* x$› **have** $A_P \sharp* Q$ **by** *simp*
from ‹$A_Q \sharp* (\!|\nu x|\!)Q$› ‹$A_Q \sharp* x$› **have** $A_Q \sharp* Q$ **by** *simp*
from *PTrans FrP* ‹*distinct* A_P› ‹$x \sharp P$› ‹$A_Q \sharp* P$› ‹*xvec* $\sharp* P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* x$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp*$ *xvec*›
obtain M' **where** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $x \sharp M'$ **and** $A_Q \sharp* M'$ **and** *xvec* $\sharp* M'$
by(*rule-tac B*=*x*#*xvec*@A_Q **in** *obtainPrefix*) (*assumption* | *force simp add*: *fresh-star-list-cons*)+
hence *MeqM'*: $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow M'$ **by**(*metis statEqEnt Associativity Commutativity Composition*)
with ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*blast intro*: *chanEqTrans chanEqSym*)
hence $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$ **by**(*metis statEqEnt Associativity Commutativity Composition*)
with *QTrans FrQ* ‹*distinct* A_Q› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* ((\!|\nu x|\!)Q)$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
have $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto M'(\!|\nu*xvec|\!)\langle N\rangle \prec xQ'$
by(*force intro*: *outputRenameSubject*)

moreover from ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› **have** $x \sharp \Psi \otimes \Psi_P$ **by** *force*
moreover from ‹$xvec \sharp* x$› **have** $x \sharp xvec$ **by** *simp*
with ‹$x \sharp M'$› ‹$x \sharp N$› **have** $x \sharp M'(\!|\nu*xvec|\!)\langle N\rangle$ **by** *simp*
moreover note ‹$x \sharp xQ'$›
moreover from ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* \Psi_P$› **have** $bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* (\Psi \otimes \Psi_P)$ **by** *force*
moreover from ‹$xvec \sharp* (\!|\nu x|\!)Q$› ‹$x \sharp xvec$› **have** $bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* Q$ **by** *simp*
moreover from ‹$xvec \sharp* P$› **have** $bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* P$ **by** *simp*
from ‹$xvec \sharp* \Psi$› **have** $bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* \Psi$ **by** *simp*
from ‹$A_Q \sharp* xvec$› ‹$A_Q \sharp* M'$› ‹$A_Q \sharp* N$› **have** $A_Q \sharp* (M'(\!|\nu*xvec|\!)\langle N\rangle)$ **by** *simp*
have $object(M'(\!|\nu*xvec|\!)\langle N\rangle) = Some\ N$ **by** *simp*
have $bn(M'(\!|\nu*xvec|\!)\langle N\rangle) = xvec$ **by** *simp*
have $subject(M'(\!|\nu*xvec|\!)\langle N\rangle) = Some\ M'$ **by** *simp*
from ‹$xvec \sharp* M'$› **have** $bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* subject(M'(\!|\nu*xvec|\!)\langle N\rangle)$ **by** *simp*
ultimately show *?case*
using ‹$x \sharp M'(\!|\nu*xvec|\!)\langle N\rangle$› ‹$bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* P$› ‹$bn(M'(\!|\nu*xvec|\!)\langle N\rangle) \sharp* \Psi$› ‹$object(M'(\!|\nu*xvec|\!)\langle N\rangle) = Some\ N$›
‹$bn(M'(\!|\nu*xvec|\!)\langle N\rangle) = xvec$› ‹$subject(M'(\!|\nu*xvec|\!)\langle N\rangle) = Some\ M'$› ‹$A_Q \sharp* (M'(\!|\nu*xvec|\!)\langle N\rangle)$›
proof(*induct rule*: *resCases*)
case(*cOpen M'' xvec1 xvec2 y N' Q'*)
from ‹$x \sharp M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle$› **have** $x \sharp xvec1$ **and** $x \neq y$ **and** $x \sharp xvec2$ **and** $x \sharp M''$
by *simp+*
from ‹$bn(M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle) \sharp* P$› **have** $(xvec1@xvec2) \sharp* P$ **and** $y \sharp P$ **by** *simp+*
from ‹$A_Q \sharp* (M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle)$› **have** $(xvec1@xvec2) \sharp* A_Q$ **and** $y \sharp A_Q$ **and** $A_Q \sharp* M''$ **by** *simp+*
from ‹$bn(M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle) \sharp* \Psi$› **have** $(xvec1@xvec2) \sharp* \Psi$ **and** $y \sharp \Psi$ **by** *simp+*
from ‹$object(M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle) = Some\ N$› **have** $N = N'$ **by** *simp*
from ‹$bn(M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle) = xvec$› **have** $xvec = xvec1@y\#xvec2$ **by** *simp*
from ‹$subject(M''(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N'\rangle) = Some\ M'$› **have** $M' = M''$ **by** *simp*
from ‹$x \sharp P$› ‹$y \sharp P$› ‹$x \neq y$› ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|([(y, x)] \cdot N)|\!) \prec ([(y, x)] \cdot P')$
by(*rule-tac xvec*=[*y*] **in** *inputAlpha*) (*auto simp add*: *calc-atm*)
hence *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|([(x, y)] \cdot N)|\!) \prec ([(x, y)] \cdot P')$
by(*simp add*: *name-swap*)
have *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto M''(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N')\rangle \prec ([(x, y)] \cdot Q')$ **by** *fact*
with ‹$A_Q \sharp* x$› ‹$y \sharp A_Q$› ‹*distinct xvec1*› ‹*distinct xvec2*› ‹$xvec1 \sharp* xvec2$› ‹$xvec1 \sharp* M''$› ‹$xvec2 \sharp* M''$›
‹$(xvec1@xvec2) \sharp* A_Q$›

have $A_Q \sharp* ([(x, y)] \cdot Q')$ **using** ‹$A_Q \sharp* Q$›
by(*rule-tac outputFreshChainDerivative*(*2*)) (*assumption* | *simp*)+

from ‹*extractFrame*($(\!|\nu x|\!)Q$) = $\langle A_Q, \Psi_Q\rangle$› **have** *FrxQ*: $(\!|\nu x|\!)$(*extractFrame* Q) = $\langle A_Q, \Psi_Q\rangle$ **by** *simp*
then obtain z A_Q' **where** A: $A_Q = z\#A_Q'$ **by**(*case-tac* A_Q) *auto*
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹(*xvec1*@*xvec2*) $\sharp* A_Q$› ‹$A_Q \sharp* M''$› ‹$A_Q \sharp* ([(x, y)] \cdot Q')$› ‹$y \sharp A_Q$› ‹$A_Q \sharp* N$›
have $A_Q' \sharp* \Psi$ **and** $A_Q' \sharp* P$ **and** $A_Q' \sharp* \Psi_P$ **and** $A_Q' \sharp* Q$
and $z \sharp \Psi$ **and** $z \sharp P$ **and** $z \sharp P'$ **and** $z \sharp \Psi_P$ **and** $z \sharp Q$ **and** $z \sharp$ *xvec1* **and** $z \sharp$ *xvec2*
and $z \sharp M''$ **and** $z \sharp ([(x, y)] \cdot Q')$ **and** $A_Q' \sharp* M''$ **and** $z \neq y$ **and** $z \sharp$ (*xvec1*@*xvec2*)
by *auto*
from A ‹$A_P \sharp* A_Q$› **have** $A_P \sharp* A_Q'$ **and** $z \sharp A_P$ **by** *simp*+
from A ‹$A_Q \sharp* x$› **have** $x \neq z$ **and** $x \sharp A_Q'$ **by** *simp*+

from ‹*distinct* A_Q› A **have** $z \sharp A_Q'$
by(*induct* A_Q') (*auto simp add*: *fresh-list-nil fresh-list-cons*)
from *PTrans* ‹$x \sharp P$› ‹$z \sharp P$› ‹$x \neq z$› **have** $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|([(x, z)] \cdot [(x, y)] \cdot N)|\!) \prec ([(x, z)] \cdot [(x, y)] \cdot P')$
by(*rule-tac xvec*=[x] **in** *inputAlpha*) (*auto simp add*: *calc-atm*)
moreover note *FrP*
moreover from *QTrans* **have** $([(x, z)] \cdot (\Psi \otimes \Psi_P)) \rhd ([(x, z)] \cdot Q) \longmapsto([(x, z)] \cdot (M''(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N')\rangle \prec ([(x, y)] \cdot Q')))$
by(*rule semantics.eqvt*)
with ‹$x \sharp \Psi$› ‹$z \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$z \sharp \Psi_P$› ‹$x \sharp M''$› ‹$z \sharp M''$› ‹$x \sharp$ *xvec1*› ‹$x \sharp$ *xvec2*› ‹$z \sharp$ *xvec1*› ‹$z \sharp$ *xvec2*›
have $\Psi \otimes \Psi_P \rhd ([(x, z)] \cdot Q) \longmapsto M''(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, z)] \cdot [(x, y)] \cdot N')\rangle \prec ([(x, z)] \cdot [(x, y)] \cdot Q')$
by(*simp add*: *eqvts*)
moreover from A ‹$A_Q \sharp* x$› *FrxQ* **have** *extractFrame*($[(x, z)] \cdot Q$) = $\langle A_Q', \Psi_Q\rangle$
by(*clarsimp simp add*: *alpha′ eqvts frame.inject fresh-list-cons name-swap*)
moreover from ‹$A_P \sharp* Q$› **have** $([(x, z)] \cdot A_P) \sharp* ([(x, z)] \cdot Q)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* x$› ‹$z \sharp A_P$› **have** $A_P \sharp* ([(x, z)] \cdot Q)$ **by** *simp*
moreover from ‹$A_Q' \sharp* Q$› **have** $([(x, z)] \cdot A_Q') \sharp* ([(x, z)] \cdot Q)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_Q'$› ‹$z \sharp A_Q'$› **have** $A_Q' \sharp* ([(x, z)] \cdot Q)$ **by** *simp*
ultimately have $\Psi \rhd (P \parallel ([(x, z)] \cdot Q)) \longmapsto\tau \prec (\!|\nu*(xvec1@xvec2)|\!)(([(x, z)] \cdot [(x, y)] \cdot P') \parallel ([(x, z)] \cdot [(x, y)] \cdot Q'))$
using *MeqM′* ‹M'=M''› ‹N=N'› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* A_Q'$› ‹$A_Q' \sharp* \Psi$› ‹$A_Q' \sharp* P$› ‹(*xvec1*@*xvec2*) $\sharp* P$› ‹$A_P \sharp* M$› ‹$A_Q' \sharp* M''$›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
with‹$z \sharp \Psi$› **have** $\Psi \rhd (\!|\nu z|\!)(P \parallel ([(x, z)] \cdot Q)) \longmapsto\tau \prec (\!|\nu z|\!)((\!|\nu*(xvec1@xvec2)|\!)(([(x, z)] \cdot [(x, y)] \cdot P') \parallel ([(x, z)] \cdot [(x, y)] \cdot Q')))$
by(*rule-tac Scope*) *auto*
moreover from ‹$x \sharp P$› ‹$z \sharp P$› ‹$z \sharp Q$› **have** $(\!|\nu z|\!)(P \parallel ([(x, z)] \cdot Q)) =$

$(\!|\nu x|\!)([(x,\ z)] \cdot (P \parallel ([(x,\ z)] \cdot Q)))$
by(*subst alphaRes*[*of x*]) (*auto simp add*: *calc-atm fresh-left name-swap*)
with ‹$x \sharp P$› ‹$z \sharp P$› **have** $(\!|\nu z|\!)(P \parallel ([(x,\ z)] \cdot Q)) = (\!|\nu x|\!)(P \parallel Q)$
by(*simp add*: *eqvts*)
moreover from ‹$z \neq y$› ‹$x \neq z$› ‹$z \sharp P'$› ‹$z \sharp [(x,\ y)] \cdot Q'$› **have** $(\!|\nu z|\!)((\!|\nu*(xvec1@xvec2)|\!)(([(x,\ z)] \cdot [(x,\ y)] \cdot P') \parallel ([(x,\ z)] \cdot [(x,\ y)] \cdot Q'))) = (\!|\nu x|\!)([(x,\ z)] \cdot ((\!|\nu*(xvec1@xvec2)|\!)(([(x,\ z)] \cdot [(x,\ y)] \cdot P') \parallel ([(x,\ z)] \cdot [(x,\ y)] \cdot Q'))))$
by(*subst alphaRes*[*of x*]) (*auto simp add*: *resChainFresh fresh-left calc-atm name-swap*)
with ‹$x \sharp xvec1$› ‹$x \sharp xvec2$› ‹$z \sharp xvec1$› ‹$z \sharp xvec2$› **have** $(\!|\nu z|\!)((\!|\nu*(xvec1@xvec2)|\!)(([(x,\ z)] \cdot [(x,\ y)] \cdot P') \parallel ([(x,\ z)] \cdot [(x,\ y)] \cdot Q'))) = (\!|\nu x|\!)((\!|\nu*(xvec1@xvec2)|\!)(([(x,\ y)] \cdot P') \parallel ([(x,\ y)] \cdot Q')))$
by(*simp add*: *eqvts*)
moreover from ‹$x \sharp P'$› ‹$x \sharp Q'$› ‹$x \sharp xvec1$› ‹$x \sharp xvec2$› ‹$y \sharp xvec1$› ‹$y \sharp xvec2$›
have $(\!|\nu x|\!)((\!|\nu*(xvec1@xvec2)|\!)(([(x,\ y)] \cdot P') \parallel ([(x,\ y)] \cdot Q'))) = (\!|\nu y|\!)((\!|\nu*(xvec1@xvec2)|\!)(P' \parallel Q'))$
by(*subst alphaRes*[*of y*]) (*auto simp add*: *resChainFresh calc-atm eqvts fresh-left name-swap*)
ultimately have $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto\tau \prec (\!|\nu y|\!)((\!|\nu*(xvec1@xvec2)|\!)(P' \parallel Q'))$
by *simp*
moreover from ‹$y \sharp \Psi$› ‹$(xvec1@xvec2) \sharp* \Psi$› ‹$xvec{=}xvec1@y\#xvec2$›
have $(\Psi,\ (\!|\nu y|\!)((\!|\nu*(xvec1@xvec2)|\!)(P' \parallel Q')),\ (\!|\nu*xvec|\!)(P' \parallel Q')) \in Rel$
by(*force intro*: *C3 simp add*: *resChainAppend*)
ultimately show *?case* **by** *blast*
next
case(*cRes* Q')
have *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **by** *fact*
with ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* xvec$› ‹$xvec \sharp* M'$› ‹*distinct xvec*› **have** $A_Q \sharp* Q'$
by(*force dest*: *outputFreshChainDerivative*)

with ‹$extractFrame((\!|\nu x|\!)Q) = \langle A_Q,\ \Psi_Q\rangle$› **have** *FrxQ*: $(\!|\nu x|\!)(extractFrame\ Q) = \langle A_Q,\ \Psi_Q\rangle$ **by** *simp*
then obtain $y\ A_Q{}'$ **where** A: $A_Q = y\#A_Q{}'$ **by**(*case-tac* A_Q) *auto*
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* xvec$› ‹$A_Q \sharp* M'$› ‹$A_Q \sharp* Q'$›
have $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* \Psi_P$ **and** $A_Q{}' \sharp* Q$ **and** $A_Q \sharp* xvec$ **and** $A_Q \sharp* Q'$
and $y \sharp \Psi$ **and** $y \sharp P$ **and** $y \sharp P'$ **and** $y \sharp \Psi_P$ **and** $y \sharp Q$ **and** $y \sharp xvec$ **and** $y \sharp M'$ **and** $y \sharp Q'$
and $A_Q{}' \sharp* M'$
by(*simp*)+
from A ‹$A_P \sharp* A_Q$› **have** $A_P \sharp* A_Q{}'$ **and** $y \sharp A_P$ **by**(*simp add*: *fresh-star-list-cons*)+
from A ‹$A_Q \sharp* x$› **have** $x \neq y$ **and** $x \sharp A_Q{}'$ **by**(*simp add*: *fresh-list-cons*)+

with A ‹*distinct* A_Q› **have** $y \sharp A_Q{}'$
by(*induct* $A_Q{}'$) (*auto simp add*: *fresh-list-nil fresh-list-cons*)

from ‹$x \sharp P$› ‹$y \sharp P$› ‹$x \neq y$› ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$›
have $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|([(y, x)] \cdot N)|\!) \prec [(y, x)] \cdot P'$
by(*rule-tac xvec*=[*y*] **in** *inputAlpha*) (*auto simp add*: *calc-atm*)
hence $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|([(x, y)] \cdot N)|\!) \prec [(x, y)] \cdot P'$
by(*simp add*: *name-swap*)
moreover note *FrP*
moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› **have** $([(x, y)] \cdot (\Psi \otimes \Psi_P)) \rhd ([(x, y)] \cdot Q) \longmapsto ([(x, y)] \cdot (M'(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'))$
by(*rule semantics.eqvt*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$y \sharp \Psi_P$› ‹$x \sharp M'$› ‹$y \sharp M'$› ‹$x \sharp xvec$› ‹$y \sharp xvec$›
have $\Psi \otimes \Psi_P \rhd ([(x, y)] \cdot Q) \longmapsto M'(\!|\nu{*}xvec|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot Q')$
by(*simp add*: *eqvts*)
moreover from A ‹$A_Q \sharp{*}\ x$› *FrxQ* **have** *FrQ*: $extractFrame([(x, y)] \cdot Q) = \langle A_Q{}', \Psi_Q\rangle$
by(*clarsimp simp add*: *alpha′ eqvts frame.inject fresh-list-cons name-swap*)
moreover from ‹$A_P \sharp{*}\ Q$› **have** $([(x, y)] \cdot A_P) \sharp{*} ([(x, y)] \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp{*}\ x$› ‹$y \sharp A_P$› **have** $A_P \sharp{*} ([(x, y)] \cdot Q)$ **by** *simp*
moreover from ‹$A_Q{}' \sharp{*}\ Q$› **have** $([(x, y)] \cdot A_Q{}') \sharp{*} ([(x, y)] \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_Q{}'$› ‹$y \sharp A_Q{}'$› **have** $A_Q{}' \sharp{*} ([(x, y)] \cdot Q)$ **by** *simp*
ultimately have $\Psi \rhd (P \parallel ([(x, y)] \cdot Q)) \longmapsto \tau \prec (\!|\nu{*}xvec|\!)([(x, y)] \cdot P') \parallel ([(x, y)] \cdot Q')$
using *MeqM′* ‹$A_P \sharp{*}\ \Psi$› ‹$A_P \sharp{*}\ P$› ‹$A_P \sharp{*}\ A_Q{}'$› ‹$A_Q{}' \sharp{*}\ \Psi$› ‹$A_Q{}' \sharp{*}\ P$› ‹$xvec \sharp{*}\ P$› ‹$A_P \sharp{*}\ M$› ‹$A_Q{}' \sharp{*}\ M'$›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
with‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) \longmapsto \tau \prec (\!|\nu y|\!)((\!|\nu{*}xvec|\!)(([(x, y)] \cdot P') \parallel ([(x, y)] \cdot Q')))$
by(*rule-tac Scope*) *auto*
moreover from ‹$x \sharp P$› ‹$y \sharp P$› ‹$y \sharp Q$› **have** $(\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) = (\!|\nu x|\!)([(x, y)] \cdot (P \parallel ([(x, y)] \cdot Q)))$
by(*subst alphaRes*[*of x*]) (*auto simp add*: *calc-atm fresh-left name-swap*)
with ‹$x \sharp P$› ‹$y \sharp P$› **have** $(\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) = (\!|\nu x|\!)(P \parallel Q)$
by(*simp add*: *eqvts*)
moreover from ‹$y \sharp P'$› ‹$y \sharp Q'$› ‹$x \sharp xvec$› ‹$y \sharp xvec$› **have** $(\!|\nu y|\!)((\!|\nu{*}xvec|\!)(([(x, y)] \cdot P') \parallel ([(x, y)] \cdot Q'))) = (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
by(*subst alphaRes*[*of y*]) (*auto simp add*: *resChainFresh calc-atm eqvts fresh-left name-swap*)
ultimately have $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto \tau \prec (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
by *simp*
moreover from ‹$x \sharp \Psi$› ‹$x \sharp P'$› ‹$xvec \sharp{*}\ \Psi$› **have** $(\Psi, (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P' \parallel Q')), (\!|\nu{*}xvec|\!)(P' \parallel (\!|\nu x|\!)Q')) \in Rel$
by(*rule C2*)
ultimately show *?case* **by** *blast*
qed
next

case(*cComm2* Ψ_Q *M xvec N* P' A_P Ψ_P *K* xQ' A_Q)
have *QTrans*: $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)\, Q \longmapsto K(\!|N|\!) \prec xQ'$ **and** *FrQ*: *extractFrame*$((\!|\nu x|\!)\, Q) = \langle A_Q, \Psi_Q\rangle$ **by** *fact+*
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **and** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact+*
from *PTrans* ‹$x \sharp P$› **have** $x \sharp N$ **and** $x \sharp P'$ **using** ‹*xvec* $\sharp*$ *x*› ‹*xvec* $\sharp*$ *M*› ‹*distinct xvec*›
by(*force intro*: *outputFreshDerivative*)+
have $x \sharp (\!|\nu x|\!)\, Q$ **by**(*simp add*: *abs-fresh*)
with *FrQ* ‹$A_Q \sharp* x$› **have** $x \sharp \Psi_Q$
by(*drule-tac extractFrameFresh*) *auto*
from ‹$x \sharp P$› *FrP* ‹$A_P \sharp* x$› **have** $x \sharp \Psi_P$
by(*drule-tac extractFrameFresh*) *auto*
from ‹$A_P \sharp* (\!|\nu x|\!)\, Q$› ‹$A_P \sharp* x$› **have** $A_P \sharp* Q$ **by** *simp*
from ‹$A_Q \sharp* (\!|\nu x|\!)\, Q$› ‹$A_Q \sharp* x$› **have** $A_Q \sharp* Q$ **by** *simp*
from ‹*xvec* $\sharp*$ *x*› ‹*xvec* $\sharp*$ $(\!|\nu x|\!)\, Q$› **have** *xvec* $\sharp*$ *Q* **by** *simp*

from *PTrans FrP* ‹*distinct* A_P› ‹$x \sharp P$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* x$› ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹*xvec* $\sharp*$ *M*› ‹*distinct xvec*›
obtain M' **where** $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $x \sharp M'$ **and** $A_Q \sharp* M'$
by(*rule-tac* $B{=}x\#A_Q$ **in** *obtainPrefix*) (*assumption* | *force simp add*: *fresh-star-list-cons*)+
hence *MeqM'*: $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition*)
with ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*blast intro*: *chanEqTrans chanEqSym*)
hence $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition*)
with *QTrans FrQ* ‹*distinct* A_Q› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* ((\!|\nu x|\!)\, Q)$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
have $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)\, Q \longmapsto M'(\!|N|\!) \prec xQ'$ **by**(*force intro*: *inputRenameSubject*)

moreover from ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› **have** $x \sharp \Psi \otimes \Psi_P$ **by** *force*
moreover note ‹$x \sharp M'$› ‹$x \sharp N$›
moreover from *QTrans* ‹$x \sharp N$› **have** $x \sharp xQ'$ **by**(*force dest*: *inputFreshDerivative simp add*: *abs-fresh*)
ultimately show *?case*
proof(*induct rule*: *resInputCases*)
case(*cRes* Q')
have *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|N|\!) \prec Q'$ **by** *fact*
with ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* N$› **have** $A_Q \sharp* Q'$
by(*rule-tac inputFreshChainDerivative*)

with ‹*extractFrame*$((\!|\nu x|\!)\, Q) = \langle A_Q, \Psi_Q\rangle$› **have** *FrxQ*: $(\!|\nu x|\!)$(*extractFrame* $Q) = \langle A_Q, \Psi_Q\rangle$ **by** *simp*
then obtain *y* $A_Q{}'$ **where** *A*: $A_Q = y\#A_Q{}'$ **by**(*case-tac* A_Q) *auto*
with ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* P'$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp*$ *xvec*› ‹$A_Q \sharp* M'$› ‹$A_Q \sharp* Q'$› ‹$A_Q \sharp* N$›
have $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* P$ **and** $A_Q{}' \sharp* \Psi_P$ **and** $A_Q{}' \sharp* Q$ **and** $A_Q \sharp*$

xvec **and** $A_Q \sharp* Q'$
and $y \sharp \Psi$ **and** $y \sharp P$ **and** $y \sharp P'$ **and** $y \sharp \Psi_P$ **and** $y \sharp Q$ **and** $y \sharp$ *xvec* **and** $y \sharp M'$ **and** $y \sharp Q'$ **and** $y \sharp N$
and $A_Q' \sharp* M'$
by(*simp*)+
from A ‹$A_P \sharp* A_Q$› **have** $A_P \sharp* A_Q'$ **and** $y \sharp A_P$ **by**(*simp add*: *fresh-star-list-cons*)+
from A ‹$A_Q \sharp* x$› **have** $x \neq y$ **and** $x \sharp A_Q'$ **by**(*simp add*: *fresh-list-cons*)+

with A ‹*distinct* A_Q› **have** $y \sharp A_Q'$
by(*induct* A_Q') (*auto simp add*: *fresh-list-nil fresh-list-cons*)

note *PTrans FrP*
moreover from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|N|\!) \prec Q'$› **have** $([(x, y)] \cdot (\Psi \otimes \Psi_P)) \rhd ([(x, y)] \cdot Q) \longmapsto([(x, y)] \cdot (M'(\!|N|\!) \prec Q'))$
by(*rule semantics.eqvt*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$y \sharp \Psi_P$› ‹$x \sharp M'$› ‹$y \sharp M'$› ‹$x \sharp N$› ‹$y \sharp N$›
have $\Psi \otimes \Psi_P \rhd ([(x, y)] \cdot Q) \longmapsto M'(\!|N|\!) \prec ([(x, y)] \cdot Q')$
by(*simp add*: *eqvts*)
moreover from A ‹$A_Q \sharp* x$› *FrxQ* **have** *FrQ*: *extractFrame*$([(x, y)] \cdot Q) = \langle A_Q', \Psi_Q\rangle$ **and** $y \sharp$ *extractFrame* Q
by(*clarsimp simp add*: *alpha′ eqvts frame.inject fresh-list-cons name-swap*)+
moreover from ‹$A_P \sharp* Q$› **have** $([(x, y)] \cdot A_P) \sharp* ([(x, y)] \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_P \sharp* x$› ‹$y \sharp A_P$› **have** $A_P \sharp* ([(x, y)] \cdot Q)$ **by** *simp*
moreover from ‹$A_Q' \sharp* Q$› **have** $([(x, y)] \cdot A_Q') \sharp* ([(x, y)] \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$x \sharp A_Q'$› ‹$y \sharp A_Q'$› **have** $A_Q' \sharp* ([(x, y)] \cdot Q)$ **by** *simp*
moreover from ‹*xvec* $\sharp* Q$› **have** $([(x, y)] \cdot$ *xvec*$) \sharp* ([(x, y)] \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* $\sharp* x$› ‹$y \sharp$ *xvec*› **have** *xvec* $\sharp* ([(x, y)] \cdot Q)$ **by** *simp*
ultimately have $\Psi \rhd (P \parallel ([(x, y)] \cdot Q)) \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel ([(x, y)] \cdot Q'))$
using *MeqM′* ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* A_Q'$› ‹$A_Q' \sharp* \Psi$› ‹$A_Q' \sharp* P$› ‹$A_P \sharp* M$› ‹$A_Q' \sharp* M'$›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
with‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) \longmapsto\tau \prec (\!|\nu y|\!)((\!|\nu*xvec|\!)(P' \parallel ([(x, y)] \cdot Q')))$
by(*rule-tac Scope*) *auto*
moreover from ‹$x \sharp P$› ‹$y \sharp P$› ‹$y \sharp Q$› **have** $(\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) = (\!|\nu x|\!)([(x, y)] \cdot (P \parallel ([(x, y)] \cdot Q)))$
by(*subst alphaRes*[*of x*]) (*auto simp add*: *calc-atm fresh-left name-swap*)
with ‹$x \sharp P$› ‹$y \sharp P$› **have** $(\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) = (\!|\nu x|\!)(P \parallel Q)$
by(*simp add*: *eqvts*)
moreover from ‹$x \sharp P'$› ‹$y \sharp P'$› ‹$y \sharp Q'$› ‹*xvec* $\sharp* x$› ‹$y \sharp$ *xvec*› **have** $(\!|\nu y|\!)((\!|\nu*xvec|\!)(P' \parallel ([(x, y)] \cdot Q'))) = (\!|\nu x|\!)((\!|\nu*xvec|\!)(P' \parallel Q'))$
by(*subst alphaRes*[*of y*]) (*auto simp add*: *resChainFresh calc-atm eqvts fresh-left name-swap*)
ultimately have $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto\tau \prec (\!|\nu x|\!)((\!|\nu*xvec|\!)(P' \parallel Q'))$
by *simp*

moreover from ‹$x \sharp \Psi$› ‹$x \sharp P'$› ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu x|\!)((\!|\nu*xvec|\!)(P' \parallel Q')), (\!|\nu*xvec|\!)(P' \parallel (\!|\nu x|\!)Q')) \in Rel$
by(*rule C2*)
ultimately show *?case* **by** *blast*
qed
qed
qed
qed

lemma *scopeExtRight*:
fixes x :: *name*
and P :: $('a, 'b, 'c)$ *psi*
and Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $x \sharp P$
and $x \sharp \Psi$
and *eqvt Rel*
and *C1*: $\bigwedge \Psi'\ R.\ (\Psi, R, R) \in Rel$
and *C2*: $\bigwedge y\ \Psi'\ R\ S\ zvec.\ [\![y \sharp \Psi'; y \sharp R; zvec \sharp* \Psi']\!] \Longrightarrow (\Psi', (\!|\nu*zvec|\!)(R \parallel (\!|\nu y|\!)S), (\!|\nu y|\!)((\!|\nu*zvec|\!)(R \parallel S))) \in Rel$

shows $\Psi \rhd P \parallel (\!|\nu x|\!)Q \rightsquigarrow[Rel]\ (\!|\nu x|\!)(P \parallel Q)$
proof −
note ‹*eqvt Rel*› ‹$x \sharp \Psi$›
moreover from ‹$x \sharp P$› **have** $x \sharp P \parallel (\!|\nu x|\!)Q$ **by**(*simp add*: *abs-fresh*)
moreover from ‹$x \sharp P$› **have** $x \sharp (\!|\nu x|\!)(P \parallel Q)$ **by**(*simp add*: *abs-fresh*)
ultimately show *?thesis*
proof(*induct rule*: *simIFresh*[*of* - - - - - ()])
case(*cSim* α *xPQ*)
from ‹$bn\ \alpha \sharp* (P \parallel (\!|\nu x|\!)Q)$› ‹$x \sharp \alpha$› **have** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ **by** *simp+*
note ‹$\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \longmapsto \alpha \prec xPQ$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp xPQ$› ‹$bn\ \alpha \sharp* \Psi$›
moreover from ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› **have** $bn\ \alpha \sharp* (P \parallel Q)$ **by** *simp*
ultimately show *?case* **using** ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$x \sharp \alpha$›
proof(*induct rule*: *resCases*)
case(*cOpen M xvec1 xvec2 y N PQ*)
from ‹$x \sharp M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle$› **have** $x \sharp xvec1$ **and** $x \neq y$ **and** $x \sharp xvec2$ **and** $x \sharp M$ **by** *simp+*
from ‹$xvec1 \sharp* (P \parallel Q)$› ‹$xvec2 \sharp* (P \parallel Q)$› ‹$y \sharp (P \parallel Q)$›
have $(xvec1@xvec2) \sharp* P$ **and** $(xvec1@xvec2) \sharp* Q$ **and** $y \sharp P$ **and** $y \sharp Q$
by *simp+*
from ‹$\Psi \rhd P \parallel Q \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot PQ)$›
have $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot (P \parallel Q)) \longmapsto ([(x, y)] \cdot (M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot PQ)))$
by(*rule semantics.eqvt*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp P$› ‹$y \sharp P$› ‹$x \sharp M$› ‹$y \sharp M$› ‹$x \sharp xvec1$› ‹$x \sharp xvec2$› ‹$y \sharp xvec1$› ‹$y \sharp xvec2$›

have $\Psi \rhd P \parallel ([(x, y)] \cdot Q) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec PQ$
by(*simp add*: *eqvts*)
moreover from ‹*xvec1* $\sharp*$ Ψ› ‹*xvec2* $\sharp*$ Ψ› **have** (*xvec1*@*xvec2*) $\sharp*$ Ψ **by** *simp*
moreover note ‹(*xvec1*@*xvec2*) $\sharp*$ *P*›
moreover from ‹(*xvec1*@*xvec2*) $\sharp*$ *Q*› **have** $([(x, y)] \cdot (xvec1@xvec2)) \sharp* ([(x, y)] \cdot Q)$
by(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*x* $\sharp$ *xvec1*› ‹*x* $\sharp$ *xvec2*› ‹*y* $\sharp$ *xvec1*› ‹*y* $\sharp$ *xvec2*› **have** (*xvec1*@*xvec2*) $\sharp*$ $([(x, y)] \cdot Q)$
by(*auto simp add*: *eqvts*)
moreover from ‹*xvec1* $\sharp*$ *M*› ‹*xvec2* $\sharp*$ *M*› **have** (*xvec1*@*xvec2*) $\sharp*$ *M* **by** *simp*
ultimately show *?case*
proof(*induct rule*: *parOutputCases*[**where** *C*=*y*])
case(*cPar1* P' A_Q Ψ_Q)
from ‹*y* $\sharp$ *xvec1*› ‹*y* $\sharp$ *xvec2*› **have** *y* $\sharp$ (*xvec1*@*xvec2*) **by**(*auto simp add*: *fresh-list-append*)
with ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec P'$› ‹(*xvec1*@*xvec2*) $\sharp*$ *M*› ‹*y* $\sharp$ *P*›
‹*distinct xvec1*› ‹*distinct xvec2*› ‹*xvec1* $\sharp*$ *xvec2*›
have *y* $\sharp$ *N* **by**(*force intro*: *outputFreshDerivative*)
with ‹*y* $\in$ *supp N*› **have** *False* **by**(*simp add*: *fresh-def*)
thus *?case* **by** *simp*
next
case(*cPar2* Q' A_P Ψ_P)
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
with ‹*y* $\sharp$ *P*› ‹A_P $\sharp*$ *y*› **have** *y* $\sharp$ Ψ_P
apply(*drule-tac extractFrameFresh*)
by(*simp add*: *frameResChainFresh*) (*simp add*: *fresh-def name-list-supp*)
from ‹$\Psi \otimes \Psi_P \rhd ([(x, y)] \cdot Q) \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec Q'$› ‹*y* $\in$ *supp N*› ‹*y* $\sharp$ Ψ› ‹*y* $\sharp$ Ψ_P› ‹*y* $\sharp$ *M*› ‹*y* $\sharp$ *xvec1*› ‹*y* $\sharp$ *xvec2*›
have $\Psi \otimes \Psi_P \rhd (\!|\nu y|\!)(([(x, y)] \cdot Q)) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec Q'$ **by**(*force intro*: *Open*)
with ‹*y* $\sharp$ *Q*› **have** $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec Q'$
by(*simp add*: *alphaRes*)
moreover from ‹$A_P \sharp* ([(x, y)] \cdot Q)$› **have** $A_P \sharp* (\!|\nu y|\!)([(x, y)] \cdot Q)$ **by**(*auto simp add*: *fresh-star-def abs-fresh*)
with ‹*y* $\sharp$ *Q*› **have** $A_P \sharp* (\!|\nu x|\!)Q$ **by**(*simp add*: *alphaRes*)
ultimately have $\Psi \rhd P \parallel ((\!|\nu x|\!)Q) \longmapsto M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec (P \parallel Q')$
using *FrP* ‹(*xvec1*@*xvec2*) $\sharp*$ *P*› ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ *M*› ‹*y* $\sharp$ *P*› ‹A_P $\sharp*$ (*xvec1*@*xvec2*)› ‹A_P $\sharp*$ *y*› ‹A_P $\sharp*$ *N*›
by(*rule-tac Par2*) *auto*
moreover have $(\Psi, P \parallel Q', P \parallel Q') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
next

case(*cRes PQ*)
from ‹$\Psi \rhd P \parallel Q \longmapsto\alpha \prec PQ$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp*$ *subject* α›
show *?case*
proof(*induct rule*: *parCases*[**where** $C{=}x$])
case(*cPar1* $P'\ A_Q\ \Psi_Q$)
note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$›
moreover with ‹$x \sharp P$› ‹$x \sharp \alpha$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹*distinct*($bn\ \alpha$)›
have $x \sharp P'$ **by**(*force dest*: *freeFreshDerivative*)
with ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› **have** *extractFrame*($(\!|\nu x|\!)\,Q$) $= \langle(x\#A_Q), \Psi_Q\rangle$
by *simp*
moreover from ‹$bn\ \alpha \sharp* Q$› **have** $bn\ \alpha \sharp* ((\!|\nu x|\!)\,Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$x \sharp \Psi$› ‹$A_Q \sharp* \Psi$› **have** $(x\#A_Q) \sharp* \Psi$ **by** *simp*
moreover from ‹$x \sharp P$› ‹$A_Q \sharp* P$› **have** $(x\#A_Q) \sharp* P$ **by** *simp*
moreover from ‹$x \sharp \alpha$› ‹$A_Q \sharp* \alpha$› **have** $(x\#A_Q) \sharp* \alpha$ **by** *simp*
ultimately have $\Psi \rhd P \parallel (\!|\nu x|\!)\,Q \longmapsto\alpha \prec (P' \parallel (\!|\nu x|\!)\,Q)$
by(*rule Par1*)
moreover from ‹$x \sharp P'$› ‹$x \sharp \Psi$› **have** $(\Psi, (\!|\nu*[]|\!)(P' \parallel ((\!|\nu x|\!)\,Q)), (\!|\nu x|\!)((\!|\nu*[]|\!)(P' \parallel Q))) \in Rel$
by(*rule-tac C2*) *auto*
ultimately show *?case*
by *force*
next
case(*cPar2* $Q'\ A_P\ \Psi_P$)
have *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact*
with ‹$x \sharp P$› ‹$A_P \sharp* x$› **have** $x \sharp \Psi_P$
apply(*drule-tac extractFrameFresh*)
by(*simp add*: *frameResChainFresh*) (*simp add*: *fresh-def name-list-supp*)
from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto\alpha \prec Q'$› ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp \alpha$›
have $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)\,Q \longmapsto\alpha \prec (\!|\nu x|\!)\,Q'$
by(*rule-tac Scope*) *auto*
moreover note *FrP* ‹$bn\ \alpha \sharp* P$› ‹$A_P \sharp* \Psi$›
moreover from ‹$A_P \sharp* Q$› **have** $A_P \sharp* (\!|\nu x|\!)\,Q$ **by**(*simp add*: *fresh-star-def abs-fresh*)
ultimately have $\Psi \rhd P \parallel (\!|\nu x|\!)\,Q \longmapsto\alpha \prec (P \parallel (\!|\nu x|\!)\,Q')$ **using** ‹$A_P \sharp* \alpha$›
by(*rule Par2*)
moreover from ‹$x \sharp P$› ‹$x \sharp \Psi$› **have** $(\Psi, (\!|\nu*[]|\!)(P \parallel ((\!|\nu x|\!)\,Q')), (\!|\nu x|\!)((\!|\nu*[]|\!)(P \parallel Q'))) \in Rel$
by(*rule-tac C2*) *auto*
ultimately show *?case*
by *force*
next
case(*cComm1* $\Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K\ xvec\ Q'\ A_Q$)
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$ **and** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **by** *fact+*
have *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **and** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact+*

from *FrP* ‹$x \sharp P$› **have** $x \sharp \langle A_P, \Psi_P\rangle$ **by**(*drule-tac extractFrameFresh*) *simp*
with ‹$A_P \sharp* x$› **have** $x \sharp \Psi_P$ **by**(*simp add*: *frameResChainFresh*) (*simp add*: *fresh-def name-list-supp*)
from *PTrans FrP* ‹*distinct* A_P› ‹$x \sharp P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* x$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_Q$›
obtain M' **where** *MeqM'*: $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $x \sharp M'$ **and** $A_Q \sharp* M'$
by(*rule-tac* $B{=}x\#A_Q$ **in** *obtainPrefix*) (*assumption* | *force simp add*: *fresh-star-list-cons*)+

from *MeqM'* **have** *MeqM'*: $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Composition Commutativity*)
with ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*blast intro*: *chanEqTrans chanEqSym*)
hence $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*metis statEqEnt Associativity Composition Commutativity*)
with *QTrans FrQ* ‹*distinct* A_Q› **have** *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto M'(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
by(*rule-tac outputRenameSubject*) (*assumption* | *force*)+
show *?case*
proof(*case-tac* $x \in$ *supp* N)
note *PTrans FrP*
moreover assume $x \in$ *supp* N
hence $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto M'(\!|\nu{*}([]@(x\#xvec))|\!)\langle N\rangle \prec Q'$ **using** *QTrans* ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp M'$› ‹$xvec \sharp* x$›
by(*rule-tac Open*) (*assumption* | *force simp add*: *fresh-list-nil*)+
hence *QTrans*: $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto M'(\!|\nu{*}(x\#xvec)|\!)\langle N\rangle \prec Q'$ **by** *simp*
moreover from ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› **have** *extractFrame*$((\!|\nu x|\!)Q) = \langle (x\#A_Q), \Psi_Q\rangle$
by *simp*
moreover note *MeqM'* ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$›
moreover from ‹$A_P \sharp* Q$› **have** $A_P \sharp* ((\!|\nu x|\!)Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* x$› **have** $A_P \sharp* (x\#A_Q)$
by(*simp add*: *fresh-star-def fresh-list-cons*) (*auto simp add*: *fresh-def name-list-supp*)
moreover from ‹$x \sharp \Psi$› ‹$A_Q \sharp* \Psi$› **have** $(x\#A_Q) \sharp* \Psi$ **by** *simp*
moreover from ‹$x \sharp P$› ‹$A_Q \sharp* P$› **have** $(x\#A_Q) \sharp* P$ **by** *simp*
moreover from ‹$x \sharp M'$› ‹$A_Q \sharp* M'$› **have** $(x\#A_Q) \sharp* M'$ **by** *simp*
moreover from ‹$A_Q \sharp* Q$› **have** $(x\#A_Q) \sharp* ((\!|\nu x|\!)Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$x \sharp P$› ‹$xvec \sharp* P$› **have** $(x\#xvec) \sharp* P$ **by**(*simp*)
ultimately have $\Psi \rhd P \parallel (\!|\nu x|\!)Q \longmapsto \tau \prec (\!|\nu{*}(x\#xvec)|\!)(P' \parallel Q')$ **using** ‹$A_P \sharp* M$›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
moreover have $(\Psi, (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P' \parallel Q')), (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P' \parallel Q'))) \in$ *Rel* **by**(*rule C1*)
ultimately show *?case* **by** *force*
next

note *PTrans FrP*
moreover assume $x \notin supp\ N$
hence $x \sharp N$ **by**(*simp add*: *fresh-def*)
with *QTrans* ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp M'$› ‹$xvec \sharp* x$›
have *QTrans*: $\Psi \otimes \Psi_P \rhd (\!|\nu x|\!)Q \longmapsto M'(\!|\nu{*}xvec|\!)\langle N\rangle \prec (\!|\nu x|\!)Q'$
by(*rule-tac Scope*) (*assumption* | *force*)+
moreover from *PTrans* ‹$x \sharp P$› ‹$x \sharp N$› **have** $x \sharp P'$ **by**(*rule input-FreshDerivative*)
with ‹$extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$› **have** $extractFrame((\!|\nu x|\!)Q) = \langle(x\#A_Q), \Psi_Q\rangle$
by *simp*
moreover note *MeqM'* ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$›
moreover from ‹$A_P \sharp* Q$› **have** $A_P \sharp* ((\!|\nu x|\!)Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* x$› **have** $A_P \sharp* (x\#A_Q)$
by(*simp add*: *fresh-star-def fresh-list-cons*) (*auto simp add*: *fresh-def name-list-supp*)
moreover from ‹$x \sharp \Psi$› ‹$A_Q \sharp* \Psi$› **have** $(x\#A_Q) \sharp* \Psi$ **by** *simp*
moreover from ‹$x \sharp P$› ‹$A_Q \sharp* P$› **have** $(x\#A_Q) \sharp* P$ **by** *simp*
moreover from ‹$x \sharp M'$› ‹$A_Q \sharp* M'$› **have** $(x\#A_Q) \sharp* M'$ **by** *simp*
moreover from ‹$A_Q \sharp* Q$› **have** $(x\#A_Q) \sharp* ((\!|\nu x|\!)Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$x \sharp P$› ‹$xvec \sharp* P$› **have** $(x\#xvec) \sharp* P$ **by**(*simp*)
ultimately have $\Psi \rhd P \parallel (\!|\nu x|\!)Q \longmapsto\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel (\!|\nu x|\!)Q')$ **using** ‹$A_P \sharp* M$›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
moreover from ‹$x \sharp \Psi$› ‹$x \sharp P'$› ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu{*}xvec|\!)(P' \parallel (\!|\nu x|\!)Q'), (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P' \parallel Q'))) \in Rel$ **by**(*rule C2*)
ultimately show *?case* **by** *blast*
qed
next
case(*cComm2* Ψ_Q M $xvec$ N P' A_P Ψ_P K Q' A_Q)
have *PTrans*: $\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **and** *FrP*: $extractFrame\ P = \langle A_P, \Psi_P\rangle$ **by** *fact*+
have *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$ **and** *FrQ*: $extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*+
from *FrP* ‹$x \sharp P$› **have** $x \sharp \langle A_P, \Psi_P\rangle$ **by**(*drule-tac extractFrameFresh*) *simp*
with ‹$A_P \sharp* x$› **have** $x \sharp \Psi_P$ **by**(*simp add*: *frameResChainFresh*) (*simp add*: *fresh-def name-list-supp*)
from *PTrans FrP* ‹*distinct* A_P› ‹$x \sharp P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi_Q$› ‹$A_P \sharp* x$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_Q \sharp* P$› ‹$A_P \sharp* A_Q$› ‹$xvec \sharp* M$› ‹*distinct xvec*›
obtain M' **where** *MeqM'*: $(\Psi \otimes \Psi_Q) \otimes \Psi_P \vdash M \leftrightarrow M'$ **and** $x \sharp M'$ **and** $A_Q \sharp* M'$
by(*rule-tac* $B{=}x\#A_Q$ **in** *obtainPrefix*) (*assumption* | *force simp add*: *fresh-star-list-cons*)+

from *MeqM'* **have** *MeqM'*: $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition*)
with ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow M'$

by(*blast intro*: *chanEqTrans chanEqSym*)
hence $(\Psi \otimes \Psi_P) \otimes \Psi_Q \vdash K \leftrightarrow M'$
by(*metis statEqEnt Associativity Commutativity Composition*)
with *QTrans FrQ* ‹*distinct* A_Q› **have** *QTrans*: $\Psi \otimes \Psi_P \rhd Q \longmapsto M'\langle\!\langle N\rangle\!\rangle \prec Q'$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹$A_Q \sharp* M'$›
by(*rule-tac inputRenameSubject*) (*assumption* | *force*)+

from *PTrans* ‹$x \sharp P$› ‹$xvec \sharp* x$› ‹*distinct xvec*› ‹$xvec \sharp* M$›
have $x \sharp N$ **and** $x \sharp P'$ **by**(*force intro*: *outputFreshDerivative*)+
from *QTrans* ‹$x \sharp \Psi$› ‹$x \sharp \Psi_P$› ‹$x \sharp M'$› ‹$x \sharp N$› **have** *QTrans*: $\Psi \otimes \Psi_P \rhd \langle\!\langle\nu x\rangle\!\rangle Q \longmapsto M'\langle\!\langle N\rangle\!\rangle \prec \langle\!\langle\nu x\rangle\!\rangle Q'$
by(*rule-tac Scope*) (*assumption* | *force*)+

note *PTrans FrP QTrans*
moreover with ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$› **have** *extractFrame*$(\langle\!\langle\nu x\rangle\!\rangle Q) = \langle(x\#A_Q), \Psi_Q\rangle$
by *simp*
moreover note *MeqM'* ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$›
moreover from ‹$A_P \sharp* Q$› **have** $A_P \sharp* (\langle\!\langle\nu x\rangle\!\rangle Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$A_P \sharp* A_Q$› ‹$A_P \sharp* x$› **have** $A_P \sharp* (x\#A_Q)$
by(*simp add*: *fresh-star-def fresh-list-cons*) (*auto simp add*: *fresh-def name-list-supp*)
moreover from ‹$x \sharp \Psi$› ‹$A_Q \sharp* \Psi$› **have** $(x\#A_Q) \sharp* \Psi$ **by** *simp*
moreover from ‹$x \sharp P$› ‹$A_Q \sharp* P$› **have** $(x\#A_Q) \sharp* P$ **by** *simp*
moreover from ‹$x \sharp M'$› ‹$A_Q \sharp* M'$› **have** $(x\#A_Q) \sharp* M'$ **by** *simp*
moreover from ‹$A_Q \sharp* Q$› **have** $(x\#A_Q) \sharp* (\langle\!\langle\nu x\rangle\!\rangle Q)$ **by**(*simp add*: *fresh-star-def abs-fresh*)
moreover from ‹$xvec \sharp* Q$› **have** $(x\#xvec) \sharp* (\langle\!\langle\nu x\rangle\!\rangle Q)$ **by**(*simp add*: *abs-fresh fresh-star-def*)
ultimately have $\Psi \rhd P \parallel \langle\!\langle\nu x\rangle\!\rangle Q \longmapsto\tau \prec \langle\!\langle\nu{*}xvec\rangle\!\rangle(P' \parallel \langle\!\langle\nu x\rangle\!\rangle Q')$ **using** ‹$A_P \sharp* M$›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
moreover from ‹$x \sharp \Psi$› ‹$x \sharp P'$› ‹$xvec \sharp* \Psi$› **have** $(\Psi, \langle\!\langle\nu{*}xvec\rangle\!\rangle(P' \parallel \langle\!\langle\nu x\rangle\!\rangle Q'), \langle\!\langle\nu x\rangle\!\rangle(\langle\!\langle\nu{*}xvec\rangle\!\rangle(P' \parallel Q'))) \in Rel$ **by**(*rule C2*)
ultimately show *?case* **by** *blast*
qed
qed
qed
qed

lemma *simParComm*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes *eqvt Rel*
and *C1*: $\bigwedge\Psi'\ R\ S.\ (\Psi', R \parallel S, S \parallel R) \in Rel$

and *C2*: $\bigwedge\Psi'\ R\ S\ xvec.\ [\![(\Psi',\ R,\ S) \in Rel;\ xvec \sharp* \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu*xvec|\!)R,\ (\!|\nu*xvec|\!)S) \in Rel$

shows $\Psi \rhd P \parallel Q \rightsquigarrow[Rel]\ Q \parallel P$
using ‹*eqvt Rel*›
proof(*induct rule*: *simI*[*of* - - - - ()])
case(*cSim* α *PQ*)
from ‹$bn\ \alpha \sharp* (P \parallel Q)$› **have** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp* P$ **by** *simp*+
with ‹$\Psi \rhd Q \parallel P \longmapsto\alpha \prec PQ$› ‹$bn\ \alpha \sharp* \Psi$› **show** *?case* **using** ‹$bn\ \alpha \sharp*$ *subject* α›
proof(*induct rule*: *parCases*[**where** C=()])
case(*cPar1* $Q'\ A_P\ \Psi_P$)
from ‹$\Psi \otimes \Psi_P \rhd Q \longmapsto\alpha\prec Q'$› ‹*extractFrame* $P = \langle A_P,\ \Psi_P\rangle$› ‹$bn\ \alpha \sharp* P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \alpha$›
have $\Psi \rhd P \parallel Q \longmapsto\alpha \prec (P \parallel Q')$ **by**(*rule Par2*)
moreover have $(\Psi,\ P \parallel Q',\ Q' \parallel P) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* $P'\ A_Q\ \Psi_Q$)
from ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto\alpha \prec P'$› ‹*extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$› ‹$bn\ \alpha \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$›
have $\Psi \rhd P \parallel Q \longmapsto\alpha \prec (P' \parallel Q)$ **by**(*rule Par1*)
moreover have $(\Psi,\ P' \parallel Q,\ Q \parallel P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* $\Psi_P\ M\ N\ Q'\ A_Q\ \Psi_Q\ K\ xvec\ P'\ A_P$)
note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› ‹*extractFrame* $P = \langle A_P,\ \Psi_P\rangle$›
‹$\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|N|\!) \prec Q'$› ‹*extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$›
moreover from ‹$\Psi \otimes \Psi_Q \otimes \Psi_P \vdash M \leftrightarrow K$›
have $\Psi \otimes \Psi_Q \otimes \Psi_P \vdash K \leftrightarrow M$
by(*rule chanEqSym*)
hence $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow M$
by(*blast intro*: *statEqEnt compositionSym Commutativity*)
ultimately have $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel Q')$
using ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* A_P$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$xvec \sharp* Q$› ‹$A_P \sharp* K$› ‹$A_Q \sharp* M$›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
moreover have $(\Psi,\ P' \parallel Q',\ Q' \parallel P') \in Rel$ **by**(*rule C1*)
hence $(\Psi,\ (\!|\nu*xvec|\!)(P' \parallel Q'),\ (\!|\nu*xvec|\!)(Q' \parallel P')) \in Rel$ **using** ‹$xvec \sharp* \Psi$› **by**(*rule C2*)
ultimately show *?case* **by** *blast*
next
case(*cComm2* $\Psi_P\ M\ xvec\ N\ Q'\ A_Q\ \Psi_Q\ K\ P'\ A_P$)
note ‹$\Psi \otimes \Psi_Q \rhd P \longmapsto K(\!|N|\!) \prec P'$› ‹*extractFrame* $P = \langle A_P,\ \Psi_P\rangle$›
‹$\Psi \otimes \Psi_P \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› ‹*extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$›
moreover from ‹$\Psi \otimes \Psi_Q \otimes \Psi_P \vdash M \leftrightarrow K$›
have $\Psi \otimes \Psi_Q \otimes \Psi_P \vdash K \leftrightarrow M$
by(*rule chanEqSym*)
hence $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash K \leftrightarrow M$

by(*blast intro*: *statEqEnt compositionSym Commutativity*)
ultimately have $\Psi \rhd P \parallel Q \longmapsto\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q')$
using ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_Q \sharp* A_P$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$xvec \sharp* P$› ‹$A_P \sharp* K$› ‹$A_Q \sharp* M$›
by(*rule-tac Comm1*) (*assumption* | *simp add*: *freshChainSym*)+
moreover have $(\Psi, P' \parallel Q', Q' \parallel P') \in Rel$ **by**(*rule C1*)
hence $(\Psi, (\!|\nu{*}xvec|\!)(P' \parallel Q'), (\!|\nu{*}xvec|\!)(Q' \parallel P')) \in Rel$ **using** ‹$xvec \sharp* \Psi$› **by**(*rule C2*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *bangExtLeft*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

assumes *guarded P*
and $\bigwedge\Psi'\ Q.\ (\Psi', Q, Q) \in Rel$

shows $\Psi \rhd !P \rightsquigarrow[Rel]\ P \parallel !P$
using *assms*
by(*auto simp add*: *simulation-def dest*: *Bang*)

lemma *bangExtRight*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

assumes *C1*: $\bigwedge\Psi'\ Q.\ (\Psi', Q, Q) \in Rel$

shows $\Psi \rhd P \parallel !P \rightsquigarrow[Rel]\ !P$
proof(*auto simp add*: *simulation-def*)
fix $\alpha\ P'$
assume $\Psi \rhd !P \longmapsto\alpha \prec P'$
hence $\Psi \rhd P \parallel !P \longmapsto\alpha \prec P'$
apply −
by(*ind-cases* $\Psi \rhd !P \longmapsto\alpha \prec P'$) (*auto simp add*: *psi.inject*)
moreover have $(\Psi, P', P') \in Rel$ **by**(*rule C1*)
ultimately show $\exists P''.\ \Psi \rhd P \parallel !P \longmapsto\alpha \prec P'' \wedge (\Psi, P'', P') \in Rel$
by *blast*
qed

end

end

theory *Structural-Congruence*
imports *Agent*
begin

inductive *structCong* :: (('a::*fs-name*), ('b::*fs-name*), ('c::*fs-name*)) *psi* ⇒ ('a, 'b, 'c) *psi* ⇒ *bool* (‹- $\equiv_s$ -› [70, 70] 70)
where
  Refl: $P \equiv_s P$
| *Sym*: $P \equiv_s Q \Longrightarrow Q \equiv_s P$
| *Trans*: $[\![P \equiv_s Q;\ Q \equiv_s R]\!] \Longrightarrow P \equiv_s R$

| *ParComm*: $P \parallel Q \equiv_s Q \parallel P$
| *ParAssoc*: $(P \parallel Q) \parallel R \equiv_s P \parallel (Q \parallel R)$
| *ParId*: $P \parallel \mathbf{0} \equiv_s P$

| *ResNil*: $(\!|\nu x|\!)\mathbf{0} \equiv_s \mathbf{0}$
| *ResComm*: $(\!|\nu x|\!)((\!|\nu y|\!)P) \equiv_s (\!|\nu y|\!)((\!|\nu x|\!)P)$
| *ScopeExtPar*: $x \sharp P \Longrightarrow (\!|\nu x|\!)(P \parallel Q) \equiv_s P \parallel (\!|\nu x|\!)Q$
| *InputRes*: $[\![x \sharp M;\ x \sharp xvec;\ x \sharp N]\!] \Longrightarrow (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \equiv_s M(\!|\lambda{*}xvec\ N|\!).((\!|\nu x|\!)P)$
| *OutputRes*: $[\![x \sharp M;\ x \sharp N]\!] \Longrightarrow (\!|\nu x|\!)(M\langle N\rangle.P) \equiv_s M\langle N\rangle.((\!|\nu x|\!)P)$
| *CaseRes*: $x \sharp (map\ fst\ Cs) \Longrightarrow (\!|\nu x|\!)(Cases\ Cs) \equiv_s Cases(map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs)$

| *BangUnfold*: $guarded\ P \Longrightarrow !P \equiv_s P \parallel !P$

end

theory *Bisim-Struct-Cong*
  imports *Bisim-Pres Sim-Struct-Cong Structural-Congruence*
begin

context *env* **begin**

lemma *bisimParComm*:
  fixes Ψ :: 'b
  and P :: ('a, 'b, 'c) *psi*
  and Q :: ('a, 'b, 'c) *psi*

  shows $\Psi \rhd P \parallel Q \sim Q \parallel P$
proof −
  let *?X* = {((Ψ::'b), $(\!|\nu{*}xvec|\!)$((P::('a, 'b, 'c) *psi*) ∥ Q), $(\!|\nu{*}xvec|\!)$(Q ∥ P)) | *xvec* Ψ P Q. *xvec* ♯* Ψ}

  have *eqvt ?X*
    by(*force simp add*: *eqvt-def pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *eqvts*)

  have (Ψ, P ∥ Q, Q ∥ P) ∈ *?X*
    apply *auto* **by**(*rule-tac x*=[] **in** *exI*) *auto*
  thus *?thesis*
  proof(*coinduct rule*: *bisimWeakCoinduct*)

case(*cStatEq* Ψ *PQ QP*)
from ‹(Ψ, *PQ*, *QP*) $\in$ *?X*›
obtain *xvec P Q* **where** *PFrQ*: *PQ* = $(\!|\nu{*}xvec|\!)(P \parallel Q)$ **and** *QFrP*: *QP* = $(\!|\nu{*}xvec|\!)(Q \parallel P)$ **and** *xvec* $\sharp*$ Ψ
by *auto*

obtain A_P Ψ_P **where** *FrP*: *extractFrame P* = $\langle A_P, \Psi_P\rangle$ **and** A_P $\sharp*$ Ψ **and** A_P $\sharp*$ *Q*
by(*rule-tac C*=(Ψ, *Q*) **in** *freshFrame*) *auto*
obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame Q* = $\langle A_Q, \Psi_Q\rangle$ **and** A_Q $\sharp*$ Ψ **and** A_Q $\sharp*$ A_P **and** A_Q $\sharp*$ Ψ_P
by(*rule-tac C*=(Ψ, A_P, Ψ_P) **in** *freshFrame*) *auto*
from *FrQ* ‹A_Q $\sharp*$ A_P› ‹A_P $\sharp*$ *Q*› **have** A_P $\sharp*$ Ψ_Q **by**(*force dest*: *extractFrameFreshChain*)
have $\langle(xvec@A_P@A_Q), \Psi \otimes \Psi_P \otimes \Psi_Q\rangle \simeq_F \langle(xvec@A_Q@A_P), \Psi \otimes \Psi_Q \otimes \Psi_P\rangle$
by(*simp add*: *frameChainAppend*)
(*metis frameResChainPres frameResChainComm frameNilStatEq compositionSym Associativity Commutativity FrameStatEqTrans*)
with *FrP FrQ PFrQ QFrP* ‹A_P $\sharp*$ Ψ_Q› ‹A_Q $\sharp*$ Ψ_P› ‹A_Q $\sharp*$ A_P› ‹*xvec* $\sharp*$ Ψ› ‹A_P $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ›
show *?case* **by**(*auto simp add*: *frameChainAppend*)
next
case(*cSim* Ψ *PQ QP*)
from ‹(Ψ, *PQ*, *QP*) $\in$ *?X*›
obtain *xvec P Q* **where** *PFrQ*: *PQ* = $(\!|\nu{*}xvec|\!)(P \parallel Q)$ **and** *QFrP*: *QP* = $(\!|\nu{*}xvec|\!)(Q \parallel P)$
and *xvec* $\sharp*$ Ψ
by *auto*
moreover have $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \rightsquigarrow[?X]\ (\!|\nu{*}xvec|\!)(Q \parallel P)$
proof −
have $\Psi \rhd P \parallel Q \rightsquigarrow[?X]\ Q \parallel P$
proof −
note ‹*eqvt ?X*›
moreover have $\bigwedge\Psi\ P\ Q.\ (\Psi, P \parallel Q, Q \parallel P) \in ?X$
apply *auto* **by**(*rule-tac x*=[] **in** *exI*) *auto*
moreover have $\bigwedge\Psi\ P\ Q\ xvec.\ [\![(\Psi, P, Q) \in ?X;\ xvec \sharp* \Psi]\!] \Longrightarrow (\Psi, (\!|\nu{*}xvec|\!)P, (\!|\nu{*}xvec|\!)Q) \in ?X$
apply(*induct xvec*, *auto*)
by(*rule-tac x*=*xvec@xveca* **in** *exI*) (*auto simp add*: *resChainAppend*)
ultimately show *?thesis* **by**(*rule simParComm*)
qed
moreover note ‹*eqvt ?X*› ‹*xvec* $\sharp*$ Ψ›
moreover have $\bigwedge\Psi\ P\ Q\ x.\ [\![(\Psi, P, Q) \in ?X;\ x \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu x|\!)P, (\!|\nu x|\!)Q) \in ?X$
apply *auto*
by(*rule-tac x*=*x#xvec* **in** *exI*) *auto*
ultimately show *?thesis* **by**(*rule resChainPres*)
qed
ultimately show *?case* **by** *simp*

next
case(*cExt* Ψ *PQ QP* Ψ')
from ‹(Ψ, *PQ*, *QP*) $\in$ *?X*›
obtain *xvec P Q* **where** *PFrQ*: $PQ = (\!|\nu{*}xvec|\!)(P \parallel Q)$ **and** *QFrP*: $QP = (\!|\nu{*}xvec|\!)(Q \parallel P)$
and *xvec* $\sharp*$ Ψ
by *auto*

obtain *p* **where** $(p \cdot xvec) \sharp* \Psi$
and $(p \cdot xvec) \sharp* P$
and $(p \cdot xvec) \sharp* Q$
and $(p \cdot xvec) \sharp* \Psi'$
and *S*: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm p*
by(*rule-tac c*=(Ψ, *P*, *Q*, Ψ') **in** *name-list-avoiding*) *auto*

from ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› *S* **have** $(\!|\nu{*}xvec|\!)(P \parallel Q) = (\!|\nu{*}(p \cdot xvec)|\!)(p \cdot (P \parallel Q))$
by(*subst resChainAlpha*) *auto*
hence *PQAlpha*: $(\!|\nu{*}xvec|\!)(P \parallel Q) = (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel (p \cdot Q))$
by(*simp add*: *eqvts*)

from ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› *S* **have** $(\!|\nu{*}xvec|\!)(Q \parallel P) = (\!|\nu{*}(p \cdot xvec)|\!)(p \cdot (Q \parallel P))$
by(*subst resChainAlpha*) *auto*
hence *QPAlpha*: $(\!|\nu{*}xvec|\!)(Q \parallel P) = (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot Q) \parallel (p \cdot P))$
by(*simp add*: *eqvts*)

from ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel (p \cdot Q)), (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot Q) \parallel (p \cdot P))) \in$ *?X*
by *auto*
with *PFrQ QFrP PQAlpha QPAlpha* **show** *?case* **by** *simp*
next
case(*cSym* Ψ *PR QR*)
thus *?case* **by** *blast*
qed
qed

lemma *bisimResComm*:
fixes *x* :: *name*
and Ψ :: *'b*
and *y* :: *name*
and *P* :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \sim (\!|\nu y|\!)((\!|\nu x|\!)P)$
proof(*cases x*=*y*)
case *True*
thus *?thesis* **by**(*blast intro*: *bisimReflexive*)
next
case *False*

{
fix x::*name* **and** y::*name* **and** P::($'a$, $'b$, $'c$) *psi*
assume $x \sharp \Psi$ **and** $y \sharp \Psi$
let $?X = \{((\Psi::'b), (\!|\nu x|\!)((\!|\nu y|\!)(P::('a, 'b, 'c)\ psi)), (\!|\nu y|\!)((\!|\nu x|\!)P)) \mid \Psi\ x\ y\ P.\ x \sharp \Psi \wedge y \sharp \Psi\}$
from ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $(\Psi, (\!|\nu x|\!)((\!|\nu y|\!)P), (\!|\nu y|\!)((\!|\nu x|\!)P)) \in ?X$ **by** *auto*
hence $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \sim (\!|\nu y|\!)((\!|\nu x|\!)P)$
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* Ψ *xyP yxP*)
from ‹$(\Psi, xyP, yxP) \in ?X$› **obtain** $x\ y\ P$ **where** $x \sharp \Psi$ **and** $y \sharp \Psi$ **and** $xyP = (\!|\nu x|\!)((\!|\nu y|\!)P)$ **and** $yxP = (\!|\nu y|\!)((\!|\nu x|\!)P)$ **by** *auto*
moreover obtain $A_P\ \Psi_P$ **where** *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **and** $x \sharp A_P$ **and** $y \sharp A_P$
by(*rule-tac* C=(x, y, Ψ) **in** *freshFrame*) *auto*
ultimately show *?case* **by**(*force intro*: *frameResComm FrameStatEqTrans*)
next
case(*cSim* Ψ *xyP yxP*)
from ‹$(\Psi, xyP, yxP) \in ?X$› **obtain** $x\ y\ P$ **where** $x \sharp \Psi$ **and** $y \sharp \Psi$ **and** $xyP = (\!|\nu x|\!)((\!|\nu y|\!)P)$ **and** $yxP = (\!|\nu y|\!)((\!|\nu x|\!)P)$ **by** *auto*
note ‹$x \sharp \Psi$› ‹$y \sharp \Psi$›
moreover have *eqvt ?X* **by**(*force simp add*: *eqvt-def pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence *eqvt*(*?X* $\cup$ *bisim*) **by** *auto*
moreover have $\bigwedge\Psi\ P.\ (\Psi, P, P) \in ?X \cup bisim$ **by**(*blast intro*: *bisimReflexive*)
moreover have $\bigwedge\Psi\ x\ y\ P.\ [\![x \sharp \Psi; y \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu x|\!)((\!|\nu y|\!)P), (\!|\nu y|\!)((\!|\nu x|\!)P)) \in ?X \cup bisim$ **by** *auto*
ultimately have $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \rightsquigarrow[(?X \cup bisim)]\ (\!|\nu y|\!)((\!|\nu x|\!)P)$ **by**(*rule resComm*)
with ‹$xyP = (\!|\nu x|\!)((\!|\nu y|\!)P)$› ‹$yxP = (\!|\nu y|\!)((\!|\nu x|\!)P)$› **show** *?case*
by *simp*
next
case(*cExt* Ψ *xyP yxP* Ψ')
from ‹$(\Psi, xyP, yxP) \in ?X$› **obtain** $x\ y\ P$ **where** $x \sharp \Psi$ **and** $y \sharp \Psi$ **and** *xyPeq*: $xyP = (\!|\nu x|\!)((\!|\nu y|\!)P)$ **and** *yxPeq*: $yxP = (\!|\nu y|\!)((\!|\nu x|\!)P)$ **by** *auto*
show *?case*
proof(*case-tac* x=y)
assume $x = y$
with *xyPeq yxPeq* **show** *?case*
by(*blast intro*: *bisimReflexive*)
next
assume $x \neq y$
obtain x' **where** $x' \sharp \Psi$ **and** $x' \sharp \Psi'$ **and** $x' \neq x$ **and** $x' \neq y$ **and** $x' \sharp P$ **by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
obtain y' **where** $y' \sharp \Psi$ **and** $y' \sharp \Psi'$ **and** $y' \neq x$ **and** $x' \neq y'$ **and** $y' \neq y$ **and** $y' \sharp P$ **by**(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
with *xyPeq* ‹$y' \sharp P$› ‹$x' \sharp P$› ‹$x \neq y$› ‹$x' \neq y$› ‹$y' \neq x$› **have** $(\!|\nu x|\!)((\!|\nu y|\!)P) = (\!|\nu x'|\!)((\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot P))$
apply(*subst alphaRes*[*of* x']) **apply**(*simp add*: *abs-fresh*) **by**(*subst alphaRes*[*of* y' - y]) (*auto simp add*: *eqvts calc-atm*)

moreover with *yxPeq* ‹$y' \sharp P$› ‹$x' \sharp P$› ‹$x \neq y$› ‹$x' \neq y$› ‹$y' \neq x$› ‹$x' \neq y'$› **have** $(\!|\nu y|\!)((\!|\nu x|\!)P) = (\!|\nu y'|\!)((\!|\nu x'|\!)([(y, y')] \cdot [(x, x')] \cdot P))$
apply(*subst alphaRes*[*of* y']) **apply**(*simp add*: *abs-fresh*) **by**(*subst alphaRes*[*of* x' - x]) (*auto simp add*: *eqvts calc-atm*)
with ‹$x \neq y$› ‹$x' \neq y$› ‹$y' \neq y$› ‹$x' \neq x$› ‹$y' \neq x$› ‹$x' \neq y'$› **have** $(\!|\nu y|\!)((\!|\nu x|\!)P) = (\!|\nu y'|\!)((\!|\nu x'|\!)([(x, x')] \cdot [(y, y')] \cdot P))$
by(*subst perm-compose*) (*simp add*: *calc-atm*)
moreover from ‹$x' \sharp \Psi$› ‹$x' \sharp \Psi'$› ‹$y' \sharp \Psi$› ‹$y' \sharp \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu x'|\!)((\!|\nu y'|\!)([(x, x')] \cdot [(y, y')] \cdot P)), (\!|\nu y'|\!)((\!|\nu x'|\!)([(x, x')] \cdot [(y, y')] \cdot P))) \in ?X$
by *auto*
ultimately show *?case* **using** *xyPeq yxPeq* **by** *simp*
qed
next
case(*cSym* Ψ *xyP yxP*)
thus *?case* **by** *auto*
qed
}
moreover obtain x'::*name* **where** $x' \sharp \Psi$ **and** $x' \sharp P$ **and** $x' \neq x$ **and** $x' \neq y$
by(*generate-fresh name*) *auto*
moreover obtain y'::*name* **where** $y' \sharp \Psi$ **and** $y' \sharp P$ **and** $y' \neq x$ **and** $y' \neq y$ **and** $y' \neq x'$
by(*generate-fresh name*) *auto*
ultimately have $\Psi \rhd (\!|\nu x'|\!)((\!|\nu y'|\!)([(y, y'), (x, x')] \cdot P)) \sim (\!|\nu y'|\!)((\!|\nu x'|\!)([(y, y'), (x, x')] \cdot P))$ **by** *auto*
thus *?thesis* **using** ‹$x' \sharp P$› ‹$x' \neq x$› ‹$x' \neq y$› ‹$y' \sharp P$› ‹$y' \neq x$› ‹$y' \neq y$› ‹$y' \neq x'$› ‹$x \neq y$›
apply(*subst alphaRes*[**where** x=x **and** y=x' **and** P=P], *auto*)
apply(*subst alphaRes*[**where** x=y **and** y=y' **and** P=P], *auto*)
apply(*subst alphaRes*[**where** x=x **and** y=x' **and** P=$(\!|\nu y'|\!)([(y, y')] \cdot P)$], *auto simp add*: *abs-fresh fresh-left*)
apply(*subst alphaRes*[**where** x=y **and** y=y' **and** P=$(\!|\nu x'|\!)([(x, x')] \cdot P)$], *auto simp add*: *abs-fresh fresh-left*)
by(*subst perm-compose*) (*simp add*: *eqvts calc-atm*)
qed

lemma *bisimResComm'*:
fixes x :: *name*
and Ψ :: *'b*
and *xvec* :: *name list*
and P :: (*'a*, *'b*, *'c*) *psi*

assumes $x \sharp \Psi$
and *xvec* $\sharp*$ Ψ

shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec|\!)P) \sim (\!|\nu{*}xvec|\!)((\!|\nu x|\!)P)$
using *assms*
by(*induct xvec*) (*auto intro*: *bisimResComm bisimReflexive bisimResPres bisimTransitive*)

lemma *bisimScopeExt*:
fixes x :: *name*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp P$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \sim P \parallel (\!|\nu x|\!)Q$
proof −
{
fix x::*name* **and** Q :: $('a,\ 'b,\ 'c)$ *psi*
assume $x \sharp \Psi$ **and** $x \sharp P$
let *?X1* = $\{((\Psi::'b),\ (\!|\nu{*}xvec|\!)((\!|\nu x|\!)((P::('a,\ 'b,\ 'c)\ psi) \parallel Q)),\ (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)) \mid \Psi\ xvec\ x\ P\ Q.\ x \sharp \Psi \wedge x \sharp P \wedge xvec \sharp{*}\ \Psi\}$
let *?X2* = $\{((\Psi::'b),\ (\!|\nu{*}xvec|\!)((P::('a,\ 'b,\ 'c)\ psi) \parallel (\!|\nu x|\!)Q),\ (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))) \mid \Psi\ xvec\ x\ P\ Q.\ x \sharp \Psi \wedge x \sharp P \wedge xvec \sharp{*}\ \Psi\}$
let *?X* = *?X1* ∪ *?X2*

from ‹$x \sharp \Psi$› ‹$x \sharp P$› **have** $(\Psi,\ (\!|\nu x|\!)(P \parallel Q),\ P \parallel (\!|\nu x|\!)Q) \in$ *?X*
by(*auto*, *rule-tac x*=[] **in** *exI*) (*auto simp add*: *fresh-list-nil*)
moreover have *eqvt ?X*
by(*rule eqvtUnion*)
(*fastforce simp add*: *eqvt-def eqvts pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])+
ultimately have $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \sim P \parallel (\!|\nu x|\!)Q$
proof(*coinduct rule*: *transitiveCoinduct*)
case(*cStatEq* Ψ R T)
show *?case*
proof(*case-tac* $(\Psi,\ R,\ T) \in$ *?X1*)
assume $(\Psi,\ R,\ T) \in$ *?X1*
then obtain *xvec x P Q* **where** $R = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** $T = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
moreover obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp{*}\ \Psi$ **and** $x \sharp A_P$ **and** $A_P \sharp{*}\ Q$
by(*rule-tac* C=$(\Psi,\ x,\ Q)$ **in** *freshFrame*) *auto*
moreover obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$ **and** $A_Q \sharp{*}\ \Psi$ **and** $x \sharp A_Q$ **and** $A_Q \sharp{*}\ A_P$ **and** $A_Q \sharp{*}\ \Psi_P$
by(*rule-tac* C=$(\Psi,\ x,\ A_P,\ \Psi_P)$ **in** *freshFrame*) *auto*
moreover from *FrQ* ‹$A_P \sharp{*}\ Q$› ‹$A_Q \sharp{*}\ A_P$› **have** $A_P \sharp{*}\ \Psi_Q$
by(*drule-tac extractFrameFreshChain*) *auto*
moreover from ‹$x \sharp P$› ‹$x \sharp A_P$› *FrP* **have** $x \sharp \Psi_P$ **by**(*drule-tac extractFrameFresh*) *auto*
ultimately show *?case*
by(*force simp add*: *frameChainAppend intro*: *frameResComm′ FrameStatEqTrans frameResChainPres*)
next
assume $(\Psi,\ R,\ T) \notin$ *?X1*

with ‹$(\Psi,\ R,\ T) \in ?X$› **have** $(\Psi,\ R,\ T) \in ?X2$ **by** *blast*
then obtain *xvec x P Q* **where** $T = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** $R = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
moreover obtain $A_P\ \Psi_P$ **where** *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp{*}\ \Psi$ **and** $x \sharp A_P$ **and** $A_P \sharp{*}\ Q$
by(*rule-tac* $C{=}(\Psi,\ x,\ Q)$ **in** *freshFrame*) *auto*
moreover obtain $A_Q\ \Psi_Q$ **where** *FrQ*: *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$ **and** $A_Q \sharp{*}\ \Psi$ **and** $x \sharp A_Q$ **and** $A_Q \sharp{*}\ A_P$ **and** $A_Q \sharp{*}\ \Psi_P$
by(*rule-tac* $C{=}(\Psi,\ x,\ A_P,\ \Psi_P)$ **in** *freshFrame*) *auto*
moreover from *FrQ* ‹$A_P \sharp{*}\ Q$› ‹$A_Q \sharp{*}\ A_P$› **have** $A_P \sharp{*}\ \Psi_Q$
by(*drule-tac extractFrameFreshChain*) *auto*
moreover from ‹$x \sharp P$› ‹$x \sharp A_P$› *FrP* **have** $x \sharp \Psi_P$ **by**(*drule-tac extractFrameFresh*) *auto*
ultimately show *?case*
apply *auto*
by(*force simp add*: *frameChainAppend intro*: *frameResComm′ FrameStatEqTrans frameResChainPres FrameStatEqSym*)
qed
next
case(*cSim* $\Psi\ R\ T$)
let $?Y = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge ((\Psi,\ P',\ Q') \in ?X \vee \Psi \rhd P' \sim Q') \wedge \Psi \rhd Q' \sim Q\}$
from ‹*eqvt ?X*› **have** *eqvt ?Y* **by** *blast*
have *C1*: $\bigwedge\Psi\ R\ T\ y.\ [\![(\Psi,\ R,\ T) \in ?Y;\ (y{::}name) \sharp \Psi]\!] \Longrightarrow (\Psi,\ (\!|\nu y|\!)R,\ (\!|\nu y|\!)T) \in ?Y$
proof −
fix $\Psi\ R\ T\ y$
assume $(\Psi,\ R,\ T) \in ?Y$
then obtain $R'\ T'$ **where** $\Psi \rhd R \sim R'$ **and** $(\Psi,\ R',\ T') \in (?X \cup bisim)$ **and** $\Psi \rhd T' \sim T$ **by** *fastforce*
assume $(y{::}name) \sharp \Psi$
show $(\Psi,\ (\!|\nu y|\!)R,\ (\!|\nu y|\!)T) \in ?Y$
proof(*case-tac* $(\Psi,\ R',\ T') \in ?X$)
assume $(\Psi,\ R',\ T') \in ?X$
show *?thesis*
proof(*case-tac* $(\Psi,\ R',\ T') \in ?X1$)
assume $(\Psi,\ R',\ T') \in ?X1$
then obtain *xvec x P Q* **where** $R'eq$: $R' = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** $T'eq$: $T' = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$
and $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
from ‹$\Psi \rhd R \sim R'$› ‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)R \sim (\!|\nu y|\!)R'$ **by**(*rule bisimResPres*)
moreover from ‹$xvec \sharp{*}\ \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp P$› ‹$x \sharp \Psi$› **have** $(\Psi,\ (\!|\nu{*}(y\#xvec)|\!)(\!|\nu x|\!)(P \parallel Q),\ (\!|\nu{*}(y\#xvec)|\!)(P \parallel (\!|\nu x|\!)Q)) \in ?X1$
by(*force simp del*: *resChain.simps*)
with $R'eq\ T'eq$ **have** $(\Psi,\ (\!|\nu y|\!)R',\ (\!|\nu y|\!)T') \in ?X \cup bisim$ **by** *simp*
moreover from ‹$\Psi \rhd T' \sim T$› ‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)T' \sim (\!|\nu y|\!)T$

by(*rule bisimResPres*)
ultimately show *?thesis* **by** *blast*
next
assume $(\Psi,\ R',\ T') \notin$ *?X1*
with ‹$(\Psi,\ R',\ T') \in$ *?X*› **have** $(\Psi,\ R',\ T') \in$ *?X2* **by** *blast*
then obtain *xvec x P Q* **where** $T'eq$: $T' = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** $R'eq$: $R' = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
from ‹$\Psi \rhd R \sim R'$› ‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)R \sim (\!|\nu y|\!)R'$ **by**(*rule bisimResPres*)
moreover from ‹$xvec \sharp{*}\ \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp P$› ‹$x \sharp \Psi$› **have** $(\Psi,\ (\!|\nu{*}(y\#xvec)|\!)(P \parallel (\!|\nu x|\!)Q),\ (\!|\nu{*}(y\#xvec)|\!)((\!|\nu x|\!)(P \parallel Q))) \in$ *?X2*
by(*force simp del*: *resChain.simps*)
with $R'eq$ $T'eq$ **have** $(\Psi,\ (\!|\nu y|\!)R',\ (\!|\nu y|\!)T') \in$ *?X* $\cup$ *bisim* **by** *simp*
moreover from ‹$\Psi \rhd T' \sim T$› ‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)T' \sim (\!|\nu y|\!)T$ **by**(*rule bisimResPres*)
ultimately show *?thesis* **by** *blast*
qed
next
assume $(\Psi,\ R',\ T') \notin$ *?X*
with ‹$(\Psi,\ R',\ T') \in$ *?X* $\cup$ *bisim*› **have** $\Psi \rhd R' \sim T'$ **by** *blast*
with ‹$\Psi \rhd R \sim R'$› ‹$\Psi \rhd T' \sim T$› ‹$y \sharp \Psi$› **show** *?thesis*
by(*blast dest*: *bisimResPres*)
qed
qed

show *?case*
proof(*case-tac* $(\Psi,\ R,\ T) \in$ *?X1*)
assume $(\Psi,\ R,\ T) \in$ *?X1*
then obtain *xvec x P Q* **where** *Req*: $R = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** *Teq*: $T = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
have $\Psi \rhd (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q)) \rightsquigarrow[?Y]\ (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$
proof −
have $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \rightsquigarrow[?Y]\ P \parallel (\!|\nu x|\!)Q$
proof −
note ‹$x \sharp P$› ‹$x \sharp \Psi$› ‹*eqvt ?Y*›
moreover have $\bigwedge\Psi\ P.\ (\Psi,\ P,\ P) \in$ *?Y* **by**(*blast intro*: *bisimReflexive*)
moreover have $\bigwedge x\ \Psi\ P\ Q\ xvec.\ [\![x \sharp \Psi;\ x \sharp P;\ xvec \sharp{*}\ \Psi]\!] \Longrightarrow (\Psi,\ (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P \parallel Q)),\ (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)) \in$ *?Y*
proof −
fix $x\ \Psi\ P\ Q\ xvec$
assume $(x::name) \sharp (\Psi::'b)$ **and** $x \sharp (P::('a,\ 'b,\ 'c)\ psi)$ **and** $(xvec::name\ list) \sharp{*}\ \Psi$
from ‹$x \sharp \Psi$› ‹$xvec \sharp{*}\ \Psi$› **have** $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P \parallel Q)) \sim (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$
by(*rule bisimResComm'*)
moreover from ‹$xvec \sharp{*}\ \Psi$› ‹$x \sharp \Psi$› ‹$x \sharp P$› **have** $(\Psi,\ (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q)),\ (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)) \in$ *?X* $\cup$ *bisim*

by *blast*
ultimately show $(\Psi, (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P \parallel Q)), (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)) \in ?Y$
by(*blast intro*: *bisimReflexive*)
qed
moreover have $\bigwedge\Psi\ xvec\ P\ x.\ [\![x \sharp \Psi;\ xvec \sharp{*} \Psi]\!] \Longrightarrow (\Psi, (\!|\nu x|\!)((\!|\nu{*}xvec|\!)P), (\!|\nu{*}xvec|\!)((\!|\nu x|\!)P)) \in ?Y$
by(*blast intro*: *bisimResComm′ bisimReflexive*)
ultimately show *?thesis* **by**(*rule scopeExtLeft*)
qed
thus *?thesis* **using** ‹*eqvt ?Y*› ‹$xvec \sharp{*} \Psi$› *C1*
by(*rule resChainPres*)
qed
with *Req Teq* **show** *?case* **by** *simp*
next
assume $(\Psi, R, T) \notin ?X1$
with ‹$(\Psi, R, T) \in ?X$› **have** $(\Psi, R, T) \in ?X2$ **by** *blast*
then obtain *xvec x P Q* **where** *Teq*: $T = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** *Req*: $R = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*} \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
have $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q) \rightsquigarrow[?Y]\ (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$
proof −
have $\Psi \rhd P \parallel (\!|\nu x|\!)Q \rightsquigarrow[?Y]\ (\!|\nu x|\!)(P \parallel Q)$
proof −
note ‹$x \sharp P$› ‹$x \sharp \Psi$› ‹*eqvt ?Y*›
moreover have $\bigwedge\Psi\ P.\ (\Psi, P, P) \in ?Y$ **by**(*blast intro*: *bisimReflexive*)
moreover have $\bigwedge x\ \Psi\ P\ Q\ xvec.\ [\![x \sharp \Psi;\ x \sharp P;\ xvec \sharp{*} \Psi]\!] \Longrightarrow (\Psi, (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q), (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P \parallel Q))) \in ?Y$
proof −
fix *x Ψ P Q xvec*
assume $(x::name) \sharp (\Psi::'b)$ **and** $x \sharp (P::('a, 'b, 'c)\ psi)$ **and** $(xvec::name\ list) \sharp{*} \Psi$
from ‹$xvec \sharp{*} \Psi$› ‹$x \sharp \Psi$› ‹$x \sharp P$› **have** $(\Psi, (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q), (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))) \in ?X \cup bisim$
by *blast*
moreover from ‹$x \sharp \Psi$› ‹$xvec \sharp{*} \Psi$› **have** $\Psi \rhd (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q)) \sim (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P \parallel Q))$
by(*blast intro*: *bisimResComm′ bisimE*)
ultimately show $(\Psi, (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q), (\!|\nu x|\!)((\!|\nu{*}xvec|\!)(P \parallel Q))) \in ?Y$
by(*blast intro*: *bisimReflexive*)
qed
ultimately show *?thesis* **by**(*rule scopeExtRight*)
qed
thus *?thesis* **using** ‹*eqvt ?Y*› ‹$xvec \sharp{*} \Psi$› *C1*
by(*rule resChainPres*)
qed
with *Req Teq* **show** *?case* **by** *simp*
qed

next
case(*cExt* Ψ R T Ψ')
show *?case*
proof(*case-tac* $(\Psi, R, T) \in$ *?X1*)
assume $(\Psi, R, T) \in$ *?X1*
then obtain *xvec* x P Q **where** *Req*: $R = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** *Teq*: $T = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$
by *auto*
obtain y::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp xvec$ **and** $y \sharp \Psi$ **and** $y \sharp \Psi'$
by(*generate-fresh name*, *auto simp add*: *fresh-prod*)

obtain p **where** $(p \cdot xvec) \sharp{*}\ \Psi$ **and** $(p \cdot xvec) \sharp{*}\ P$ **and** $(p \cdot xvec) \sharp{*}\ Q$ **and** $(p \cdot xvec) \sharp{*}\ \Psi'$
and $x \sharp (p \cdot xvec)$ **and** $y \sharp (p \cdot xvec)$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm* p
by(*rule-tac* c=$(\Psi, P, Q, x, y, \Psi')$ **in** *name-list-avoiding*) *auto*

from ‹$y \sharp P$› **have** $(p \cdot y) \sharp (p \cdot P)$ **by**(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with S ‹$y \sharp xvec$› ‹$y \sharp (p \cdot xvec)$› **have** $y \sharp (p \cdot P)$ **by** *simp*
with ‹$(p \cdot xvec) \sharp{*}\ \Psi$› ‹$y \sharp \Psi$› ‹$(p \cdot xvec) \sharp{*}\ \Psi'$› ‹$y \sharp \Psi'$›
have $(\Psi \otimes \Psi', (\!|\nu{*}(p \cdot xvec)|\!)((\!|\nu y|\!)((p \cdot P) \parallel (p \cdot [(x, y)] \cdot Q))), (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel ((\!|\nu y|\!)(p \cdot [(x, y)] \cdot Q)))) \in$ *?X*
by *auto*
moreover from *Req* ‹$(p \cdot xvec) \sharp{*}\ P$› ‹$(p \cdot xvec) \sharp{*}\ Q$› ‹$y \sharp xvec$› ‹$y \sharp (p \cdot xvec)$› ‹$x \sharp (p \cdot xvec)$› ‹$y \sharp P$› ‹$y \sharp Q$› ‹$x \sharp P$› S
have $R = (\!|\nu{*}(p \cdot xvec)|\!)((\!|\nu y|\!)((p \cdot P) \parallel (p \cdot [(x, y)] \cdot Q)))$
apply(*erule-tac rev-mp*)
apply(*subst alphaRes*[*of y*])
apply(*clarsimp simp add*: *eqvts*)
apply(*subst resChainAlpha*[*of p*])
by(*auto simp add*: *eqvts*)
moreover from *Teq* ‹$(p \cdot xvec) \sharp{*}\ P$› ‹$(p \cdot xvec) \sharp{*}\ Q$› ‹$y \sharp xvec$› ‹$y \sharp (p \cdot xvec)$› ‹$x \sharp (p \cdot xvec)$› ‹$y \sharp P$› ‹$y \sharp Q$› ‹$x \sharp P$› S
have $T = (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel (\!|\nu y|\!)(p \cdot [(x, y)] \cdot Q))$
apply(*erule-tac rev-mp*)
apply(*subst alphaRes*[*of y*])
apply(*clarsimp simp add*: *eqvts*)
apply(*subst resChainAlpha*[*of p*])
by(*auto simp add*: *eqvts*)
ultimately show *?case*
by *blast*
next
assume $(\Psi, R, T) \notin$ *?X1*
with ‹$(\Psi, R, T) \in$ *?X*› **have** $(\Psi, R, T) \in$ *?X2* **by** *blast*
then obtain *xvec* x P Q **where** *Teq*: $T = (\!|\nu{*}xvec|\!)((\!|\nu x|\!)(P \parallel Q))$ **and** *Req*: $R = (\!|\nu{*}xvec|\!)(P \parallel (\!|\nu x|\!)Q)$ **and** $xvec \sharp{*}\ \Psi$ **and** $x \sharp P$ **and** $x \sharp \Psi$

by *auto*
obtain y::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp xvec$ **and** $y \sharp \Psi$ **and** $y \sharp \Psi'$
by(*generate-fresh name*, *auto simp add*: *fresh-prod*)

obtain p **where** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi'$
and $x \sharp (p \cdot xvec)$ **and** $y \sharp (p \cdot xvec)$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm* p
by(*rule-tac* c=(Ψ, P, Q, x, y, Ψ') **in** *name-list-avoiding*) *auto*

from ‹$y \sharp P$› **have** $(p \cdot y) \sharp (p \cdot P)$ **by**(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with S ‹$y \sharp xvec$› ‹$y \sharp (p \cdot xvec)$› **have** $y \sharp (p \cdot P)$ **by** *simp*
with ‹$(p \cdot xvec) \sharp* \Psi$› ‹$y \sharp \Psi$› ‹$(p \cdot xvec) \sharp* \Psi'$› ‹$y \sharp \Psi'$›
have $(\Psi \otimes \Psi', (\!|\nu*(p \cdot xvec)|\!)((p \cdot P) \parallel (\!|\nu y|\!)(p \cdot [(x, y)] \cdot Q)), (\!|\nu*(p \cdot xvec)|\!)((\!|\nu y|\!)((p \cdot P) \parallel (p \cdot [(x, y)] \cdot Q)))) \in ?X2$
by *auto*
moreover from *Teq* ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› ‹$y \sharp xvec$› ‹$y \sharp (p \cdot xvec)$› ‹$x \sharp (p \cdot xvec)$› ‹$y \sharp P$› ‹$y \sharp Q$› ‹$x \sharp P$› S
have $T = (\!|\nu*(p \cdot xvec)|\!)((\!|\nu y|\!)((p \cdot P) \parallel (p \cdot [(x, y)] \cdot Q)))$
apply(*erule-tac rev-mp*)
apply(*subst alphaRes*[*of y*])
apply(*clarsimp simp add*: *eqvts*)
apply(*subst resChainAlpha*[*of p*])
by(*auto simp add*: *eqvts*)
moreover from *Req* ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› ‹$y \sharp xvec$› ‹$y \sharp (p \cdot xvec)$› ‹$x \sharp (p \cdot xvec)$› ‹$y \sharp P$› ‹$y \sharp Q$› ‹$x \sharp P$› S
have $R = (\!|\nu*(p \cdot xvec)|\!)((p \cdot P) \parallel (\!|\nu y|\!)(p \cdot [(x, y)] \cdot Q))$
apply(*erule-tac rev-mp*)
apply(*subst alphaRes*[*of y*])
apply(*clarsimp simp add*: *eqvts*)
apply(*subst resChainAlpha*[*of p*])
by(*auto simp add*: *eqvts*)
ultimately show *?case*
by *blast*
qed
next
case(*cSym* Ψ P Q)
thus *?case*
by(*blast dest*: *bisimE*)
qed
}
moreover obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp P$ $y \sharp Q$
by(*generate-fresh name*) *auto*
ultimately have $\Psi \rhd (\!|\nu y|\!)(P \parallel ([(x, y)] \cdot Q)) \sim P \parallel (\!|\nu y|\!)([(x, y)] \cdot Q)$ **by** *auto*
thus *?thesis* **using** *assms* ‹$y \sharp P$› ‹$y \sharp Q$›
apply(*subst alphaRes*[**where** x=x **and** y=y **and** P=Q], *auto*)
by(*subst alphaRes*[**where** x=x **and** y=y **and** P=$P \parallel Q$]) *auto*

qed

lemma *bisimScopeExtChain*:
fixes *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes *xvec* $\sharp*$ Ψ
and *xvec* $\sharp*$ P

shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \sim P \parallel ((\!|\nu{*}xvec|\!)\, Q)$
using *assms*
by(*induct xvec*) (*auto intro*: *bisimScopeExt bisimReflexive bisimTransitive bisimResPres*)

lemma *bisimParAssoc*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd (P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$
proof −
let $?X = \{(\Psi,\ (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R),\ (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R)))\ |\ \Psi\ xvec\ P\ Q\ R.\ xvec \sharp* \Psi\}$
let $?Y = \{(\Psi,\ P,\ Q)\ |\ \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in ?X \wedge \Psi \rhd Q' \sim Q\}$

have $(\Psi,\ (P \parallel Q) \parallel R,\ P \parallel (Q \parallel R)) \in ?X$
by(*auto*, *rule-tac x*=[] **in** *exI*) *auto*
moreover have *eqvt ?X* **by**(*force simp add*: *eqvt-def simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*] *eqvts*)
ultimately show *?thesis*
proof(*coinduct rule*: *weakTransitiveCoinduct'*)
case(*cStatEq* Ψ *PQR PQR'*)
from ‹$(\Psi,\ PQR,\ PQR') \in ?X$› **obtain** *xvec P Q R* **where** *xvec* $\sharp*$ Ψ **and** $PQR = (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R)$ **and** $PQR' = (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R))$
by *auto*
moreover obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* Q$ **and** $A_P \sharp* R$
by(*rule-tac* C=$(\Psi,\ Q,\ R)$ **in** *freshFrame*) *auto*
moreover obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* A_P$ **and** $A_Q \sharp* \Psi_P$ **and** $A_Q \sharp* R$
by(*rule-tac* C=$(\Psi,\ A_P,\ \Psi_P,\ R)$ **in** *freshFrame*) *auto*
moreover obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* A_P$ **and** $A_R \sharp* \Psi_P$ **and** $A_R \sharp* A_Q$ **and** $A_R \sharp* \Psi_Q$
by(*rule-tac* C=$(\Psi,\ A_P,\ \Psi_P,\ A_Q,\ \Psi_Q)$ **in** *freshFrame*) *auto*
moreover from *FrQ* ‹$A_P \sharp* Q$› ‹$A_Q \sharp* A_P$› **have** $A_P \sharp* \Psi_Q$

by(*drule-tac extractFrameFreshChain*) *auto*
moreover from *FrR* ‹A_P ♯* R› ‹A_R ♯* A_P› **have** A_P ♯* Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*
moreover from *FrR* ‹A_Q ♯* R› ‹A_R ♯* A_Q› **have** A_Q ♯* Ψ_R
by(*drule-tac extractFrameFreshChain*) *auto*
ultimately show *?case* **using** *freshCompChain*
by *auto* (*metis frameChainAppend compositionSym Associativity frameNilStatEq frameResChainPres*)
next
case(*cSim* Ψ T S)
from ‹$(\Psi, T, S) \in ?X$› **obtain** *xvec* P Q R **where** *xvec* ♯* Ψ **and** *TEq*: $T = (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R)$
and *SEq*: $S = (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R))$
by *auto*
from ‹*eqvt ?X*›**have** *eqvt ?Y* **by** *blast*
have *C1*: $\bigwedge\Psi\ T\ S\ yvec.\ [\![(\Psi, T, S) \in ?Y;\ yvec \sharp* \Psi]\!] \Longrightarrow (\Psi, (\!|\nu{*}yvec|\!)T, (\!|\nu{*}yvec|\!)S) \in ?Y$
proof −
fix Ψ T S *yvec*
assume $(\Psi, T, S) \in ?Y$
then obtain T' S' **where** $\Psi \rhd T \sim T'$ **and** $(\Psi, T', S') \in ?X$ **and** $\Psi \rhd S' \sim S$ **by** *fastforce*
assume (*yvec*::*name list*) ♯* Ψ
from ‹$(\Psi, T', S') \in ?X$› **obtain** *xvec* P Q R **where** $T'eq$: $T' = (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R)$ **and** $S'eq$: $S' = (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R))$
and *xvec* ♯* Ψ
by *auto*
from ‹$\Psi \rhd T \sim T'$› ‹*yvec* ♯* Ψ› **have** $\Psi \rhd (\!|\nu{*}yvec|\!)T \sim (\!|\nu{*}yvec|\!)T'$ **by**(*rule bisimResChainPres*)
moreover from ‹*xvec* ♯* Ψ› ‹*yvec* ♯* Ψ› **have** $(\Psi, (\!|\nu{*}(yvec@xvec)|\!)((P \parallel Q) \parallel R), (\!|\nu{*}(yvec@xvec)|\!)(P \parallel (Q \parallel R))) \in ?X$
by *force*
with $T'eq$ $S'eq$ **have** $(\Psi, (\!|\nu{*}yvec|\!)T', (\!|\nu{*}yvec|\!)S') \in ?X$ **by**(*simp add*: *resChainAppend*)
moreover from ‹$\Psi \rhd S' \sim S$› ‹*yvec* ♯* Ψ› **have** $\Psi \rhd (\!|\nu{*}yvec|\!)S' \sim (\!|\nu{*}yvec|\!)S$ **by**(*rule bisimResChainPres*)
ultimately show $(\Psi, (\!|\nu{*}yvec|\!)T, (\!|\nu{*}yvec|\!)S) \in ?Y$ **by** *blast*
qed
have *C2*: $\bigwedge\Psi\ T\ S\ y.\ [\![(\Psi, T, S) \in ?Y;\ y \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu y|\!)T, (\!|\nu y|\!)S) \in ?Y$
by(*drule-tac yvec2*=[*y*] **in** *C1*) *auto*

have $\Psi \rhd (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R) \rightsquigarrow[?Y]\ (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R))$
proof −
have $\Psi \rhd (P \parallel Q) \parallel R \rightsquigarrow[?Y]\ P \parallel (Q \parallel R)$
proof −
note ‹*eqvt ?Y*›
moreover have $\bigwedge\Psi\ P\ Q\ R.\ (\Psi, (P \parallel Q) \parallel R, P \parallel (Q \parallel R)) \in ?Y$
proof −
fix Ψ P Q R

have $(\Psi::'b, ((P::('a, 'b, 'c)\ psi) \parallel Q) \parallel R, P \parallel (Q \parallel R)) \in ?X$
by(*auto*, *rule-tac x*=[] **in** *exI*) *auto*
thus $(\Psi, (P \parallel Q) \parallel R, P \parallel (Q \parallel R)) \in ?Y$
by(*blast intro*: *bisimReflexive*)
qed
moreover have $\bigwedge xvec\ \Psi\ P\ Q\ R.\ [\![xvec \sharp* \Psi; xvec \sharp* P]\!] \Longrightarrow (\Psi, (\!|\nu*xvec|\!)((P \parallel Q) \parallel R), P \parallel ((\!|\nu*xvec|\!)(Q \parallel R))) \in ?Y$
proof −
fix *xvec* Ψ P Q R
assume $(xvec::name\ list) \sharp* (\Psi::'b)$ **and** $xvec \sharp* (P::('a, 'b, 'c)\ psi)$
from ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu*xvec|\!)((P \parallel Q) \parallel R), (\!|\nu*xvec|\!)(P \parallel (Q \parallel R))) \in ?X$ **by** *blast*
moreover from ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P \parallel (Q \parallel R)) \sim P \parallel ((\!|\nu*xvec|\!)(Q \parallel R))$
by(*rule bisimScopeExtChain*)
ultimately show $(\Psi, (\!|\nu*xvec|\!)((P \parallel Q) \parallel R), P \parallel ((\!|\nu*xvec|\!)(Q \parallel R))) \in ?Y$
by(*blast intro*: *bisimReflexive*)
qed
moreover have $\bigwedge xvec\ \Psi\ P\ Q\ R.\ [\![xvec \sharp* \Psi; xvec \sharp* R]\!] \Longrightarrow (\Psi, ((\!|\nu*xvec|\!)(P \parallel Q)) \parallel R, (\!|\nu*xvec|\!)(P \parallel (Q \parallel R))) \in ?Y$
proof −
fix *xvec* Ψ P Q R
assume $(xvec::name\ list) \sharp* (\Psi::'b)$ **and** $xvec \sharp* (R::('a, 'b, 'c)\ psi)$
have $\Psi \rhd ((\!|\nu*xvec|\!)(P \parallel Q)) \parallel R \sim R \parallel ((\!|\nu*xvec|\!)(P \parallel Q))$ **by**(*rule bisimParComm*)
moreover from ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* R$› **have** $\Psi \rhd (\!|\nu*xvec|\!)(R \parallel (P \parallel Q)) \sim R \parallel ((\!|\nu*xvec|\!)(P \parallel Q))$ **by**(*rule bisimScopeExtChain*)
hence $\Psi \rhd R \parallel ((\!|\nu*xvec|\!)(P \parallel Q)) \sim (\!|\nu*xvec|\!)(R \parallel (P \parallel Q))$ **by**(*rule bisimE*)
moreover from ‹$xvec \sharp* \Psi$› **have** $\Psi \rhd (\!|\nu*xvec|\!)(R \parallel (P \parallel Q)) \sim (\!|\nu*xvec|\!)((P \parallel Q) \parallel R)$
by(*metis bisimResChainPres bisimParComm*)
moreover from ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu*xvec|\!)((P \parallel Q) \parallel R), (\!|\nu*xvec|\!)(P \parallel (Q \parallel R))) \in ?X$ **by** *blast*
ultimately show $(\Psi, ((\!|\nu*xvec|\!)(P \parallel Q)) \parallel R, (\!|\nu*xvec|\!)(P \parallel (Q \parallel R))) \in ?Y$ **by**(*blast dest*: *bisimTransitive intro*: *bisimReflexive*)
qed
ultimately show *?thesis* **using** *C1*
by(*rule parAssocLeft*)
qed
thus *?thesis* **using** ‹*eqvt ?Y*› ‹$xvec \sharp* \Psi$› *C2*
by(*rule resChainPres*)
qed
with *TEq SEq* **show** *?case* **by** *simp*
next
case(*cExt* Ψ T S Ψ')
from ‹$(\Psi, T, S) \in ?X$› **obtain** *xvec* P Q R **where** $xvec \sharp* \Psi$ **and** *TEq*: $T = (\!|\nu*xvec|\!)((P \parallel Q) \parallel R)$

and *SEq*: $S = (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R))$
by *auto*
obtain p **where** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* R$ **and** $(p \cdot xvec) \sharp* \Psi'$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm p*
by(*rule-tac c*=(Ψ, P, Q, R, Ψ') **in** *name-list-avoiding*) *auto*

from ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu{*}(p \cdot xvec)|\!)(((p \cdot P) \parallel (p \cdot Q)) \parallel (p \cdot R)), (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel ((p \cdot Q) \parallel (p \cdot R)))) \in ?X$
by *auto*
moreover from *TEq* ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› ‹$(p \cdot xvec) \sharp* R$› S **have** $T = (\!|\nu{*}(p \cdot xvec)|\!)(((p \cdot P) \parallel (p \cdot Q)) \parallel (p \cdot R))$
apply *auto* **by**(*subst resChainAlpha*[*of p*]) *auto*
moreover from *SEq* ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* Q$› ‹$(p \cdot xvec) \sharp* R$› S **have** $S = (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel ((p \cdot Q) \parallel (p \cdot R)))$
apply *auto* **by**(*subst resChainAlpha*[*of p*]) *auto*
ultimately show *?case* **by** *simp*
next
case(*cSym* Ψ T S)
from ‹$(\Psi, T, S) \in ?X$› **obtain** *xvec P Q R* **where** $xvec \sharp* \Psi$ **and** *TEq*: $T = (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R)$
and *SEq*: $(\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R)) = S$
by *auto*

from ‹$xvec \sharp* \Psi$› **have** $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel (Q \parallel R)) \sim (\!|\nu{*}xvec|\!)((R \parallel Q) \parallel P)$
by(*metis bisimParComm bisimParPres bisimTransitive bisimResChainPres*)
moreover from ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu{*}xvec|\!)((R \parallel Q) \parallel P), (\!|\nu{*}xvec|\!)(R \parallel (Q \parallel P))) \in ?X$ **by** *blast*
moreover from ‹$xvec \sharp* \Psi$› **have** $\Psi \rhd (\!|\nu{*}xvec|\!)(R \parallel (Q \parallel P)) \sim (\!|\nu{*}xvec|\!)((P \parallel Q) \parallel R)$
by(*metis bisimParComm bisimParPres bisimTransitive bisimResChainPres*)
ultimately show *?case* **using** *TEq SEq* **by**(*blast dest*: *bisimTransitive*)
qed
qed

lemma *bisimParNil*:
fixes P :: $('a, 'b, 'c)$ *psi*

shows $\Psi \rhd P \parallel \mathbf{0} \sim P$
proof −
let $?X1 = \{(\Psi, P \parallel \mathbf{0}, P) \mid \Psi\ P.\ True\}$
let $?X2 = \{(\Psi, P, P \parallel \mathbf{0}) \mid \Psi\ P.\ True\}$
let $?X = ?X1 \cup ?X2$
have *eqvt ?X* **by**(*auto simp add*: *eqvt-def*)
have $(\Psi, P \parallel \mathbf{0}, P) \in ?X$ **by** *simp*
thus *?thesis*
proof(*coinduct rule*: *bisimWeakCoinduct*)
case(*cStatEq* Ψ Q R)
show *?case*

```
    proof(case-tac (Ψ, Q, R) ∈ ?X1)
      assume (Ψ, Q, R) ∈ ?X1
      then obtain P where Q = P ∥ 0 and R = P by auto
      moreover obtain A_P Ψ_P where extractFrame P = ⟨A_P, Ψ_P⟩ and A_P ♯* Ψ
        by(rule freshFrame)
      ultimately show ?case
        apply auto by(metis frameResChainPres frameNilStatEq Identity Associa-
tivity AssertionStatEqTrans Commutativity)
    next
      assume (Ψ, Q, R) ∉ ?X1
      with ‹(Ψ, Q, R) ∈ ?X› have (Ψ, Q, R) ∈ ?X2 by blast
      then obtain P where Q = P and R = P ∥ 0 by auto
      moreover obtain A_P Ψ_P where extractFrame P = ⟨A_P, Ψ_P⟩ and A_P ♯* Ψ
        by(rule freshFrame)
      ultimately show ?case
        apply auto by(metis frameResChainPres frameNilStatEq Identity Associa-
tivity AssertionStatEqTrans AssertionStatEqSym Commutativity)
    qed
  next
    case(cSim Ψ Q R)
    thus ?case using ‹eqvt ?X›
      by(auto intro: parNilLeft parNilRight)
  next
    case(cExt Ψ Q R Ψ')
    thus ?case by auto
  next
    case(cSym Ψ Q R)
    thus ?case by auto
  qed
qed

lemma bisimResNil:
  fixes x :: name
  and   Ψ :: 'b

  shows Ψ ▷ ⦇νx⦈0 ∼ 0
proof −
  {
    fix x::name
    assume x ♯ Ψ
    have Ψ ▷ ⦇νx⦈0 ∼ 0
    proof −
      let ?X1 = {(Ψ, ⦇νx⦈0, 0) | Ψ x. x ♯ Ψ}
      let ?X2 = {(Ψ, 0, ⦇νx⦈0) | Ψ x. x ♯ Ψ}
      let ?X = ?X1 ∪ ?X2

      from ‹x ♯ Ψ› have (Ψ, ⦇νx⦈0, 0) ∈ ?X by auto
      thus ?thesis
      proof(coinduct rule: bisimWeakCoinduct)
```

```
      case(cStatEq Ψ P Q)
    thus ?case using freshComp by(force intro: frameResFresh FrameStatEqSym)
    next
      case(cSim Ψ P Q)
      thus ?case
        by(force intro: resNilLeft resNilRight)
    next
      case(cExt Ψ P Q Ψ')
      obtain y where y ♯ Ψ and y ♯ Ψ' and y ≠ x
        by(generate-fresh name) (auto simp add: fresh-prod)
      show ?case
      proof(case-tac (Ψ, P, Q) ∈ ?X1)
        assume (Ψ, P, Q) ∈ ?X1
        then obtain x where P = (|νx|)0 and Q = 0 by auto
        moreover have (|νx|)0 = (|νy|) 0 by(subst alphaRes) auto
        ultimately show ?case using ‹y ♯ Ψ› ‹y ♯ Ψ'› by auto
      next
        assume (Ψ, P, Q) ∉ ?X1
        with ‹(Ψ, P, Q) ∈ ?X› have (Ψ, P, Q) ∈ ?X2 by auto
        then obtain x where Q = (|νx|)0 and P = 0 by auto
        moreover have (|νx|)0 = (|νy|) 0 by(subst alphaRes) auto
        ultimately show ?case using ‹y ♯ Ψ› ‹y ♯ Ψ'› by auto
      qed
    next
      case(cSym Ψ P Q)
      thus ?case by auto
    qed
  qed
}
moreover obtain y::name where y ♯ Ψ by(generate-fresh name) auto
ultimately have Ψ ▷ (|νy|)0 ∼ 0 by auto
thus ?thesis by(subst alphaRes[where x=x and y=y]) auto
qed

lemma bisimOutputPushRes:
  fixes x :: name
  and   Ψ :: 'b
  and   M :: 'a
  and   N :: 'a
  and   P :: ('a, 'b, 'c) psi

  assumes x ♯ M
  and     x ♯ N

  shows Ψ ▷ (|νx|)(M⟨N⟩.P) ∼ M⟨N⟩.(|νx|)P
proof −
  {
    fix x::name and P::('a, 'b, 'c) psi
    assume x ♯ Ψ and x ♯ M and x ♯ N
```

let *?X1* = {(Ψ, ⦇νx⦈(M⟨N⟩.P), M⟨N⟩.⦇νx⦈P) | Ψ x M N P. x ♯ Ψ ∧ x ♯ M ∧ x ♯ N}
let *?X2* = {(Ψ, M⟨N⟩.⦇νx⦈P, ⦇νx⦈(M⟨N⟩.P)) | Ψ x M N P. x ♯ Ψ ∧ x ♯ M ∧ x ♯ N}
let *?X* = *?X1* ∪ *?X2*

have *eqvt ?X* **by**(*rule-tac eqvtUnion*) (*force simp add*: *eqvt-def pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*] *eqvts*)+
from ‹x ♯ Ψ› ‹x ♯ M› ‹x ♯ N› **have** (Ψ, ⦇νx⦈(M⟨N⟩.P), M⟨N⟩.⦇νx⦈P) ∈ *?X* **by** *auto*
hence Ψ ⊳ ⦇νx⦈(M⟨N⟩.P) ∼ M⟨N⟩.⦇νx⦈P
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* Ψ Q R)
thus *?case* **using** *freshComp* **by**(*force intro*: *frameResFresh FrameStatEqSym*)
next
case(*cSim* Ψ Q R)
thus *?case* **using** ‹*eqvt ?X*›
by(*fastforce intro*: *outputPushResLeft outputPushResRight bisimReflexive*)
next
case(*cExt* Ψ Q R Ψ′)
show *?case*
proof(*case-tac* (Ψ, Q, R) ∈ *?X1*)
assume (Ψ, Q, R) ∈ *?X1*
then obtain x M N P **where** *Qeq*: Q = ⦇νx⦈(M⟨N⟩.P) **and** *Req*: R = M⟨N⟩.⦇νx⦈P **and** x ♯ Ψ **and** x ♯ M **and** x ♯ N **by** *auto*
obtain y::*name* **where** y ♯ Ψ **and** y ♯ Ψ′ **and** y ♯ M **and** y ♯ N **and** y ♯ P
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

moreover hence (Ψ ⊗ Ψ′, ⦇νy⦈(M⟨N⟩.([(x, y)] ∙ P)), M⟨N⟩.⦇νy⦈([(x, y)] ∙ P)) ∈ *?X* **by** *auto*
moreover from *Qeq* ‹x ♯ M› ‹y ♯ M› ‹x ♯ N› ‹y ♯ N› ‹y ♯ P› **have** Q = ⦇νy⦈(M⟨N⟩.([(x, y)] ∙ P))
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)
moreover from *Req* ‹y ♯ P› **have** R = M⟨N⟩.⦇νy⦈([(x, y)] ∙ P)
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
assume (Ψ, Q, R) ∉ *?X1*
with ‹(Ψ, Q, R) ∈ *?X*› **have** (Ψ, Q, R) ∈ *?X2* **by** *blast*
then obtain x M N P **where** *Req*: R = ⦇νx⦈(M⟨N⟩.P) **and** *Qeq*: Q = M⟨N⟩.⦇νx⦈P **and** x ♯ Ψ **and** x ♯ M **and** x ♯ N **by** *auto*
obtain y::*name* **where** y ♯ Ψ **and** y ♯ Ψ′ **and** y ♯ M **and** y ♯ N **and** y ♯ P
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

moreover hence (Ψ ⊗ Ψ′, ⦇νy⦈(M⟨N⟩.([(x, y)] ∙ P)), M⟨N⟩.⦇νy⦈([(x, y)] ∙ P)) ∈ *?X* **by** *auto*
moreover from *Req* ‹x ♯ M› ‹y ♯ M› ‹x ♯ N› ‹y ♯ N› ‹y ♯ P› **have** R = ⦇νy⦈(M⟨N⟩.([(x, y)] ∙ P))
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)

```
      moreover from Qeq ‹y ♯ P› have Q = M⟨N⟩.⦇νy⦈([(x, y)] • P)
        apply auto by(subst alphaRes[of y]) (auto simp add: eqvts)
      ultimately show ?case by blast
    qed
  next
    case(cSym Ψ R Q)
    thus ?case by blast
  qed
 }
 moreover obtain y::name where y ♯ Ψ and y ♯ M and y ♯ N y ♯ P
  by(generate-fresh name) auto
 ultimately have Ψ ▷ ⦇νy⦈(M⟨N⟩.([(x, y)] • P)) ∼ M⟨N⟩.⦇νy⦈([(x, y)] • P) by
auto
 thus ?thesis using assms ‹y ♯ P› ‹y ♯ M› ‹y ♯ N›
  apply(subst alphaRes[where x=x and y=y and P=P], auto)
  by(subst alphaRes[where x=x and y=y and P=M⟨N⟩.P]) auto
qed

lemma bisimInputPushRes:
 fixes x    :: name
 and   Ψ    :: 'b
 and   M    :: 'a
 and   xvec :: name list
 and   N    :: 'a
 and   P    :: ('a, 'b, 'c) psi

 assumes x ♯ M
 and     x ♯ xvec
 and     x ♯ N

 shows Ψ ▷ ⦇νx⦈(M⦇λ*xvec N⦈.P) ∼ M⦇λ*xvec N⦈.⦇νx⦈P
proof −
 {
  fix x::name and P::('a, 'b, 'c) psi
  assume x ♯ Ψ and x ♯ M and x ♯ N and x ♯ xvec
  let ?X1 = {(Ψ, ⦇νx⦈(M⦇λ*xvec N⦈.P), M⦇λ*xvec N⦈.⦇νx⦈P) | Ψ x M xvec N
P. x ♯ Ψ ∧ x ♯ M ∧ x ♯ xvec ∧ x ♯ N}
  let ?X2 = {(Ψ, M⦇λ*xvec N⦈.⦇νx⦈P, ⦇νx⦈(M⦇λ*xvec N⦈.P)) | Ψ x M xvec N
P. x ♯ Ψ ∧ x ♯ M ∧ x ♯ xvec ∧ x ♯ N}
  let ?X = ?X1 ∪ ?X2

  have eqvt ?X by(rule-tac eqvtUnion) (force simp add: eqvt-def pt-fresh-bij[OF
pt-name-inst, OF at-name-inst] eqvts)+

  from ‹x ♯ Ψ› ‹x ♯ M› ‹x ♯ xvec› ‹x ♯ N› have (Ψ, ⦇νx⦈(M⦇λ*xvec N⦈.P),
M⦇λ*xvec N⦈.⦇νx⦈P) ∈ ?X by blast
  hence Ψ ▷ ⦇νx⦈(M⦇λ*xvec N⦈.P) ∼ M⦇λ*xvec N⦈.⦇νx⦈P
  proof(coinduct rule: bisimCoinduct)
    case(cStatEq Ψ Q R)
```

thus *?case* **using** *freshComp* **by**(*force intro*: *frameResFresh FrameStatEqSym*)
next
case(*cSim* Ψ Q R)
thus *?case* **using** ‹*eqvt ?X*›
by(*fastforce intro*: *inputPushResLeft inputPushResRight bisimReflexive*)
next
case(*cExt* Ψ Q R Ψ')
show *?case*
proof(*case-tac* $(\Psi, Q, R) \in$ *?X1*)
assume $(\Psi, Q, R) \in$ *?X1*
then obtain x M *xvec* N P **where** *Qeq*: $Q = (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P)$ **and** *Req*: $R = M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$ **and** $x \sharp \Psi$
and $x \sharp M$ **and** $x \sharp xvec$ **and** $x \sharp N$ **by** *auto*
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \Psi'$ **and** $y \sharp M$ **and** $y \sharp N$ **and** $y \sharp P$ **and** $y \sharp xvec$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

moreover hence $(\Psi \otimes \Psi', (\!|\nu y|\!)(M(\!|\lambda{*}xvec\ N|\!).([(x, y)] \cdot P)), M(\!|\lambda{*}xvec\ N|\!).(\!|\nu y|\!)([(x, y)] \cdot P)) \in$ *?X* **by** *fastforce*
moreover from *Qeq* ‹$x \sharp M$› ‹$y \sharp M$› ‹$x \sharp xvec$› ‹$y \sharp xvec$› ‹$x \sharp N$› ‹$y \sharp N$› ‹$y \sharp P$› **have** $Q = (\!|\nu y|\!)(M(\!|\lambda{*}xvec\ N|\!).([(x, y)] \cdot P))$
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts inputChainFresh*)
moreover from *Req* ‹$y \sharp P$› **have** $R = M(\!|\lambda{*}xvec\ N\ |\!).(\!|\nu y|\!)([(x, y)] \cdot P)$
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
assume $(\Psi, Q, R) \notin$ *?X1*
with ‹$(\Psi, Q, R) \in$ *?X*› **have** $(\Psi, Q, R) \in$ *?X2* **by** *blast*
then obtain x M *xvec* N P **where** *Req*: $R = (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P)$ **and** *Qeq*: $Q = M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$ **and** $x \sharp \Psi$
and $x \sharp M$ **and** $x \sharp xvec$ **and** $x \sharp N$ **by** *auto*
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \Psi'$ **and** $y \sharp M$ **and** $y \sharp N$ **and** $y \sharp P$ **and** $y \sharp xvec$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

moreover hence $(\Psi \otimes \Psi', (\!|\nu y|\!)(M(\!|\lambda{*}xvec\ N|\!).([(x, y)] \cdot P)), M(\!|\lambda{*}xvec\ N|\!).(\!|\nu y|\!)([(x, y)] \cdot P)) \in$ *?X* **by** *fastforce*
moreover from *Req* ‹$x \sharp M$› ‹$y \sharp M$› ‹$x \sharp xvec$› ‹$y \sharp xvec$› ‹$x \sharp N$› ‹$y \sharp N$› ‹$y \sharp P$› **have** $R = (\!|\nu y|\!)(M(\!|\lambda{*}xvec\ N|\!).([(x, y)] \cdot P))$
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts inputChainFresh*)
moreover from *Qeq* ‹$y \sharp P$› **have** $Q = M(\!|\lambda{*}xvec\ N\ |\!).(\!|\nu y|\!)([(x, y)] \cdot P)$
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
qed
next
case(*cSym* Ψ R Q)
thus *?case* **by** *blast*
qed
}

moreover obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp M$ **and** $y \sharp N$ **and** $y \sharp P$ **and** $y \sharp xvec$
by(*generate-fresh name*) *auto*
ultimately have $\Psi \rhd (\!|\nu y|\!)(M(\!|\lambda{*}xvec\ N|\!).([(x, y)] \cdot P)) \sim M(\!|\lambda{*}xvec\ N|\!).(\!|\nu y|\!)([(x, y)] \cdot P)$ **by** *auto*
thus *?thesis* **using** *assms* ‹$y \sharp P$› ‹$y \sharp M$› ‹$y \sharp N$› ‹$y \sharp xvec$›
apply(*subst alphaRes*[**where** $x{=}x$ **and** $y{=}y$ **and** $P{=}P$], *auto*)
by(*subst alphaRes*[**where** $x{=}x$ **and** $y{=}y$ **and** $P{=}M(\!|\lambda{*}xvec\ N|\!).P$]) (*auto simp add*: *inputChainFresh eqvts*)
qed

lemma *bisimCasePushRes*:
fixes x :: *name*
and Ψ :: $'b$
and Cs :: $('c \times ('a, 'b, 'c)\ psi)\ list$

assumes $x \sharp (map\ fst\ Cs)$

shows $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \sim Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs)$
proof −
{
fix x::*name* **and** Cs::$('c \times ('a, 'b, 'c)\ psi)\ list$
assume $x \sharp \Psi$ **and** $x \sharp (map\ fst\ Cs)$
let $?X1 = \{(\Psi, (\!|\nu x|\!)(Cases\ Cs), Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs)) \mid \Psi\ x\ Cs.\ x \sharp \Psi \wedge x \sharp (map\ fst\ Cs)\}$
let $?X2 = \{(\Psi, Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs), (\!|\nu x|\!)(Cases\ Cs)) \mid \Psi\ x\ Cs.\ x \sharp \Psi \wedge x \sharp (map\ fst\ Cs)\}$
let $?X = ?X1 \cup ?X2$

have *eqvt* $?X$ **apply**(*rule-tac eqvtUnion*)
apply(*auto simp add*: *eqvt-def eqvts*)
apply(*rule-tac* $x{=}p \cdot x$ **in** *exI*)
apply(*rule-tac* $x{=}p \cdot Cs$ **in** *exI*)
apply(*perm-extend-simp*)
apply(*auto simp add*: *eqvts*)
apply(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *eqvts*)
apply(*perm-extend-simp*)
apply(*simp add*: *eqvts*)
apply(*rule-tac* $x{=}p \cdot x$ **in** *exI*)
apply(*rule-tac* $x{=}p \cdot Cs$ **in** *exI*)
apply *auto*
apply(*perm-extend-simp*)
apply(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*drule-tac pi=p* **in** *pt-fresh-bij1*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *eqvts*)

apply(*perm-extend-simp*)
by(*simp add*: *eqvts*)

from ‹$x \sharp \Psi$› ‹$x \sharp$ *map fst Cs*› **have** $(\Psi, (\!|\nu x|\!)(\mathit{Cases}\ \mathit{Cs}), \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ \mathit{Cs})) \in ?X$ **by** *auto*
hence $\Psi \rhd (\!|\nu x|\!)(\mathit{Cases}\ \mathit{Cs}) \sim \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ \mathit{Cs})$
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* Ψ Q R)
thus *?case* **using** *freshComp* **by**(*force intro*: *frameResFresh FrameStatEqSym*)
next
case(*cSim* Ψ Q R)
thus *?case* **using** ‹*eqvt ?X*›
by(*fastforce intro*: *casePushResLeft casePushResRight bisimReflexive*)
next
case(*cExt* Ψ Q R Ψ')
show *?case*
proof(*case-tac* $(\Psi, Q, R) \in ?X1$)
assume $(\Psi, Q, R) \in ?X1$
then obtain x *Cs* **where** *Qeq*: $Q = (\!|\nu x|\!)(\mathit{Cases}\ \mathit{Cs})$ **and** *Req*: $R = \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ \mathit{Cs})$
and $x \sharp \Psi$ **and** $x \sharp$ (*map fst Cs*) **by** *blast*
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \Psi'$ **and** $y \sharp \mathit{Cs}$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
from ‹$y \sharp \mathit{Cs}$› ‹$x \sharp$ (*map fst Cs*)› **have** $y \sharp \mathit{map\ fst}\ ([(x, y)] \cdot \mathit{Cs})$ **by**(*induct Cs*) (*auto simp add*: *fresh-list-cons fresh-list-nil*)

moreover with ‹$y \sharp \Psi$› ‹$y \sharp \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu y|\!)(\mathit{Cases}\ ([(x, y)] \cdot \mathit{Cs})), \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot \mathit{Cs}))) \in ?X$
by *auto*
moreover from *Qeq* ‹$y \sharp \mathit{Cs}$› **have** $Q = (\!|\nu y|\!)(\mathit{Cases}([(x, y)] \cdot \mathit{Cs}))$
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)
moreover from *Req* ‹$y \sharp \mathit{Cs}$› ‹$x \sharp$ (*map fst Cs*)› **have** $R = \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot \mathit{Cs}))$
by(*induct Cs arbitrary*: *R*) (*auto simp add*: *fresh-list-cons fresh-prod alphaRes*)
ultimately show *?case* **by** *blast*
next
assume $(\Psi, Q, R) \notin ?X1$
with ‹$(\Psi, Q, R) \in ?X$› **have** $(\Psi, Q, R) \in ?X2$ **by** *blast*
then obtain x *Cs* **where** *Req*: $R = (\!|\nu x|\!)(\mathit{Cases}\ \mathit{Cs})$ **and** *Qeq*: $Q = \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ \mathit{Cs})$
and $x \sharp \Psi$ **and** $x \sharp$ (*map fst Cs*) **by** *blast*
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \Psi'$ **and** $y \sharp \mathit{Cs}$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
from ‹$y \sharp \mathit{Cs}$› ‹$x \sharp$ (*map fst Cs*)› **have** $y \sharp \mathit{map\ fst}\ ([(x, y)] \cdot \mathit{Cs})$ **by**(*induct Cs*) (*auto simp add*: *fresh-list-cons fresh-list-nil*)

moreover with ‹$y \sharp \Psi$› ‹$y \sharp \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu y|\!)(\mathit{Cases}\ ([(x, y)] \cdot \mathit{Cs})), \mathit{Cases}(\mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot \mathit{Cs}))) \in ?X$

by *auto*
moreover from *Req* ‹$y \sharp Cs$› **have** $R = (\!|\nu y|\!)(Cases([(x, y)] \cdot Cs))$
apply *auto* **by**(*subst alphaRes*[*of y*]) (*auto simp add*: *eqvts*)
moreover from *Qeq* ‹$y \sharp Cs$› ‹$x \sharp (map\ fst\ Cs)$› **have** $Q = Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot Cs))$
by(*induct Cs arbitrary*: *Q*) (*auto simp add*: *fresh-list-cons fresh-prod alphaRes*)
ultimately show *?case* **by** *blast*
qed
next
case(*cSym* Ψ *R Q*)
thus *?case* **by** *blast*
qed
}
moreover obtain *y*::*name* **where** $y \sharp \Psi$ **and** $y \sharp Cs$ **by**(*generate-fresh name*) *auto*
moreover from ‹$x \sharp map\ fst\ Cs$› **have** $y \sharp map\ fst([(x, y)] \cdot Cs)$
by(*induct Cs*) (*auto simp add*: *fresh-left calc-atm*)
ultimately have $\Psi \rhd (\!|\nu y|\!)(Cases\ ([(x, y)] \cdot Cs)) \sim Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot Cs))$
by *auto*
moreover from ‹$y \sharp Cs$› **have** $(\!|\nu y|\!)(Cases\ ([(x, y)] \cdot Cs)) = (\!|\nu x|\!)(Cases\ Cs)$
by(*simp add*: *alphaRes eqvts*)
moreover from ‹$x \sharp map\ fst\ Cs$› ‹$y \sharp Cs$› **have** $Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot Cs)) = Cases(map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs)$
by(*induct Cs*) (*auto simp add*: *alphaRes*)
ultimately show *?thesis* **by** *auto*
qed

lemma *bangExt*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

assumes *guarded P*

shows $\Psi \rhd !P \sim P \parallel !P$
proof −
let $?X = \{(\Psi, !P, P \parallel !P) \mid \Psi\ P.\ guarded\ P\} \cup \{(\Psi, P \parallel !P, !P) \mid \Psi\ P.\ guarded\ P\}$
from ‹*guarded P*› **have** $(\Psi, !P, P \parallel !P) \in ?X$ **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *bisimCoinduct*)
case(*cStatEq* Ψ *Q R*)
from ‹$(\Psi, Q, R) \in ?X$› **obtain** *P* **where** *Eq*: $(Q = !P \wedge R = P \parallel !P) \vee (Q = P \parallel !P \wedge R = !P)$ **and** *guarded P*
by *auto*
obtain $A_P\ \Psi_P$ **where** *FrP*: $extractFrame\ P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **by**(*rule freshFrame*)
from *FrP* ‹*guarded P*› **have** $\Psi_P \simeq SBottom'$ **by**(*blast dest*: *guardedStatEq*)

from ‹$\Psi_P \simeq SBottom'$› **have** $\Psi \otimes SBottom' \simeq \Psi \otimes \Psi_P \otimes SBottom'$ **by**(*metis Identity Composition AssertionStatEqTrans Commutativity AssertionStatEqSym*)
hence $\langle A_P, \Psi \otimes SBottom' \rangle \simeq_F \langle A_P, \Psi \otimes \Psi_P \otimes SBottom' \rangle$
by(*force intro*: *frameResChainPres*)
moreover from ‹$A_P \sharp* \Psi$› **have** $\langle \varepsilon, \Psi \otimes SBottom' \rangle \simeq_F \langle A_P, \Psi \otimes SBottom' \rangle$
by(*rule-tac FrameStatEqSym*) (*fastforce intro*: *frameResFreshChain*)
ultimately show *?case* **using** *Eq* ‹$A_P \sharp* \Psi$› *FrP*
by *auto* (*blast dest*: *FrameStatEqTrans FrameStatEqSym*)+
next
case(*cSim* Ψ Q R)
thus *?case* **by**(*auto intro*: *bangExtLeft bangExtRight bisimReflexive*)
next
case(*cExt* Ψ Q R)
thus *?case* **by** *auto*
next
case(*cSym* Ψ Q R)
thus *?case* **by** *auto*
qed
qed

lemma *bisimParPresSym*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \sim Q$

shows $\Psi \rhd R \parallel P \sim R \parallel Q$
using *assms*
by(*metis bisimParComm bisimParPres bisimTransitive*)

lemma *bisimScopeExtSym*:
fixes x :: *name*
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp Q$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \sim ((\!|\nu x|\!)P) \parallel Q$
using *assms*
by(*metis bisimScopeExt bisimTransitive bisimParComm bisimSymmetric bisimResPres*)

lemma *bisimScopeExtChainSym*:
fixes *xvec* :: *name list*
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*

assumes $xvec \sharp* \Psi$
and $xvec \sharp* Q$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \sim ((\!|\nu{*}xvec|\!)P) \parallel Q$
using *assms*
by(*induct xvec*) (*auto intro*: *bisimScopeExtSym bisimReflexive bisimTransitive bisimResPres*)

lemma *bisimParPresAuxSym*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $R :: ('a, 'b, 'c)\ psi$

assumes $\Psi \otimes \Psi_R \rhd P \sim Q$
and $extractFrame\ R = \langle A_R, \Psi_R \rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$

shows $\Psi \rhd R \parallel P \sim R \parallel Q$
using *assms*
by(*metis bisimParComm bisimParPresAux bisimTransitive*)

lemma *bangDerivative*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $\alpha :: 'a\ action$
and $P' :: ('a, 'b, 'c)\ psi$

assumes $\Psi \rhd !P \longmapsto \alpha \prec P'$
and $\Psi \rhd P \sim Q$
and $bn\ \alpha \sharp* \Psi$
and $bn\ \alpha \sharp* P$
and $bn\ \alpha \sharp* Q$
and $bn\ \alpha \sharp* subject\ \alpha$
and $guarded\ Q$

obtains $Q'\ R\ T$ **where** $\Psi \rhd !Q \longmapsto \alpha \prec Q'$ **and** $\Psi \rhd P' \sim R \parallel !P$ **and** $\Psi \rhd Q' \sim T \parallel !Q$ **and** $\Psi \rhd R \sim T$
and $((supp\ R)::name\ set) \subseteq supp\ P'$ **and** $((supp\ T)::name\ set) \subseteq supp\ Q'$
proof −
from ‹$\Psi \rhd !P \longmapsto \alpha \prec P'$› **have** *guarded P* **apply** − **by**(*ind-cases* $\Psi \rhd !P \longmapsto \alpha \prec P'$) (*auto simp add*: *psi.inject*)
assume $\bigwedge Q'\ R\ T.\ [\![\Psi \rhd !Q \longmapsto \alpha \prec Q';\ \Psi \rhd P' \sim R \parallel !P;\ \Psi \rhd Q' \sim T \parallel !Q;\ \Psi \rhd R \sim T;\ ((supp\ R)::name\ set) \subseteq supp\ P';$
$((supp\ T)::name\ set) \subseteq supp\ Q']\!] \Longrightarrow thesis$

moreover from ‹$\Psi \rhd !P \longmapsto\alpha \prec P'$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$\Psi \rhd P \sim Q$› ‹*guarded* Q›
have $\exists\, Q'\ T\ R\ .\ \Psi \rhd !Q \longmapsto\alpha \prec Q' \wedge \Psi \rhd P' \sim R \parallel !P \wedge \Psi \rhd Q' \sim T \parallel !Q \wedge \Psi \rhd R \sim T \wedge$
$((supp\ R)::name\ set) \subseteq supp\ P' \wedge ((supp\ T)::name\ set) \subseteq supp\ Q'$
proof(*nominal-induct avoiding*: Q *rule*: *bangInduct'*)
case(*cAlpha* $\alpha\ P'\ p\ Q$)
then obtain $Q'\ T\ R$ **where** *QTrans*: $\Psi \rhd !Q \longmapsto\alpha \prec Q'$ **and** $\Psi \rhd P' \sim R \parallel (P \parallel !P)$ **and** $\Psi \rhd Q' \sim T \parallel !Q$ **and** $\Psi \rhd R \sim T$
and *suppR*: $((supp\ R)::name\ set) \subseteq supp\ P'$ **and** *suppT*: $((supp\ T)::name\ set) \subseteq supp\ Q'$
by *blast*
from *QTrans* **have** $distinct(bn\ \alpha)$ **by**(*rule boundOutputDistinct*)
have S: $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$ **by** *fact*
from *QTrans* ‹$bn(p \cdot \alpha) \sharp* Q$› ‹$bn(p \cdot \alpha) \sharp* \alpha$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$distinct(bn\ \alpha)$› **have** $bn(p \cdot \alpha) \sharp* Q'$
by(*drule-tac freeFreshChainDerivative*) *simp+*
with *QTrans* ‹$bn(p \cdot \alpha) \sharp* \alpha$› S ‹$bn\ \alpha \sharp* subject\ \alpha$› **have** $\Psi \rhd !Q \longmapsto(p \cdot \alpha) \prec (p \cdot Q')$
by(*force simp add*: *residualAlpha*)
moreover from ‹$\Psi \rhd P' \sim R \parallel (P \parallel !P)$› **have** $(p \cdot \Psi) \rhd (p \cdot P') \sim (p \cdot (R \parallel (P \parallel !P)))$
by(*rule bisimClosed*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn(p \cdot \alpha) \sharp* \Psi$› ‹$bn(p \cdot \alpha) \sharp* P$› S **have** $\Psi \rhd (p \cdot P') \sim (p \cdot R) \parallel (P \parallel !P)$
by(*simp add*: *eqvts*)
moreover from ‹$\Psi \rhd Q' \sim T \parallel !Q$› **have** $(p \cdot \Psi) \rhd (p \cdot Q') \sim (p \cdot (T \parallel !Q))$
by(*rule bisimClosed*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* Q$› ‹$bn(p \cdot \alpha) \sharp* \Psi$› ‹$bn(p \cdot \alpha) \sharp* Q$› S **have** $\Psi \rhd (p \cdot Q') \sim (p \cdot T) \parallel !Q$
by(*simp add*: *eqvts*)
moreover from ‹$\Psi \rhd R \sim T$› **have** $(p \cdot \Psi) \rhd (p \cdot R) \sim (p \cdot T)$
by(*rule bisimClosed*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn(p \cdot \alpha) \sharp* \Psi$› S **have** $\Psi \rhd (p \cdot R) \sim (p \cdot T)$
by(*simp add*: *eqvts*)
moreover from *suppR* **have** $((supp(p \cdot R))::name\ set) \subseteq supp(p \cdot P')$
apply(*erule-tac rev-mp*)
by(*subst subsetClosed*[*of p*, *symmetric*]) (*simp add*: *eqvts*)
moreover from *suppT* **have** $((supp(p \cdot T))::name\ set) \subseteq supp(p \cdot Q')$
apply(*erule-tac rev-mp*)
by(*subst subsetClosed*[*of p*, *symmetric*]) (*simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
case(*cPar1* $\alpha\ P'\ Q$)
from ‹$\Psi \rhd P \sim Q$› ‹$\Psi \rhd P \longmapsto\alpha \prec P'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* Q$›
obtain Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto\alpha \prec Q'$ **and** $\Psi \rhd P' \sim Q'$
by(*blast dest*: *bisimE simE*)
from *QTrans* **have** $\Psi \otimes SBottom' \rhd Q \longmapsto\alpha \prec Q'$ **by**(*metis statEqTransition Identity AssertionStatEqSym*)

hence $\Psi \rhd Q \parallel !Q \longmapsto\alpha \prec (Q' \parallel !Q)$ **using** ‹*bn* α $\sharp*$ Q› **by**(*rule-tac Par1*) (*assumption* | *simp*)+

hence $\Psi \rhd !Q \longmapsto\alpha \prec (Q' \parallel !Q)$ **using** ‹*guarded* Q› **by**(*rule Bang*)

moreover from ‹*guarded* P› **have** $\Psi \rhd P' \parallel !P \sim P' \parallel (P \parallel !P)$ **by**(*metis bangExt bisimParPresSym*)

moreover have $\Psi \rhd Q' \parallel !Q \sim Q' \parallel !Q$ **by**(*rule bisimReflexive*)

ultimately show *?case* **using** ‹$\Psi \rhd P' \sim Q'$› **by**(*force simp add*: *psi.supp*)

next

case(*cPar2* α P' Q)

then obtain Q' T R **where** *QTrans*: $\Psi \rhd !Q \longmapsto\alpha \prec Q'$ **and** $\Psi \rhd P' \sim R \parallel !P$ **and** $\Psi \rhd Q' \sim T \parallel !Q$ **and** $\Psi \rhd R \sim T$

and *suppR*: ((*supp* R)::*name set*) $\subseteq$ *supp* P' **and** *suppT*: ((*supp* T)::*name set*) $\subseteq$ *supp* Q'

by *blast*

note *QTrans*

from ‹$\Psi \rhd P' \sim R \parallel !P$› **have** $\Psi \rhd P \parallel P' \sim R \parallel (P \parallel !P)$

by(*metis bisimParPresSym bisimParComm bisimTransitive bisimParAssoc*)

with *QTrans* **show** *?case* **using** ‹$\Psi \rhd Q' \sim T \parallel !Q$› ‹$\Psi \rhd R \sim T$› *suppR* *suppT*

by(*force simp add*: *psi.supp*)

next

case(*cComm1* M N P' K *xvec* P'' Q)

from ‹$\Psi \rhd P \sim Q$› **have** $\Psi \rhd Q \rightsquigarrow[bisim]\ P$ **by**(*metis bisimE*)

with ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› **obtain** Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto M(\!|N|\!) \prec Q'$ **and** $\Psi \rhd Q' \sim P'$

by(*force dest*: *simE*)

from *QTrans* **have** $\Psi \otimes SBottom' \rhd Q \longmapsto M(\!|N|\!) \prec Q'$ **by**(*metis statEqTransition Identity AssertionStatEqSym*)

moreover obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* Q$ **and** $A_Q \sharp* M$

by(*rule-tac* C=(Ψ, Q, M) **in** *freshFrame*) *auto*

note *FrQ*

moreover from *FrQ* ‹*guarded* Q› **have** $\Psi_Q \simeq SBottom'$ **by**(*blast dest*: *guardedStatEq*)

from ‹$\Psi \rhd !P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P''$› ‹*xvec* $\sharp*$ K› ‹$\Psi \rhd P \sim Q$› ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ P› ‹*xvec* $\sharp*$ Q› ‹*guarded* Q›

obtain Q'' T R **where** *QTrans'*: $\Psi \rhd !Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q''$ **and** $\Psi \rhd P'' \sim R \parallel !P$ **and** $\Psi \rhd Q'' \sim T \parallel !Q$ **and** $\Psi \rhd R \sim T$

and *suppR*: ((*supp* R)::*name set*) $\subseteq$ *supp* P'' **and** *suppT*: ((*supp* T)::*name set*) $\subseteq$ *supp* Q'' **using** *cComm1*

by *fastforce*

from *QTrans'* ‹$\Psi_Q \simeq SBottom'$› **have** $\Psi \otimes \Psi_Q \rhd !Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q''$

by(*metis statEqTransition Identity compositionSym AssertionStatEqSym*)

moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹$\Psi_Q \simeq SBottom'$› **have** $\Psi \otimes \Psi_Q \otimes SBottom' \vdash M \leftrightarrow K$ **by**(*metis statEqEnt Identity compositionSym AssertionStatEqSym*)

ultimately have $\Psi \rhd Q \parallel !Q \longmapsto\tau \prec ((\!|\nu*xvec|\!)(Q' \parallel Q''))$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* M$› ‹*xvec* $\sharp*$ Q›

by(*rule-tac Comm1*) (*assumption* | *simp*)+

hence $\Psi \rhd !Q \longmapsto\tau \prec ((\!|\nu*xvec|\!)(Q' \parallel Q''))$ **using** ‹*guarded Q*› **by**(*rule Bang*)
moreover from ‹$\Psi \rhd P'' \sim R \parallel !P$› ‹*guarded P*› ‹*xvec* $\sharp*$ Ψ› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P' \parallel P'') \sim (\!|\nu*xvec|\!)((P' \parallel R) \parallel (P \parallel !P))$
by(*metis bisimParPresSym bangExt bisimTransitive bisimParAssoc bisimSymmetric bisimResChainPres*)
with ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ *P*› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P' \parallel P'') \sim ((\!|\nu*xvec|\!)(P' \parallel R)) \parallel (P \parallel !P)$
by(*metis bisimScopeExtChainSym bisimTransitive psiFreshVec*)
moreover from ‹$\Psi \rhd Q'' \sim T \parallel !Q$› ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ *Q*› **have** $\Psi \rhd (\!|\nu*xvec|\!)(Q' \parallel Q'') \sim ((\!|\nu*xvec|\!)(Q' \parallel T)) \parallel !Q$
by(*metis bisimParPresSym bisimTransitive bisimParAssoc bisimSymmetric bisimResChainPres bisimScopeExtChainSym psiFreshVec*)
moreover from ‹$\Psi \rhd R \sim T$› ‹$\Psi \rhd Q' \sim P'$› ‹*xvec* $\sharp*$ Ψ› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P' \parallel R) \sim (\!|\nu*xvec|\!)(Q' \parallel T)$
by(*metis bisimParPresSym bisimTransitive bisimResChainPres bisimParComm bisimE*(*4*))
moreover from *suppR* **have** $((supp((\!|\nu*xvec|\!)(P' \parallel R)))::name\ set) \subseteq supp(((\!|\nu*xvec|\!)(P' \parallel P''))$
by(*auto simp add*: *psi.supp resChainSupp*)
moreover from *suppT* **have** $((supp((\!|\nu*xvec|\!)(Q' \parallel T)))::name\ set) \subseteq supp(((\!|\nu*xvec|\!)(Q' \parallel Q''))$
by(*auto simp add*: *psi.supp resChainSupp*)
ultimately show *?case* **by** *blast*
next
case(*cComm2 M xvec N P' K P'' Q*)
from ‹$\Psi \rhd P \sim Q$› ‹$\Psi \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ *Q*›
obtain Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **and** $\Psi \rhd P' \sim Q'$
by(*metis bisimE simE bn.simps*)
from *QTrans* **have** $\Psi \otimes SBottom' \rhd Q \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$ **by**(*metis statEqTransition Identity AssertionStatEqSym*)
moreover obtain A_Q Ψ_Q **where** *FrQ*: $extractFrame\ Q = \langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* Q$ **and** $A_Q \sharp* M$
by(*rule-tac* C=(Ψ, Q, M) **in** *freshFrame*) *auto*
note *FrQ*
moreover from *FrQ* ‹*guarded Q*› **have** $\Psi_Q \simeq SBottom'$ **by**(*blast dest*: *guardedStatEq*)
from ‹$\Psi \rhd !P \longmapsto K(\!|N|\!) \prec P''$› ‹$\Psi \rhd P \sim Q$› ‹*guarded Q*›
obtain Q'' T R **where** *QTrans'*: $\Psi \rhd !Q \longmapsto K(\!|N|\!) \prec Q''$ **and** $\Psi \rhd P'' \sim R \parallel !P$ **and** $\Psi \rhd Q'' \sim T \parallel !Q$ **and** $\Psi \rhd R \sim T$
and *suppR*: $((supp\ R)::name\ set) \subseteq supp\ P''$ **and** *suppT*: $((supp\ T)::name\ set) \subseteq supp\ Q''$ **using** *cComm2*
by *fastforce*
from *QTrans'* ‹$\Psi_Q \simeq SBottom'$› **have** $\Psi \otimes \Psi_Q \rhd !Q \longmapsto K(\!|N|\!) \prec Q''$
by(*metis statEqTransition Identity compositionSym AssertionStatEqSym*)
moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹$\Psi_Q \simeq SBottom'$› **have** $\Psi \otimes \Psi_Q \otimes SBottom' \vdash M \leftrightarrow K$ **by**(*metis statEqEnt Identity compositionSym AssertionStatEqSym*)
ultimately have $\Psi \rhd Q \parallel !Q \longmapsto\tau \prec ((\!|\nu*xvec|\!)(Q' \parallel Q''))$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* M$› ‹*xvec* $\sharp*$ *Q*›
by(*rule-tac Comm2*) (*assumption* | *simp*)+

hence $\Psi \rhd !Q \longmapsto\tau \prec ((\!|\nu*xvec|\!)(Q' \parallel Q''))$ **using** ‹*guarded Q*› **by**(*rule Bang*)
moreover from ‹$\Psi \rhd P'' \sim R \parallel !P$› ‹*guarded P*› ‹*xvec* $\sharp*$ Ψ› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P' \parallel P'') \sim (\!|\nu*xvec|\!)((P' \parallel R) \parallel (P \parallel !P))$
by(*metis bisimParPresSym bangExt bisimTransitive bisimParAssoc bisimSymmetric bisimResChainPres*)
with ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ *P*› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P' \parallel P'') \sim ((\!|\nu*xvec|\!)(P' \parallel R)) \parallel (P \parallel !P)$
by(*metis bisimScopeExtChainSym bisimTransitive psiFreshVec*)
moreover from ‹$\Psi \rhd Q'' \sim T \parallel !Q$› ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ *Q*› **have** $\Psi \rhd (\!|\nu*xvec|\!)(Q' \parallel Q'') \sim ((\!|\nu*xvec|\!)(Q' \parallel T)) \parallel !Q$
by(*metis bisimParPresSym bisimTransitive bisimParAssoc bisimSymmetric bisimResChainPres bisimScopeExtChainSym psiFreshVec*)
moreover from ‹$\Psi \rhd R \sim T$› ‹$\Psi \rhd P' \sim Q'$› ‹*xvec* $\sharp*$ Ψ› **have** $\Psi \rhd (\!|\nu*xvec|\!)(P' \parallel R) \sim (\!|\nu*xvec|\!)(Q' \parallel T)$
by(*metis bisimParPresSym bisimTransitive bisimResChainPres bisimParComm*)
moreover from *suppR* **have** $((supp((\!|\nu*xvec|\!)(P' \parallel R)))::name\ set) \subseteq supp(((\!|\nu*xvec|\!)(P' \parallel P''))$
by(*auto simp add*: *psi.supp resChainSupp*)
moreover from *suppT* **have** $((supp((\!|\nu*xvec|\!)(Q' \parallel T)))::name\ set) \subseteq supp(((\!|\nu*xvec|\!)(Q' \parallel Q''))$
by(*auto simp add*: *psi.supp resChainSupp*)
ultimately show *?case* **by** *blast*
next
case(*cBang* α P' Q)
then obtain Q' T R **where** *QTrans*: $\Psi \rhd !Q \longmapsto\alpha \prec Q'$ **and** $\Psi \rhd P' \sim R \parallel (P \parallel !P)$ **and** $\Psi \rhd Q' \sim T \parallel !Q$ **and** $\Psi \rhd R \sim T$
and *suppR*: $((supp\ R)::name\ set) \subseteq supp\ P'$ **and** *suppT*: $((supp\ T)::name\ set) \subseteq supp\ Q'$
by *blast*
from ‹$\Psi \rhd P' \sim R \parallel (P \parallel !P)$› ‹*guarded P*› **have** $\Psi \rhd P' \sim R \parallel !P$ **by**(*metis bangExt bisimParPresSym bisimTransitive bisimSymmetric*)
with *QTrans* **show** *?case* **using** ‹$\Psi \rhd Q' \sim T \parallel !Q$› ‹$\Psi \rhd R \sim T$› *suppR suppT*
by *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *structCongBisim*:
fixes P :: ('*a*, '*b*, '*c*) *psi*
and Q :: ('*a*, '*b*, '*c*) *psi*

assumes $P \equiv_s Q$

shows $P \sim Q$
using *assms*
by(*induct rule*: *structCong.induct*)
(*auto intro*: *bisimReflexive bisimSymmetric bisimTransitive bisimParComm bisim-*

ParAssoc bisimParNil bisimResNil bisimResComm bisimScopeExt bisimCasePushRes bisimInputPushRes bisimOutputPushRes bangExt)

lemma *bisimBangPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \sim Q$
and *guarded* P
and *guarded* Q

shows $\Psi \rhd !P \sim !Q$
proof −
let $?X = \{(\Psi, R \parallel !P, R \parallel !Q) \mid \Psi\ P\ Q\ R.\ \Psi \rhd P \sim Q \wedge$ *guarded* $P \wedge$ *guarded* $Q\}$
let $?Y = \{(\Psi, P, Q) \mid \Psi\ P\ P'\ Q'\ Q.\ \Psi \rhd P \sim P' \wedge (\Psi, P', Q') \in ?X \wedge \Psi \rhd Q' \sim Q\}$
from *assms* **have** $(\Psi, \mathbf{0} \parallel !P, \mathbf{0} \parallel !Q) \in ?X$ **by**(*blast intro*: *bisimReflexive*)

moreover have *eqvt* $?X$
apply(*auto simp add*: *eqvt-def*)
apply(*drule-tac p=p* **in** *bisimClosed*)
by *fastforce*
ultimately have $\Psi \rhd \mathbf{0} \parallel !P \sim \mathbf{0} \parallel !Q$
proof(*coinduct rule*: *weakTransitiveCoinduct*)
case(*cStatEq* $\Psi\ P\ Q$)
thus *?case* **by** *auto*
next
case(*cSim* $\Psi\ RP\ RQ$)
from ‹$(\Psi, RP, RQ) \in ?X$› **obtain** $P\ Q\ R$ **where** $\Psi \rhd P \sim Q$ **and** *guarded* P **and** *guarded* Q
and $RP = R \parallel !P$ **and** $RQ = R \parallel !Q$
by *auto*
note ‹$\Psi \rhd P \sim Q$›
moreover from ‹*eqvt* $?X$› **have** *eqvt* $?Y$ **by** *blast*
moreover note ‹*guarded* P› ‹*guarded* Q› *bisimE*(*2*) *bisimE*(*3*) *bisimE*(*4*) *statEqBisim bisimClosed bisimParAssoc*[*THEN bisimSymmetric*]
bisimParPres bisimParPresAuxSym bisimResChainPres bisimScopeExtChainSym bisimTransitive
moreover have $\bigwedge \Psi\ P\ Q\ R\ T.\ [\![\Psi \rhd P \sim Q; (\Psi, Q, R) \in ?Y; \Psi \rhd R \sim T]\!] \Longrightarrow (\Psi, P, T) \in ?Y$
by *auto* (*metis bisimTransitive*)
moreover have $\bigwedge \Psi\ P\ Q\ R.\ [\![\Psi \rhd P \sim Q;$ *guarded* P; *guarded* $Q]\!] \Longrightarrow (\Psi, R \parallel !P, R \parallel !Q) \in ?Y$ **by**(*blast intro*: *bisimReflexive*)
moreover have $\bigwedge \Psi\ P\ \alpha\ P'\ Q.\ [\![\Psi \rhd !P \longmapsto \alpha \prec P'; \Psi \rhd P \sim Q;$ *bn* $\alpha \sharp* \Psi$; *bn* $\alpha \sharp* P$; *bn* $\alpha \sharp* Q$; *guarded* Q; *bn* $\alpha \sharp*$ *subject* $\alpha]\!] \Longrightarrow$
$\exists\, Q'\ R\ T.\ \Psi \rhd !Q \longmapsto \alpha \prec Q' \wedge \Psi \rhd P' \sim R \parallel !P \wedge\ \Psi \rhd Q' \sim T \parallel !Q \wedge$

$\Psi \rhd R \sim T \wedge ((supp\ R)::name\ set) \subseteq$
$supp\ P' \wedge$
$((supp\ T)::name\ set) \subseteq supp\ Q'$
by(*blast elim*: *bangDerivative*)
ultimately have $\Psi \rhd R \parallel !P \rightsquigarrow[?Y]\ R \parallel !Q$
by(*rule bangPres*)
with ‹$RP = R \parallel !P$› ‹$RQ = R \parallel !Q$› **show** *?case*
by *blast*
next
case(*cExt* Ψ *RP RQ* Ψ')
thus *?case* **by**(*blast dest*: *bisimE*)
next
case(*cSym* Ψ *RP RQ*)
thus *?case* **by**(*blast dest*: *bisimE*)
qed
thus *?thesis*
by(*metis bisimTransitive bisimParNil bisimSymmetric bisimParComm*)
qed

end

end

theory *Weak-Bisimulation*
imports *Weak-Simulation Weak-Stat-Imp Bisim-Struct-Cong*
begin

context *env* **begin**

lemma *monoCoinduct*: $\bigwedge x\ y\ xa\ xb\ xc\ P\ Q\ \Psi.$
$x \leq y \Longrightarrow$
$(\Psi \rhd Q \rightsquigarrow<\{(xc,\ xb,\ xa).\ x\ xc\ xb\ xa\}>\ P) \longrightarrow$
$(\Psi \rhd Q \rightsquigarrow<\{(xb,\ xa,\ xc).\ y\ xb\ xa\ xc\}>\ P)$
apply *auto*
apply(*rule weakSimMonotonic*)
by(*auto dest*: *le-funE*)

lemma *monoCoinduct2*: $\bigwedge x\ y\ xa\ xb\ xc\ P\ Q\ \Psi.$
$x \leq y \Longrightarrow$
$(\Psi \rhd Q \lessapprox<\{(xc,\ xb,\ xa).\ x\ xc\ xb\ xa\}>\ P) \longrightarrow$
$(\Psi \rhd Q \lessapprox<\{(xb,\ xa,\ xc).\ y\ xb\ xa\ xc\}>\ P)$
apply *auto*
apply(*rule weakStatImpMonotonic*)
by(*auto dest*: *le-funE*)

coinductive-set *weakBisim* :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
where
step: $[\![\Psi \rhd P \lessapprox<weakBisim>\ Q;\ \Psi \rhd P \rightsquigarrow<weakBisim>\ Q;$
$\forall \Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in weakBisim;\ (\Psi,\ Q,\ P) \in weakBisim]\!] \Longrightarrow (\Psi,\ P,$

$Q) \in$ *weakBisim*
monos *monoCoinduct monoCoinduct2*

abbreviation
weakBisimJudge (‹- ▷ - ≈ -› [*70*, *70*, *70*] *65*) **where** $\Psi \triangleright P \approx Q \equiv (\Psi, P, Q) \in$ *weakBisim*
abbreviation
weakBisimNilJudge (‹- ≈ -› [*70*, *70*] *65*) **where** $P \approx Q \equiv \mathbf{1} \triangleright P \approx Q$

lemma *weakBisimCoinductAux*[*consumes 1*]:
fixes F :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and X :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*) *set*

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi \triangleright P \lessapprox <(X \cup \mathit{weakBisim})> Q) \wedge$
$(\Psi \triangleright P \rightsquigarrow <(X \cup \mathit{weakBisim})> Q) \wedge$
$(\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in X \vee (\Psi \otimes \Psi', P, Q) \in$ *weakBisim*$) \wedge$
$((\Psi, Q, P) \in X \vee (\Psi, Q, P) \in$ *weakBisim*$)$

shows $(\Psi, P, Q) \in$ *weakBisim*
proof −
have $X \cup$ *weakBisim* $= \{(\Psi, P, Q).\ (\Psi, P, Q) \in X \vee (\Psi, P, Q) \in$ *weakBisim*$\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct* (*simp add*: *rtrancl-def*)
qed

lemma *weakBisimCoinduct*[*consumes 1*, *case-names cStatImp cSim cExt cSym*]:
fixes F :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and X :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*) *set*

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi'\ R\ S.\ (\Psi', R, S) \in X \Longrightarrow \Psi' \triangleright R \lessapprox <(X \cup \mathit{weakBisim})> S$
and $\bigwedge \Psi'\ R\ S.\ (\Psi', R, S) \in X \Longrightarrow \Psi' \triangleright R \rightsquigarrow <(X \cup \mathit{weakBisim})> S$
and $\bigwedge \Psi'\ R\ S\ \Psi''.\ (\Psi', R, S) \in X \Longrightarrow (\Psi' \otimes \Psi'', R, S) \in X \vee \Psi' \otimes \Psi'' \triangleright R \approx S$
and $\bigwedge \Psi'\ R\ S.\ (\Psi', R, S) \in X \Longrightarrow (\Psi', S, R) \in X \vee \Psi' \triangleright S \approx R$

shows $\Psi \triangleright P \approx Q$
proof −
have $X \cup$ *weakBisim* $= \{(\Psi, P, Q).\ (\Psi, P, Q) \in X \vee (\Psi, P, Q) \in$ *weakBisim*$\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct* (*simp add*: *rtrancl-def*)

qed

lemma *weakBisimWeakCoinductAux*[*consumes 1*]:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $(\Psi,\ P,\ Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \lessapprox <X> Q \wedge \Psi \rhd P \rightsquigarrow <X> Q \wedge (\forall \Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in X) \wedge (\Psi,\ Q,\ P) \in X$

shows $\Psi \rhd P \approx Q$
using *assms*
by(*coinduct rule*: *weakBisimCoinductAux*) (*blast intro*: *weakSimMonotonic weakStatImpMonotonic*)

lemma *weakBisimWeakCoinduct*[*consumes 1*, *case-names cStatImp cSim cExt cSym*]:
fixes F :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $(\Psi,\ P,\ Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \lessapprox <X> Q$
and $\bigwedge \Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow <X> Q$
and $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi,\ Q,\ P) \in X$

shows $(\Psi,\ P,\ Q) \in weakBisim$
proof −
have $X \cup weakBisim = \{(\Psi,\ P,\ Q).\ (\Psi,\ P,\ Q) \in X \vee (\Psi,\ P,\ Q) \in weakBisim\}$
by *auto*
with *assms* **show** *?thesis*
by(*coinduct rule*: *weakBisimWeakCoinductAux*) *blast*
qed

lemma *weakBisimE*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and Ψ :: $'b$
and Ψ' :: $'b$

assumes $\Psi \rhd P \approx Q$

shows $\Psi \rhd P \lessapprox <weakBisim> Q$
and $\Psi \rhd P \rightsquigarrow <weakBisim> Q$
and $\Psi \otimes \Psi' \rhd P \approx Q$
and $\Psi \rhd Q \approx P$

using *assms*
by(*auto intro*: *weakBisim.cases simp add*: *rtrancl-def*)

lemma *weakBisimI*:
  fixes P :: ($'a$, $'b$, $'c$) *psi*
  and Q :: ($'a$, $'b$, $'c$) *psi*
  and Ψ :: $'b$

  assumes $\Psi \rhd P \lessapprox$<*weakBisim*> Q
  and $\Psi \rhd P \rightsquigarrow$<*weakBisim*> Q
  and $\forall \Psi'.\ \Psi \otimes \Psi' \rhd P \approx Q$
  and $\Psi \rhd Q \approx P$

  shows $\Psi \rhd P \approx Q$
using *assms*
by(*rule-tac weakBisim.step*) (*auto simp add*: *rtrancl-def*)

lemma *weakBisimReflexive*:
  fixes Ψ :: $'b$
  and P :: ($'a$, $'b$, $'c$) *psi*

  shows $\Psi \rhd P \approx P$
proof −
  let *?X* = $\{(\Psi, P, P) \mid \Psi\ P.\ True\}$
  have $(\Psi, P, P) \in$ *?X* **by** *simp*
  thus *?thesis*
    by(*coinduct rule*: *weakBisimWeakCoinduct*, *auto intro*: *weakSimReflexive weakStatImpReflexive*)
qed

lemma *weakBisimClosed*:
  fixes Ψ :: $'b$
  and P :: ($'a$, $'b$, $'c$) *psi*
  and Q :: ($'a$, $'b$, $'c$) *psi*
  and p :: *name prm*

  assumes $\Psi \rhd P \approx Q$

  shows $(p \cdot \Psi) \rhd (p \cdot P) \approx (p \cdot Q)$
proof −
  let *?X* = $\{(p \cdot \Psi, p \cdot P, p \cdot Q) \mid (p{::}name\ prm)\ \Psi\ P\ Q.\ \Psi \rhd P \approx Q\}$
  have *eqvt ?X*
    apply(*auto simp add*: *eqvt-def*)
    apply(*rule-tac x=pa@p* **in** *exI*)
    by(*auto simp add*: *pt2*[*OF pt-name-inst*])
  from ‹$\Psi \rhd P \approx Q$› **have** $(p \cdot \Psi, p \cdot P, p \cdot Q) \in$ *?X* **by** *blast*
  thus *?thesis*
  proof(*coinduct rule*: *weakBisimWeakCoinduct*)

```
    case(cStatImp Ψ P Q)
    {
      fix Ψ P Q p
      assume Ψ ⊳ P ≈ (Q::('a, 'b, 'c) psi)
      hence Ψ ⊳ P ≲≈<weakBisim> Q by(rule weakBisimE)
      hence Ψ ⊳ P ≲≈<?X> Q
        apply(rule-tac A=weakBisim in weakStatImpMonotonic, auto)
        by(rule-tac x=[]::name prm in exI) auto
      with ‹eqvt ?X› have ((p::name prm) • Ψ) ⊳ (p • P) ≲≈<?X> (p • Q)
        by(rule weakStatImpClosed)
    }
    with ‹(Ψ, P, Q) ∈ ?X› show ?case by blast
  next
    case(cSim Ψ P Q)
    {
      fix p :: name prm
      fix Ψ P Q
      assume Ψ ⊳ P ⇝<weakBisim> Q
      hence Ψ ⊳ P ⇝<?X> Q
        apply(rule-tac A=weakBisim in weakSimMonotonic, auto)
        by(rule-tac x=[]::name prm in exI) auto
      with ‹eqvt ?X› have ((p::name prm) • Ψ) ⊳ (p • P) ⇝<?X> (p • Q)
        by(rule-tac weakSimClosed)
    }
    with ‹(Ψ, P, Q) ∈ ?X› show ?case
      by(blast dest: weakBisimE)
  next
    case(cExt Ψ P Q Ψ')
    {
      fix p :: name prm
      fix Ψ P Q Ψ'
      assume ∀Ψ'. (Ψ ⊗ Ψ', P, Q) ∈ weakBisim
      hence ((p • Ψ) ⊗ Ψ', p • P, p • Q) ∈ ?X
        apply(auto, rule-tac x=p in exI)
        apply(rule-tac x=Ψ ⊗ (rev p • Ψ') in exI)
        by(auto simp add: eqvts)
    }
    with ‹(Ψ, P, Q) ∈ ?X› show ?case
      by(blast dest: weakBisimE)
  next
    case(cSym Ψ P Q)
    thus ?case
      by(blast dest: weakBisimE)
  qed
qed

lemma weakBisimEqvt[simp]:
  shows eqvt weakBisim
by(auto simp add: eqvt-def weakBisimClosed)
```

lemma *statEqWeakBisim*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*
  and Ψ' :: $'b$

  assumes $\Psi \rhd P \approx Q$
  and $\Psi \simeq \Psi'$

  shows $\Psi' \rhd P \approx Q$
proof −
  let $?X = \{(\Psi',\ P,\ Q) \mid \Psi\ P\ Q\ \Psi'.\ \Psi \rhd P \approx Q \wedge \Psi \simeq \Psi'\}$
  from ‹$\Psi \rhd P \approx Q$› ‹$\Psi \simeq \Psi'$› **have** $(\Psi',\ P,\ Q) \in ?X$ **by** *auto*
  thus *?thesis*
  proof(*coinduct rule*: *weakBisimCoinduct*)
    case(*cStatImp* $\Psi'\ P\ Q$)
    from ‹$(\Psi',\ P,\ Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \approx Q$ **and** $\Psi \simeq \Psi'$
      by *auto*
    from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \rhd P \lessapprox$<*weakBisim*> Q **by**(*rule weakBisimE*)
    moreover note ‹$\Psi \simeq \Psi'$›
    moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ [\![\Psi \rhd P \approx Q;\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi',\ P,\ Q) \in ?X \cup$
weakBisim
      by *auto*
    ultimately show *?case* **by**(*rule weakStatImpStatEq*)
  next
    case(*cSim* $\Psi'\ P\ Q$)
    from ‹$(\Psi',\ P,\ Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \approx Q$ **and** $\Psi \simeq \Psi'$
      by *auto*
    from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \rhd P \rightsquigarrow$<*weakBisim*> Q **by**(*blast dest*: *weakBisimE*)
    moreover have *eqvt ?X*
      by(*auto simp add*: *eqvt-def*) (*metis weakBisimClosed AssertionStatEqClosed*)
    hence *eqvt*($?X \cup$ *weakBisim*) **by** *auto*
    moreover note ‹$\Psi \simeq \Psi'$›
    moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ [\![\Psi \rhd P \approx Q;\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi',\ P,\ Q) \in ?X \cup$
weakBisim
      by *auto*
    ultimately show *?case* **by**(*rule weakSimStatEq*)
  next
    case(*cExt* $\Psi'\ P\ Q\ \Psi''$)
    from ‹$(\Psi',\ P,\ Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \approx Q$ **and** $\Psi \simeq \Psi'$
      by *auto*
    from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \otimes \Psi'' \rhd P \approx Q$ **by**(*rule weakBisimE*)
    moreover from ‹$\Psi \simeq \Psi'$› **have** $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$ **by**(*rule Composition*)
    ultimately show *?case* **by** *blast*
  next
    case(*cSym* $\Psi'\ P\ Q$)
    from ‹$(\Psi',\ P,\ Q) \in ?X$› **obtain** Ψ **where** $\Psi \rhd P \approx Q$ **and** $\Psi \simeq \Psi'$
      by *auto*

```
    from ‹Ψ ▷ P ≈ Q› have Ψ ▷ Q ≈ P by(rule weakBisimE)
    thus ?case using ‹Ψ ≃ Ψ'› by auto
  qed
qed

lemma weakBisimTransitive:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   R :: ('a, 'b, 'c) psi

  assumes PQ: Ψ ▷ P ≈ Q
  and     QR: Ψ ▷ Q ≈ R

  shows Ψ ▷ P ≈ R
proof −
  let ?X = {(Ψ, P, R) | Ψ P R. ∃ Q. Ψ ▷ P ≈ Q ∧ Ψ ▷ Q ≈ R}
  from PQ QR have (Ψ, P, R) ∈ ?X by auto
  thus ?thesis
  proof(coinduct rule: weakBisimCoinduct)
    case(cStatImp Ψ P R)
    from ‹(Ψ, P, R) ∈ ?X› obtain Q where Ψ ▷ P ≈ Q and  Ψ ▷ Q ≈ R by
blast
    from ‹Ψ ▷ P ≈ Q› have Ψ ▷ P ≲<weakBisim> Q by(rule weakBisimE)
    moreover note ‹Ψ ▷ Q ≈ R›
    moreover have ?X ⊆ ?X ∪ weakBisim by auto
    moreover note weakBisimE(1)
   moreover from weakBisimE(2) have ⋀Ψ P Q P'. ⟦Ψ ▷ P ≈ Q; Ψ ▷ P ⟹^τ
P'⟧ ⟹ ∃ Q'. Ψ ▷ Q ⟹^τ Q' ∧ Ψ ▷ P' ≈ Q'
      by(metis weakBisimE(4) weakSimTauChain)
    ultimately show ?case by(rule weakStatImpTransitive)
  next
    case(cSim Ψ P R)
    {
      fix Ψ P Q R
      assume Ψ ▷ P ≈ Q and Ψ ▷ Q ⇝<weakBisim> R
      moreover have eqvt ?X
        by(force simp add: eqvt-def dest: weakBisimClosed)
      with weakBisimEqvt have eqvt (?X ∪ weakBisim) by blast
      moreover have ?X ⊆ ?X ∪ weakBisim by auto
      moreover note weakBisimE(2)
      ultimately have Ψ ▷ P ⇝<(?X ∪ weakBisim)> R
        by(rule-tac weakSimTransitive) auto
    }
    with ‹(Ψ, P, R) ∈ ?X› show ?case
      by(blast dest: weakBisimE)
  next
    case(cExt Ψ P R Ψ')
    thus ?case by(blast dest: weakBisimE)
```

```
  next
    case(cSym Ψ P R)
    thus ?case by(blast dest: weakBisimE)
  qed
qed

lemma strongBisimWeakBisim:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi

  assumes Ψ ▷ P ∼ Q

  shows Ψ ▷ P ≈ Q
proof −
  from ‹Ψ ▷ P ∼ Q›
  show ?thesis
  proof(coinduct rule: weakBisimWeakCoinduct)
    case(cStatImp Ψ P Q)
    from ‹Ψ ▷ P ∼ Q› have insertAssertion(extractFrame P) Ψ ↪F insertAsser-
tion(extractFrame Q) Ψ
      by(metis bisimE FrameStatEq-def)
    moreover from ‹Ψ ▷ P ∼ Q› have ⋀Ψ'. Ψ ⊗ Ψ' ▷ P ∼ Q by(rule bisimE)
    ultimately show ?case by(rule statImpWeakStatImp)
  next
    case(cSim Ψ P Q)
    note ‹Ψ ▷ P ∼ Q›
    moreover have ⋀Ψ P Q. Ψ ▷ P ∼ Q ⟹ insertAssertion(extractFrame Q)
Ψ ↪F insertAssertion(extractFrame P) Ψ
      by(drule-tac bisimE) (simp add: FrameStatEq-def)
    ultimately show ?case using bisimE(2) bisimE(3)
      by(rule strongSimWeakSim)
  next
    case(cExt Ψ P Q Ψ')
    from ‹Ψ ▷ P ∼ Q› show ?case
      by(rule bisimE)
  next
    case(cSym Ψ P Q)
    from ‹Ψ ▷ P ∼ Q› show ?case
      by(rule bisimE)
  qed
qed

lemma structCongWeakBisim:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi

  assumes P ≡s Q
```

shows $P \approx Q$
using *assms*
by(*metis structCongBisim strongBisimWeakBisim*)

lemma *simTauChain*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and Q' :: $('a, 'b, 'c)$ *psi*

assumes $(\Psi, P, Q) \in Rel$
and $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'$
and Sim: $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in Rel \Longrightarrow \Psi \rhd P \rightsquigarrow[Rel]\ Q$

obtains P' **where** $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'$ **and** $(\Psi, P', Q') \in Rel$
proof −
assume A: $\bigwedge P'.\ [\![\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'; (\Psi, P', Q') \in Rel]\!] \Longrightarrow thesis$
from ‹$\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'$› ‹$(\Psi, P, Q) \in Rel$› A **show** *?thesis*
proof(*induct arbitrary*: P *thesis rule*: *tauChainInduct*)
case(*TauBase* Q P)
moreover have $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P$ **by** *simp*
ultimately show *?case* **by** *blast*
next
case(*TauStep* Q Q' Q'' P)
from ‹$(\Psi, P, Q) \in Rel$› **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'$ **and** $(\Psi, P', Q') \in Rel$
by(*rule TauStep*)
from ‹$(\Psi, P', Q') \in Rel$› **have** $\Psi \rhd P' \rightsquigarrow[Rel]\ Q'$ **by**(*rule Sim*)
then obtain P'' **where** *P'Chain*: $\Psi \rhd P' \longmapsto\tau \prec P''$ **and** $(\Psi, P'', Q'') \in Rel$
using ‹$\Psi \rhd Q' \longmapsto\tau \prec Q''$› **by**(*drule-tac simE*) *auto*
from *PChain P'Chain* **have** $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P''$ **by**(*auto dest*: *tauActTauChain*)
thus *?case* **using** ‹$(\Psi, P'', Q'') \in Rel$› **by**(*rule TauStep*)
qed
qed

lemma *quietBisimNil*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

assumes *quiet* P

shows $\Psi \rhd P \approx \mathbf{0}$
proof −
let $?X = \{(\Psi, \mathbf{0}, P) \mid \Psi\ P.\ quiet\ P\} \cup \{(\Psi, P, \mathbf{0}) \mid \Psi\ P.\ quiet\ P\}$

from ‹*quiet* P› **have** $(\Psi, P, \mathbf{0}) \in ?X$ **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *weakBisimWeakCoinduct*)

```
    case(cStatImp Ψ P Q)
    thus ?case
      apply(simp add: weakStatImp-def)
      apply(rule allI)
      apply(rule-tac x=Q in exI)
      apply auto
      apply(drule-tac Ψ=Ψ in quietFrame)
      apply(rule-tac G=⟨ε, Ψ⟩ in FrameStatImpTrans)
      using Identity
      apply(simp add: AssertionStatEq-def)
      apply(simp add: FrameStatEq-def)
      apply(drule-tac Ψ=Ψ in quietFrame)
      apply(rule-tac G=⟨ε, Ψ⟩ in FrameStatImpTrans)
      apply auto
      defer
      using Identity
      apply(simp add: AssertionStatEq-def)
      apply(simp add: FrameStatEq-def)
      done
  next
    case(cSim Ψ P Q)
    moreover have eqvt ?X by(auto simp add: eqvt-def intro: quietEqvt)
    ultimately show ?case
      apply auto
      apply(rule quietSim)
      apply auto
      apply(auto simp add: weakSimulation-def)
      done
  next
    case(cExt Ψ P Q Ψ')
    thus ?case by blast
  next
    case(cSym Ψ P Q)
    thus ?case by blast
  qed
qed

lemma weakTransitiveWeakCoinduct[case-names cStatImp cSim cExt cSym, case-conclusion
bisim step, consumes 2]:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   X :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

  assumes p: (Ψ, P, Q) ∈ X
  and Eqvt: eqvt X
  and rStatImp: ⋀Ψ P Q. (Ψ, P, Q) ∈ X ⟹ Ψ ▷ P ≲<X> Q
  and rSim: ⋀Ψ P Q. (Ψ, P, Q) ∈ X ⟹ Ψ ▷ P ⇝<({(Ψ, P, Q) | Ψ P Q. ∃P'
Q'. Ψ ▷ P ∼ P' ∧
```

$(\Psi,\ P',\ Q') \in X \wedge$
$\Psi \rhd Q' \sim Q\})> Q$
and *rExt*: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X$
and *rSym*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi,\ Q,\ P) \in X$

shows $\Psi \rhd P \approx Q$
proof −
let $?X = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \sim Q\}$
from *p* **have** $(\Psi,\ P,\ Q) \in ?X$
by(*blast intro*: *bisimReflexive*)
thus *?thesis*
proof(*coinduct rule*: *weakBisimWeakCoinduct*)
case(*cStatImp* $\Psi\ P\ Q$)
{
fix Ψ'
from ‹$(\Psi\ ,\ P,\ Q) \in ?X$› **obtain** $P'\ Q'$ **where** $\Psi \rhd P \sim P'$ **and** $(\Psi,\ P',\ Q') \in X$ **and** $\Psi \rhd Q' \sim Q$ **by** *auto*
from ‹$(\Psi,\ P',\ Q') \in X$› **obtain** $Q''\ Q'''$ **where** *Q'Chain*: $\Psi \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$
and *PImpQ*: *insertAssertion* (*extractFrame* P') $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'') Ψ
and *Q''Chain*: $\Psi \otimes \Psi' \rhd Q'' \Longrightarrow\hat{}_\tau\ Q'''$ **and** $(\Psi \otimes \Psi',\ P',\ Q''') \in X$
apply(*drule-tac rStatImp*) **by**(*auto simp add*: *weakStatImp-def*) *blast*
from ‹$\Psi \rhd Q' \sim Q$› **have** $\Psi \rhd Q \sim Q'$ **by**(*rule bisimE*)
then obtain Q'''' **where** $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q''''$ **and** $\Psi \rhd Q'''' \sim Q''$ **using** ‹$\Psi \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$› *bisimE*(*2*)
by(*rule simTauChain*)
note ‹$\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q''''$›
moreover have *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'''') Ψ
proof −
from ‹$\Psi \rhd P \sim P'$› **have** *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P') Ψ
by(*drule-tac bisimE*) (*simp add*: *FrameStatEq-def*)
moreover from ‹$\Psi \rhd Q'''' \sim Q''$› **have** *insertAssertion* (*extractFrame* Q'') $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'''') Ψ
by(*drule-tac bisimE*) (*simp add*: *FrameStatEq-def*)
ultimately show *?thesis* **using** *PImpQ* **by**(*blast intro*: *FrameStatImpTrans*)
qed
moreover from ‹$\Psi \rhd Q'''' \sim Q''$› **have** $\Psi \otimes \Psi' \rhd Q'''' \sim Q''$ **by**(*metis bisimE*)
then obtain Q''''' **where** *Q''''Chain*: $\Psi \otimes \Psi' \rhd Q'''' \Longrightarrow\hat{}_\tau\ Q'''''$ **and** $\Psi \otimes \Psi' \rhd Q''''' \sim Q'''$ **using** *Q''Chain bisimE*(*2*)
by(*rule simTauChain*)
moreover from ‹$\Psi \rhd P \sim P'$› ‹$(\Psi \otimes \Psi'\ ,\ P',\ Q''') \in X$› ‹$\Psi \otimes \Psi' \rhd Q''''' \sim Q'''$› **have** $(\Psi \otimes \Psi',\ P,\ Q''''') \in ?X$ **by**(*auto dest*: *bisimE*)
ultimately have $\exists Q'\ Q''.\ \Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'' \wedge$ *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'') $\Psi \wedge \Psi \otimes \Psi' \rhd Q'' \Longrightarrow\hat{}_\tau\ Q' \wedge (\Psi$

$\otimes$ Ψ', P, Q') $\in$ *?X* **by** *blast*
}
with ‹(Ψ, P, Q) $\in$ *?X*› **show** *?case* **by**(*simp add*: *weakStatImp-def*) *blast*
next
case(*cSim* Ψ P Q)
from ‹(Ψ, P, Q) $\in$ *?X*› **obtain** P' Q' **where** $\Psi \rhd P \sim P'$ **and** (Ψ, P', Q') $\in$ X **and** $\Psi \rhd Q' \sim Q$ **by** *auto*
from ‹(Ψ, P', Q') $\in$ X› **have** $\Psi \rhd P' \rightsquigarrow$<*?X*> Q'
by(*rule rSim*)
moreover from ‹$\Psi \rhd Q' \sim Q$› **have** $\Psi \rhd Q' \rightsquigarrow$[*bisim*] Q **by**(*rule bisimE*)
moreover from ‹*eqvt X*› **have** *eqvt ?X*
apply(*auto simp add*: *eqvt-def*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*rule-tac x=p* $\cdot$ P' **in** *exI*, *simp*)
by(*rule-tac x=p* $\cdot$ Q' **in** *exI*, *auto*)
moreover from ‹$\Psi \rhd Q' \sim Q$› **have** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q') Ψ
by(*drule-tac bisimE*) (*simp add*: *FrameStatEq-def*)
moreover have $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi, P, Q) \in ?X \wedge \Psi \rhd Q \sim R\} \subseteq$ *?X*
by(*blast intro*: *bisimTransitive*)
moreover note *bisimE*(*3*)
ultimately have $\Psi \rhd P' \rightsquigarrow$<*?X*> Q **by**(*rule strongAppend*)
moreover have $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ \Psi \rhd P \sim Q \wedge (\Psi, Q, R) \in ?X\} \subseteq$ *?X*
by(*blast intro*: *bisimTransitive*)
moreover {
fix Ψ P Q

assume $\Psi \rhd P \sim Q$
moreover have $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow$ *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P) Ψ
by(*drule-tac bisimE*) (*simp add*: *FrameStatEq-def*)
ultimately have $\Psi \rhd P \rightsquigarrow$<*bisim*> Q **using** *bisimE*(*2*) *bisimE*(*3*)
by(*rule strongSimWeakSim*)
}
ultimately show *?case* **using** ‹$\Psi \rhd P \sim P'$› ‹*eqvt ?X*›
by(*rule-tac weakSimTransitive*)
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*blast dest*: *bisimE intro*: *rExt*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *bisimE intro*: *rSym*)
qed
qed

lemma *weakTransitiveCoinduct*[*case-names cStatImp cSim cExt cSym*, *case-conclusion*

bisim step, *consumes 2*]:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *p*: $(\Psi,\ P,\ Q) \in X$
and *Eqvt*: *eqvt X*
and *rStatImp*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \lessapprox <(X \cup weakBisim)> Q$
and *rSim*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow <(\{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge$
$(\Psi,\ P',\ Q') \in (X \cup weakBisim) \wedge$
$\Psi \rhd Q' \sim Q\})> Q$
and *rExt*: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X \cup weakBisim$
and *rSym*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi,\ Q,\ P) \in X \cup weakBisim$

shows $\Psi \rhd P \approx Q$
proof −
from *p* **have** $(\Psi,\ P,\ Q) \in X \cup weakBisim$ **by** *auto*
moreover from ‹*eqvt X*› **have** *eqvt*$(X \cup weakBisim)$ **by** *auto*
ultimately show *?thesis*
proof(*coinduct rule*: *weakTransitiveWeakCoinduct*)
case(*cStatImp* $\Psi\ P\ Q$)
thus *?case* **by**(*blast dest*: *rStatImp weakBisimE*(*1*) *weakStatImpMonotonic*)
next
case(*cSim* $\Psi\ P\ Q$)
thus *?case* **by**(*fastforce intro*: *rSim weakBisimE*(*2*) *weakSimMonotonic bisimReflexive*)
next
case(*cExt* $\Psi\ P\ Q\ \Psi'$)
thus *?case* **by**(*blast dest*: *weakBisimE rExt*)
next
case(*cSym* $\Psi\ P\ Q$)
thus *?case* **by**(*blast dest*: *weakBisimE rSym*)
qed
qed

lemma *weakTransitiveCoinduct2*[*case-names cStatImp cSim cExt cSym*, *case-conclusion bisim step*, *consumes 2*]:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *p*: $(\Psi,\ P,\ Q) \in X$
and *Eqvt*: *eqvt X*
and *rStatImp*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \lessapprox <X> Q$
and *rSim*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow <(\{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'$

Q'. $\Psi \rhd P \approx P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \sim Q\}) > Q$
and *rExt*: $\bigwedge \Psi\ P\ Q\ \Psi'$. $(\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X$
and *rSym*: $\bigwedge \Psi\ P\ Q$. $(\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi,\ Q,\ P) \in X$

shows $\Psi \rhd P \approx Q$
proof −
let $?X = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \approx P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \sim Q\}$
let $?Y = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \approx P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \approx Q\}$

from ‹*eqvt X*› **have** *eqvt ?X*
apply(*auto simp add*: *eqvt-def*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*drule-tac p=p* **in** *weakBisimClosed*)
apply(*rule-tac x=p* · P' **in** *exI*, *simp*)
by(*rule-tac x=p* · Q' **in** *exI*, *auto*)
from ‹*eqvt X*› **have** *eqvt ?Y*
apply(*auto simp add*: *eqvt-def*)
apply(*drule-tac p=p* **in** *weakBisimClosed*)
apply(*drule-tac p=p* **in** *weakBisimClosed*)
apply(*rule-tac x=p* · P' **in** *exI*, *simp*)
by(*rule-tac x=p* · Q' **in** *exI*, *auto*)

{
fix $\Psi\ P\ Q$
assume $(\Psi,\ P,\ Q) \in ?X$
then obtain $P'\ Q'$ **where** $\Psi \rhd P \approx P'$ **and** $(\Psi,\ P',\ Q') \in X$ **and** $\Psi \rhd Q' \sim Q$
by *auto*
note ‹$\Psi \rhd P \approx P'$›
moreover from ‹$(\Psi,\ P',\ Q') \in X$› **have** $\Psi \rhd P' \rightsquigarrow <?X> Q'$ **by**(*rule rSim*)
moreover note ‹*eqvt ?X*›
moreover have $\{(\Psi,\ P,\ R) \mid \Psi\ P\ R.\ \exists Q.\ \Psi \rhd P \approx Q \wedge (\Psi,\ Q,\ R) \in ?X\} \subseteq ?X$
by(*blast intro*: *weakBisimTransitive*)
ultimately have $\Psi \rhd P \rightsquigarrow <?X> Q'$ **using** *weakBisimE*(*2*) **by**(*rule weakSimTransitive*)
moreover from ‹$\Psi \rhd Q' \sim Q$› **have** $\Psi \rhd Q' \rightsquigarrow [bisim]\ Q$ **by**(*rule bisimE*)
moreover note ‹*eqvt ?X*›
moreover from ‹$\Psi \rhd Q' \sim Q$› **have** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q') Ψ
by(*drule-tac bisimE*) (*simp add*: *FrameStatEq-def*)
moreover have $\{(\Psi,\ P,\ R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi,\ P,\ Q) \in ?X \wedge \Psi \rhd Q \sim R\} \subseteq ?X$
by(*blast dest*: *bisimTransitive*)
moreover note *bisimE*(*3*)
ultimately have $\Psi \rhd P \rightsquigarrow <?X> Q$ **by**(*rule-tac strongAppend*)
}

note *Goal* = *this*

from *p* **have** $(\Psi, P, Q) \in ?Y$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive rSym*)
thus *?thesis*
proof(*coinduct rule*: *weakBisimWeakCoinduct*)
next
case(*cStatImp* Ψ P Q)
{
fix Ψ'

from ‹$(\Psi, P, Q) \in ?Y$› **obtain** R S **where** $\Psi \rhd P \approx R$ **and** $(\Psi, R, S) \in X$ **and** $\Psi \rhd S \approx Q$ **by** *auto*
from ‹$\Psi \rhd P \approx R$› **obtain** R'' R'
where *RChain*: $\Psi \rhd R \Longrightarrow\hat{}_\tau R''$
and *PImpR''*: *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* R'') Ψ
and *R''Chain*: $\Psi \otimes \Psi' \rhd R'' \Longrightarrow\hat{}_\tau R'$
and $\Psi \otimes \Psi' \rhd P \approx R'$
apply(*drule-tac weakBisimE*) **by**(*simp add*: *weakStatImp-def*) *blast*

from ‹$(\Psi, R, S) \in X$› **have** $(\Psi, S, R) \in ?X$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive rSym*)
with *RChain* **obtain** S'' **where** *SChain*: $\Psi \rhd S \Longrightarrow\hat{}_\tau S''$ **and** $(\Psi, S'', R'') \in ?X$ **using** *Goal*
by(*rule weakSimTauChain*)

from ‹$(\Psi, S'', R'') \in ?X$› **obtain** T U **where** $\Psi \rhd S'' \approx T$ **and** $(\Psi, T, U) \in X$ **and** $\Psi \rhd U \sim R''$
by *auto*
from ‹$\Psi \rhd U \sim R''$› **have** *R''ImpU*: *insertAssertion* (*extractFrame* R'') $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* U) Ψ
by(*drule-tac bisimE*) (*simp add*: *FrameStatEq-def*)

from ‹$(\Psi, T, U) \in X$› *weakStatImp-def*
obtain T'' T' **where** *TChain*: $\Psi \rhd T \Longrightarrow\hat{}_\tau T''$
and *UImpT''*: *insertAssertion* (*extractFrame* U) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* T'') Ψ
and *T''Chain*: $\Psi \otimes \Psi' \rhd T'' \Longrightarrow\hat{}_\tau T'$
and $(\Psi \otimes \Psi', U, T') \in X$
by(*blast dest*: *rStatImp rSym*)

from *TChain* ‹$\Psi \rhd S'' \approx T$› *weakBisimE*(*2*) **obtain** S''' **where** *S''Chain*: $\Psi \rhd S'' \Longrightarrow\hat{}_\tau S'''$ **and** $\Psi \rhd S''' \approx T''$
by(*rule weakSimTauChain*)

from ‹$\Psi \rhd S''' \approx T''$› *weakStatImp-def*
obtain S'''''' S'''' **where** *S'''Chain*: $\Psi \rhd S''' \Longrightarrow\hat{}_\tau S''''''$
and *T''ImpS'''''*: *insertAssertion* (*extractFrame* T'') $\Psi \hookrightarrow_F$

insertAssertion (*extractFrame* $S''''')$ Ψ
and $S'''''Chain$: $\Psi \otimes \Psi' \rhd S''''' \Longrightarrow\hat{}_\tau S''''$
and $\Psi \otimes \Psi' \rhd T'' \approx S''''$
by(*metis weakBisimE*)

from *SChain S''Chain S'''Chain* **have** $\Psi \rhd S \Longrightarrow\hat{}_\tau S'''''$ **by** *auto*
moreover from ‹$\Psi \rhd S \approx Q$› **have** $\Psi \rhd Q \approx S$ **by**(*rule weakBisimE*)
ultimately obtain Q''' **where** $QChain$: $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'''$ **and** $\Psi \rhd Q''' \approx S'''''$ **using** *weakBisimE*(*2*)
by(*rule weakSimTauChain*)
from ‹$\Psi \rhd Q''' \approx S'''''$› **have** $\Psi \rhd S''''' \approx Q'''$ **by**(*rule weakBisimE*)
then obtain Q'' Q' **where** $Q'''Chain$: $\Psi \rhd Q''' \Longrightarrow\hat{}_\tau Q''$
and $S'''''ImpQ''$: *insertAssertion* (*extractFrame* $S''''')$ $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* $Q'')$ Ψ
and $Q''Chain$: $\Psi \otimes \Psi' \rhd Q'' \Longrightarrow\hat{}_\tau Q'$
and $\Psi \otimes \Psi' \rhd S''''' \approx Q'$ **using** *weakStatImp-def*
by(*metis weakBisimE*)

from *QChain Q'''Chain* **have** $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q''$ **by** *auto*
moreover from *PImpR'' R''ImpU UImpT'' T''ImpS''''' S'''''ImpQ''*
have *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* $Q'')$ Ψ
by(*blast dest*: *FrameStatImpTrans*)

moreover from ‹$\Psi \rhd U \sim R''$› **have** $\Psi \otimes \Psi' \rhd U \approx R''$ **by**(*metis weakBisimE strongBisimWeakBisim*)
with $R''Chain$ **obtain** U' **where** $UChain$: $\Psi \otimes \Psi' \rhd U \Longrightarrow\hat{}_\tau U'$ **and** $\Psi \otimes \Psi' \rhd U' \approx R'$ **using** *weakBisimE*(*2*)
by(*rule weakSimTauChain*)
from ‹$\Psi \otimes \Psi' \rhd U' \approx R'$› **have** $\Psi \otimes \Psi' \rhd R' \approx U'$ **by**(*rule weakBisimE*)
from ‹$(\Psi \otimes \Psi', U, T') \in X$› **have** $(\Psi \otimes \Psi', T', U) \in ?X$ **by**(*blast intro*: *rSym weakBisimReflexive bisimReflexive*)
with *UChain* **obtain** T''' **where** $T'Chain$: $\Psi \otimes \Psi' \rhd T' \Longrightarrow\hat{}_\tau T'''$ **and** $(\Psi \otimes \Psi', T''', U') \in ?X$ **using** *Goal*
by(*rule weakSimTauChain*)
from ‹$(\Psi \otimes \Psi', T''', U') \in ?X$› **have** $(\Psi \otimes \Psi', U', T''') \in ?Y$
by(*blast dest*: *weakBisimE rSym strongBisimWeakBisim*)
from *T''Chain T'Chain* **have** $\Psi \otimes \Psi' \rhd T'' \Longrightarrow\hat{}_\tau T'''$ **by** *auto*
then obtain S'''''' **where** $S''''Chain$: $\Psi \otimes \Psi' \rhd S'''' \Longrightarrow\hat{}_\tau S''''''$ **and** $\Psi \otimes \Psi' \rhd T''' \approx S''''''$
using ‹$\Psi \otimes \Psi' \rhd T'' \approx S''''$› *weakBisimE*(*2*)
apply(*drule-tac weakBisimE*(*4*))
by(*rule weakSimTauChain*) (*auto dest*: *weakBisimE*(*4*))
from $S'''''Chain$ $S''''Chain$ **have** $\Psi \otimes \Psi' \rhd S''''' \Longrightarrow\hat{}_\tau S''''''$ **by** *auto*

with ‹$\Psi \otimes \Psi' \rhd S''''' \approx Q'$›
obtain Q'''' **where** $Q'Chain$: $\Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau Q''''$ **and** $\Psi \otimes \Psi' \rhd S'''''' \approx Q''''$ **using** *weakBisimE*(*2*)
apply(*drule-tac weakBisimE*(*4*))

by(*rule weakSimTauChain*) (*auto dest*: *weakBisimE*(*4*))

from *Q''Chain Q'Chain* **have** $\Psi \otimes \Psi' \rhd Q'' \Longrightarrow\hat{}_\tau Q''''$ **by** *auto*
moreover from ‹$\Psi \otimes \Psi' \rhd P \approx R'$› ‹$\Psi \otimes \Psi' \rhd R' \approx U'$› ‹$(\Psi \otimes \Psi', U', T''') \in ?Y$› ‹$\Psi \otimes \Psi' \rhd T''' \approx S''''''$›
‹$\Psi \otimes \Psi' \rhd S'''''' \approx Q''''$›
have $(\Psi \otimes \Psi', P, Q'''') \in ?Y$
by *auto* (*blast dest*: *weakBisimTransitive*)
ultimately have $\exists Q'' Q'. \Psi \rhd Q \Longrightarrow\hat{}_\tau Q'' \wedge$ *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'') $\Psi \wedge \Psi \otimes \Psi' \rhd Q'' \Longrightarrow\hat{}_\tau Q' \wedge (\Psi \otimes \Psi', P, Q') \in ?Y$
by *blast*
}
thus *?case* **by**(*simp add*: *weakStatImp-def*)
next
case(*cSim* Ψ P Q)
moreover {
fix Ψ P P' Q' Q
assume $\Psi \rhd P \approx P'$ **and** $(\Psi, P', Q') \in X$ **and** $\Psi \rhd Q' \approx Q$
from ‹$(\Psi, P', Q') \in X$› **have** $(\Psi, P', Q') \in ?X$
by(*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover from ‹$\Psi \rhd Q' \approx Q$› **have** $\Psi \rhd Q' \rightsquigarrow$<*weakBisim*> Q **by**(*rule weakBisimE*)
moreover note ‹*eqvt ?Y*›
moreover have $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi, P, Q) \in ?X \wedge \Psi \rhd Q \approx R\} \subseteq ?Y$
by(*blast dest*: *weakBisimTransitive strongBisimWeakBisim*)
ultimately have $\Psi \rhd P' \rightsquigarrow$<*?Y*> Q **using** *Goal*
by(*rule weakSimTransitive*)
note ‹$\Psi \rhd P \approx P'$› *this* ‹*eqvt ?Y*›
moreover have $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ \Psi \rhd P \approx Q \wedge (\Psi, Q, R) \in ?Y\} \subseteq ?Y$
by(*blast dest*: *weakBisimTransitive*)
ultimately have $\Psi \rhd P \rightsquigarrow$<*?Y*> Q **using** *weakBisimE*(*2*)
by(*rule weakSimTransitive*)
}
ultimately show *?case* **by** *auto*
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*blast dest*: *bisimE weakBisimE intro*: *rExt*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *bisimE*(*4*) *weakBisimE*(*4*) *rSym*)
qed
qed

end

end

theory *Weak-Sim-Pres*
imports *Sim-Pres Weak-Simulation Weak-Stat-Imp*
begin

context *env* **begin**

lemma *weakInputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$

assumes *PRelQ*: $\bigwedge$*Tvec* Ψ'. *length xvec* = *length Tvec* $\Longrightarrow$ $(\Psi \otimes \Psi', P[xvec::=Tvec], Q[xvec::=Tvec]) \in Rel$

shows $\Psi \rhd M(\!|\lambda*xvec\ N|\!).P \rightsquigarrow$<*Rel*> $M(\!|\lambda*xvec\ N|\!).Q$
proof(*induct rule*: *weakSimI2*)
case(*cAct* Ψ' α Q')
from ‹$\Psi \rhd M(\!|\lambda*xvec\ N|\!).Q \longmapsto\alpha \prec Q'$› **show** *?case*
proof(*induct rule*: *inputCases*)
case(*cInput K Tvec*)
from ‹$\Psi \vdash M \leftrightarrow K$› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*› ‹*distinct xvec*›
have Ψ : $(M(\!|\lambda*xvec\ N|\!).Q) \rhd M(\!|\lambda*xvec\ N|\!).P \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec (P[xvec::=Tvec])$
by(*rule-tac weakInput*) *auto*
moreover have $\Psi \otimes \Psi' \rhd P[xvec::=Tvec] \Longrightarrow\hat{}_{\tau}\ P[xvec::=Tvec]$ **by** *simp*
moreover from ‹*length xvec* = *length Tvec*› **have** $(\Psi \otimes \Psi', P[xvec::=Tvec], Q[xvec::=Tvec]) \in Rel$
by(*rule PRelQ*)
ultimately show *?case* **by** *blast*
qed
next
case(*cTau* Q')
from ‹$\Psi \rhd M(\!|\lambda*xvec\ N|\!).Q \longmapsto\tau \prec Q'$› **have** *False* **by** *auto*
thus *?case* **by** *simp*
qed

lemma *weakOutputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$

assumes *PRelQ*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in Rel$

shows $\Psi \rhd M\langle N\rangle.P \rightsquigarrow\!<Rel>\ M\langle N\rangle.Q$
proof(*induct rule*: *weakSimI2*)
case(*cAct* $\Psi'\ \alpha\ Q'$)
from ‹$\Psi \rhd M\langle N\rangle.Q \longmapsto\alpha \prec Q'$› **show** *?case*
proof(*induct rule*: *outputCases*)
case(*cOutput* K)
from ‹$\Psi \vdash M \leftrightarrow K$›
have $\Psi : (M\langle N\rangle.Q) \rhd M\langle N\rangle.P \Longrightarrow K\langle N\rangle \prec P$ **by**(*rule weakOutput*) *auto*
moreover have $\Psi \otimes \Psi' \rhd P \Longrightarrow\hat{}_{\tau}\ \ P$ **by** *simp*
moreover have $(\Psi \otimes \Psi',\ \ P,\ Q) \in Rel$ **by**(*rule PRelQ*)
ultimately show *?case* **by** *blast*
qed
next
case(*cTau* Q')
from ‹$\Psi \rhd M\langle N\rangle.Q \longmapsto\tau \prec Q'$› **have** *False* **by** *auto*
thus *?case* **by** *simp*
qed

lemma *resTauCases*[*consumes 3*, *case-names cRes*]:
fixes Ψ :: $'b$
and x :: *name*
and P :: $('a,\ 'b,\ 'c)$ *psi*
and P' :: $('a,\ 'b,\ 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd (\!|\nu x|\!)P \longmapsto\tau \prec P'$
and $x \sharp \Psi$
and $x \sharp P'$
and *rScope*: $\bigwedge P'.\ [\![\Psi \rhd P \longmapsto\tau \prec P']\!] \Longrightarrow Prop\ ((\!|\nu x|\!)P')$

shows *Prop* P'
proof −
note *Trans* ‹$x \sharp \Psi$›
moreover have $x \sharp (\tau)$ **by** *simp*
moreover note ‹$x \sharp P'$›
moreover have $bn(\tau) \sharp\!* \Psi$ **and** $bn(\tau) \sharp\!* P$ **and** $bn(\tau) \sharp\!* subject(\tau)$ **and** $bn(\tau)$
$= []$ **by** *simp+*
ultimately show *?thesis*
by(*induct rule*: *resCases*) (*auto intro*: *rScope*)
qed

lemma *weakResPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and *Rel* :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)$ *psi*

and x :: *name*
and Rel' :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow\!<Rel>\ Q$
and *eqvt* Rel'
and $x \sharp \Psi$
and $Rel \subseteq Rel'$
and *C1*: $\bigwedge \Psi'\ R\ S\ y.\ [\![(\Psi', R, S) \in Rel;\ y \sharp \Psi']\!] \Longrightarrow (\Psi', (\!|\nu y|\!)R, (\!|\nu y|\!)S) \in Rel'$

shows $\Psi \rhd (\!|\nu x|\!)P \rightsquigarrow\!<Rel'>\ (\!|\nu x|\!)Q$
proof −
note ‹*eqvt* Rel'› ‹$x \sharp \Psi$›
moreover have $x \sharp (\!|\nu x|\!)P$ **and** $x \sharp (\!|\nu x|\!)Q$ **by**(*simp add*: *abs-fresh*)+
ultimately show *?thesis*
proof(*induct rule*: *weakSimIFresh*[*of* - - - - - ()])
case(*cAct* $\alpha\ Q'\ \Psi'$)
from ‹$bn\ \alpha \sharp\!* (\!|\nu x|\!)P$› ‹$bn\ \alpha \sharp\!* (\!|\nu x|\!)Q$› ‹$x \sharp \alpha$› **have** $bn\ \alpha \sharp\!* P$ **and** $bn\ \alpha \sharp\!* Q$ **by** *simp*+
from ‹$\Psi \rhd (\!|\nu x|\!)Q \longmapsto \alpha \prec Q'$› ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› ‹$x \sharp Q'$› ‹$bn\ \alpha \sharp\!* \Psi$› ‹$bn\ \alpha \sharp\!* Q$› ‹$bn\ \alpha \sharp\!* subject\ \alpha$› ‹$bn\ \alpha \sharp\!* P$› ‹$x \sharp \alpha$›
show *?case*
proof(*induct rule*: *resCases*)
case(*cOpen* $M\ xvec1\ xvec2\ y\ N\ Q'$)
from ‹$bn(M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle) \sharp\!* P$› **have** $xvec1 \sharp\!* P$ **and** $xvec2 \sharp\!* P$ **and** $y \sharp P$ **by** *simp*+
from ‹$x \sharp (M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle)$› **have** $x \sharp xvec1$ **and** $x \neq y$ **and** $x \sharp xvec2$ **and** $x \sharp M$ **by** *simp*+
from *PSimQ* ‹$\Psi \rhd Q \longmapsto M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot Q')$› ‹$xvec1 \sharp\!* \Psi$› ‹$xvec2 \sharp\!* \Psi$› ‹$xvec1 \sharp\!* P$› ‹$xvec2 \sharp\!* P$› ‹$\alpha \neq \tau$›
obtain $P''\ P'$ **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle \prec P''$
and $P''Chain$: $\Psi \otimes ([(x, y)] \cdot \Psi') \rhd P'' \Longrightarrow\hat{}_{\tau}\ P'$ **and** $P'RelQ'$: $(\Psi \otimes ([(x, y)] \cdot \Psi'), P', ([(x, y)] \cdot Q')) \in Rel$
by (*fastforce dest*: *weakSimE*)
from *PTrans* **have** $([(x, y)] \cdot \Psi) : ([(x, y)] \cdot Q) \rhd ([(x, y)] \cdot P) \Longrightarrow([(x, y)] \cdot (M(\!|\nu*(xvec1@xvec2)|\!)\langle([(x, y)] \cdot N)\rangle)) \prec ([(x, y)] \cdot P'')$
by(*rule eqvts*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› ‹$x \sharp M$› ‹$y \sharp M$› ‹$x \sharp xvec1$› ‹$y \sharp xvec1$› ‹$x \sharp xvec2$› ‹$y \sharp xvec2$›
have $\Psi : ([(x, y)] \cdot Q) \rhd ([(x, y)] \cdot P) \Longrightarrow M(\!|\nu*(xvec1@xvec2)|\!)\langle N\rangle \prec ([(x, y)] \cdot P'')$
by(*simp add*: *eqvts*)
hence $\Psi : (\!|\nu y|\!)([(x, y)] \cdot Q) \rhd (\!|\nu y|\!)([(x, y)] \cdot P) \Longrightarrow M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec ([(x, y)] \cdot P'')$
using ‹$y \in supp\ N$› ‹$y \sharp \Psi$› ‹$y \sharp M$› ‹$y \sharp xvec1$› ‹$y \sharp xvec2$›
by(*rule weakOpen*)
with ‹$y \sharp P$› ‹$y \sharp Q$› **have** $\Psi : (\!|\nu x|\!)Q \rhd (\!|\nu x|\!)P \Longrightarrow M(\!|\nu*(xvec1@y\#xvec2)|\!)\langle N\rangle \prec ([(x, y)] \cdot P'')$

by(*simp add*: *alphaRes*)
moreover from $P''Chain$ **have** $([(x, y)] \cdot (\Psi \otimes ([(x, y)] \cdot \Psi'))) \rhd ([(x, y)] \cdot P'') \Longrightarrow\hat{}_\tau ([(x, y)] \cdot P')$
by(*rule eqvts*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \otimes \Psi' \rhd ([(x, y)] \cdot P'') \Longrightarrow\hat{}_\tau ([(x, y)] \cdot P')$ **by**(*simp add*: *eqvts*)
moreover from $P'RelQ'$ ‹$Rel \subseteq Rel'$› **have** $(\Psi \otimes ([(x, y)] \cdot \Psi'), P', ([(x, y)] \cdot Q')) \in Rel'$ **by** *auto*
with ‹*eqvt Rel'*› **have** $([(x, y)] \cdot (\Psi \otimes ([(x, y)] \cdot \Psi'), P', ([(x, y)] \cdot Q'))) \in Rel'$ **by**(*rule eqvtI*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $(\Psi \otimes \Psi', [(x, y)] \cdot P', Q') \in Rel'$ **by**(*simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
case(*cRes Q'*)
from $PSimQ$ ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$\alpha \neq \tau$›
obtain $P''\ P'$ **where** $PTrans$: $\Psi : Q \rhd P \Longrightarrow\alpha \prec P''$
and $P''Chain$: $\Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **and** $P'RelQ'$: $(\Psi \otimes \Psi', P', Q') \in Rel$
by(*blast dest*: *weakSimE*)
from $PTrans$ ‹$x \sharp \Psi$› ‹$x \sharp \alpha$› **have** $\Psi : (\!|\nu x|\!)Q \rhd (\!|\nu x|\!)P \Longrightarrow\alpha \prec (\!|\nu x|\!)P''$
by(*rule-tac weakScope*)
moreover from $P''Chain$ ‹$x \sharp \Psi$› ‹$x \sharp \Psi'$› **have** $\Psi \otimes \Psi' \rhd (\!|\nu x|\!)P'' \Longrightarrow\hat{}_\tau (\!|\nu x|\!)P'$
by(*rule-tac tauChainResPres*) *auto*
moreover from $P'RelQ'$ ‹$x \sharp \Psi$› ‹$x \sharp \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu x|\!)P', (\!|\nu x|\!)Q') \in Rel'$
apply(*rule-tac C1*) **by**(*auto simp add*: *fresh-left calc-atm*)
ultimately show *?case* **by** *blast*
qed
next
case(*cTau Q'*)
from ‹$\Psi \rhd (\!|\nu x|\!)Q \longmapsto\tau \prec Q'$› ‹$x \sharp \Psi$› ‹$x \sharp Q'$›
show *?case*
proof(*induct rule*: *resTauCases*)
case(*cRes Q'*)
from $PSimQ$ ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$› **obtain** P' **where** $PChain$: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **and** $P'RelQ'$: $(\Psi, P', Q') \in Rel$
by(*blast dest*: *weakSimE*)
from $PChain$ ‹$x \sharp \Psi$› **have** $\Psi \rhd (\!|\nu x|\!)P \Longrightarrow\hat{}_\tau (\!|\nu x|\!)P'$ **by**(*rule tauChainResPres*)
moreover from $P'RelQ'$ ‹$x \sharp \Psi$› **have** $(\Psi, (\!|\nu x|\!)P', (\!|\nu x|\!)Q') \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
qed
qed

lemma *weakResChainPres*:

fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow$<*Rel*> Q
and *eqvt Rel*
and *xvec* $\sharp*$ Ψ
and *C1*: $\bigwedge \Psi'\ R\ S\ yvec.\ [\![(\Psi', R, S) \in Rel;\ yvec \sharp* \Psi']\!] \Longrightarrow (\Psi', (\!|\nu{*}yvec|\!)R, (\!|\nu{*}yvec|\!)S) \in Rel$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow$<*Rel*> $(\!|\nu{*}xvec|\!)Q$
using ‹*xvec* $\sharp*$ Ψ›
proof(*induct xvec*)
case *Nil*
from *PSimQ* **show** *?case* **by** *simp*
next
case(*Cons x xvec*)
from ‹(x#*xvec*) $\sharp*$ Ψ› **have** $x \sharp \Psi$ **and** *xvec* $\sharp*$ Ψ **by** *simp*+
from ‹*xvec* $\sharp*$ $\Psi \Longrightarrow \Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow$<*Rel*> $(\!|\nu{*}xvec|\!)Q$› ‹*xvec* $\sharp*$ Ψ›
have $\Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow$<*Rel*> $(\!|\nu{*}xvec|\!)Q$ **by** *simp*
moreover note ‹*eqvt Rel*› ‹$x \sharp \Psi$›
moreover have $Rel \subseteq Rel$ **by** *simp*
moreover have $\bigwedge \Psi\ P\ Q\ x.\ [\![(\Psi, P, Q) \in Rel;\ x \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu{*}[x]|\!)P, (\!|\nu{*}[x]|\!)Q) \in Rel$
by(*rule-tac yvec*=[x] **in** *C1*) *auto*
hence $\bigwedge \Psi\ P\ Q\ x.\ [\![(\Psi, P, Q) \in Rel;\ x \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu x|\!)P, (\!|\nu x|\!)Q) \in Rel$
by *simp*
ultimately have $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec|\!)P) \rightsquigarrow$<*Rel*> $(\!|\nu x|\!)((\!|\nu{*}xvec|\!)Q)$
by(*rule weakResPres*)
thus *?case* **by** *simp*
qed

lemma *parTauCases*[*consumes 1*, *case-names cPar1 cPar2 cComm1 cComm2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto\tau \prec R$
and *rPar1*: $\bigwedge P'\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto\tau \prec P';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$A_Q \sharp* \Psi;\ A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* C]\!] \Longrightarrow Prop\ (P' \parallel Q)$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto\tau \prec Q';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* C]\!] \Longrightarrow Prop\ (P \parallel Q')$
and *rComm1*: $\bigwedge \Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K\ xvec\ Q'\ A_Q.$

$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C]\!] \Longrightarrow$
Prop $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$
and *rComm2*: $\bigwedge \Psi_Q\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ K\ Q'\ A_Q$.
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C]\!] \Longrightarrow$
Prop $((\!|\nu{*}xvec|\!)(P' \parallel Q'))$

shows *Prop R*
proof –
from *Trans* **obtain** α **where** $\Psi \rhd P \parallel Q \longmapsto \alpha \prec R$ **and** $bn\ \alpha \sharp* \Psi$ **and** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp*$ *subject* α **and** $\alpha = \tau$ **by** *auto*
thus *?thesis* **using** *rPar1 rPar2 rComm1 rComm2*
by(*induct rule*: *parCases*[**where** *C*=*C*]) (*auto simp add*: *residualInject*)
qed

lemma *weakParPres*:
fixes Ψ :: *'b*
and *P* :: *('a, 'b, 'c) psi*
and *Rel* :: *('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set*
and *Q* :: *('a, 'b, 'c) psi*
and *R* :: *('a, 'b, 'c) psi*
and *Rel'* :: *('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set*

assumes *PRelQ*: $\bigwedge A_R\ \Psi_R.\ [\![$*extractFrame* $R = \langle A_R, \Psi_R\rangle$; $A_R \sharp* \Psi$; $A_R \sharp* P$; $A_R \sharp* Q]\!] \Longrightarrow (\Psi \otimes \Psi_R, P, Q) \in Rel$
and *Eqvt*: *eqvt Rel*
and *Eqvt'*: *eqvt Rel'*

and *Sim*: $\bigwedge \Psi'\ S\ T.\ (\Psi', S, T) \in Rel \Longrightarrow \Psi' \rhd S \rightsquigarrow\!<Rel>\ T$
and *Sym*: $\bigwedge \Psi'\ S\ T.\ (\Psi', S, T) \in Rel \Longrightarrow (\Psi', T, S) \in Rel$
and *Ext*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi', S, T) \in Rel]\!] \Longrightarrow (\Psi' \otimes \Psi'', S, T) \in Rel$
and *StatImp*: $\bigwedge \Psi'\ S\ T.\ (\Psi', S, T) \in Rel \Longrightarrow \Psi' \rhd S \lessapprox\!<Rel>\ T$

and *C1*: $\bigwedge\Psi'\ S\ T\ A_U\ \Psi_U\ U.\ [\![(\Psi' \otimes \Psi_U,\ S,\ T) \in Rel;\ extractFrame\ U = \langle A_U,\ \Psi_U\rangle;\ A_U \sharp* \Psi';\ A_U \sharp* S;\ A_U \sharp* T]\!] \Longrightarrow (\Psi',\ S \parallel U,\ T \parallel U) \in Rel'$
and *C2*: $\bigwedge\Psi'\ S\ T\ xvec.\ [\![(\Psi',\ S,\ T) \in Rel';\ xvec \sharp* \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)S,\ (\!|\nu{*}xvec|\!)T) \in Rel'$
and *C3*: $\bigwedge\Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel;\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'',\ S,\ T) \in Rel$

shows $\Psi \rhd P \parallel R \rightsquigarrow\!<Rel'>\ Q \parallel R$
proof −
from *Eqvt'* **show** *?thesis*
proof(*induct rule*: *weakSimI*[**where** *C*=()])
case(*cAct* $\Psi'\ \alpha\ QR$)
from ‹$bn\ \alpha \sharp* (P \parallel R)$› ‹$bn\ \alpha \sharp* (Q \parallel R)$›
have $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp* R$
by *simp*+
from ‹$\Psi \rhd Q \parallel R \longmapsto\alpha \prec QR$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* R$› ‹$bn\ \alpha \sharp*\ subject\ \alpha$› ‹$\alpha \neq \tau$› **show** *?case*
proof(*induct rule*: *parCases*[**where** $C=(P,\ R,\ \Psi')$])
case(*cPar1* $Q'\ A_R\ \Psi_R$)
from ‹$A_R \sharp* (P,\ R,\ \Psi')$› **have** $A_R \sharp* P$ **and** $A_R \sharp* \Psi'$ **by** *simp*+
have *FrR*: $extractFrame\ R = \langle A_R,\ \Psi_R\rangle$ **by** *fact*
from ‹$A_R \sharp* \alpha$› ‹$bn\ \alpha \sharp* R$› *FrR*
have $bn\ \alpha \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto\alpha \prec Q'$ **by** *fact*
from *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› **have** $\Psi \otimes \Psi_R \rhd P \rightsquigarrow\!<Rel>\ Q$
by(*blast intro*: *PRelQ Sim*)
then obtain $P''\ P'$ **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow\alpha \prec P''$
and *P''Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P'$ **and** *P'RelQ'*: $((\Psi \otimes \Psi_R) \otimes \Psi',\ P',\ Q') \in Rel$
using *QTrans* ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_R$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* R$› ‹$A_R \sharp* \alpha$› ‹$\alpha \neq \tau$›
by(*drule-tac* Ψ'=Ψ' **in** *weakSimE*(*1*)) *auto*

from *PTrans QTrans* ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* \alpha$› ‹$bn\ \alpha \sharp*\ subject\ \alpha$› ‹$distinct(bn\ \alpha)$›
have $A_R \sharp* P''$ **and** $A_R \sharp* Q'$
by(*fastforce dest*: *weakFreshChainDerivative freeFreshChainDerivative*)+
with *P''Chain* **have** $A_R \sharp* P'$ **by**(*force dest*: *tauChainFreshChain*)

from *PTrans FrR* ‹$bn\ \alpha \sharp* R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* \alpha$› ‹$A_R \sharp* Q$›
have $\Psi : Q \parallel R \rhd P \parallel R \Longrightarrow\alpha \prec (P'' \parallel R)$
by(*rule-tac weakPar1*)
moreover from *P''Chain* **have** $(\Psi \otimes \Psi') \otimes \Psi_R \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
by(*metis tauChainStatEq Associativity Commutativity Composition*)
with *FrR* **have** $\Psi \otimes \Psi' \rhd P'' \parallel R \Longrightarrow\hat{}_\tau\ P' \parallel R$ **using** ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi'$› ‹$A_R \sharp* P''$›
by(*rule-tac tauChainPar1*) *auto*

moreover from *P'RelQ'* **have** $((\Psi \otimes \Psi') \otimes \Psi_R,\ P',\ Q') \in Rel$
by(*metis C3 compositionSym Associativity Commutativity*)

hence $(\Psi \otimes \Psi', P' \parallel R, Q' \parallel R) \in Rel'$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi'$› ‹$A_R \sharp* P'$› ‹$A_R \sharp* Q'$›
by(*rule-tac C1*) *auto*
ultimately show *?case* **by** *blast*
next
case(*cPar2* R' A_Q Ψ_Q)
from ‹$A_Q \sharp* (P, R, \Psi')$› **have** $A_Q \sharp* P$ **and** $A_Q \sharp* R$ **and** $A_Q \sharp* \Psi'$ **by** *simp*+
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* (\Psi, A_Q, \Psi_Q, \alpha, R)$
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* A_Q$ **and** $A_P \sharp* \Psi_Q$ **and** $A_P \sharp* \alpha$ **and** $A_P \sharp* R$
by *simp*+

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹$A_Q \sharp* P$› *FrP* ‹$A_P \sharp* A_Q$› **have** $A_Q \sharp* \Psi_P$
by(*drule-tac extractFrameFreshChain*) *auto*

from *FrP FrQ* ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$A_P \sharp* \alpha$› ‹$A_Q \sharp* \alpha$›
have $bn\ \alpha \sharp* \Psi_P$ **and** $bn\ \alpha \sharp* \Psi_Q$
by(*force dest*: *extractFrameFreshChain*)+

obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **and** $A_R \sharp* (\Psi, P, Q, A_Q, A_P, \Psi_Q, \Psi_P, \alpha, R, \Psi')$
and *distinct* A_R
by(*rule freshFrame*)
then have $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* A_Q$ **and** $A_R \sharp* A_P$ **and** $A_R \sharp* \Psi_Q$ **and** $A_R \sharp* \Psi_P$
and $A_R \sharp* \alpha$ **and** $A_R \sharp* R$ **and** $A_R \sharp* \Psi'$
by *simp*+

from ‹$A_Q \sharp* R$› *FrR* ‹$A_R \sharp* A_Q$› **have** $A_Q \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from ‹$A_P \sharp* R$› ‹$A_R \sharp* A_P$› *FrR* **have** $A_P \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*

moreover from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto \alpha \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* \alpha$› ‹$A_R \sharp* \Psi'$›
‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$bn\ \alpha \sharp* Q$› ‹$bn\ \alpha \sharp* R$› ‹$bn\ \alpha \sharp*$ *subject* α› ‹*distinct*$(bn\ \alpha)$›
obtain p Ψ'' A_R' Ψ_R' **where** *S*: *set* $p \subseteq$ *set*$(bn\ \alpha) \times$ *set*$(bn(p \cdot \alpha))$ **and** $(p \cdot \Psi_R) \otimes \Psi'' \simeq \Psi_R'$
and *FrR'*: *extractFrame* $R' = \langle A_R', \Psi_R'\rangle$
and $bn(p \cdot \alpha) \sharp* \Psi$ **and** $bn(p \cdot \alpha) \sharp* P$ **and** $bn(p \cdot \alpha) \sharp* Q$
and $A_R' \sharp* \Psi$ **and** $A_R' \sharp* P$ **and** $A_R' \sharp* Q$ **and** $A_R' \sharp* \Psi'$ **and** *distinctPerm* p
by(*rule-tac* C=(Ψ, Ψ', P, Q) **and** C'=(Ψ, P, Q) **in** *expandFrame*) (*assumption* | *simp*)+

from *FrR* ‹A_R ♯* Ψ› ‹A_R ♯* P› ‹A_R ♯* Q›
have $(\Psi \otimes \Psi_R,\ P,\ Q) \in$ *Rel* **by**(*rule PRelQ*)
hence $(\Psi \otimes \Psi_R,\ Q,\ P) \in$ *Rel* **by**(*rule Sym*)
hence $\Psi \otimes \Psi_R \rhd Q \lessapprox$<*Rel*> P **by**(*rule StatImp*)
then obtain P' P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P'$
and *QimpP'*: *insertAssertion*(*extractFrame* Q) $(\Psi \otimes \Psi_R)$ $\hookrightarrow_F$ *insertAssertion*(*extractFrame* P') $(\Psi \otimes \Psi_R)$
and *P'Chain*: $(\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')) \rhd P' \Longrightarrow\hat{}_\tau P''$
and *P'RelQ*: $((\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')),\ Q,\ P'') \in$ *Rel*
by(*rule weakStatImpE*)
obtain $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P' = \langle A_P{}',\ \Psi_P{}'\rangle$ **and** $A_P{}'$ ♯* Ψ **and** $A_P{}'$ ♯* Ψ_R **and** $A_P{}'$ ♯* Ψ_Q
and $A_P{}'$ ♯* A_Q **and** $A_P{}'$ ♯* R **and** $A_P{}'$ ♯* A_R **and** $A_P{}'$ ♯* α
by(*rule-tac* C=(Ψ, Ψ_R, Ψ_Q, A_Q, R, A_R, α) **in** *freshFrame*) *auto*

from *PChain* ‹A_R ♯* P› ‹A_Q ♯* P› ‹*bn* α ♯* P› **have** A_Q ♯* P' **and** A_R ♯* P' **and** *bn* α ♯* P'
by(*force intro*: *tauChainFreshChain*)+
from ‹A_R ♯* P'› ‹$A_P{}'$ ♯* A_R› ‹A_Q ♯* P'› ‹$A_P{}'$ ♯* A_Q› *FrP'* **have** A_Q ♯* $\Psi_P{}'$ **and** A_R ♯* $\Psi_P{}'$
by(*force dest*: *extractFrameFreshChain*)+

from *PChain FrR* ‹A_R ♯* Ψ› ‹A_R ♯* P› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau P' \parallel R$ **by**(*rule tauChainPar1*)
moreover have *insertAssertion* (*extractFrame*($Q \parallel R$)) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame*($P' \parallel R$)) Ψ
proof −
have $\langle A_Q,\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle \simeq_F \langle A_Q,\ (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
moreover from *QimpP' FrQ FrP'* ‹A_Q ♯* Ψ_R› ‹$A_P{}'$ ♯* Ψ_R› ‹$A_P{}'$ ♯* Ψ› ‹A_Q ♯* Ψ›
have $\langle A_Q,\ (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}',\ (\Psi \otimes \Psi_R) \otimes \Psi_P{}'\rangle$ **using** *freshCompChain* **by** *simp*
moreover have $\langle A_P{}',\ (\Psi \otimes \Psi_R) \otimes \Psi_P{}'\rangle \simeq_F \langle A_P{}',\ \Psi \otimes \Psi_P{}' \otimes \Psi_R\rangle$
by(*metis Associativity Composition Commutativity AssertionStatEqTrans AssertionStatEqSym frameNilStatEq frameResChainPres*)
ultimately have $\langle A_Q,\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}',\ \Psi \otimes \Psi_P{}' \otimes \Psi_R\rangle$
by(*rule FrameStatEqImpCompose*)
hence $\langle (A_R@A_Q),\ \Psi \otimes \Psi_Q \otimes \Psi_R\rangle \hookrightarrow_F \langle (A_R@A_P{}'),\ \Psi \otimes \Psi_P{}' \otimes \Psi_R\rangle$
by(*force intro*: *frameImpResChainPres simp add*: *frameChainAppend*)
with *FrQ FrR FrP'* ‹A_R ♯* A_Q› ‹$A_P{}'$ ♯* A_R› ‹A_Q ♯* Ψ_R› ‹A_R ♯* Ψ_Q› ‹A_R ♯* $\Psi_P{}'$› ‹$A_P{}'$ ♯* Ψ_R›
‹$A_P{}'$ ♯* Ψ› ‹A_Q ♯* Ψ› ‹A_R ♯* Ψ›
show *?thesis*
by(*simp add*: *frameChainAppend*) (*metis frameImpChainComm FrameStatImpTrans*)

qed
moreover have *RTrans*: $\Psi \otimes \Psi_P{}' \rhd R \longmapsto\alpha \prec R'$
proof −
have $\Psi \otimes \Psi_Q \rhd R \longmapsto\alpha \prec R'$ **by** *fact*
moreover have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}', (\Psi \otimes \Psi_P{}') \otimes \Psi_R\rangle$
proof −
have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym FrameStatEqSym*)
moreover with *FrP′ FrQ QimpP′* ‹$A_P{}' \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P{}' \sharp* \Psi_R$› ‹$A_Q \sharp* \Psi_R$›
have $\langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'\rangle$ **using** *freshCompChain*
by *simp*
moreover have $\langle A_P{}', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'\rangle \simeq_F \langle A_P{}', (\Psi \otimes \Psi_P{}') \otimes \Psi_R\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym frameIntAssociativity*[*THEN FrameStatEqSym*])
ultimately show *?thesis*
by(*rule FrameStatEqImpCompose*)
qed
ultimately show *?thesis*
using ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P{}'$› ‹$A_P{}' \sharp* R$› ‹$A_Q \sharp* R$› ‹$A_P{}' \sharp* \alpha$› ‹$A_Q \sharp* \alpha$›
‹$A_P{}' \sharp* A_R$› ‹$A_R \sharp* A_Q$› ‹$A_R \sharp* \Psi_P{}'$› ‹$A_R \sharp* \Psi_Q$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P{}'$› ‹$A_Q \sharp* \Psi_P{}'$› *FrR* ‹*distinct* A_R›
by(*force intro*: *transferFrame*)
qed
hence $\Psi \rhd P' \parallel R \longmapsto\alpha \prec (P' \parallel R')$ **using** *FrP′* ‹$bn\ \alpha \sharp* P'$› ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* R$› ‹$A_P{}' \sharp* \alpha$› ‹$A_P{}' \sharp* \alpha$›
by(*rule-tac Par2*)
ultimately have $\Psi : (Q \parallel R) \rhd P \parallel R \Longrightarrow\alpha \prec (P' \parallel R')$
by(*rule-tac weakTransitionI*)
moreover from *PChain P′Chain* ‹$bn\ \alpha \sharp* P'$› ‹$bn(p \cdot \alpha) \sharp* P$› ‹$A_R{}' \sharp* P$›
have $bn\ \alpha \sharp* P''$ **and** $bn(p \cdot \alpha) \sharp* P'$ **and** $bn(p \cdot \alpha) \sharp* P''$ **and** $A_R{}' \sharp* P'$ **and** $A_R{}' \sharp* P''$
by(*metis tauChainFreshChain*)+
from *P′Chain* **have** $(p \cdot ((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'') \otimes (p \cdot \Psi'))) \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot P'')$
by(*rule eqvts*)
with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn(p \cdot \alpha) \sharp* \Psi$› ‹$bn\ \alpha \sharp* P'$› ‹$bn(p \cdot \alpha) \sharp* P'$› ‹$bn\ \alpha \sharp* P''$› ‹$bn(p \cdot \alpha) \sharp* P''$›
S ‹*distinctPerm p*›
have $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi'' \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P''$
by(*simp add*: *eqvts*)
hence $(\Psi \otimes \Psi') \otimes (p \cdot \Psi_R) \otimes \Psi'' \rhd P' \Longrightarrow\hat{}_\tau P''$
by(*rule tauChainStatEq*) (*metis Associativity Commutativity Composition AssertionStatEqTrans*)
with ‹$(p \cdot \Psi_R) \otimes \Psi'' \simeq \Psi_R{}'$› **have** $(\Psi \otimes \Psi') \otimes \Psi_R{}' \rhd P' \Longrightarrow\hat{}_\tau P''$ **by**(*metis tauChainStatEq compositionSym*)

hence $\Psi \otimes \Psi' \rhd P' \parallel R' \Longrightarrow\hat{}_\tau P'' \parallel R'$ **using** *FrR'* ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* \Psi'$› ‹$A_R{}' \sharp* P'$› **by**(*rule-tac tauChainPar1*) *auto*

moreover from *P'RelQ* **have** $((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'') \otimes (p \cdot \Psi'), P'', Q) \in$ *Rel* **by**(*rule Sym*)

with ‹*eqvt Rel*› **have** $(p \cdot ((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'') \otimes (p \cdot \Psi'), P'', Q)) \in$ *Rel* **by**(*rule eqvtI*)

with ‹$bn\ \alpha \sharp* \Psi$› ‹$bn(p \cdot \alpha) \sharp* \Psi$› ‹$bn\ \alpha \sharp* P''$› ‹$bn(p \cdot \alpha) \sharp* P''$› ‹$bn\ \alpha \sharp* Q$› ‹$bn(p \cdot \alpha) \sharp* Q$› *S* ‹*distinctPerm p*›

have $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi'' \otimes \Psi', P'', Q) \in$ *Rel* **by**(*simp add*: *eqvts*)

with ‹$(p \cdot \Psi_R) \otimes \Psi'' \simeq \Psi_R{}'$› **have** $((\Psi \otimes \Psi') \otimes \Psi_R{}', P'', Q) \in$ *Rel*

by(*rule-tac C3*, *auto*) (*metis Associativity Commutativity Composition AssertionStatEqTrans*)

with *FrR'* ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* \Psi'$› ‹$A_R{}' \sharp* P''$› ‹$A_R{}' \sharp* Q$›

have $(\Psi \otimes \Psi', P'' \parallel R', Q \parallel R') \in$ *Rel'* **by**(*rule-tac C1*) *auto*

ultimately show *?case* **by** *blast*

next

case *cComm1*

from ‹$\tau \neq \tau$› **have** *False* **by** *simp*

thus *?case* **by** *simp*

next

case *cComm2*

from ‹$\tau \neq \tau$› **have** *False* **by** *simp*

thus *?case* **by** *simp*

qed

next

case(*cTau QR*)

from ‹$\Psi \rhd Q \parallel R \longmapsto\tau \prec QR$› **show** *?case*

proof(*induct rule*: *parTauCases*[**where** $C=(P, R)$])

case(*cPar1* Q' A_R Ψ_R)

from ‹$A_R \sharp* (P, R)$› **have** $A_R \sharp* P$

by *simp+*

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*

with ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› **have** $\Psi \otimes \Psi_R \rhd P \rightsquigarrow$<*Rel*> Q

by(*blast intro*: *PRelQ Sim*)

moreover have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto\tau \prec Q'$ **by** *fact*

ultimately obtain P' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P'$ **and** *P'RelQ'*: $(\Psi \otimes \Psi_R, P', Q') \in$ *Rel*

by(*force dest*: *weakSimE*)

from *PChain QTrans* ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› **have** $A_R \sharp* P'$ **and** $A_R \sharp* Q'$

by(*force dest*: *freeFreshChainDerivative tauChainFreshChain*)+

from *PChain FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau (P' \parallel R)$

by(*rule tauChainPar1*)

moreover from *P'RelQ' FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P'$› ‹$A_R \sharp* Q'$› **have** $(\Psi, P' \parallel R, Q' \parallel R) \in$ *Rel'* **by**(*rule C1*)

ultimately show *?case* **by** *blast*

next

case(*cPar2* R' A_Q Ψ_Q)

from ‹$A_Q \sharp* (P, R)$› **have** $A_Q \sharp* P$ **and** $A_Q \sharp* R$ **by** *simp+*

obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* (\Psi, A_Q,$

Ψ_Q, R)
by(*rule freshFrame*)
hence $A_P \sharp* \Psi$ **and** $A_P \sharp* A_Q$ **and** $A_P \sharp* \Psi_Q$ **and** $A_P \sharp* R$
by *simp*+

have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$ **by** *fact*
from ‹$A_Q \sharp* P$› *FrP* ‹$A_P \sharp* A_Q$› **have** $A_Q \sharp* \Psi_P$
by(*drule-tac extractFrameFreshChain*) *auto*

obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R \rangle$ **and** $A_R \sharp* (\Psi, P, Q, A_Q, A_P, \Psi_Q, \Psi_P, R)$ **and** *distinct* A_R
by(*rule freshFrame*)
then have $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* A_Q$ **and** $A_R \sharp* A_P$ **and** $A_R \sharp* \Psi_Q$ **and** $A_R \sharp* \Psi_P$
and $A_R \sharp* R$
by *simp*+

from ‹$A_Q \sharp* R$› *FrR* ‹$A_R \sharp* A_Q$› **have** $A_Q \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from ‹$A_P \sharp* R$› ‹$A_R \sharp* A_P$› *FrR* **have** $A_P \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*

moreover from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto\tau \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$›
obtain Ψ' $A_R{}'$ $\Psi_R{}'$ **where** $\Psi_R \otimes \Psi' \simeq \Psi_R{}'$ **and** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}' \rangle$
and $A_R{}' \sharp* \Psi$ **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q$
by(*rule-tac* C=(Ψ, P, Q, R) **in** *expandTauFrame*) (*assumption* | *simp*)+

from *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$›
obtain P' P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P'$
and *QimpP'*: *insertAssertion*(*extractFrame* Q) ($\Psi \otimes \Psi_R$) $\hookrightarrow_F$ *insertAssertion*(*extractFrame* P') ($\Psi \otimes \Psi_R$)
and *P'Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P''$
and *P'RelQ*: $((\Psi \otimes \Psi_R) \otimes \Psi', Q, P'') \in Rel$
by(*metis StatImp PRelQ Sym weakStatImpE*)
obtain $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}' \rangle$ **and** $A_P{}' \sharp* \Psi$ **and** $A_P{}' \sharp* \Psi_R$ **and** $A_P{}' \sharp* \Psi_Q$
and $A_P{}' \sharp* A_Q$ **and** $A_P{}' \sharp* R$ **and** $A_P{}' \sharp* A_R$
by(*rule-tac* C=(Ψ, Ψ_R, Ψ_Q, A_Q, R, A_R) **in** *freshFrame*) *auto*

from *PChain P'Chain* ‹$A_R \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_R{}' \sharp* P$› **have** $A_Q \sharp* P'$ **and** $A_R \sharp* P'$ **and** $A_R{}' \sharp* P'$ **and** $A_R{}' \sharp* P''$
by(*force intro*: *tauChainFreshChain*)+
from ‹$A_R \sharp* P'$› ‹$A_P{}' \sharp* A_R$› ‹$A_Q \sharp* P'$› ‹$A_P{}' \sharp* A_Q$› *FrP'* **have** $A_Q \sharp* \Psi_P{}'$ **and** $A_R \sharp* \Psi_P{}'$
by(*force dest*: *extractFrameFreshChain*)+

from *PChain FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau P' \parallel R$

by(*rule tauChainPar1*)
moreover have *RTrans*: $\Psi \otimes \Psi_P{}' \rhd R \longmapsto\tau \prec R'$
proof −
have $\Psi \otimes \Psi_Q \rhd R \longmapsto\tau \prec R'$ **by** *fact*
moreover have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}', (\Psi \otimes \Psi_P{}') \otimes \Psi_R\rangle$
proof −
have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym FrameStatEqSym*)
moreover with *FrP′ FrQ QimpP′* ‹$A_P{}' \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P{}' \sharp* \Psi_R$› ‹$A_Q \sharp* \Psi_R$›
have $\langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'\rangle$ **using** *freshCompChain*
by *simp*
moreover have $\langle A_P{}', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'\rangle \simeq_F \langle A_P{}', (\Psi \otimes \Psi_P{}') \otimes \Psi_R\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym frameIntAssociativity*[*THEN FrameStatEqSym*])
ultimately show *?thesis*
by(*rule FrameStatEqImpCompose*)
qed
ultimately show *?thesis*
using ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_P{}'$› ‹$A_P{}' \sharp* R$› ‹$A_Q \sharp* R$›
‹$A_P{}' \sharp* A_R$› ‹$A_R \sharp* A_Q$› ‹$A_R \sharp* \Psi_P{}'$› ‹$A_R \sharp* \Psi_Q$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* \Psi_P{}'$› ‹$A_Q \sharp* \Psi_P{}'$› *FrR* ‹*distinct* A_R›
by(*force intro*: *transferTauFrame*)
qed
hence $\Psi \rhd P' \parallel R \longmapsto\tau \prec (P' \parallel R')$ **using** *FrP′* ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* R$›
by(*rule-tac Par2*) *auto*
moreover from *P′Chain* **have** $\Psi \otimes \Psi_R \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P''$
by(*rule tauChainStatEq*) (*metis Associativity*)
with ‹$\Psi_R \otimes \Psi' \simeq \Psi_R{}'$› **have** $\Psi \otimes \Psi_R{}' \rhd P' \Longrightarrow\hat{}_\tau P''$ **by**(*metis tauChainStatEq compositionSym*)
hence $\Psi \rhd P' \parallel R' \Longrightarrow\hat{}_\tau P'' \parallel R'$ **using** *FrR′* ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* P'$› **by**(*rule-tac tauChainPar1*)
ultimately have $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau (P'' \parallel R')$
by(*fastforce dest*: *rtrancl-into-rtrancl*)

moreover from *P′RelQ* ‹$\Psi_R \otimes \Psi' \simeq \Psi_R{}'$› **have** $(\Psi \otimes \Psi_R{}', P'', Q) \in Rel$ **by**(*blast intro*: *C3 Associativity compositionSym Sym*)
with *FrR′* ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* P''$› ‹$A_R{}' \sharp* Q$› **have** $(\Psi, P'' \parallel R', Q \parallel R') \in Rel'$ **by**(*rule-tac C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_R M N Q' A_Q Ψ_Q K *xvec* R' A_R)
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹$A_Q \sharp* (P, R)$› **have** $A_Q \sharp* P$ **and** $A_Q \sharp* R$ **by** *simp*+

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*

from ‹A_R ♯* (P, R)› **have** A_R ♯* P **and** A_R ♯* R **by** *simp*+
from ‹*xvec* ♯* (P, R)› **have** *xvec* ♯* P **and** *xvec* ♯* R **by** *simp*+

have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto M(\!|N|\!) \prec Q'$ **and** *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$ **by** *fact*+

from *RTrans FrR* ‹*distinct* A_R› ‹A_R ♯* R› ‹A_R ♯* *xvec*› ‹A_R ♯* N› ‹*xvec* ♯* R› ‹*xvec* ♯* Q› ‹*xvec* ♯* Ψ› ‹*xvec* ♯* Ψ_Q› ‹A_R ♯* Q›
‹A_R ♯* Ψ› ‹A_R ♯* Ψ_Q› ‹*xvec* ♯* K› ‹A_R ♯* K› ‹A_R ♯* R› ‹*xvec* ♯* R› ‹A_R ♯* P› ‹*xvec* ♯* P›
‹A_Q ♯* A_R› ‹A_Q ♯* *xvec*› ‹*xvec* ♯* K› ‹*distinct xvec*› ‹A_R ♯* N›
obtain p Ψ' $A_R{}'$ $\Psi_R{}'$ **where** S: *set* $p \subseteq$ *set xvec* $\times$ *set*($p \cdot$ *xvec*) **and** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$
and $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}'$ ♯* Q **and** $A_R{}'$ ♯* Ψ **and** ($p \cdot$ *xvec*) ♯* Ψ
and ($p \cdot$ *xvec*) ♯* Q **and** ($p \cdot$ *xvec*) ♯* Ψ_Q **and** ($p \cdot$ *xvec*) ♯* K **and** ($p \cdot$ *xvec*) ♯* R
and ($p \cdot$ *xvec*) ♯* P **and** ($p \cdot$ *xvec*) ♯* A_Q **and** $A_R{}'$ ♯* P **and** $A_R{}'$ ♯* N
by(*rule-tac* C=(Ψ, Q, Ψ_Q, K, R, P, A_Q) **and** C'=(Ψ, Q, Ψ_Q, K, R, P, A_Q) **in** *expandFrame*) (*assumption* | *simp*)+

from ‹A_R ♯* Ψ› **have** ($p \cdot A_R$) ♯* ($p \cdot \Psi$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* Ψ› ‹($p \cdot$ *xvec*) ♯* Ψ› S **have** ($p \cdot A_R$) ♯* Ψ **by** *simp*
from ‹A_R ♯* P› **have** ($p \cdot A_R$) ♯* ($p \cdot P$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* P› ‹($p \cdot$ *xvec*) ♯* P› S **have** ($p \cdot A_R$) ♯* P **by** *simp*
from ‹A_R ♯* Q› **have** ($p \cdot A_R$) ♯* ($p \cdot Q$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* Q› ‹($p \cdot$ *xvec*) ♯* Q› S **have** ($p \cdot A_R$) ♯* Q **by** *simp*
from ‹A_R ♯* R› **have** ($p \cdot A_R$) ♯* ($p \cdot R$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* R› ‹($p \cdot$ *xvec*) ♯* R› S **have** ($p \cdot A_R$) ♯* R **by** *simp*
from ‹A_R ♯* K› **have** ($p \cdot A_R$) ♯* ($p \cdot K$) **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* K› ‹($p \cdot$ *xvec*) ♯* K› S **have** ($p \cdot A_R$) ♯* K **by** *simp*

from ‹A_Q ♯* *xvec*› ‹($p \cdot$ *xvec*) ♯* A_Q› ‹A_Q ♯* M› S **have** A_Q ♯* ($p \cdot M$) **by**(*simp add*: *freshChainSimps*)
from ‹A_Q ♯* *xvec*› ‹($p \cdot$ *xvec*) ♯* A_Q› ‹A_Q ♯* A_R› S **have** A_Q ♯* ($p \cdot A_R$) **by**(*simp add*: *freshChainSimps*)

from *QTrans* S ‹*xvec* ♯* Q› ‹($p \cdot$ *xvec*) ♯* Q› **have** $(p \cdot (\Psi \otimes \Psi_R)) \rhd Q \longmapsto (p \cdot M)(\!|N|\!) \prec Q'$
by(*rule-tac inputPermFrameSubject*) (*assumption* | *auto simp add*: *fresh-star-def*)+
with ‹*xvec* ♯* Ψ› ‹($p \cdot$ *xvec*) ♯* Ψ› S **have** *QTrans*: $(\Psi \otimes (p \cdot \Psi_R)) \rhd Q \longmapsto (p \cdot M)(\!|N|\!) \prec Q'$

by(*simp add*: *eqvts*)
from *FrR* **have** $(p \cdot extractFrame\ R) = p \cdot \langle A_R, \Psi_R\rangle$ **by** *simp*
with ‹*xvec* $\sharp*$ *R*› ‹$(p \cdot xvec) \sharp* R$› *S* **have** *FrR*: $extractFrame\ R = \langle (p \cdot A_R), (p \cdot \Psi_R)\rangle$
by(*simp add*: *eqvts*)

from *MeqK* **have** $(p \cdot (\Psi \otimes \Psi_Q \otimes \Psi_R)) \vdash (p \cdot M) \leftrightarrow (p \cdot K)$ **by**(*rule chanEqClosed*)
with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec) \sharp* \Psi$› ‹*xvec* $\sharp*$ Ψ_Q› ‹$(p \cdot xvec) \sharp* \Psi_Q$› ‹*xvec* $\sharp*$ *K*› ‹$(p \cdot xvec) \sharp* K$› *S*
have *MeqK*: $\Psi \otimes \Psi_Q \otimes (p \cdot \Psi_R) \vdash (p \cdot M) \leftrightarrow K$ **by**(*simp add*: *eqvts*)

from *FrR* ‹$(p \cdot A_R) \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* P$› ‹$(p \cdot A_R) \sharp* Q$›
have $\Psi \otimes (p \cdot \Psi_R) \rhd P \rightsquigarrow$<*Rel*> *Q* **by**(*force intro*: *Sim PRelQ*)

with *QTrans* **obtain** P' P'' **where** *PTrans*: $\Psi \otimes (p \cdot \Psi_R) : Q \rhd P \Longrightarrow (p \cdot M)(\!|N|\!) \prec P''$
and *P''Chain*: $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
and *P'RelQ'*: $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P', Q') \in Rel$
by(*fastforce dest*: *weakSimE*)
from *PTrans QTrans* ‹$A_R{}' \sharp* P$› ‹$A_R{}' \sharp* Q$› ‹$A_R{}' \sharp* N$› **have** $A_R{}' \sharp* P''$ **and** $A_R{}' \sharp* Q'$
by(*auto dest*: *weakInputFreshChainDerivative inputFreshChainDerivative*)

from *PTrans FrQ RTrans FrR MeqK* ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* R$› ‹$A_Q \sharp* (p \cdot M)$› ‹$A_Q \sharp* (p \cdot A_R)$›
‹$(p \cdot A_R) \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* P$› ‹$(p \cdot A_R) \sharp* Q$› ‹$(p \cdot A_R) \sharp* R$› ‹$(p \cdot A_R) \sharp* K$› ‹*xvec* $\sharp*$ *P*› ‹*distinct* A_R›
have $\Psi \rhd P \parallel R \Longrightarrow_\tau (\!|\nu*xvec|\!)(P'' \parallel R')$ **apply**(*rule-tac weakComm1*)
by(*assumption* | *simp*)+

moreover from *P''Chain* ‹$A_R{}' \sharp* P''$› **have** $A_R{}' \sharp* P'$ **by**(*rule tauChainFreshChain*)
from ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$› **have** $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \simeq \Psi \otimes \Psi_R{}'$
by(*metis Associativity AssertionStatEqTrans AssertionStatEqSym compositionSym*)
with *P''Chain* **have** $\Psi \otimes \Psi_R{}' \rhd P'' \Longrightarrow\hat{}_\tau\ P'$ **by**(*rule tauChainStatEq*)
hence $\Psi \rhd P'' \parallel R' \Longrightarrow\hat{}_\tau\ P' \parallel R'$ **using** *FrR'* ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* P''$› **by**(*rule tauChainPar1*)
hence $\Psi \rhd (\!|\nu*xvec|\!)(P'' \parallel R') \Longrightarrow\hat{}_\tau\ (\!|\nu*xvec|\!)(P' \parallel R')$ **using** ‹*xvec* $\sharp*$ Ψ› **by**(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau\ (\!|\nu*xvec|\!)(P' \parallel R')$
by *auto*
moreover from *P'RelQ'* ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$› **have** $(\Psi \otimes \Psi_R{}', P', Q') \in Rel$ **by**(*metis C3 Associativity compositionSym*)
with *FrR'* ‹$A_R{}' \sharp* P'$› ‹$A_R{}' \sharp* Q'$› ‹$A_R{}' \sharp* \Psi$› **have** $(\Psi, P' \parallel R', Q' \parallel R') \in Rel'$ **by**(*rule-tac C1*)
with ‹*xvec* $\sharp*$ Ψ› **have** $(\Psi, (\!|\nu*xvec|\!)(P' \parallel R'), (\!|\nu*xvec|\!)(Q' \parallel R')) \in Rel'$
by(*rule-tac C2*)

ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_R M *xvec* N Q' A_Q Ψ_Q K R' A_R)
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹$A_Q \sharp* (P, R)$› **have** $A_Q \sharp* P$ **and** $A_Q \sharp* R$ **by** *simp+*

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹$A_R \sharp* (P, R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* R$ **by** *simp+*
from ‹*xvec* $\sharp* (P, R)$› **have** *xvec* $\sharp* P$ **and** *xvec* $\sharp* R$ **by** *simp+*

have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$ **and** *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$ **by** *fact+*

from *RTrans FrR* ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* R$› ‹$A_R \sharp* Q'$› ‹$A_R \sharp* N$› ‹$A_R \sharp* P$› ‹$A_R \sharp* xvec$› ‹$A_R \sharp* K$›
obtain Ψ' $A_R{}'$ $\Psi_R{}'$ **where** *ReqR'*: $\Psi_R \otimes \Psi' \simeq \Psi_R{}'$ **and** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$
and $A_R{}' \sharp* \Psi$ **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q'$ **and** $A_R{}' \sharp* N$ **and** $A_R{}' \sharp* xvec$
by(*rule-tac* C=(Ψ, P, Q', N, *xvec*) **and** C'=(Ψ, P, Q', N, *xvec*) **in** *expandFrame*) (*assumption* | *simp*)+

from *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› **have** $\Psi \otimes \Psi_R \rhd P \rightsquigarrow$<*Rel*> Q **by**(*force intro*: *Sim PRelQ*)

with *QTrans* ‹*xvec* $\sharp* \Psi$› ‹*xvec* $\sharp* \Psi_R$› ‹*xvec* $\sharp* P$›
obtain P'' P' **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$
and *P''Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$
and *P'RelQ'*: $((\Psi \otimes \Psi_R) \otimes \Psi', P', Q') \in Rel$
by(*fastforce dest*: *weakSimE*)
from *PTrans* **obtain** P''' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P'''$
and *QimpP'''*: *insertAssertion* (*extractFrame* Q) ($\Psi \otimes \Psi_R$) $\hookrightarrow_F$ *insertAssertion* (*extractFrame* P''') ($\Psi \otimes \Psi_R$)
and *P'''Trans*: $\Psi \otimes \Psi_R \rhd P''' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$
by(*rule weakTransitionE*)

from *PChain* ‹$A_R \sharp* P$› **have** $A_R \sharp* P'''$ **by**(*rule tauChainFreshChain*)

obtain $A_P{}'''$ $\Psi_P{}'''$ **where** *FrP'''*: *extractFrame* $P''' = \langle A_P{}''', \Psi_P{}'''\rangle$ **and** $A_P{}''' \sharp* (\Psi, A_Q, \Psi_Q, A_R, \Psi_R, M, N, K, R, P''', xvec)$ **and** *distinct* $A_P{}'''$
by(*rule freshFrame*)
hence $A_P{}''' \sharp* \Psi$ **and** $A_P{}''' \sharp* A_Q$ **and** $A_P{}''' \sharp* \Psi_Q$ **and** $A_P{}''' \sharp* M$ **and** $A_P{}''' \sharp* R$
and $A_P{}''' \sharp* N$ **and** $A_P{}''' \sharp* K$ **and** $A_P{}''' \sharp* A_R$ **and** $A_P{}''' \sharp* P'''$ **and** $A_P{}''' \sharp* xvec$ **and** $A_P{}''' \sharp* \Psi_R$
by *simp+*

have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$

by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover with *QimpP''' FrP''' FrQ* ‹$A_P{}'''$ $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ› ‹$A_P{}'''$ $\sharp*$ Ψ_R› ‹A_Q $\sharp*$ Ψ_R›
have $\langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}''', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'''\rangle$ **using** *freshCompChain*
by *simp*
moreover have $\langle A_P{}''', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'''\rangle \simeq_F \langle A_P{}''', (\Psi \otimes \Psi_P{}''') \otimes \Psi_R\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
ultimately have *QImpP'''*: $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}''', (\Psi \otimes \Psi_P{}''') \otimes \Psi_R\rangle$
by(*rule FrameStatEqImpCompose*)

from *PChain FrR* ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ P› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau P''' \parallel R$ **by**(*rule tauChainPar1*)
moreover from *RTrans FrR P'''Trans MeqK QImpP''' FrP''' FrQ* ‹*distinct* $A_P{}'''$› ‹*distinct* A_R› ‹$A_P{}'''$ $\sharp*$ A_R› ‹A_Q $\sharp*$ A_R›
‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ P'''› ‹A_R $\sharp*$ Q› ‹A_R $\sharp*$ R› ‹A_R $\sharp*$ K› ‹$A_P{}'''$ $\sharp*$ Ψ› ‹$A_P{}'''$ $\sharp*$ R›
‹$A_P{}'''$ $\sharp*$ P'''› ‹$A_P{}'''$ $\sharp*$ M› ‹A_Q $\sharp*$ R› ‹A_Q $\sharp*$ M› ‹A_R $\sharp*$ *xvec*› ‹*xvec* $\sharp*$ M›
obtain K' **where** $\Psi \otimes \Psi_P{}''' \rhd R \longmapsto K'(\!|N|\!) \prec R'$ **and** $\Psi \otimes \Psi_P{}''' \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $A_R \sharp* K'$
by(*rule-tac comm2Aux*) (*assumption* | *simp*)+

with *P'''Trans FrP'''* **have** $\Psi \rhd P''' \parallel R \longmapsto\tau \prec (\!|\nu*xvec|\!)(P'' \parallel R')$ **using** *FrR* ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ P'''› ‹A_R $\sharp*$ R›
‹*xvec* $\sharp*$ R› ‹$A_P{}'''$ $\sharp*$ Ψ› ‹$A_P{}'''$ $\sharp*$ P'''› ‹$A_P{}'''$ $\sharp*$ R› ‹$A_P{}'''$ $\sharp*$ M› ‹A_R $\sharp*$ K'› ‹$A_P{}'''$ $\sharp*$ A_R›
by(*rule-tac Comm2*)
moreover from *P'''Trans* ‹A_R $\sharp*$ P'''› ‹A_R $\sharp*$ *xvec*› ‹*xvec* $\sharp*$ M› ‹*distinct xvec*› **have** $A_R \sharp* P''$
by(*rule-tac outputFreshChainDerivative*) *auto*

from *PChain* ‹$A_R{}'$ $\sharp*$ P› **have** $A_R{}'$ $\sharp*$ P''' **by**(*rule tauChainFreshChain*)
with *P'''Trans* **have** $A_R{}'$ $\sharp*$ P'' **using** ‹$A_R{}'$ $\sharp*$ *xvec*› ‹*xvec* $\sharp*$ M› ‹*distinct xvec*›
by(*rule-tac outputFreshChainDerivative*) *auto*

with *P''Chain* **have** $A_R{}'$ $\sharp*$ P' **by**(*rule tauChainFreshChain*)
from ‹$\Psi_R \otimes \Psi' \simeq \Psi_R{}'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi' \simeq \Psi \otimes \Psi_R{}'$
by(*metis Associativity AssertionStatEqTrans AssertionStatEqSym compositionSym*)
with *P''Chain* **have** $\Psi \otimes \Psi_R{}' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauChainStatEq*)
hence $\Psi \rhd P'' \parallel R' \Longrightarrow\hat{}_\tau P' \parallel R'$ **using** *FrR'* ‹$A_R{}'$ $\sharp*$ Ψ› ‹$A_R{}'$ $\sharp*$ P''›
by(*rule tauChainPar1*)
hence $\Psi \rhd (\!|\nu*xvec|\!)(P'' \parallel R') \Longrightarrow\hat{}_\tau (\!|\nu*xvec|\!)(P' \parallel R')$
using ‹*xvec* $\sharp*$ Ψ› **by**(*rule tauChainResChainPres*)

ultimately have $\Psi \rhd P \parallel R \Longrightarrow\hat{}_{\tau}$ $(\!|\nu{*}xvec|\!)(P' \parallel R')$ **by**(*drule-tac tauAct-TauChain*) *auto*

moreover from $P'RelQ'$ ‹$\Psi_R \otimes \Psi' \simeq \Psi_R{}'$› **have** $(\Psi \otimes \Psi_R{}', P', Q') \in Rel$ **by**(*metis C3 Associativity compositionSym*)

with FrR' ‹$A_R{}' \sharp{*}\ P'$› ‹$A_R{}' \sharp{*}\ Q'$› ‹$A_R{}' \sharp{*}\ \Psi$› **have** $(\Psi, P' \parallel R', Q' \parallel R') \in Rel'$ **by**(*rule-tac C1*)

with ‹$xvec \sharp{*}\ \Psi$› **have** $(\Psi, (\!|\nu{*}xvec|\!)(P' \parallel R'), (\!|\nu{*}xvec|\!)(Q' \parallel R')) \in Rel'$
by(*rule-tac C2*)

ultimately show *?case* **by** *blast*

qed

qed

qed

unbundle *no relcomp-syntax*

lemma *weakSimBangPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $Rel' :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $Rel'' :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in Rel$
and *eqvt* Rel''
and *guarded* P
and *guarded* Q
and $Rel'Rel$: $Rel' \subseteq Rel$

and *FrameParPres*: $\bigwedge \Psi'\ \Psi_U\ S\ T\ U\ A_U.\ [\![(\Psi' \otimes \Psi_U, S, T) \in Rel;\ extractFrame\ U = \langle A_U, \Psi_U\rangle;\ A_U \sharp{*}\ \Psi';\ A_U \sharp{*}\ S;\ A_U \sharp{*}\ T]\!] \Longrightarrow$
$(\Psi', U \parallel S, U \parallel T) \in Rel$

and *C1*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi', S, T) \in Rel;\ guarded\ S;\ guarded\ T]\!] \Longrightarrow (\Psi', U \parallel !S, U \parallel !T) \in Rel''$

and *ResPres*: $\bigwedge \Psi'\ S\ T\ xvec.\ [\![(\Psi', S, T) \in Rel;\ xvec \sharp{*}\ \Psi']\!] \Longrightarrow (\Psi', (\!|\nu{*}xvec|\!)S, (\!|\nu{*}xvec|\!)T) \in Rel$

and *ResPres'*: $\bigwedge \Psi'\ S\ T\ xvec.\ [\![(\Psi', S, T) \in Rel';\ xvec \sharp{*}\ \Psi']\!] \Longrightarrow (\Psi', (\!|\nu{*}xvec|\!)S, (\!|\nu{*}xvec|\!)T) \in Rel'$

and *Closed*: $\bigwedge \Psi'\ S\ T\ p.\ (\Psi', S, T) \in Rel \Longrightarrow ((p{::}name\ prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel$

and *Closed'*: $\bigwedge \Psi'\ S\ T\ p.\ (\Psi', S, T) \in Rel' \Longrightarrow ((p{::}name\ prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel'$

and *StatEq*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi', S, T) \in Rel;\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'', S, T) \in Rel$

and *StatEq'*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi', S, T) \in Rel';\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'', S, T) \in Rel'$

and *Trans*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi', S, T) \in Rel;\ (\Psi', T, U) \in Rel]\!] \Longrightarrow (\Psi', S, U) \in Rel$

and *Trans'*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi', S, T) \in Rel';\ (\Psi', T, U) \in Rel']\!] \Longrightarrow (\Psi', S, U) \in Rel'$

and *cSim*: $\bigwedge\Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow \Psi' \rhd S \rightsquigarrow\!<Rel>\ T$
and *cExt*: $\bigwedge\Psi'\ S\ T\ \Psi''.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi' \otimes \Psi'',\ S,\ T) \in Rel$
and *cExt'*: $\bigwedge\Psi'\ S\ T\ \Psi''.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow (\Psi' \otimes \Psi'',\ S,\ T) \in Rel'$
and *cSym*: $\bigwedge\Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi',\ T,\ S) \in Rel$
and *cSym'*: $\bigwedge\Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow (\Psi',\ T,\ S) \in Rel'$

and *ParPres*: $\bigwedge\Psi'\ S\ T\ U.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi',\ S \parallel U,\ T \parallel U) \in Rel$
and *ParPres2*: $\bigwedge\Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi',\ S \parallel S,\ T \parallel T) \in Rel$
and *ParPres'*: $\bigwedge\Psi'\ S\ T\ U.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow (\Psi',\ U \parallel S,\ U \parallel T) \in Rel'$

and *Assoc*: $\bigwedge\Psi'\ S\ T\ U.\ (\Psi',\ S \parallel (T \parallel U),\ (S \parallel T) \parallel U) \in Rel$
and *Assoc'*: $\bigwedge\Psi'\ S\ T\ U.\ (\Psi',\ S \parallel (T \parallel U),\ (S \parallel T) \parallel U) \in Rel'$
and *ScopeExt*: $\bigwedge xvec\ \Psi'\ T\ S.\ [\![xvec \sharp* \Psi';\ xvec \sharp* T]\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)(S \parallel T),\ ((\!|\nu{*}xvec|\!)S) \parallel T) \in Rel$
and *ScopeExt'*: $\bigwedge xvec\ \Psi'\ T\ S.\ [\![xvec \sharp* \Psi';\ xvec \sharp* T]\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)(S \parallel T),\ ((\!|\nu{*}xvec|\!)S) \parallel T) \in Rel'$

and *Compose*: $\bigwedge\Psi'\ S\ T\ U\ O.\ [\![(\Psi',\ S,\ T) \in Rel;\ (\Psi',\ T,\ U) \in Rel'';\ (\Psi',\ U,\ O) \in Rel']\!] \Longrightarrow (\Psi',\ S,\ O) \in Rel''$

and *rBangActE*: $\bigwedge\Psi'\ S\ \alpha\ S'.\ [\![\Psi' \rhd !S \longmapsto\alpha \prec S';\ guarded\ S;\ bn\ \alpha \sharp* S;\ \alpha \neq \tau;\ bn\ \alpha \sharp* subject\ \alpha]\!] \Longrightarrow \exists\ T.\ \Psi' \rhd S \longmapsto\alpha \prec T \wedge (\mathbf{1},\ S',\ T \parallel !S) \in Rel'$
and *rBangTauE*: $\bigwedge\Psi'\ S\ S'.\ [\![\Psi' \rhd !S \longmapsto\tau \prec S';\ guarded\ S]\!] \Longrightarrow \exists\ T.\ \Psi' \rhd S \parallel S \longmapsto\tau \prec T \wedge (\mathbf{1},\ S',\ T \parallel !S) \in Rel'$
and *rBangTauI*: $\bigwedge\Psi'\ S\ S'.\ [\![\Psi' \rhd S \parallel S \Longrightarrow\hat{}_\tau\ S';\ guarded\ S]\!] \Longrightarrow \exists\ T.\ \Psi' \rhd !S \Longrightarrow\hat{}_\tau\ T \wedge (\Psi',\ T,\ S' \parallel !S) \in Rel'$
shows $\Psi \rhd R \parallel !P \rightsquigarrow\!<Rel''>\ R \parallel !Q$
using ‹*eqvt Rel''*›
proof(*induct rule*: *weakSimI*[**where** *C*=()])
case(*cAct* $\Psi'\ \alpha\ RQ'$)
from ‹$bn\ \alpha \sharp* (R \parallel !P)$› ‹$bn\ \alpha \sharp* (R \parallel !Q)$› **have** $bn\ \alpha \sharp* P$ **and** $bn\ \alpha \sharp* (!Q)$ **and** $bn\ \alpha \sharp* Q$ **and** $bn\ \alpha \sharp* R$
by *simp*+
from ‹$\Psi \rhd R \parallel !Q \longmapsto\alpha \prec RQ'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* R$› ‹$bn\ \alpha \sharp* !Q$› ‹$bn\ \alpha \sharp* subject\ \alpha$› ‹$\alpha \neq \tau$› **show** *?case*
proof(*induct rule*: *parCases*[**where** $C=(\Psi',\ P,\ Q,\ R)$])
case(*cPar1* $R'\ A_Q\ \Psi_Q$)
from ‹$extractFrame\ (!Q) = \langle A_Q,\ \Psi_Q\rangle$› **have** $A_Q = []$ **and** $\Psi_Q = SBottom'$ **by** *simp*+
with ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto\alpha \prec R'$› ‹$\Psi_Q = SBottom'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$›
have $\Psi \rhd R \parallel !P \longmapsto\alpha \prec (R' \parallel !P)$ **by**(*rule-tac Par1*) (*assumption* | *simp*)+
hence $\Psi : R \parallel !Q \rhd R \parallel !P \Longrightarrow\alpha \prec R' \parallel !P$ **by**(*rule-tac transitionWeakTransition*) *auto*
moreover have $\Psi \otimes \Psi' \rhd R' \parallel !P \Longrightarrow\hat{}_\tau\ R' \parallel !P$ **by** *auto*
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi \otimes \Psi',\ P,\ Q) \in Rel$ **by**(*rule cExt*)
hence $(\Psi \otimes \Psi',\ R' \parallel !P,\ R' \parallel !Q) \in Rel''$ **using** ‹*guarded P*› ‹*guarded Q*›
by(*rule C1*)
ultimately show *?case* **by** *blast*

next
case(*cPar2* Q' A_R Ψ_R)
from ‹$A_R \sharp* (\Psi', P, Q, R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* \Psi'$ **and** $A_R \sharp* R$ **by** *simp+*
have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
with ‹$bn\ \alpha \sharp* R$› ‹$A_R \sharp* \alpha$› **have** $bn\ \alpha \sharp* \Psi_R$ **by**(*auto dest*: *extractFrameFreshChain*)

obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* \Psi_R$ **and** $A_Q \sharp* A_R$
by(*rule-tac* C=(Ψ, Ψ_R, A_R) **in** *freshFrame*) *auto*
from *FrQ* ‹*guarded* Q› **have** $\Psi_Q \simeq \mathbf{1}$ **and** *supp* Ψ_Q = ({}::*name set*) **by**(*blast dest*: *guardedStatEq*)+
hence $A_R \sharp* \Psi_Q$ **and** $A_Q \sharp* \Psi_Q$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto\alpha \prec Q'$› ‹*guarded* Q› ‹$bn\ \alpha \sharp* Q$› ‹$\alpha \neq \tau$› ‹$bn\ \alpha \sharp*$ *subject* α›
obtain T **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto\alpha \prec T$ **and** $(\mathbf{1}, Q', T \parallel !Q) \in Rel'$
by(*blast dest*: *rBangActE*)

from ‹$(\Psi, P, Q) \in Rel$› **have** $(\Psi \otimes \Psi_R, P, Q) \in Rel$ **by**(*rule cExt*)
with *QTrans* ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* \Psi_R$› ‹$bn\ \alpha \sharp* P$› ‹$\alpha \neq \tau$›
obtain P'' S **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow\alpha \prec P''$
and *P''Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ S$
and *SRelT*: $((\Psi \otimes \Psi_R) \otimes \Psi', S, T) \in Rel$
by(*blast dest*: *cSim weakSimE*)
from *PTrans* **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} : Q \rhd P \Longrightarrow\alpha \prec P''$
by(*metis weakTransitionStatEq Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R : Q \parallel !P \rhd P \parallel !P \Longrightarrow\alpha \prec P'' \parallel !P$ **using** ‹$bn\ \alpha \sharp* P$›
by(*force intro*: *weakPar1*)
hence $\Psi \otimes \Psi_R : Q \parallel !P \rhd !P \Longrightarrow\alpha \prec P'' \parallel !P$ **using** ‹*guarded* P›
by(*rule weakBang*)
hence $\Psi : R \parallel (Q \parallel !P) \rhd R \parallel !P \Longrightarrow\alpha \prec R \parallel (P'' \parallel !P)$
using *FrR* ‹$bn\ \alpha \sharp* R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* Q$›‹$A_R \sharp* P$› ‹$A_R \sharp* \alpha$› ‹$A_R \sharp* \alpha$›
by(*rule-tac weakPar2*) *auto*
moreover have *insertAssertion* (*extractFrame*($R \parallel !Q$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($R \parallel (Q \parallel !P)$)) Ψ
proof −
have *insertAssertion* (*extractFrame*($R \parallel !P$)) $\Psi = \langle A_R, \Psi \otimes \Psi_R \otimes \mathbf{1}\rangle$ **using** *FrR* ‹$A_R \sharp* \Psi$›
by *auto*
moreover from ‹$\Psi_Q \simeq \mathbf{1}$› **have** $\langle A_R, \Psi \otimes \Psi_R \otimes \mathbf{1}\rangle \simeq_F \langle A_R, \Psi \otimes \Psi_R \otimes \Psi_Q \otimes \mathbf{1}\rangle$
by(*rule-tac frameResChainPres*, *auto*) (*metis Identity compositionSym AssertionStatEqTrans AssertionStatEqSym*)
moreover have $\langle A_R, \Psi \otimes \Psi_R \otimes \Psi_Q \otimes \mathbf{1}\rangle \simeq_F (\!|\nu{*}A_Q|\!)(\langle A_R, \Psi \otimes \Psi_R \otimes \Psi_Q \otimes \mathbf{1}\rangle)$ **using** ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi_Q$› ‹$A_Q \sharp* \Psi_R$› ‹$A_Q \sharp* A_R$› *freshCompChain*
by(*subst frameResFreshChain*[**where** *xvec*=A_Q, *THEN FrameStatEqSym*])

auto
moreover have $(\!|\nu{*}A_Q|\!)(\langle A_R,\ \Psi \otimes \Psi_R \otimes \Psi_Q \otimes \mathbf{1}\rangle) \simeq_F (\!|\nu{*}A_R|\!)(\langle A_Q,\ \Psi \otimes \Psi_R \otimes \Psi_Q \otimes \mathbf{1}\rangle)$
by(*rule frameResChainComm*)
moreover have *insertAssertion* (*extractFrame*($R \parallel (Q \parallel !P)$)) $\Psi = \langle (A_R@A_Q),\ \Psi \otimes \Psi_R \otimes \Psi_Q \otimes \mathbf{1}\rangle$
using *FrR FrQ* ‹$A_R \sharp{*}\ \Psi$› ‹$A_Q \sharp{*}\ \Psi$› ‹$A_Q \sharp{*}\ A_R$› ‹$A_Q \sharp{*}\ \Psi_R$› ‹$A_R \sharp{*}\ \Psi_Q$› *freshCompChain*
by *auto*
ultimately have *insertAssertion* (*extractFrame*($R \parallel !P$)) $\Psi \simeq_F$ *insertAssertion* (*extractFrame*($R \parallel (Q \parallel !P)$)) Ψ
by(*auto simp add*: *frameChainAppend*) (*blast dest*: *FrameStatEqTrans*)
thus *?thesis* **by**(*simp add*: *FrameStatEq-def*)
qed
ultimately have $\Psi : R \parallel !Q \rhd R \parallel !P \Longrightarrow \alpha \prec R \parallel (P'' \parallel !P)$
by(*rule weakTransitionFrameImp*)

moreover from *PTrans* ‹$A_R \sharp{*}\ P$› ‹$A_R \sharp{*}\ \alpha$› ‹*bn* $\alpha \sharp{*}$ *subject* α› ‹*distinct*(*bn* α)›
have $A_R \sharp{*}\ P''$ **by**(*force dest*: *weakFreshChainDerivative*)
with *P''Chain* **have** $A_R \sharp{*}\ S$ **by**(*force dest*: *tauChainFreshChain*)
from ‹$\Psi \otimes \Psi_R \rhd Q \longmapsto \alpha \prec T$› ‹$A_R \sharp{*}\ Q$› ‹$A_R \sharp{*}\ \alpha$› ‹*bn* $\alpha \sharp{*}$ *subject* α› ‹*distinct*(*bn* α)› **have** $A_R \sharp{*}\ T$
by(*force dest*: *freeFreshChainDerivative*)

from *P''Chain* **have** $((\Psi \otimes \Psi') \otimes \Psi_R) \otimes \mathbf{1} \rhd P'' \Longrightarrow\hat{}_\tau\ S$
by(*rule tauChainStatEq*) (*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym Identity*)
hence $(\Psi \otimes \Psi') \otimes \Psi_R \rhd P'' \parallel !P \Longrightarrow\hat{}_\tau\ S \parallel !P$
by(*rule-tac tauChainPar1*) *auto*
hence $\Psi \otimes \Psi' \rhd R \parallel (P'' \parallel !P) \Longrightarrow\hat{}_\tau\ R \parallel (S \parallel !P)$ **using** *FrR* ‹$A_R \sharp{*}\ \Psi$› ‹$A_R \sharp{*}\ \Psi'$› ‹$A_R \sharp{*}\ P''$› ‹$A_R \sharp{*}\ P$›
by(*rule-tac tauChainPar2*) *auto*
moreover have $(\Psi \otimes \Psi',\ R \parallel (S \parallel !P),\ R \parallel Q') \in Rel''$
proof −
from ‹$((\Psi \otimes \Psi_R) \otimes \Psi',\ S,\ T) \in Rel$› **have** $((\Psi \otimes \Psi') \otimes \Psi_R,\ S,\ T) \in Rel$
by(*rule StatEq*) (*metis Associativity Commutativity Composition Assertion-StatEqTrans AssertionStatEqSym*)
with *FrR* ‹$A_R \sharp{*}\ \Psi$› ‹$A_R \sharp{*}\ \Psi'$› ‹$A_R \sharp{*}\ S$› ‹$A_R \sharp{*}\ T$› **have** $(\Psi \otimes \Psi',\ R \parallel S,\ R \parallel T) \in Rel$
by(*rule-tac FrameParPres*) *auto*
hence $(\Psi \otimes \Psi',\ R \parallel T,\ R \parallel S) \in Rel$ **by**(*rule cSym*)
hence $(\Psi \otimes \Psi',\ (R \parallel T) \parallel !P,\ (R \parallel S) \parallel !P) \in Rel$ **by**(*rule ParPres*)
hence $(\Psi \otimes \Psi',\ (R \parallel S) \parallel !P,\ (R \parallel T) \parallel !P) \in Rel$ **by**(*rule cSym*)
hence $(\Psi \otimes \Psi',\ R \parallel (S \parallel !P),\ (R \parallel T) \parallel !P) \in Rel$ **by**(*metis Trans Assoc*)
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi \otimes \Psi',\ P,\ Q) \in Rel$ **by**(*rule cExt*)
hence $(\Psi \otimes \Psi',\ (R \parallel T) \parallel !P,\ (R \parallel T) \parallel !Q) \in Rel''$ **using** ‹*guarded P*› ‹*guarded Q*› **by**(*rule C1*)
moreover from ‹$(\mathbf{1},\ Q',\ T \parallel !Q) \in Rel'$› **have** $(\mathbf{1} \otimes \Psi \otimes \Psi',\ Q',\ T \parallel !Q) \in$

Rel′ **by**(*rule cExt′*)
hence $(\Psi \otimes \Psi', Q', T \parallel !Q) \in Rel'$
by(*rule StatEq′*) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence $(\Psi \otimes \Psi', R \parallel Q', R \parallel (T \parallel !Q)) \in Rel'$ **by**(*rule ParPres′*)
hence $(\Psi \otimes \Psi', R \parallel Q', (R \parallel T) \parallel !Q) \in Rel'$ **by**(*metis Trans′ Assoc′*)
hence $(\Psi \otimes \Psi', (R \parallel T) \parallel !Q, R \parallel Q') \in Rel'$ **by**(*rule cSym′*)
ultimately show *?thesis* **by**(*rule-tac Compose*)
qed
ultimately show *?case* **by** *blast*
next
case *cComm1*
from ‹$\tau \neq \tau$› **have** *False* **by** *simp*
thus *?case* **by** *simp*
next
case *cComm2*
from ‹$\tau \neq \tau$› **have** *False* **by** *simp*
thus *?case* **by** *simp*
qed
next
case(*cTau RQ′*)
from ‹$\Psi \rhd R \parallel !Q \longmapsto\tau \prec RQ'$› **show** *?case*
proof(*induct rule*: *parTauCases*[**where** C=(P, Q, R)])
case(*cPar1 R′ A_Q Ψ_Q*)
from ‹*extractFrame* $(!Q) = \langle A_Q, \Psi_Q\rangle$› **have** $A_Q = []$ **and** $\Psi_Q = SBottom'$ **by** *simp*+
with ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto\tau \prec R'$› ‹$\Psi_Q = SBottom'$›
have $\Psi \rhd R \parallel !P \longmapsto\tau \prec (R' \parallel !P)$ **by**(*rule-tac Par1*) (*assumption* | *simp*)+
hence $\Psi \rhd R \parallel !P \Longrightarrow\hat{}_\tau R' \parallel !P$ **by** *auto*
moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded P*› ‹*guarded Q*› **have** $(\Psi, R' \parallel !P, R' \parallel !Q) \in Rel''$
by(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 Q′ A_R Ψ_R*)
from ‹$A_R \sharp* (P, Q, R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* R$ **by** *simp*+
have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*

obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **and** $A_Q \sharp* \Psi$ **and** $A_Q \sharp* \Psi_R$ **and** $A_Q \sharp* A_R$
by(*rule-tac* C=(Ψ, Ψ_R, A_R) **in** *freshFrame*) *auto*
from *FrQ* ‹*guarded Q*› **have** $\Psi_Q \simeq \mathbf{1}$ **and** *supp* Ψ_Q = ({}::*name set*) **by**(*blast dest*: *guardedStatEq*)+
hence $A_R \sharp* \Psi_Q$ **and** $A_Q \sharp* \Psi_Q$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto\tau \prec Q'$› ‹*guarded Q*›
obtain T **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \parallel Q \longmapsto\tau \prec T$ **and** $(\mathbf{1}, Q', T \parallel !Q) \in Rel'$
by(*blast dest*: *rBangTauE*)

from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi \otimes \Psi_R,\ P,\ Q) \in Rel$ **by**(*rule cExt*)
hence $(\Psi \otimes \Psi_R,\ P \parallel P,\ Q \parallel Q) \in Rel$ **by**(*rule ParPres2*)
with *QTrans*
obtain S **where** *PTrans*: $\Psi \otimes \Psi_R \rhd P \parallel P \Longrightarrow\hat{}_\tau\ S$ **and** *SRelT*: $(\Psi \otimes \Psi_R,\ S,\ T) \in Rel$
by(*blast dest*: *cSim weakSimE*)
from *PTrans* ‹*guarded P*› **obtain** U **where** *PChain*: $\Psi \otimes \Psi_R \rhd !P \Longrightarrow\hat{}_\tau\ U$ **and** $(\Psi \otimes \Psi_R,\ U,\ S \parallel !P) \in Rel'$
by(*blast dest*: *rBangTauI*)
from *PChain* ‹$A_R \sharp* P$› **have** $A_R \sharp* U$ **by**(*force dest*: *tauChainFreshChain*)
from ‹$\Psi \otimes \Psi_R \rhd !P \Longrightarrow\hat{}_\tau\ U$› *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› **have** $\Psi \rhd R \parallel !P \Longrightarrow\hat{}_\tau\ R \parallel U$
by(*rule-tac tauChainPar2*) *auto*
moreover from *PTrans* ‹$A_R \sharp* P$› **have** $A_R \sharp* S$ **by**(*force dest*: *tauChainFreshChain*)
from *QTrans* ‹$A_R \sharp* Q$› **have** $A_R \sharp* T$ **by**(*force dest*: *tauFreshChainDerivative*)
have $(\Psi,\ R \parallel U,\ R \parallel Q') \in Rel''$
proof −
from ‹$(\Psi \otimes \Psi_R,\ U,\ S \parallel !P) \in Rel'$› *Rel'Rel* **have** $(\Psi \otimes \Psi_R,\ U,\ S \parallel !P) \in Rel$
by *auto*
hence $(\Psi,\ R \parallel U,\ R \parallel (S \parallel !P)) \in Rel$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* U$› ‹$A_R \sharp* S$› ‹$A_R \sharp* P$›
by(*rule-tac FrameParPres*) *auto*

moreover from ‹$(\Psi \otimes \Psi_R,\ S,\ T) \in Rel$› *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* S$› ‹$A_R \sharp* T$› **have** $(\Psi,\ R \parallel S,\ R \parallel T) \in Rel$
by(*rule-tac FrameParPres*) *auto*
hence $(\Psi,\ R \parallel T,\ R \parallel S) \in Rel$ **by**(*rule cSym*)
hence $(\Psi,\ (R \parallel T) \parallel !P,\ (R \parallel S) \parallel !P) \in Rel$ **by**(*rule ParPres*)
hence $(\Psi,\ (R \parallel S) \parallel !P,\ (R \parallel T) \parallel !P) \in Rel$ **by**(*rule cSym*)
hence $(\Psi,\ R \parallel (S \parallel !P),\ (R \parallel T) \parallel !P) \in Rel$ **by**(*metis Trans Assoc*)
ultimately have $(\Psi,\ R \parallel U,\ (R \parallel T) \parallel !P) \in Rel$ **by**(*rule Trans*)
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi,\ (R \parallel T) \parallel !P,\ (R \parallel T) \parallel !Q) \in Rel''$ **using** ‹*guarded P*› ‹*guarded Q*› **by**(*rule C1*)
moreover from ‹$(\mathbf{1},\ Q',\ T \parallel !Q) \in Rel'$› **have** $(\mathbf{1} \otimes \Psi,\ Q',\ T \parallel !Q) \in Rel'$ **by**(*rule cExt'*)
hence $(\Psi,\ Q',\ T \parallel !Q) \in Rel'$
by(*rule StatEq'*) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence $(\Psi,\ R \parallel Q',\ R \parallel (T \parallel !Q)) \in Rel'$ **by**(*rule ParPres'*)
hence $(\Psi,\ R \parallel Q',\ (R \parallel T) \parallel !Q) \in Rel'$ **by**(*metis Trans' Assoc'*)
hence $(\Psi,\ (R \parallel T) \parallel !Q,\ R \parallel Q') \in Rel'$ **by**(*rule cSym'*)
ultimately show *?thesis* **by**(*rule-tac Compose*)
qed
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_Q M N R' A_R Ψ_R K *xvec* Q' A_Q)

from ‹A_R ♯* (P, Q, R)› **have** A_R ♯* P **and** A_R ♯* Q **and** A_R ♯* R **by** *simp*+
from ‹*xvec* ♯* (P, Q, R)› **have** *xvec* ♯* P **and** *xvec* ♯* Q **and** *xvec* ♯* R **by** *simp*+
have *FrQ*: *extractFrame*(!Q) = ⟨A_Q, Ψ_Q⟩ **by** *fact*
have *FrR*: *extractFrame* R = ⟨A_R, Ψ_R⟩ **by** *fact*
from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$› *FrR* ‹*distinct* A_R› ‹A_R ♯* R› ‹A_R ♯* N› ‹A_R ♯* *xvec*› ‹A_R ♯* P› ‹A_R ♯* Q› ‹A_R ♯* Ψ› ‹A_R ♯* M›
obtain $A_R{}'$ $\Psi_R{}'$ Ψ' **where** *FrR'*: *extractFrame* R' = ⟨$A_R{}'$, $\Psi_R{}'$⟩ **and** $\Psi_R \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}'$ ♯* *xvec* **and** $A_R{}'$ ♯* P **and** $A_R{}'$ ♯* Q **and** $A_R{}'$ ♯* Ψ
by(*rule-tac* C=(Ψ, *xvec*, P, Q) **and** C'=(Ψ, *xvec*, P, Q) **in** *expandFrame*) *auto*
from ‹(Ψ, P, Q) ∈ *Rel*› **have** ($\Psi \otimes \Psi_R$, P, Q) ∈ *Rel* **by**(*rule cExt*)
moreover from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$› ‹*guarded* Q› ‹*xvec* ♯* Q› ‹*xvec* ♯* K›
obtain S **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec S$ **and** (**1**, Q', S ‖ !Q) ∈ *Rel'*
by(*fastforce dest*: *rBangActE*)
ultimately obtain P' T **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **and** *P'Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau T$ **and** $((\Psi \otimes \Psi_R) \otimes \Psi', T, S)$ ∈ *Rel*
using ‹*xvec* ♯* Ψ› ‹*xvec* ♯* Ψ_R› ‹*xvec* ♯* P›
by(*fastforce dest*: *cSim weakSimE*)

from *PTrans* ‹A_R ♯* P› ‹A_R ♯* *xvec*› ‹$A_R{}'$ ♯* P› ‹$A_R{}'$ ♯* *xvec*› ‹*xvec* ♯* K› ‹*distinct xvec*›
have A_R ♯* P' **and** $A_R{}'$ ♯* P'
by(*force dest*: *weakOutputFreshChainDerivative*)+
with *P'Chain* **have** $A_R{}'$ ♯* T **by**(*force dest*: *tauChainFreshChain*)+
from *QTrans* ‹$A_R{}'$ ♯* Q› ‹$A_R{}'$ ♯* *xvec*› ‹*xvec* ♯* K› ‹*distinct xvec*›
have $A_R{}'$ ♯* S **by**(*force dest*: *outputFreshChainDerivative*)

obtain $A_Q{}'$ $\Psi_Q{}'$ **where** *FrQ'*: *extractFrame* Q = ⟨$A_Q{}'$, $\Psi_Q{}'$⟩ **and** $A_Q{}'$ ♯* Ψ **and** $A_Q{}'$ ♯* Ψ_R **and** $A_Q{}'$ ♯* A_R **and** $A_Q{}'$ ♯* M **and** $A_Q{}'$ ♯* R **and** $A_Q{}'$ ♯* K
by(*rule-tac* C=(Ψ, Ψ_R, A_R, K, M, R) **in** *freshFrame*) *auto*
from *FrQ'* ‹*guarded* Q› **have** $\Psi_Q{}' \simeq$ **1** **and** *supp* $\Psi_Q{}'$ = ({}::*name set*) **by**(*blast dest*: *guardedStatEq*)+
hence $A_Q{}'$ ♯* $\Psi_Q{}'$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P''$
and *NilImpP''*: ⟨$A_Q{}'$, $(\Psi \otimes \Psi_R) \otimes \Psi_Q{}'$⟩ $\hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') $(\Psi \otimes \Psi_R)$
and *P''Trans*: $\Psi \otimes \Psi_R \rhd P'' \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
using *FrQ'* ‹$A_Q{}'$ ♯* Ψ› ‹$A_Q{}'$ ♯* Ψ_R› *freshCompChain*
by(*drule-tac weakTransitionE*) *auto*

from *PChain* **have** $\Psi \rhd R$ ‖ !$P \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)(R'$ ‖ (P' ‖ !P))
proof(*induct rule*: *tauChainCases*)
case *TauBase*
from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$› *FrQ* **have** $\Psi \otimes$ **1** $\rhd R \longmapsto M(\!|N|\!) \prec R'$

by *simp*
moreover note *FrR*
moreover from *P''Trans* ‹$P = P''$› **have** $\Psi \otimes \Psi_R \rhd P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by** *simp*
hence $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$ **by**(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec (P' \parallel !P)$ **using** ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* \Psi_R$› ‹$xvec \sharp* P$›
by(*force intro*: *Par1*)
hence $\Psi \otimes \Psi_R \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec (P' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover from ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› *FrQ* **have** $\Psi \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K$ **by** *simp*
ultimately have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(R' \parallel (P' \parallel !P))$ **using** ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* R$› ‹$A_R \sharp* P$› ‹$A_R \sharp* M$› ‹$xvec \sharp* R$›
by(*force intro*: *Comm1*)
thus *?case* **by**(*rule tauActTauChain*)
next
case *TauStep*
obtain A_P'' Ψ_P'' **where** *FrP''*: *extractFrame* $P'' = \langle A_P'', \Psi_P''\rangle$ **and** $A_P'' \sharp* \Psi$ **and** $A_P'' \sharp* K$ **and** $A_P'' \sharp* \Psi_R$ **and** $A_P'' \sharp* R$ **and** $A_P'' \sharp* P''$ **and** $A_P'' \sharp* P$
and $A_P'' \sharp* A_R$ **and** *distinct* A_P''
by(*rule-tac* $C=(\Psi, K, A_R, \Psi_R, R, P'', P)$ **in** *freshFrame*) *auto*
from *PChain* ‹$A_R \sharp* P$› **have** $A_R \sharp* P''$ **by**(*drule-tac tauChainFreshChain*) *auto*
with *FrP''* ‹$A_P'' \sharp* A_R$› **have** $A_R \sharp* \Psi_P''$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from ‹$\Psi \otimes \Psi_R \rhd P \Longrightarrow_\tau P''$› **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \Longrightarrow_\tau P''$ **by**(*rule tauStepChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \Longrightarrow_\tau P'' \parallel !P$ **by**(*rule-tac tauStepChainPar1*) *auto*
hence $\Psi \otimes \Psi_R \rhd !P \Longrightarrow_\tau P'' \parallel !P$ **using** ‹*guarded P*› **by**(*rule tauStepChainBang*)
hence $\Psi \rhd R \parallel !P \Longrightarrow_\tau R \parallel (P'' \parallel !P)$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$›
by(*rule-tac tauStepChainPar2*) *auto*
moreover have $\Psi \rhd R \parallel (P'' \parallel !P) \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(R' \parallel (P' \parallel !P))$
proof −
from *FrQ* ‹$\Psi_Q' \simeq \mathbf{1}$› ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$› **have** $\Psi \otimes \Psi_Q' \rhd R \longmapsto M(\!|N|\!) \prec R'$
by *simp* (*metis statEqTransition AssertionStatEqSym compositionSym*)
moreover from *P''Trans* **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P'' \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence *P''PTrans*: $\Psi \otimes \Psi_R \rhd P'' \parallel !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec (P' \parallel !P)$ **using** ‹$xvec \sharp* P$›
by(*rule-tac Par1*) *auto*
moreover from *FrP''* **have** *FrP''P*: *extractFrame*$(P'' \parallel !P) = \langle A_P'', \Psi_P'' \otimes \mathbf{1}\rangle$
by *auto*

moreover from *FrQ* ‹$\Psi_Q{}' \simeq \mathbf{1}$› ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_Q{}' \otimes \Psi_R \vdash M \leftrightarrow K$
by *simp* (*metis statEqEnt Composition AssertionStatEqSym Commutativity*)
hence $\Psi \otimes \Psi_Q{}' \otimes \Psi_R \vdash K \leftrightarrow M$ **by**(*rule chanEqSym*)
moreover have $\langle A_Q{}', (\Psi \otimes \Psi_Q{}') \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1})) \otimes \Psi_R\rangle$
proof −
have $\langle A_Q{}', (\Psi \otimes \Psi_Q{}') \otimes \Psi_R\rangle \simeq_F \langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle$
by(*rule-tac frameResChainPres*, *simp*)
(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)
moreover from *NilImpP'' FrQ FrP''* ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* \Psi_R$› *freshCompChain* **have** $\langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_R) \otimes \Psi_P{}''\rangle$
by *auto*
moreover have $\langle A_P{}'', (\Psi \otimes \Psi_R) \otimes \Psi_P{}''\rangle \simeq_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R\rangle$
by(*rule frameResChainPres*, *simp*)
(*metis Identity AssertionStatEqSym Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
ultimately obtain M' **where** *RTrans*: $\Psi \otimes \Psi_P{}'' \otimes \mathbf{1} \rhd R \longmapsto M'(\!|N|\!) \prec R'$ **and** $\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$ **and** $A_R \sharp* M'$
using *FrR FrQ'* ‹*distinct* A_R› ‹*distinct* $A_P{}''$› ‹$A_P{}'' \sharp* A_R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P''$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* M$› ‹$A_Q{}' \sharp* R$› ‹$A_Q{}' \sharp* K$› ‹$A_Q{}' \sharp* A_R$› ‹$A_R \sharp* P$› ‹$A_P{}'' \sharp* P$› ‹$A_R \sharp*$ *xvec*›
‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* R$› ‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* K$› ‹*xvec* $\sharp* K$› ‹*distinct xvec*›
by(*rule-tac* $A_Q{=}A_Q{}'$ **and** $Q{=}Q$ **in** *comm2Aux*) (*assumption* | *simp*)+

note *RTrans FrR P''PTrans FrP''P*
moreover from ‹$\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$› **have** $\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash M' \leftrightarrow K$ **by**(*rule chanEqSym*)
hence $\Psi \otimes \Psi_R \otimes \Psi_P{}'' \otimes \mathbf{1} \vdash M' \leftrightarrow K$ **by**(*metis statEqEnt Composition AssertionStatEqSym Commutativity*)
ultimately show *?thesis* **using** ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* R$› ‹$A_R \sharp* P''$› ‹$A_R \sharp* P$› ‹$A_R \sharp* M'$› ‹$A_P{}'' \sharp* A_R$› ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* R$› ‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* P$› ‹$A_P{}'' \sharp* K$› ‹*xvec* $\sharp* R$›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
qed
ultimately show *?thesis*
by(*drule-tac tauActTauChain*) *auto*
qed

moreover from *P'Chain* **have** $((\Psi \otimes \Psi_R) \otimes \Psi') \otimes \mathbf{1} \rhd P' \Longrightarrow\hat{}_\tau T$
by(*rule tauChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P' \parallel !P \Longrightarrow\hat{}_\tau T \parallel !P$
by(*rule-tac tauChainPar1*) *auto*
hence $\Psi \otimes \Psi_R \otimes \Psi' \rhd P' \parallel !P \Longrightarrow\hat{}_\tau T \parallel !P$

by(*rule tauChainStatEq*) (*metis Associativity*)
hence $\Psi \otimes {\Psi_R}' \rhd P' \parallel !P \Longrightarrow\hat{}_\tau\ T \parallel !P$ **using** ‹$\Psi_R \otimes \Psi' \simeq {\Psi_R}'$›
by(*rule-tac tauChainStatEq*) (*auto intro*: *compositionSym*)
hence $\Psi \rhd R' \parallel (P' \parallel !P) \Longrightarrow\hat{}_\tau\ R' \parallel (T \parallel !P)$ **using** *FrR'* ‹${A_R}' \sharp* \Psi$› ‹${A_R}' \sharp* P$› ‹${A_R}' \sharp* P'$›
by(*rule-tac tauChainPar2*) *auto*
hence $\Psi \rhd (\!|\nu*xvec|\!)(R' \parallel (P' \parallel !P)) \Longrightarrow\hat{}_\tau (\!|\nu*xvec|\!)(R' \parallel (T \parallel !P))$ **using** ‹$xvec \sharp* \Psi$›
by(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd R \parallel !P \Longrightarrow\hat{}_\tau (\!|\nu*xvec|\!)(R' \parallel (T \parallel !P))$
by *auto*
moreover have $(\Psi, (\!|\nu*xvec|\!)(R' \parallel (T \parallel !P)), (\!|\nu*xvec|\!)(R' \parallel Q')) \in Rel''$
proof −
from ‹$((\Psi \otimes \Psi_R) \otimes \Psi', T, S) \in Rel$› **have** $(\Psi \otimes \Psi_R \otimes \Psi', T, S) \in Rel$
by(*rule StatEq*) (*metis Associativity*)
hence $(\Psi \otimes {\Psi_R}', T, S) \in Rel$ **using** ‹$\Psi_R \otimes \Psi' \simeq {\Psi_R}'$›
by(*rule-tac StatEq*) (*auto dest*: *compositionSym*)

with *FrR'* ‹${A_R}' \sharp* \Psi$› ‹${A_R}' \sharp* S$› ‹${A_R}' \sharp* T$› **have** $(\Psi, R' \parallel T, R' \parallel S) \in Rel$
by(*rule-tac FrameParPres*) *auto*
hence $(\Psi, (R' \parallel T) \parallel !P, (R' \parallel S) \parallel !P) \in Rel$ **by**(*rule ParPres*)
hence $(\Psi, (\!|\nu*xvec|\!)((R' \parallel T) \parallel !P), (\!|\nu*xvec|\!)((R' \parallel S) \parallel !P)) \in Rel$ **using** ‹$xvec \sharp* \Psi$›
by(*rule ResPres*)
hence $(\Psi, (\!|\nu*xvec|\!)(R' \parallel T) \parallel !P, ((\!|\nu*xvec|\!)(R' \parallel S)) \parallel !P) \in Rel$ **using** ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$›
by(*force intro*: *Trans ScopeExt*)
hence $(\Psi, (\!|\nu*xvec|\!)(R' \parallel (T \parallel !P)), ((\!|\nu*xvec|\!)(R' \parallel S)) \parallel !P) \in Rel$ **using** ‹$xvec \sharp* \Psi$›
by(*force intro*: *Trans ResPres Assoc*)

moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded P*› ‹*guarded Q*› **have** $(\Psi, ((\!|\nu*xvec|\!)(R' \parallel S)) \parallel !P, ((\!|\nu*xvec|\!)(R' \parallel S)) \parallel !Q) \in Rel''$
by(*rule C1*)
moreover from ‹$(\mathbf{1}, Q', S \parallel !Q) \in Rel'$› **have** $(\mathbf{1} \otimes \Psi, Q', S \parallel !Q) \in Rel'$ **by**(*rule cExt'*)
hence $(\Psi, Q', S \parallel !Q) \in Rel'$
by(*rule StatEq'*) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence $(\Psi, R' \parallel Q', R' \parallel (S \parallel !Q)) \in Rel'$ **by**(*rule ParPres'*)
hence $(\Psi, R' \parallel Q', (R' \parallel S) \parallel !Q) \in Rel'$ **by**(*metis Trans' Assoc'*)
hence $(\Psi, (R' \parallel S) \parallel !Q, R' \parallel Q') \in Rel'$ **by**(*rule cSym'*)
hence $(\Psi, (\!|\nu*xvec|\!)((R' \parallel S) \parallel !Q), (\!|\nu*xvec|\!)(R' \parallel Q')) \in Rel'$ **using** ‹$xvec \sharp* \Psi$›
by(*rule ResPres'*)
hence $(\Psi, ((\!|\nu*xvec|\!)(R' \parallel S)) \parallel !Q, (\!|\nu*xvec|\!)(R' \parallel Q')) \in Rel'$ **using** ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* Q$›
by(*force intro*: *Trans' ScopeExt'*[*THEN cSym'*])
ultimately show *?thesis* **by**(*rule-tac Compose*)

qed
ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_Q M *xvec* N R' A_R Ψ_R K Q' A_Q)
from ‹$A_R \sharp* (P, Q, R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* R$ **by** *simp*+
from ‹*xvec* $\sharp* (P, Q, R)$› **have** *xvec* $\sharp* P$ **and** *xvec* $\sharp* Q$ **and** *xvec* $\sharp* R$ **by** *simp*+
have *FrQ*: *extractFrame*$(!Q) = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R \sharp* R$› ‹$A_R \sharp* N$› ‹$A_R \sharp*$ *xvec*› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* \Psi$› ‹*xvec* $\sharp* R$› ‹*xvec* $\sharp* \Psi$› ‹*xvec* $\sharp* P$› ‹*xvec* $\sharp* Q$› ‹*xvec* $\sharp* M$› ‹*distinct xvec*› ‹$A_R \sharp* M$›
obtain p $A_R{}'$ $\Psi_R{}'$ Ψ' **where** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$ **and** $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}' \sharp*$ *xvec* **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q$ **and** $A_R{}' \sharp* \Psi$ **and** S: *set* $p \subseteq$ *set xvec* $\times$ *set*$(p \cdot$ *xvec*$)$ **and** *distinctPerm* p **and** $(p \cdot$ *xvec*$) \sharp* N$ **and** $(p \cdot$ *xvec*$) \sharp* Q$ **and** $(p \cdot$ *xvec*$) \sharp* R'$ **and** $(p \cdot$ *xvec*$) \sharp* P$ **and** $(p \cdot$ *xvec*$) \sharp* \Psi$ **and** $A_R{}' \sharp* N$ **and** $A_R{}' \sharp*$ *xvec* **and** $A_R{}' \sharp* (p \cdot$ *xvec*$)$
by(*rule-tac* C=(Ψ, P, Q) **and** C'=(Ψ, P, Q) **in** *expandFrame*) (*assumption* | *simp*)+

from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec R'$› S ‹$(p \cdot$ *xvec*$) \sharp* N$› ‹$(p \cdot$ *xvec*$) \sharp* R'$›
have *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$
by(*simp add*: *boundOutputChainAlpha''* *residualInject*)
from ‹$(\Psi, P, Q) \in$ *Rel*› **have** $(\Psi \otimes \Psi_R, P, Q) \in$ *Rel* **by**(*rule cExt*)
moreover from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|N|\!) \prec Q'$› S ‹$(p \cdot$ *xvec*$) \sharp* Q$› ‹*xvec* $\sharp* Q$› ‹*distinctPerm* p›
have $\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|(p \cdot N)|\!) \prec (p \cdot Q')$
by(*rule-tac inputAlpha*) *auto*
then obtain S **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto K(\!|(p \cdot N)|\!) \prec S$ **and** $(\mathbf{1}, (p \cdot Q'), S \parallel !Q) \in Rel'$
using ‹*guarded* Q›
by(*fastforce dest*: *rBangActE*)
ultimately obtain P' T **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow K(\!|(p \cdot N)|\!) \prec P'$
and *P'Chain*: $(\Psi \otimes \Psi_R) \otimes (p \cdot \Psi') \rhd P' \Longrightarrow\hat{}_\tau\ T$
and $((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'), T, S) \in Rel$
by(*fastforce dest*: *cSim weakSimE*)

from ‹$A_R{}' \sharp* N$› ‹$A_R{}' \sharp*$ *xvec*› ‹$A_R{}' \sharp* (p \cdot$ *xvec*$)$› S **have** $A_R{}' \sharp* (p \cdot N)$
by(*simp add*: *freshChainSimps*)
with *PTrans* ‹$A_R{}' \sharp* P$› **have** $A_R{}' \sharp* P'$ **by**(*force dest*: *weakInputFreshChainDerivative*)
with *P'Chain* **have** $A_R{}' \sharp* T$ **by**(*force dest*: *tauChainFreshChain*)+
from *QTrans* ‹$A_R{}' \sharp* Q$› ‹$A_R{}' \sharp* (p \cdot N)$› **have** $A_R{}' \sharp* S$ **by**(*force dest*: *inputFreshChainDerivative*)

obtain $A_Q{}'$ $\Psi_Q{}'$ **where** *FrQ'*: *extractFrame* $Q = \langle A_Q{}', \Psi_Q{}'\rangle$ **and** $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* \Psi_R$ **and** $A_Q{}' \sharp* A_R$ **and** $A_Q{}' \sharp* M$ **and** $A_Q{}' \sharp* R$ **and** $A_Q{}' \sharp* K$

by(*rule-tac* C=(Ψ, Ψ_R, A_R, K, M, R) **in** *freshFrame*) *auto*
from *FrQ'* ‹*guarded Q*› **have** $\Psi_Q{}' \simeq \mathbf{1}$ **and** *supp* $\Psi_Q{}' = (\{\}::name\ set)$ **by**(*blast dest*: *guardedStatEq*)+
hence $A_Q{}' \sharp* \Psi_Q{}'$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P''$
and *NilImpP''*: $\langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') $(\Psi \otimes \Psi_R)$
and *P''Trans*: $\Psi \otimes \Psi_R \rhd P'' \longmapsto K(\!|(p \cdot N)|\!) \prec P'$
using *FrQ'* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* \Psi_R$› *freshCompChain*
by(*drule-tac weakTransitionE*) *auto*

from ‹$(p \cdot xvec) \sharp* P$› ‹$xvec \sharp* P$› *PChain* **have** $(p \cdot xvec) \sharp* P''$ **and** $xvec \sharp* P''$
by(*force dest*: *tauChainFreshChain*)+
from ‹$(p \cdot xvec) \sharp* N$› ‹*distinctPerm p*› **have** $xvec \sharp* (p \cdot N)$
by(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, **where** *pi*=*p*, *symmetric*]) *simp*
with *P''Trans* ‹$xvec \sharp* P''$› **have** $xvec \sharp* P'$ **by**(*force dest*: *inputFreshChainDerivative*)
hence $(p \cdot xvec) \sharp* (p \cdot P')$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])

from *PChain* **have** $\Psi \rhd R \parallel !P \Longrightarrow\hat{}_\tau (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel (P' \parallel !P))$
proof(*induct rule*: *tauChainCases*)
case *TauBase*
from *RTrans FrQ* **have** $\Psi \otimes \mathbf{1} \rhd R \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$ **by** *simp*
moreover note *FrR*
moreover from *P''Trans* ‹$P = P''$› **have** $\Psi \otimes \Psi_R \rhd P \longmapsto K(\!|(p \cdot N)|\!) \prec P'$ **by** *simp*
hence $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \longmapsto K(\!|(p \cdot N)|\!) \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \longmapsto K(\!|(p \cdot N)|\!) \prec (P' \parallel !P)$ **by**(*force intro*: *Par1*)
hence $\Psi \otimes \Psi_R \rhd !P \longmapsto K(\!|(p \cdot N)|\!) \prec (P' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover from ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› *FrQ* **have** $\Psi \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K$ **by** *simp*
ultimately have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel (P' \parallel !P))$ **using** ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* R$› ‹$A_R \sharp* P$› ‹$A_R \sharp* M$› ‹$(p \cdot xvec) \sharp* P$›
by(*force intro*: *Comm2*)
thus *?case* **by**(*rule tauActTauChain*)
next
case *TauStep*
obtain $A_P{}''$ $\Psi_P{}''$ **where** *FrP''*: *extractFrame* $P'' = \langle A_P{}'', \Psi_P{}''\rangle$ **and** $A_P{}'' \sharp* \Psi$ **and** $A_P{}'' \sharp* K$ **and** $A_P{}'' \sharp* \Psi_R$ **and** $A_P{}'' \sharp* R$ **and** $A_P{}'' \sharp* P''$ **and** $A_P{}'' \sharp* P$
and $A_P{}'' \sharp* A_R$ **and** *distinct* $A_P{}''$
by(*rule-tac* C=(Ψ, K, A_R, Ψ_R, R, P'', P) **in** *freshFrame*) *auto*

from *PChain* ‹$A_R \sharp* P$› **have** $A_R \sharp* P''$ **by**(*drule-tac tauChainFreshChain*) *auto*

with *FrP''* ‹$A_P{}'' \sharp* A_R$› **have** $A_R \sharp* \Psi_P{}''$ **by**(*drule-tac extractFrameFreshChain*) *auto*

from ‹$\Psi \otimes \Psi_R \rhd P \Longrightarrow_\tau P''$› **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \Longrightarrow_\tau P''$ **by**(*rule tauStepChainStatEq*) (*metis Identity AssertionStatEqSym*)

hence $\Psi \otimes \Psi_R \rhd P \parallel !P \Longrightarrow_\tau P'' \parallel !P$ **by**(*rule-tac tauStepChainPar1*) *auto*

hence $\Psi \otimes \Psi_R \rhd !P \Longrightarrow_\tau P'' \parallel !P$ **using** ‹*guarded P*› **by**(*rule tauStepChainBang*)

hence $\Psi \rhd R \parallel !P \Longrightarrow_\tau R \parallel (P'' \parallel !P)$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$›
by(*rule-tac tauStepChainPar2*) *auto*

moreover have $\Psi \rhd R \parallel (P'' \parallel !P) \longmapsto\tau \prec (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel (P' \parallel !P))$

proof −

from *FrQ* ‹$\Psi_Q{}' \simeq \mathbf{1}$› *RTrans* **have** $\Psi \otimes \Psi_Q{}' \rhd R \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$

by *simp* (*metis statEqTransition AssertionStatEqSym compositionSym*)

moreover from *P''Trans* **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P'' \longmapsto K(\!|(p \cdot N)|\!) \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)

hence *P''PTrans*: $\Psi \otimes \Psi_R \rhd P'' \parallel !P \longmapsto K(\!|(p \cdot N)|\!) \prec (P' \parallel !P)$
by(*rule-tac Par1*) *auto*

moreover from *FrP''* **have** *FrP''P*: $extractFrame(P'' \parallel !P) = \langle A_P{}'', \Psi_P{}'' \otimes \mathbf{1}\rangle$

by *auto*

moreover from *FrQ* ‹$\Psi_Q{}' \simeq \mathbf{1}$› ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_Q{}' \otimes \Psi_R \vdash M \leftrightarrow K$

by *simp* (*metis statEqEnt Composition AssertionStatEqSym Commutativity*)

hence $\Psi \otimes \Psi_Q{}' \otimes \Psi_R \vdash K \leftrightarrow M$ **by**(*rule chanEqSym*)

moreover have $\langle A_Q{}', (\Psi \otimes \Psi_Q{}') \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1})) \otimes \Psi_R\rangle$

proof −

have $\langle A_Q{}', (\Psi \otimes \Psi_Q{}') \otimes \Psi_R\rangle \simeq_F \langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle$
by(*rule-tac frameResChainPres*, *simp*)
(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)

moreover from *NilImpP''* *FrQ* *FrP''* ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* \Psi_R$› *freshCompChain* **have** $\langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_R) \otimes \Psi_P{}''\rangle$

by *auto*

moreover have $\langle A_P{}'', (\Psi \otimes \Psi_R) \otimes \Psi_P{}''\rangle \simeq_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R\rangle$

by(*rule frameResChainPres*, *simp*)
(*metis Identity AssertionStatEqSym Associativity Commutativity Composition AssertionStatEqTrans*)

ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)

qed

ultimately obtain M' **where** *RTrans*: $\Psi \otimes \Psi_P{}'' \otimes \mathbf{1} \rhd R \longmapsto M'(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$ **and** $\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$ **and** $A_R \sharp* M'$

using *FrR* *FrQ'* ‹*distinct* A_R› ‹*distinct* $A_P{}''$› ‹$A_P{}'' \sharp* A_R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P''$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* M$› ‹$A_Q{}' \sharp* R$› ‹$A_Q{}' \sharp* K$› ‹$A_Q{}' \sharp* A_R$› ‹A_R

$\sharp*$ P› ‹$A_P{}''$ $\sharp*$ P›
‹$A_P{}''$ $\sharp*$ Ψ› ‹$A_P{}''$ $\sharp*$ R› ‹$A_P{}''$ $\sharp*$ P''› ‹$A_P{}''$ $\sharp*$ K›
by(*rule-tac* A_Q=$A_Q{}'$ **and** Q=Q **in** *comm1Aux*) (*assumption* | *simp*)+

note *RTrans FrR P''PTrans FrP''P*
moreover from ‹$\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$› **have** $\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash M' \leftrightarrow K$ **by**(*rule chanEqSym*)
hence $\Psi \otimes \Psi_R \otimes \Psi_P{}'' \otimes \mathbf{1} \vdash M' \leftrightarrow K$ **by**(*metis statEqEnt Composition AssertionStatEqSym Commutativity*)
ultimately show *?thesis* **using** ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ R› ‹A_R $\sharp*$ P''› ‹A_R $\sharp*$ P› ‹A_R $\sharp*$ M'› ‹$A_P{}''$ $\sharp*$ A_R› ‹$A_P{}''$ $\sharp*$ Ψ› ‹$A_P{}''$ $\sharp*$ R› ‹$A_P{}''$ $\sharp*$ P''› ‹$A_P{}''$ $\sharp*$ P› ‹$A_P{}''$ $\sharp*$ K› ‹$(p \cdot xvec)$ $\sharp*$ P''› ‹$(p \cdot xvec)$ $\sharp*$ P›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
qed
ultimately show *?thesis*
by(*drule-tac tauActTauChain*) *auto*
qed
hence $\Psi \rhd R \parallel !P \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot P') \parallel !P))$
using ‹$xvec$ $\sharp*$ P› ‹$(p \cdot xvec)$ $\sharp*$ P› ‹$(p \cdot xvec)$ $\sharp*$ $(p \cdot P')$› ‹$(p \cdot xvec)$ $\sharp*$ R'› S ‹*distinctPerm p*›
by(*subst resChainAlpha*[**where** p=p]) *auto*
moreover from *P'Chain* **have** $(p \cdot ((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'))) \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot T)$
by(*rule tauChainEqvt*)
with ‹$xvec$ $\sharp*$ Ψ› ‹$(p \cdot xvec)$ $\sharp*$ Ψ› S ‹*distinctPerm p*›
have $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot T)$ **by**(*simp add*: *eqvts*)
hence $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi') \otimes \mathbf{1} \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot T)$
by(*rule tauChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd (p \cdot P') \parallel !P \Longrightarrow\hat{}_\tau (p \cdot T) \parallel !P$
by(*rule-tac tauChainPar1*) *auto*
hence $\Psi \otimes (p \cdot \Psi_R) \otimes \Psi' \rhd (p \cdot P') \parallel !P \Longrightarrow\hat{}_\tau (p \cdot T) \parallel !P$
by(*rule tauChainStatEq*) (*metis Associativity*)
hence $\Psi \otimes \Psi_R{}' \rhd (p \cdot P') \parallel !P \Longrightarrow\hat{}_\tau (p \cdot T) \parallel !P$ **using** ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$›
by(*rule-tac tauChainStatEq*) (*auto intro*: *compositionSym*)
hence $\Psi \rhd R' \parallel ((p \cdot P') \parallel !P) \Longrightarrow\hat{}_\tau R' \parallel ((p \cdot T) \parallel !P)$
using *FrR'* ‹$A_R{}'$ $\sharp*$ Ψ› ‹$A_R{}'$ $\sharp*$ P› ‹$A_R{}'$ $\sharp*$ P'› ‹$A_R{}'$ $\sharp*$ $xvec$› ‹$A_R{}'$ $\sharp*$ $(p \cdot xvec)$› S
by(*rule-tac tauChainPar2*) (*auto simp add*: *freshChainSimps*)
hence $\Psi \rhd (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot P') \parallel !P)) \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot T) \parallel !P))$ **using** ‹$xvec$ $\sharp*$ Ψ›
by(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd R \parallel !P \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot T) \parallel !P))$
by *auto*
moreover have $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot T) \parallel !P)), (\!|\nu{*}xvec|\!)(R' \parallel Q')) \in Rel''$
proof −
from ‹$((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'), T, S) \in Rel$›
have $(p \cdot ((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi')), (p \cdot T), (p \cdot S)) \in Rel$
by(*rule Closed*)

with ‹*xvec* ♯∗ Ψ› ‹(*p* · *xvec*) ♯∗ Ψ› ‹*distinctPerm p*› *S*
have ((Ψ ⊗ (*p* · Ψ_R)) ⊗ Ψ′, *p* · *T*, *p* · *S*) ∈ *Rel*
by(*simp add*: *eqvts*)
hence (Ψ ⊗ (*p* · Ψ_R) ⊗ Ψ′, *p* · *T*, *p* · *S*) ∈ *Rel*
by(*rule StatEq*) (*metis Associativity*)
hence (Ψ ⊗ Ψ_R′, *p* · *T*, *p* · *S*) ∈ *Rel* **using** ‹(*p* · Ψ_R) ⊗ Ψ′ ≃ Ψ_R′›
by(*rule-tac StatEq*) (*auto dest*: *compositionSym*)
moreover from ‹A_R′ ♯∗ *S*› ‹A_R′ ♯∗ *T*› ‹A_R′ ♯∗ *xvec*› ‹A_R′ ♯∗ (*p* · *xvec*)› *S*
have A_R′ ♯∗ (*p* · *S*) **and** A_R′ ♯∗ (*p* · *T*)
by(*simp add*: *freshChainSimps*)+
ultimately have (Ψ, *R*′ ∥ (*p* · *T*), *R*′ ∥ (*p* · *S*)) ∈ *Rel* **using** *FrR*′ ‹A_R′ ♯∗ Ψ›
by(*rule-tac FrameParPres*) *auto*
hence (Ψ, (*R*′ ∥ (*p* · *T*)) ∥ !*P*, (*R*′ ∥ (*p* · *S*)) ∥ !*P*) ∈ *Rel* **by**(*rule ParPres*)
hence (Ψ, ⦇ν∗*xvec*⦈((*R*′ ∥ (*p* · *T*)) ∥ !*P*), ⦇ν∗*xvec*⦈((*R*′ ∥ (*p* · *S*)) ∥ !*P*)) ∈ *Rel*
using ‹*xvec* ♯∗ Ψ›
by(*rule ResPres*)
hence (Ψ, ⦇ν∗*xvec*⦈(*R*′ ∥ (*p* · *T*)) ∥ !*P*, (⦇ν∗*xvec*⦈(*R*′ ∥ (*p* · *S*))) ∥ !*P*) ∈ *Rel*
using ‹*xvec* ♯∗ Ψ› ‹*xvec* ♯∗ *P*›
by(*force intro*: *Trans ScopeExt*)
hence (Ψ, ⦇ν∗*xvec*⦈(*R*′ ∥ ((*p* · *T*) ∥ !*P*)), (⦇ν∗*xvec*⦈(*R*′ ∥ (*p* · *S*))) ∥ !*P*) ∈ *Rel*
using ‹*xvec* ♯∗ Ψ›
by(*force intro*: *Trans ResPres Assoc*)
moreover from ‹(Ψ, *P*, *Q*) ∈ *Rel*› ‹*guarded P*› ‹*guarded Q*›
have (Ψ, (⦇ν∗*xvec*⦈(*R*′ ∥ (*p* · *S*))) ∥ !*P*, (⦇ν∗*xvec*⦈(*R*′ ∥ (*p* · *S*))) ∥ !*Q*) ∈ *Rel*″
by(*rule C1*)
moreover from ‹(**1**, (*p* · *Q*′), *S* ∥ !*Q*) ∈ *Rel*′›
have (*p* · **1**, *p* · *p* · *Q*′, *p* · (*S* ∥ !*Q*)) ∈ *Rel*′ **by**(*rule Closed*′)
with ‹*xvec* ♯∗ *Q*› ‹(*p* · *xvec*) ♯∗ *Q*› *S* ‹*distinctPerm p*›
have (**1**, *Q*′, (*p* · *S*) ∥ !*Q*) ∈ *Rel*′ **by**(*simp add*: *eqvts*)
hence (**1** ⊗ Ψ, *Q*′, (*p* · *S*) ∥ !*Q*) ∈ *Rel*′ **by**(*rule cExt*′)
hence (Ψ, *Q*′, (*p* · *S*) ∥ !*Q*) ∈ *Rel*′
by(*rule StatEq*′) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence (Ψ, *R*′ ∥ *Q*′, *R*′ ∥ ((*p* · *S*) ∥ !*Q*)) ∈ *Rel*′ **by**(*rule ParPres*′)
hence (Ψ, *R*′ ∥ *Q*′, (*R*′ ∥ (*p* · *S*)) ∥ !*Q*) ∈ *Rel*′ **by**(*metis Trans*′ *Assoc*′)
hence (Ψ, (*R*′ ∥ (*p* · *S*)) ∥ !*Q*, *R*′ ∥ *Q*′) ∈ *Rel*′ **by**(*rule cSym*′)
hence (Ψ, ⦇ν∗*xvec*⦈((*R*′ ∥ (*p* · *S*)) ∥ !*Q*), ⦇ν∗*xvec*⦈(*R*′ ∥ *Q*′)) ∈ *Rel*′ **using** ‹*xvec* ♯∗ Ψ›
by(*rule ResPres*′)
hence (Ψ, (⦇ν∗*xvec*⦈(*R*′ ∥ (*p* · *S*))) ∥ !*Q*, ⦇ν∗*xvec*⦈(*R*′ ∥ *Q*′)) ∈ *Rel*′ **using** ‹*xvec* ♯∗ Ψ› ‹*xvec* ♯∗ *Q*›
by(*force intro*: *Trans*′ *ScopeExt*′[*THEN cSym*′])
ultimately show *?thesis* **by**(*rule-tac Compose*)
qed
ultimately show *?case* **by** *blast*
qed

qed
unbundle *relcomp-syntax*
end

end

theory *Weak-Stat-Imp-Pres*
  imports *Weak-Stat-Imp*
begin

context *env* **begin**

lemma *weakStatImpInputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and *xvec* :: *name list*
  and N :: $'a$

  assumes *PRelQ*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ M(\!|\lambda{*}xvec\ N|\!).P,\ M(\!|\lambda{*}xvec\ N|\!).Q) \in Rel$

  shows $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \lessapprox\!<Rel>\ M(\!|\lambda{*}xvec\ N|\!).Q$
using *assms*
by(*fastforce simp add*: *weakStatImp-def*)

lemma *weakStatImpOutputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$

  assumes *PRelQ*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ M\langle N\rangle.P,\ M\langle N\rangle.Q) \in Rel$

  shows $\Psi \rhd M\langle N\rangle.P \lessapprox\!<Rel>\ M\langle N\rangle.Q$
using *assms*
by(*fastforce simp add*: *weakStatImp-def*)

lemma *weakStatImpResPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and Q :: $('a, 'b, 'c)$ *psi*
  and x :: *name*
  and *Rel'* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*

assumes *PSimQ*: $\Psi \rhd P \lessapprox <Rel> Q$
and *eqvt Rel*
and $x \sharp \Psi$
and *C1*: $\bigwedge \Psi'\ R\ S\ y.\ [\![(\Psi', R, S) \in Rel;\ y \sharp \Psi']\!] \Longrightarrow (\Psi', (\!|\nu y|\!)R, (\!|\nu y|\!)S) \in Rel'$

shows $\Psi \rhd (\!|\nu x|\!)P \lessapprox <Rel'> (\!|\nu x|\!)Q$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
obtain *y*::*name* **where** $y \sharp \Psi$ **and** $y \sharp \Psi'$ **and** $y \sharp P$ **and** $y \sharp Q$ **by**(*generate-fresh name*) *auto*
from ‹*eqvt Rel*› ‹$\Psi \rhd P \lessapprox <Rel> Q$› **have** $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \lessapprox <Rel> ([(x, y)] \cdot Q)$ **by**(*rule weakStatImpClosed*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \rhd ([(x, y)] \cdot P) \lessapprox <Rel> ([(x, y)] \cdot Q)$ **by** *simp*
then obtain $Q'\ Q''$ **where** *QChain*: $\Psi \rhd ([(x, y)] \cdot Q) \Longrightarrow\hat{}_\tau\ Q'$
and *PimpQ'*: *insertAssertion* (*extractFrame* $([(x, y)] \cdot P)$) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q') Ψ
and *Q'Chain*: $\Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$ **and** $(\Psi \otimes \Psi', ([(x, y)] \cdot P), Q'') \in Rel$
by(*rule weakStatImpE*)
from *QChain* ‹$y \sharp \Psi$› **have** $\Psi \rhd (\!|\nu y|\!)([(x, y)] \cdot Q) \Longrightarrow\hat{}_\tau (\!|\nu y|\!)Q'$ **by**(*rule tauChainResPres*)
with ‹$y \sharp Q$› **have** $\Psi \rhd (\!|\nu x|\!)Q \Longrightarrow\hat{}_\tau (\!|\nu y|\!)Q'$ **by**(*simp add*: *alphaRes*)
moreover from *PimpQ'* ‹$y \sharp \Psi$› **have** *insertAssertion* (*extractFrame*$((\!|\nu y|\!)([(x, y)] \cdot P))$) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*$((\!|\nu y|\!)Q')$) Ψ
by(*force intro*: *frameImpResPres*)
with ‹$y \sharp P$› **have** *insertAssertion* (*extractFrame*$((\!|\nu x|\!)P)$) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*$((\!|\nu y|\!)Q')$) Ψ
by(*simp add*: *alphaRes*)
moreover from *Q'Chain* ‹$y \sharp \Psi$› ‹$y \sharp \Psi'$› **have** $\Psi \otimes \Psi' \rhd (\!|\nu y|\!)Q' \Longrightarrow\hat{}_\tau (\!|\nu y|\!)Q''$ **by**(*rule-tac tauChainResPres*) *auto*
moreover from ‹$(\Psi \otimes \Psi', ([(x, y)] \cdot P), Q'') \in Rel$› ‹$y \sharp \Psi$› ‹$y \sharp \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu y|\!)([(x, y)] \cdot P), (\!|\nu y|\!)Q'') \in Rel'$
by(*blast intro*: *C1*)
with ‹$y \sharp P$› **have** $(\Psi \otimes \Psi', (\!|\nu x|\!)P, (\!|\nu y|\!)Q'') \in Rel'$ **by**(*simp add*: *alphaRes*)
ultimately show *?case*
by *blast*
qed

lemma *weakStatImpParPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*
and *Rel'* :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes *PStatImpQ*: $\bigwedge A_R\ \Psi_R.\ [\![extractFrame\ R = \langle A_R, \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q]\!] \Longrightarrow \Psi \otimes \Psi_R \rhd P \lessapprox <Rel> Q$

and $xvec \sharp* \Psi$
and $Eqvt$: $eqvt\ Rel$

and $C1$: $\bigwedge \Psi'\ S\ T\ A_U\ \Psi_U\ U.\ [\![(\Psi' \otimes \Psi_U,\ S,\ T) \in Rel;\ extractFrame\ U = \langle A_U,\ \Psi_U\rangle;\ A_U \sharp* \Psi';\ A_U \sharp* S;\ A_U \sharp* T]\!] \Longrightarrow (\Psi',\ S \parallel U,\ T \parallel U) \in Rel'$
and $C2$: $\bigwedge \Psi'\ S\ T\ yvec.\ [\![(\Psi',\ S,\ T) \in Rel';\ yvec \sharp* \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu{*}yvec|\!)S,\ (\!|\nu{*}yvec|\!)T) \in Rel'$
and $C3$: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel;\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'',\ S,\ T) \in Rel$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel R) \lessapprox{<}Rel'{>} (\!|\nu{*}xvec|\!)(Q \parallel R)$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
obtain $A_R\ \Psi_R$ **where** FrR: $extractFrame\ R = \langle A_R,\ \Psi_R\rangle$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* \Psi'$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* R$ **and** $A_R \sharp* xvec$
by(*rule-tac* $F{=}extractFrame\ R$ **and** $C{=}(xvec,\ \Psi,\ \Psi',\ P,\ Q,\ R,\ xvec)$ **in** *freshFrame*) *auto*

hence $\Psi \otimes \Psi_R \rhd P \lessapprox{<}Rel{>}\ Q$ **by**(*rule-tac PStatImpQ*)

obtain $p{::}name\ prm$ **where** $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* \Psi'$ **and** $(p \cdot xvec) \sharp* \Psi_R$
and $(p \cdot xvec) \sharp* A_R$ **and** $(p \cdot xvec) \sharp* R$ **and** $(p \cdot xvec) \sharp* P$
and $distinctPerm\ p$ **and** S: $set\ p \subseteq set\ xvec \times set\ (p \cdot xvec)$
by(*rule-tac* $c{=}(P,\ Q,\ R,\ \Psi,\ \Psi',\ A_R,\ \Psi_R,\ P)$ **in** *name-list-avoiding*) *auto*

from FrR **have** $(p \cdot extractFrame\ R) = (p \cdot \langle A_R,\ \Psi_R\rangle)$ **by**(*rule pt-bij3*)
with ‹$A_R \sharp* xvec$› ‹$(p \cdot xvec) \sharp* A_R$› S **have** $FrpR$: $extractFrame(p \cdot R) = \langle A_R,\ p \cdot \Psi_R\rangle$ **by**(*simp add*: *eqvts*)
from ‹$eqvt\ Rel$› ‹$\Psi \otimes \Psi_R \rhd P \lessapprox{<}Rel{>}\ Q$› **have** $(p \cdot (\Psi \otimes \Psi_R)) \rhd (p \cdot P) \lessapprox{<}Rel{>}\ (p \cdot Q)$ **by**(*rule weakStatImpClosed*)
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S **have** $\Psi \otimes (p \cdot \Psi_R) \rhd (p \cdot P) \lessapprox{<}Rel{>}\ (p \cdot Q)$ **by**(*simp add*: *eqvts*)
then obtain $Q'\ Q''$ **where** $QChain$: $\Psi \otimes (p \cdot \Psi_R) \rhd (p \cdot Q) \Longrightarrow\hat{}_\tau\ Q'$
and $PimpQ'$: $insertAssertion\ (extractFrame\ (p \cdot P))\ (\Psi \otimes (p \cdot \Psi_R)) \hookrightarrow_F insertAssertion\ (extractFrame\ Q')\ (\Psi \otimes (p \cdot \Psi_R))$
and $Q'Chain$: $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$ **and** $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi',\ (p \cdot P),\ Q'') \in Rel$
by(*rule weakStatImpE*)

from ‹$A_R \sharp* xvec$› ‹$(p \cdot xvec) \sharp* A_R$› ‹$A_R \sharp* Q$› S **have** $A_R \sharp* (p \cdot Q)$ **by**(*simp add*: *freshChainSimps*)
moreover from ‹$(p \cdot xvec) \sharp* Q$› **have** $(p \cdot p \cdot xvec) \sharp* (p \cdot Q)$
by(*simp only*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence $xvec \sharp* (p \cdot Q)$ **using** ‹$distinctPerm\ p$› **by** *simp*
ultimately have $A_R \sharp* Q'$ **and** $A_R \sharp* Q''$ **and** $xvec \sharp* Q'$ **and** $xvec \sharp* Q''$ **using** $QChain\ Q'Chain$
by(*metis tauChainFreshChain*)+

obtain $A_P\ \Psi_P$ **where** FrP: $extractFrame(p \cdot P) = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp* \Psi\ A_P$

$\sharp*$ Ψ' **and** A_P $\sharp*$ A_R **and** A_P $\sharp*$ $(p \cdot \Psi_R)$
by(*rule-tac* C=(Ψ, Ψ', A_R, $p \cdot \Psi_R$) **in** *freshFrame*) *auto*
obtain $A_Q{}'$ $\Psi_Q{}'$ **where** *FrQ'*: *extractFrame* $Q' = \langle A_Q{}', \Psi_Q{}'\rangle$ **and** $A_Q{}'$ $\sharp*$ Ψ**and** $A_Q{}'$ $\sharp*$ Ψ' **and** $A_Q{}'$ $\sharp*$ A_R **and** $A_Q{}'$ $\sharp*$ $(p \cdot \Psi_R)$
by(*rule-tac* C=(Ψ, Ψ', A_R, $p \cdot \Psi_R$) **in** *freshFrame*) *auto*
from ‹A_R $\sharp*$ *xvec*› ‹$(p \cdot xvec)$ $\sharp*$ A_R› ‹A_R $\sharp*$ P› S **have** A_R $\sharp*$ $(p \cdot P)$ **by**(*simp add*: *freshChainSimps*)
with ‹A_R $\sharp*$ Q'› ‹A_P $\sharp*$ A_R› ‹$A_Q{}'$ $\sharp*$ A_R› *FrP FrQ'* **have** A_R $\sharp*$ Ψ_P **and** A_R $\sharp*$ $\Psi_Q{}'$
by(*force dest*: *extractFrameFreshChain*)+

from *QChain FrpR* ‹A_R $\sharp*$ Ψ› ‹A_R $\sharp*$ $(p \cdot Q)$› **have** $\Psi \rhd (p \cdot Q) \parallel (p \cdot R) \Longrightarrow\hat{}_\tau\ Q' \parallel (p \cdot R)$ **by**(*rule tauChainPar1*)
hence $(p \cdot \Psi) \rhd (p \cdot ((p \cdot Q) \parallel (p \cdot R))) \Longrightarrow\hat{}_\tau\ p \cdot (Q' \parallel (p \cdot R))$ **by**(*rule eqvts*)
with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec)$ $\sharp*$ Ψ› S ‹*distinctPerm p*› **have** $\Psi \rhd Q \parallel R \Longrightarrow\hat{}_\tau (p \cdot Q') \parallel R$ **by**(*simp add*: *eqvts*)
hence $\Psi \rhd (\!|\nu{*}xvec|\!)(Q \parallel R) \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)((p \cdot Q') \parallel R)$ **using** ‹*xvec* $\sharp*$ Ψ› **by**(*rule tauChainResChainPres*)
moreover have $\langle (A_P@A_R), \Psi \otimes \Psi_P \otimes (p \cdot \Psi_R)\rangle \hookrightarrow_F \langle (A_Q{}'@A_R), \Psi \otimes \Psi_Q{}' \otimes (p \cdot \Psi_R)\rangle$
proof −
have $\langle A_P, \Psi \otimes \Psi_P \otimes (p \cdot \Psi_R)\rangle \simeq_F \langle A_P, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_P\rangle$
by(*metis frameResChainPres frameNilStatEq Associativity Commutativity AssertionStatEqTrans Composition*)
moreover with *FrP FrQ' PimpQ'* ‹A_P $\sharp*$ Ψ› ‹A_P $\sharp*$ $(p \cdot \Psi_R)$› ‹$A_Q{}'$ $\sharp*$ Ψ› ‹$A_Q{}'$ $\sharp*$ $(p \cdot \Psi_R)$›
have $\langle A_P, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_P\rangle \hookrightarrow_F \langle A_Q{}', (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_Q{}'\rangle$ **using** *freshCompChain*
by *simp*
moreover have $\langle A_Q{}', (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_Q{}'\rangle \simeq_F \langle A_Q{}', \Psi \otimes \Psi_Q{}' \otimes (p \cdot \Psi_R)\rangle$
by(*metis frameResChainPres frameNilStatEq Associativity Commutativity AssertionStatEqTrans Composition*)
ultimately have $\langle A_P, \Psi \otimes \Psi_P \otimes (p \cdot \Psi_R)\rangle \hookrightarrow_F \langle A_Q{}', \Psi \otimes \Psi_Q{}' \otimes (p \cdot \Psi_R)\rangle$
by(*rule FrameStatEqImpCompose*)
hence $\langle (A_R@A_P), \Psi \otimes \Psi_P \otimes (p \cdot \Psi_R)\rangle \hookrightarrow_F \langle (A_R@A_Q{}'), \Psi \otimes \Psi_Q{}' \otimes (p \cdot \Psi_R)\rangle$
by(*drule-tac frameImpResChainPres*) (*simp add*: *frameChainAppend*)
thus *?thesis*
apply(*simp add*: *frameChainAppend*)
by(*metis frameImpChainComm FrameStatImpTrans*)
qed
with *FrP FrpR FrQ'* ‹A_P $\sharp*$ A_R› ‹A_P $\sharp*$ $(p \cdot \Psi_R)$› ‹$A_Q{}'$ $\sharp*$ A_R› ‹$A_Q{}'$ $\sharp*$ $(p \cdot \Psi_R)$› ‹A_R $\sharp*$ Ψ_P› ‹A_R $\sharp*$ $\Psi_Q{}'$›
‹A_P $\sharp*$ Ψ› ‹$A_Q{}'$ $\sharp*$ Ψ› ‹A_R $\sharp*$ Ψ›
have *insertAssertion*(*extractFrame*($(p \cdot P) \parallel (p \cdot R)$)) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame*($Q' \parallel (p \cdot R)$)) Ψ
by *simp*
hence ($p \cdot$ *insertAssertion*(*extractFrame*($(p \cdot P) \parallel (p \cdot R)$)) Ψ) $\hookrightarrow_F$ ($p \cdot$ *insertAssertion*(*extractFrame*($Q' \parallel (p \cdot R)$)) Ψ)

by(*rule FrameStatImpClosed*)
with ‹*xvec* ♯* Ψ› ‹(*p* · *xvec*) ♯* Ψ› *S* ‹*distinctPerm p*›
have *insertAssertion*(*extractFrame*(*P* ∥ *R*)) Ψ ↪$_F$ *insertAssertion*(*extractFrame*((*p* · *Q*′) ∥ *R*)) Ψ
by(*simp add*: *eqvts*)
with ‹*xvec* ♯* Ψ› **have** *insertAssertion*(*extractFrame*(⦇ν*xvec⦈(*P* ∥ *R*))) Ψ ↪$_F$ *insertAssertion*(*extractFrame*(⦇ν*xvec⦈((*p* · *Q*′) ∥ *R*))) Ψ
by(*force intro*: *frameImpResChainPres*)
moreover from *Q′Chain* **have** (Ψ ⊗ Ψ′) ⊗ (*p* · Ψ$_R$) ⊳ *Q*′ ⟹̂$_\tau$ *Q*″
by(*rule tauChainStatEq*) (*metis Associativity AssertionStatEqTrans Commutativity Composition*)
hence Ψ ⊗ Ψ′ ⊳ *Q*′ ∥ (*p* · *R*) ⟹̂$_\tau$ *Q*″ ∥ (*p* · *R*) **using** *FrpR* ‹*A*$_R$ ♯* Ψ› ‹*A*$_R$ ♯* Ψ′› ‹*A*$_R$ ♯* *Q*′› ‹*A*$_R$ ♯* *xvec*› ‹(*p* · *xvec*) ♯* *A*$_R$› *S*
by(*force intro*: *tauChainPar1 simp add*: *freshChainSimps*)
hence Ψ ⊗ Ψ′ ⊳ ⦇ν*(*p* · *xvec*)⦈(*Q*′ ∥ (*p* · *R*)) ⟹̂$_\tau$ ⦇ν*(*p* · *xvec*)⦈(*Q*″ ∥ (*p* · *R*)) **using** ‹(*p* · *xvec*) ♯* Ψ› ‹(*p* · *xvec*) ♯* Ψ′›
by(*rule-tac tauChainResChainPres*) *auto*
hence Ψ ⊗ Ψ′ ⊳ ⦇ν*xvec⦈((*p* · *Q*′) ∥ *R*) ⟹̂$_\tau$ ⦇ν*xvec⦈((*p* · *Q*″) ∥ *R*) **using** ‹*xvec* ♯* *Q*′› ‹*xvec* ♯* *Q*″› ‹(*p* · *xvec*) ♯* *R*› *S* ‹*distinctPerm p*›
apply(*subst resChainAlpha*) **apply**(*auto simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
by(*subst resChainAlpha*[*of* - *xvec*]) (*auto simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
moreover from ‹((Ψ ⊗ (*p* · Ψ$_R$)) ⊗ Ψ′, (*p* · *P*), *Q*″) ∈ *Rel*› **have** ((Ψ ⊗ Ψ′) ⊗ (*p* · Ψ$_R$), (*p* · *P*), *Q*″) ∈ *Rel*
by(*rule C3*) (*metis Associativity AssertionStatEqTrans Commutativity Composition*)
hence (Ψ ⊗ Ψ′, (*p* · *P*) ∥ (*p* · *R*), *Q*″ ∥ (*p* · *R*)) ∈ *Rel*′
using *FrpR* ‹*A*$_R$ ♯* Ψ› ‹*A*$_R$ ♯* Ψ′› ‹*A*$_R$ ♯* *Q*″› ‹*A*$_R$ ♯* *P*› ‹*A*$_R$ ♯* *xvec*› ‹(*p* · *xvec*) ♯* *A*$_R$› *S*
by(*rule-tac C1*) (*auto simp add*: *freshChainSimps*)
hence (Ψ ⊗ Ψ′, ⦇ν*(*p* · *xvec*)⦈((*p* · *P*) ∥ (*p* · *R*)), ⦇ν*(*p* · *xvec*)⦈(*Q*″ ∥ (*p* · *R*))) ∈ *Rel*′ **using** ‹(*p* · *xvec*) ♯* Ψ› ‹(*p* · *xvec*) ♯* Ψ′›
by(*rule-tac C2*) *auto*
hence (Ψ ⊗ Ψ′, ⦇ν*xvec⦈(*P* ∥ *R*), ⦇ν*xvec⦈((*p* · *Q*″) ∥ *R*)) ∈ *Rel*′ **using** ‹(*p* · *xvec*) ♯* *P*› ‹*xvec* ♯* *Q*″› ‹(*p* · *xvec*) ♯* *R*› *S* ‹*distinctPerm p*›
apply(*subst resChainAlpha*[**where** *p*=*p*])
apply *simp*
apply *simp*
apply(*subst resChainAlpha*[**where** *xvec*=*xvec* **and** *p*=*p*])
by(*auto simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
ultimately show *?case*
by *blast*
qed

end

end

theory *Weak-Bisim-Pres*
imports *Weak-Bisimulation Weak-Sim-Pres Weak-Stat-Imp-Pres*
begin

context *env* **begin**

lemma *weakBisimInputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$

assumes $\bigwedge Tvec.\ length\ xvec = length\ Tvec \Longrightarrow \Psi \rhd P[xvec::=Tvec] \approx Q[xvec::=Tvec]$

shows $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \approx M(\!|\lambda{*}xvec\ N|\!).Q$
proof −
let $?X = \{(\Psi,\ M(\!|\lambda{*}xvec\ N|\!).P,\ M(\!|\lambda{*}xvec\ N|\!).Q) \mid \Psi\ M\ xvec\ N\ P\ Q.\ \forall Tvec.$
$length\ xvec = length\ Tvec \longrightarrow \Psi \rhd P[xvec::=Tvec] \approx Q[xvec::=Tvec]\}$

from *assms* **have** $(\Psi,\ M(\!|\lambda{*}xvec\ N|\!).P,\ M(\!|\lambda{*}xvec\ N|\!).Q) \in ?X$ **by** *blast*
thus *?thesis*
proof(*coinduct rule*: *weakBisimCoinduct*)
case(*cStatImp* Ψ P Q)
thus *?case* **by**(*fastforce intro*: *weakStatImpInputPres dest*: *weakBisimE*(*3*))
next
case(*cSim* Ψ P Q)
thus *?case*
by *auto* (*blast intro*: *weakInputPres dest*: *weakBisimE*)
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*blast dest*: *weakBisimE*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *weakBisimE*)
qed
qed

lemma *weakBisimOutputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$

assumes $\Psi \rhd P \approx Q$

shows $\Psi \rhd M\langle N\rangle.P \approx M\langle N\rangle.Q$
proof −
let $?X = \{(\Psi,\ M\langle N\rangle.P,\ M\langle N\rangle.Q) \mid \Psi\ M\ N\ P\ Q.\ \Psi \rhd P \approx Q\}$

from *assms* **have** $(\Psi,\ M\langle N\rangle.P,\ M\langle N\rangle.Q) \in ?X$ **by** *blast*
thus *?thesis*
proof(*coinduct rule*: *weakBisimCoinduct*)
case(*cStatImp* Ψ P Q)
thus *?case* **by** *auto* (*blast intro*: *weakStatImpOutputPres dest*: *weakBisimE*(*3*))
next
case(*cSim* Ψ P Q)
thus *?case*
by(*auto intro*: *weakOutputPres dest*: *weakBisimE*)
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*blast dest*: *weakBisimE*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *weakBisimE*)
qed
qed

lemma *weakBisimResPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and x :: *name*

assumes $\Psi \rhd P \approx Q$
and $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)P \approx (\!|\nu x|\!)Q$
proof −
let $?X = \{(\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \mid \Psi\ x\ P\ Q.\ \Psi \rhd P \approx Q \wedge x \sharp \Psi\}$

from *assms* **have** $(\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X$ **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *weakBisimCoinduct*)
case(*cStatImp* Ψ *xP xQ*)
{
fix Ψ P Q x
assume $\Psi \rhd P \approx Q$
hence $\Psi \rhd P \lessapprox$<*weakBisim*> Q **by**(*rule weakBisimE*)
moreover have *eqvt weakBisim* **by** *auto*
moreover assume (x::*name*) $\sharp \Psi$
moreover have $\bigwedge\Psi\ P\ Q\ x.\ [\![(\Psi,\ P,\ Q) \in \mathit{weakBisim};\ x \sharp \Psi]\!] \Longrightarrow (\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X \cup \mathit{weakBisim}$
by *auto*
ultimately have $\Psi \rhd (\!|\nu x|\!)P \lessapprox$<($?X \cup$ *weakBisim*)> $(\!|\nu x|\!)Q$

by(*rule weakStatImpResPres*)
}
with ‹$(\Psi, xP, xQ) \in ?X$› **show** *?case* **by** *auto*
next
case(*cSim* Ψ xP xQ)
from ‹$(\Psi, xP, xQ) \in ?X$› **obtain** x P Q **where** $\Psi \rhd P \approx Q$ **and** $x \sharp \Psi$ **and** $xP = (\!|\nu x|\!)P$ **and** $xQ = (\!|\nu x|\!)Q$
by *auto*
from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \rhd P \leadsto$<*weakBisim*> Q **by**(*rule weakBisimE*)
moreover have *eqvt ?X*
by(*force simp add*: *eqvt-def weakBisimClosed pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
hence *eqvt*(*?X* $\cup$ *weakBisim*) **by** *auto*
moreover note ‹$x \sharp \Psi$›
moreover have *weakBisim* $\subseteq$ *?X* $\cup$ *weakBisim* **by** *auto*
moreover have $\bigwedge\Psi\ P\ Q\ x.\ [\![(\Psi, P, Q) \in weakBisim;\ x \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu x|\!)P, (\!|\nu x|\!)Q) \in ?X \cup weakBisim$
by *auto*
ultimately have $\Psi \rhd (\!|\nu x|\!)P \leadsto$<(*?X* $\cup$ *weakBisim*)> $(\!|\nu x|\!)Q$
by(*rule weakResPres*)
with ‹$xP = (\!|\nu x|\!)P$› ‹$xQ = (\!|\nu x|\!)Q$› **show** *?case*
by *simp*
next
case(*cExt* Ψ xP xQ Ψ')
from ‹$(\Psi, xP, xQ) \in ?X$› **obtain** x P Q **where** $\Psi \rhd P \approx Q$ **and** $x \sharp \Psi$ **and** $xP = (\!|\nu x|\!)P$ **and** $xQ = (\!|\nu x|\!)Q$
by *auto*
obtain y::*name* **where** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \Psi$ **and** $y \sharp \Psi'$
by(*generate-fresh name*, *auto simp add*: *fresh-prod*)
from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \otimes ([(x, y)] \cdot \Psi') \rhd P \approx Q$
by(*rule weakBisimE*)
hence $([(x, y)] \cdot (\Psi \otimes ([(x, y)] \cdot \Psi'))) \rhd ([(x, y)] \cdot P) \approx ([(x, y)] \cdot Q)$
by(*rule weakBisimClosed*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \otimes \Psi' \rhd ([(x, y)] \cdot P) \approx ([(x, y)] \cdot Q)$
by(*simp add*: *eqvts*)
with ‹$y \sharp \Psi$› ‹$y \sharp \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu y|\!)([(x, y)] \cdot P), (\!|\nu y|\!)([(x, y)] \cdot Q)) \in ?X$
by *auto*
moreover from ‹$y \sharp P$› ‹$y \sharp Q$› **have** $(\!|\nu x|\!)P = (\!|\nu y|\!)([(x, y)] \cdot P)$ **and** $(\!|\nu x|\!)Q = (\!|\nu y|\!)([(x, y)] \cdot Q)$
by(*simp add*: *alphaRes*)+
ultimately show *?case* **using** ‹$xP = (\!|\nu x|\!)P$› ‹$xQ = (\!|\nu x|\!)Q$› **by** *simp*
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*blast dest*: *weakBisimE*)
qed
qed

lemma *weakBisimResChainPres*:

fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \approx Q$
and $xvec \sharp* \Psi$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \approx (\!|\nu{*}xvec|\!)Q$
using *assms*
by(*induct xvec*) (*auto intro*: *weakBisimResPres*)

lemma *weakBisimParPresAux*:
fixes Ψ :: $'b$
and Ψ_R :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*
and A_R :: *name list*

assumes $\Psi \otimes \Psi_R \rhd P \approx Q$
and *FrR*: $extractFrame\ R = \langle A_R, \Psi_R\rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$

shows $\Psi \rhd P \parallel R \approx Q \parallel R$
proof −
let $?X = \{(\Psi, (\!|\nu{*}xvec|\!)(P \parallel R), (\!|\nu{*}xvec|\!)(Q \parallel R)) \mid xvec\ \Psi\ P\ Q\ R.\ xvec \sharp* \Psi \wedge (\forall A_R\ \Psi_R.\ (extractFrame\ R = \langle A_R, \Psi_R\rangle \wedge A_R \sharp* \Psi \wedge A_R \sharp* P \wedge A_R \sharp* Q) \longrightarrow \Psi \otimes \Psi_R \rhd P \approx Q)\}$
{
fix *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*

assume $xvec \sharp* \Psi$
and $\bigwedge A_R\ \Psi_R.\ [\![extractFrame\ R = \langle A_R, \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q]\!] \Longrightarrow \Psi \otimes \Psi_R \rhd P \approx Q$

hence $(\Psi, (\!|\nu{*}xvec|\!)(P \parallel R), (\!|\nu{*}xvec|\!)(Q \parallel R)) \in ?X$
by *blast*
}
note $XI = this$

{

fix *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*
and C :: $'d$::*fs-name*

assume *xvec* $\sharp*$ Ψ
and A: $\bigwedge A_R\ \Psi_R.\ [\![$*extractFrame* $R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q;\ A_R \sharp* C]\!] \Longrightarrow \Psi \otimes \Psi_R \rhd P \approx Q$

from ‹*xvec* $\sharp*$ Ψ› **have** $(\Psi,\ (\!|\nu{*}xvec|\!)(P \parallel R),\ (\!|\nu{*}xvec|\!)(Q \parallel R)) \in ?X$
proof(*rule XI*)
fix $A_R\ \Psi_R$
assume *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$
obtain p::*name prm* **where** $(p \cdot A_R) \sharp* \Psi$ **and** $(p \cdot A_R) \sharp* P$ **and** $(p \cdot A_R) \sharp* Q$ **and** $(p \cdot A_R) \sharp* R$ **and** $(p \cdot A_R) \sharp* C$
and $(p \cdot A_R) \sharp* \Psi_R$ **and** S: $(set\ p) \subseteq (set\ A_R) \times (set(p \cdot A_R))$ **and** *distinctPerm p*
by(*rule-tac c*=$(\Psi,\ P,\ Q,\ R,\ \Psi_R,\ C)$ **in** *name-list-avoiding*) *auto*
from *FrR* ‹$(p \cdot A_R) \sharp* \Psi_R$› S **have** *extractFrame* $R = \langle (p \cdot A_R),\ p \cdot \Psi_R\rangle$ **by**(*simp add*: *frameChainAlpha'*)

moreover assume $A_R \sharp* \Psi$
hence $(p \cdot A_R) \sharp* (p \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* \Psi$› S **have** $(p \cdot A_R) \sharp* \Psi$ **by** *simp*
moreover assume $A_R \sharp* P$
hence $(p \cdot A_R) \sharp* (p \cdot P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* P$› ‹$(p \cdot A_R) \sharp* P$› S **have** $(p \cdot A_R) \sharp* P$ **by** *simp*
moreover assume $A_R \sharp* Q$
hence $(p \cdot A_R) \sharp* (p \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$A_R \sharp* Q$› ‹$(p \cdot A_R) \sharp* Q$› S **have** $(p \cdot A_R) \sharp* Q$ **by** *simp*
ultimately have $\Psi \otimes (p \cdot \Psi_R) \rhd P \approx Q$ **using** ‹$(p \cdot A_R) \sharp* C$› A **by** *blast*
hence $(p \cdot (\Psi \otimes (p \cdot \Psi_R))) \rhd (p \cdot P) \approx (p \cdot Q)$ **by**(*rule weakBisimClosed*)
with ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$(p \cdot A_R) \sharp* P$› ‹$A_R \sharp* Q$› ‹$(p \cdot A_R) \sharp* Q$› S ‹*distinctPerm p*›
show $\Psi \otimes \Psi_R \rhd P \approx Q$ **by**(*simp add*: *eqvts*)
qed
}
note $XI' = this$

have *eqvt ?X*
apply(*auto simp add*: *eqvt-def*)
apply(*rule-tac x*=$p \cdot xvec$ **in** *exI*)
apply(*rule-tac x*=$p \cdot P$ **in** *exI*)
apply(*rule-tac x*=$p \cdot Q$ **in** *exI*)

apply(*rule-tac x=p* · *R* **in** *exI*)
apply(*simp add*: *eqvts*)
apply(*simp add*: *fresh-star-bij*)
apply(*clarify*)
apply(*erule-tac x=*(*rev p*) · A_R **in** *allE*)
apply(*erule-tac x=*(*rev p*) · Ψ_R **in** *allE*)
apply(*drule mp*)
apply(*rule conjI*)
apply(*rule-tac pi=p* **in** *pt-bij4*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *eqvts*)
defer
apply(*drule-tac p=p* **in** *weakBisimClosed*)
apply(*simp add*: *eqvts*)
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of p*, *THEN sym*])
apply *simp*
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of p*, *THEN sym*])
apply *simp*
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of p*, *THEN sym*])
by *simp*

moreover have *Res*: $\bigwedge\Psi\ P\ Q\ x.\ [\![(\Psi,\ P,\ Q) \in ?X \cup weakBisim;\ x \sharp \Psi]\!] \Longrightarrow (\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X \cup weakBisim$
proof −
fix $\Psi\ P\ Q\ x$
assume $(\Psi,\ P,\ Q) \in ?X \cup weakBisim$ **and** $(x::name) \sharp \Psi$
show $(\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X \cup weakBisim$
proof(*case-tac* $(\Psi,\ P,\ Q) \in ?X$)
assume $(\Psi,\ P,\ Q) \in ?X$
with ‹$x \sharp \Psi$› **have** $(\Psi,\ (\!|\nu x|\!)P,\ (\!|\nu x|\!)Q) \in ?X$
apply *auto*
by(*rule-tac x=x#xvec* **in** *exI*) *auto*
thus *?thesis* **by** *simp*
next
assume $\neg(\Psi,\ P,\ Q) \in ?X$
with ‹$(\Psi,\ P,\ Q) \in ?X \cup weakBisim$› **have** $\Psi \rhd P \approx Q$
by *blast*
hence $\Psi \rhd (\!|\nu x|\!)P \approx (\!|\nu x|\!)Q$ **using** ‹$x \sharp \Psi$›
by(*rule weakBisimResPres*)
thus *?thesis*
by *simp*
qed
qed

{
fix Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*

and Q :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assume $\Psi \rhd P \approx Q$

hence $\Psi \rhd Q \approx P$ **by**(*rule weakBisimE*)
then obtain P' P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
and *QimpP'*: *insertAssertion*(*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P') Ψ
and *P'Chain*: $\Psi \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P''$
and $\Psi \otimes \Psi' \rhd Q \approx P''$ **using** *weakStatImp-def*
by(*blast dest*: *weakBisimE*)
note *PChain QimpP' P'Chain*
moreover from ‹$\Psi \otimes \Psi' \rhd Q \approx P''$› **have** $\Psi \otimes \Psi' \rhd P'' \approx Q$ **by**(*rule weakBisimE*)
ultimately have $\exists P'\ P''.\ \Psi \rhd P \Longrightarrow\hat{}_\tau P' \wedge$ *insertAssertion*(*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P') $\Psi \wedge$
$\Psi \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P'' \wedge \Psi \otimes \Psi' \rhd P'' \approx Q$
by *blast*
}
moreover
{
fix Ψ P Q A_R Ψ_R R
assume *PSimQ*: $\Psi \otimes \Psi_R \rhd P \approx Q$
and *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$
hence $(\Psi, P \parallel R, Q \parallel R) \in ?X$
proof −
have $P \parallel R = (\!|\nu*[]|\!)(P \parallel R)$ **by** *simp*
moreover have $Q \parallel R = (\!|\nu*[]|\!)(Q \parallel R)$ **by** *simp*
moreover have ([]::*name list*) $\sharp* \Psi$ **by** *simp*
moreover
{
fix A_R' Ψ_R'

assume *FrR'*: *extractFrame* $R = \langle A_R', \Psi_R'\rangle$
and $A_R' \sharp* \Psi$
and $A_R' \sharp* P$
and $A_R' \sharp* Q$
obtain p **where** $(p \cdot A_R') \sharp* A_R$
and $(p \cdot A_R') \sharp* \Psi_R'$
and $(p \cdot A_R') \sharp* \Psi$
and $(p \cdot A_R') \sharp* P$
and $(p \cdot A_R') \sharp* Q$
and S: $(set\ p) \subseteq (set\ A_R') \times (set(p \cdot A_R'))$ **and** *distinctPerm* p
by(*rule-tac* c=$(A_R, \Psi, \Psi_R', P, Q)$ **in** *name-list-avoiding*) *auto*

from ‹$(p \cdot A_R{}') \sharp* \Psi_R{}'$› S **have** $\langle A_R{}', \Psi_R{}'\rangle = \langle p \cdot A_R{}', p \cdot \Psi_R{}'\rangle$
by(*simp add*: *frameChainAlpha*)

with *FrR'* **have** *FrR''*: *extractFrame* $R = \langle p \cdot A_R{}', p \cdot \Psi_R{}'\rangle$ **by** *simp*
with *FrR* ‹$(p \cdot A_R{}') \sharp* A_R$›
obtain q **where** $p \cdot \Psi_R{}' = (q::name\ prm) \cdot \Psi_R$ **and** S': $set\ q \subseteq (set\ A_R) \times set(p \cdot A_R{}')$ **and** *distinctPerm q*
apply *auto*
apply(*drule-tac sym*) **apply** *simp*
by(*drule-tac frameChainEq*) *auto*
from *PSimQ* **have** $(q \cdot (\Psi \otimes \Psi_R)) \rhd (q \cdot P) \approx (q \cdot Q)$
by(*rule weakBisimClosed*)
with ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$(p \cdot A_R{}') \sharp* Q$› S'
have $\Psi \otimes (q \cdot \Psi_R) \rhd P \approx Q$ **by**(*simp add*: *eqvts*)
hence $(p \cdot (\Psi \otimes (q \cdot \Psi_R))) \rhd (p \cdot P) \approx (p \cdot Q)$ **by**(*rule weakBisimClosed*)
with ‹$A_R{}' \sharp* \Psi$› ‹$A_R{}' \sharp* P$› ‹$A_R{}' \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$(p \cdot A_R{}') \sharp* Q$› S ‹*distinctPerm p*› ‹$(p \cdot \Psi_R{}') = q \cdot \Psi_R$›
have $\Psi \otimes \Psi_R{}' \rhd P \approx Q$
by(*drule-tac sym*) (*simp add*: *eqvts*)
}
ultimately show *?thesis*
by *blast*
qed
hence $(\Psi, P \parallel R, Q \parallel R) \in ?X \cup weakBisim$
by *simp*
}
note *C1* = *this*

have *C2*: $\bigwedge \Psi\ P\ Q\ xvec.\ [\![(\Psi, P, Q) \in ?X \cup weakBisim; (xvec::name\ list) \sharp* \Psi]\!] \Longrightarrow (\Psi, (\!|\nu*xvec|\!)P, (\!|\nu*xvec|\!)Q) \in ?X \cup weakBisim$
proof −
fix $\Psi\ P\ Q\ xvec$
assume $(\Psi, P, Q) \in ?X \cup weakBisim$
assume $(xvec::name\ list) \sharp* \Psi$
thus $(\Psi, (\!|\nu*xvec|\!)P, (\!|\nu*xvec|\!)Q) \in ?X \cup weakBisim$
proof(*induct xvec*)
case *Nil*
thus *?case* **using** ‹$(\Psi, P, Q) \in ?X \cup weakBisim$› **by** *simp*
next
case(*Cons x xvec*)
thus *?case* **by**(*simp only*: *resChain.simps*) (*rule-tac Res*, *auto*)
qed
qed

{
fix $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*
and A_R :: *name list*
and Ψ_R :: $'b$

assume $\Psi \otimes \Psi_R \rhd P \approx Q$
and *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$

have $(\Psi,\ P \parallel R,\ Q \parallel R) \in ?X$
proof −
{
fix $A_R{}'$ :: *name list*
and $\Psi_R{}'$ :: $'b$

assume *FrR'*: *extractFrame* $R = \langle A_R{}',\ \Psi_R{}'\rangle$
and $A_R{}' \sharp* \Psi$
and $A_R{}' \sharp* P$
and $A_R{}' \sharp* Q$

obtain p **where** $(p \cdot A_R{}') \sharp* A_R$ **and** $(p \cdot A_R{}') \sharp* \Psi_R{}'$ **and** $(p \cdot A_R{}') \sharp* \Psi$ **and** $(p \cdot A_R{}') \sharp* P$ **and** $(p \cdot A_R{}') \sharp* Q$
and *Sp*: $(set\ p) \subseteq (set\ A_R{}') \times (set(p \cdot A_R{}'))$ **and** *distinctPerm* p
by(*rule-tac c*=$(A_R,\ \Psi,\ \Psi_R{}',\ P,\ Q)$ **in** *name-list-avoiding*) *auto*

from *FrR'* ‹$(p \cdot A_R{}') \sharp*\ \Psi_R{}'$› *Sp* **have** *extractFrame* $R = \langle (p \cdot A_R{}'),\ p \cdot \Psi_R{}'\rangle$
by(*simp add*: *frameChainAlpha eqvts*)
with *FrR* ‹$(p \cdot A_R{}') \sharp* A_R$› **obtain** q::*name prm*
where *Sq*: $set\ q \subseteq set(p \cdot A_R{}') \times set\ A_R$ **and** *distinctPerm* q **and** $\Psi_R = q \cdot p \cdot \Psi_R{}'$
by(*force elim*: *frameChainEq*)

from ‹$\Psi \otimes \Psi_R \rhd P \approx Q$› ‹$\Psi_R = q \cdot p \cdot \Psi_R{}'$› **have** $\Psi \otimes (q \cdot p \cdot \Psi_R{}') \rhd P \approx Q$ **by** *simp*
hence $(q \cdot (\Psi \otimes (q \cdot p \cdot \Psi_R{}'))) \rhd (q \cdot P) \approx (q \cdot Q)$ **by**(*rule weakBisimClosed*)
with *Sq* ‹$A_R \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$A_R \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* Q$› ‹*distinctPerm* q›
have $\Psi \otimes (p \cdot \Psi_R{}') \rhd P \approx Q$ **by**(*simp add*: *eqvts*)
hence $(p \cdot (\Psi \otimes (p \cdot \Psi_R{}'))) \rhd (p \cdot P) \approx (p \cdot Q)$ **by**(*rule weakBisimClosed*)
with *Sp* ‹$A_R{}' \sharp* \Psi$› ‹$(p \cdot A_R{}') \sharp* \Psi$› ‹$A_R{}' \sharp* P$› ‹$(p \cdot A_R{}') \sharp* P$› ‹$A_R{}' \sharp* Q$› ‹$(p \cdot A_R{}') \sharp* Q$› ‹*distinctPerm* p›
have $\Psi \otimes \Psi_R{}' \rhd P \approx Q$ **by**(*simp add*: *eqvts*)
}
thus *?thesis*
apply *auto*

apply(*rule-tac x*=[] **in** *exI*)
by *auto blast*
qed
}
note *Goal* = *this*
with *assms* **have** $(\Psi,\ P \parallel R,\ Q \parallel R) \in ?X$ **by** *blast*
thus *?thesis*
proof(*coinduct rule*: *weakBisimCoinduct*)
case(*cStatImp* Ψ *PR QR*)
{
fix *xvec* :: *name list*
fix *P Q R*
assume *A*: $\forall A_R\ \Psi_R.\ extractFrame\ R = \langle A_R,\ \Psi_R\rangle \wedge A_R \sharp* \Psi \wedge A_R \sharp* P \wedge A_R \sharp* Q \longrightarrow \Psi \otimes \Psi_R \rhd P \approx Q$
{
fix $A_R\ \Psi_R$
assume $extractFrame\ R = \langle A_R,\ \Psi_R\rangle$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$
with *A* **have** $\Psi \otimes \Psi_R \rhd P \lessapprox$<*weakBisim*> Q **by**(*auto dest*: *weakBisimE*)
}
moreover assume $xvec \sharp* \Psi$
moreover have *eqvt weakBisim* **by** *auto*
moreover note *C1 C2 statEqWeakBisim*
ultimately have $\Psi \rhd (\!|\nu*xvec|\!)(P \parallel R) \lessapprox$<$(?X \cup weakBisim)$> $(\!|\nu*xvec|\!)(Q \parallel R)$
by(*rule weakStatImpParPres*)
}
with ‹$(\Psi,\ PR,\ QR) \in ?X$› **show** *?case* **by** *auto*
next
case(*cSim* Ψ *PR QR*)
from ‹$(\Psi,\ PR,\ QR) \in ?X$›
obtain *xvec P Q R* $A_R\ \Psi_R$ **where** *PFrR*: $PR = (\!|\nu*xvec|\!)(P \parallel R)$ **and** *QFrR*: $QR = (\!|\nu*xvec|\!)(Q \parallel R)$
and $xvec \sharp* \Psi$
by *auto*
with ‹$(\Psi,\ PR,\ QR) \in ?X$› **have** $(\Psi,\ (\!|\nu*xvec|\!)(P \parallel R),\ (\!|\nu*xvec|\!)(Q \parallel R)) \in ?X$ **by** *simp*
hence $\Psi \rhd (\!|\nu*xvec|\!)(P \parallel R) \rightsquigarrow$<$(?X \cup weakBisim)$> $(\!|\nu*xvec|\!)(Q \parallel R)$ **using** ‹$xvec \sharp* \Psi$›
proof(*induct xvec*)
case *Nil*
from ‹$(\Psi,\ (\!|\nu*[]|\!)(P \parallel R),\ (\!|\nu*[]|\!)(Q \parallel R)) \in ?X$› **have** *PRQR*: $(\Psi,\ P \parallel R,\ Q \parallel R) \in ?X$ **by** *simp*
from *PRQR* **have** $\bigwedge A_R\ \Psi_R.\ [\![extractFrame\ R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q]\!] \Longrightarrow (\Psi \otimes \Psi_R,\ P,\ Q) \in weakBisim$
by *auto*
moreover note *weakBisimEqvt*
moreover from ‹*eqvt ?X*› **have** $eqvt(?X \cup weakBisim)$ **by** *auto*
moreover note *weakBisimE*(*2*) *weakBisimE*(*4*) *weakBisimE*(*3*) *weakBisimE*(*1*)

moreover note *C1 C2*
ultimately have $\Psi \rhd P \parallel R \rightsquigarrow <(?X \cup weakBisim)> Q \parallel R$ **using** *statEqWeakBisim*
by(*rule weakParPres*)
thus *?case* **by** *simp*
next
case(*Cons x xvec'*)
from ‹$(x\#xvec') \sharp* \Psi$› **have** $x \sharp \Psi$ **and** $xvec' \sharp* \Psi$ **by** *simp+*
with ‹$(\Psi, (\!|\nu*(x\#xvec')|\!)P \parallel R, (\!|\nu*(x\#xvec')|\!)Q \parallel R) \in ?X$›
have $(\Psi, (\!|\nu*(xvec')|\!)P \parallel R, (\!|\nu*(xvec')|\!)Q \parallel R) \in ?X$
apply *auto*
apply(*subgoal-tac* $\exists\, y\ yvec.\ xvec=y\#yvec$)
apply(*clarify*)
apply *simp*
apply(*simp add*: *psi.inject alpha*)
apply(*clarify*)
apply(*erule disjE*)
apply(*erule disjE*)
apply(*clarify*)
apply *blast*
apply(*clarify*)
apply(*clarify*)
apply(*simp add*: *eqvts*)
apply(*rule-tac* $x=[(x, y)] \cdot yvec$ **in** *exI*)
apply(*rule-tac* $x=[(x, y)] \cdot P$ **in** *exI*)
apply(*rule-tac* $x=[(x, y)] \cdot Q$ **in** *exI*)
apply(*rule-tac* $x=[(x, y)] \cdot R$ **in** *exI*)
apply(*clarsimp*)
apply(*rule conjI*)
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of* $[(x, y)]$, *THEN sym*])
apply *simp*
apply(*clarify*)
apply(*erule-tac* $x=[(x, y)] \cdot A_R$ **in** *allE*)
apply(*erule-tac* $x=[(x, y)] \cdot \Psi_R$ **in** *allE*)
apply(*drule mp*)
apply(*rule conjI*)
apply(*rule-tac* $pi=[(x, y)]$ **in** *pt-bij4*[*OF pt-name-inst*, *OF at-name-inst*])
apply(*simp add*: *eqvts*)
apply(*rule conjI*)
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of* $[(x, y)]$, *THEN sym*])
apply *simp*
apply(*rule conjI*)
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of* $[(x, y)]$, *THEN sym*])
apply *simp*
apply(*subst pt-fresh-star-bij*[*OF pt-name-inst*,*OF at-name-inst*, *of* $[(x, y)]$, *THEN sym*])

apply *simp*
apply(*drule-tac p*=[(*x*, *y*)] **in** *weakBisimClosed*)
apply(*simp add*: *eqvts*)
by(*case-tac xvec*) *auto*

with ‹$[\![(\Psi, (\!|\nu{*}xvec'|\!)(P \parallel R), (\!|\nu{*}xvec'|\!)(Q \parallel R)) \in ?X;\ xvec' \sharp{*}\ \Psi]\!] \Longrightarrow \Psi \rhd (\!|\nu{*}xvec'|\!)(P \parallel R) \rightsquigarrow<(?X \cup weakBisim)> (\!|\nu{*}xvec'|\!)(Q \parallel R)$› ‹$xvec' \sharp{*}\ \Psi$›
have $\Psi \rhd (\!|\nu{*}xvec'|\!)(P \parallel R) \rightsquigarrow<(?X \cup weakBisim)> (\!|\nu{*}xvec'|\!)(Q \parallel R)$ **by** *blast*
moreover note ‹*eqvt ?X*›
moreover from ‹*eqvt ?X*› **have** $eqvt(?X \cup weakBisim)$ **by** *auto*
moreover note ‹$x \sharp \Psi$›
moreover have $?X \cup weakBisim \subseteq ?X \cup weakBisim$ **by** *simp*
ultimately have $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec'|\!)(P \parallel R)) \rightsquigarrow<(?X \cup weakBisim)> (\!|\nu x|\!)((\!|\nu{*}xvec'|\!)(Q \parallel R))$ **using** *Res*
by(*rule-tac weakResPres*)
thus *?case*
by *simp*
qed
with *PFrR QFrR* **show** *?case*
by *simp*
next
case(*cExt* Ψ *PR QR* Ψ')

from ‹$(\Psi, PR, QR) \in ?X$›
obtain *xvec P Q R* A_R Ψ_R **where** *PFrR*: $PR = (\!|\nu{*}xvec|\!)(P \parallel R)$ **and** *QFrR*: $QR = (\!|\nu{*}xvec|\!)(Q \parallel R)$
and $xvec \sharp{*}\ \Psi$ **and** A: $\forall A_R\ \Psi_R.\ (extractFrame\ R = \langle A_R, \Psi_R\rangle \wedge A_R \sharp{*}\ \Psi \wedge A_R \sharp{*}\ P \wedge A_R \sharp{*}\ Q) \longrightarrow \Psi \otimes \Psi_R \rhd P \approx Q$
by *auto*

obtain *p* **where** $(p \cdot xvec) \sharp{*}\ \Psi$
and $(p \cdot xvec) \sharp{*}\ P$
and $(p \cdot xvec) \sharp{*}\ Q$
and $(p \cdot xvec) \sharp{*}\ R$
and $(p \cdot xvec) \sharp{*}\ \Psi'$
and S: $(set\ p) \subseteq (set\ xvec) \times (set(p \cdot xvec))$ **and** *distinctPerm p*
by(*rule-tac c*=(Ψ, *P*, *Q*, *R*, Ψ') **in** *name-list-avoiding*) *auto*

from ‹$(p \cdot xvec) \sharp{*}\ P$› ‹$(p \cdot xvec) \sharp{*}\ R$› *S* **have** $(\!|\nu{*}xvec|\!)(P \parallel R) = (\!|\nu{*}(p \cdot xvec)|\!)(p \cdot (P \parallel R))$
by(*subst resChainAlpha*) *auto*
hence *PRAlpha*: $(\!|\nu{*}xvec|\!)(P \parallel R) = (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot P) \parallel (p \cdot R))$
by(*simp add*: *eqvts*)

from ‹$(p \cdot xvec) \sharp{*}\ Q$› ‹$(p \cdot xvec) \sharp{*}\ R$› *S* **have** $(\!|\nu{*}xvec|\!)(Q \parallel R) = (\!|\nu{*}(p \cdot xvec)|\!)(p \cdot (Q \parallel R))$
by(*subst resChainAlpha*) *auto*
hence *QRAlpha*: $(\!|\nu{*}xvec|\!)(Q \parallel R) = (\!|\nu{*}(p \cdot xvec)|\!)((p \cdot Q) \parallel (p \cdot R))$

by(*simp add*: *eqvts*)

from ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi'$› **have** $(\Psi \otimes \Psi', (\!|\nu*(p \cdot xvec)|\!)((p \cdot P) \parallel (p \cdot R)), (\!|\nu*(p \cdot xvec)|\!)((p \cdot Q) \parallel (p \cdot R))) \in ?X$
proof(*rule-tac C2*=$(\Psi, (p \cdot P), (p \cdot Q), R, \Psi', xvec, p \cdot xvec)$ **in** *XI'*, *auto*)
fix A_R Ψ_R
assume *FrR*: *extractFrame* $(p \cdot R) = \langle A_R, \Psi_R\rangle$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* \Psi'$ **and** $A_R \sharp* (p \cdot P)$ **and** $A_R \sharp* (p \cdot Q)$
from *FrR* **have** $(p \cdot (extractFrame\ (p \cdot R))) = (p \cdot \langle A_R, \Psi_R\rangle)$ **by** *simp*
with ‹*distinctPerm p*› **have** *extractFrame* $R = \langle p \cdot A_R, p \cdot \Psi_R\rangle$ **by**(*simp add*: *eqvts*)
moreover from ‹$A_R \sharp* \Psi$› **have** $(p \cdot A_R) \sharp* (p \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S **have** $(p \cdot A_R) \sharp* \Psi$ **by** *simp*
moreover from ‹$A_R \sharp* (p \cdot P)$› **have** $(p \cdot A_R) \sharp* (p \cdot p \cdot P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $(p \cdot A_R) \sharp* P$ **by** *simp*
moreover from ‹$A_R \sharp* (p \cdot Q)$› **have** $(p \cdot A_R) \sharp* (p \cdot p \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*distinctPerm p*› **have** $(p \cdot A_R) \sharp* Q$ **by** *simp*
ultimately have $\Psi \otimes (p \cdot \Psi_R) \rhd P \approx Q$ **using** A **by** *blast*
hence $(\Psi \otimes (p \cdot \Psi_R)) \otimes (p \cdot \Psi') \rhd P \approx Q$ **by**(*rule weakBisimE*)
moreover have $(\Psi \otimes (p \cdot \Psi_R)) \otimes (p \cdot \Psi') \simeq (\Psi \otimes (p \cdot \Psi')) \otimes (p \cdot \Psi_R)$
by(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)
ultimately have $(\Psi \otimes (p \cdot \Psi')) \otimes (p \cdot \Psi_R) \rhd P \approx Q$
by(*rule statEqWeakBisim*)
hence $(p \cdot ((\Psi \otimes (p \cdot \Psi')) \otimes (p \cdot \Psi_R))) \rhd (p \cdot P) \approx (p \cdot Q)$
by(*rule weakBisimClosed*)
with ‹*distinctPerm p*› ‹$xvec \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* \Psi$› S **show** $(\Psi \otimes \Psi') \otimes \Psi_R \rhd (p \cdot P) \approx (p \cdot Q)$
by(*simp add*: *eqvts*)
qed
with *PFrR QFrR PRAlpha QRAlpha* **show** *?case* **by** *simp*
next
case(*cSym* Ψ *PR QR*)
thus *?case* **by**(*blast dest*: *weakBisimE*)
qed
qed

lemma *weakBisimParPres*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and R :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \approx Q$

shows $\Psi \rhd P \parallel R \approx Q \parallel R$

proof −
  obtain A_R Ψ_R **where** *extractFrame* $R = \langle A_R, \Psi_R \rangle$ **and** $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$
    by(*rule-tac* C=(Ψ, P, Q) **in** *freshFrame*) *auto*
  moreover from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \otimes \Psi_R \rhd P \approx Q$ **by**(*rule weakBisimE*)
  ultimately show *?thesis* **by**(*rule-tac weakBisimParPresAux*)
qed

end

end

theory *Weak-Bisim-Struct-Cong*
  imports *Weak-Bisim-Pres Bisim-Struct-Cong*
begin

context *env* **begin**

lemma *weakBisimParComm*:
  fixes Ψ :: $'b$
  and P :: ($'a$, $'b$, $'c$) *psi*
  and Q :: ($'a$, $'b$, $'c$) *psi*

  shows $\Psi \rhd P \parallel Q \approx Q \parallel P$
by(*metis bisimParComm strongBisimWeakBisim*)

lemma *weakBisimResComm*:
  fixes x :: *name*
  and Ψ :: $'b$
  and y :: *name*
  and P :: ($'a$, $'b$, $'c$) *psi*

  assumes $x \sharp \Psi$
  and $y \sharp \Psi$

  shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \approx (\!|\nu y|\!)((\!|\nu x|\!)P)$
using *assms*
by(*metis bisimResComm strongBisimWeakBisim*)

lemma *weakBisimResComm'*:
  fixes x :: *name*
  and Ψ :: $'b$
  and *xvec* :: *name list*
  and P :: ($'a$, $'b$, $'c$) *psi*

  assumes $x \sharp \Psi$
  and *xvec* $\sharp* \Psi$

  shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu * xvec|\!)P) \approx (\!|\nu * xvec|\!)((\!|\nu x|\!)P)$

using *assms*
by(*metis bisimResComm′ strongBisimWeakBisim*)

lemma *weakBisimScopeExt*:
fixes x :: *name*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp P$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \approx P \parallel (\!|\nu x|\!)Q$
using *assms*
by(*metis bisimScopeExt strongBisimWeakBisim*)

lemma *weakBisimScopeExtChain*:
fixes *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes *xvec* $\sharp*$ Ψ
and *xvec* $\sharp*$ P

shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \approx P \parallel ((\!|\nu{*}xvec|\!)Q)$
using *assms*
by(*metis bisimScopeExtChain strongBisimWeakBisim*)

lemma *weakBisimParAssoc*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd (P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$
by(*metis bisimParAssoc strongBisimWeakBisim*)

lemma *weakBisimParNil*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd P \parallel \mathbf{0} \approx P$
by(*metis bisimParNil strongBisimWeakBisim*)

lemma *weakBisimResNil*:
fixes x :: *name*
and Ψ :: $'b$

assumes $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)\mathbf{0} \approx \mathbf{0}$
using *assms*
by(*metis bisimResNil strongBisimWeakBisim*)

lemma *weakBisimOutputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp M$
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M\langle N\rangle.P) \approx M\langle N\rangle.(\!|\nu x|\!)P$
using *assms*
by(*metis bisimOutputPushRes strongBisimWeakBisim*)

lemma *weakBisimInputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp M$
and $x \sharp$ *xvec*
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \approx M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
using *assms*
by(*metis bisimInputPushRes strongBisimWeakBisim*)

lemma *weakBisimCasePushRes*:
fixes x :: *name*
and Ψ :: $'b$
and *Cs* :: $('c \times ('a,\ 'b,\ 'c)$ *psi*$)$ *list*

assumes $x \sharp \Psi$
and $x \sharp$ (*map fst Cs*)

shows $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \approx Cases(map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs)$
using *assms*
by(*metis bisimCasePushRes strongBisimWeakBisim*)

lemma *weakBangExt*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes *guarded* P

shows $\Psi \rhd !P \approx P \parallel !P$
using *assms*
by(*metis bangExt strongBisimWeakBisim*)

lemma *weakBisimParSym*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd P \approx Q$

shows $\Psi \rhd R \parallel P \approx R \parallel Q$
using *assms*
by(*metis weakBisimParComm weakBisimParPres weakBisimTransitive*)

lemma *weakBisimScopeExtSym*:
fixes x :: *name*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp Q$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \approx ((\!|\nu x|\!)P) \parallel Q$
using *assms*
by(*metis weakBisimScopeExt weakBisimTransitive weakBisimParComm weakBisimE weakBisimResPres*)

lemma *weakBisimScopeExtChainSym*:
fixes *xvec* :: *name list*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes *xvec* $\sharp*$ Ψ
and *xvec* $\sharp*$ Q

shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \approx ((\!|\nu{*}xvec|\!)P) \parallel Q$
using *assms*
by(*induct xvec*) (*auto intro*: *weakBisimScopeExtSym weakBisimReflexive weakBisim-Transitive weakBisimResPres*)

lemma *weakBisimParPresAuxSym*:

fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \otimes \Psi_R \rhd P \approx Q$
and *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$
and $A_R \sharp* \Psi$
and $A_R \sharp* P$
and $A_R \sharp* Q$

shows $\Psi \rhd R \parallel P \approx R \parallel Q$
using *assms*
by(*metis weakBisimParComm weakBisimParPresAux weakBisimTransitive*)

lemma *weakBisimParPresSym*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd P \approx Q$

shows $\Psi \rhd R \parallel P \approx R \parallel Q$
using *assms*
by(*metis weakBisimParComm weakBisimParPres weakBisimTransitive*)

lemma *guardedFrameStatEq*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*

assumes *guarded* P

shows *extractFrame* $P \simeq_F \langle\varepsilon,\ \mathbf{1}\rangle$
proof −
obtain A_P Ψ_P **where** *FrR*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$
by(*rule freshFrame*)

with ‹*guarded* P› **have** $\Psi_P \simeq \mathbf{1}$ **and** ((*supp* Ψ_P)::*name set*) = {}
by(*metis guardedStatEq*)+
from ‹*supp* Ψ_P = {}› **have** $A_P \sharp* \Psi_P$ **by**(*auto simp add*: *fresh-star-def fresh-def*)
hence $\langle A_P,\ \Psi_P\rangle \simeq_F \langle[],\ \Psi_P\rangle$ **by**(*force intro*: *frameResFreshChain*)
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle[],\ \Psi_P\rangle \simeq_F \langle[],\ \mathbf{1}\rangle$
by(*simp add*: *FrameStatEq-def FrameStatImp-def AssertionStatEq-def AssertionStatImp-def*)
ultimately show *?thesis* **using** *FrR* **by**(*rule-tac FrameStatEqTrans*) *auto*
qed

lemma *guardedInsertAssertion*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*

and Ψ :: $'b$

assumes *guarded P*

shows *insertAssertion* (*extractFrame P*) $\Psi \simeq_F \langle\varepsilon, \Psi\rangle$
proof −
obtain A_P Ψ_P **where** *FrR*: *extractFrame P* = $\langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* \Psi$
by(*rule freshFrame*)

with ‹*guarded P*› **have** $\Psi_P \simeq \mathbf{1}$ **and** ((*supp* Ψ_P)::*name set*) = {}
by(*metis guardedStatEq*)+
from ‹*supp* Ψ_P = {}› **have** $A_P \sharp* \Psi_P$ **by**(*auto simp add*: *fresh-star-def fresh-def*)
hence $\langle A_P, \Psi \otimes \Psi_P\rangle \simeq_F \langle[], \Psi \otimes \Psi_P\rangle$ **using** ‹$A_P \sharp* \Psi$› **by**(*force intro*: *frameRes-FreshChain*)
moreover from ‹$\Psi_P \simeq \mathbf{1}$› **have** $\langle\varepsilon, \Psi \otimes \Psi_P\rangle \simeq_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$ **by**(*force intro*: *compositionSym*)
moreover have $\langle\varepsilon, \Psi \otimes \mathbf{1}\rangle \simeq_F \langle\varepsilon, \Psi\rangle$ **by**(*force intro*: *Identity*)
ultimately show *?thesis* **using** *FrR* ‹$A_P \sharp* \Psi$›
by(*force intro*: *FrameStatEqTrans AssertionStatEqTrans*)
qed

lemma *insertDoubleAssertionStatEq'*:
fixes F :: $'b$ *frame*
and Ψ :: $'b$
and Ψ' :: $'b$

shows *insertAssertion*(*insertAssertion F* Ψ) $\Psi' \simeq_F$ (*insertAssertion F*) ($\Psi' \otimes \Psi$)
proof −
obtain A_F Ψ_F **where** $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* \Psi$ **and** $A_F \sharp* \Psi'$ **and** $A_F \sharp* (\Psi' \otimes \Psi)$
by(*rule-tac C*=(Ψ, Ψ') **in** *freshFrame*) *auto*
thus *?thesis*
by *auto* (*metis frameIntAssociativity FrameStatEqSym*)
qed

lemma *bangActE*:
assumes $\Psi \rhd !P \longmapsto\alpha \prec P'$
and *bn* $\alpha \sharp*$ *subject* α
and *guarded P*
and $\alpha \neq \tau$
and *bn* $\alpha \sharp* P$

obtains Q **where** $\Psi \rhd P \longmapsto\alpha \prec Q$ **and** $P' \sim Q \parallel !P$
proof −
assume $\bigwedge Q.\ [\![\Psi \rhd P \longmapsto\alpha \prec Q;\ P' \sim Q \parallel !P]\!] \Longrightarrow$ *thesis*
moreover from ‹$\Psi \rhd !P \longmapsto\alpha \prec P'$› ‹*bn* $\alpha \sharp*$ *subject* α› ‹$\alpha \neq \tau$› ‹*bn* $\alpha \sharp* P$› **have** $\exists Q.\ \Psi \rhd P \longmapsto\alpha \prec Q \wedge P' \sim Q \parallel !P$
proof(*nominal-induct rule*: *bangInduct'*)

case(*cAlpha* α P' p)
then obtain Q **where** $\Psi \rhd P \longmapsto\alpha \prec Q$ **and** $P' \sim Q \parallel (P \parallel !P)$ **by** *fastforce*
from ‹$\Psi \rhd P \longmapsto\alpha \prec Q$› **have** *distinct*(*bn* α) **by**(*rule boundOutputDistinct*)
have S: *set* $p \subseteq$ *set*(*bn* α) $\times$ *set*(*bn*($p \cdot \alpha$)) **by** *fact*
from ‹$\Psi \rhd P \longmapsto\alpha \prec Q$› ‹*bn*($p \cdot \alpha$) $\sharp*$ α› ‹*bn*($p \cdot \alpha$) $\sharp*$ P› ‹*bn* α $\sharp*$ *subject* α› ‹*distinct*(*bn* α)›
have *bn*($p \cdot \alpha$) $\sharp*$ Q **by**(*force dest*: *freeFreshChainDerivative*)
with ‹$\Psi \rhd P \longmapsto\alpha \prec Q$› ‹*bn*($p \cdot \alpha$) $\sharp*$ α› S ‹*bn* α $\sharp*$ *subject* α› ‹*distinct*(*bn* α)› **have** $\Psi \rhd P \longmapsto(p \cdot \alpha) \prec (p \cdot Q)$
by(*fastforce simp add*: *residualAlpha*)
moreover from ‹$P' \sim Q \parallel (P \parallel !P)$› **have** $(p \cdot \mathbf{1}) \rhd (p \cdot P') \sim (p \cdot (Q \parallel (P \parallel !P)))$
by(*rule bisimClosed*)
with ‹(*bn* α) $\sharp*$ P› ‹*bn*($p \cdot \alpha$) $\sharp*$ P› S **have** $(p \cdot P') \sim (p \cdot Q) \parallel (P \parallel !P)$
by(*simp add*: *eqvts*)
ultimately show *?case* **by** *blast*
next
case(*cPar1* α P')
from ‹*guarded* P› **have** $P' \parallel !P \sim P' \parallel (P \parallel !P)$ **by**(*metis bangExt bisimParPresSym*)
with ‹$\Psi \rhd P \longmapsto\alpha \prec P'$› **show** *?case* **by** *blast*
next
case(*cPar2* α P')
then obtain Q **where** *PTrans*: $\Psi \rhd P \longmapsto\alpha \prec Q$ **and** $P' \sim Q \parallel !P$ **by** *blast*
from ‹$P' \sim Q \parallel !P$› **have** $P \parallel P' \sim Q \parallel (P \parallel !P)$
by(*metis bisimParPresSym bisimParAssoc bisimTransitive bisimParComm*)
with *PTrans* **show** *?case* **by** *blast*
next
case *cComm1*
from ‹$\tau \neq \tau$› **have** *False* **by** *simp*
thus *?case* **by** *simp*
next
case *cComm2*
from ‹$\tau \neq \tau$› **have** *False* **by** *simp*
thus *?case* **by** *simp*
next
case(*cBang* α P')
then obtain Q **where** *PTrans*: $\Psi \rhd P \longmapsto\alpha \prec Q$ **and** $P' \sim Q \parallel (P \parallel !P)$ **by** *blast*
from ‹$P' \sim Q \parallel (P \parallel !P)$› ‹*guarded* P› **have** $P' \sim Q \parallel !P$ **by**(*metis bisimTransitive bisimParPresSym bangExt bisimSymmetric*)
with *PTrans* **show** *?case* **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *bangTauE*:
assumes $\Psi \rhd !P \longmapsto\tau \prec P'$
and *guarded* P

obtains Q **where** $\Psi \rhd P \parallel P \longmapsto\tau \prec Q$ **and** $P' \sim Q \parallel !P$
using *assms*
proof −
assume $\bigwedge Q.\ [\![\Psi \rhd P \parallel P \longmapsto\tau \prec Q;\ P' \sim Q \parallel !P]\!] \Longrightarrow$ *thesis*
moreover from *assms* **have** $\exists\, Q.\ \Psi \rhd P \parallel P \longmapsto\tau \prec Q \wedge P' \sim Q \parallel !P$
proof(*nominal-induct rule*: *bangTauInduct*)
case(*cPar1 P'*)
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* P$
by(*rule-tac* C=(Ψ, P) **in** *freshFrame*) *auto*
from ‹*guarded P*› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **by**(*drule-tac guardedStatEq*) *auto*
with ‹$\Psi \rhd P \longmapsto\tau \prec P'$› **have** $\Psi \otimes \Psi_P \rhd P \longmapsto\tau \prec P'$
by(*rule-tac statEqTransition*, *auto*) (*metis Identity AssertionStatEqSym compositionSym AssertionStatEqTrans*)
hence $\Psi \rhd P \parallel P \longmapsto\tau \prec P' \parallel P$ **using** *FrP* ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› **by**(*rule-tac Par1*) *auto*
moreover from ‹*guarded P*› **have** $P' \parallel !P \sim (P' \parallel P) \parallel (P \parallel !P)$
by(*metis bisimParPresSym bisimParAssoc bisimTransitive bisimParComm bangExt*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 P'*)
then obtain n Q **where** *PTrans*: $\Psi \rhd P \parallel P \longmapsto\tau \prec Q$ **and** $P' \sim Q \parallel !P$ **by** *blast*
from ‹$P' \sim Q \parallel !P$› **have** $P \parallel P' \sim Q \parallel (P \parallel !P)$
by(*metis bisimParPresSym bisimParAssoc bisimTransitive bisimParComm*)
with *PTrans* **show** *?case* **by** *blast*
next
case(*cComm1 M N P' K xvec P''*)
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp* \Psi$ **and** $A_P \sharp* P$ **and** $A_P \sharp* M$
by(*rule-tac* C=(Ψ, P, M) **in** *freshFrame*) *auto*
from ‹*guarded P*› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **by**(*drule-tac guardedStatEq*) *auto*
obtain $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P = \langle A_P{}',\ \Psi_P{}'\rangle$ **and** $A_P{}' \sharp* \Psi$ **and** $A_P{}' \sharp* P$ **and** $A_P{}' \sharp* K$ **and** $A_P{}' \sharp* A_P$
by(*rule-tac* C=(Ψ, P, K, A_P) **in** *freshFrame*) *auto*
from ‹*guarded P*› *FrP'* **have** $\Psi_P{}' \simeq \mathbf{1}$ **by**(*drule-tac guardedStatEq*) *auto*
with ‹$\Psi \rhd P \longmapsto M(\!|N|\!) \prec P'$› **have** $\Psi \otimes \Psi_P{}' \rhd P \longmapsto M(\!|N|\!) \prec P'$
by(*rule-tac statEqTransition*, *auto*) (*metis Identity AssertionStatEqSym compositionSym AssertionStatEqTrans*)
moreover from ‹$\Psi \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$› ‹*guarded P*› ‹*xvec* $\sharp*$ *P*› ‹*xvec* $\sharp*$ *K*›
obtain Q **where** *PTrans*: $\Psi \rhd P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q$ **and** $P'' \sim Q \parallel !P$
by(*drule-tac bangActE*) *auto*
from *PTrans* ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \rhd P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q$
by(*rule-tac statEqTransition*, *auto*) (*metis Identity AssertionStatEqSym compositionSym AssertionStatEqTrans*)
moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi_P{}' \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \Psi_P{}' \vdash$

$M \leftrightarrow K$
by(*rule-tac statEqEnt*, *auto*) (*metis Identity AssertionStatEqSym composition-Sym AssertionStatEqTrans*)
ultimately have $\Psi \rhd P \parallel P \longmapsto\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q)$ **using** *FrP FrP'* ‹$A_P \sharp{*} \Psi$› ‹$A_P \sharp{*} P$› ‹$A_P \sharp{*} M$› ‹$A_P' \sharp{*} \Psi$› ‹$A_P' \sharp{*} P$› ‹$A_P' \sharp{*} K$› ‹$A_P' \sharp{*} A_P$› ‹$xvec \sharp{*} P$›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
moreover from ‹$P'' \sim Q \parallel !P$› ‹*guarded P*› **have** $P' \parallel P'' \sim (P' \parallel Q) \parallel (P \parallel !P)$
by(*metis bisimTransitive bangExt bisimParPresSym bisimParAssoc bisimSym-metric*)
hence $(\!|\nu{*}xvec|\!)(P' \parallel P'') \sim (\!|\nu{*}xvec|\!)((P' \parallel Q) \parallel (P \parallel !P))$ **by**(*rule-tac bisim-ResChainPres*) *auto*
with ‹$xvec \sharp{*} P$› ‹$xvec \sharp{*} \Psi$› **have** $(\!|\nu{*}xvec|\!)(P' \parallel P'') \sim ((\!|\nu{*}xvec|\!)(P' \parallel Q)) \parallel (P \parallel !P)$
by(*force intro*: *bisimTransitive bisimScopeExtChainSym*)
ultimately show *?case* **by** *blast*
next
case(*cComm2 M N P' K xvec P''*)
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp{*} \Psi$ **and** $A_P \sharp{*} P$ **and** $A_P \sharp{*} M$
by(*rule-tac* $C{=}(\Psi, P, M)$ **in** *freshFrame*) *auto*
from ‹*guarded P*› *FrP* **have** $\Psi_P \simeq \mathbf{1}$ **by**(*drule-tac guardedStatEq*) *auto*
obtain A_P' Ψ_P' **where** *FrP'*: *extractFrame* $P = \langle A_P', \Psi_P'\rangle$ **and** $A_P' \sharp{*} \Psi$ **and** $A_P' \sharp{*} P$ **and** $A_P' \sharp{*} K$ **and** $A_P' \sharp{*} A_P$
by(*rule-tac* $C{=}(\Psi, P, K, A_P)$ **in** *freshFrame*) *auto*
from ‹*guarded P*› *FrP'* **have** $\Psi_P' \simeq \mathbf{1}$ **by**(*drule-tac guardedStatEq*) *auto*
with ‹$\Psi \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$› **have** $\Psi \otimes \Psi_P' \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule-tac statEqTransition*, *auto*) (*metis Identity AssertionStatEqSym compositionSym AssertionStatEqTrans*)
moreover from ‹$\Psi \rhd !P \longmapsto K(\!|N|\!) \prec P''$› ‹*guarded P*›
obtain Q **where** *PTrans*: $\Psi \rhd P \longmapsto K(\!|N|\!) \prec Q$ **and** $P'' \sim Q \parallel !P$ **by**(*rule-tac bangActE*) *auto*
from *PTrans* ‹$\Psi_P \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \rhd P \longmapsto K(\!|N|\!) \prec Q$
by(*rule-tac statEqTransition*, *auto*) (*metis Identity AssertionStatEqSym compositionSym AssertionStatEqTrans*)
moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹$\Psi_P \simeq \mathbf{1}$› ‹$\Psi_P' \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \Psi_P' \vdash M \leftrightarrow K$
by(*rule-tac statEqEnt*, *auto*) (*metis Identity AssertionStatEqSym composition-Sym AssertionStatEqTrans*)
ultimately have $\Psi \rhd P \parallel P \longmapsto\tau \prec (\!|\nu{*}xvec|\!)(P' \parallel Q)$ **using** *FrP FrP'* ‹$A_P \sharp{*} \Psi$› ‹$A_P \sharp{*} P$› ‹$A_P \sharp{*} M$› ‹$A_P' \sharp{*} \Psi$› ‹$A_P' \sharp{*} P$› ‹$A_P' \sharp{*} K$› ‹$A_P' \sharp{*} A_P$› ‹$xvec \sharp{*} P$›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
moreover from ‹$P'' \sim Q \parallel !P$› ‹*guarded P*› **have** $P' \parallel P'' \sim (P' \parallel Q) \parallel (P \parallel !P)$
by(*metis bisimTransitive bangExt bisimParPresSym bisimParAssoc bisimSym-metric*)

hence $(\!|\nu{*}xvec|\!)(P' \parallel P'') \sim (\!|\nu{*}xvec|\!)((P' \parallel Q) \parallel (P \parallel !P))$ **by**(*rule-tac bisimResChainPres*) *auto*
with ‹*xvec* $\sharp*$ *P*› ‹*xvec* $\sharp*$ Ψ› **have** $(\!|\nu{*}xvec|\!)(P' \parallel P'') \sim ((\!|\nu{*}xvec|\!)(P' \parallel Q)) \parallel (P \parallel !P)$
by(*force intro*: *bisimTransitive bisimScopeExtChainSym*)
ultimately show *?case* **by** *blast*
next
case(*cBang P'*)
then obtain *Q* **where** *PTrans*: $\Psi \rhd P \parallel P \longmapsto\tau \prec Q$ **and** $P' \sim Q \parallel (P \parallel !P)$ **by** *blast*
from ‹$P' \sim Q \parallel (P \parallel !P)$› ‹*guarded P*› **have** $P' \sim Q \parallel !P$ **by**(*metis bisimTransitive bisimParPresSym bangExt bisimSymmetric*)
with *PTrans* **show** *?case* **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *tauBangI*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *P'* :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \parallel P \longmapsto\tau \prec P'$
and *guarded P*

obtains *Q* **where** $\Psi \rhd !P \longmapsto\tau \prec Q$ **and** $Q \sim P' \parallel !P$
proof −
assume $\bigwedge Q.\ [\![\Psi \rhd !P \longmapsto\tau \prec Q;\ Q \sim P' \parallel !P]\!] \Longrightarrow$ *thesis*
moreover from ‹$\Psi \rhd P \parallel P \longmapsto\tau \prec P'$› **have** $\exists\, Q.\ \Psi \rhd !P \longmapsto\tau \prec Q \wedge Q \sim P' \parallel !P$
proof(*induct rule*: *parTauCases*[**where** *C*=()])
case(*cPar1 P' A_P Ψ_P*)
from ‹$\Psi \otimes \Psi_P \rhd P \longmapsto\tau \prec P'$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \rhd P \longmapsto\tau \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_P \rhd P \parallel !P \longmapsto\tau \prec P' \parallel !P$ **by**(*rule-tac Par1*) *auto*
hence $\Psi \otimes \Psi_P \rhd !P \longmapsto\tau \prec P' \parallel !P$ **using** ‹*guarded P*› **by**(*rule Bang*)
hence $\Psi \rhd P \parallel !P \longmapsto\tau \prec P \parallel (P' \parallel !P)$ **using** ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$› ‹$A_P$ $\sharp*$ Ψ› ‹A_P $\sharp*$ *P*›
by(*rule-tac Par2*) *auto*
hence $\Psi \rhd !P \longmapsto\tau \prec P \parallel (P' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover have $P \parallel (P' \parallel !P) \sim P' \parallel P \parallel !P$
by(*metis bisimParAssoc bisimParComm bisimTransitive bisimSymmetric bisimParPres*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 P' A_P Ψ_P*)
from ‹$\Psi \otimes \Psi_P \rhd P \longmapsto\tau \prec P'$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \rhd P \longmapsto\tau \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_P \rhd P \parallel !P \longmapsto\tau \prec P' \parallel !P$ **by**(*rule-tac Par1*) *auto*

hence $\Psi \otimes \Psi_P \rhd !P \longmapsto\tau \prec P' \parallel !P$ **using** ‹*guarded P*› **by**(*rule Bang*)
hence $\Psi \rhd P \parallel !P \longmapsto\tau \prec P \parallel (P' \parallel !P)$ **using** ‹*extractFrame P* = $\langle A_P, \Psi_P\rangle$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$›
by(*rule-tac Par2*) *auto*
hence $\Psi \rhd !P \longmapsto\tau \prec P \parallel (P' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover have $P \parallel (P' \parallel !P) \sim P \parallel P' \parallel !P$
by(*metis bisimParAssoc bisimSymmetric*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* $\Psi_P{}'$ M N P' A_P Ψ_P K *xvec* P'' $A_P{}'$)
from ‹*extractFrame P* = $\langle A_P{}', \Psi_P{}'\rangle$› ‹*guarded P*› **have** $\Psi_P{}' \simeq \mathbf{1}$ **by**(*blast dest*: *guardedStatEq*)
with ‹$\Psi \otimes \Psi_P{}' \rhd P \longmapsto M(\!|N|\!) \prec P'$› **have** $\Psi \otimes \mathbf{1} \rhd P \longmapsto M(\!|N|\!) \prec P'$
by(*rule-tac statEqTransition*, *auto*) (*metis compositionSym AssertionStatEqSym*)
moreover note ‹*extractFrame P* = $\langle A_P, \Psi_P\rangle$›
moreover from ‹$\Psi \otimes \Psi_P \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P''$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P''$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_P \rhd P \parallel !P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec (P'' \parallel !P)$ **using** ‹*xvec* $\sharp*$ *P*› **by**(*force intro*: *Par1*)
hence $\Psi \otimes \Psi_P \rhd !P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec (P'' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_P{}' \vdash M \leftrightarrow K$› ‹$\Psi_P{}' \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*rule-tac statEqEnt*, *auto*) (*metis compositionSym AssertionStatEqSym*)
ultimately have $\Psi \rhd P \parallel !P \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel (P'' \parallel !P))$
using ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* A_P{}'$› ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* P$› ‹$A_P{}' \sharp* K$› ‹*xvec* $\sharp*$ *P*›
by(*force intro*: *Comm1*)
hence $\Psi \rhd !P \longmapsto\tau \prec (\!|\nu*xvec|\!)(P' \parallel (P'' \parallel !P))$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover have $(\!|\nu*xvec|\!)(P' \parallel (P'' \parallel !P)) \sim ((\!|\nu*xvec|\!)(P' \parallel P'')) \parallel !P$
proof −
have $(\!|\nu*xvec|\!)(P' \parallel (P'' \parallel !P)) \sim (\!|\nu*xvec|\!)((P' \parallel P'') \parallel !P)$
by(*force intro*: *bisimResChainPres bisimParAssoc*[*THEN bisimSymmetric*])
moreover have $(\!|\nu*xvec|\!)((P' \parallel P'') \parallel !P) \sim ((\!|\nu*xvec|\!)(P' \parallel P'')) \parallel !P$ **using** ‹*xvec* $\sharp*$ *P*›
by(*rule-tac bisimScopeExtChainSym*) *auto*
ultimately show *?thesis* **by**(*rule bisimTransitive*)
qed
ultimately show *?case* **by** *blast*
next
case(*cComm2* $\Psi_P{}'$ M *xvec* N P' A_P Ψ_P K P'' $A_P{}'$)
from ‹*extractFrame P* = $\langle A_P{}', \Psi_P{}'\rangle$› ‹*guarded P*› **have** $\Psi_P{}' \simeq \mathbf{1}$ **by**(*blast dest*: *guardedStatEq*)
with ‹$\Psi \otimes \Psi_P{}' \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$› **have** $\Psi \otimes \mathbf{1} \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*rule-tac statEqTransition*, *auto*) (*metis compositionSym AssertionStatE-*

qSym)
moreover note ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$›
moreover from ‹$\Psi \otimes \Psi_P \rhd P \longmapsto K(\!|N|\!) \prec P''$› **have** $(\Psi \otimes \Psi_P) \otimes \mathbf{1} \rhd P \longmapsto K(\!|N|\!) \prec P''$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_P \rhd P \parallel !P \longmapsto K(\!|N|\!) \prec (P'' \parallel !P)$ **by**(*force intro*: *Par1*)
hence $\Psi \otimes \Psi_P \rhd !P \longmapsto K(\!|N|\!) \prec (P'' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_P{}' \vdash M \leftrightarrow K$› ‹$\Psi_P{}' \simeq \mathbf{1}$› **have** $\Psi \otimes \Psi_P \otimes \mathbf{1} \vdash M \leftrightarrow K$
by(*rule-tac statEqEnt*, *auto*) (*metis compositionSym AssertionStatEqSym*)
ultimately have $\Psi \rhd P \parallel !P \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(P' \parallel (P'' \parallel !P))$
using ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* P$› ‹$A_P \sharp* M$› ‹$A_P \sharp* A_P{}'$› ‹$A_P{}' \sharp* \Psi$› ‹$A_P{}' \sharp* P$› ‹$A_P{}' \sharp* K$› ‹$xvec \sharp* P$›
by(*force intro*: *Comm2*)
hence $\Psi \rhd !P \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(P' \parallel (P'' \parallel !P))$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover have $(\!|\nu{*}xvec|\!)(P' \parallel (P'' \parallel !P)) \sim ((\!|\nu{*}xvec|\!)(P' \parallel P'')) \parallel !P$
proof −
have $(\!|\nu{*}xvec|\!)(P' \parallel (P'' \parallel !P)) \sim (\!|\nu{*}xvec|\!)((P' \parallel P'') \parallel !P)$
by(*force intro*: *bisimResChainPres bisimParAssoc*[*THEN bisimSymmetric*])
moreover have $(\!|\nu{*}xvec|\!)((P' \parallel P'') \parallel !P) \sim ((\!|\nu{*}xvec|\!)(P' \parallel P'')) \parallel !P$ **using** ‹$xvec \sharp* P$›
by(*rule-tac bisimScopeExtChainSym*) *auto*
ultimately show *?thesis* **by**(*rule bisimTransitive*)
qed
ultimately show *?case* **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *tauChainBangI*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and P' :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \parallel P \Longrightarrow\hat{}_\tau P'$
and *guarded P*

obtains Q **where** $\Psi \rhd !P \Longrightarrow\hat{}_\tau Q$ **and** $\Psi \rhd Q \sim P' \parallel !P$
proof −
assume $\bigwedge Q.\ [\![\Psi \rhd !P \Longrightarrow\hat{}_\tau Q;\ \Psi \rhd Q \sim P' \parallel !P]\!] \Longrightarrow$ *thesis*
moreover from ‹$\Psi \rhd P \parallel P \Longrightarrow\hat{}_\tau P'$› **have** $\exists Q.\ \Psi \rhd !P \Longrightarrow\hat{}_\tau Q \wedge \Psi \rhd Q \sim P' \parallel !P$
proof(*induct x1*==$P \parallel P\ P'$ *rule*: *tauChainInduct*)
case *TauBase*
have $\Psi \rhd !P \Longrightarrow\hat{}_\tau !P$ **by** *simp*
moreover have $\Psi \rhd !P \sim (P \parallel P) \parallel !P$ **using** ‹*guarded P*›
by(*metis bisimParAssoc bisimTransitive bisimParPresSym bangExt bisimParComm*)

ultimately show *?case* **by** *blast*
next
case(*TauStep* R' R'')
then obtain Q **where** *PChain*: $\Psi \rhd !P \Longrightarrow\hat{}_\tau\ Q$ **and** $\Psi \rhd Q \sim R' \parallel !P$ **by** *auto*
from ‹$\Psi \rhd R' \longmapsto\tau \prec R''$› **have** $\Psi \otimes \mathbf{1} \rhd R' \longmapsto\tau \prec R''$ **by**(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \rhd R' \parallel !P \longmapsto\tau \prec R'' \parallel !P$ **by**(*rule-tac Par1*) *auto*
with ‹$\Psi \rhd Q \sim R' \parallel !P$› **obtain** Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto\tau \prec Q'$ **and** $\Psi \rhd Q' \sim R'' \parallel !P$
by(*force dest*: *bisimE*(*2*) *simE*)
from *PChain QTrans* **have** $\Psi \rhd !P \Longrightarrow\hat{}_\tau\ Q'$ **by**(*auto dest*: *tauActTauChain*)
thus *?case* **using** ‹$\Psi \rhd Q' \sim R'' \parallel !P$› **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *weakBisimBangPresAux*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \approx Q$
and *guarded* P
and *guarded* Q

shows $\Psi \rhd R \parallel !P \approx R \parallel !Q$
proof −
let *?X* = $\{(\Psi, R \parallel !P, R \parallel !Q) \mid \Psi\ R\ P\ Q.\ guarded\ P \wedge guarded\ Q \wedge \Psi \rhd P \approx Q\}$
let *?Y* = $\{(\Psi, P, Q) \mid \Psi\ P\ Q.\ \exists R\ S.\ \Psi \rhd P \approx R \wedge (\Psi, R, S) \in ?X \wedge \Psi \rhd S \sim Q\}$

from *assms* **have** $(\Psi, R \parallel !P, R \parallel !Q) \in ?X$ **by** *auto*
moreover have *eqvt ?X*
by(*fastforce simp add*: *eqvt-def intro*: *weakBisimClosed*)
ultimately show *?thesis*
proof(*coinduct rule*: *weakTransitiveCoinduct2*)
case(*cStatImp* Ψ P Q)
thus *?case* **by**(*force dest*: *weakBisimE*(*3*) *simp add*: *weakStatImp-def*)
next
case(*cSim* Ψ P Q)
moreover {
fix Ψ P Q R
assume $\Psi \rhd P \approx Q$
moreover have *eqvt ?Y*
apply(*auto simp add*: *eqvt-def*)
apply(*rule-tac x=p* · $(Ra \parallel !P)$ **in** *exI*, *auto*)

apply(*fastforce dest*: *weakBisimClosed simp add*: *eqvts*)
apply(*rule-tac x*=$(p \cdot Ra) \parallel !(p \cdot Q)$ **in** *exI*, *auto*)
apply(*rule-tac x*=$p \cdot Ra$ **in** *exI*)
apply(*rule-tac x*=$p \cdot P$ **in** *exI*, *auto*)
apply(*rule-tac x*=$p \cdot Q$ **in** *exI*, *auto*)
apply(*blast intro*: *weakBisimClosed*)
by(*fastforce dest*: *bisimClosed simp add*: *eqvts*)
moreover assume *guarded P* **and** *guarded Q*
moreover note *weakBisimClosed bisimClosed weakBisimE*(*3*) *bisimE*(*3*) *weakBisimE*(*2*) *weakBisimE*(*4*) *bisimE*(*4*) *statEqWeakBisim statEqBisim weakBisimTransitive bisimTransitive weakBisimParAssoc*[*THEN weakBisimE*(*4*)] *bisimParAssoc*[*THEN bisimE*(*4*)] *weakBisimParPres*
moreover have $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \approx Q \Longrightarrow \Psi \rhd P \parallel P \approx Q \parallel Q$
by(*metis weakBisimParPres weakBisimParComm weakBisimE*(*4*) *weakBisimTransitive*)
moreover note *bisimParPresSym*
moreover have *bisim* $\subseteq$ *weakBisim* **by**(*auto dest*: *strongBisimWeakBisim*)
moreover have $\bigwedge\Psi\ \Psi_R\ P\ Q\ R\ A_R.\ [\![\Psi \otimes \Psi_R \rhd P \approx Q;\ extractFrame\ R = \langle A_R, \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q]\!] \Longrightarrow \Psi \rhd R \parallel P \approx R \parallel Q$
by(*metis weakBisimParComm weakBisimTransitive weakBisimParPresAux*)
moreover note *weakBisimResChainPres bisimResChainPres weakBisimScopeExtChainSym bisimScopeExtChainSym*
moreover have $\bigwedge\Psi\ P\ R\ S\ Q.\ [\![\Psi \rhd P \approx R;\ (\Psi, R, S) \in ?Y;\ \Psi \rhd S \sim Q]\!] \Longrightarrow (\Psi, P, Q) \in ?Y$
by(*blast dest*: *weakBisimTransitive bisimTransitive*)
moreover have $\bigwedge\Psi\ P\ Q\ R.\ [\![\Psi \rhd P \approx Q;\ guarded\ P;\ guarded\ Q]\!] \Longrightarrow (\Psi, R \parallel !P, R \parallel !Q) \in ?Y$
by(*blast intro*: *bisimReflexive weakBisimReflexive*)
moreover from *bangActE* **have** $\bigwedge\Psi\ P\ \alpha\ P'.\ [\![\Psi \rhd !P \longmapsto\alpha \prec P';\ bn\ \alpha \sharp* P;\ guarded\ P;\ \alpha \neq \tau;\ bn\ \alpha \sharp* subject\ \alpha]\!] \Longrightarrow \exists\, Q.\ \Psi \rhd P \longmapsto\alpha \prec Q \wedge P' \sim Q \parallel !P$
by *blast*
moreover from *bangTauE* **have** $\bigwedge\Psi\ P\ P'.\ [\![\Psi \rhd !P \longmapsto\tau \prec P';\ guarded\ P]\!] \Longrightarrow \exists\, Q.\ \Psi \rhd P \parallel P \longmapsto\tau \prec Q \wedge P' \sim Q \parallel !P$
by *blast*
moreover from *tauChainBangI* **have** $\bigwedge\Psi\ P\ P'.\ [\![\Psi \rhd P \parallel P \Longrightarrow\hat{}_\tau\ P';\ guarded\ P]\!] \Longrightarrow \exists\, Q.\ \Psi \rhd !P \Longrightarrow\hat{}_\tau\ Q \wedge \Psi \rhd Q \sim P' \parallel !P$
by *blast*
ultimately have $\Psi \rhd R \parallel !P \rightsquigarrow <?Y> R \parallel !Q$
by(*rule-tac weakSimBangPres*)
}
ultimately show *?case* **by** *blast*
next
case(*cExt* $\Psi\ P\ Q\ \Psi'$)
thus *?case* **by**(*blast dest*: *weakBisimE*)
next
case(*cSym* $\Psi\ P\ Q$)
thus *?case* **by**(*blast dest*: *weakBisimE*)
qed

qed

lemma *weakBisimBangPres*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*

  assumes $\Psi \rhd P \approx Q$
  and *guarded* P
  and *guarded* Q

  shows $\Psi \rhd !P \approx !Q$
proof −
  from *assms* **have** $\Psi \rhd \mathbf{0} \parallel !P \approx \mathbf{0} \parallel !Q$ **by**(*rule weakBisimBangPresAux*)
  thus *?thesis*
    by(*metis weakBisimParNil weakBisimParComm weakBisimTransitive weakBisimE*(*4*))
qed

end

end

theory *Close-Subst*
  imports *Agent*
begin

context *substPsi*
begin

definition *closeSubst* :: ($'b$::*fs-name* × ($'a$::*fs-name*, $'b$, $'c$::*fs-name*) *psi* × ($'a$, $'b$, $'c$) *psi*) *set* ⇒ ($'b$ × ($'a$, $'b$, $'c$) *psi* × ($'a$, $'b$, $'c$) *psi*) *set*
where *closeSubst Rel* ≡ {(Ψ, P, Q) | Ψ P Q. (∀σ. *wellFormedSubst* σ ⟶ (Ψ, P[<σ>], Q[<σ>]) ∈ *Rel*)}

lemma *closeSubstI*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*

  assumes $\bigwedge\sigma$. *wellFormedSubst* $\sigma \Longrightarrow (\Psi,\ P[<\sigma>],\ Q[<\sigma>]) \in Rel$

  shows (Ψ, P, Q) ∈ *closeSubst Rel*
using *assms*
by(*unfold closeSubst-def*) *auto*

lemma *closeSubstE*:
  fixes Ψ :: $'b$

and P :: ($'a$, $'b$, $'c$) *psi*
and Q :: ($'a$, $'b$, $'c$) *psi*
and σ :: (*name list* × $'a$ *list*) *list*

assumes (Ψ, P, Q) ∈ *closeSubst Rel*
and *wellFormedSubst* σ

shows (Ψ, P[<σ>], Q[<σ>]) ∈ *Rel*
using *assms*
by(*unfold closeSubst-def*) *auto*

lemma *closeSubstClosed*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and Q :: ($'a$, $'b$, $'c$) *psi*
and p :: *name prm*

assumes *eqvt Rel*
and (Ψ, P, Q) ∈ *closeSubst Rel*

shows (p · Ψ, p · P, p · Q) ∈ *closeSubst Rel*
proof(*rule closeSubstI*)
fix σ
assume *wellFormedSubst*(σ::(*name list* × $'a$ *list*) *list*)
with ‹(Ψ, P, Q) ∈ *closeSubst Rel*› ‹*wellFormedSubst* σ›
have (Ψ, P[<(*rev* p · σ)>], Q[<(*rev* p · σ)>]) ∈ *Rel*
by(*rule-tac closeSubstE*) *auto*
hence (p · Ψ, p · (P[<(*rev* p · σ)>]), p · (Q[<(*rev* p · σ)>])) ∈ *Rel*
by(*drule-tac p=p* **in** *eqvtI*[*OF* ‹*eqvt Rel*›]) (*simp add*: *eqvts*)
thus (p · Ψ, (p · P)[<σ>], (p · Q)[<σ>]) ∈ *Rel*
by(*simp del*: *seqSubs-def add*: *eqvts*)
qed

lemma *closeSubstEqvt*:
assumes *eqvt Rel*

shows *eqvt*(*closeSubst Rel*)
proof(*auto simp add*: *eqvt-def*)
fix Ψ P Q p
assume (Ψ, P, Q) ∈ *closeSubst Rel*
thus ((p::*name prm*) · Ψ, p · P, p · Q) ∈ *closeSubst Rel*
by(*drule-tac p=p* **in** *closeSubstClosed*[*OF* ‹*eqvt Rel*›]) (*simp add*: *eqvts*)
qed

lemma *closeSubstUnfold*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and Q :: ($'a$, $'b$, $'c$) *psi*
and σ :: (*name list* × $'a$ *list*) *list*

assumes $(\Psi,\ P,\ Q) \in$ *closeSubst Rel*
and *wellFormedSubst* σ

shows $(\Psi,\ P[<\sigma>],\ Q[<\sigma>]) \in$ *closeSubst Rel*
proof(*rule closeSubstI*)
fix σ'::(*name list* $\times$ $'a$ *list*) *list*
assume *wellFormedSubst* σ'
with ‹*wellFormedSubst* σ› **have** *wellFormedSubst*$(\sigma@\sigma')$ **by** *simp*
with ‹$(\Psi,\ P,\ Q) \in$ *closeSubst Rel*› **have** $(\Psi,\ P[<(\sigma@\sigma')>],\ Q[<(\sigma@\sigma')>]) \in$ *Rel*
by(*rule closeSubstE*)
thus $(\Psi,\ P[<\sigma>][<\sigma'>],\ Q[<\sigma>][<\sigma'>]) \in$ *Rel*
by *simp*
qed

end

end

theory *Bisim-Subst*
imports *Bisim-Struct-Cong Close-Subst*
begin

context *env* **begin**

abbreviation
bisimSubstJudge (‹- $\triangleright$ - $\sim_s$ -› [*70*, *70*, *70*] *65*) **where** $\Psi \triangleright P \sim_s Q \equiv (\Psi,\ P,\ Q) \in$ *closeSubst bisim*
abbreviation
bisimSubstNilJudge (‹- $\sim_s$ -› [*70*, *70*] *65*) **where** $P \sim_s Q \equiv$ *SBottom'* $\triangleright P \sim_s Q$

lemmas *bisimSubstClosed*[*eqvt*] = *closeSubstClosed*[*OF bisimEqvt*]
lemmas *bisimSubstEqvt*[*simp*] = *closeSubstEqvt*[*OF bisimEqvt*]

lemma *bisimSubstOutputPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and M :: $'a$
and N :: $'a$

assumes $\Psi \triangleright P \sim_s Q$

shows $\Psi \triangleright M\langle N\rangle.P \sim_s M\langle N\rangle.Q$
using *assms*
by(*fastforce intro*: *closeSubstI closeSubstE bisimOutputPres*)

lemma *seqSubstInputChain*[*simp*]:
  fixes *xvec* :: *name list*
  and N :: $'a$
  and P :: $('a, 'b, 'c)$ *psi*
  and σ :: (*name list* × $'a$ *list*) *list*

  assumes *xvec* $\sharp*$ σ

  shows *seqSubs'* (*inputChain xvec N P*) σ = *inputChain xvec* (*substTerm.seqSubst N* σ) (*seqSubs P* σ)
using *assms*
by(*induct xvec*) *auto*

lemma *bisimSubstInputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and *xvec* :: *name list*
  and N :: $'a$

  assumes $\Psi \rhd P \sim_s Q$
  and *xvec* $\sharp*$ Ψ
  and *distinct xvec*

  shows $\Psi \rhd M(\!|\lambda*xvec\ N|\!).P \sim_s M(\!|\lambda*xvec\ N|\!).Q$
proof(*rule-tac closeSubstI*)
  fix σ
  assume *wellFormedSubst*(σ::(*name list* × $'a$ *list*) *list*)
  obtain p **where** $(p \cdot xvec) \sharp* \sigma$
        and $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* N$
        and S: *set* $p \subseteq$ *set xvec* × *set* $(p \cdot xvec)$
    by(*rule-tac* c=(σ, P, Q, Ψ, N) **in** *name-list-avoiding*) *auto*

  from ‹$\Psi \rhd P \sim_s Q$› **have** $(p \cdot \Psi) \rhd (p \cdot P) \sim_s (p \cdot Q)$
    by(*rule bisimSubstClosed*)
  with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec) \sharp* \Psi$› S **have** $\Psi \rhd (p \cdot P) \sim_s (p \cdot Q)$
    by *simp*

  {
    fix *Tvec* :: $'a$ *list*
    from ‹$\Psi \rhd (p \cdot P) \sim_s (p \cdot Q)$› ‹*wellFormedSubst* σ› **have** $\Psi \rhd (p \cdot P)[<\sigma>] \sim_s (p \cdot Q)[<\sigma>]$
      by(*rule closeSubstUnfold*)
    moreover assume *length xvec* = *length Tvec* **and** *distinct xvec*
    ultimately have $\Psi \rhd ((p \cdot P)[<\sigma>])[(p \cdot xvec)::=Tvec] \sim ((p \cdot Q)[<\sigma>])[(p \cdot xvec)::=Tvec]$
      by(*drule-tac closeSubstE*[**where** σ=[$((p \cdot xvec), Tvec)$]]) *auto*

}

with ‹$(p \cdot xvec)$ $\sharp*$ σ› ‹*distinct xvec*›
have $\Psi \rhd (M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P))[<\sigma>] \sim (M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot Q))[<\sigma>]$
by(*force intro*: *bisimInputPres*)
moreover from ‹$(p \cdot xvec)$ $\sharp*$ N› ‹$(p \cdot xvec)$ $\sharp*$ P› S **have** $M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P) = M(\!|\lambda*xvec\ N|\!).P$
apply(*simp add*: *psi.inject*) **by**(*rule inputChainAlpha*[*symmetric*]) *auto*
moreover from ‹$(p \cdot xvec)$ $\sharp*$ N› ‹$(p \cdot xvec)$ $\sharp*$ Q› S **have** $M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot Q) = M(\!|\lambda*xvec\ N|\!).Q$
apply(*simp add*: *psi.inject*) **by**(*rule inputChainAlpha*[*symmetric*]) *auto*
ultimately show $\Psi \rhd (M(\!|\lambda*xvec\ N|\!).P)[<\sigma>] \sim (M(\!|\lambda*xvec\ N|\!).Q)[<\sigma>]$
by *force*
qed

lemma *bisimSubstCasePresAux*:
fixes Ψ :: $'b$
and CsP :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and CsQ :: $('c \times ('a, 'b, 'c)\ psi)\ list$

assumes $C1$: $\bigwedge\varphi\ P.\ (\varphi, P)\ mem\ CsP \Longrightarrow \exists\, Q.\ (\varphi, Q)\ mem\ CsQ \wedge guarded\ Q \wedge \Psi \rhd P \sim_s Q$
and $C2$: $\bigwedge\varphi\ Q.\ (\varphi, Q)\ mem\ CsQ \Longrightarrow \exists\, P.\ (\varphi, P)\ mem\ CsP \wedge guarded\ P \wedge \Psi \rhd P \sim_s Q$

shows $\Psi \rhd Cases\ CsP \sim_s Cases\ CsQ$
proof −
{
fix σ :: $(name\ list \times 'a\ list)\ list$

assume *wellFormedSubst* σ

have $\Psi \rhd Cases(caseListSeqSubst\ CsP\ \sigma) \sim Cases(caseListSeqSubst\ CsQ\ \sigma)$
proof(*rule bisimCasePres*)
fix φ P
assume $(\varphi, P)\ mem\ (caseListSeqSubst\ CsP\ \sigma)$
then obtain φ' P' **where** $(\varphi', P')\ mem\ CsP$ **and** $\varphi = substCond.seqSubst\ \varphi'\ \sigma$ **and** $PeqP'$: $P = (P'[<\sigma>])$
by(*induct CsP*) *force*+
from ‹(φ', P') *mem CsP*› **obtain** Q' **where** $(\varphi', Q')\ mem\ CsQ$ **and** $guarded\ Q'$ **and** $\Psi \rhd P' \sim_s Q'$ **by**(*blast dest*: *C1*)
from ‹(φ', Q') *mem CsQ*› ‹$\varphi = substCond.seqSubst\ \varphi'\ \sigma$› **obtain** Q **where** $(\varphi, Q)\ mem\ (caseListSeqSubst\ CsQ\ \sigma)$ **and** $Q = Q'[<\sigma>]$
by(*induct CsQ*) *auto*
with $PeqP'$ ‹$guarded\ Q'$› ‹$\Psi \rhd P' \sim_s Q'$› ‹*wellFormedSubst* σ› **show** $\exists\, Q.\ (\varphi, Q)\ mem\ (caseListSeqSubst\ CsQ\ \sigma) \wedge guarded\ Q \wedge \Psi \rhd P \sim Q$
by(*blast dest*: *closeSubstE guardedSeqSubst*)
next

fix φ Q
assume $(\varphi,\ Q)$ *mem* (*caseListSeqSubst CsQ* σ)
then obtain φ' Q' **where** $(\varphi',\ Q')$ *mem CsQ* **and** φ = *substCond.seqSubst* φ' σ **and** *QeqQ'*: $Q = Q'[<\sigma>]$
by(*induct CsQ*) *force+*
from ‹$(\varphi',\ Q')$ *mem CsQ*› **obtain** P' **where** $(\varphi',\ P')$ *mem CsP* **and** *guarded* P' **and** $\Psi \rhd P' \sim_s Q'$ **by**(*blast dest*: *C2*)
from ‹$(\varphi',\ P')$ *mem CsP*› ‹φ = *substCond.seqSubst* φ' σ› **obtain** P **where** $(\varphi,\ P)$ *mem* (*caseListSeqSubst CsP* σ) **and** $P = P'[<\sigma>]$
by(*induct CsP*) *auto*
with *QeqQ'* ‹*guarded* P'› ‹$\Psi \rhd P' \sim_s Q'$› ‹*wellFormedSubst* σ› **show** $\exists P.$ $(\varphi,\ P)$ *mem* (*caseListSeqSubst CsP* σ) $\wedge$ *guarded* $P \wedge \Psi \rhd P \sim Q$
by(*blast dest*: *closeSubstE guardedSeqSubst*)
qed
}
thus *?thesis*
by(*rule-tac closeSubstI*) *auto*
qed

lemma *bisimSubstReflexive*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd P \sim_s P$
by(*auto intro*: *closeSubstI bisimReflexive*)

lemma *bisimSubstTransitive*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and R :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \sim_s Q$
and $\Psi \rhd Q \sim_s R$

shows $\Psi \rhd P \sim_s R$
using *assms*
by(*auto intro*: *closeSubstI closeSubstE bisimTransitive*)

lemma *bisimSubstSymmetric*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \sim_s Q$

shows $\Psi \rhd Q \sim_s P$
using *assms*
by(*auto intro*: *closeSubstI closeSubstE bisimE*)

lemma *bisimSubstParPres*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*
  and R :: $('a,\ 'b,\ 'c)$ *psi*

  assumes $\Psi \rhd P \sim_s Q$

  shows $\Psi \rhd P \parallel R \sim_s Q \parallel R$
using *assms*
by(*fastforce intro*: *closeSubstI closeSubstE bisimParPres*)

lemma *bisimSubstResPres*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*
  and x :: *name*

  assumes $\Psi \rhd P \sim_s Q$
  and $x \sharp \Psi$

  shows $\Psi \rhd (\!|\nu x|\!)P \sim_s (\!|\nu x|\!)Q$
proof(*rule-tac closeSubstI*)
  fix σ :: (*name list* $\times$ $'a$ *list*) *list*

  assume *wellFormedSubst* σ
  obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \sigma$
    by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

  from ‹$\Psi \rhd P \sim_s Q$› **have** $([(x,\ y)] \cdot \Psi) \rhd ([(x,\ y)] \cdot P) \sim_s ([(x,\ y)] \cdot Q)$
    by(*rule bisimSubstClosed*)
  with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \rhd ([(x,\ y)] \cdot P) \sim_s ([(x,\ y)] \cdot Q)$
    by *simp*
  hence $\Psi \rhd ([(x,\ y)] \cdot P)[<\sigma>] \sim ([(x,\ y)] \cdot Q)[<\sigma>]$ **using** ‹*wellFormedSubst* σ›
    by(*rule closeSubstE*)
  hence $\Psi \rhd (\!|\nu y|\!)(([(x,\ y)] \cdot P)[<\sigma>]) \sim (\!|\nu y|\!)(([(x,\ y)] \cdot Q)[<\sigma>])$ **using** ‹$y \sharp \Psi$›
    by(*rule bisimResPres*)
  with ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \sharp \sigma$›
  show $\Psi \rhd ((\!|\nu x|\!)P)[<\sigma>] \sim ((\!|\nu x|\!)Q)[<\sigma>]$
    by(*simp add*: *alphaRes*)
qed

lemma *bisimSubstBangPres*:
  fixes Ψ :: $'b$
  and P :: $('a,\ 'b,\ 'c)$ *psi*
  and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd P \sim_s Q$
and *guarded* P
and *guarded* Q

shows $\Psi \rhd !P \sim_s !Q$
using *assms*
by(*fastforce intro*: *closeSubstI closeSubstE bisimBangPres guardedSeqSubst*)

lemma *substNil*[*simp*]:
fixes *xvec* :: *name list*
and *Tvec* :: *'a list*

assumes *wellFormedSubst* σ
and *distinct xvec*

shows $(\mathbf{0}[<\sigma>]) = \mathbf{0}$
using *assms*
by *simp*

lemma *bisimSubstParNil*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd P \parallel \mathbf{0} \sim_s P$
by(*fastforce intro*: *closeSubstI bisimParNil*)

lemma *bisimSubstParComm*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd P \parallel Q \sim_s Q \parallel P$
apply(*rule closeSubstI*)
by(*fastforce intro*: *closeSubstI bisimParComm*)

lemma *bisimSubstParAssoc*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and R :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd (P \parallel Q) \parallel R \sim_s P \parallel (Q \parallel R)$
apply(*rule closeSubstI*)
by(*fastforce intro*: *closeSubstI bisimParAssoc*)

lemma *bisimSubstResNil*:
fixes Ψ :: *'b*
and x :: *name*

shows $\Psi \rhd (\!|\nu x|\!)\mathbf{0} \sim_s \mathbf{0}$
proof(*rule closeSubstI*)
fix σ:: (*name list* × *'a list*) *list*

assume *wellFormedSubst* σ
obtain *y*::*name* **where** $y \sharp \Psi$ **and** $y \sharp \sigma$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
have $\Psi \rhd (\!|\nu y|\!)\mathbf{0} \sim \mathbf{0}$ **by**(*rule bisimResNil*)
with ‹$y \sharp \sigma$› ‹*wellFormedSubst* σ› **show** $\Psi \rhd ((\!|\nu x|\!)\mathbf{0})[<\sigma>] \sim \mathbf{0}[<\sigma>]$
by(*subst alphaRes*[*of y*]) *auto*
qed

lemma *seqSubst2*:
fixes *x* :: *name*
and *P* :: (*'a*, *'b*, *'c*) *psi*

assumes *wellFormedSubst* σ
and $x \sharp \sigma$
and $x \sharp P$

shows $x \sharp P[<\sigma>]$
using *assms*
by(*induct* σ *arbitrary*: *P*, *auto*) (*blast dest*: *subst2*)

notation *substTerm.seqSubst* (‹-[<->]› [*100*, *100*] *100*)

lemma *bisimSubstScopeExt*:
fixes Ψ :: *'b*
and *x* :: *name*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Q* :: (*'a*, *'b*, *'c*) *psi*

assumes $x \sharp P$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \sim_s P \parallel (\!|\nu x|\!)Q$
proof(*rule closeSubstI*)
fix σ:: (*name list* × *'a list*) *list*

assume *wellFormedSubst* σ
obtain *y*::*name* **where** $y \sharp \Psi$ **and** $y \sharp \sigma$ **and** $y \sharp P$ **and** $y \sharp Q$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
moreover from ‹*wellFormedSubst* σ› ‹$y \sharp \sigma$› ‹$y \sharp P$› **have** $y \sharp P[<\sigma>]$
by(*rule seqSubst2*)
hence $\Psi \rhd (\!|\nu y|\!)((P[<\sigma>]) \parallel (([(x, y)] \cdot Q)[<\sigma>])) \sim (P[<\sigma>]) \parallel (\!|\nu y|\!)(([(x, y)] \cdot Q)[<\sigma>])$
by(*rule bisimScopeExt*)
with ‹$x \sharp P$› ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \sharp \sigma$› **show** $\Psi \rhd ((\!|\nu x|\!)(P \parallel Q))[<\sigma>] \sim (P \parallel (\!|\nu x|\!)Q)[<\sigma>]$
apply(*subst alphaRes*[*of y*], *simp*)

apply(*subst alphaRes*[*of y Q*], *simp*)
by(*simp add*: *eqvts*)
qed

lemma *bisimSubstCasePushRes*:
fixes x :: *name*
and Ψ :: $'b$
and *Cs* :: $('c \times ('a, 'b, 'c)\ psi)\ list$

assumes $x \sharp$ *map fst Cs*

shows $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \sim_s Cases\ map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs$
proof(*rule closeSubstI*)
fix σ:: $(name\ list \times 'a\ list)\ list$

assume *wellFormedSubst* σ
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \sigma$ **and** $y \sharp Cs$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

{
fix x :: *name*
and *Cs* :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and σ :: $(name\ list \times 'a\ list)\ list$

assume $x \sharp \sigma$

hence $(Cases\ map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs)[<\sigma>] = Cases\ map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ (caseListSeqSubst\ Cs\ \sigma)$
by(*induct Cs*) *auto*
}
note *C1* = *this*

{
fix x :: *name*
and y :: *name*
and *Cs* :: $('c \times ('a, 'b, 'c)\ psi)\ list$

assume $x \sharp$ *map fst Cs*
and $y \sharp$ *map fst Cs*
and $y \sharp Cs$

hence $(Cases\ map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu x|\!)P))\ Cs) = Cases\ map\ (\lambda(\varphi, P).\ (\varphi, (\!|\nu y|\!)P))\ ([(x, y)] \cdot Cs)$
by(*induct Cs*) (*auto simp add*: *fresh-list-cons alphaRes*)
}
note *C2* = *this*

from ‹$y \sharp Cs$› **have** $y \sharp$ *map fst Cs* **by**(*induct Cs*) (*auto simp add*: *fresh-list-cons fresh-list-nil*)

from ‹$y \sharp Cs$› ‹$y \sharp \sigma$› ‹$x \sharp map\ fst\ Cs$› ‹$wellFormedSubst\ \sigma$› **have** $y \sharp map\ fst\ (caseListSeqSubst\ ([(x,\ y)] \cdot Cs)\ \sigma)$
by(*induct Cs*) (*auto intro*: *substCond.seqSubst2 simp add*: *fresh-list-cons fresh-list-nil fresh-prod*)
hence $\Psi \rhd (\!|\nu y|\!)(Cases(caseListSeqSubst\ ([(x,\ y)] \cdot Cs)\ \sigma)) \sim Cases\ map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu y|\!)P))\ (caseListSeqSubst\ ([(x,\ y)] \cdot Cs)\ \sigma)$
by(*rule bisimCasePushRes*)

with ‹$y \sharp Cs$› ‹$x \sharp map\ fst\ Cs$› ‹$y \sharp map\ fst\ Cs$› ‹$y \sharp \sigma$› ‹$wellFormedSubst\ \sigma$›
show $\Psi \rhd ((\!|\nu x|\!)(Cases\ Cs))[<\sigma>] \sim (Cases\ map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs)[<\sigma>]$
apply(*subst C2*[*of x Cs y*])
apply *assumption*+
apply(*subst C1*)
apply *assumption*+
apply(*subst alphaRes*[*of y*], *simp*)
by(*simp add*: *eqvts*)
qed

lemma *bisimSubstOutputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)\ psi$

assumes $x \sharp M$
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M\langle N\rangle.P) \sim_s M\langle N\rangle.(\!|\nu x|\!)P$
proof(*rule closeSubstI*)
fix σ:: $(name\ list \times 'a\ list)\ list$

assume $wellFormedSubst\ \sigma$
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \sigma$ **and** $y \sharp P$ **and** $y \sharp M$ **and** $y \sharp N$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
from ‹$wellFormedSubst\ \sigma$› ‹$y \sharp M$› ‹$y \sharp \sigma$› **have** $y \sharp M[<\sigma>]$ **by** *auto*
moreover from ‹$wellFormedSubst\ \sigma$› ‹$y \sharp N$› ‹$y \sharp \sigma$› **have** $y \sharp N[<\sigma>]$ **by** *auto*
ultimately have $\Psi \rhd (\!|\nu y|\!)((M[<\sigma>])\langle(N[<\sigma>])\rangle.(([(x,\ y)] \cdot P)[<\sigma>])) \sim (M[<\sigma>])\langle(N[<\sigma>])\rangle.((\!|\nu y|\!)(([(x,\ y)] \cdot P)[<\sigma>]))$
by(*rule bisimOutputPushRes*)
with ‹$y \sharp M$› ‹$y \sharp N$› ‹$y \sharp P$› ‹$x \sharp M$› ‹$x \sharp N$› ‹$y \sharp \sigma$› ‹$wellFormedSubst\ \sigma$›
show $\Psi \rhd ((\!|\nu x|\!)(M\langle N\rangle.P))[<\sigma>] \sim (M\langle N\rangle.(\!|\nu x|\!)P)[<\sigma>]$
apply(*subst alphaRes*[*of y*], *simp*)
apply(*subst alphaRes*[*of y P*], *simp*)
by(*simp add*: *eqvts*)
qed

lemma *bisimSubstInputPushRes*:

fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$

assumes $x \sharp M$
and $x \sharp$ *xvec*
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \sim_s M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
proof(*rule closeSubstI*)
fix σ:: (*name list* $\times$ $'a$ *list*) *list*

assume *wellFormedSubst* σ
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp \sigma$ **and** $y \sharp P$ **and** $y \sharp M$ **and** $y \sharp$ *xvec* **and** $y \sharp N$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
obtain p::*name prm* **where** $(p \cdot xvec) \sharp{*}\ N$ **and** $(p \cdot xvec) \sharp{*}\ P$ **and** $x \sharp (p \cdot xvec)$ **and** $y \sharp (p \cdot xvec)$ **and** $(p \cdot xvec) \sharp{*}\ \sigma$
and S: *set* $p \subseteq$ *set xvec* $\times$ *set*$(p \cdot xvec)$
by(*rule-tac* c=(N, P, x, y, σ) **in** *name-list-avoiding*) *auto*

from ‹*wellFormedSubst* σ› ‹$y \sharp M$› ‹$y \sharp \sigma$ › **have** $y \sharp M[<\sigma>]$ **by** *auto*
moreover note ‹$y \sharp (p \cdot xvec)$›
moreover from ‹$y \sharp N$› **have** $(p \cdot y) \sharp (p \cdot N)$ **by**(*simp add*: *pt-fresh-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹$y \sharp$ *xvec*› ‹$y \sharp (p \cdot xvec)$› S **have** $y \sharp p \cdot N$ **by** *simp*
with ‹*wellFormedSubst* σ› **have** $y \sharp (p \cdot N)[<\sigma>]$ **using** ‹$y \sharp \sigma$› **by** *auto*
ultimately have $\Psi \rhd (\!|\nu y|\!)((M[<\sigma>])(\!|\lambda{*}(p \cdot xvec)\ ((p \cdot N)[<\sigma>])|\!).(([(x, y)] \cdot (p \cdot P))[<\sigma>])) \sim (M[<\sigma>])(\!|\lambda{*}(p \cdot xvec)\ ((p \cdot N)[<\sigma>])|\!).((\!|\nu y|\!)(([(x, y)] \cdot p \cdot P)[<\sigma>]))$
by(*rule bisimInputPushRes*)
with ‹$y \sharp M$› ‹$y \sharp N$› ‹$y \sharp P$› ‹$x \sharp M$› ‹$x \sharp N$› ‹$y \sharp$ *xvec*› ‹$x \sharp$ *xvec*› ‹$(p \cdot xvec) \sharp{*}\ N$› ‹$(p \cdot xvec) \sharp{*}\ P$›
‹$x \sharp (p \cdot xvec)$› ‹$y \sharp (p \cdot xvec)$› ‹$y \sharp \sigma$› ‹$(p \cdot xvec) \sharp{*}\ \sigma$› S ‹*wellFormedSubst* σ›
show $\Psi \rhd ((\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P))[<\sigma>] \sim (M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P)[<\sigma>]$
apply(*subst inputChainAlpha'*)
apply *assumption*+
apply(*subst inputChainAlpha'*[*of p xvec*])
apply(*simp add*: *abs-fresh-star*)
apply *assumption*+
apply(*simp add*: *eqvts*)
apply(*subst alphaRes*[*of y*], *simp*)
apply(*simp add*: *inputChainFresh*)
apply(*simp add*: *freshChainSimps*)
apply(*subst alphaRes*[*of y* $(p \cdot P)$])
apply(*simp add*: *freshChainSimps*)

by(*simp add*: *freshChainSimps eqvts*)
qed

lemma *bisimSubstResComm*:
fixes x :: *name*
and y :: *name*

shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \sim_s (\!|\nu y|\!)((\!|\nu x|\!)P)$
proof(*case-tac* $x = y$)
assume $x = y$
thus *?thesis* **by**(*force intro*: *bisimSubstReflexive*)
next
assume $x \neq y$
show *?thesis*
proof(*rule closeSubstI*)
fix σ:: (*name list* $\times$ $'a$ *list*) *list*
assume *wellFormedSubst* σ

obtain x'::*name* **where** $x' \sharp \Psi$ **and** $x' \sharp \sigma$ **and** $x' \sharp P$ **and** $x \neq x'$ **and** $y \neq x'$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)
obtain y'::*name* **where** $y' \sharp \Psi$ **and** $y' \sharp \sigma$ **and** $y' \sharp P$ **and** $x \neq y'$ **and** $y \neq y'$ **and** $x' \neq y'$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

have $\Psi \rhd (\!|\nu x'|\!)((\!|\nu y'|\!)(([(x, x')] \cdot [(y, y')] \cdot P)[<\sigma>])) \sim (\!|\nu y'|\!)((\!|\nu x'|\!)(([(x, x')] \cdot [(y, y')] \cdot P)[<\sigma>]))$
by(*rule bisimResComm*)
moreover from ‹$x' \sharp P$› ‹$y' \sharp P$› ‹$x \neq y'$› ‹$x' \neq y'$› **have** $(\!|\nu x|\!)((\!|\nu y|\!)P) = (\!|\nu x'|\!)((\!|\nu y'|\!)(([(x, x')] \cdot [(y, y')] \cdot P)))$
apply(*subst alphaRes*[*of* y' P], *simp*)
by(*subst alphaRes*[*of* x']) (*auto simp add*: *abs-fresh fresh-left calc-atm eqvts*)
moreover from ‹$x' \sharp P$› ‹$y' \sharp P$› ‹$y \neq x'$› ‹$x \neq y'$› ‹$x' \neq y'$› ‹$x \neq x'$› ‹$x \neq y$› **have** $(\!|\nu y|\!)((\!|\nu x|\!)P) = (\!|\nu y'|\!)((\!|\nu x'|\!)(([(x, x')] \cdot [(y, y')] \cdot P)))$
apply(*subst alphaRes*[*of* x' P], *simp*)
apply(*subst alphaRes*[*of* y'], *simp add*: *abs-fresh fresh-left calc-atm*)
apply(*simp add*: *eqvts calc-atm*)
by(*subst perm-compose*) (*simp add*: *calc-atm*)

ultimately show $\Psi \rhd ((\!|\nu x|\!)((\!|\nu y|\!)P))[<\sigma>] \sim ((\!|\nu y|\!)((\!|\nu x|\!)P))[<\sigma>]$
using ‹*wellFormedSubst* σ› ‹$x' \sharp \sigma$› ‹$y' \sharp \sigma$›
by *simp*
qed
qed

lemma *bisimSubstExtBang*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*

assumes *guarded P*

shows $\Psi \triangleright !P \sim_s P \parallel !P$
using *assms*
by(*fastforce intro*: *closeSubstI bangExt guardedSeqSubst*)

lemma *structCongBisimSubst*:
fixes P :: ($'a$, $'b$, $'c$) *psi*
and Q :: ($'a$, $'b$, $'c$) *psi*

assumes $P \equiv_s Q$

shows $P \sim_s Q$
using *assms*
by(*induct rule*: *structCong.induct*)
(*auto intro*: *bisimSubstReflexive bisimSubstSymmetric bisimSubstTransitive bisimSubstParComm bisimSubstParAssoc bisimSubstParNil bisimSubstResNil bisimSubstResComm bisimSubstScopeExt bisimSubstCasePushRes bisimSubstInputPushRes bisimSubstOutputPushRes bisimSubstExtBang*)

end

end

theory *Weak-Bisim-Subst*
imports *Weak-Bisim-Struct-Cong Weak-Bisim-Pres Bisim-Subst*
begin

context *env* **begin**

abbreviation
weakBisimSubstJudge (‹- ▷ - $\approx_s$ -› [*70*, *70*, *70*] *65*) **where** $\Psi \triangleright P \approx_s Q \equiv (\Psi, P, Q) \in$ *closeSubst weakBisim*
abbreviation
weakBisimSubstNilJudge (‹- $\approx_s$ -› [*70*, *70*] *65*) **where** $P \approx_s Q \equiv \mathbf{1} \triangleright P \approx_s Q$

lemmas *weakBisimSubstClosed*[*eqvt*] = *closeSubstClosed*[*OF weakBisimEqvt*]
lemmas *weakBisimEqvt*[*simp*] = *closeSubstEqvt*[*OF weakBisimEqvt*]

lemma *strongBisimSubstWeakBisimSubst*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and Q :: ($'a$, $'b$, $'c$) *psi*

assumes $\Psi \triangleright P \sim_s Q$

shows $\Psi \triangleright P \approx_s Q$
using *assms*
by(*metis closeSubstI closeSubstE strongBisimWeakBisim*)

lemma *weakBisimSubstOutputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and N :: $'a$

  assumes $\Psi \rhd P \approx_s Q$

  shows $\Psi \rhd M\langle N\rangle.P \approx_s M\langle N\rangle.Q$
using *assms*
by(*fastforce intro*: *closeSubstI closeSubstE weakBisimOutputPres*)

lemma *bisimSubstInputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and *xvec* :: *name list*
  and N :: $'a$

  assumes $\Psi \rhd P \approx_s Q$
  and *xvec* $\sharp*$ Ψ
  and *distinct xvec*

  shows $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \approx_s M(\!|\lambda{*}xvec\ N|\!).Q$
proof(*rule-tac closeSubstI*)
  fix σ :: (*name list* $\times$ $'a$ *list*) *list*
  assume *wellFormedSubst* σ
  obtain p **where** $(p \cdot xvec) \sharp* \sigma$
    and $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* N$
    and S: *set* $p \subseteq$ *set xvec* $\times$ *set* $(p \cdot xvec)$
    by(*rule-tac* c=(σ, P, Q, Ψ, N) **in** *name-list-avoiding*) *auto*

  from ‹$\Psi \rhd P \approx_s Q$› **have** $(p \cdot \Psi) \rhd (p \cdot P) \approx_s (p \cdot Q)$
    by(*rule weakBisimSubstClosed*)
  with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec) \sharp* \Psi$› S **have** $\Psi \rhd (p \cdot P) \approx_s (p \cdot Q)$
    by *simp*

  {
    fix $Tvec'$ :: $'a$ *list*
    assume *length* $(p \cdot xvec)$ = *length* $Tvec'$
    with ‹*wellFormedSubst* σ› ‹*distinct xvec*› **have** *wellFormedSubst* (σ @ [$(p \cdot xvec, Tvec')$])
      by *simp*
    with ‹$\Psi \rhd (p \cdot P) \approx_s (p \cdot Q)$› **have** $\Psi \rhd (p \cdot P)[<(\sigma$ @ $[(p \cdot xvec, Tvec')])>] \approx (p \cdot Q)[<(\sigma$ @ $[(p \cdot xvec, Tvec')])>]$

by (*rule closeSubstE*)
then have $\Psi \triangleright ((p \cdot P)[<\sigma>])[(p \cdot xvec)::=Tvec'] \approx ((p \cdot Q)[<\sigma>])[(p \cdot xvec)::=Tvec']$
by (*metis seqSubsCons seqSubsNil seqSubsTermAppend*)
}

then have $\Psi \triangleright M[<\sigma>](\!|\lambda*(p \cdot xvec)\ (p \cdot N)[<\sigma>]|\!).((p \cdot P)[<\sigma>]) \approx M[<\sigma>](\!|\lambda*(p \cdot xvec)\ (p \cdot N)[<\sigma>]|\!).((p \cdot Q)[<\sigma>])$
using *weakBisimInputPres* **by** *metis*
with ‹$(p \cdot xvec)\ \sharp*\ \sigma$› **have** $\Psi \triangleright (M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P))[<\sigma>] \approx (M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot Q))[<\sigma>]$
by (*metis seqSubstInputChain seqSubstSimps*(*3*))
moreover from ‹$(p \cdot xvec)\ \sharp*\ N$› ‹$(p \cdot xvec)\ \sharp*\ P$› S **have** $M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P) = M(\!|\lambda*xvec\ N|\!).P$
apply (*simp add*: *psi.inject*) **by** (*rule inputChainAlpha*[*symmetric*]) *auto*
moreover from ‹$(p \cdot xvec)\ \sharp*\ N$› ‹$(p \cdot xvec)\ \sharp*\ Q$› S **have** $M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot Q) = M(\!|\lambda*xvec\ N|\!).Q$
apply (*simp add*: *psi.inject*) **by** (*rule inputChainAlpha*[*symmetric*]) *auto*
ultimately show $\Psi \triangleright (M(\!|\lambda*xvec\ N|\!).P)[<\sigma>] \approx (M(\!|\lambda*xvec\ N|\!).Q)[<\sigma>]$
by *force*
qed

lemma *weakBisimSubstReflexive*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

shows $\Psi \triangleright P \approx_s P$
by(*auto intro*: *closeSubstI weakBisimReflexive*)

lemma *bisimSubstTransitive*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $R :: ('a, 'b, 'c)\ psi$

assumes $\Psi \triangleright P \approx_s Q$
and $\Psi \triangleright Q \approx_s R$

shows $\Psi \triangleright P \approx_s R$
using *assms*
by(*auto intro*: *closeSubstI closeSubstE weakBisimTransitive*)

lemma *weakBisimSubstSymmetric*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$

assumes $\Psi \triangleright P \approx_s Q$

shows $\Psi \rhd Q \approx_s P$
using *assms*
by(*auto intro*: *closeSubstI closeSubstE weakBisimE*)

lemma *weakBisimSubstParPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \approx_s Q$

shows $\Psi \rhd P \parallel R \approx_s Q \parallel R$
using *assms*
by(*fastforce intro*: *closeSubstI closeSubstE weakBisimParPres*)

lemma *weakBisimSubstResPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and x :: *name*

assumes $\Psi \rhd P \approx_s Q$
and $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)P \approx_s (\!|\nu x|\!)Q$
proof(*rule closeSubstI*)
fix σ :: (*name list* $\times$ $'a$ *list*) *list*
assume *wellFormedSubst* σ
obtain y::*name* **where** $y \sharp \Psi$ **and** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \sigma$
by (*generate-fresh name*) *auto*

from ‹$\Psi \rhd P \approx_s Q$› **have** $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \approx_s ([(x, y)] \cdot Q)$
by (*rule weakBisimSubstClosed*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \rhd ([(x, y)] \cdot P) \approx_s ([(x, y)] \cdot Q)$
by *simp*
hence $\Psi \rhd ([(x, y)] \cdot P)[<\sigma>] \approx ([(x, y)] \cdot Q)[<\sigma>]$
using ‹*wellFormedSubst* σ› **by** (*rule closeSubstE*)
hence $\Psi \rhd (\!|\nu y|\!)(([(x, y)] \cdot P)[<\sigma>]) \approx (\!|\nu y|\!)(([(x, y)] \cdot Q)[<\sigma>])$
using ‹$y \sharp \Psi$› **by** (*rule weakBisimResPres*)
with ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \sharp \sigma$› **show** $\Psi \rhd ((\!|\nu x|\!)P)[<\sigma>] \approx ((\!|\nu x|\!)Q)[<\sigma>]$
by (*simp add*: *alphaRes*)
qed

lemma *weakBisimSubstParNil*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

shows $\Psi \rhd P \parallel \mathbf{0} \approx_s P$
by(*metis strongBisimSubstWeakBisimSubst bisimSubstParNil*)

lemma *weakBisimSubstParComm*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd P \parallel Q \approx_s Q \parallel P$
by(*metis strongBisimSubstWeakBisimSubst bisimSubstParComm*)

lemma *weakBisimSubstParAssoc*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd (P \parallel Q) \parallel R \approx_s P \parallel (Q \parallel R)$
by(*metis strongBisimSubstWeakBisimSubst bisimSubstParAssoc*)

lemma *weakBisimSubstResNil*:
fixes Ψ :: $'b$
and x :: *name*

shows $\Psi \rhd (\!|\nu x|\!)\mathbf{0} \sim_s \mathbf{0}$
by(*metis strongBisimSubstWeakBisimSubst bisimSubstResNil*)

lemma *weakBisimSubstScopeExt*:
fixes Ψ :: $'b$
and x :: *name*
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp P$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \approx_s P \parallel (\!|\nu x|\!)Q$
using *assms*
by(*metis strongBisimSubstWeakBisimSubst bisimSubstScopeExt*)

lemma *weakBisimSubstCasePushRes*:
fixes x :: *name*
and Ψ :: $'b$
and Cs :: $('c \times ('a,\ 'b,\ 'c)\ psi)\ list$

assumes $x \sharp map\ fst\ Cs$

shows $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \approx_s Cases\ map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs$
using *assms*

by(*metis strongBisimSubstWeakBisimSubst bisimSubstCasePushRes*)

lemma *weakBisimSubstOutputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp M$
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M\langle N\rangle.P) \approx_s M\langle N\rangle.(\!|\nu x|\!)P$
using *assms*
by(*metis strongBisimSubstWeakBisimSubst bisimSubstOutputPushRes*)

lemma *weakBisimSubstInputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$

assumes $x \sharp M$
and $x \sharp$ *xvec*
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \approx_s M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
using *assms*
by(*metis strongBisimSubstWeakBisimSubst bisimSubstInputPushRes*)

lemma *weakBisimSubstResComm*:
fixes x :: *name*
and y :: *name*

shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \approx_s (\!|\nu y|\!)((\!|\nu x|\!)P)$
by(*metis strongBisimSubstWeakBisimSubst bisimSubstResComm*)

lemma *weakBisimSubstExtBang*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes *guarded* P

shows $\Psi \rhd !P \approx_s P \parallel !P$
using *assms*
by(*metis strongBisimSubstWeakBisimSubst bisimSubstExtBang*)

end

end

theory *Weakening*
  imports *Weak-Bisimulation*
begin

locale *weak* = *env* +
  assumes *weaken*: $\Psi \hookrightarrow \Psi \otimes \Psi'$
begin

lemma *entWeaken*:
  fixes Ψ :: $'b$
  and φ :: $'c$

  assumes $\Psi \vdash \varphi$

  shows $\Psi \otimes \Psi' \vdash \varphi$
using *assms weaken*
by(*auto simp add*: *AssertionStatImp-def*)

lemma *assertWeaken*:
  fixes Ψ :: $'b$
  and Ψ' :: $'b$

  shows $\Psi \hookrightarrow \Psi \otimes \Psi'$
by(*auto simp add*: *AssertionStatImp-def entWeaken*)

lemma *frameWeaken*:
  fixes F :: $'b$ *frame*
  and G :: $'b$ *frame*

  shows $F \hookrightarrow_F F \otimes_F G$
proof −
  obtain A_F Ψ_F **where** *FrF*: $F = \langle A_F, \Psi_F\rangle$ **and** $A_F \sharp* F$ **and** $A_F \sharp* G$
    by(*rule-tac F=F* **and** *C=(F, G)* **in** *freshFrame*) *auto*
  obtain A_G Ψ_G **where** *FrG*: $G = \langle A_G, \Psi_G\rangle$ **and** $A_G \sharp* F$ **and** $A_G \sharp* G$ **and** $A_G \sharp* A_F$ **and** $A_G \sharp* \Psi_F$
    by(*rule-tac F=G* **and** *C=(F, G, A_F, Ψ_F)* **in** *freshFrame*) *auto*
  from *FrG* ‹$A_F \sharp* G$› ‹$A_G \sharp* A_F$› **have** $A_F \sharp* \Psi_G$ **by** *auto*
  have $\Psi_F \hookrightarrow \Psi_F \otimes \Psi_G$ **by**(*rule weaken*)
  hence $\langle A_G, \Psi_F\rangle \hookrightarrow_F \langle A_G, \Psi_F \otimes \Psi_G\rangle$ **by**(*rule-tac frameImpResChainPres*) *auto*
  with ‹$A_G \sharp* \Psi_F$› **have** $\langle \varepsilon, \Psi_F\rangle \hookrightarrow_F \langle A_G, \Psi_F \otimes \Psi_G\rangle$ **using** *frameResFreshChain*
    by(*rule-tac FrameStatImpTrans*) (*auto simp add*: *FrameStatEq-def*)
  with *FrF FrG* ‹$A_G \sharp* A_F$› ‹$A_G \sharp* \Psi_F$› ‹$A_F \sharp* \Psi_G$› **show** *?thesis*
    by(*force simp add*: *frameChainAppend intro*: *frameImpResChainPres*)
qed

lemma *unitAssertWeaken*:
fixes Ψ :: *'b*

shows $\mathbf{1} \hookrightarrow \Psi$
proof –
have $\mathbf{1} \hookrightarrow \mathbf{1} \otimes \Psi$ **by**(*rule assertWeaken*)
moreover have $\mathbf{1} \otimes \Psi \hookrightarrow \Psi$ **by**(*metis Identity AssertionStatEq-def Commutativity AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule AssertionStatImpTrans*)
qed

lemma *unitFrameWeaken*:
fixes F :: *'b frame*

shows $\langle \varepsilon, \mathbf{1} \rangle \hookrightarrow_F F$
proof –
have $\langle \varepsilon, \mathbf{1} \rangle \hookrightarrow_F ((\langle \varepsilon, \mathbf{1} \rangle) \otimes_F F)$ **by**(*rule frameWeaken*)
moreover obtain A_F Ψ_F **where** *FrF*: $F = \langle A_F, \Psi_F \rangle$
by(*rule-tac F=F* **and** *C=()* **in** *freshFrame*) *auto*
hence $(\langle \varepsilon, \mathbf{1} \rangle) \otimes_F F \simeq_F F$
by *simp* (*metis frameIntIdentity frameIntCommutativity FrameStatEqTrans FrameStatEqSym*)
ultimately show *?thesis* **by**(*metis FrameStatImpTrans FrameStatEq-def*)
qed

lemma *insertAssertionWeaken*:
fixes F :: *'b frame*
and Ψ :: *'b*

shows $\langle \varepsilon, \Psi \rangle \hookrightarrow_F$ *insertAssertion* F Ψ
proof –
have $\langle \varepsilon, \Psi \rangle \hookrightarrow_F (\langle \varepsilon, \Psi \rangle) \otimes_F F$ **by**(*rule frameWeaken*)
thus *?thesis* **by** *simp*
qed

lemma *frameImpStatEq*:
fixes A_F :: *name list*
and Ψ :: *'b*
and Ψ' :: *'b*
and φ :: *'c*

assumes $(\langle A_F, \Psi \rangle) \vdash_F \varphi$
and $\Psi \simeq \Psi'$

shows $(\langle A_F, \Psi' \rangle) \vdash_F \varphi$
proof –
obtain *p*::*name prm* **where** $(p \cdot A_F) \sharp* \varphi$ **and** $(p \cdot A_F) \sharp* \Psi$ **and** $(p \cdot A_F) \sharp* \Psi'$
and *distinctPerm p* **and** *S*: *set p* $\subseteq$ *set* $A_F \times$ *set*$(p \cdot A_F)$

by(*rule-tac c*=(φ, Ψ, Ψ') **in** *name-list-avoiding*) *auto*
from ‹($\langle A_F, \Psi\rangle$) $\vdash_F \varphi$› ‹($p \cdot A_F$) $\sharp*$ Ψ› S **have** ($\langle(p \cdot A_F), p \cdot \Psi\rangle$) $\vdash_F \varphi$ **by**(*simp add*: *frameChainAlpha*)
hence ($p \cdot \Psi$) $\vdash \varphi$ **using** ‹($p \cdot A_F$) $\sharp*$ φ› **by**(*rule frameImpE*)
moreover from ‹$\Psi \simeq \Psi'$› **have** ($p \cdot \Psi$) $\simeq$ ($p \cdot \Psi'$) **by**(*rule AssertionStatEq-Closed*)
ultimately have ($p \cdot \Psi'$) $\vdash \varphi$ **by**(*simp add*: *AssertionStatEq-def Assertion-StatImp-def*)
hence ($\langle(p \cdot A_F), p \cdot \Psi'\rangle$) $\vdash_F \varphi$ **using** ‹($p \cdot A_F$) $\sharp*$ φ›
by(*rule-tac frameImpI*) *auto*
with ‹($p \cdot A_F$) $\sharp*$ Ψ'› S **show** *?thesis* **by**(*simp add*: *frameChainAlpha*)
qed

lemma *statImpTauDerivative*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and P' :: ($'a$, $'b$, $'c$) *psi*

assumes $\Psi \rhd P \longmapsto\tau \prec P'$

shows *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P') Ψ
proof(*auto simp add*: *FrameStatImp-def*)
fix φ :: $'c$
obtain A_P Ψ_P **where** *FrP*: *extractFrame* $P = \langle A_P, \Psi_P\rangle$ **and** $A_P \sharp* P$ **and** $A_P \sharp* \varphi$ **and** $A_P \sharp* \Psi$ **and** *distinct* A_P
by(*rule-tac C*=(P, φ, Ψ) **in** *freshFrame*) *auto*
with ‹$\Psi \rhd P \longmapsto\tau \prec P'$› **obtain** Ψ' $A_P{}'$ $\Psi_P{}'$ **where** *FrP'*: *extractFrame* $P' = \langle A_P{}', \Psi_P{}'\rangle$ **and** $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$
and $A_P{}' \sharp* P'$ **and** $A_P{}' \sharp* \varphi$ **and** $A_P{}' \sharp* \Psi$
by(*rule-tac C*=(Ψ, φ) **in** *expandTauFrame*) *auto*
assume *insertAssertion* (*extractFrame* P) $\Psi \vdash_F \varphi$
with *FrP* ‹$A_P \sharp* \varphi$› ‹$A_P \sharp* \Psi$› **have** $\Psi \otimes \Psi_P \vdash \varphi$ **by**(*auto dest*: *frameImpE*)
hence ($\Psi \otimes \Psi_P$) $\otimes \Psi' \vdash \varphi$ **by**(*rule entWeaken*)
hence $\Psi \otimes \Psi_P{}' \vdash \varphi$ **using** ‹$\Psi_P \otimes \Psi' \simeq \Psi_P{}'$›
by(*rule-tac statEqEnt*, *auto*) (*metis Associativity compositionSym Assertion-StatEqTrans AssertionStatEqSym Commutativity*)
with ‹$A_P{}' \sharp* \varphi$› ‹$A_P{}' \sharp* \Psi$› *FrP'* **show** *insertAssertion* (*extractFrame* P') $\Psi \vdash_F \varphi$
by(*force intro*: *frameImpI*)
qed

lemma *weakenTransition*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and Rs :: ($'a$, $'b$, $'c$) *residual*
and Ψ' :: $'b$

assumes $\Psi \rhd P \longmapsto Rs$

shows $\Psi \otimes \Psi' \rhd P \longmapsto Rs$
using *assms*
proof(*nominal-induct avoiding*: Ψ' *rule*: *semantics.strong-induct*)
case(*cInput* Ψ M K *xvec* N *Tvec* P Ψ')
from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi' \vdash M \leftrightarrow K$ **by**(*rule entWeaken*)
thus *?case* **using** ‹*distinct xvec*› ‹*set xvec* $\subseteq$ (*supp* N)› ‹*length xvec* = *length Tvec*›
by(*rule Input*)
next
case(*Output* Ψ M K N P Ψ')
from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi' \vdash M \leftrightarrow K$ **by**(*rule entWeaken*)
thus *?case* **by**(*rule semantics.Output*)
next
case(*Case* Ψ P *Rs* φ *Cs* Ψ')
have $\Psi \otimes \Psi' \rhd P \longmapsto Rs$ **by**(*rule Case*)
moreover note ‹(φ, P) *mem Cs*›
moreover from ‹$\Psi \vdash \varphi$› **have** $\Psi \otimes \Psi' \vdash \varphi$ **by**(*rule entWeaken*)
ultimately show *?case* **using** ‹*guarded P*›
by(*rule semantics.Case*)
next
case(*cPar1* Ψ Ψ_Q P α P' Q A_Q Ψ')
have $(\Psi \otimes \Psi_Q) \otimes \Psi' \rhd P \longmapsto \alpha \prec P'$ **by**(*rule cPar1*)
hence $(\Psi \otimes \Psi') \otimes \Psi_Q \rhd P \longmapsto \alpha \prec P'$
by(*metis statEqTransition Composition Associativity Commutativity Assertion-StatEqTrans*)
thus *?case* **using** ‹*extractFrame* $Q = \langle A_Q, \Psi_Q \rangle$› ‹*bn* $\alpha \sharp* Q$› ‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi'$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* \alpha$›
by(*rule-tac Par1*) *auto*
next
case(*cPar2* Ψ Ψ_P Q α Q' P A_P Ψ')
have $(\Psi \otimes \Psi_P) \otimes \Psi' \rhd Q \longmapsto \alpha \prec Q'$ **by**(*rule cPar2*)
hence $(\Psi \otimes \Psi') \otimes \Psi_P \rhd Q \longmapsto \alpha \prec Q'$
by(*metis statEqTransition Composition Associativity Commutativity Assertion-StatEqTrans*)
thus *?case* **using** ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$› ‹*bn* $\alpha \sharp* P$› ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi'$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* \alpha$›
by(*rule-tac Par2*) *auto*
next
case(*cComm1* Ψ Ψ_Q P M N P' A_P Ψ_P Q K *xvec* Q' A_Q Ψ')
have $(\Psi \otimes \Psi_Q) \otimes \Psi' \rhd P \longmapsto M(\!|N|\!) \prec P'$ **by**(*rule cComm1*)
hence $(\Psi \otimes \Psi') \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P'$
by(*metis statEqTransition Composition Associativity Commutativity Assertion-StatEqTrans*)
moreover note ‹*extractFrame* $P = \langle A_P, \Psi_P \rangle$›
moreover have $(\Psi \otimes \Psi_P) \otimes \Psi' \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N \rangle \prec Q'$ **by**(*rule cComm1*)
hence $(\Psi \otimes \Psi') \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N \rangle \prec Q'$
by(*metis statEqTransition Composition Associativity Commutativity Assertion-StatEqTrans*)

moreover note ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$›
moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_P \otimes \Psi_Q) \otimes \Psi' \vdash M \leftrightarrow K$ **by**(*rule entWeaken*)
hence $(\Psi \otimes \Psi') \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$ **by**(*metis statEqEnt Composition Associativity Commutativity AssertionStatEqTrans*)
ultimately show *?case* **using** ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi'$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* M$› ‹$A_P \sharp* A_Q$›
‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi'$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹*xvec* $\sharp* P$›
by(*rule-tac Comm1*) (*assumption* | *auto*)+
next
case(*cComm2* Ψ Ψ_Q P M *xvec* N P' A_P Ψ_P Q K Q' A_Q Ψ')
have $(\Psi \otimes \Psi_Q) \otimes \Psi' \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **by**(*rule cComm2*)
hence $(\Psi \otimes \Psi') \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
by(*metis statEqTransition Composition Associativity Commutativity Assertion-StatEqTrans*)
moreover note ‹*extractFrame* $P = \langle A_P, \Psi_P\rangle$›
moreover have $(\Psi \otimes \Psi_P) \otimes \Psi' \rhd Q \longmapsto K(\!|N|\!) \prec Q'$ **by**(*rule cComm2*)
hence $(\Psi \otimes \Psi') \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$
by(*metis statEqTransition Composition Associativity Commutativity Assertion-StatEqTrans*)
moreover note ‹*extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$›
moreover from ‹$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $(\Psi \otimes \Psi_P \otimes \Psi_Q) \otimes \Psi' \vdash M \leftrightarrow K$ **by**(*rule entWeaken*)
hence $(\Psi \otimes \Psi') \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$ **by**(*metis statEqEnt Composition Associativity Commutativity AssertionStatEqTrans*)
ultimately show *?case* **using** ‹$A_P \sharp* \Psi$› ‹$A_P \sharp* \Psi'$› ‹$A_P \sharp* P$› ‹$A_P \sharp* Q$› ‹$A_P \sharp* M$› ‹$A_P \sharp* A_Q$›
‹$A_Q \sharp* \Psi$› ‹$A_Q \sharp* \Psi'$› ‹$A_Q \sharp* P$› ‹$A_Q \sharp* Q$› ‹$A_Q \sharp* K$› ‹*xvec* $\sharp* Q$›
by(*rule-tac Comm2*) (*assumption* | *auto*)+
next
case(*cOpen* Ψ P M *xvec* *yvec* N P' x Ψ')
have $\Psi \otimes \Psi' \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle N\rangle \prec P'$ **by**(*rule cOpen*)
thus *?case* **using** ‹$x \in$ *supp* N› ‹$x \sharp \Psi$› ‹$x \sharp \Psi'$› ‹$x \sharp M$› ‹$x \sharp$ *xvec*› ‹$x \sharp$ *yvec*›
by(*rule-tac Open*) *auto*
next
case(*cScope* Ψ P α P' x Ψ')
have $\Psi \otimes \Psi' \rhd P \longmapsto \alpha \prec P'$ **by**(*rule cScope*)
thus *?case* **using** ‹$x \sharp \Psi$› ‹$x \sharp \Psi'$› ‹$x \sharp \alpha$› **by**(*rule-tac Scope*) *auto*
next
case(*Bang* Ψ P *Rs* Ψ')
have $\Psi \otimes \Psi' \rhd P \parallel !P \longmapsto$ *Rs* **by**(*rule Bang*)
thus *?case* **using** ‹*guarded* P› **by**(*rule semantics.Bang*)
qed

end

end

theory *Weaken-Transition*
  imports *Weakening*
begin

context *weak*
begin

definition *weakenTransition* :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ $'a$ *action* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- ▷ - ⟹- ≺ -› [*80*, *80*, *80*, *80*] *80*)
where
  $\Psi \triangleright P \Longrightarrow \alpha \prec P' \equiv (\exists P''' P''. \Psi \triangleright P \Longrightarrow\hat{}_\tau P''' \wedge \Psi \triangleright P''' \longmapsto \alpha \prec P'' \wedge \Psi \triangleright P'' \Longrightarrow\hat{}_\tau P') \vee (P = P' \wedge \alpha = \tau)$

lemma *weakenTransitionCases*[*consumes 1*, *case-names cBase cStep*]:
  assumes $\Psi \triangleright P \Longrightarrow \alpha \prec P'$
  and *Prop* (τ) P
  and $\bigwedge P''' P''. [\![\Psi \triangleright P \Longrightarrow\hat{}_\tau P'''; \Psi \triangleright P''' \longmapsto \alpha \prec P''; \Psi \triangleright P'' \Longrightarrow\hat{}_\tau P']\!] \Longrightarrow$ *Prop* α P'

  shows *Prop* α P'
using *assms*
by(*auto simp add*: *weakenTransition-def*)

lemma *statImpTauChainDerivative*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*

  assumes $\Psi \triangleright P \Longrightarrow\hat{}_\tau P'$

  shows *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P') Ψ
using *assms*
by(*induct rule*: *tauChainInduct*) (*auto intro*: *statImpTauDerivative dest*: *FrameStatImpTrans*)

lemma *weakenTauChain*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*
  and Ψ' :: $'b$

  assumes $\Psi \triangleright P \Longrightarrow\hat{}_\tau P'$
  shows $\Psi \otimes \Psi' \triangleright P \Longrightarrow\hat{}_\tau P'$
using *assms*
proof(*induct rule*: *tauChainInduct*)
  case(*TauBase P*)
  thus *?case* **by** *simp*

next
  case(*TauStep P P′ P″*)
  note ‹$\Psi \otimes \Psi' \rhd P \Longrightarrow\hat{}_\tau P'$›
  moreover from ‹$\Psi \rhd P' \longmapsto\tau \prec P''$› **have** $\Psi \otimes \Psi' \rhd P' \longmapsto\tau \prec P''$ **by**(*rule weakenTransition*)
  ultimately show *?case* **by**(*auto dest*: *tauActTauChain*)
qed

end

end

theory *Weaken-Stat-Imp*
  imports *Weaken-Transition*
begin

context *weak* **begin**

definition
  weakenStatImp :: $'b \Rightarrow ('a, 'b, 'c)\ psi \Rightarrow$
    $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set \Rightarrow$
    $('a, 'b, 'c)\ psi \Rightarrow bool$ (‹- ⊳ - $\lessapprox_w$<-> -› [80, 80, 80, 80] 80)
where $\Psi \rhd P \lessapprox_w <Rel> Q \equiv \exists Q'.\ \Psi \rhd Q \Longrightarrow\hat{}_\tau Q' \wedge insertAssertion(extractFrame\ P)\ \Psi \hookrightarrow_F insertAssertion(extractFrame\ Q')\ \Psi \wedge (\Psi, P, Q') \in Rel$

lemma *weakenStatImpMonotonic*:
  fixes $\Psi :: 'b$
  and $P :: ('a, 'b, 'c)\ psi$
  and $A :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
  and $Q :: ('a, 'b, 'c)\ psi$
  and $B :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

  assumes $\Psi \rhd P \lessapprox_w <A> Q$
  and $A \subseteq B$

  shows $\Psi \rhd P \lessapprox_w <B> Q$
using *assms*
by(*auto simp add*: *weakenStatImp-def*)

lemma *weakenStatImpI*:
  fixes $\Psi :: 'b$
  and $P :: ('a, 'b, 'c)\ psi$
  and $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
  and $Q :: ('a, 'b, 'c)\ psi$
  and $\Psi' :: 'b$

  assumes $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$
  and $insertAssertion(extractFrame\ P)\ \Psi \hookrightarrow_F insertAssertion(extractFrame\ Q')\ \Psi$

and $(\Psi,\ P,\ Q') \in Rel$

shows $\Psi \rhd P \lessapprox_w <Rel> Q$
using *assms*
by(*auto simp add*: *weakenStatImp-def*)

lemma *weakenStatImpE*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and Ψ' :: $'b$

assumes $\Psi \rhd P \lessapprox_w <Rel> Q$

obtains Q' **where** $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'$ **and** *insertAssertion*(*extractFrame* P) Ψ $\hookrightarrow_F$ *insertAssertion*(*extractFrame* Q') Ψ **and** $(\Psi,\ P,\ Q') \in Rel$
using *assms*
by(*auto simp add*: *weakenStatImp-def*)

lemma *weakStatImpWeakenStatImp*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes *cSim*: $\Psi \rhd P \lessapprox <Rel> Q$
and *cStatEq*: $\bigwedge \Psi'\ R\ S\ \Psi''.\ [\![(\Psi',\ R,\ S) \in Rel;\ \Psi' \simeq \Psi']\!] \Longrightarrow (\Psi'',\ R,\ S) \in Rel$

shows $\Psi \rhd P \lessapprox_w <Rel> Q$
proof −
from ‹$\Psi \rhd P \lessapprox <Rel> Q$›
obtain $Q'\ Q''$ **where** *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'$
and *PImpQ'*: *insertAssertion*(*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* Q') Ψ
and *Q'Chain*: $\Psi \otimes \mathbf{1} \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$ **and** $(\Psi \otimes \mathbf{1},\ P,\ Q'') \in Rel$
by(*rule weakStatImpE*)
from *Q'Chain Identity* **have** *Q'Chain*: $\Psi \rhd Q' \Longrightarrow\hat{}_\tau\ Q''$ **by**(*rule tauChainStatEq*)
with *QChain* **have** $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q''$ **by** *auto*
moreover from *Q'Chain* **have** *insertAssertion*(*extractFrame* Q') $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* Q'') Ψ
by(*rule statImpTauChainDerivative*)
with *PImpQ'* **have** *insertAssertion*(*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* Q'') Ψ
by(*rule FrameStatImpTrans*)
moreover from ‹$(\Psi \otimes \mathbf{1},\ P,\ Q'') \in Rel$› *Identity* **have** $(\Psi,\ P,\ Q'') \in Rel$ **by**(*rule cStatEq*)

ultimately show *?thesis* **by**(*rule weakenStatImpI*)
qed

lemma *weakenStatImpWeakStatImp*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *Q* :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \lessapprox_w$<*Rel*> *Q*
and *cExt*: $\bigwedge \Psi'\ R\ S\ \Psi''.\ (\Psi', R, S) \in Rel \Longrightarrow (\Psi' \otimes \Psi'', R, S) \in Rel$

shows $\Psi \rhd P \lessapprox$<*Rel*> *Q*
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')

from ‹$\Psi \rhd P \lessapprox_w$<*Rel*> *Q*›
obtain Q' **where** *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q'$
and *PImpQ'*: *insertAssertion*(*extractFrame P*) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* Q') Ψ
and $(\Psi, P, Q') \in Rel$
by(*rule weakenStatImpE*)
note *QChain PImpQ'*
moreover have $\Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau\ Q'$ **by** *simp*
moreover from ‹$(\Psi, P, Q') \in Rel$› **have** $(\Psi \otimes \Psi', P, Q') \in Rel$ **by**(*rule cExt*)
ultimately show *?case* **by** *blast*
qed

end

end

theory *Weaken-Simulation*
imports *Weaken-Stat-Imp*
begin

context *weak*
begin

definition
weakenSimulation :: *'b* ⇒ (*'a*, *'b*, *'c*) *psi* ⇒
(*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set* ⇒
(*'a*, *'b*, *'c*) *psi* ⇒ *bool* (‹- ⊳ - $\rightsquigarrow_w$<-> -› [*80*, *80*, *80*, *80*] *80*)
where
$\Psi \rhd P \rightsquigarrow_w$<*Rel*> $Q \equiv \forall \alpha\ Q'.\ \Psi \rhd Q \longmapsto \alpha \prec Q' \longrightarrow bn\ \alpha \sharp* \Psi \longrightarrow bn\ \alpha \sharp* P \longrightarrow (\exists P'.\ \Psi \rhd P \Longrightarrow \alpha \prec P' \wedge (\Psi, P', Q') \in Rel)$

lemma *weakenSimI*[*case-names cAct*]:
fixes Ψ :: *'b*

and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *rOutput*: $\bigwedge\alpha\ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp* \Psi;\ bn\ \alpha \sharp* P]\!] \Longrightarrow$
$\exists P'.\ \Psi \rhd P \Longrightarrow\alpha \prec P' \wedge (\Psi, P', Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow_w$<*Rel*> Q
using *assms*
by(*auto simp add*: *weakenSimulation-def*)

lemma *weakenSimWeakSim*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $(\Psi, P, Q) \in Rel$
and *cStatImp*: $\bigwedge\Psi'\ R\ S.\ (\Psi, R, S) \in Rel \Longrightarrow \Psi \rhd R \lessapprox_w$<*Rel*> S
and *cSim*: $\bigwedge\Psi'\ R\ S.\ (\Psi, R, S) \in Rel \Longrightarrow \Psi \rhd R \rightsquigarrow_w$<*Rel'*> S
and *cExt*: $\bigwedge\Psi'\ R\ S\ \Psi'.\ (\Psi, R, S) \in Rel' \Longrightarrow (\Psi \otimes \Psi', R, S) \in Rel'$
and *cSym*: $\bigwedge\Psi'\ R\ S.\ (\Psi, R, S) \in Rel \Longrightarrow (\Psi, S, R) \in Rel$

shows $\Psi \rhd P \rightsquigarrow$<*Rel'*> Q
proof(*induct rule*: *weakSimI2*)
case(*cAct* $\Psi'\ \alpha\ Q'$)
from ‹$(\Psi, P, Q) \in Rel$› **obtain** P''''
where *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P''''$
and *QImpP''''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'''') Ψ
and $(\Psi, P'''', Q) \in Rel$ **using** *weakenStatImp-def*
by(*metis cStatImp cSym*)

from ‹$(\Psi, P'''', Q) \in Rel$› **have** $\Psi \rhd P'''' \rightsquigarrow_w$<*Rel'*> Q **by**(*rule cSim*)
moreover from *PChain* ‹$bn\ \alpha \sharp* P$› **have** $bn\ \alpha \sharp* P''''$ **by**(*rule tauChainFreshChain*)
ultimately obtain P' **where** *P''''Trans*: $\Psi \rhd P'''' \Longrightarrow\alpha \prec P'$ **and** $(\Psi, P', Q') \in Rel'$
using ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹$bn\ \alpha \sharp* \Psi$›
by(*unfold weakenSimulation-def*, *auto*)

from *P''''Trans* ‹$\alpha \neq \tau$› **obtain** $P'''\ P''$ **where** *P''''Chain*: $\Psi \rhd P'''' \Longrightarrow\hat{}_\tau\ P'''$ **and** *P'''Trans*: $\Psi \rhd P''' \longmapsto\alpha \prec P''$ **and** *P''Chain*: $\Psi \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
by(*force simp add*: *weakenTransition-def*)
from *P''''Chain QImpP''''* **have** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P''') Ψ
by(*blast intro*: *statImpTauChainDerivative FrameStatImpTrans*)
with *PChain P''''Chain* **have** $\Psi : Q \rhd P \Longrightarrow\alpha \prec P''$ **using** *P'''Trans* **by**(*rule-tac*

weakTransitionI) *auto*
moreover from P''*Chain* **have** $\Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **by**(*rule weakenTauChain*)

moreover from ‹$(\Psi, P', Q') \in Rel'$› **have** $(\Psi \otimes \Psi', P', Q') \in Rel'$ **by**(*rule cExt*)
ultimately show *?case* **by** *blast*
next
case(*cTau* Q')
from ‹$(\Psi, P, Q) \in Rel$› **have** $\Psi \rhd P \rightsquigarrow_w <Rel'> Q$ **by**(*rule cSim*)
then obtain P' **where** $\Psi \rhd P \Longrightarrow \tau \prec P'$ **and** $(\Psi, P', Q') \in Rel'$ **using** ‹$\Psi \rhd Q \longmapsto \tau \prec Q'$›
by(*unfold weakenSimulation-def*, *fastforce*)
from ‹$\Psi \rhd P \Longrightarrow \tau \prec P'$› **have** $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **by**(*auto simp add*: *weakenTransition-def dest*: *tauActTauChain*)
with ‹$(\Psi, P', Q') \in Rel'$› **show** *?case* **by** *blast*
qed

lemma *weakSimWeakenSim*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*

assumes *cSim*: $\Psi \rhd P \rightsquigarrow <Rel> Q$
and *cStatEq*: $\bigwedge \Psi'\ R\ S\ \Psi''.\ [\![(\Psi', R, S) \in Rel;\ \Psi' \simeq \Psi']\!] \Longrightarrow (\Psi'', R, S) \in Rel$

shows $\Psi \rhd P \rightsquigarrow_w <Rel> Q$
proof(*induct rule*: *weakenSimI*)
case(*cAct* α Q')
show *?case*
proof(*cases* $\alpha = \tau$)
case *True*
from ‹$\Psi \rhd P \rightsquigarrow <Rel> Q$› ‹$\Psi \rhd Q \longmapsto \alpha \prec Q'$› ‹$\alpha = \tau$›
obtain P' **where** $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$ **and** $(\Psi, P', Q') \in Rel$
by(*blast dest*: *weakSimE*)
from ‹$\Psi \rhd P \Longrightarrow\hat{}_\tau P'$› **have** $\Psi \rhd P \Longrightarrow \tau \prec P'$
by(*induct rule*: *tauChainInduct*) (*auto simp add*: *weakenTransition-def*)
thus *?thesis* **using** ‹$(\Psi, P', Q') \in Rel$› ‹$\alpha = \tau$› **by** *blast*
next
case *False*
from ‹$\Psi \rhd P \rightsquigarrow <Rel> Q$› ‹$\Psi \rhd Q \longmapsto \alpha \prec Q'$› ‹$bn\ \alpha \sharp* \Psi$› ‹$bn\ \alpha \sharp* P$› ‹$\alpha \neq \tau$›
obtain $P''\ P'$ **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P''$ **and** P''*Chain*: $\Psi \otimes \mathbf{1} \rhd P'' \Longrightarrow\hat{}_\tau P'$ **and** $(\Psi \otimes \mathbf{1}, P', Q') \in Rel$
by(*blast dest*: *weakSimE*)
from *PTrans* **have** $\Psi \rhd P \Longrightarrow \alpha \prec P''$ **by**(*auto simp add*: *weakTransition-def weakenTransition-def*)
moreover from P''*Chain* **have** $\Psi \rhd P'' \Longrightarrow\hat{}_\tau P'$ **by**(*metis tauChainStatEq*

Identity)
moreover from ‹$(\Psi \otimes \mathbf{1},\ P',\ Q') \in Rel$› **have** $(\Psi,\ P',\ Q') \in Rel$ **by**(*metis cStatEq Identity*)
ultimately show *?thesis*
proof(*induct rule*: *weakenTransitionCases*)
case *cBase*
from ‹$\Psi \rhd P \Longrightarrow\hat{}_{\tau}\ P'$› **have** $\Psi \rhd P \Longrightarrow\tau \prec P'$
by(*induct rule*: *tauChainInduct*) (*auto simp add*: *weakenTransition-def*)
with ‹$(\Psi,\ P',\ Q') \in Rel$› **show** *?case* **by** *blast*
next
case(*cStep P'''' P'''*)
thus *?case*
apply(*unfold weakenTransition-def*)
by(*rule-tac x=P'* **in** *exI*) *fastforce*
qed
qed
qed

lemma *weakenSimE*:
fixes F :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)\ psi$

assumes $\Psi \rhd P \rightsquigarrow_w <Rel>\ Q$

shows $\bigwedge\alpha\ \ Q'.\ [\![\Psi \rhd Q \longmapsto\alpha \prec Q';\ bn\ \alpha \sharp*\ \Psi;\ bn\ \alpha \sharp*\ P]\!] \Longrightarrow$
$\exists P'.\ \Psi \rhd P \Longrightarrow\alpha \prec P' \wedge (\Psi,\ P',\ Q') \in Rel$
using *assms*
by(*auto simp add*: *weakenSimulation-def*)

lemma *weakenSimMonotonic*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and A :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and B :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $\Psi \rhd P \rightsquigarrow_w <A>\ Q$
and $A \subseteq B$

shows $\Psi \rhd P \rightsquigarrow_w <B>\ Q$
using *assms*
by(*simp* (*no-asm*) *add*: *weakenSimulation-def*) (*blast dest*: *weakenSimE*)

end

end

theory *Weaken-Bisimulation*
imports *Weaken-Simulation Weaken-Stat-Imp*
begin

context *weak*
begin

lemma *weakenMonoCoinduct*: $\bigwedge x\ y\ xa\ xb\ xc\ P\ Q\ \Psi.$
$x \leq y \Longrightarrow$
$(\Psi \rhd Q \rightsquigarrow_w <\{(xc,\ xb,\ xa).\ x\ xc\ xb\ xa\}> P) \longrightarrow$
$(\Psi \rhd Q \rightsquigarrow_w <\{(xb,\ xa,\ xc).\ y\ xb\ xa\ xc\}> P)$
apply *auto*
apply(*rule weakenSimMonotonic*)
by(*auto dest*: *le-funE*)

lemma *weakenMonoCoinduct2*: $\bigwedge x\ y\ xa\ xb\ xc\ P\ Q\ \Psi.$
$x \leq y \Longrightarrow$
$(\Psi \rhd Q \lesssim_w <\{(xc,\ xb,\ xa).\ x\ xc\ xb\ xa\}> P) \longrightarrow$
$(\Psi \rhd Q \lesssim_w <\{(xb,\ xa,\ xc).\ y\ xb\ xa\ xc\}> P)$
apply *auto*
apply(*rule weakenStatImpMonotonic*)
by(*auto dest*: *le-funE*)

coinductive-set *weakenBisim* :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
where
step: $[\![\Psi \rhd P \lesssim_w <weakenBisim> Q;\ \Psi \rhd P \rightsquigarrow_w <weakenBisim> Q;$
$\forall \Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in weakenBisim;\ (\Psi,\ Q,\ P) \in weakenBisim]\!] \Longrightarrow (\Psi,$
$P,\ Q) \in weakenBisim$
monos *weakenMonoCoinduct weakenMonoCoinduct2*

abbreviation
weakenBisimJudge (‹- ⊳ - $\approx_w$ -› [*70*, *70*, *70*] *65*) **where** $\Psi \rhd P \approx_w Q \equiv (\Psi,$
$P,\ Q) \in weakenBisim$
abbreviation
weakenBisimNilJudge (‹- $\approx_w$ -› [*70*, *70*] *65*) **where** $P \approx_w Q \equiv \mathbf{1} \rhd P \approx_w Q$

lemma *weakenBisimCoinductAux*[*consumes 1*]:
fixes $F :: 'b$
and $P :: ('a,\ 'b,\ 'c)\ psi$
and $Q :: ('a,\ 'b,\ 'c)\ psi$
and $X :: ('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $(\Psi,\ P,\ Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \rhd P \lesssim_w <(X \cup weakenBisim)> Q) \wedge$
$(\Psi \rhd P \rightsquigarrow_w <(X \cup weakenBisim)> Q) \wedge$
$(\forall \Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in X \vee (\Psi \otimes \Psi',\ P,\ Q) \in$
$weakenBisim) \wedge$
$((\Psi,\ Q,\ P) \in X \vee (\Psi,\ Q,\ P) \in weakenBisim)$

shows $(\Psi,\ P,\ Q) \in weakenBisim$
proof −
have $X \cup weakenBisim = \{(\Psi,\ P,\ Q).\ (\Psi,\ P,\ Q) \in X \vee (\Psi,\ P,\ Q) \in weakenBisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct* (*simp add*: *rtrancl-def*)
qed

lemma *weakenBisimCoinduct*[*consumes 1*, *case-names cStatImp cSim cExt cSym*]:
fixes $F :: 'b$
and $P :: ('a,\ 'b,\ 'c)\ psi$
and $Q :: ('a,\ 'b,\ 'c)\ psi$
and $X :: ('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $(\Psi,\ P,\ Q) \in X$
and $\bigwedge \Psi'\ R\ S.\ (\Psi',\ R,\ S) \in X \Longrightarrow \Psi' \rhd R \lessapprox_w <(X \cup weakenBisim)> S$
and $\bigwedge \Psi'\ R\ S.\ (\Psi',\ R,\ S) \in X \Longrightarrow \Psi' \rhd R \rightsquigarrow_w <(X \cup weakenBisim)> S$
and $\bigwedge \Psi'\ R\ S\ \Psi''.\ (\Psi',\ R,\ S) \in X \Longrightarrow (\Psi' \otimes \Psi'',\ R,\ S) \in X \vee \Psi' \otimes \Psi'' \rhd R \approx_w S$
and $\bigwedge \Psi'\ R\ S.\ (\Psi',\ R,\ S) \in X \Longrightarrow (\Psi',\ S,\ R) \in X \vee \Psi' \rhd S \approx_w R$

shows $\Psi \rhd P \approx_w Q$
proof −
have $X \cup weakenBisim = \{(\Psi,\ P,\ Q).\ (\Psi,\ P,\ Q) \in X \vee (\Psi,\ P,\ Q) \in weakenBisim\}$ **by** *auto*
with *assms* **show** *?thesis*
by *coinduct* (*simp add*: *rtrancl-def*)
qed

lemma *weakenBisimWeakCoinductAux*[*consumes 1*]:
fixes $\Psi :: 'b$
and $P :: ('a,\ 'b,\ 'c)\ psi$
and $Q :: ('a,\ 'b,\ 'c)\ psi$
and $X :: ('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes $(\Psi,\ P,\ Q) \in X$
and $\bigwedge \Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \lessapprox_w <X> Q \wedge$
$\Psi \rhd P \rightsquigarrow_w <X> Q \wedge (\forall \Psi'.\ (\Psi \otimes \Psi',\ P,\ Q) \in X) \wedge$
$(\Psi,\ Q,\ P) \in X$

shows $\Psi \rhd P \approx_w Q$
using *assms*
by(*coinduct rule*: *weakenBisimCoinductAux*) (*blast intro*: *weakenSimMonotonic weakenStatImpMonotonic*)

lemma *weakenBisimE*:
fixes $P :: ('a,\ 'b,\ 'c)\ psi$
and $Q :: ('a,\ 'b,\ 'c)\ psi$
and $\Psi :: 'b$

and $\Psi' :: 'b$

assumes $\Psi \rhd P \approx_w Q$

shows $\Psi \rhd P \lessapprox_w$<*weakenBisim*> Q
and $\Psi \rhd P \rightsquigarrow_w$<*weakenBisim*> Q
and $\Psi \otimes \Psi' \rhd P \approx_w Q$
and $\Psi \rhd Q \approx_w P$
using *assms*
by(*auto intro*: *weakenBisim.cases simp add*: *rtrancl-def*)

lemma *weakenBisimWeakCoinduct*[*consumes 1*, *case-names cStatImp cSim cExt cSym*]:
fixes $F :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $X :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \rhd P \lessapprox_w$<$X$> Q
and $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow_w$<$X$> Q
and $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi \otimes \Psi', P, Q) \in X$
and $\bigwedge\Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi, Q, P) \in X$

shows $(\Psi, P, Q) \in$ *weakenBisim*
proof −
have $X \cup$ *weakenBisim* $= \{(\Psi, P, Q).\ (\Psi, P, Q) \in X \vee (\Psi, P, Q) \in$ *weakenBisim*$\}$ **by** *auto*
with *assms* **show** *?thesis*
by(*coinduct rule*: *weakenBisimWeakCoinductAux*) *blast*
qed

lemma *weakenBisimEqWeakBisim*[*simp*]: *weakenBisim* = *weakBisim*
proof *auto*
fix $\Psi\ P\ Q$
assume $\Psi \rhd P \approx_w Q$ **thus** $\Psi \rhd P \approx Q$
proof(*coinduct rule*: *weakBisimWeakCoinduct*)
case(*cStatImp* $\Psi\ P\ Q$)
from ‹$\Psi \rhd P \approx_w Q$› **have** $\Psi \rhd P \lessapprox_w$<*weakenBisim*> Q **by**(*rule weakenBisimE*)
thus *?case* **using** *weakenBisimE*(*3*) **by**(*rule weakenStatImpWeakStatImp*)
next
case(*cSim* $\Psi\ P\ Q$)
from ‹$\Psi \rhd P \approx_w Q$› *weakenBisimE*
show *?case* **by**(*rule weakenSimWeakSim*)
next
case(*cExt* $\Psi\ P\ Q\ \Psi'$)
thus *?case* **by**(*rule weakenBisimE*)
next

case(*cSym* Ψ P Q)
thus *?case* **by**(*rule weakenBisimE*)
qed
next
fix Ψ P Q
assume $\Psi \rhd P \approx Q$ **thus** $\Psi \rhd P \approx_w Q$
proof(*coinduct rule*: *weakenBisimWeakCoinduct*)
case(*cStatImp* Ψ P Q)
from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \rhd P \lessapprox$<*weakBisim*> Q **by**(*rule weakBisimE*)
thus *?case* **using** *statEqWeakBisim* **by**(*rule weakStatImpWeakenStatImp*)
next
case(*cSim* Ψ P Q)
from ‹$\Psi \rhd P \approx Q$› **have** $\Psi \rhd P \rightsquigarrow$<*weakBisim*> Q **by**(*rule weakBisimE*)
thus *?case* **using** *statEqWeakBisim* **by**(*rule weakSimWeakenSim*)
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by**(*rule weakBisimE*)
next
case(*cSym* Ψ P Q)
thus *?case* **by**(*rule weakBisimE*)
qed
qed

lemma *weakenTransitiveWeakCoinduct*[*case-names cStatImp cSim cExt cSym*, *case-conclusion bisim step*, *consumes 2*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and X :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*) *set*

assumes *p*: $(\Psi, P, Q) \in X$
and *Eqvt*: *eqvt* X
and *rStatImp*: $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \rhd P \lessapprox_w$<$X$> Q
and *rSim*: $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow_w$<($\{(\Psi, P, Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge$
$(\Psi, P', Q') \in X \wedge$
$\Psi \rhd Q' \sim Q\}$)> Q
and *rExt*: $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi \otimes \Psi', P, Q) \in X$
and *rSym*: $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in X \Longrightarrow (\Psi, Q, P) \in X$

shows $\Psi \rhd P \approx_w Q$
proof −
from *p* ‹*eqvt* X› **have** $\Psi \rhd P \approx Q$
proof(*coinduct rule*: *weakTransitiveWeakCoinduct*)
case(*cStatImp* Ψ P Q)
from ‹$(\Psi, P, Q) \in X$› **have** $\Psi \rhd P \lessapprox_w$<$X$> Q **by**(*rule rStatImp*)
thus *?case* **using** *rExt* **by**(*rule weakenStatImpWeakStatImp*)
next
case(*cSim* Ψ P Q)

let $?Y = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in X \wedge \Psi \rhd Q' \sim Q\}$
note ‹$(\Psi,\ P,\ Q) \in X$›
moreover note *rStatImp rSim*
moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in ?Y \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in ?Y$
by(*blast dest*: *bisimE rExt*)
ultimately show *?case* **using** *rSym* **by**(*rule weakenSimWeakSim*)
next
case(*cExt* $\Psi\ P\ Q\ \Psi'$)
thus *?case* **by**(*rule rExt*)
next
case(*cSym* $\Psi\ P\ Q$)
thus *?case* **by**(*rule rSym*)
qed
thus *?thesis* **by**(*simp add*: *weakenBisimEqWeakBisim*)
qed

lemma *weakenTransitiveCoinduct*[*case-names cStatImp cSim cExt cSym*, *case-conclusion bisim step*, *consumes 2*]:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and X :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *p*: $(\Psi,\ P,\ Q) \in X$
and *Eqvt*: *eqvt X*
and *rStatImp*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \lessapprox_w <(X \cup weakenBisim)> Q$
and *rSim*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow \Psi \rhd P \rightsquigarrow_w <(\{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in (X \cup weakenBisim) \wedge \Psi \rhd Q' \sim Q\})> Q$
and *rExt*: $\bigwedge\Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in X \cup weakenBisim$
and *rSym*: $\bigwedge\Psi\ P\ Q.\ (\Psi,\ P,\ Q) \in X \Longrightarrow (\Psi,\ Q,\ P) \in X \cup weakenBisim$

shows $\Psi \rhd P \approx_w Q$
proof −
from *p* **have** $(\Psi,\ P,\ Q) \in X \cup weakenBisim$ **by** *auto*
moreover from ‹*eqvt X*› **have** $eqvt(X \cup weakenBisim)$ **by** *auto*
ultimately show *?thesis*
proof(*coinduct rule*: *weakenTransitiveWeakCoinduct*)
case(*cStatImp* $\Psi\ P\ Q$)
thus *?case* **by**(*fastforce intro*: *rStatImp weakenBisimE*(*1*) *weakenStatImpMonotonic*)
next
case(*cSim* $\Psi\ P\ Q$)
thus *?case* **by**(*fastforce intro*: *rSim weakenBisimE*(*2*) *weakenSimMonotonic bisimReflexive*)

next
case(*cExt* Ψ *P Q* Ψ')
thus *?case* **by**(*blast dest*: *weakenBisimE rExt*)
next
case(*cSym* Ψ *P Q*)
thus *?case* **by**(*blast dest*: *weakenBisimE rSym*)
qed
qed

end

end

theory *Weak-Cong-Simulation*
imports *Weak-Simulation Tau-Chain*
begin

context *env* **begin**

definition
weakCongSimulation :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$
$('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*) *set* $\Rightarrow$
$('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- ⊳ - ⇝«-» -› [*80*, *80*, *80*, *80*] *80*)
where
$\Psi \rhd P \rightsquigarrow$«*Rel*» $Q \equiv \forall Q'.\ \Psi \rhd Q \longmapsto\tau \prec Q' \longrightarrow (\exists P'.\ \Psi \rhd P \Longrightarrow_\tau P' \wedge (\Psi, P', Q') \in Rel)$

abbreviation
weakCongSimulationNilJudge (‹- ⇝«-» -› [*80*, *80*, *80*] *80*) **where** $P \rightsquigarrow$«*Rel*» $Q \equiv SBottom' \rhd P \rightsquigarrow$«*Rel*» Q

lemma *weakCongSimI*[*case-names cTau*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*) *set*
and Q :: $('a, 'b, 'c)$ *psi*
and C :: $'d$::*fs-name*

assumes *rTau*: $\bigwedge Q'.\ \Psi \rhd Q \longmapsto\tau \prec Q' \Longrightarrow \exists P'.\ \Psi \rhd P \Longrightarrow_\tau P' \wedge (\Psi, P', Q') \in Rel$

shows $\Psi \rhd P \rightsquigarrow$«*Rel*» Q
using *assms*
by(*auto simp add*: *weakCongSimulation-def*)

lemma *weakCongSimE*:
fixes F :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)\ psi$

assumes $\Psi \rhd P \rightsquigarrow «Rel»\ Q$
and $\Psi \rhd Q \longmapsto \tau \prec Q'$

obtains P' **where** $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $(\Psi, P', Q') \in Rel$
using *assms*
by(*auto simp add*: *weakCongSimulation-def*)

lemma *weakCongSimClosedAux*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)\ psi$
and p :: $name\ prm$

assumes *EqvtRel*: *eqvt Rel*
and *PSimQ*: $\Psi \rhd P \rightsquigarrow «Rel»\ Q$

shows $(p \cdot \Psi) \rhd (p \cdot P) \rightsquigarrow «Rel»\ (p \cdot Q)$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* Q')
from ‹$p \cdot \Psi \rhd p \cdot Q \longmapsto \tau \prec Q'$›
have $(rev\ p \cdot p \cdot \Psi) \rhd (rev\ p \cdot p \cdot Q) \longmapsto (rev\ p \cdot (\tau \prec Q'))$
by(*blast dest*: *semantics.eqvt*)
hence $\Psi \rhd Q \longmapsto \tau \prec (rev\ p \cdot Q')$ **by**(*simp add*: *eqvts*)
with *PSimQ* **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(\Psi, P', rev\ p \cdot Q') \in Rel$
by(*blast dest*: *weakCongSimE*)
from *PChain* **have** $(p \cdot \Psi) \rhd (p \cdot P) \Longrightarrow_\tau (p \cdot P')$ **by**(*rule tauStepChainEqvt*)
moreover from $P'RelQ'$ *EqvtRel* **have** $(p \cdot (\Psi, P', rev\ p \cdot Q')) \in Rel$
by(*simp only*: *eqvt-def*)
hence $(p \cdot \Psi, p \cdot P', Q') \in Rel$ **by**(*simp add*: *eqvts*)
ultimately show *?case*
by *blast*
qed

lemma *weakCongSimClosed*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)\ psi$
and Rel :: $('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and Q :: $('a, 'b, 'c)\ psi$
and p :: $name\ prm$

assumes *EqvtRel*: *eqvt Rel*

shows $\Psi \rhd P \rightsquigarrow «Rel»\ Q \Longrightarrow (p \cdot \Psi) \rhd (p \cdot P) \rightsquigarrow «Rel»\ (p \cdot Q)$
and $P \rightsquigarrow «Rel»\ Q \Longrightarrow (p \cdot P) \rightsquigarrow «Rel»\ (p \cdot Q)$

using *EqvtRel*
by(*force dest*: *weakCongSimClosedAux simp add*: *permBottom*)+

lemma *weakCongSimReflexive*:
fixes $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$
and $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

assumes $\{(\Psi, P, P) \mid \Psi\ P.\ True\} \subseteq Rel$

shows $\Psi \rhd P \rightsquigarrow «Rel»\ P$
using *assms*
by(*auto simp add*: *weakCongSimulation-def dest*: *rtrancl-into-rtrancl*)

lemma *weakStepSimTauChain*:
fixes $\Psi :: 'b$
and $Q :: ('a, 'b, 'c)\ psi$
and $Q' :: ('a, 'b, 'c)\ psi$
and $P :: ('a, 'b, 'c)\ psi$
and $Rel :: ('b \times ('a, 'b, 'c)\ psi \times ('a, 'b, 'c)\ psi)\ set$

assumes $\Psi \rhd Q \Longrightarrow_\tau Q'$
and $\Psi \rhd P \rightsquigarrow «Rel»\ Q$
and Sim: $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in Rel \Longrightarrow \Psi \rhd P \rightsquigarrow <Rel>\ Q$

obtains P' **where** $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $(\Psi, P', Q') \in Rel$
proof −
assume A: $\bigwedge P'.\ [\![\Psi \rhd P \Longrightarrow_\tau P';\ (\Psi, P', Q') \in Rel]\!] \Longrightarrow thesis$
from ‹$\Psi \rhd Q \Longrightarrow_\tau Q'$› ‹$\Psi \rhd P \rightsquigarrow «Rel»\ Q$› A **show** *?thesis*
proof(*induct arbitrary*: *P thesis rule*: *tauStepChainInduct*)
case(*TauBase Q Q' P*)
with ‹$\Psi \rhd P \rightsquigarrow «Rel»\ Q$› ‹$\Psi \rhd Q \longmapsto \tau \prec Q'$› **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(\Psi, P', Q') \in Rel$
by(*rule-tac weakCongSimE*)
thus *?case* **by**(*rule TauBase*)
next
case(*TauStep Q Q' Q'' P*)
from ‹$\Psi \rhd P \rightsquigarrow «Rel»\ Q$› **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $(\Psi, P', Q') \in Rel$
by(*rule TauStep*)
from ‹$(\Psi, P', Q') \in Rel$› **have** $\Psi \rhd P' \rightsquigarrow <Rel>\ Q'$ **by**(*rule Sim*)
then obtain P'' **where** $P'Chain$: $\Psi \rhd P' \Longrightarrow\hat{}_\tau P''$ **and** $(\Psi, P'', Q'') \in Rel$
using ‹$\Psi \rhd Q' \longmapsto \tau \prec Q''$› **by**(*blast dest*: *weakSimE*)
from *PChain P'Chain* **have** $\Psi \rhd P \Longrightarrow_\tau P''$ **by** *simp*
thus *?case* **using** ‹$(\Psi, P'', Q'') \in Rel$› **by**(*rule TauStep*)
qed
qed

lemma *weakCongSimTransitive*:

fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and *Rel'* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and T :: $('a, 'b, 'c)$ *psi*
and *Rel''* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*

assumes *PRelQ*: $(\Psi, P, Q) \in Rel$
and *PSimQ*: $\Psi \rhd P \rightsquigarrow$«*Rel*» Q
and *QSimR*: $\Psi \rhd Q \rightsquigarrow$«*Rel'*» R
and *Set*: $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ (\Psi, P, Q) \in Rel \wedge (\Psi, Q, R) \in Rel'\} \subseteq Rel''$
and *Sim*: $\bigwedge \Psi\ P\ Q.\ (\Psi, P, Q) \in Rel \Longrightarrow \Psi \rhd P \rightsquigarrow <Rel> Q$

shows $\Psi \rhd P \rightsquigarrow$«*Rel''*» R
proof(*induct rule*: *weakCongSimI*)
case(*cTau R'*)
from *QSimR* ‹$\Psi \rhd R \longmapsto \tau \prec R'$› **obtain** Q' **where** *QChain*: $\Psi \rhd Q \Longrightarrow_\tau Q'$ **and** *Q'RelR'*: $(\Psi, Q', R') \in Rel'$
by(*blast dest*: *weakCongSimE*)
from *QChain PSimQ Sim* **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** *P'RelQ'*: $(\Psi, P', Q') \in Rel$
by(*rule weakStepSimTauChain*)
moreover from *P'RelQ' Q'RelR' Set* **have** $(\Psi, P', R') \in Rel''$ **by** *blast*
ultimately show *?case* **by** *blast*
qed

lemma *weakCongSimStatEq*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow$«*Rel*» Q
and $\Psi \simeq \Psi'$
and *C1*: $\bigwedge \Psi\ P\ Q\ \Psi'.\ [\![(\Psi, P, Q) \in Rel; \Psi \simeq \Psi']\!] \Longrightarrow (\Psi', P, Q) \in Rel'$

shows $\Psi' \rhd P \rightsquigarrow$«*Rel'*» Q
proof(*induct rule*: *weakCongSimI*)
case(*cTau Q'*)
from ‹$\Psi' \rhd Q \longmapsto \tau \prec Q'$› ‹$\Psi \simeq \Psi'$›
have $\Psi \rhd Q \longmapsto \tau \prec Q'$ **by**(*metis statEqTransition AssertionStatEqSym*)
with *PSimQ* **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** *P'RelQ'*: $(\Psi, P', Q') \in Rel$
by(*blast dest*: *weakCongSimE*)

from *PChain* ‹$\Psi \simeq \Psi'$› **have** $\Psi' \rhd P \Longrightarrow_\tau P'$ **by**(*rule tauStepChainStatEq*)

moreover from ‹$(\Psi, P', Q') \in Rel$› ‹$\Psi \simeq \Psi'$› **have** $(\Psi', P', Q') \in Rel'$
by(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *weakCongSimMonotonic*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and A :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and B :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*

assumes $\Psi \rhd P \rightsquigarrow$«$A$» Q
and $A \subseteq B$

shows $\Psi \rhd P \rightsquigarrow$«$B$» Q
using *assms*
by(*simp* (*no-asm*) *add*: *weakCongSimulation-def*) (*blast dest*: *weakCongSimE*)+

lemma *strongSimWeakCongSim*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \rightsquigarrow[Rel]$ Q
and $Rel \subseteq Rel'$

shows $\Psi \rhd P \rightsquigarrow$«$Rel'$» Q
using *assms*
apply(*auto simp add*: *simulation-def weakCongSimulation-def*)
by(*erule-tac* x=τ **in** *allE*) *fastforce*

end

end

theory *Weak-Psi-Congruence*
imports *Weak-Cong-Simulation Weak-Bisimulation*
begin

context *env* **begin**

definition *weakPsiCongruence* :: $'b \Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow ('a, 'b, 'c)$ *psi* $\Rightarrow$ *bool* (‹- $\rhd$ - $\doteq$ -› [*70*, *70*, *70*] *65*)
where
$\Psi \rhd P \doteq Q \equiv \Psi \rhd P \approx Q \wedge \Psi \rhd P \rightsquigarrow$«*weakBisim*» $Q \wedge \Psi \rhd Q \rightsquigarrow$«*weakBisim*» P

abbreviation
weakPsiCongNilJudge (‹- ≐ -› [*70*, *70*] *65*) **where** $P \doteq Q \equiv \mathbf{1} \rhd P \doteq Q$

lemma *weakPsiCongSym*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \doteq Q$

shows $\Psi \rhd Q \doteq P$
using *assms*
by(*auto simp add*: *weakPsiCongruence-def weakBisimE*)

lemma *weakPsiCongE*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes $\Psi \rhd P \doteq Q$

shows $\Psi \rhd P \approx Q$
and $\Psi \rhd P \rightsquigarrow$«*weakBisim*» Q
and $\Psi \rhd Q \rightsquigarrow$«*weakBisim*» P
using *assms*
by(*auto simp add*: *weakPsiCongruence-def*)

lemma *weakPsiCongI*[*case-names cWeakBisim cSimLeft cSimRight*]:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and Ψ' :: $'b$

assumes $\Psi \rhd P \approx Q$
and $\Psi \rhd P \rightsquigarrow$«*weakBisim*» Q
and $\Psi \rhd Q \rightsquigarrow$«*weakBisim*» P

shows $\Psi \rhd P \doteq Q$
using *assms*
by(*auto simp add*: *weakPsiCongruence-def*)

lemma *weakPsiCongSymI*[*consumes 1*, *case-names cSym cWeakBisim cSim*]:
fixes Ψ :: $'b$
and P :: $'d$::*fs-name*
and Q :: $'d$
and Ψ' :: $'b$

assumes *Prop P Q*
and $\bigwedge P\ Q.\ Prop\ P\ Q \Longrightarrow Prop\ Q\ P$

and $\bigwedge P\ Q.\ Prop\ P\ Q \Longrightarrow \Psi \rhd (C\ P) \approx (C\ Q)$

and $\bigwedge P\ Q.\ Prop\ P\ Q \Longrightarrow \Psi \rhd (C\ P) \rightsquigarrow$«*weakBisim*» $(C\ Q)$

shows $\Psi \rhd (C\ P) \doteq (C\ Q)$
using *assms*
by(*rule-tac weakPsiCongI*) *auto*

lemma *weakPsiCongSym2*[*consumes 1*, *case-names cWeakBisim cSim*]:
fixes Ψ :: *'b*
and Ψ' :: *'b*

assumes $\Psi \rhd P \doteq Q$

and $\bigwedge P\ Q.\ \Psi \rhd P \doteq Q \Longrightarrow \Psi \rhd (C\ P) \approx (C\ Q)$

and $\bigwedge P\ Q.\ \Psi \rhd P \doteq Q \Longrightarrow \Psi \rhd (C\ P) \rightsquigarrow$«*weakBisim*» $(C\ Q)$

shows $\Psi \rhd (C\ P) \doteq (C\ Q)$
using *assms*
apply(*rule-tac weakPsiCongSymI*[**where** *C*=*C*])
apply *assumption*
by(*auto simp add*: *weakPsiCongruence-def dest*: *weakBisimE*)

lemma *statEqWeakCong*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and Ψ' :: *'b*

assumes $\Psi \rhd P \doteq Q$
and $\Psi \simeq \Psi'$

shows $\Psi' \rhd P \doteq Q$
proof −
let *?Prop* = $\lambda P\ Q.\ \Psi \rhd P \doteq Q \wedge \Psi \simeq \Psi'$
from *assms* **have** *?Prop P Q* **by** *auto*
thus *?thesis*
proof(*induct rule*: *weakPsiCongSymI*)
case(*cSym P Q*)
thus *?case* **by**(*blast dest*: *weakPsiCongSym*)
next
case(*cSim P Q*)
from ‹$\Psi \rhd P \doteq Q \wedge \Psi \simeq \Psi'$› **have** $\Psi \rhd P \doteq Q$ **and** $\Psi \simeq \Psi'$ **by** *simp*+
from ‹$\Psi \rhd P \doteq Q$› **have** $\Psi \rhd P \rightsquigarrow$«*weakBisim*» *Q* **by**(*rule weakPsiCongE*)
with ‹$\Psi \simeq \Psi'$› **show** *?case* **using** *statEqWeakBisim*

```
    by(rule-tac weakCongSimStatEq) auto
  next
    case(cWeakBisim P Q)
    from ‹Ψ ▷ P ≐ Q ∧ Ψ ≃ Ψ'› 
    have Ψ ▷ P ≈ Q and Ψ ≃ Ψ' by(auto dest: weakPsiCongE)
    thus ?case by(rule statEqWeakBisim)
  qed
qed

lemma weakPsiCongReflexive:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi


  shows Ψ ▷ P ≐ P
by(fastforce intro: weakPsiCongI weakCongSimReflexive weakBisimReflexive)

lemma weakPsiCongClosed:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   p :: name prm

  assumes Ψ ▷ P ≐ Q

  shows (p • Ψ) ▷  (p • P) ≐ (p • Q)
using assms
proof(induct rule: weakPsiCongSymI)
  case(cSym P Q)
  thus ?case by(rule weakPsiCongSym)
next
  case(cSim P Q)
  from ‹Ψ ▷ P ≐ Q› have Ψ ▷ P ⇝«weakBisim» Q by(rule weakPsiCongE)
   thus ?case by(drule-tac p=p in weakCongSimClosed(1)[OF weakBisimEqvt])
(simp add: eqvts)
next
  case(cWeakBisim P Q)
  from ‹Ψ ▷ P ≐ Q› have Ψ ▷ P ≈ Q by(rule weakPsiCongE)
  thus ?case by(rule weakBisimClosed)
qed

lemma weakPsiCongTransitive:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   R :: ('a, 'b, 'c) psi

  assumes Ψ ▷ P ≐ Q
  and     Ψ ▷ Q ≐ R
```

shows $\Psi \rhd P \doteq R$
proof −
from *assms* **have** $\Psi \rhd P \doteq Q \wedge \Psi \rhd Q \doteq R$ **by** *auto*
thus *?thesis*
proof(*induct rule*: *weakPsiCongSymI*)
case(*cSym P R*)
thus *?case* **by**(*auto dest*: *weakPsiCongSym*)
next
case(*cSim P R*)
hence $\Psi \rhd P \doteq Q$ **and** $\Psi \rhd Q \doteq R$ **by** *auto*
moreover from ‹$\Psi \rhd P \doteq Q$› **have** $\Psi \rhd P \approx Q$ **by**(*metis weakBisimE weakPsiCongE*)
moreover from ‹$\Psi \rhd P \doteq Q$› **have** $\Psi \rhd P \rightsquigarrow$«*weakBisim*» Q **by**(*rule weakPsiCongE*)
moreover from ‹$\Psi \rhd Q \doteq R$› **have** $\Psi \rhd Q \rightsquigarrow$«*weakBisim*» R **by**(*rule weakPsiCongE*)
moreover have $\{(\Psi, P, R) \mid \Psi\ P\ R.\ \exists Q.\ \Psi \rhd P \approx Q \wedge \Psi \rhd Q \approx R\} \subseteq$ *weakBisim*
by(*auto dest*: *weakBisimTransitive*)
ultimately show *?case* **using** *weakBisimE*(*2*) **by**(*rule-tac weakCongSimTransitive*)
next
case(*cWeakBisim P R*)
thus *?case* **by**(*auto dest*: *weakBisimTransitive weakPsiCongE*)
qed
qed

lemma *strongBisimWeakPsiCong*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \sim Q$

shows $\Psi \rhd P \doteq Q$
using *assms*
proof(*induct rule*: *weakPsiCongSymI*)
case(*cSym P Q*)
from ‹$\Psi \rhd P \sim Q$› **show** *?case* **by**(*rule bisimE*)
next
case(*cSim P Q*)
from ‹$\Psi \rhd P \sim Q$› **have** $\Psi \rhd P \rightsquigarrow$[*bisim*] Q **by**(*rule bisimE*)
thus $\Psi \rhd P \rightsquigarrow$«*weakBisim*» Q **using** *strongBisimWeakBisim*
by(*rule-tac strongSimWeakCongSim*) *auto*
next
case(*cWeakBisim P Q*)
thus *?case* **by**(*rule strongBisimWeakBisim*)
qed

lemma *structCongWeakPsiCong*:
  fixes P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*

  assumes $P \equiv_s Q$

  shows $P \doteq Q$
using *assms*
by(*metis structCongBisim strongBisimWeakPsiCong*)

end

end

theory *Weak-Cong-Sim-Pres*
  imports *Weak-Sim-Pres Weak-Cong-Simulation*
begin

context *env* **begin**

lemma *caseWeakSimPres*:
  fixes Ψ :: $'b$
  and *CsP* :: $('c \times ('a, 'b, 'c)$ *psi*$)$ *list*
  and *Rel* :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
  and *CsQ* :: $('c \times ('a, 'b, 'c)$ *psi*$)$ *list*
  and M :: $'a$
  and N :: $'a$

  assumes *PRelQ*: $\bigwedge\varphi\ Q.\ (\varphi, Q)$ *mem CsQ* $\Longrightarrow \exists P.\ (\varphi, P)$ *mem CsP* $\wedge$ *guarded* $P \wedge$ *Eq* $\Psi\ P\ Q$
  and *Sim*: $\bigwedge\Psi'\ P\ Q.\ (\Psi', P, Q) \in Rel \Longrightarrow \Psi' \rhd P \rightsquigarrow$<*Rel*> Q
  and *EqRel*: $\bigwedge\Psi'\ P\ Q.$ *Eq* $\Psi'\ P\ Q \Longrightarrow (\Psi', P, Q) \in Rel$
  and *EqSim*: $\bigwedge\Psi'\ P\ Q.$ *Eq* $\Psi'\ P\ Q \Longrightarrow \Psi' \rhd P \rightsquigarrow$«*Rel*» Q

  shows $\Psi \rhd$ *Cases CsP* $\rightsquigarrow$<*Rel*> *Cases CsQ*
proof(*induct rule*: *weakSimI2*)
  case(*cAct* $\Psi'\ \alpha\ Q'$)
  from ‹*bn* $\alpha \sharp*$ (*Cases CsP*)› **have** *bn* $\alpha \sharp*$ *CsP* **by** *auto*
  from ‹$\Psi \rhd$ *Cases CsQ* $\longmapsto\alpha \prec Q'$›
  show *?case*
  proof(*induct rule*: *caseCases*)
    case(*cCase* $\varphi\ Q$)
    from ‹(φ, Q) *mem CsQ*› **obtain** P **where** (φ, P) *mem CsP* **and** *guarded* P **and** *Eq* $\Psi\ P\ Q$
      by(*metis PRelQ*)
    from ‹*Eq* $\Psi\ P\ Q$› **have** $\Psi \rhd P \rightsquigarrow$<*Rel*> Q **by**(*metis EqRel Sim*)
    moreover note ‹$\Psi \rhd Q \longmapsto\alpha \prec Q'$› ‹*bn* $\alpha \sharp* \Psi$›
    moreover from ‹*bn* $\alpha \sharp*$ *CsP*› ‹(φ, P) *mem CsP*› **have** *bn* $\alpha \sharp* P$ **by**(*auto*

dest: *memFreshChain*)
ultimately obtain P'' P' **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow\alpha \prec P''$
and *P''Chain*: $\Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **and** *P'RelQ'*: $(\Psi \otimes \Psi', P', Q') \in Rel$
using ‹$\alpha \neq \tau$›
by(*blast dest*: *weakSimE*)
note *PTrans* ‹(φ, P) *mem CsP*› ‹$\Psi \vdash \varphi$› ‹*guarded P*›
moreover from ‹*guarded Q*› **have** *insertAssertion* (*extractFrame Q*) $\Psi \simeq_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$
by(*rule insertGuardedAssertion*)
hence *insertAssertion* (*extractFrame*(*Cases CsQ*)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame Q*) Ψ
by(*auto simp add*: *FrameStatEq-def*)
moreover from *Identity* **have** *insertAssertion* (*extractFrame*(*Cases CsQ*)) $\Psi \hookrightarrow_F \langle\varepsilon, \Psi\rangle$
by(*auto simp add*: *AssertionStatEq-def*)
ultimately have $\Psi : (Cases\ CsQ) \rhd Cases\ CsP \Longrightarrow\alpha \prec P''$
by(*rule weakCase*)
with *P''Chain P'RelQ'* **show** *?case* **by** *blast*
qed
next
case(*cTau Q'*)
from ‹$\Psi \rhd Cases\ CsQ \longmapsto\tau \prec Q'$› **show** *?case*
proof(*induct rule*: *caseCases*)
case(*cCase* φ *Q*)
from ‹(φ, Q) *mem CsQ*› **obtain** *P* **where** (φ, P) *mem CsP* **and** *guarded P* **and** *Eq* Ψ *P Q*
by(*metis PRelQ*)
from ‹*Eq* Ψ *P Q*› ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$›
obtain P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** *P'RelQ'*: $(\Psi, P', Q') \in Rel$
by(*blast dest*: *EqSim weakCongSimE*)
from *PChain* ‹(φ, P) *mem CsP*› ‹$\Psi \vdash \varphi$› ‹*guarded P*› **have** $\Psi \rhd Cases\ CsP \Longrightarrow_\tau P'$
by(*rule tauStepChainCase*)
hence $\Psi \rhd Cases\ CsP \Longrightarrow\hat{}_\tau P'$ **by**(*simp add*: *trancl-into-rtrancl*)
with *P'RelQ'* **show** *?case* **by** *blast*
qed
qed

lemma *weakCongSimCasePres*:
fixes Ψ :: *'b*
and *CsP* :: (*'c* × (*'a*, *'b*, *'c*) *psi*) *list*
and *Rel* :: (*'b* × (*'a*, *'b*, *'c*) *psi* × (*'a*, *'b*, *'c*) *psi*) *set*
and *CsQ* :: (*'c* × (*'a*, *'b*, *'c*) *psi*) *list*
and *M* :: *'a*
and *N* :: *'a*

assumes *PRelQ*: $\bigwedge\varphi\ Q.\ (\varphi, Q)$ *mem CsQ* $\Longrightarrow \exists P.\ (\varphi, P)$ *mem CsP* $\wedge$ *guarded P* $\wedge$ *Eq* Ψ *P Q*

and *EqSim*: $\bigwedge\Psi'\ P\ Q.\ Eq\ \Psi'\ P\ Q \Longrightarrow \Psi' \rhd P \rightsquigarrow\text{«}Rel\text{»}\ Q$

shows $\Psi \rhd Cases\ CsP \rightsquigarrow\text{«}Rel\text{»}\ Cases\ CsQ$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* Q')
from ‹$\Psi \rhd Cases\ CsQ \longmapsto\tau \prec Q'$› **show** *?case*
proof(*induct rule*: *caseCases*)
case(*cCase* $\varphi\ Q$)
from ‹$(\varphi,\ Q)$ *mem CsQ*› **obtain** P **where** $(\varphi,\ P)$ *mem CsP* **and** *guarded P* **and** $Eq\ \Psi\ P\ Q$
by(*metis PRelQ*)
from ‹$Eq\ \Psi\ P\ Q$› ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$›
obtain P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ Q') \in Rel$
by(*blast dest*: *EqSim weakCongSimE*)
from *PChain* ‹$(\varphi,\ P)$ *mem CsP*› ‹$\Psi \vdash \varphi$› ‹*guarded P*› **have** $\Psi \rhd Cases\ CsP \Longrightarrow_\tau P'$
by(*rule tauStepChainCase*)
with $P'RelQ'$ **show** *?case* **by** *blast*
qed
qed

lemma *weakCongSimResPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and x :: *name*
and Rel' :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow\text{«}Rel\text{»}\ Q$
and $eqvt\ Rel'$
and $x \sharp \Psi$
and $Rel \subseteq Rel'$
and *C1*: $\bigwedge\Psi'\ R\ S\ x.\ [\![(\Psi',\ R,\ S) \in Rel;\ x \sharp \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu x|\!)R,\ (\!|\nu x|\!)S) \in Rel'$

shows $\Psi \rhd (\!|\nu x|\!)P \rightsquigarrow\text{«}Rel'\text{»}\ (\!|\nu x|\!)Q$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* Q')
from ‹$\Psi \rhd (\!|\nu x|\!)Q \longmapsto\tau \prec Q'$› **have** $x \sharp Q'$ **by**(*auto dest*: *tauFreshDerivative simp add*: *abs-fresh*)
with ‹$\Psi \rhd (\!|\nu x|\!)Q \longmapsto\tau \prec Q'$› ‹$x \sharp \Psi$› **show** *?case*
proof(*induct rule*: *resTauCases*)
case(*cRes* Q')
from *PSimQ* ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$› **obtain** P' **where** *PChain*: $\Psi \rhd P \Longrightarrow_\tau P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ Q') \in Rel$
by(*blast dest*: *weakCongSimE*)
from *PChain* ‹$x \sharp \Psi$› **have** $\Psi \rhd (\!|\nu x|\!)P \Longrightarrow_\tau (\!|\nu x|\!)P'$ **by**(*rule tauStepChainResPres*)

moreover from $P'RelQ'$ ‹$x \sharp \Psi$› **have** $(\Psi, (\!|\nu x|\!)P', (\!|\nu x|\!)Q') \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *weakCongSimResChainPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and *xvec* :: *name list*

assumes *PSimQ*: $\Psi \rhd P \rightsquigarrow$«*Rel*» Q
and *eqvt Rel*
and *xvec* $\sharp*$ Ψ
and *C1*: $\bigwedge \Psi'\ R\ S\ xvec.\ [\![(\Psi', R, S) \in Rel; xvec \sharp* \Psi']\!] \Longrightarrow (\Psi', (\!|\nu{*}xvec|\!)R, (\!|\nu{*}xvec|\!)S) \in Rel$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow$«*Rel*» $(\!|\nu{*}xvec|\!)Q$
using ‹*xvec* $\sharp*$ Ψ›
proof(*induct xvec*)
case *Nil*
from *PSimQ* **show** *?case* **by** *simp*
next
case(*Cons x xvec*)
from ‹$(x\#xvec) \sharp* \Psi$› **have** $x \sharp \Psi$ **and** *xvec* $\sharp*$ Ψ **by** *simp*+
from ‹*xvec* $\sharp*$ $\Psi \Longrightarrow \Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow$«*Rel*» $(\!|\nu{*}xvec|\!)Q$› ‹*xvec* $\sharp*$ Ψ›
have $\Psi \rhd (\!|\nu{*}xvec|\!)P \rightsquigarrow$«*Rel*» $(\!|\nu{*}xvec|\!)Q$ **by** *simp*
moreover note ‹*eqvt Rel*› ‹$x \sharp \Psi$›
moreover have $Rel \subseteq Rel$ **by** *simp*
moreover have $\bigwedge \Psi\ P\ Q\ x.\ [\![(\Psi, P, Q) \in Rel; x \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu{*}[x]|\!)P, (\!|\nu{*}[x]|\!)Q) \in Rel$
by(*rule-tac xvec*=[*x*] **in** *C1*) *auto*
hence $\bigwedge \Psi\ P\ Q\ x.\ [\![(\Psi, P, Q) \in Rel; x \sharp \Psi]\!] \Longrightarrow (\Psi, (\!|\nu x|\!)P, (\!|\nu x|\!)Q) \in Rel$
by *simp*
ultimately have $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec|\!)P) \rightsquigarrow$«*Rel*» $(\!|\nu x|\!)((\!|\nu{*}xvec|\!)Q)$
by(*rule weakCongSimResPres*)
thus *?case* **by** *simp*
qed

lemma *weakCongSimParPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Rel :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*
and Rel' :: $('b \times ('a, 'b, 'c)$ *psi* $\times ('a, 'b, 'c)$ *psi*$)$ *set*

assumes *PSimQ*: $\bigwedge\Psi'.\ \Psi' \rhd P \rightsquigarrow\text{«}Rel\text{»}\ Q$
and *PSimQ'*: $\bigwedge\Psi'.\ \Psi' \rhd P \rightsquigarrow\text{<}Rel\text{>}\ Q$
and *StatImp*: $\bigwedge\Psi'.\ \Psi' \rhd Q \lesssim\text{<}Rel\text{>}\ P$

and *eqvt Rel*
and *eqvt Rel'*

and *Sym*: $\bigwedge\Psi'\ S\ T.\ [\![(\Psi',\ S,\ T) \in Rel]\!] \Longrightarrow (\Psi',\ T,\ S) \in Rel$
and *Ext*: $\bigwedge\Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel]\!] \Longrightarrow (\Psi' \otimes \Psi'',\ S,\ T) \in Rel$

and *C1*: $\bigwedge\Psi'\ S\ T\ A_U\ \Psi_U\ U.\ [\![(\Psi' \otimes \Psi_U,\ S,\ T) \in Rel;\ extractFrame\ U = \langle A_U,\ \Psi_U\rangle;\ A_U \sharp*\ \Psi';\ A_U \sharp*\ S;\ A_U \sharp*\ T]\!] \Longrightarrow (\Psi',\ S \parallel U,\ T \parallel U) \in Rel'$
and *C2*: $\bigwedge\Psi'\ S\ T\ xvec.\ [\![(\Psi',\ S,\ T) \in Rel';\ xvec \sharp*\ \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)S,\ (\!|\nu{*}xvec|\!)T) \in Rel'$
and *C3*: $\bigwedge\Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel;\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'',\ S,\ T) \in Rel$

shows $\Psi \rhd P \parallel R \rightsquigarrow\text{«}Rel'\text{»}\ Q \parallel R$
proof(*induct rule*: *weakCongSimI*)
case(*cTau QR*)
from ‹$\Psi \rhd Q \parallel R \longmapsto\tau \prec QR$› **show** *?case*
proof(*induct rule*: *parTauCases*[**where** C=(P, R)])
case(*cPar1* $Q'\ A_R\ \Psi_R$)
from ‹$A_R \sharp*\ (P,\ R)$› **have** $A_R \sharp*\ P$
by *simp*+
have *FrR*: $extractFrame\ R = \langle A_R,\ \Psi_R\rangle$ **by** *fact*
with ‹$A_R \sharp*\ \Psi$› ‹$A_R \sharp*\ P$› ‹$A_R \sharp*\ Q$› **have** $\Psi \otimes \Psi_R \rhd P \rightsquigarrow\text{«}Rel\text{»}\ Q$
by(*rule-tac PSimQ*)
moreover have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto\tau \prec Q'$ **by** *fact*
ultimately obtain P' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow_\tau P'$ **and** *P'RelQ'*: $(\Psi \otimes \Psi_R,\ P',\ Q') \in Rel$
by(*rule weakCongSimE*)
from *PChain QTrans* ‹$A_R \sharp*\ P$› ‹$A_R \sharp*\ Q$› **have** $A_R \sharp*\ P'$ **and** $A_R \sharp*\ Q'$
by(*force dest*: *freeFreshChainDerivative tauStepChainFreshChain*)+
from *PChain FrR* ‹$A_R \sharp*\ \Psi$› ‹$A_R \sharp*\ P$› **have** $\Psi \rhd P \parallel R \Longrightarrow_\tau (P' \parallel R)$
by(*rule tauStepChainPar1*)
moreover from *P'RelQ' FrR* ‹$A_R \sharp*\ \Psi$› ‹$A_R \sharp*\ P'$› ‹$A_R \sharp*\ Q'$› **have** $(\Psi,\ P' \parallel R,\ Q' \parallel R) \in Rel'$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2* $R'\ A_Q\ \Psi_Q$)
from ‹$A_Q \sharp*\ (P,\ R)$› **have** $A_Q \sharp*\ P$ **and** $A_Q \sharp*\ R$ **by** *simp*+
obtain $A_P\ \Psi_P$ **where** *FrP*: $extractFrame\ P = \langle A_P,\ \Psi_P\rangle$ **and** $A_P \sharp*\ (\Psi,\ A_Q,\ \Psi_Q,\ R)$
by(*rule freshFrame*)
hence $A_P \sharp*\ \Psi$ **and** $A_P \sharp*\ A_Q$ **and** $A_P \sharp*\ \Psi_Q$ **and** $A_P \sharp*\ R$
by *simp*+

have *FrQ*: $extractFrame\ Q = \langle A_Q,\ \Psi_Q\rangle$ **by** *fact*
from ‹$A_Q \sharp*\ P$› *FrP* ‹$A_P \sharp*\ A_Q$› **have** $A_Q \sharp*\ \Psi_P$

by(*drule-tac extractFrameFreshChain*) *auto*

obtain A_R Ψ_R **where** *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **and** $A_R \sharp* (\Psi, P, Q, A_Q, A_P, \Psi_Q, \Psi_P, R)$ **and** *distinct* A_R
by(*rule freshFrame*)
then have $A_R \sharp* \Psi$ **and** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* A_Q$ **and** $A_R \sharp* A_P$ **and** $A_R \sharp* \Psi_Q$ **and** $A_R \sharp* \Psi_P$
and $A_R \sharp* R$
by *simp*+

from ‹$A_Q \sharp* R$› *FrR* ‹$A_R \sharp* A_Q$› **have** $A_Q \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from ‹$A_P \sharp* R$› ‹$A_R \sharp* A_P$› *FrR* **have** $A_P \sharp* \Psi_R$ **by**(*drule-tac extractFrameFreshChain*) *auto*

moreover from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto\tau \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$›
obtain Ψ' A_R' Ψ_R' **where** $\Psi_R \otimes \Psi' \simeq \Psi_R'$ **and** *FrR'*: *extractFrame* $R' = \langle A_R', \Psi_R'\rangle$
and $A_R' \sharp* \Psi$ **and** $A_R' \sharp* P$ **and** $A_R' \sharp* Q$
by(*rule-tac* C=(Ψ, P, Q, R) **in** *expandTauFrame*) (*assumption* | *simp*)+

from *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$›
obtain P' P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P'$
and *QimpP'*: *insertAssertion*(*extractFrame* Q) ($\Psi \otimes \Psi_R$) $\hookrightarrow_F$ *insertAssertion*(*extractFrame* P') ($\Psi \otimes \Psi_R$)
and *P'Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P''$
and *P'RelQ*: $((\Psi \otimes \Psi_R) \otimes \Psi', P'', Q) \in Rel$
by(*metis StatImp weakStatImp-def Sym*)
obtain A_P' Ψ_P' **where** *FrP'*: *extractFrame* $P' = \langle A_P', \Psi_P'\rangle$ **and** $A_P' \sharp* \Psi$ **and** $A_P' \sharp* \Psi_R$ **and** $A_P' \sharp* \Psi_Q$
and $A_P' \sharp* A_Q$ **and** $A_P' \sharp* R$ **and** $A_P' \sharp* A_R$
by(*rule-tac* C=(Ψ, Ψ_R, Ψ_Q, A_Q, R, A_R) **in** *freshFrame*) *auto*

from *PChain P'Chain* ‹$A_R \sharp* P$› ‹$A_Q \sharp* P$› ‹$A_R' \sharp* P$› **have** $A_Q \sharp* P'$ **and** $A_R \sharp* P'$ **and** $A_R' \sharp* P'$ **and** $A_R' \sharp* P''$
by(*force intro*: *tauChainFreshChain*)+
from ‹$A_R \sharp* P'$› ‹$A_P' \sharp* A_R$› ‹$A_Q \sharp* P'$› ‹$A_P' \sharp* A_Q$› *FrP'* **have** $A_Q \sharp* \Psi_P'$ **and** $A_R \sharp* \Psi_P'$
by(*force dest*: *extractFrameFreshChain*)+

from *PChain FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau P' \parallel R$ **by**(*rule tauChainPar1*)
moreover have *RTrans*: $\Psi \otimes \Psi_P' \rhd R \longmapsto\tau \prec R'$
proof −
have $\Psi \otimes \Psi_Q \rhd R \longmapsto\tau \prec R'$ **by** *fact*
moreover have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P', (\Psi \otimes \Psi_P') \otimes \Psi_R\rangle$
proof −
have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$

by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym FrameStatEqSym*)
moreover with *FrP′ FrQ QimpP′* ‹A_P′ ♯* Ψ› ‹A_Q ♯* Ψ› ‹A_P′ ♯* Ψ_R› ‹A_Q ♯* Ψ_R›
have $\langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P', (\Psi \otimes \Psi_R) \otimes \Psi_P'\rangle$ **using** *freshCompChain*
by *simp*
moreover have $\langle A_P', (\Psi \otimes \Psi_R) \otimes \Psi_P'\rangle \simeq_F \langle A_P', (\Psi \otimes \Psi_P') \otimes \Psi_R\rangle$
by(*metis frameIntAssociativity Commutativity FrameStatEqTrans frameIntCompositionSym frameIntAssociativity*[*THEN FrameStatEqSym*])
ultimately show *?thesis*
by(*rule FrameStatEqImpCompose*)
qed
ultimately show *?thesis*
using ‹A_P′ ♯* Ψ› ‹A_P′ ♯* Ψ_Q› ‹A_Q ♯* Ψ› ‹A_Q ♯* Ψ_P′› ‹A_P′ ♯* R› ‹A_Q ♯* R›
‹A_P′ ♯* A_R› ‹A_R ♯* A_Q› ‹A_R ♯* Ψ_P′› ‹A_R ♯* Ψ_Q› ‹A_R ♯* Ψ› ‹A_R ♯* Ψ_P′› ‹A_Q ♯* Ψ_P′› *FrR* ‹*distinct* A_R›
by(*force intro*: *transferTauFrame*)
qed
hence $\Psi \rhd P' \parallel R \longmapsto\tau \prec (P' \parallel R')$ **using** *FrP′* ‹A_P′ ♯* Ψ› ‹A_P′ ♯* R›
by(*rule-tac Par2*) *auto*
moreover from *P′Chain* **have** $\Psi \otimes \Psi_R \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau P''$
by(*metis tauChainStatEq Associativity*)
with ‹$\Psi_R \otimes \Psi' \simeq \Psi_R$′› **have** $\Psi \otimes \Psi_R' \rhd P' \Longrightarrow\hat{}_\tau P''$
by(*rule-tac tauChainStatEq*, *auto*) (*metis compositionSym*)
hence $\Psi \rhd P' \parallel R' \Longrightarrow\hat{}_\tau P'' \parallel R'$ **using** *FrR′* ‹A_R′ ♯* Ψ› ‹A_R′ ♯* P′› **by**(*rule-tac tauChainPar1*)
ultimately have $\Psi \rhd P \parallel R \Longrightarrow_\tau (P'' \parallel R')$
by(*drule-tac tauActTauStepChain*) *auto*

moreover from *P′RelQ* ‹$\Psi_R \otimes \Psi' \simeq \Psi_R$′› **have** $(\Psi \otimes \Psi_R', P'', Q) \in Rel$ **by**(*blast intro*: *C3 Associativity compositionSym*)
with *FrR′* ‹A_R′ ♯* Ψ› ‹A_R′ ♯* P''› ‹A_R′ ♯* Q› **have** $(\Psi, P'' \parallel R', Q \parallel R') \in Rel'$ **by**(*rule-tac C1*)
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_R M N Q' A_Q Ψ_Q K *xvec* R' A_R)
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹A_Q ♯* (P, R)› **have** A_Q ♯* P **and** A_Q ♯* R **by** *simp+*

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹A_R ♯* (P, R)› **have** A_R ♯* P **and** A_R ♯* R **by** *simp+*
from ‹*xvec* ♯* (P, R)› **have** *xvec* ♯* P **and** *xvec* ♯* R **by** *simp+*

have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto M(\!|N|\!) \prec Q'$ **and** *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$ **by** *fact+*

from *RTrans FrR* ‹*distinct* A_R› ‹A_R ♯* R› ‹A_R ♯* *xvec*› ‹*xvec* ♯* R› ‹*xvec* ♯* Q› ‹*xvec* ♯* Ψ› ‹*xvec* ♯* Ψ_Q› ‹A_R ♯* Q›
‹A_R ♯* Ψ› ‹A_R ♯* Ψ_Q› ‹*xvec* ♯* K› ‹A_R ♯* K› ‹A_R ♯* R› ‹*xvec* ♯* R› ‹A_R ♯* P› ‹*xvec* ♯* P›
‹A_Q ♯* A_R› ‹A_Q ♯* *xvec*› ‹A_R ♯* K› ‹A_R ♯* N› ‹*xvec* ♯* K› ‹*distinct xvec*›
obtain p Ψ' $A_R{}'$ $\Psi_R{}'$ **where** S: *set* $p \subseteq$ *set xvec* $\times$ *set*($p \cdot$ *xvec*) **and** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$
and $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}'$ ♯* Q **and** $A_R{}'$ ♯* Ψ **and** $(p \cdot$ *xvec*$)$ ♯* Ψ
and $(p \cdot$ *xvec*$)$ ♯* Q **and** $(p \cdot$ *xvec*$)$ ♯* Ψ_Q **and** $(p \cdot$ *xvec*$)$ ♯* K **and** $(p \cdot$ *xvec*$)$ ♯* R
and $(p \cdot$ *xvec*$)$ ♯* P **and** $(p \cdot$ *xvec*$)$ ♯* A_Q **and** $A_R{}'$ ♯* P **and** $A_R{}'$ ♯* N
by(*rule-tac* C=(Ψ, Q, Ψ_Q, K, R, P, A_Q) **and** C'=(Ψ, Q, Ψ_Q, K, R, P, A_Q) **in** *expandFrame*) (*assumption* | *simp*)+

from ‹A_R ♯* Ψ› **have** $(p \cdot A_R)$ ♯* $(p \cdot \Psi)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* Ψ› ‹$(p \cdot$ *xvec*$)$ ♯* Ψ› S **have** $(p \cdot A_R)$ ♯* Ψ **by** *simp*
from ‹A_R ♯* P› **have** $(p \cdot A_R)$ ♯* $(p \cdot P)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* P› ‹$(p \cdot$ *xvec*$)$ ♯* P› S **have** $(p \cdot A_R)$ ♯* P **by** *simp*
from ‹A_R ♯* Q› **have** $(p \cdot A_R)$ ♯* $(p \cdot Q)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* Q› ‹$(p \cdot$ *xvec*$)$ ♯* Q› S **have** $(p \cdot A_R)$ ♯* Q **by** *simp*
from ‹A_R ♯* R› **have** $(p \cdot A_R)$ ♯* $(p \cdot R)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* R› ‹$(p \cdot$ *xvec*$)$ ♯* R› S **have** $(p \cdot A_R)$ ♯* R **by** *simp*
from ‹A_R ♯* K› **have** $(p \cdot A_R)$ ♯* $(p \cdot K)$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])
with ‹*xvec* ♯* K› ‹$(p \cdot$ *xvec*$)$ ♯* K› S **have** $(p \cdot A_R)$ ♯* K **by** *simp*

from ‹A_Q ♯* *xvec*› ‹$(p \cdot$ *xvec*$)$ ♯* A_Q› ‹A_Q ♯* M› S **have** A_Q ♯* $(p \cdot M)$ **by**(*simp add*: *freshChainSimps*)
from ‹A_Q ♯* *xvec*› ‹$(p \cdot$ *xvec*$)$ ♯* A_Q› ‹A_Q ♯* A_R› S **have** A_Q ♯* $(p \cdot A_R)$ **by**(*simp add*: *freshChainSimps*)

from *QTrans* S ‹*xvec* ♯* Q› ‹$(p \cdot$ *xvec*$)$ ♯* Q› **have** $(p \cdot (\Psi \otimes \Psi_R)) \rhd Q \longmapsto (p \cdot M)\llparenthesis N \rrparenthesis \prec Q'$
by(*rule-tac inputPermFrameSubject*) (*assumption* | *auto simp add*: *fresh-star-def*)+
with ‹*xvec* ♯* Ψ› ‹$(p \cdot$ *xvec*$)$ ♯* Ψ› S **have** *QTrans*: $(\Psi \otimes (p \cdot \Psi_R)) \rhd Q \longmapsto (p \cdot M)\llparenthesis N \rrparenthesis \prec Q'$
by(*simp add*: *eqvts*)
from *FrR* **have** $(p \cdot$ *extractFrame* $R) = p \cdot \langle A_R, \Psi_R\rangle$ **by** *simp*
with ‹*xvec* ♯* R› ‹$(p \cdot$ *xvec*$)$ ♯* R› S **have** *FrR*: *extractFrame* $R = \langle (p \cdot A_R), (p \cdot \Psi_R)\rangle$
by(*simp add*: *eqvts*)

from *MeqK* **have** $(p \cdot (\Psi \otimes \Psi_Q \otimes \Psi_R)) \vdash (p \cdot M) \leftrightarrow (p \cdot K)$ **by**(*rule chanEqClosed*)
with ‹*xvec* $\sharp*$ Ψ› ‹$(p \cdot xvec)$ $\sharp*$ Ψ› ‹*xvec* $\sharp*$ Ψ_Q› ‹$(p \cdot xvec)$ $\sharp*$ Ψ_Q› ‹*xvec* $\sharp*$ K› ‹$(p \cdot xvec)$ $\sharp*$ K› S
have *MeqK*: $\Psi \otimes \Psi_Q \otimes (p \cdot \Psi_R) \vdash (p \cdot M) \leftrightarrow K$ **by**(*simp add*: *eqvts*)

have $\Psi \otimes (p \cdot \Psi_R) \rhd P \rightsquigarrow$<*Rel*> Q **by**(*rule PSimQ'*)

with *QTrans* **obtain** P' P'' **where** *PTrans*: $\Psi \otimes (p \cdot \Psi_R) : Q \rhd P \Longrightarrow (p \cdot M)(\!|N|\!) \prec P''$
and *P''Chain*: $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$
and *P'RelQ'*: $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P', Q') \in Rel$
by(*fastforce dest*: *weakSimE*)
from *PTrans QTrans* ‹$A_R{}'$ $\sharp*$ P› ‹$A_R{}'$ $\sharp*$ Q› ‹$A_R{}'$ $\sharp*$ N› **have** $A_R{}'$ $\sharp*$ P'' **and** $A_R{}'$ $\sharp*$ Q'
by(*blast dest*: *weakInputFreshChainDerivative inputFreshChainDerivative*)+

from *PTrans* **obtain** P''' **where** *PChain*: $\Psi \otimes (p \cdot \Psi_R) \rhd P \Longrightarrow\hat{}_\tau P'''$
and *QimpP'''*: *insertAssertion* (*extractFrame* Q) $(\Psi \otimes (p \cdot \Psi_R)) \hookrightarrow_F$ *insertAssertion* (*extractFrame* P''') $(\Psi \otimes (p \cdot \Psi_R))$
and *P'''Trans*: $\Psi \otimes (p \cdot \Psi_R) \rhd P''' \longmapsto (p \cdot M)(\!|N|\!) \prec P''$
by(*rule weakTransitionE*)

from *PChain* ‹*xvec* $\sharp*$ P› ‹$(p \cdot A_R)$ $\sharp*$ P› ‹$A_R{}'$ $\sharp*$ P› **have** *xvec* $\sharp*$ P''' **and** $(p \cdot A_R)$ $\sharp*$ P''' **and** $A_R{}'$ $\sharp*$ P'''
by(*force intro*: *tauChainFreshChain*)+
from *P'''Trans* ‹$A_R{}'$ $\sharp*$ P'''› ‹$A_R{}'$ $\sharp*$ N› **have** $A_R{}'$ $\sharp*$ P'' **by**(*force dest*: *inputFreshChainDerivative*)

obtain $A_P{}'''$ $\Psi_P{}'''$ **where** *FrP'''*: *extractFrame* $P''' = \langle A_P{}''', \Psi_P{}'''\rangle$ **and** $A_P{}'''$ $\sharp*$ $(\Psi, A_Q, \Psi_Q, p \cdot A_R, p \cdot \Psi_R, p \cdot M, N, K, R, P''', xvec)$ **and** *distinct* $A_P{}'''$
by(*rule freshFrame*)
hence $A_P{}'''$ $\sharp*$ Ψ **and** $A_P{}'''$ $\sharp*$ A_Q **and** $A_P{}'''$ $\sharp*$ Ψ_Q **and** $A_P{}'''$ $\sharp*$ $(p \cdot M)$ **and** $A_P{}'''$ $\sharp*$ R
and $A_P{}'''$ $\sharp*$ N **and** $A_P{}'''$ $\sharp*$ K **and** $A_P{}'''$ $\sharp*$ $(p \cdot A_R)$ **and** $A_P{}'''$ $\sharp*$ P''' **and** $A_P{}'''$ $\sharp*$ *xvec* **and** $A_P{}'''$ $\sharp*$ $(p \cdot \Psi_R)$
by *simp*+

have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes (p \cdot \Psi_R)\rangle \simeq_F \langle A_Q, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_Q\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover with *QimpP''' FrP''' FrQ* ‹$A_P{}'''$ $\sharp*$ Ψ› ‹A_Q $\sharp*$ Ψ› ‹$A_P{}'''$ $\sharp*$ $(p \cdot \Psi_R)$› ‹A_Q $\sharp*$ Ψ_R› ‹A_Q $\sharp*$ *xvec*› ‹$(p \cdot xvec)$ $\sharp*$ A_Q› S
have $\langle A_Q, (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}''', (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_P{}'''\rangle$ **using** *freshCompChain*
by(*simp add*: *freshChainSimps*)
moreover have $\langle A_P{}''', (\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi_P{}'''\rangle \simeq_F \langle A_P{}''', (\Psi \otimes \Psi_P{}''') \otimes (p \cdot \Psi_R)\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatE-*

qTrans Composition Associativity)
ultimately have *QImpP'''*: $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes (p \cdot \Psi_R)\rangle \hookrightarrow_F \langle A_P''', (\Psi \otimes \Psi_P''') \otimes (p \cdot \Psi_R)\rangle$
by(*rule FrameStatEqImpCompose*)

from *PChain FrR* ‹$(p \cdot A_R) \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* P$› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau P''' \parallel R$ **by**(*rule tauChainPar1*)
moreover from *RTrans FrR P'''Trans MeqK QImpP''' FrP''' FrQ* ‹*distinct* A_P'''› ‹*distinct* A_R› ‹$A_P''' \sharp* (p \cdot A_R)$› ‹$A_Q \sharp* (p \cdot A_R)$›
‹$(p \cdot A_R) \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* P'''$› ‹$(p \cdot A_R) \sharp* Q$› ‹$(p \cdot A_R) \sharp* R$› ‹$(p \cdot A_R) \sharp* K$› ‹$A_P''' \sharp* \Psi$› ‹$A_P''' \sharp* R$›
‹$A_P''' \sharp* P'''$› ‹$A_P''' \sharp* (p \cdot M)$› ‹$A_Q \sharp* R$› ‹$A_Q \sharp* (p \cdot M)$›
obtain K' **where** $\Psi \otimes \Psi_P''' \rhd R \longmapsto K'(\!|\nu*xvec|\!)\langle N\rangle \prec R'$ **and** $\Psi \otimes \Psi_P''' \otimes (p \cdot \Psi_R) \vdash (p \cdot M) \leftrightarrow K'$ **and** $(p \cdot A_R) \sharp* K'$
by(*rule-tac comm1Aux*) (*assumption* | *simp*)+

with *P'''Trans FrP'''* **have** $\Psi \rhd P''' \parallel R \longmapsto\tau \prec (\!|\nu*xvec|\!)(P'' \parallel R')$ **using** *FrR* ‹$(p \cdot A_R) \sharp* \Psi$› ‹$(p \cdot A_R) \sharp* P'''$› ‹$(p \cdot A_R) \sharp* R$›
‹$xvec \sharp* P'''$› ‹$A_P''' \sharp* \Psi$› ‹$A_P''' \sharp* P'''$› ‹$A_P''' \sharp* R$› ‹$A_P''' \sharp* (p \cdot M)$› ‹$(p \cdot A_R) \sharp* K'$› ‹$A_P''' \sharp* (p \cdot A_R)$›
by(*rule-tac Comm1*)

moreover from *P''Chain* ‹$A_R' \sharp* P''$› **have** $A_R' \sharp* P'$ **by**(*rule tauChainFreshChain*)
from ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R'$› **have** $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \simeq \Psi \otimes \Psi_R'$
by(*metis Associativity AssertionStatEqTrans AssertionStatEqSym compositionSym*)
with *P''Chain* **have** $\Psi \otimes \Psi_R' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauChainStatEq*)
hence $\Psi \rhd P'' \parallel R' \Longrightarrow\hat{}_\tau P' \parallel R'$ **using** *FrR'* ‹$A_R' \sharp* \Psi$› ‹$A_R' \sharp* P''$› **by**(*rule tauChainPar1*)
hence $\Psi \rhd (\!|\nu*xvec|\!)(P'' \parallel R') \Longrightarrow\hat{}_\tau (\!|\nu*xvec|\!)(P' \parallel R')$ **using** ‹$xvec \sharp* \Psi$› **by**(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd P \parallel R \Longrightarrow_\tau (\!|\nu*xvec|\!)(P' \parallel R')$
by(*drule-tac tauActTauStepChain*) *auto*
moreover from *P'RelQ'* ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R'$› **have** $(\Psi \otimes \Psi_R', P', Q') \in Rel$ **by**(*metis C3 Associativity compositionSym*)
with *FrR'* ‹$A_R' \sharp* P'$› ‹$A_R' \sharp* Q'$› ‹$A_R' \sharp* \Psi$› **have** $(\Psi, P' \parallel R', Q' \parallel R') \in Rel'$ **by**(*rule-tac C1*)
with ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu*xvec|\!)(P' \parallel R'), (\!|\nu*xvec|\!)(Q' \parallel R')) \in Rel'$
by(*rule-tac C2*)
ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_R M $xvec$ N Q' A_Q Ψ_Q K R' A_R)
have *FrQ*: *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$ **by** *fact*
from ‹$A_Q \sharp* (P, R)$› **have** $A_Q \sharp* P$ **and** $A_Q \sharp* R$ **by** *simp*+

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹$A_R \sharp* (P, R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* R$ **by** *simp*+
from ‹$xvec \sharp* (P, R)$› **have** $xvec \sharp* P$ **and** $xvec \sharp* R$ **by** *simp*+

have *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q'$ **and** *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto K(\!|N|\!) \prec R'$
and *MeqK*: $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$ **by** *fact+*

from *RTrans FrR* ‹*distinct* A_R› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* R$› ‹$A_R \sharp* Q'$› ‹$A_R \sharp* N$› ‹$A_R \sharp* P$› ‹$A_R \sharp*$ *xvec*› ‹$A_R \sharp* K$›
obtain Ψ' $A_R{}'$ $\Psi_R{}'$ **where** *ReqR'*: $\Psi_R \otimes \Psi' \simeq \Psi_R{}'$ **and** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$
and $A_R{}' \sharp* \Psi$ **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q'$ **and** $A_R{}' \sharp* N$ **and** $A_R{}' \sharp*$ *xvec*
by(*rule-tac* C=(Ψ, P, Q', N, *xvec*) **and** C'=(Ψ, P, Q', N, *xvec*) **in** *expandFrame*) *auto*

have $\Psi \otimes \Psi_R \rhd P \rightsquigarrow$<*Rel*> Q **by**(*rule PSimQ'*)

with *QTrans* ‹*xvec* $\sharp* \Psi$› ‹*xvec* $\sharp* \Psi_R$› ‹*xvec* $\sharp* P$›
obtain P'' P' **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$
and *P''Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau P'$
and *P'RelQ'*: $((\Psi \otimes \Psi_R) \otimes \Psi', P', Q') \in Rel$
by(*fastforce dest*: *weakSimE*)

from *PTrans* **obtain** P''' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P'''$
and *QimpP'''*: *insertAssertion* (*extractFrame* Q) $(\Psi \otimes \Psi_R) \hookrightarrow_F$ *insertAssertion* (*extractFrame* P''') $(\Psi \otimes \Psi_R)$
and *P'''Trans*: $\Psi \otimes \Psi_R \rhd P''' \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P''$
by(*rule weakTransitionE*)

from *PChain* ‹$A_R \sharp* P$› **have** $A_R \sharp* P'''$ **by**(*rule tauChainFreshChain*)

obtain $A_P{}'''$ $\Psi_P{}'''$ **where** *FrP'''*: *extractFrame* $P''' = \langle A_P{}''', \Psi_P{}'''\rangle$ **and** $A_P{}''' \sharp* (\Psi, A_Q, \Psi_Q, A_R, \Psi_R, M, N, K, R, P''', xvec)$ **and** *distinct* $A_P{}'''$
by(*rule freshFrame*)
hence $A_P{}''' \sharp* \Psi$ **and** $A_P{}''' \sharp* A_Q$ **and** $A_P{}''' \sharp* \Psi_Q$ **and** $A_P{}''' \sharp* M$ **and** $A_P{}''' \sharp* R$
and $A_P{}''' \sharp* N$ **and** $A_P{}''' \sharp* K$ **and** $A_P{}''' \sharp* A_R$ **and** $A_P{}''' \sharp* P'''$ **and** $A_P{}''' \sharp*$ *xvec* **and** $A_P{}''' \sharp* \Psi_R$
by *simp+*

have $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \simeq_F \langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatEqTrans Composition Associativity*)
moreover with *QimpP''' FrP''' FrQ* ‹$A_P{}''' \sharp* \Psi$› ‹$A_Q \sharp* \Psi$› ‹$A_P{}''' \sharp* \Psi_R$› ‹$A_Q \sharp* \Psi_R$›
have $\langle A_Q, (\Psi \otimes \Psi_R) \otimes \Psi_Q\rangle \hookrightarrow_F \langle A_P{}''', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'''\rangle$ **using** *freshCompChain*
by *simp*
moreover have $\langle A_P{}''', (\Psi \otimes \Psi_R) \otimes \Psi_P{}'''\rangle \simeq_F \langle A_P{}''', (\Psi \otimes \Psi_P{}''') \otimes \Psi_R\rangle$
by(*metis frameResChainPres frameNilStatEq Commutativity AssertionStatE-*

qTrans Composition Associativity)
ultimately have *QImpP'''*: $\langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P''', (\Psi \otimes \Psi_P''') \otimes \Psi_R\rangle$
by(*rule FrameStatEqImpCompose*)

from *PChain FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› **have** $\Psi \rhd P \parallel R \Longrightarrow\hat{}_\tau P''' \parallel R$ **by**(*rule tauChainPar1*)
moreover from *RTrans FrR P'''Trans MeqK QImpP''' FrP''' FrQ* ‹*distinct* A_P'''› ‹*distinct* A_R› ‹$A_P''' \sharp* A_R$› ‹$A_Q \sharp* A_R$›
‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P'''$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* K$› ‹$A_P''' \sharp* \Psi$› ‹$A_P''' \sharp* R$›
‹$A_P''' \sharp* P'''$› ‹$A_P''' \sharp* M$› ‹$A_Q \sharp* R$› ‹$A_Q \sharp* M$› ‹$A_R \sharp* xvec$› ‹$xvec \sharp* M$›
obtain K' **where** $\Psi \otimes \Psi_P''' \rhd R \longmapsto K'(\!|N|\!) \prec R'$ **and** $\Psi \otimes \Psi_P''' \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $A_R \sharp* K'$
by(*rule-tac comm2Aux*) (*assumption* | *simp*)+

with *P'''Trans FrP'''* **have** $\Psi \rhd P''' \parallel R \longmapsto\tau \prec (\!|\nu*xvec|\!)(P'' \parallel R')$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P'''$› ‹$A_R \sharp* R$›
‹$xvec \sharp* R$› ‹$A_P''' \sharp* \Psi$› ‹$A_P''' \sharp* P'''$› ‹$A_P''' \sharp* R$› ‹$A_P''' \sharp* M$› ‹$A_R \sharp* K'$› ‹$A_P''' \sharp* A_R$›
by(*rule-tac Comm2*)
moreover from *P'''Trans* ‹$A_R \sharp* P'''$› ‹$A_R \sharp* xvec$› ‹$xvec \sharp* M$› ‹*distinct xvec*› **have** $A_R \sharp* P''$
by(*rule-tac outputFreshChainDerivative*) *auto*

from *PChain* ‹$A_R' \sharp* P$› **have** $A_R' \sharp* P'''$ **by**(*rule tauChainFreshChain*)
with *P'''Trans* ‹$xvec \sharp* M$› ‹*distinct xvec*› **have** $A_R' \sharp* P''$ **using** ‹$A_R' \sharp* xvec$›
by(*rule-tac outputFreshChainDerivative*) *auto*

with *P''Chain* **have** $A_R' \sharp* P'$ **by**(*rule tauChainFreshChain*)
from ‹$\Psi_R \otimes \Psi' \simeq \Psi_R'$› **have** $(\Psi \otimes \Psi_R) \otimes \Psi' \simeq \Psi \otimes \Psi_R'$
by(*metis Associativity AssertionStatEqTrans AssertionStatEqSym compositionSym*)
with *P''Chain* **have** $\Psi \otimes \Psi_R' \rhd P'' \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauChainStatEq*)
hence $\Psi \rhd P'' \parallel R' \Longrightarrow\hat{}_\tau P' \parallel R'$ **using** *FrR'* ‹$A_R' \sharp* \Psi$› ‹$A_R' \sharp* P''$›
by(*rule tauChainPar1*)
hence $\Psi \rhd (\!|\nu*xvec|\!)(P'' \parallel R') \Longrightarrow\hat{}_\tau (\!|\nu*xvec|\!)(P' \parallel R')$
using ‹$xvec \sharp* \Psi$› **by**(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd P \parallel R \Longrightarrow_\tau (\!|\nu*xvec|\!)(P' \parallel R')$ **by**(*drule-tac tauActTauStepChain*) *auto*
moreover from *P'RelQ'* ‹$\Psi_R \otimes \Psi' \simeq \Psi_R'$› **have** $(\Psi \otimes \Psi_R', P', Q') \in Rel$ **by**(*metis C3 Associativity compositionSym*)
with *FrR'* ‹$A_R' \sharp* P'$› ‹$A_R' \sharp* Q'$› ‹$A_R' \sharp* \Psi$› **have** $(\Psi, P' \parallel R', Q' \parallel R') \in Rel'$ **by**(*rule-tac C1*)
with ‹$xvec \sharp* \Psi$› **have** $(\Psi, (\!|\nu*xvec|\!)(P' \parallel R'), (\!|\nu*xvec|\!)(Q' \parallel R')) \in Rel'$
by(*rule-tac C2*)
ultimately show *?case* **by** *blast*
qed

qed
unbundle *no relcomp-syntax*

lemma *weakCongSimBangPres*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and Rel :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Rel' :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$
and Rel'' :: $('b \times ('a,\ 'b,\ 'c)\ psi \times ('a,\ 'b,\ 'c)\ psi)\ set$

assumes *PEqQ*: $Eq\ P\ Q$
and *PRelQ*: $(\Psi,\ P,\ Q) \in Rel$
and $guarded\ P$
and $guarded\ Q$
and $Rel'Rel$: $Rel' \subseteq Rel$
and *FrameParPres*: $\bigwedge \Psi'\ \Psi_U\ S\ T\ U\ A_U.\ [\![(\Psi' \otimes \Psi_U,\ S,\ T) \in Rel;\ extractFrame\ U = \langle A_U,\ \Psi_U\rangle;\ A_U \sharp\!*\ \Psi';\ A_U \sharp\!*\ S;\ A_U \sharp\!*\ T]\!] \Longrightarrow (\Psi',\ U \parallel S,\ U \parallel T) \in Rel$
and *C1*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi',\ S,\ T) \in Rel;\ guarded\ S;\ guarded\ T]\!] \Longrightarrow (\Psi',\ U \parallel !S,\ U \parallel !T) \in Rel''$
and *Closed*: $\bigwedge \Psi'\ S\ T\ p.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow ((p{::}name\ prm) \cdot \Psi',\ p \cdot S,\ p \cdot T) \in Rel$
and *Closed'*: $\bigwedge \Psi'\ S\ T\ p.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow ((p{::}name\ prm) \cdot \Psi',\ p \cdot S,\ p \cdot T) \in Rel'$
and *StatEq*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel;\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'',\ S,\ T) \in Rel$
and *StatEq'*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ [\![(\Psi',\ S,\ T) \in Rel';\ \Psi' \simeq \Psi'']\!] \Longrightarrow (\Psi'',\ S,\ T) \in Rel'$
and *Trans*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi',\ S,\ T) \in Rel;\ (\Psi',\ T,\ U) \in Rel]\!] \Longrightarrow (\Psi',\ S,\ U) \in Rel$
and *Trans'*: $\bigwedge \Psi'\ S\ T\ U.\ [\![(\Psi',\ S,\ T) \in Rel';\ (\Psi',\ T,\ U) \in Rel']\!] \Longrightarrow (\Psi',\ S,\ U) \in Rel'$
and *EqSim*: $\bigwedge \Psi'\ S\ T.\ Eq\ S\ T \Longrightarrow \Psi' \rhd S \rightsquigarrow «Rel»\ T$
and *cSim*: $\bigwedge \Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow \Psi' \rhd S \rightsquigarrow <Rel>\ T$
and *cSym*: $\bigwedge \Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi',\ T,\ S) \in Rel$
and *cSym'*: $\bigwedge \Psi'\ S\ T.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow (\Psi',\ T,\ S) \in Rel'$
and *cExt*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi' \otimes \Psi'',\ S,\ T) \in Rel$
and *cExt'*: $\bigwedge \Psi'\ S\ T\ \Psi''.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow (\Psi' \otimes \Psi'',\ S,\ T) \in Rel'$
and *ParPres*: $\bigwedge \Psi'\ S\ T\ U.\ (\Psi',\ S,\ T) \in Rel \Longrightarrow (\Psi',\ S \parallel U,\ T \parallel U) \in Rel$
and *ParPres'*: $\bigwedge \Psi'\ S\ T\ U.\ (\Psi',\ S,\ T) \in Rel' \Longrightarrow (\Psi',\ U \parallel S,\ U \parallel T) \in Rel'$
and *ParPres2*: $\bigwedge \Psi'\ S\ T.\ Eq\ S\ T \Longrightarrow Eq\ (S \parallel S)\ (T \parallel T)$
and *ResPres*: $\bigwedge \Psi'\ S\ T\ xvec.\ [\![(\Psi',\ S,\ T) \in Rel;\ xvec \sharp\!*\ \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)S,\ (\!|\nu{*}xvec|\!)T) \in Rel$
and *ResPres'*: $\bigwedge \Psi'\ S\ T\ xvec.\ [\![(\Psi',\ S,\ T) \in Rel';\ xvec \sharp\!*\ \Psi']\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)S,\ (\!|\nu{*}xvec|\!)T) \in Rel'$
and *Assoc*: $\bigwedge \Psi'\ S\ T\ U.\ (\Psi',\ S \parallel (T \parallel U),\ (S \parallel T) \parallel U) \in Rel$
and *Assoc'*: $\bigwedge \Psi'\ S\ T\ U.\ (\Psi',\ S \parallel (T \parallel U),\ (S \parallel T) \parallel U) \in Rel'$
and *ScopeExt*: $\bigwedge xvec\ \Psi'\ T\ S.\ [\![xvec \sharp\!*\ \Psi';\ xvec \sharp\!*\ T]\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)(S \parallel$

T), ($(\!|\nu{*}xvec|\!)S$) $\parallel$ T) $\in$ *Rel*
and *ScopeExt'*: $\bigwedge xvec\ \Psi'\ T\ S.\ [\![xvec \sharp{*}\ \Psi';\ xvec \sharp{*}\ T]\!] \Longrightarrow (\Psi',\ (\!|\nu{*}xvec|\!)(S \parallel T),\ ((\!|\nu{*}xvec|\!)S) \parallel T) \in Rel'$
and *Compose*: $\bigwedge \Psi'\ S\ T\ U\ O.\ [\![(\Psi',\ S,\ T) \in Rel;\ (\Psi',\ T,\ U) \in Rel'';\ (\Psi',\ U,\ O) \in Rel']\!] \Longrightarrow (\Psi',\ S,\ O) \in Rel''$
and *rBangActE*: $\bigwedge \Psi'\ S\ \alpha\ S'.\ [\![\Psi' \rhd !S \longmapsto \alpha \prec S';\ guarded\ S;\ bn\ \alpha \sharp{*}\ S;\ \alpha \neq \tau;\ bn\ \alpha \sharp{*}\ subject\ \alpha]\!] \Longrightarrow \exists\ T.\ \Psi' \rhd S \longmapsto \alpha \prec T \wedge (\mathbf{1},\ S',\ T \parallel !S) \in Rel'$
and *rBangTauE*: $\bigwedge \Psi'\ S\ S'.\ [\![\Psi' \rhd !S \longmapsto \tau \prec S';\ guarded\ S]\!] \Longrightarrow \exists\ T.\ \Psi' \rhd S \parallel S \longmapsto \tau \prec T \wedge (\mathbf{1},\ S',\ T \parallel !S) \in Rel'$
and *rBangTauI*: $\bigwedge \Psi'\ S\ S'.\ [\![\Psi' \rhd S \parallel S \Longrightarrow_\tau S';\ guarded\ S]\!] \Longrightarrow \exists\ T.\ \Psi' \rhd !S \Longrightarrow_\tau T \wedge (\Psi',\ T,\ S' \parallel !S) \in Rel'$
shows $\Psi \rhd R \parallel !P \rightsquigarrow$«*Rel''*» $R \parallel !Q$
proof(*induct rule*: *weakCongSimI*)
case(*cTau RQ'*)
from ‹$\Psi \rhd R \parallel !Q \longmapsto \tau \prec RQ'$› **show** *?case*
proof(*induct rule*: *parTauCases*[**where** C=(P, Q, R)])
case(*cPar1 R' A_Q Ψ_Q*)
from ‹*extractFrame* $(!Q) = \langle A_Q,\ \Psi_Q\rangle$› **have** $A_Q = []$ **and** $\Psi_Q = SBottom'$ **by** *simp*+
with ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto \tau \prec R'$› ‹$\Psi_Q = SBottom'$›
have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (R' \parallel !P)$ **by**(*rule-tac Par1*) (*assumption* | *simp*)+
hence $\Psi \rhd R \parallel !P \Longrightarrow_\tau R' \parallel !P$ **by** *auto*
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi,\ R' \parallel !P,\ R' \parallel !Q) \in Rel''$ **using** ‹*guarded P*› ‹*guarded Q*›
by(*rule C1*)
ultimately show *?case* **by** *blast*
next
case(*cPar2 Q' A_R Ψ_R*)
from ‹$A_R \sharp{*}\ (P,\ Q,\ R)$› **have** $A_R \sharp{*}\ P$ **and** $A_R \sharp{*}\ Q$ **and** $A_R \sharp{*}\ R$ **by** *simp*+
have *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$ **by** *fact*

obtain A_Q Ψ_Q **where** *FrQ*: *extractFrame* $Q = \langle A_Q,\ \Psi_Q\rangle$ **and** $A_Q \sharp{*}\ \Psi$ **and** $A_Q \sharp{*}\ \Psi_R$ **and** $A_Q \sharp{*}\ A_R$
by(*rule-tac* C=(Ψ, Ψ_R, A_R) **in** *freshFrame*) *auto*
from *FrQ* ‹*guarded Q*› **have** $\Psi_Q \simeq \mathbf{1}$ **and** *supp* Ψ_Q = ({}::*name set*) **by**(*blast dest*: *guardedStatEq*)+
hence $A_R \sharp{*}\ \Psi_Q$ **and** $A_Q \sharp{*}\ \Psi_Q$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto \tau \prec Q'$› ‹*guarded Q*›
obtain T **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \parallel Q \longmapsto \tau \prec T$ **and** $(\mathbf{1},\ Q',\ T \parallel !Q) \in Rel'$
by(*blast dest*: *rBangTauE*)

from ‹*Eq P Q*› **have** *Eq* $(P \parallel P)$ $(Q \parallel Q)$ **by**(*rule ParPres2*)
with *QTrans*
obtain S **where** *PTrans*: $\Psi \otimes \Psi_R \rhd P \parallel P \Longrightarrow_\tau S$ **and** *SRelT*: $(\Psi \otimes \Psi_R,\ S,\ T) \in Rel$
by(*blast dest*: *EqSim weakCongSimE*)
from *PTrans* ‹*guarded P*› **obtain** U **where** *PChain*: $\Psi \otimes \Psi_R \rhd !P \Longrightarrow_\tau U$

and $(\Psi \otimes \Psi_R,\ U,\ S \parallel !P) \in Rel'$
by(*blast dest*: *rBangTauI*)
from *PChain* ‹$A_R \sharp* P$› **have** $A_R \sharp* U$ **by**(*force dest*: *tauStepChainFreshChain*)
from ‹$\Psi \otimes \Psi_R \rhd !P \Longrightarrow_\tau U$› *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$› **have** $\Psi \rhd R \parallel !P \Longrightarrow_\tau R \parallel U$
by(*rule-tac tauStepChainPar2*) *auto*
moreover from *PTrans* ‹$A_R \sharp* P$› **have** $A_R \sharp* S$ **by**(*force dest*: *tauStepChainFreshChain*)
from *QTrans* ‹$A_R \sharp* Q$› **have** $A_R \sharp* T$ **by**(*force dest*: *tauFreshChainDerivative*)
have $(\Psi,\ R \parallel U,\ R \parallel Q') \in Rel''$
proof −
from ‹$(\Psi \otimes \Psi_R,\ U,\ S \parallel !P) \in Rel'$› *Rel'Rel* **have** $(\Psi \otimes \Psi_R,\ U,\ S \parallel !P) \in Rel$
by *auto*
hence $(\Psi,\ R \parallel U,\ R \parallel (S \parallel !P)) \in Rel$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* U$› ‹$A_R \sharp* S$› ‹$A_R \sharp* P$›
by(*rule-tac FrameParPres*) *auto*

moreover from ‹$(\Psi \otimes \Psi_R,\ S,\ T) \in Rel$› *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* S$› ‹$A_R \sharp* T$› **have** $(\Psi,\ R \parallel S,\ R \parallel T) \in Rel$
by(*rule-tac FrameParPres*) *auto*
hence $(\Psi,\ R \parallel T,\ R \parallel S) \in Rel$ **by**(*rule cSym*)
hence $(\Psi,\ (R \parallel T) \parallel !P,\ (R \parallel S) \parallel !P) \in Rel$ **by**(*rule ParPres*)
hence $(\Psi,\ (R \parallel S) \parallel !P,\ (R \parallel T) \parallel !P) \in Rel$ **by**(*rule cSym*)
hence $(\Psi,\ R \parallel (S \parallel !P),\ (R \parallel T) \parallel !P) \in Rel$ **by**(*metis Trans Assoc*)
ultimately have $(\Psi,\ R \parallel U,\ (R \parallel T) \parallel !P) \in Rel$ **by**(*rule Trans*)
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi,\ (R \parallel T) \parallel !P,\ (R \parallel T) \parallel !Q) \in Rel''$ **using** ‹*guarded P*› ‹*guarded Q*› **by**(*rule C1*)
moreover from ‹$(\mathbf{1},\ Q',\ T \parallel !Q) \in Rel'$› **have** $(\mathbf{1} \otimes \Psi,\ Q',\ T \parallel !Q) \in Rel'$ **by**(*rule cExt'*)
hence $(\Psi,\ Q',\ T \parallel !Q) \in Rel'$
by(*rule StatEq'*) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence $(\Psi,\ R \parallel Q',\ R \parallel (T \parallel !Q)) \in Rel'$ **by**(*rule ParPres'*)
hence $(\Psi,\ R \parallel Q',\ (R \parallel T) \parallel !Q) \in Rel'$ **by**(*metis Trans' Assoc'*)
hence $(\Psi,\ (R \parallel T) \parallel !Q,\ R \parallel Q') \in Rel'$ **by**(*rule cSym'*)
ultimately show *?thesis* **by**(*rule-tac Compose*)
qed
ultimately show *?case* **by** *blast*
next
case(*cComm1* Ψ_Q M N R' A_R Ψ_R K *xvec* Q' A_Q)
from ‹$A_R \sharp* (P,\ Q,\ R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* R$ **by** *simp+*
from ‹*xvec* $\sharp* (P,\ Q,\ R)$› **have** *xvec* $\sharp* P$ **and** *xvec* $\sharp* Q$ **and** *xvec* $\sharp* R$ **by** *simp+*
have *FrQ*: *extractFrame*$(!Q) = \langle A_Q,\ \Psi_Q\rangle$ **by** *fact*
have *FrR*: *extractFrame* $R = \langle A_R,\ \Psi_R\rangle$ **by** *fact*
from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R \sharp* R$› ‹$A_R \sharp* N$› ‹$A_R \sharp*$ *xvec*› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* M$›
obtain A_R' Ψ_R' Ψ' **where** *FrR'*: *extractFrame* $R' = \langle A_R',\ \Psi_R'\rangle$ **and** $\Psi_R \otimes \Psi'$

$\simeq \Psi_R{}'$ **and** $A_R{}' \sharp* xvec$ **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q$ **and** $A_R{}' \sharp* \Psi$
by(*rule-tac* C=(Ψ, *xvec*, P, Q) **and** C'=(Ψ, *xvec*, P, Q) **in** *expandFrame*) *auto*
from ‹(Ψ, P, Q) $\in$ *Rel*› **have** ($\Psi \otimes \Psi_R$, P, Q) $\in$ *Rel* **by**(*rule cEx*t)
moreover from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec Q'$› ‹*guarded* Q› ‹*xvec* $\sharp*$ Q› ‹*xvec* $\sharp*$ K›
obtain S **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec S$ **and** ($\mathbf{1}$, Q', $S \parallel !Q$) $\in$ *Rel'*
by(*fastforce dest*: *rBangActE*)
ultimately obtain P' T **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **and** *P'Chain*: $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P' \Longrightarrow\hat{}_\tau T$ **and** $((\Psi \otimes \Psi_R) \otimes \Psi', T, S) \in$ *Rel*
using ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ Ψ_R› ‹*xvec* $\sharp*$ P›
by(*fastforce dest*: *cSim weakSimE*)

from *PTrans* ‹$A_R \sharp* P$› ‹$A_R \sharp*$ *xvec*› ‹$A_R{}' \sharp* P$› ‹$A_R{}' \sharp*$ *xvec*› ‹*xvec* $\sharp*$ K› ‹*distinct xvec*›
have $A_R \sharp* P'$ **and** $A_R{}' \sharp* P'$
by(*force dest*: *weakOutputFreshChainDerivative*)+
with *P'Chain* **have** $A_R{}' \sharp* T$ **by**(*force dest*: *tauChainFreshChain*)+
from *QTrans* ‹$A_R{}' \sharp* Q$› ‹$A_R{}' \sharp*$ *xvec*› ‹*xvec* $\sharp*$ K› ‹*distinct xvec*›
have $A_R{}' \sharp* S$ **by**(*force dest*: *outputFreshChainDerivative*)

obtain $A_Q{}'$ $\Psi_Q{}'$ **where** *FrQ'*: *extractFrame* $Q = \langle A_Q{}', \Psi_Q{}'\rangle$ **and** $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* \Psi_R$ **and** $A_Q{}' \sharp* A_R$ **and** $A_Q{}' \sharp* M$ **and** $A_Q{}' \sharp* R$ **and** $A_Q{}' \sharp* K$
by(*rule-tac* C=(Ψ, Ψ_R, A_R, K, M, R) **in** *freshFrame*) *auto*
from *FrQ'* ‹*guarded* Q› **have** $\Psi_Q{}' \simeq \mathbf{1}$ **and** *supp* $\Psi_Q{}' = (\{\}$::*name set*) **by**(*blast dest*: *guardedStatEq*)+
hence $A_Q{}' \sharp* \Psi_Q{}'$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P''$
and *NilImpP''*: $\langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') ($\Psi \otimes \Psi_R$)
and *P''Trans*: $\Psi \otimes \Psi_R \rhd P'' \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$
using *FrQ'* ‹$A_Q{}' \sharp* \Psi$› ‹$A_Q{}' \sharp* \Psi_R$› *freshCompChain*
by(*drule-tac weakTransitionE*) *auto*

from *PChain* **have** $\Psi \rhd R \parallel !P \Longrightarrow_\tau (\!|\nu*xvec|\!)(R' \parallel (P' \parallel !P))$
proof(*induct rule*: *tauChainCases*)
case *TauBase*
from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$› *FrQ* **have** $\Psi \otimes \mathbf{1} \rhd R \longmapsto M(\!|N|\!) \prec R'$ **by** *simp*
moreover note *FrR*
moreover from *P''Trans* ‹$P = P''$› **have** $\Psi \otimes \Psi_R \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **by** *simp*
hence $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec P'$ **by**(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \longmapsto K(\!|\nu*xvec|\!)\langle N\rangle \prec (P' \parallel !P)$ **using** ‹*xvec* $\sharp*$ Ψ› ‹*xvec* $\sharp*$ Ψ_R› ‹*xvec* $\sharp*$ P›

by(*force intro*: *Par1*)
hence $\Psi \otimes \Psi_R \rhd !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec (P' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover from ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› *FrQ* **have** $\Psi \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K$ **by** *simp*
ultimately have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(R' \parallel (P' \parallel !P))$ **using** ‹$A_R \sharp{*} \Psi$› ‹$A_R \sharp{*} R$› ‹$A_R \sharp{*} P$› ‹$A_R \sharp{*} M$› ‹$xvec \sharp{*} R$›
by(*force intro*: *Comm1*)
thus *?case* **by**(*rule tauActTauStepChain*)
next
case *TauStep*
obtain $A_P{''}\ \Psi_P{''}$ **where** *FrP''*: *extractFrame* $P'' = \langle A_P{''}, \Psi_P{''}\rangle$ **and** $A_P{''} \sharp{*} \Psi$ **and** $A_P{''} \sharp{*} K$ **and** $A_P{''} \sharp{*} \Psi_R$ **and** $A_P{''} \sharp{*} R$ **and** $A_P{''} \sharp{*} P''$ **and** $A_P{''} \sharp{*} P$
and $A_P{''} \sharp{*} A_R$ **and** *distinct* $A_P{''}$
by(*rule-tac* $C{=}(\Psi, K, A_R, \Psi_R, R, P'', P)$ **in** *freshFrame*) *auto*
from *PChain* ‹$A_R \sharp{*} P$› **have** $A_R \sharp{*} P''$ **by**(*drule-tac tauChainFreshChain*) *auto*
with *FrP''* ‹$A_P{''} \sharp{*} A_R$› **have** $A_R \sharp{*} \Psi_P{''}$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from ‹$\Psi \otimes \Psi_R \rhd P \Longrightarrow_\tau P''$› **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \Longrightarrow_\tau P''$ **by**(*rule tauStepChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \Longrightarrow_\tau P'' \parallel !P$ **by**(*rule-tac tauStepChainPar1*) *auto*
hence $\Psi \otimes \Psi_R \rhd !P \Longrightarrow_\tau P'' \parallel !P$ **using** ‹*guarded P*› **by**(*rule tauStepChainBang*)
hence $\Psi \rhd R \parallel !P \Longrightarrow_\tau R \parallel (P'' \parallel !P)$ **using** *FrR* ‹$A_R \sharp{*} \Psi$› ‹$A_R \sharp{*} P$›
by(*rule-tac tauStepChainPar2*) *auto*
moreover have $\Psi \rhd R \parallel (P'' \parallel !P) \longmapsto \tau \prec (\!|\nu{*}xvec|\!)(R' \parallel (P' \parallel !P))$
proof −
from *FrQ* ‹$\Psi_Q{'} \simeq \mathbf{1}$› ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|N|\!) \prec R'$› **have** $\Psi \otimes \Psi_Q{'} \rhd R \longmapsto M(\!|N|\!) \prec R'$
by *simp* (*metis statEqTransition AssertionStatEqSym compositionSym*)
moreover from *P''Trans* **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P'' \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence *P''PTrans*: $\Psi \otimes \Psi_R \rhd P'' \parallel !P \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec (P' \parallel !P)$ **using** ‹$xvec \sharp{*} P$›
by(*rule-tac Par1*) *auto*
moreover from *FrP''* **have** *FrP''P*: *extractFrame*$(P'' \parallel !P) = \langle A_P{''}, \Psi_P{''} \otimes \mathbf{1}\rangle$
by *auto*
moreover from *FrQ* ‹$\Psi_Q{'} \simeq \mathbf{1}$› ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_Q{'} \otimes \Psi_R \vdash M \leftrightarrow K$
by *simp* (*metis statEqEnt Composition AssertionStatEqSym Commutativity*)
hence $\Psi \otimes \Psi_Q{'} \otimes \Psi_R \vdash K \leftrightarrow M$ **by**(*rule chanEqSym*)
moreover have $\langle A_Q{'}, (\Psi \otimes \Psi_Q{'}) \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{''}, (\Psi \otimes (\Psi_P{''} \otimes \mathbf{1})) \otimes \Psi_R\rangle$
proof −
have $\langle A_Q{'}, (\Psi \otimes \Psi_Q{'}) \otimes \Psi_R\rangle \simeq_F \langle A_Q{'}, (\Psi \otimes \Psi_R) \otimes \Psi_Q{'}\rangle$

by(*rule-tac frameResChainPres*, *simp*)
(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)
moreover from *NilImpP'' FrQ FrP''* ‹A_P'' ♯* Ψ› ‹A_P'' ♯* Ψ_R› *freshCompChain* **have** $\langle A_Q', (\Psi \otimes \Psi_R) \otimes \Psi_Q' \rangle \hookrightarrow_F \langle A_P'', (\Psi \otimes \Psi_R) \otimes \Psi_P'' \rangle$
by *auto*
moreover have $\langle A_P'', (\Psi \otimes \Psi_R) \otimes \Psi_P'' \rangle \simeq_F \langle A_P'', (\Psi \otimes \Psi_P'' \otimes \mathbf{1}) \otimes \Psi_R \rangle$
by(*rule frameResChainPres*, *simp*)
(*metis Identity AssertionStatEqSym Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
ultimately obtain M' **where** *RTrans*: $\Psi \otimes \Psi_P'' \otimes \mathbf{1} \rhd R \longmapsto M'(\!|N|\!) \prec R'$ **and** $\Psi \otimes (\Psi_P'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$ **and** A_R ♯* M'
using *FrR FrQ'* ‹*distinct* A_R› ‹*distinct* A_P''› ‹A_P'' ♯* A_R› ‹A_R ♯* Ψ› ‹A_R ♯* P''› ‹A_R ♯* Q› ‹A_R ♯* R› ‹A_R ♯* M› ‹A_Q' ♯* R› ‹A_Q' ♯* K› ‹A_Q' ♯* A_R› ‹A_R ♯* P› ‹A_P'' ♯* P› ‹A_R ♯* *xvec*›
‹A_P'' ♯* Ψ› ‹A_P'' ♯* R› ‹A_P'' ♯* P''› ‹A_P'' ♯* K› ‹*xvec* ♯* K› ‹*distinct xvec*›
by(*rule-tac* A_Q=A_Q' **and** Q=Q **in** *comm2Aux*) (*assumption* | *simp*)+

note *RTrans FrR P''PTrans FrP''P*
moreover from ‹$\Psi \otimes (\Psi_P'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$› **have** $\Psi \otimes (\Psi_P'' \otimes \mathbf{1}) \otimes \Psi_R \vdash M' \leftrightarrow K$ **by**(*rule chanEqSym*)
hence $\Psi \otimes \Psi_R \otimes \Psi_P'' \otimes \mathbf{1} \vdash M' \leftrightarrow K$ **by**(*metis statEqEnt Composition AssertionStatEqSym Commutativity*)
ultimately show *?thesis* **using** ‹A_R ♯* Ψ› ‹A_R ♯* R› ‹A_R ♯* P''› ‹A_R ♯* P› ‹A_R ♯* M'› ‹A_P'' ♯* A_R› ‹A_P'' ♯* Ψ› ‹A_P'' ♯* R› ‹A_P'' ♯* P''› ‹A_P'' ♯* P› ‹A_P'' ♯* K› ‹*xvec* ♯* R›
by(*rule-tac Comm1*) (*assumption* | *simp*)+
qed
ultimately show *?thesis*
by(*drule-tac tauActTauStepChain*) *auto*
qed

moreover from *P'Chain* **have** $((\Psi \otimes \Psi_R) \otimes \Psi') \otimes \mathbf{1} \rhd P' \Longrightarrow\hat{}_\tau T$
by(*rule tauChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P' \parallel !P \Longrightarrow\hat{}_\tau T \parallel !P$
by(*rule-tac tauChainPar1*) *auto*
hence $\Psi \otimes \Psi_R \otimes \Psi' \rhd P' \parallel !P \Longrightarrow\hat{}_\tau T \parallel !P$
by(*rule tauChainStatEq*) (*metis Associativity*)
hence $\Psi \otimes \Psi_R' \rhd P' \parallel !P \Longrightarrow\hat{}_\tau T \parallel !P$ **using** ‹$\Psi_R \otimes \Psi' \simeq \Psi_R'$›
by(*rule-tac tauChainStatEq*) (*auto intro*: *compositionSym*)
hence $\Psi \rhd R' \parallel (P' \parallel !P) \Longrightarrow\hat{}_\tau R' \parallel (T \parallel !P)$ **using** *FrR'* ‹A_R' ♯* Ψ› ‹A_R' ♯* P› ‹A_R' ♯* P'›
by(*rule-tac tauChainPar2*) *auto*
hence $\Psi \rhd (\!|\nu{*}xvec|\!)(R' \parallel (P' \parallel !P)) \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)(R' \parallel (T \parallel !P))$ **using** ‹*xvec* ♯* Ψ›

by(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd R \parallel !P \Longrightarrow_\tau (\!|\nu{*}xvec|\!)(R' \parallel (T \parallel !P))$
by *auto*
moreover have $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel (T \parallel !P)), (\!|\nu{*}xvec|\!)(R' \parallel Q')) \in Rel''$
proof −
from ‹$((\Psi \otimes \Psi_R) \otimes \Psi', T, S) \in Rel$› **have** $(\Psi \otimes \Psi_R \otimes \Psi', T, S) \in Rel$
by(*rule StatEq*) (*metis Associativity*)
hence $(\Psi \otimes \Psi_R', T, S) \in Rel$ **using** ‹$\Psi_R \otimes \Psi' \simeq \Psi_R'$›
by(*rule-tac StatEq*) (*auto dest*: *compositionSym*)

with *FrR'* ‹$A_R' \sharp* \Psi$› ‹$A_R' \sharp* S$› ‹$A_R' \sharp* T$› **have** $(\Psi, R' \parallel T, R' \parallel S) \in Rel$
by(*rule-tac FrameParPres*) *auto*
hence $(\Psi, (R' \parallel T) \parallel !P, (R' \parallel S) \parallel !P) \in Rel$ **by**(*rule ParPres*)
hence $(\Psi, (\!|\nu{*}xvec|\!)((R' \parallel T) \parallel !P), (\!|\nu{*}xvec|\!)((R' \parallel S) \parallel !P)) \in Rel$ **using** ‹$xvec \sharp* \Psi$›
by(*rule ResPres*)
hence $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel T) \parallel !P, ((\!|\nu{*}xvec|\!)(R' \parallel S)) \parallel !P) \in Rel$ **using** ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$›
by(*force intro*: *Trans ScopeExt*)
hence $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel (T \parallel !P)), ((\!|\nu{*}xvec|\!)(R' \parallel S)) \parallel !P) \in Rel$ **using** ‹$xvec \sharp* \Psi$›
by(*force intro*: *Trans ResPres Assoc*)

moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded P*› ‹*guarded Q*› **have** $(\Psi, ((\!|\nu{*}xvec|\!)(R' \parallel S)) \parallel !P, ((\!|\nu{*}xvec|\!)(R' \parallel S)) \parallel !Q) \in Rel''$
by(*rule C1*)
moreover from ‹$(\mathbf{1}, Q', S \parallel !Q) \in Rel'$› **have** $(\mathbf{1} \otimes \Psi, Q', S \parallel !Q) \in Rel'$ **by**(*rule cExt'*)
hence $(\Psi, Q', S \parallel !Q) \in Rel'$
by(*rule StatEq'*) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence $(\Psi, R' \parallel Q', R' \parallel (S \parallel !Q)) \in Rel'$ **by**(*rule ParPres'*)
hence $(\Psi, R' \parallel Q', (R' \parallel S) \parallel !Q) \in Rel'$ **by**(*metis Trans' Assoc'*)
hence $(\Psi, (R' \parallel S) \parallel !Q, R' \parallel Q') \in Rel'$ **by**(*rule cSym'*)
hence $(\Psi, (\!|\nu{*}xvec|\!)((R' \parallel S) \parallel !Q), (\!|\nu{*}xvec|\!)(R' \parallel Q')) \in Rel'$ **using** ‹$xvec \sharp* \Psi$›
by(*rule ResPres'*)
hence $(\Psi, ((\!|\nu{*}xvec|\!)(R' \parallel S)) \parallel !Q, (\!|\nu{*}xvec|\!)(R' \parallel Q')) \in Rel'$ **using** ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* Q$›
by(*force intro*: *Trans' ScopeExt'*[*THEN cSym'*])
ultimately show *?thesis* **by**(*rule-tac Compose*)
qed
ultimately show *?case* **by** *blast*
next
case(*cComm2* Ψ_Q *M xvec N* R' A_R Ψ_R *K* Q' A_Q)
from ‹$A_R \sharp* (P, Q, R)$› **have** $A_R \sharp* P$ **and** $A_R \sharp* Q$ **and** $A_R \sharp* R$ **by** *simp*+
from ‹$xvec \sharp* (P, Q, R)$› **have** $xvec \sharp* P$ **and** $xvec \sharp* Q$ **and** $xvec \sharp* R$ **by** *simp*+
have *FrQ*: $extractFrame(!Q) = \langle A_Q, \Psi_Q \rangle$ **by** *fact*

have *FrR*: *extractFrame* $R = \langle A_R, \Psi_R\rangle$ **by** *fact*
from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$› *FrR* ‹*distinct* A_R› ‹$A_R \sharp* R$› ‹$A_R \sharp* N$› ‹$A_R \sharp* xvec$› ‹$A_R \sharp* P$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* \Psi$› ‹$xvec \sharp* R$› ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$› ‹$xvec \sharp* Q$› ‹$xvec \sharp* M$› ‹*distinct xvec*› ‹$A_R \sharp* M$›
obtain p $A_R{}'$ $\Psi_R{}'$ Ψ' **where** *FrR'*: *extractFrame* $R' = \langle A_R{}', \Psi_R{}'\rangle$ **and** $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$ **and** $A_R{}' \sharp* xvec$ **and** $A_R{}' \sharp* P$ **and** $A_R{}' \sharp* Q$ **and** $A_R{}' \sharp* \Psi$ **and** S: *set* $p \subseteq$ *set xvec* $\times$ *set*$(p \cdot xvec)$ **and** *distinctPerm* p **and** $(p \cdot xvec) \sharp* N$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* R'$ **and** $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* \Psi$ **and** $A_R{}' \sharp* N$ **and** $A_R{}' \sharp* xvec$ **and** $A_R{}' \sharp* (p \cdot xvec)$
by(*rule-tac* C=(Ψ, P, Q) **and** C'=(Ψ, P, Q) **in** *expandFrame*) (*assumption* | *simp*)+

from ‹$\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec R'$› S ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* R'$›
have *RTrans*: $\Psi \otimes \Psi_Q \rhd R \longmapsto M(\!|\nu{*}(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$
by(*simp add*: *boundOutputChainAlpha''* *residualInject*)
from ‹$(\Psi, P, Q) \in Rel$› **have** $(\Psi \otimes \Psi_R, P, Q) \in Rel$ **by**(*rule cExt*)
moreover from ‹$\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|N|\!) \prec Q'$› S ‹$(p \cdot xvec) \sharp* Q$› ‹$xvec \sharp* Q$› ‹*distinctPerm* p›
have $\Psi \otimes \Psi_R \rhd !Q \longmapsto K(\!|(p \cdot N)|\!) \prec (p \cdot Q')$ **by**(*rule-tac inputAlpha*) *auto*
then obtain S **where** *QTrans*: $\Psi \otimes \Psi_R \rhd Q \longmapsto K(\!|(p \cdot N)|\!) \prec S$ **and** $(\mathbf{1}, (p \cdot Q'), S \parallel !Q) \in Rel'$
using ‹*guarded* Q›
by(*fastforce dest*: *rBangActE*)
ultimately obtain P' T **where** *PTrans*: $\Psi \otimes \Psi_R : Q \rhd P \Longrightarrow K(\!|(p \cdot N)|\!) \prec P'$
and *P'Chain*: $(\Psi \otimes \Psi_R) \otimes (p \cdot \Psi') \rhd P' \Longrightarrow\hat{}_\tau T$
and $((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'), T, S) \in Rel$
by(*fastforce dest*: *cSim weakSimE*)

from ‹$A_R{}' \sharp* N$› ‹$A_R{}' \sharp* xvec$› ‹$A_R{}' \sharp* (p \cdot xvec)$› S **have** $A_R{}' \sharp* (p \cdot N)$
by(*simp add*: *freshChainSimps*)
with *PTrans* ‹$A_R{}' \sharp* P$› **have** $A_R{}' \sharp* P'$ **by**(*force dest*: *weakInputFreshChainDerivative*)
with *P'Chain* **have** $A_R{}' \sharp* T$ **by**(*force dest*: *tauChainFreshChain*)+
from *QTrans* ‹$A_R{}' \sharp* Q$› ‹$A_R{}' \sharp* (p \cdot N)$› **have** $A_R{}' \sharp* S$ **by**(*force dest*: *inputFreshChainDerivative*)

obtain $A_Q{}'$ $\Psi_Q{}'$ **where** *FrQ'*: *extractFrame* $Q = \langle A_Q{}', \Psi_Q{}'\rangle$ **and** $A_Q{}' \sharp* \Psi$ **and** $A_Q{}' \sharp* \Psi_R$ **and** $A_Q{}' \sharp* A_R$ **and** $A_Q{}' \sharp* M$ **and** $A_Q{}' \sharp* R$ **and** $A_Q{}' \sharp* K$
by(*rule-tac* C=$(\Psi, \Psi_R, A_R, K, M, R)$ **in** *freshFrame*) *auto*
from *FrQ'* ‹*guarded* Q› **have** $\Psi_Q{}' \simeq \mathbf{1}$ **and** *supp* $\Psi_Q{}' = (\{\}::name\ set)$ **by**(*blast dest*: *guardedStatEq*)+
hence $A_Q{}' \sharp* \Psi_Q{}'$ **by**(*auto simp add*: *fresh-star-def fresh-def*)

from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \otimes \Psi_R \rhd P \Longrightarrow\hat{}_\tau P''$
and *NilImpP''*: $\langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') $(\Psi \otimes \Psi_R)$
and *P''Trans*: $\Psi \otimes \Psi_R \rhd P'' \longmapsto K(\!|(p \cdot N)|\!) \prec P'$

using *FrQ'* ‹$A_Q{}'$ ♯* Ψ› ‹$A_Q{}'$ ♯* Ψ_R› *freshCompChain*
by(*drule-tac weakTransitionE*) *auto*

from ‹$(p \cdot xvec)$ ♯* P› ‹$xvec$ ♯* P› *PChain* **have** $(p \cdot xvec)$ ♯* P'' **and** $xvec$ ♯* P''
by(*force dest*: *tauChainFreshChain*)+
from ‹$(p \cdot xvec)$ ♯* N› ‹*distinctPerm p*› **have** $xvec$ ♯* $(p \cdot N)$
by(*subst pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*, **where** *pi=p*, *symmetric*]) *simp*
with *P''Trans* ‹$xvec$ ♯* P''› **have** $xvec$ ♯* P' **by**(*force dest*: *inputFreshChainDerivative*)
hence $(p \cdot xvec)$ ♯* $(p \cdot P')$ **by**(*simp add*: *pt-fresh-star-bij*[*OF pt-name-inst*, *OF at-name-inst*])

from *PChain* **have** $\Psi \rhd R \parallel !P \Longrightarrow_\tau (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel (P' \parallel !P))$
proof(*induct rule*: *tauChainCases*)
case *TauBase*
from *RTrans FrQ* **have** $\Psi \otimes \mathbf{1} \rhd R \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$ **by** *simp*
moreover note *FrR*
moreover from *P''Trans* ‹$P = P''$› **have** $\Psi \otimes \Psi_R \rhd P \longmapsto K(\!|(p \cdot N)|\!) \prec P'$ **by** *simp*
hence $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \longmapsto K(\!|(p \cdot N)|\!) \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \longmapsto K(\!|(p \cdot N)|\!) \prec (P' \parallel !P)$ **by**(*force intro*: *Par1*)
hence $\Psi \otimes \Psi_R \rhd !P \longmapsto K(\!|(p \cdot N)|\!) \prec (P' \parallel !P)$ **using** ‹*guarded P*› **by**(*rule Bang*)
moreover from ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› *FrQ* **have** $\Psi \otimes \Psi_R \otimes \mathbf{1} \vdash M \leftrightarrow K$ **by** *simp*
ultimately have $\Psi \rhd R \parallel !P \longmapsto \tau \prec (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel (P' \parallel !P))$ **using** ‹A_R ♯* Ψ› ‹A_R ♯* R› ‹A_R ♯* P› ‹A_R ♯* M› ‹$(p \cdot xvec)$ ♯* P›
by(*force intro*: *Comm2*)
thus *?case* **by**(*rule tauActTauStepChain*)
next
case *TauStep*
obtain $A_P{}''$ $\Psi_P{}''$ **where** *FrP''*: *extractFrame* $P'' = \langle A_P{}'', \Psi_P{}''\rangle$ **and** $A_P{}''$ ♯* Ψ **and** $A_P{}''$ ♯* K **and** $A_P{}''$ ♯* Ψ_R **and** $A_P{}''$ ♯* R **and** $A_P{}''$ ♯* P'' **and** $A_P{}''$ ♯* P
and $A_P{}''$ ♯* A_R **and** *distinct* $A_P{}''$
by(*rule-tac* $C=(\Psi, K, A_R, \Psi_R, R, P'', P)$ **in** *freshFrame*) *auto*
from *PChain* ‹A_R ♯* P› **have** A_R ♯* P'' **by**(*drule-tac tauChainFreshChain*) *auto*
with *FrP''* ‹$A_P{}''$ ♯* A_R› **have** A_R ♯* $\Psi_P{}''$ **by**(*drule-tac extractFrameFreshChain*) *auto*
from ‹$\Psi \otimes \Psi_R \rhd P \Longrightarrow_\tau P''$› **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P \Longrightarrow_\tau P''$ **by**(*rule tauStepChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \otimes \Psi_R \rhd P \parallel !P \Longrightarrow_\tau P'' \parallel !P$ **by**(*rule-tac tauStepChainPar1*) *auto*
hence $\Psi \otimes \Psi_R \rhd !P \Longrightarrow_\tau P'' \parallel !P$ **using** ‹*guarded P*› **by**(*rule tauStepChainBang*)

hence $\Psi \rhd R \parallel !P \Longrightarrow_\tau R \parallel (P'' \parallel !P)$ **using** *FrR* ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P$›
by(*rule-tac tauStepChainPar2*) *auto*
moreover have $\Psi \rhd R \parallel (P'' \parallel !P) \longmapsto\tau \prec (\!|\nu*(p \cdot xvec)|\!)((p \cdot R') \parallel (P' \parallel !P))$
proof −
from *FrQ* ‹$\Psi_Q{}' \simeq \mathbf{1}$› *RTrans* **have** $\Psi \otimes \Psi_Q{}' \rhd R \longmapsto M(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$
by *simp* (*metis statEqTransition AssertionStatEqSym compositionSym*)
moreover from *P''Trans* **have** $(\Psi \otimes \Psi_R) \otimes \mathbf{1} \rhd P'' \longmapsto K(\!|(p \cdot N)|\!) \prec P'$
by(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence *P''PTrans*: $\Psi \otimes \Psi_R \rhd P'' \parallel !P \longmapsto K(\!|(p \cdot N)|\!) \prec (P' \parallel !P)$
by(*rule-tac Par1*) *auto*
moreover from *FrP''* **have** *FrP''P*: *extractFrame*$(P'' \parallel !P) = \langle A_P{}'', \Psi_P{}'' \otimes \mathbf{1}\rangle$
by *auto*
moreover from *FrQ* ‹$\Psi_Q{}' \simeq \mathbf{1}$› ‹$\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$› **have** $\Psi \otimes \Psi_Q{}' \otimes \Psi_R \vdash M \leftrightarrow K$
by *simp* (*metis statEqEnt Composition AssertionStatEqSym Commutativity*)
hence $\Psi \otimes \Psi_Q{}' \otimes \Psi_R \vdash K \leftrightarrow M$ **by**(*rule chanEqSym*)
moreover have $\langle A_Q{}', (\Psi \otimes \Psi_Q{}') \otimes \Psi_R\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1})) \otimes \Psi_R\rangle$
proof −
have $\langle A_Q{}', (\Psi \otimes \Psi_Q{}') \otimes \Psi_R\rangle \simeq_F \langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle$
by(*rule-tac frameResChainPres*, *simp*)
(*metis Associativity Commutativity Composition AssertionStatEqTrans AssertionStatEqSym*)
moreover from *NilImpP'' FrQ FrP''* ‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* \Psi_R$› *freshCompChain* **have** $\langle A_Q{}', (\Psi \otimes \Psi_R) \otimes \Psi_Q{}'\rangle \hookrightarrow_F \langle A_P{}'', (\Psi \otimes \Psi_R) \otimes \Psi_P{}''\rangle$
by *auto*
moreover have $\langle A_P{}'', (\Psi \otimes \Psi_R) \otimes \Psi_P{}''\rangle \simeq_F \langle A_P{}'', (\Psi \otimes \Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R\rangle$
by(*rule frameResChainPres*, *simp*)
(*metis Identity AssertionStatEqSym Associativity Commutativity Composition AssertionStatEqTrans*)
ultimately show *?thesis* **by**(*rule FrameStatEqImpCompose*)
qed
ultimately obtain M' **where** *RTrans*: $\Psi \otimes \Psi_P{}'' \otimes \mathbf{1} \rhd R \longmapsto M'(\!|\nu*(p \cdot xvec)|\!)\langle(p \cdot N)\rangle \prec (p \cdot R')$ **and** $\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$ **and** $A_R \sharp* M'$
using *FrR FrQ'* ‹*distinct* A_R› ‹*distinct* $A_P{}''$› ‹$A_P{}'' \sharp* A_R$› ‹$A_R \sharp* \Psi$› ‹$A_R \sharp* P''$› ‹$A_R \sharp* Q$› ‹$A_R \sharp* R$› ‹$A_R \sharp* M$› ‹$A_Q{}' \sharp* R$› ‹$A_Q{}' \sharp* K$› ‹$A_Q{}' \sharp* A_R$› ‹$A_R \sharp* P$› ‹$A_P{}'' \sharp* P$›
‹$A_P{}'' \sharp* \Psi$› ‹$A_P{}'' \sharp* R$› ‹$A_P{}'' \sharp* P''$› ‹$A_P{}'' \sharp* K$›
by(*rule-tac* $A_Q{=}A_Q{}'$ **and** $Q{=}Q$ **in** *comm1Aux*) (*assumption* | *simp*)+

note *RTrans FrR P''PTrans FrP''P*
moreover from ‹$\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash K \leftrightarrow M'$› **have** $\Psi \otimes (\Psi_P{}'' \otimes \mathbf{1}) \otimes \Psi_R \vdash M' \leftrightarrow K$ **by**(*rule chanEqSym*)
hence $\Psi \otimes \Psi_R \otimes \Psi_P{}'' \otimes \mathbf{1} \vdash M' \leftrightarrow K$ **by**(*metis statEqEnt Composition AssertionStatEqSym Commutativity*)

ultimately show *?thesis* **using** ‹A_R ♯* Ψ› ‹A_R ♯* R› ‹A_R ♯* P''› ‹A_R ♯* P› ‹A_R ♯* M'› ‹A_P'' ♯* A_R› ‹A_P'' ♯* Ψ› ‹A_P'' ♯* R› ‹A_P'' ♯* P''› ‹A_P'' ♯* P› ‹A_P'' ♯* K› ‹$(p \cdot xvec)$ ♯* P''› ‹$(p \cdot xvec)$ ♯* P›
by(*rule-tac Comm2*) (*assumption* | *simp*)+
qed
ultimately show *?thesis*
by(*drule-tac tauActTauStepChain*) *auto*
qed
hence $\Psi \rhd R \parallel !P \Longrightarrow_\tau (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot P') \parallel !P))$
using ‹$xvec$ ♯* P› ‹$(p \cdot xvec)$ ♯* P› ‹$(p \cdot xvec)$ ♯* $(p \cdot P')$› ‹$(p \cdot xvec)$ ♯* R'› S ‹*distinctPerm p*›
by(*subst resChainAlpha*[**where** *p*=*p*]) *auto*
moreover from *P'Chain* **have** $(p \cdot ((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'))) \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot T)$
by(*rule tauChainEqvt*)
with ‹$xvec$ ♯* Ψ› ‹$(p \cdot xvec)$ ♯* Ψ› S ‹*distinctPerm p*›
have $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot T)$ **by**(*simp add*: *eqvts*)
hence $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi') \otimes \mathbf{1} \rhd (p \cdot P') \Longrightarrow\hat{}_\tau (p \cdot T)$
by(*rule tauChainStatEq*) (*metis Identity AssertionStatEqSym*)
hence $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd (p \cdot P') \parallel !P \Longrightarrow\hat{}_\tau (p \cdot T) \parallel !P$
by(*rule-tac tauChainPar1*) *auto*
hence $\Psi \otimes (p \cdot \Psi_R) \otimes \Psi' \rhd (p \cdot P') \parallel !P \Longrightarrow\hat{}_\tau (p \cdot T) \parallel !P$
by(*rule tauChainStatEq*) (*metis Associativity*)
hence $\Psi \otimes \Psi_R' \rhd (p \cdot P') \parallel !P \Longrightarrow\hat{}_\tau (p \cdot T) \parallel !P$ **using** ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R'$›
by(*rule-tac tauChainStatEq*) (*auto intro*: *compositionSym*)
hence $\Psi \rhd R' \parallel ((p \cdot P') \parallel !P) \Longrightarrow\hat{}_\tau R' \parallel ((p \cdot T) \parallel !P)$
using *FrR'* ‹A_R' ♯* Ψ› ‹A_R' ♯* P› ‹A_R' ♯* P'› ‹A_R' ♯* $xvec$› ‹A_R' ♯* $(p \cdot xvec)$› S
by(*rule-tac tauChainPar2*) (*auto simp add*: *freshChainSimps*)
hence $\Psi \rhd (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot P') \parallel !P)) \Longrightarrow\hat{}_\tau (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot T) \parallel !P))$ **using** ‹$xvec$ ♯* Ψ›
by(*rule tauChainResChainPres*)
ultimately have $\Psi \rhd R \parallel !P \Longrightarrow_\tau (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot T) \parallel !P))$
by *auto*
moreover have $(\Psi, (\!|\nu{*}xvec|\!)(R' \parallel ((p \cdot T) \parallel !P)), (\!|\nu{*}xvec|\!)(R' \parallel Q')) \in Rel''$
proof −
from ‹$((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi'), T, S) \in Rel$›
have $(p \cdot ((\Psi \otimes \Psi_R) \otimes (p \cdot \Psi')), (p \cdot T), (p \cdot S)) \in Rel$
by(*rule Closed*)
with ‹$xvec$ ♯* Ψ› ‹$(p \cdot xvec)$ ♯* Ψ› ‹*distinctPerm p*› S
have $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', p \cdot T, p \cdot S) \in Rel$
by(*simp add*: *eqvts*)
hence $(\Psi \otimes (p \cdot \Psi_R) \otimes \Psi', p \cdot T, p \cdot S) \in Rel$
by(*rule StatEq*) (*metis Associativity*)
hence $(\Psi \otimes \Psi_R', p \cdot T, p \cdot S) \in Rel$ **using** ‹$(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R'$›
by(*rule-tac StatEq*) (*auto dest*: *compositionSym*)
moreover from ‹A_R' ♯* S› ‹A_R' ♯* T› ‹A_R' ♯* $xvec$› ‹A_R' ♯* $(p \cdot xvec)$› S
have A_R' ♯* $(p \cdot S)$ **and** A_R' ♯* $(p \cdot T)$

by(*simp add*: *freshChainSimps*)+
ultimately have $(\Psi, R' \parallel (p \cdot T), R' \parallel (p \cdot S)) \in Rel$ **using** FrR' ‹$A_R{}' \sharp* \Psi$›
by(*rule-tac FrameParPres*) *auto*
hence $(\Psi, (R' \parallel (p \cdot T)) \parallel !P, (R' \parallel (p \cdot S)) \parallel !P) \in Rel$ **by**(*rule ParPres*)
hence $(\Psi, (\!|\nu*xvec|\!)((R' \parallel (p \cdot T)) \parallel !P), (\!|\nu*xvec|\!)((R' \parallel (p \cdot S)) \parallel !P)) \in Rel$
using ‹$xvec \sharp* \Psi$›
by(*rule ResPres*)
hence $(\Psi, (\!|\nu*xvec|\!)(R' \parallel (p \cdot T)) \parallel !P, ((\!|\nu*xvec|\!)(R' \parallel (p \cdot S))) \parallel !P) \in Rel$
using ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* P$›
by(*force intro*: *Trans ScopeExt*)
hence $(\Psi, (\!|\nu*xvec|\!)(R' \parallel ((p \cdot T) \parallel !P)), ((\!|\nu*xvec|\!)(R' \parallel (p \cdot S))) \parallel !P) \in Rel$
using ‹$xvec \sharp* \Psi$›
by(*force intro*: *Trans ResPres Assoc*)
moreover from ‹$(\Psi, P, Q) \in Rel$› ‹*guarded* P› ‹*guarded* Q›
have $(\Psi, ((\!|\nu*xvec|\!)(R' \parallel (p \cdot S))) \parallel !P, ((\!|\nu*xvec|\!)(R' \parallel (p \cdot S))) \parallel !Q) \in Rel''$
by(*rule C1*)
moreover from ‹$(\mathbf{1}, (p \cdot Q'), S \parallel !Q) \in Rel'$›
have $(p \cdot \mathbf{1}, p \cdot p \cdot Q', p \cdot (S \parallel !Q)) \in Rel'$ **by**(*rule Closed'*)
with ‹$xvec \sharp* Q$› ‹$(p \cdot xvec) \sharp* Q$› S ‹*distinctPerm* p›
have $(\mathbf{1}, Q', (p \cdot S) \parallel !Q) \in Rel'$ **by**(*simp add*: *eqvts*)
hence $(\mathbf{1} \otimes \Psi, Q', (p \cdot S) \parallel !Q) \in Rel'$ **by**(*rule cExt'*)
hence $(\Psi, Q', (p \cdot S) \parallel !Q) \in Rel'$
by(*rule StatEq'*) (*metis Identity AssertionStatEqSym Commutativity AssertionStatEqTrans*)
hence $(\Psi, R' \parallel Q', R' \parallel ((p \cdot S) \parallel !Q)) \in Rel'$ **by**(*rule ParPres'*)
hence $(\Psi, R' \parallel Q', (R' \parallel (p \cdot S)) \parallel !Q) \in Rel'$ **by**(*metis Trans' Assoc'*)
hence $(\Psi, (R' \parallel (p \cdot S)) \parallel !Q, R' \parallel Q') \in Rel'$ **by**(*rule cSym'*)
hence $(\Psi, (\!|\nu*xvec|\!)((R' \parallel (p \cdot S)) \parallel !Q), (\!|\nu*xvec|\!)(R' \parallel Q')) \in Rel'$ **using** ‹$xvec \sharp* \Psi$›
by(*rule ResPres'*)
hence $(\Psi, ((\!|\nu*xvec|\!)(R' \parallel (p \cdot S))) \parallel !Q, (\!|\nu*xvec|\!)(R' \parallel Q')) \in Rel'$ **using** ‹$xvec \sharp* \Psi$› ‹$xvec \sharp* Q$›
by(*force intro*: *Trans' ScopeExt'*[*THEN cSym'*])
ultimately show *?thesis* **by**(*rule-tac Compose*)
qed
ultimately show *?case* **by** *blast*
qed
qed
unbundle *relcomp-syntax*

end

end

theory *Weak-Cong-Pres*
imports *Weak-Psi-Congruence Weak-Cong-Sim-Pres Weak-Bisim-Pres*

begin

context *env* **begin**

lemma *weakPsiCongInputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and *xvec* :: *name list*
  and N :: $'a$

 assumes $\bigwedge$*Tvec*. *length xvec* = *length Tvec* $\Longrightarrow$ $\Psi \rhd P[xvec::=Tvec] \approx Q[xvec::=Tvec]$

  shows $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \doteq M(\!|\lambda{*}xvec\ N|\!).Q$
proof −
  from *assms* **have** $\forall$ *Tvec*. *length xvec* = *length Tvec* $\longrightarrow$ $\Psi \rhd P[xvec::=Tvec] \approx Q[xvec::=Tvec]$
    by *simp*
  thus *?thesis*
  proof(*induct rule*: *weakPsiCongSymI*)
    case(*cSym P Q*)
    thus *?case* **by**(*auto dest*: *weakBisimE*)
  next
    case(*cSim P Q*)
    show *?case*
      by(*induct rule*: *weakCongSimI*) *auto*
  next
    case(*cWeakBisim P Q*)
    thus *?case* **by**(*rule-tac weakBisimInputPres*) *auto*
  qed
qed

lemma *weakPsiCongOutputPres*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*
  and M :: $'a$
  and *xvec* :: *name list*
  and N :: $'a$

  assumes $\Psi \rhd P \doteq Q$

  shows $\Psi \rhd M\langle N\rangle.P \doteq M\langle N\rangle.Q$
using *assms*
proof(*induct rule*: *weakPsiCongSymI*)
  case(*cSym P Q*)
  thus *?case* **by**(*rule weakPsiCongSym*)
next

case(*cSim P Q*)
show *?case* **by**(*induct rule*: *weakCongSimI*) *auto*
next
case(*cWeakBisim P Q*)
thus *?case* **by**(*metis weakPsiCongE weakBisimOutputPres*)
qed

lemma *weakBisimCasePres*:
fixes Ψ :: *'b*
and *CsP* :: (*'c* × (*'a*, *'b*, *'c*) *psi*) *list*
and *CsQ* :: (*'c* × (*'a*, *'b*, *'c*) *psi*) *list*

assumes *A*: $\bigwedge\varphi$ *P*. (φ, *P*) *mem CsP* $\Longrightarrow$ $\exists$ *Q*. (φ, *Q*) *mem CsQ* $\wedge$ *guarded Q* $\wedge$ ($\forall\Psi$. $\Psi \rhd P \doteq Q$)
and *B*: $\bigwedge\varphi$ *Q*. (φ, *Q*) *mem CsQ* $\Longrightarrow$ $\exists$ *P*. (φ, *P*) *mem CsP* $\wedge$ *guarded P* $\wedge$ ($\forall\Psi$. $\Psi \rhd P \doteq Q$)

shows $\Psi \rhd$ *Cases CsP* $\approx$ *Cases CsQ*
proof −
let *?X* = {(Ψ, *Cases CsP*, *Cases CsQ*) | Ψ *CsP CsQ*. ($\forall\varphi$ *P*. (φ, *P*) *mem CsP* $\longrightarrow$ ($\exists$ *Q*. (φ, *Q*) *mem CsQ* $\wedge$ *guarded Q* $\wedge$ ($\forall\Psi$. $\Psi \rhd P \doteq Q$))) $\wedge$
($\forall\varphi$ *Q*. (φ, *Q*) *mem CsQ* $\longrightarrow$ ($\exists$ *P*. (φ, *P*) *mem CsP* $\wedge$ *guarded P* $\wedge$ ($\forall\Psi$. $\Psi \rhd P \doteq Q$)))}
from *assms* **have** (Ψ, *Cases CsP*, *Cases CsQ*) $\in$ *?X* **by** *auto*
thus *?thesis*
proof(*coinduct rule*: *weakBisimCoinduct*)
case(*cStatImp* Ψ *CasesP CasesQ*)
thus *?case*
apply(*auto simp add*: *weakStatImp-def*)
by(*rule-tac x=Cases CsQ* **in** *exI*, *auto*)
next
case(*cSim* Ψ *CasesP CasesQ*)
then obtain *CsP CsQ* **where** *C1*: $\bigwedge\varphi$ *Q*. (φ, *Q*) *mem CsQ* $\Longrightarrow$ $\exists$ *P*. (φ, *P*) *mem CsP* $\wedge$ *guarded P* $\wedge$ ($\forall\Psi$. $\Psi \rhd P \doteq Q$)
and *A*: *CasesP* = *Cases CsP* **and** *B*: *CasesQ* = *Cases CsQ*
by *auto*
note *C1*
moreover have $\bigwedge\Psi$ *P Q*. $\Psi \rhd P \approx Q \Longrightarrow \Psi \rhd P \rightsquigarrow$<*weakBisim*> *Q* **by**(*rule weakBisimE*)
moreover note *weakPsiCongE*(*1*) *weakPsiCongE*(*2*)
ultimately have $\Psi \rhd$ *Cases CsP* $\rightsquigarrow$<*weakBisim*> *Cases CsQ*
by(*rule-tac caseWeakSimPres*) (*assumption* | *blast*)+
hence $\Psi \rhd$ *Cases CsP* $\rightsquigarrow$<(*?X* $\cup$ *weakBisim*)> *Cases CsQ*
by(*rule-tac weakSimMonotonic*) *auto*
with *A B* **show** *?case* **by** *blast*
next
case(*cExt* Ψ *P Q* Ψ')
thus *?case* **by** *auto*
next

case(*cSym* Ψ *P Q*)
thus *?case* **by** *auto* (*metis weakPsiCongSym*)+
qed
qed

lemma *weakPsiCongCasePres*:
fixes Ψ :: *'b*
and *CsP* :: (*'c* × (*'a*, *'b*, *'c*) *psi*) *list*
and *CsQ* :: (*'c* × (*'a*, *'b*, *'c*) *psi*) *list*

assumes *A*: $\bigwedge\varphi$ *P*. (φ, *P*) *mem CsP* $\Longrightarrow$ $\exists$ *Q*. (φ, *Q*) *mem CsQ* $\wedge$ *guarded Q* $\wedge$ ($\forall\,\Psi$. $\Psi \rhd P \doteq Q$)
and *B*: $\bigwedge\varphi$ *Q*. (φ, *Q*) *mem CsQ* $\Longrightarrow$ $\exists$ *P*. (φ, *P*) *mem CsP* $\wedge$ *guarded P* $\wedge$ ($\forall\,\Psi$. $\Psi \rhd P \doteq Q$)

shows $\Psi \rhd$ *Cases CsP* $\doteq$ *Cases CsQ*
proof −
let *?Prop* = λ*CsP CsQ*. $\forall\,\varphi$ *P*. (φ, *P*) *mem CsP* $\longrightarrow$ ($\exists$ *Q*. (φ, *Q*) *mem CsQ* $\wedge$ *guarded Q* $\wedge$ ($\forall\,\Psi$. $\Psi \rhd P \doteq Q$))
from *A B* **have** *?Prop CsP CsQ* $\wedge$ *?Prop CsQ CsP* **by**(*metis weakPsiCongSym*)
thus *?thesis*
proof(*induct rule*: *weakPsiCongSymI*[**where** *C*=λ*P*. *Cases P*])
case(*cSym P Q*)
thus *?case* **by** *auto*
next
case(*cWeakBisim CsP CsQ*)
thus *?case*
by(*rule-tac weakBisimCasePres*) (*metis weakPsiCongSym*)+
next
case(*cSim CsP CsQ*)
thus *?case*
apply *auto*
apply(*rule-tac weakCongSimCasePres*, *auto*)
by(*blast dest*: *weakPsiCongE*)
qed
qed

lemma *weakPsiCongResPres*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and *x* :: *name*

assumes $\Psi \rhd P \doteq Q$
and $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)P \doteq (\!|\nu x|\!)Q$
using ‹$\Psi \rhd P \doteq Q$›
proof(*induct rule*: *weakPsiCongSymI*)

case(*cSym P Q*)
thus *?case* **by**(*rule weakPsiCongSym*)
next
case(*cWeakBisim P Q*)
thus *?case* **using** ‹$x \sharp \Psi$› **by**(*metis weakPsiCongE weakBisimResPres*)
next
case(*cSim P Q*)
obtain *y*::*name* **where** $y \sharp \Psi$ **and** $y \sharp P$ **and** $y \sharp Q$
by(*generate-fresh name*) *auto*
from ‹$\Psi \rhd P \doteq Q$› **have** $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \doteq ([(x, y)] \cdot Q)$ **by**(*rule weakPsiCongClosed*)
hence $([(x, y)] \cdot \Psi) \rhd ([(x, y)] \cdot P) \rightsquigarrow$«*weakBisim*» $([(x, y)] \cdot Q)$ **by**(*rule weakPsiCongE*)
with ‹$x \sharp \Psi$› ‹$y \sharp \Psi$› **have** $\Psi \rhd ([(x, y)] \cdot P) \rightsquigarrow$«*weakBisim*» $([(x, y)] \cdot Q)$ **by** *simp*
moreover have *eqvt weakBisim* **by** *simp*
moreover note ‹$y \sharp \Psi$›
moreover have $weakBisim \subseteq weakBisim$ **by** *auto*
moreover note *weakBisimResPres*
ultimately have $\Psi \rhd (\!|\nu y|\!)([(x, y)] \cdot P) \rightsquigarrow$«*weakBisim*» $(\!|\nu y|\!)([(x, y)] \cdot Q)$ **by**(*rule weakCongSimResPres*)
with ‹$y \sharp P$› ‹$y \sharp Q$› **show** *?case* **by**(*simp add*: *alphaRes*)
qed

lemma *weakPsiCongResChainPres*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and *xvec* :: *name list*

assumes $\Psi \rhd P \doteq Q$
and $xvec \sharp* \Psi$

shows $\Psi \rhd (\!|\nu*xvec|\!)P \doteq (\!|\nu*xvec|\!)Q$
using *assms*
by(*induct xvec*) (*auto intro*: *weakPsiCongResPres*)

lemma *weakPsiCongParPres*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and R :: (*'a*, *'b*, *'c*) *psi*

assumes $\forall \Psi.\ \Psi \rhd P \doteq Q$

shows $\Psi \rhd P \parallel R \doteq Q \parallel R$
using *assms*
proof(*induct rule*: *weakPsiCongSymI*[**where** $C{=}\lambda P.\ P \parallel R$])
case(*cSym P Q*)

thus *?case* **by**(*blast dest*: *weakPsiCongSym*)
next
case(*cWeakBisim P Q*)
thus *?case* **by**(*metis weakBisimParPres weakPsiCongE*)
next
case(*cSim P Q*)
{
fix Ψ
from ‹$\forall \Psi.\ \Psi \rhd P \doteq Q$› **have** $\Psi \rhd P \rightsquigarrow$«*weakBisim*» Q **by**(*auto dest*: *weakPsiCongE*)
}
moreover {
fix Ψ
from ‹$\forall \Psi.\ \Psi \rhd P \doteq Q$› **have** $\Psi \rhd P \rightsquigarrow$<*weakBisim*> Q **by**(*auto dest*: *weakPsiCongE weakBisimE*)
}
moreover {
fix Ψ
from ‹$\forall \Psi.\ \Psi \rhd P \doteq Q$› **have** $\Psi \rhd P \approx Q$ **by**(*auto dest*: *weakPsiCongE*)
hence $\Psi \rhd Q \lessapprox$<*weakBisim*> P **by**(*metis weakBisimE*)
}
ultimately show *?case* **using** *weakBisimEqvt weakBisimEqvt weakBisimE*(*4*) *weakBisimE*(*3*) *weakBisimParPresAux weakBisimResChainPres statEqWeakBisim*
by(*rule-tac weakCongSimParPres*)
qed

end

end

theory *Weak-Cong-Struct-Cong*
imports *Weak-Cong-Pres Weak-Bisim-Struct-Cong*
begin

context *env* **begin**

lemma *weakPsiCongParComm*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd P \parallel Q \doteq Q \parallel P$
by(*metis bisimParComm strongBisimWeakPsiCong*)

lemma *weakPsiCongResComm*:
fixes x :: *name*
and Ψ :: *'b*
and y :: *name*
and P :: (*'a*, *'b*, *'c*) *psi*

assumes $x \sharp \Psi$
and $y \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu y|\!)P) \doteq (\!|\nu y|\!)((\!|\nu x|\!)P)$
using *assms*
by(*metis bisimResComm strongBisimWeakPsiCong*)

lemma *weakPsiCongResComm'*:
fixes x :: *name*
and Ψ :: $'b$
and *xvec* :: *name list*
and P :: $('a, 'b, 'c)$ *psi*

assumes $x \sharp \Psi$
and $xvec \sharp* \Psi$

shows $\Psi \rhd (\!|\nu x|\!)((\!|\nu{*}xvec|\!)P) \doteq (\!|\nu{*}xvec|\!)((\!|\nu x|\!)P)$
using *assms*
by(*metis bisimResComm' strongBisimWeakPsiCong*)

lemma *weakPsiCongScopeExt*:
fixes x :: *name*
and Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp P$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \doteq P \parallel (\!|\nu x|\!)Q$
using *assms*
by(*metis bisimScopeExt strongBisimWeakPsiCong*)

lemma *weakPsiCongScopeExtChain*:
fixes *xvec* :: *name list*
and Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $xvec \sharp* \Psi$
and $xvec \sharp* P$

shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \doteq P \parallel ((\!|\nu{*}xvec|\!)Q)$
using *assms*
by(*metis bisimScopeExtChain strongBisimWeakPsiCong*)

lemma *weakPsiCongParAssoc*:
fixes Ψ :: $'b$

and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and R :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd (P \parallel Q) \parallel R \doteq P \parallel (Q \parallel R)$
by(*metis bisimParAssoc strongBisimWeakPsiCong*)

lemma *weakPsiCongParNil*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd P \parallel \mathbf{0} \doteq P$
by(*metis bisimParNil strongBisimWeakPsiCong*)

lemma *weakPsiCongResNil*:
fixes x :: *name*
and Ψ :: $'b$

assumes $x \sharp \Psi$

shows $\Psi \rhd (\!|\nu x|\!)\mathbf{0} \doteq \mathbf{0}$
using *assms*
by(*metis bisimResNil strongBisimWeakPsiCong*)

lemma *weakPsiCongOutputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp M$
and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M\langle N\rangle.P) \doteq M\langle N\rangle.(\!|\nu x|\!)P$
using *assms*
by(*metis bisimOutputPushRes strongBisimWeakPsiCong*)

lemma *weakPsiCongInputPushRes*:
fixes x :: *name*
and Ψ :: $'b$
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$
and P :: $('a,\ 'b,\ 'c)$ *psi*

assumes $x \sharp \Psi$
and $x \sharp M$
and $x \sharp$ *xvec*

and $x \sharp N$

shows $\Psi \rhd (\!|\nu x|\!)(M(\!|\lambda{*}xvec\ N|\!).P) \doteq M(\!|\lambda{*}xvec\ N|\!).(\!|\nu x|\!)P$
using *assms*
by(*metis bisimInputPushRes strongBisimWeakPsiCong*)

lemma *weakPsiCongCasePushRes*:
fixes x :: *name*
and Ψ :: $'b$
and Cs :: $('c \times ('a,\ 'b,\ 'c)\ psi)\ list$

assumes $x \sharp \Psi$
and $x \sharp (map\ fst\ Cs)$

shows $\Psi \rhd (\!|\nu x|\!)(Cases\ Cs) \doteq Cases(map\ (\lambda(\varphi,\ P).\ (\varphi,\ (\!|\nu x|\!)P))\ Cs)$
using *assms*
by(*metis bisimCasePushRes strongBisimWeakPsiCong*)

lemma *weakBangExt*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$

assumes *guarded P*

shows $\Psi \rhd !P \doteq P \parallel !P$
using *assms*
by(*metis bangExt strongBisimWeakPsiCong*)

lemma *weakPsiCongParSym*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)\ psi$
and Q :: $('a,\ 'b,\ 'c)\ psi$
and R :: $('a,\ 'b,\ 'c)\ psi$

assumes $\forall \Psi.\ \Psi \rhd P \doteq Q$

shows $\Psi \rhd R \parallel P \doteq R \parallel Q$
using *assms*
by(*metis weakPsiCongParComm weakPsiCongParPres weakPsiCongTransitive*)

lemma *weakPsiCongScopeExtSym*:
fixes x :: *name*
and Q :: $('a,\ 'b,\ 'c)\ psi$
and P :: $('a,\ 'b,\ 'c)\ psi$

assumes $x \sharp \Psi$
and $x \sharp Q$

shows $\Psi \rhd (\!|\nu x|\!)(P \parallel Q) \doteq ((\!|\nu x|\!)P) \parallel Q$

using *assms*
by(*metis weakPsiCongScopeExt weakPsiCongTransitive weakPsiCongParComm weakPsiCongE weakPsiCongResPres*)

lemma *weakPsiCongScopeExtChainSym*:
  fixes *xvec* :: *name list*
  and Q :: $('a, 'b, 'c)$ *psi*
  and P :: $('a, 'b, 'c)$ *psi*

  assumes *xvec* $\sharp*$ Ψ
  and *xvec* $\sharp*$ Q

  shows $\Psi \rhd (\!|\nu{*}xvec|\!)(P \parallel Q) \doteq ((\!|\nu{*}xvec|\!)P) \parallel Q$
using *assms*
by(*induct xvec*) (*auto intro*: *weakPsiCongScopeExtSym weakPsiCongReflexive weakPsiCongTransitive weakPsiCongResPres*)

lemma *weakPsiCongParPresSym*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and Q :: $('a, 'b, 'c)$ *psi*
  and R :: $('a, 'b, 'c)$ *psi*

  assumes $\bigwedge\Psi.\ \Psi \rhd P \doteq Q$

  shows $\Psi \rhd R \parallel P \doteq R \parallel Q$
using *assms*
by(*metis weakPsiCongParComm weakPsiCongParPres weakPsiCongTransitive*)

lemma *tauCongChainBangI*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*
  and P' :: $('a, 'b, 'c)$ *psi*

  assumes $\Psi \rhd P \parallel P \Longrightarrow_\tau P'$
  and *guarded* P

  obtains Q **where** $\Psi \rhd !P \Longrightarrow_\tau Q$ **and** $\Psi \rhd Q \sim P' \parallel !P$
proof −
  assume $\bigwedge Q.\ [\![\Psi \rhd !P \Longrightarrow_\tau Q;\ \Psi \rhd Q \sim P' \parallel !P]\!] \Longrightarrow$ *thesis*
  moreover from ‹$\Psi \rhd P \parallel P \Longrightarrow_\tau P'$› **have** $\exists\, Q.\ \Psi \rhd !P \Longrightarrow_\tau Q \wedge \Psi \rhd Q \sim P' \parallel !P$
  proof(*induct x1*==$P \parallel P\ P'$ *rule*: *tauStepChainInduct*)
    case(*TauBase* R')
    from ‹$\Psi \rhd P \parallel P \longmapsto\tau \prec R'$›
    obtain Q **where** $\Psi \rhd !P \longmapsto\tau \prec Q$ **and** $Q \sim R' \parallel !P$ **using** ‹*guarded* P›
      by(*rule tauBangI*)
    from ‹$\Psi \rhd !P \longmapsto\tau \prec Q$› **have** $\Psi \rhd !P \Longrightarrow_\tau Q$ **by** *auto*
    moreover from ‹$Q \sim R' \parallel !P$› **have** $\Psi \rhd Q \sim R' \parallel !P$

apply(*drule-tac bisimE*(*3*)[**where** Ψ'=Ψ])
by(*rule-tac statEqBisim*, *assumption*) (*metis Identity AssertionStatEqSym AssertionStatEqTrans Commutativity*)
ultimately show *?case* **by** *blast*
next
case(*TauStep* R' R'')
then obtain Q **where** *PChain*: $\Psi \rhd !P \Longrightarrow_\tau Q$ **and** $\Psi \rhd Q \sim R' \parallel !P$ **by** *auto*
from ‹$\Psi \rhd R' \longmapsto\tau \prec R''$› **have** $\Psi \otimes \mathbf{1} \rhd R' \longmapsto\tau \prec R''$ **by**(*rule statEqTransition*) (*metis Identity AssertionStatEqSym*)
hence $\Psi \rhd R' \parallel !P \longmapsto\tau \prec R'' \parallel !P$ **by**(*rule-tac Par1*) *auto*
with ‹$\Psi \rhd Q \sim R' \parallel !P$› **obtain** Q' **where** *QTrans*: $\Psi \rhd Q \longmapsto\tau \prec Q'$ **and** $\Psi \rhd Q' \sim R'' \parallel !P$
by(*force dest*: *bisimE*(*2*) *simE*)
from *PChain QTrans* **have** $\Psi \rhd !P \Longrightarrow_\tau Q'$ **by**(*auto dest*: *tauActTauStepChain*)
thus *?case* **using** ‹$\Psi \rhd Q' \sim R'' \parallel !P$› **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *weakPsiCongBangPres*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*
and Q :: ($'a$, $'b$, $'c$) *psi*

assumes *PeqQ*: $\forall \Psi.\ \Psi \rhd P \doteq Q$
and *guarded P*
and *guarded Q*

shows $\Psi \rhd !P \doteq !Q$
proof −
from *assms* **have** $(\forall \Psi.\ \Psi \rhd P \doteq Q) \wedge$ *guarded P* $\wedge$ *guarded Q* **by** *auto*
hence $\Psi \rhd \mathbf{0} \parallel !P \doteq \mathbf{0} \parallel !Q$
proof(*induct rule*: *weakPsiCongSymI*[**where** C=$\lambda P.\ \mathbf{0} \parallel !P$])
case(*cSym P Q*)
thus *?case* **by**(*auto dest*: *weakPsiCongSym*)
next
case(*cWeakBisim P Q*)
thus *?case* **by**(*metis weakPsiCongE weakBisimBangPresAux*)
next
case(*cSim P Q*)
then have $\forall \Psi.\ \Psi \rhd P \doteq Q$ **and** *guarded P* **and** *guarded Q* **by** *auto*
moreover hence $\Psi \rhd P \approx Q$ **by**(*metis weakPsiCongE weakBisimE*)
moreover have $\bigwedge \Psi\ P\ Q.\ (\forall \Psi.\ (\Psi \rhd P \doteq Q)) \Longrightarrow \Psi \rhd P \rightsquigarrow$«*weakBisim*» Q
by(*blast dest*: *weakPsiCongE*)
moreover note *weakBisimClosed bisimClosed weakBisimE*(*3*) *bisimE*(*3*) *weakBisimE*(*2*)
weakBisimE(*4*) *bisimE*(*4*) *statEqWeakBisim statEqBisim weakBisimTransitive bisimTransitive weakBisimParAssoc*[*THEN weakBisimE*(*4*)]

bisimParAssoc[*THEN bisimE*(*4*)] *weakBisimParPres*
moreover have $\bigwedge P\ Q.\ \forall \Psi.\ \Psi \rhd P \doteq Q \Longrightarrow \forall \Psi.\ \Psi \rhd P \parallel P \doteq Q \parallel Q$
by(*metis weakPsiCongParPres weakPsiCongParComm weakPsiCongSym weakPsiCongTransitive*)
moreover note *bisimParPresSym*
moreover from *strongBisimWeakBisim* **have** $bisim \subseteq weakBisim$ **by** *auto*
moreover have $\bigwedge \Psi\ \Psi_R\ P\ Q\ R\ A_R.\ [\![\Psi \otimes \Psi_R \rhd P \approx Q;\ extractFrame\ R = \langle A_R,\ \Psi_R\rangle;\ A_R \sharp* \Psi;\ A_R \sharp* P;\ A_R \sharp* Q]\!] \Longrightarrow \Psi \rhd R \parallel P \approx R \parallel Q$
by(*metis weakBisimParComm weakBisimTransitive weakBisimParPresAux*)
moreover note *weakBisimResChainPres bisimResChainPres weakBisimScopeExtChainSym bisimScopeExtChainSym*
moreover have $\bigwedge \Psi\ P\ R\ S\ Q.\ [\![\Psi \rhd P \approx R;\ \Psi \rhd R \approx S;\ \Psi \rhd S \sim Q]\!] \Longrightarrow \Psi \rhd P \approx Q$
by(*blast dest*: *weakBisimTransitive strongBisimWeakBisim*)
moreover note *weakBisimBangPresAux*
moreover from *bangActE* **have** $\bigwedge \Psi\ P\ \alpha\ P'.\ [\![\Psi \rhd !P \longmapsto \alpha \prec P';\ bn\ \alpha \sharp* P;\ guarded\ P;\ \alpha \neq \tau;\ bn\ \alpha \sharp* subject\ \alpha]\!] \Longrightarrow \exists\, Q.\ \Psi \rhd P \longmapsto \alpha \prec Q \wedge P' \sim Q \parallel !P$
by *blast*
moreover from *bangTauE* **have** $\bigwedge \Psi\ P\ P'.\ [\![\Psi \rhd !P \longmapsto \tau \prec P';\ guarded\ P]\!] \Longrightarrow \exists\, Q.\ \Psi \rhd P \parallel P \longmapsto \tau \prec Q \wedge P' \sim Q \parallel !P$
by *blast*
moreover from *tauCongChainBangI* **have** $\bigwedge \Psi\ P\ P'.\ [\![\Psi \rhd P \parallel P \Longrightarrow_\tau P';\ guarded\ P]\!] \Longrightarrow \exists\, Q.\ \Psi \rhd !P \Longrightarrow_\tau Q \wedge \Psi \rhd Q \sim P' \parallel !P$
by *blast*
ultimately show *?case*
by(*rule-tac weakCongSimBangPres*[**where** *Rel*=*weakBisim* **and** *Rel'*=*bisim* **and** *Rel''*=*weakBisim* **and** *Eq*=$\lambda P\ Q.\ \forall \Psi.\ \Psi \rhd P \doteq Q$])
qed
thus *?thesis*
by(*metis weakPsiCongParNil weakPsiCongParComm weakPsiCongTransitive weakPsiCongSym*)
qed

end

end

theory *Weak-Congruence*
imports *Weak-Cong-Struct-Cong Bisim-Subst*
begin

context *env* **begin**

definition *weakCongruence* :: $('a,\ 'b,\ 'c)\ psi \Rightarrow ('a,\ 'b,\ 'c)\ psi \Rightarrow bool$ (‹- $\doteq_c$ -› [*70*, *70*] *65*)
where
$P \doteq_c Q \equiv \forall \Psi\ \sigma.\ wellFormedSubst\ \sigma \longrightarrow \Psi \rhd P[<\sigma>] \doteq Q[<\sigma>]$

lemma *weakCongE*:

fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and σ :: (*name list* $\times$ $'a$ *list*) *list*

assumes $P \doteq_c Q$
wellFormedSubst σ

shows $\Psi \rhd P[<\sigma>] \doteq Q[<\sigma>]$
using *assms*
by(*auto simp add*: *weakCongruence-def*)

lemma *weakCongI*[*case-names cWeakPsiCong*]:
fixes P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $\bigwedge\Psi\ \sigma.$ *wellFormedSubst* $\sigma \Longrightarrow \Psi \rhd P[<\sigma>] \doteq Q[<\sigma>]$

shows $P \doteq_c Q$
using *assms*
by(*auto simp add*: *weakCongruence-def*)

lemma *weakCongClosed*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and p :: *name prm*

assumes $P \doteq_c Q$

shows $(p \cdot P) \doteq_c (p \cdot Q)$
proof(*induct rule*: *weakCongI*)
case(*cWeakPsiCong* Ψ σ)
note ‹$P \doteq_c Q$›
moreover from ‹*wellFormedSubst* σ› **have** *wellFormedSubst* (*rev* $p \cdot \sigma$) **by** *simp*
ultimately have $((rev\ p) \cdot \Psi) \rhd P[<(rev\ p \cdot \sigma)>] \doteq Q[<(rev\ p \cdot \sigma)>]$
by(*rule weakCongE*)
thus *?case* **by**(*drule-tac p=p* **in** *weakPsiCongClosed*) (*simp add*: *eqvts*)
qed

lemma *weakCongReflexive*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

shows $P \doteq_c P$
by(*auto intro*: *weakCongI weakPsiCongReflexive*)

lemma *weakCongSym*:
fixes Ψ :: $'b$

and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $P \doteq_c Q$

shows $Q \doteq_c P$
using *assms*
by(*auto simp add*: *weakCongruence-def weakPsiCongSym*)

lemma *weakCongTransitive*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and R :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \doteq Q$
and $\Psi \rhd Q \doteq R$

shows $\Psi \rhd P \doteq R$
using *assms*
by(*auto intro*: *weakCongI weakPsiCongTransitive dest*: *weakCongE*)

lemma *weakCongWeakBisim*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $P \doteq_c Q$

shows $\Psi \rhd P \approx Q$
using *assms*
apply(*drule-tac* σ=[] **in** *weakCongE*)
by(*auto dest*: *weakPsiCongE*)

lemma *weakCongWeakPsiCong*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $P \doteq_c Q$

shows $\Psi \rhd P \doteq Q$
using *assms*
by(*drule-tac weakCongE*[**where** Ψ=Ψ **and** σ=[]]) *auto*

lemma *strongBisimWeakCong*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $P \sim_s Q$

shows $P \doteq_c Q$
proof(*induct rule*: *weakCongI*)
case(*cWeakPsiCong* Ψ σ)
from *assms* ‹*wellFormedSubst* σ› **have** $P[<\sigma>] \sim Q[<\sigma>]$
by(*rule closeSubstE*)
hence $\Psi \rhd P[<\sigma>] \sim Q[<\sigma>]$ **by**(*metis bisimE*(*3*) *statEqBisim Identity Commutativity*)
thus *?case* **by**(*rule strongBisimWeakPsiCong*)
qed

lemma *structCongWeakCong*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*

assumes $P \equiv_s Q$

shows $P \doteq_c Q$
using *assms*
by(*metis strongBisimWeakCong structCongBisimSubst*)

lemma *weakCongUnfold*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and σ :: (*name list* $\times$ $'a$ *list*) *list*

assumes $P \doteq_c Q$
and *wellFormedSubst* σ

shows $P[<\sigma>] \doteq_c Q[<\sigma>]$
proof(*induct rule*: *weakCongI*)
case(*cWeakPsiCong* Ψ σ')
with ‹*wellFormedSubst* σ› ‹*wellFormedSubst* σ'› **have** *wellFormedSubst*($\sigma@\sigma'$)
by *simp*
with ‹$P \doteq_c Q$› **have** $\Psi \rhd P[<(\sigma@\sigma')>] \doteq Q[<(\sigma@\sigma')>]$
by(*rule weakCongE*)
thus $\Psi \rhd P[<\sigma>][<\sigma'>] \doteq Q[<\sigma>][<\sigma'>]$
by *simp*
qed

lemma *weakCongOutputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$

and N :: $'a$

assumes $P \doteq_c Q$

shows $M\langle N\rangle.P \doteq_c M\langle N\rangle.Q$
using *assms*
by(*fastforce intro*: *weakCongI weakCongE weakPsiCongOutputPres*)

lemma *weakCongInputPres*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and *xvec* :: *name list*
and N :: $'a$

assumes $P \doteq_c Q$
and *distinct xvec*

shows $M(\!|\lambda*xvec\ N|\!).P \doteq_c M(\!|\lambda*xvec\ N|\!).Q$
proof(*induct rule*: *weakCongI*)
case(*cWeakPsiCong* Ψ σ)
obtain p **where** $(p \cdot xvec) \sharp* \sigma$
and $(p \cdot xvec) \sharp* P$ **and** $(p \cdot xvec) \sharp* Q$ **and** $(p \cdot xvec) \sharp* \Psi$ **and** $(p \cdot xvec) \sharp* N$
and S: $set\ p \subseteq set\ xvec \times set\ (p \cdot xvec)$
by(*rule-tac* $c=(\sigma, P, Q, \Psi, N)$ **in** *name-list-avoiding*) *auto*

from ‹$P \doteq_c Q$› **have** $(p \cdot P) \doteq_c (p \cdot Q)$
by(*rule weakCongClosed*)
{
fix *Tvec* :: $'a$ *list*
from ‹$(p \cdot P) \doteq_c (p \cdot Q)$› ‹*wellFormedSubst* σ› **have** $(p \cdot P)[<\sigma>] \doteq_c (p \cdot Q)[<\sigma>]$
by(*rule weakCongUnfold*)
moreover assume *length xvec* = *length Tvec* **and** *distinct xvec*
ultimately have $\Psi \rhd ((p \cdot P)[<\sigma>])[(p \cdot xvec)::=Tvec] \doteq ((p \cdot Q)[<\sigma>])[(p \cdot xvec)::=Tvec]$
by(*drule-tac weakCongE*[**where** $\sigma=[((p \cdot xvec), Tvec)]]$) *auto*
hence $\Psi \rhd ((p \cdot P)[<\sigma>])[(p \cdot xvec)::=Tvec] \approx ((p \cdot Q)[<\sigma>])[(p \cdot xvec)::=Tvec]$
by(*rule weakPsiCongE*)
}

with ‹$(p \cdot xvec) \sharp* \sigma$› ‹*distinct xvec*›
have $\Psi \rhd (M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P))[<\sigma>] \doteq (M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot Q))[<\sigma>]$
by(*force intro*: *weakPsiCongInputPres*)
moreover from ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* P$› S **have** $M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P) = M(\!|\lambda*xvec\ N|\!).P$

apply(*simp add*: *psi.inject*) **by**(*rule inputChainAlpha*[*symmetric*]) *auto*
moreover from ‹$(p \cdot xvec)$ $\sharp*$ N› ‹$(p \cdot xvec)$ $\sharp*$ Q› S **have** $M(\!|\lambda*(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot Q) = M(\!|\lambda*xvec\ N|\!).Q$
apply(*simp add*: *psi.inject*) **by**(*rule inputChainAlpha*[*symmetric*]) *auto*
ultimately show *?case* **by** *force*
qed

lemma *weakCongCasePresAux*:
fixes Ψ :: $'b$
and CsP :: $('c \times ('a, 'b, 'c)\ psi)\ list$
and CsQ :: $('c \times ('a, 'b, 'c)\ psi)\ list$

assumes *C1*: $\bigwedge\varphi\ P.\ (\varphi, P)\ mem\ CsP \Longrightarrow \exists\, Q.\ (\varphi, Q)\ mem\ CsQ \wedge guarded\ Q \wedge P \doteq_c Q$
and *C2*: $\bigwedge\varphi\ Q.\ (\varphi, Q)\ mem\ CsQ \Longrightarrow \exists\, P.\ (\varphi, P)\ mem\ CsP \wedge guarded\ P \wedge P \doteq_c Q$

shows $Cases\ CsP \doteq_c Cases\ CsQ$
proof −
{
fix Ψ :: $'b$
fix σ :: $(name\ list \times 'a\ list)\ list$

assume *wellFormedSubst* σ
have $\Psi \rhd Cases(caseListSeqSubst\ CsP\ \sigma) \doteq Cases(caseListSeqSubst\ CsQ\ \sigma)$
proof(*rule weakPsiCongCasePres*)
fix $\varphi\ P$
assume $(\varphi, P)\ mem\ (caseListSeqSubst\ CsP\ \sigma)$
then obtain $\varphi'\ P'$ **where** $(\varphi', P')\ mem\ CsP$ **and** $\varphi = substCond.seqSubst\ \varphi'\ \sigma$ **and** *PeqP'*: $P = (P'[<\sigma>])$
by(*induct CsP*) *force+*
from ‹$(\varphi', P')\ mem\ CsP$› **obtain** Q' **where** $(\varphi', Q')\ mem\ CsQ$ **and** $guarded\ Q'$ **and** $P' \doteq_c Q'$ **by**(*blast dest*: *C1*)
from ‹$(\varphi', Q')\ mem\ CsQ$› ‹$\varphi = substCond.seqSubst\ \varphi'\ \sigma$› **obtain** Q **where** $(\varphi, Q)\ mem\ (caseListSeqSubst\ CsQ\ \sigma)$ **and** $Q = Q'[<\sigma>]$
by(*induct CsQ*) *auto*
with *PeqP'* ‹$guarded\ Q'$› ‹$P' \doteq_c Q'$› ‹*wellFormedSubst* σ› **show** $\exists\, Q.\ (\varphi, Q)\ mem\ (caseListSeqSubst\ CsQ\ \sigma) \wedge guarded\ Q \wedge (\forall\, \Psi.\ \Psi \rhd P \doteq Q)$
by(*blast dest*: *weakCongE guardedSeqSubst*)
next
fix $\varphi\ Q$
assume $(\varphi, Q)\ mem\ (caseListSeqSubst\ CsQ\ \sigma)$
then obtain $\varphi'\ Q'$ **where** $(\varphi', Q')\ mem\ CsQ$ **and** $\varphi = substCond.seqSubst\ \varphi'\ \sigma$ **and** *QeqQ'*: $Q = Q'[<\sigma>]$
by(*induct CsQ*) *force+*
from ‹$(\varphi', Q')\ mem\ CsQ$› **obtain** P' **where** $(\varphi', P')\ mem\ CsP$ **and** $guarded\ P'$ **and** $P' \doteq_c Q'$ **by**(*blast dest*: *C2*)
from ‹$(\varphi', P')\ mem\ CsP$› ‹$\varphi = substCond.seqSubst\ \varphi'\ \sigma$› **obtain** P **where** $(\varphi, P)\ mem\ (caseListSeqSubst\ CsP\ \sigma)$ **and** $P = P'[<\sigma>]$

by(*induct CsP*) *auto*
with *QeqQ′* ‹*guarded P′*› ‹$P' \doteq_c Q'$› ‹*wellFormedSubst* σ›
show $\exists P.\ (\varphi,\ P)$ *mem* (*caseListSeqSubst CsP* σ) $\wedge$ *guarded* $P \wedge (\forall \Psi.\ \Psi \rhd P \doteq Q)$
by(*blast dest*: *weakCongE guardedSeqSubst*)
qed
}
thus *?thesis*
by(*rule-tac weakCongI*) *auto*
qed

lemma *weakCongParPres*:
fixes Ψ :: *′b*
and P :: (*′a*, *′b*, *′c*) *psi*
and Q :: (*′a*, *′b*, *′c*) *psi*
and R :: (*′a*, *′b*, *′c*) *psi*

assumes $P \doteq_c Q$

shows $P \parallel R \doteq_c Q \parallel R$
using *assms*
by(*fastforce intro*: *weakCongI weakCongE weakPsiCongParPres*)

lemma *weakCongResPres*:
fixes P :: (*′a*, *′b*, *′c*) *psi*
and Q :: (*′a*, *′b*, *′c*) *psi*
and x :: *name*

assumes $P \doteq_c Q$

shows $(\!|\nu x|\!)P \doteq_c (\!|\nu x|\!)Q$
proof(*induct rule*: *weakCongI*)
case(*cWeakPsiCong* Ψ σ)
obtain *y*::*name* **where** $y \sharp (\Psi::'b)$ **and** $y \sharp P$ **and** $y \sharp Q$ **and** $y \sharp \sigma$
by(*generate-fresh name*) (*auto simp add*: *fresh-prod*)

from ‹$P \doteq_c Q$› **have** $([(x,\ y)] \cdot P) \doteq_c ([(x,\ y)] \cdot Q)$ **by**(*rule weakCongClosed*)
hence $\Psi \rhd ([(x,\ y)] \cdot P)[<\sigma>] \doteq ([(x,\ y)] \cdot Q)[<\sigma>]$ **using** ‹*wellFormedSubst* σ›
by(*rule weakCongE*)
hence $\Psi \rhd (\!|\nu y|\!)(([(x,\ y)] \cdot P)[<\sigma>]) \doteq (\!|\nu y|\!)(([(x,\ y)] \cdot Q)[<\sigma>])$ **using** ‹$y \sharp \Psi$›
by(*rule weakPsiCongResPres*)
with ‹$y \sharp P$› ‹$y \sharp Q$› ‹$y \sharp \sigma$›
show $\Psi \rhd ((\!|\nu x|\!)P)[<\sigma>] \doteq ((\!|\nu x|\!)Q)[<\sigma>]$
by(*simp add*: *alphaRes*)
qed

lemma *weakCongBangPres*:
fixes P :: (*′a*, *′b*, *′c*) *psi*

and Q :: $('a, 'b, 'c)$ *psi*

assumes $P \doteq_c Q$
and *guarded* P
and *guarded* Q

shows $!P \doteq_c !Q$
using *assms*
by(*fastforce intro*: *weakCongI weakCongE weakPsiCongBangPres guardedSeqSubst*)

end

end

theory *Tau*
imports *Weak-Congruence Bisim-Struct-Cong*
begin

locale *tau* = *env* +
fixes *nameTerm* :: *name* $\Rightarrow$ $'a$

assumes *ntEqvt*[*eqvt*]: $(p{::}name\ prm) \cdot (nameTerm\ x) = nameTerm(p \cdot x)$
and *ntSupp*: $supp(nameTerm\ x) = \{x\}$
and *ntEq*: $\Psi \vdash (nameTerm\ x) \leftrightarrow M = (M = nameTerm\ x)$
and *subst4*: $[\![length\ xvec = length\ Tvec;\ distinct\ xvec;\ xvec \sharp* (M{::}'a)]\!] \Longrightarrow$
$M[xvec{::=}Tvec] = M$
begin

lemma *ntChanEq*[*simp*]:
fixes Ψ :: $'b$
and x :: *name*

shows $\Psi \vdash (nameTerm\ x) \leftrightarrow (nameTerm\ x)$
using *ntEq*
by *auto*

lemma *nameTermFresh*[*simp*]:
fixes x :: *name*
and y :: *name*

shows $x \sharp (nameTerm\ y) = (x \neq y)$
using *ntSupp*
by(*auto simp add*: *fresh-def*)

lemma *nameTermFreshChain*[*simp*]:
fixes *xvec* :: *name list*
and x :: *name*

shows $xvec \sharp* (nameTerm\ x) = x \sharp xvec$

by(*induct xvec*) *auto*

definition *tauPrefix* :: ('*a*, '*b*, '*c*) *psi* ⇒ ('*a*, '*b*, '*c*) *psi* (‹τ.-› [*85*] *85*)
where *tauPrefix P* ≡ *THE P'*. ∃*x*::*name*. *x* ♯ *P* ∧ *P'* = ⦇νx⦈(((*nameTerm x*)⦇λ∗([]) (*nameTerm x*)⦈.**0**) ∥ ((*nameTerm x*)⟨(*nameTerm x*)⟩.*P*))

lemma *tauActionUnfold*:
fixes *P* :: ('*a*, '*b*, '*c*) *psi*
and *C* :: '*d*::*fs-name*

obtains *x*::*name* **where** *x* ♯ *P* **and** *x* ♯ *C* **and** τ.(*P*) = ⦇νx⦈(((*nameTerm x*)⦇λ∗([]) (*nameTerm x*)⦈.**0**) ∥ ((*nameTerm x*)⟨(*nameTerm x*)⟩.*P*))
proof −
obtain *x*::*name* **where** *x* ♯ *P* **and** *x* ♯ *C* **by**(*generate-fresh name*) *auto*
hence ∃*x*::*name*. *x* ♯ *P* ∧ *x* ♯ *C* ∧ τ.(*P*) = ⦇νx⦈(((*nameTerm x*)⦇λ∗([]) (*nameTerm x*)⦈.**0**) ∥ ((*nameTerm x*)⟨(*nameTerm x*)⟩.*P*))
apply(*simp add*: *tauPrefix-def*)
apply(*subst the-equality*)
apply(*rule-tac x=x* **in** *exI*)
apply *simp*
apply(*clarify*)
apply(*simp add*: *psi.inject alpha*)
by(*auto simp add*: *calc-atm eqvts abs-fresh*)
moreover assume ⋀*x*. ⟦*x* ♯ *P*; *x* ♯ *C*;
τ.(*P*) =
⦇νx⦈((*nameTerm x*⦇λ∗[] *nameTerm x*⦈.**0**) ∥ *nameTerm x*⟨*nameTerm x*⟩.*P*)⟧
⟹ *thesis*
ultimately show *?thesis* **by** *blast*
qed

lemma *tauActionI*:
fixes *P* :: ('*a*, '*b*, '*c*) *psi*

shows ∃*P'*. Ψ ⊳ τ.(*P*) ⟼τ ≺ *P'* ∧ Ψ ⊳ *P* ∼ *P'*
proof −
obtain *x*:: *name* **where** *x* ♯ Ψ **and** *x* ♯ *P* **and** τ.(*P*) = ⦇νx⦈(((*nameTerm x*)⦇λ∗([]) (*nameTerm x*)⦈.**0**) ∥ ((*nameTerm x*)⟨(*nameTerm x*)⟩.*P*))
by(*rule tauActionUnfold*)

then have Ψ ⊳ τ.(*P*) ⟼τ ≺ ⦇νx⦈(**0** ∥ *P*)
apply *simp*
apply(*rule Scope*)
apply *auto*
apply(*subgoal-tac* Ψ ⊳ ((*nameTerm x*)⦇λ∗([]) (*nameTerm x*)⦈.**0**) ∥ ((*nameTerm x*)⟨(*nameTerm x*)⟩.*P*) ⟼τ ≺ ⦇ν∗[]⦈(**0** ∥ *P*))
apply *simp*
apply(*rule-tac M*=(*nameTerm x*) **and** *N*=(*nameTerm x*) **and** *K*=(*nameTerm x*) **in** *Comm1*)
apply(*auto intro*: *ntChanEq Output Input*)

apply(*subgoal-tac* $\Psi \otimes \mathbf{1} \rhd (nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0} \longmapsto (nameTerm\ x)(\!|((nameTerm\ x)[([])::=[]])|\!) \prec (\mathbf{0}[[]::=[]])$)
apply(*simp add*: *subst4*)
by(*rule-tac Input*) *auto*
moreover from ‹$x \sharp \Psi$› ‹$x \sharp P$› **have** $\Psi \rhd P \sim (\!|\nu x|\!)(\mathbf{0} \parallel P)$
by(*metis bisimTransitive bisimParNil bisimScopeExtSym bisimResNil bisimPar-PresSym bisimParComm bisimE*(*4*))
ultimately show *?thesis* **by** *blast*
qed

lemma *outputEmpy*[*dest*]:
assumes $\Psi \rhd M\langle N\rangle.P \longmapsto K(\!|\nu*xvec|\!)\langle N'\rangle \prec P'$

shows $xvec = []$
using *assms*
apply −
by(*ind-cases* $\Psi \rhd M\langle N\rangle.P \longmapsto K(\!|\nu*xvec|\!)\langle N'\rangle \prec P'$) (*auto simp add*: *psi.inject residualInject*)

lemma *tauActionE*:
fixes P :: ($'a$, $'b$, $'c$) *psi*

assumes $\Psi \rhd \tau.(P) \longmapsto \tau \prec P'$

shows $\Psi \rhd P \sim P'$ **and** $supp\ P' = ((supp\ P)::name\ set)$
proof −
obtain x:: *name* **where** $x \sharp (\Psi, P')$ **and** $x \sharp P$ **and** $\tau.(P) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule tauActionUnfold*)
with *assms* **have** $\Psi \rhd P \sim P' \wedge supp\ P' = ((supp\ P)::name\ set)$
apply *simp*
apply(*erule-tac resTauCases*)
apply *simp*+
apply(*erule-tac C*=*x* **in** *parCases*)
apply *simp*+
apply(*drule-tac nilTrans*(*3*)[**where** *xvec*=[], *simplified*])
apply *simp*
apply *force*
apply *simp*
apply(*subgoal-tac* $\Psi \otimes \mathbf{1} \rhd (nameTerm\ x)(\!|\lambda*[]\ (nameTerm\ x)|\!).\mathbf{0} \longmapsto M(\!|N|\!) \prec P'a$)
apply(*erule-tac inputCases*)
apply *simp*
apply(*subgoal-tac xvec* = [])
apply(*erule-tac outputCases*)
apply *simp*
apply(*clarsimp*)
apply(*rule conjI*)
apply(*metis bisimTransitive bisimParNil bisimScopeExtSym bisimResNil bisim-*

ParPresSym bisimParComm bisimE(*4*))
apply(*auto simp add*: *psi.supp abs-supp suppBottom*)
by(*simp add*: *fresh-def*)
thus $\Psi \rhd P \sim P'$ **and** *supp* $P' = ((supp\ P)::name\ set)$
by *auto*
qed

lemma *tauActionEqvt*[*eqvt*]:
fixes P :: ($'a$, $'b$, $'c$) *psi*
and p :: *name prm*

shows $(p \cdot \tau.(P)) = \tau.(p \cdot P)$
proof −
obtain x::*name* **where** $x \sharp P$ **and** $x \sharp p$ **and** $\tau.(P) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule tauActionUnfold*)
moreover obtain y::*name* **where** $y \sharp p \cdot P$ **and** $y \sharp (p, x)$ **and** $\tau.(p \cdot P) = (\!|\nu y|\!)(((nameTerm\ y)(\!|\lambda*([])\ (nameTerm\ y)|\!).\mathbf{0}) \parallel ((nameTerm\ y)\langle(nameTerm\ y)\rangle.(p \cdot P)))$
by(*rule tauActionUnfold*)
ultimately show *?thesis*
by(*simp add*: *eqvts calc-atm at-prm-fresh*[*OF at-name-inst*] *psi.inject alpha pt-fresh-perm-app*[*OF pt-name-inst*, *OF at-name-inst*])
qed

lemma *resCases'*[*consumes 7*, *case-names cOpen cRes*]:
fixes Ψ :: $'b$
and x :: *name*
and P :: ($'a$, $'b$, $'c$) *psi*
and α :: $'a$ *action*
and P' :: ($'a$, $'b$, $'c$) *psi*
and C :: $'d$::*fs-name*

assumes *Trans*: $\Psi \rhd (\!|\nu x|\!)P \longmapsto x\alpha \prec xP'$
and $x \sharp \Psi$
and $x \sharp x\alpha$
and $x \sharp xP'$
and $bn\ x\alpha \sharp* \Psi$
and $bn\ x\alpha \sharp* P$
and $bn\ x\alpha \sharp*$ *subject* $x\alpha$
and *rOpen*: $\bigwedge M\ xvec\ yvec\ y\ N\ P'.\ [\![\Psi \rhd P \longmapsto M(\!|\nu*(xvec@yvec)|\!)\langle([(x, y)] \cdot N)\rangle \prec ([(x, y)] \cdot P');\ y \in supp\ N;$
$x \sharp N;\ x \sharp P';\ x \neq y;\ y \sharp xvec;\ y \sharp yvec;\ y \sharp M;$
$distinct\ xvec;\ distinct\ yvec;$
$xvec \sharp* \Psi;\ y \sharp \Psi;\ yvec \sharp* \Psi;\ xvec \sharp* P;\ y \sharp P;$
$yvec \sharp* P;\ xvec \sharp* M;\ y \sharp M;$
$yvec \sharp* M;\ xvec \sharp* yvec;\ x\alpha = M(\!|\nu*(xvec@y\#yvec)|\!)\langle N\rangle;$
$xP' = P']\!] \Longrightarrow$
Prop

and *rScope*: $\bigwedge P'.\ [\![\Psi \rhd P \longmapsto x\alpha \prec P';\ xP' = (\!|\nu x|\!)P']\!] \Longrightarrow Prop$

shows *Prop*
proof −
from *Trans* **have** *distinct*(*bn* $x\alpha$) **by**(*auto dest*: *boundOutputDistinct*)
have *length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xP'$) **by** *simp*
note *Trans*
moreover have *length* [] = *inputLength*($(\!|\nu x|\!)P$) **and** *distinct* []
by(*auto simp add*: *inputLength-inputLength′-inputLength″.simps*)
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xP'$)› ‹*distinct*(*bn* $x\alpha$)›
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xP'$)› ‹*distinct*(*bn* $x\alpha$)›
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xP'$)› ‹*distinct*(*bn* $x\alpha$)›
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xP'$)› ‹*distinct*(*bn* $x\alpha$)›
ultimately show *?thesis* **using** ‹*bn* $x\alpha \sharp* \Psi$› ‹*bn* $x\alpha \sharp* P$› ‹*bn* $x\alpha \sharp*$ *subject* $x\alpha$› ‹$x \sharp \Psi$› ‹$x \sharp x\alpha$› ‹$x \sharp xP'$› ‹*distinct*(*bn* $x\alpha$)› *rScope rOpen*
apply(*cases rule*: *semanticsCases*[*of* - - - - - - - - - - *x x*])
apply(*auto simp add*: *psi.inject alpha abs-fresh residualInject boundOutputApp boundOutput.inject eqvts*)
apply(*subgoal-tac* $y \in$ *supp Na*)
apply(*auto simp add*: *residualInject boundOutputApp*)
apply(*drule-tac pi*=[(*x*, *y*)] **in** *pt-set-bij2*[*OF pt-name-inst*, *OF at-name-inst*])
by(*simp add*: *calc-atm eqvts*)
qed

lemma *parCases′*[*consumes 5*, *case-names cPar1 cPar2 cComm1 cComm2*]:
fixes Ψ :: *′b*
and P :: (*′a*, *′b*, *′c*) *psi*
and Q :: (*′a*, *′b*, *′c*) *psi*
and α :: *′a action*
and T :: (*′a*, *′b*, *′c*) *psi*
and C :: *′d*::*fs-name*

assumes *Trans*: $\Psi \rhd P \parallel Q \longmapsto x\alpha \prec xT$
and *bn* $x\alpha \sharp* \Psi$
and *bn* $x\alpha \sharp* P$
and *bn* $x\alpha \sharp* Q$
and *bn* $x\alpha \sharp*$ *subject* $x\alpha$
and *rPar1*: $\bigwedge P'\ A_Q\ \Psi_Q.\ [\![\Psi \otimes \Psi_Q \rhd P \longmapsto x\alpha \prec P';\ \ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;\ distinct\ A_Q;$
$A_Q \sharp* \Psi;\ A_Q \sharp* P;\ A_Q \sharp* Q;\ A_Q \sharp* x\alpha;\ A_Q \sharp* P';\ A_Q \sharp* C;\ xT = P' \parallel Q]\!] \Longrightarrow Prop$
and *rPar2*: $\bigwedge Q'\ A_P\ \Psi_P.\ [\![\Psi \otimes \Psi_P \rhd Q \longmapsto x\alpha \prec Q';\ \ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$A_P \sharp* \Psi;\ A_P \sharp* P;\ A_P \sharp* Q;\ A_P \sharp* x\alpha;\ A_P \sharp* Q';\ A_P \sharp* C;\ xT = P \parallel Q']\!] \Longrightarrow Prop$
and *rComm1*: $\bigwedge \Psi_Q\ M\ N\ P'\ A_P\ \Psi_P\ K\ xvec\ Q'\ A_Q.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|N|\!) \prec P';\ extractFrame\ P = \langle A_P, \Psi_P\rangle;\ distinct\ A_P;$
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|\nu{*}xvec|\!)\langle N\rangle \prec Q';\ extractFrame\ Q = \langle A_Q, \Psi_Q\rangle;$
$distinct\ A_Q;$

$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C$; $x\alpha{=}\tau$; $xT = (\!|\nu{*}xvec|\!)(P' \parallel Q')]\!] \Longrightarrow Prop$
and *rComm2*: $\bigwedge \Psi_Q\ M\ xvec\ N\ P'\ A_P\ \Psi_P\ K\ Q'\ A_Q.$
$[\![\Psi \otimes \Psi_Q \rhd P \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$; *extractFrame* $P = \langle A_P, \Psi_P\rangle$; *distinct* A_P;
$\Psi \otimes \Psi_P \rhd Q \longmapsto K(\!|N|\!) \prec Q'$; *extractFrame* $Q = \langle A_Q, \Psi_Q\rangle$; *distinct* A_Q;
$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$; *distinct xvec*;
$A_P \sharp* \Psi$; $A_P \sharp* \Psi_Q$; $A_P \sharp* P$; $A_P \sharp* M$; $A_P \sharp* N$; $A_P \sharp* P'$; $A_P \sharp* Q$; $A_P \sharp* xvec$; $A_P \sharp* Q'$; $A_P \sharp* A_Q$; $A_P \sharp* C$;
$A_Q \sharp* \Psi$; $A_Q \sharp* \Psi_P$; $A_Q \sharp* P$; $A_Q \sharp* K$; $A_Q \sharp* N$; $A_Q \sharp* P'$; $A_Q \sharp* Q$; $A_Q \sharp* xvec$; $A_Q \sharp* Q'$; $A_Q \sharp* C$;
$xvec \sharp* \Psi$; $xvec \sharp* \Psi_P$; $xvec \sharp* P$; $xvec \sharp* M$; $xvec \sharp* K$; $xvec \sharp* Q$; $xvec \sharp* \Psi_Q$; $xvec \sharp* C$; $x\alpha{=}\tau$; $xT = (\!|\nu{*}xvec|\!)(P' \parallel Q')]\!] \Longrightarrow Prop$

shows *Prop*
proof −
from *Trans* **have** *distinct*(*bn* $x\alpha$) **by**(*auto dest*: *boundOutputDistinct*)
have *length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xT$) **by** *simp*
note *Trans*
moreover have *length* [] = *inputLength*($P \parallel Q$) **and** *distinct* []
by(*auto simp add*: *inputLength-inputLength′-inputLength″.simps*)
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xT$)› ‹*distinct*(*bn* $x\alpha$)›
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xT$)› ‹*distinct*(*bn* $x\alpha$)›
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xT$)› ‹*distinct*(*bn* $x\alpha$)›
moreover note ‹*length*(*bn* $x\alpha$) = *residualLength*($x\alpha \prec xT$)› ‹*distinct*(*bn* $x\alpha$)›
moreover obtain *x*::*name* **where** $x \sharp \Psi$ **and** $x \sharp P$ **and** $x \sharp Q$ **and** $x \sharp x\alpha$ **and** $x \sharp xT$
by(*generate-fresh name*) *auto*
ultimately show *?thesis* **using** ‹*bn* $x\alpha \sharp* \Psi$› ‹*bn* $x\alpha \sharp* P$› ‹*bn* $x\alpha \sharp* Q$› ‹*bn* $x\alpha \sharp*$ *subject* $x\alpha$› **using** *rPar1 rPar2 rComm1 rComm2*
by(*cases rule*: *semanticsCases*[*of* - - - - - - - - - *C x x*]) (*auto simp add*: *psi.inject residualInject residualInject′*)
qed

lemma *inputCases′*[*consumes 1*, *case-names cInput*]:
fixes Ψ :: *′b*
and M :: *′a*
and *xvec* :: *name list*
and N :: *′a*
and P :: (*′a*, *′b*, *′c*) *psi*
and α :: *′a action*
and P' :: (*′a*, *′b*, *′c*) *psi*

assumes *Trans*: $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto \alpha \prec P'$

and *rInput*: $\bigwedge K\ Tvec.\ [\![\Psi \vdash M \leftrightarrow K;\ set\ xvec \subseteq supp\ N;\ length\ xvec = length\ Tvec;\ distinct\ xvec;\ \alpha{=}K(\!|N[xvec{::=}Tvec]|\!);\ P' = P[xvec{::=}Tvec]]\!] \Longrightarrow Prop\ (K(\!|N[xvec{::=}Tvec]|\!))\ (P[xvec{::=}Tvec])$

shows $Prop\ \alpha\ P'$
proof −
{
fix *xvec N P*
assume *Trans*: $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto\alpha \prec P'$
and $xvec \sharp{*}\ \Psi$ **and** $xvec \sharp{*}\ M$ **and** $xvec \sharp{*}\ \alpha$ **and** $xvec \sharp{*}\ P'$ **and** *distinct xvec*
and *rInput*: $\bigwedge K\ Tvec.\ [\![\Psi \vdash M \leftrightarrow K;\ set\ xvec \subseteq supp\ N;\ length\ xvec = length\ Tvec;\ distinct\ xvec;\ \alpha{=}K(\!|N[xvec{::=}Tvec]|\!);\ P'{=}P[xvec{::=}Tvec]]\!] \Longrightarrow Prop\ (K(\!|N[xvec{::=}Tvec]|\!))\ (P[xvec{::=}Tvec])$

from *Trans* **have** $bn\ \alpha = []$
apply −
by(*ind-cases* $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto\alpha \prec P'$) (*auto simp add*: *residualInject*)
from *Trans* **have** $distinct(bn\ \alpha)$ **by**(*auto dest*: *boundOutputDistinct*)
have $length(bn\ \alpha) = residualLength(\alpha \prec P')$ **by** *simp*
note *Trans*
moreover have $length\ xvec = inputLength(M(\!|\lambda{*}xvec\ N|\!).P)$ **by** *auto*
moreover note ‹*distinct xvec*›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover note ‹$length(bn\ \alpha) = residualLength(\alpha \prec P')$› ‹$distinct(bn\ \alpha)$›
moreover obtain $x{::}name$ **where** $x \sharp \Psi$ **and** $x \sharp P$ **and** $x \sharp M$ **and** $x \sharp xvec$ **and** $x \sharp \alpha$ **and** $x \sharp P'$ **and** $x \sharp N$
by(*generate-fresh name*) *auto*
ultimately have $Prop\ \alpha\ P'$ **using** ‹$bn\ \alpha = []$› ‹$xvec \sharp{*}\ \Psi$›‹$xvec \sharp{*}\ M$› ‹$xvec \sharp{*}\ \alpha$› ‹$xvec \sharp{*}\ P'$› *rInput*
apply(*cases rule*: *semanticsCases*[*of* - - - - - - - - - *C x*])
by(*fastforce simp add*: *residualInject psi.inject inputChainFresh*)+
}
note *Goal* = *this*
moreover obtain p :: *name prm* **where** $(p \cdot xvec) \sharp{*}\ \Psi$ **and** $(p \cdot xvec) \sharp{*}\ M$ **and** $(p \cdot xvec) \sharp{*}\ N$ **and** $(p \cdot xvec) \sharp{*}\ P$
and $(p \cdot xvec) \sharp{*}\ \alpha$ **and** $(p \cdot xvec) \sharp{*}\ P'$ **and** S: $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$
and $distinctPerm\ p$
by(*rule-tac xvec*=*xvec* **and** $c{=}(\Psi, M, N, P, \alpha, P')$ **in** *name-list-avoiding*) *auto*
from *Trans* ‹$(p \cdot xvec) \sharp{*}\ N$› ‹$(p \cdot xvec) \sharp{*}\ P$› S **have** $\Psi \rhd M(\!|\lambda{*}(p \cdot xvec)\ (p \cdot N)|\!).(p \cdot P) \longmapsto\alpha \prec P'$
by(*simp add*: *inputChainAlpha'*)
moreover {
fix *K Tvec*
assume $\Psi \vdash M \leftrightarrow K$

moreover assume $set(p \cdot xvec) \subseteq supp(p \cdot N)$
hence $(p \cdot set(p \cdot xvec)) \subseteq (p \cdot supp(p \cdot N))$ **by** *simp*
with ‹*distinctPerm p*› **have** $set\ xvec \subseteq supp\ N$ **by**(*simp add*: *eqvts*)
moreover assume $length(p \cdot xvec) = length(Tvec::'a\ list)$
hence $length\ xvec = length\ Tvec$ **by** *simp*
moreover assume *distinct xvec*
moreover assume $\alpha{=}K(\!|(p \cdot N)[(p \cdot xvec)::{=}Tvec]|\!)$ **and** $P' = (p \cdot P)[(p \cdot xvec)::{=}Tvec]$
with ‹$(p \cdot xvec) \sharp* P$› ‹$(p \cdot xvec) \sharp* N$› ‹*distinctPerm p*› ‹$length\ xvec = length\ Tvec$› S
have $\alpha{=}K(\!|(N[xvec::{=}Tvec])|\!)$ **and** $P' = P[xvec::{=}Tvec]$
by(*simp add*: *renaming substTerm.renaming*)+
ultimately have $Prop\ (K(\!|N[xvec::{=}Tvec]|\!))\ (P[xvec::{=}Tvec])$
by(*rule rInput*)
with ‹$length\ xvec = length\ Tvec$› S ‹*distinctPerm p*› ‹$(p \cdot xvec) \sharp* N$› ‹$(p \cdot xvec) \sharp* P$›
have $Prop\ (K(\!|(p \cdot N)[(p \cdot xvec)::{=}Tvec]|\!))\ ((p \cdot P)[(p \cdot xvec)::{=}Tvec])$
by(*simp add*: *renaming substTerm.renaming*)
}
moreover from *Trans* **have** *distinct xvec* **by**(*rule inputDistinct*)
hence $distinct(p \cdot xvec)$ **by** *simp*
ultimately show *?thesis* **using** ‹$(p \cdot xvec) \sharp* \Psi$› ‹$(p \cdot xvec) \sharp* M$› ‹$(p \cdot xvec) \sharp* \alpha$› ‹$(p \cdot xvec) \sharp* P'$› ‹*distinct xvec*›
by(*rule-tac Goal*) *assumption*+
qed

lemma *outputCases'*[*consumes 1*, *case-names cOutput*]:
fixes $\Psi :: 'b$
and $M :: 'a$
and $N :: 'a$
and $P :: ('a, 'b, 'c)\ psi$
and $\alpha :: 'a\ action$
and $P' :: ('a, 'b, 'c)\ psi$

assumes $\Psi \rhd M\langle N\rangle.P \longmapsto\alpha \prec P'$
and $\bigwedge K.\ [\![\Psi \vdash M \leftrightarrow K;\ subject\ \alpha{=}Some\ K]\!] \Longrightarrow Prop\ (K\langle N\rangle)\ P$

shows $Prop\ \alpha\ P'$
using *assms*
by(*cases rule*: *semantics.cases*) (*auto simp add*: *residualInject psi.inject*)

lemma *tauOutput*[*dest*]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $M :: 'a$
and $xvec :: name\ list$
and $N :: 'a$
and $P' :: ('a, 'b, 'c)\ psi$

assumes $\Psi \rhd \tau.(P) \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$

shows *False*
proof −
{
fix *xvec N P′*

assume $\Psi \rhd \tau.(P) \longmapsto M(\!|\nu{*}xvec|\!)\langle N\rangle \prec P'$
and $xvec \sharp{*} \Psi$
and $xvec \sharp{*} P$
and $xvec \sharp{*} M$

moreover obtain x:: *name* **where** $x \sharp \Psi$ **and** $x \sharp P$ **and** $x \sharp xvec$ **and** $x \sharp M$ **and** $x \sharp N$ **and** $x \sharp P'$
and $\tau.(P) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda{*}([])\ (nameTerm\ x)|\!).\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule-tac C*=(Ψ, *M*, *xvec*, *N*, *P′*) **in** *tauActionUnfold*) *auto*
ultimately have *False*
apply *simp*
apply(*erule-tac resCases′*)
apply *simp*+
apply(*erule-tac parOutputCases*)
apply(*simp add*: *action.inject psi.inject*)+
apply(*drule-tac nilTrans*(*2*)[**where** *xvec*=[], *simplified*])
apply *simp*+
apply(*simp add*: *action.inject*)
apply(*clarify*)
apply(*erule outputCases′*)
apply(*auto simp add*: *ntEq*)
apply(*erule-tac parCases′*)
apply(*auto simp add*: *abs-fresh*)
apply(*drule-tac nilTrans*(*2*)[**where** *xvec*=[], *simplified*])
apply *simp*
apply(*erule-tac outputCases′*)
apply *auto*
by(*simp add*: *ntEq*)
}
moreover obtain *p* **where** $set\ p \subseteq set\ xvec \times set(p \cdot xvec)$ **and** $(p \cdot xvec) \sharp{*} N$ **and** $(p \cdot xvec) \sharp{*} P'$
and $(p \cdot xvec) \sharp{*} \Psi$ **and** $(p \cdot xvec) \sharp{*} P$ **and** $(p \cdot xvec) \sharp{*} M$
by(*rule-tac c*=(*N*, *P′*, Ψ, *P*, *M*) **and** *xvec*=*xvec* **in** *name-list-avoiding*) *auto*
ultimately show *False* **using** *assms* **by**(*simp add*: *boundOutputChainAlpha″ residualInject*) *auto*
qed

lemma *tauInput*[*dest*]:
fixes Ψ :: *′b*
and *P* :: (*′a*, *′b*, *′c*) *psi*
and *M* :: *′a*

and N :: $'a$
and P' :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd \tau.(P) \longmapsto M(\!|N|\!) \prec P'$

shows *False*
proof −
obtain x:: *name* **where** $x \sharp \Psi$ **and** $x \sharp P$ **and** $x \sharp M$ **and** $x \sharp N$ **and** $x \sharp P'$
and $\tau.(P) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule-tac* C=(Ψ, M, N, P') **in** *tauActionUnfold*) *auto*
with *assms* **show** *False*
apply *simp*
apply(*erule-tac resCases'*)
apply *auto*
apply(*drule-tac parCases'*)
apply(*auto simp add*: *abs-fresh*)
apply(*subgoal-tac* $\Psi \otimes \mathbf{1} \rhd (nameTerm\ x)(\!|\lambda*[]\ (nameTerm\ x)|\!).\mathbf{0} \longmapsto M(\!|N|\!) \prec P'a$)
apply(*erule-tac inputCases'*)
by(*auto simp add*: *action.inject subst4*)
qed

lemma *tauPrefixFrame*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*

shows $extractFrame(\tau.(P)) \simeq_F \langle\varepsilon,\ \mathbf{1}\rangle$
proof −
obtain x:: *name* **where** $x \sharp P$
and $\tau.(P) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule-tac* C=() **in** *tauActionUnfold*) *auto*
thus *?thesis*
by(*force intro*: *frameResFresh Identity FrameStatEqTrans*)
qed

lemma *insertTauAssertion*:
fixes P :: $('a,\ 'b,\ 'c)$ *psi*
and Ψ :: $'b$

shows $insertAssertion\ (extractFrame(\tau.(P)))\ \Psi \simeq_F \langle\varepsilon,\ \Psi\rangle$
proof −
obtain x:: *name* **where** $x \sharp P$ **and** $x \sharp \Psi$
and $\tau.(P) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule-tac* C=Ψ **in** *tauActionUnfold*) *auto*
thus *?thesis*
apply *auto*
apply(*rule-tac* G=$\langle\varepsilon,\ \Psi \otimes \mathbf{1} \otimes \mathbf{1}\rangle$ **in** *FrameStatEqTrans*)

apply(*rule-tac frameResFresh*) **apply** *auto*
apply(*subgoal-tac* $x \sharp \mathbf{1} \otimes \mathbf{1}$)
apply *auto*
by(*metis Identity AssertionStatEqTrans AssertionStatEqSym Associativity*)
qed

lemma *seqSubst4*:
assumes $y \sharp \sigma$
and *wellFormedSubst* σ

shows *substTerm.seqSubst* (*nameTerm y*) σ = *nameTerm y*
using *assms*
by(*induct* σ) (*auto simp add*: *subst4*)

lemma *tauSeqSubst*[*simp*]:
fixes P :: ($'a$, $'b$, $'c$) *psi*
and σ :: (*name list* × $'a$ *list*) *list*

assumes *wellFormedSubst* σ

shows $(\tau.(P))[<\sigma>] = \tau.(P[<\sigma>])$
proof −
obtain x:: *name* **where** $x \sharp P[<\sigma>]$ **and** $x \sharp \sigma$ **and** $x \sharp P$
and $\tau.(P[<\sigma>]) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm\ x)|\!).\mathbf{0})$
$\|\ ((nameTerm\ x)\langle(nameTerm\ x)\rangle.(P[<\sigma>])))$
by(*rule-tac* C=(σ, P) **in** *tauActionUnfold*) *auto*
moreover obtain y:: *name* **where** $y \sharp P[<\sigma>]$ **and** $y \sharp \sigma$ **and** $y \sharp P$ **and** $x \neq y$
and $\tau.(P) = (\!|\nu y|\!)(((nameTerm\ y)(\!|\lambda*([])\ (nameTerm\ y)|\!).\mathbf{0})\ \|$
$((nameTerm\ y)\langle(nameTerm\ y)\rangle.(P)))$
by(*rule-tac* C=(σ, $P[<\sigma>]$, x) **in** *tauActionUnfold*) *auto*
ultimately show *?thesis* **using** *assms*
by(*auto simp add*: *psi.inject alpha eqvts calc-atm psi.inject input.inject seqSubst4*)
qed

lemma *tauSubst*[*simp*]:
fixes P :: ($'a$, $'b$, $'c$) *psi*
and *xvec* :: *name list*
and *Tvec* :: $'a$ *list*

assumes *distinct xvec*
and *length xvec* = *length Tvec*

shows $(\tau.(P))[xvec::=Tvec] = \tau.(P[xvec::=Tvec])$
proof −
obtain x:: *name* **where** $x \sharp P[xvec::=Tvec]$ **and** $x \sharp xvec$ **and** $x \sharp Tvec$ **and** $x \sharp P$
and $\tau.(P[xvec::=Tvec]) = (\!|\nu x|\!)(((nameTerm\ x)(\!|\lambda*([])\ (nameTerm$

$x)\rparen\!\rparen.\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.(P[xvec::=Tvec])))$
by(*rule-tac C*=(*xvec*, *Tvec*, *P*) **in** *tauActionUnfold*) *auto*
moreover obtain *y*:: *name* **where** $y \sharp P[xvec::=Tvec]$ **and** $y \sharp xvec$ **and** $y \sharp Tvec$ **and** $y \sharp P$ **and** $x \neq y$
and $\tau.(P) = \lparen\!\lparen\nu y\rparen\!\rparen(((nameTerm\ y)\lparen\!\lparen\lambda*([])\ (nameTerm\ y)\rparen\!\rparen.\mathbf{0}) \parallel ((nameTerm\ y)\langle(nameTerm\ y)\rangle.(P)))$
by(*rule-tac C*=(*xvec*, *Tvec*, *P*[*xvec*::=*Tvec*], *x*) **in** *tauActionUnfold*) *auto*
ultimately show *?thesis* **using** *assms*
by(*auto simp add*: *psi.inject alpha eqvts calc-atm psi.inject input.inject subst4*)
qed

lemma *tauFresh*[*simp*]:
fixes P :: ('*a*, '*b*, '*c*) *psi*
and x :: *name*

shows $x \sharp \tau.(P) = x \sharp P$
proof −
obtain *y*::*name* **where** $y \neq x$ **and** $y \sharp P$ **and** $\tau.(P) = \lparen\!\lparen\nu y\rparen\!\rparen(((nameTerm\ y)\lparen\!\lparen\lambda*([])\ (nameTerm\ y)\rparen\!\rparen.\mathbf{0}) \parallel ((nameTerm\ y)\langle(nameTerm\ y)\rangle.P))$
by(*rule-tac C*=*x* **in** *tauActionUnfold*) *auto*

thus *?thesis* **by**(*auto simp add*: *abs-fresh*)
qed

lemma *tauFreshChain*[*simp*]:
fixes P :: ('*a*, '*b*, '*c*) *psi*
and *xvec* :: *name list*

shows $xvec \sharp* (\tau.(P)) = (xvec \sharp* P)$
by(*induct xvec*) *auto*

lemma *guardedTau*:
fixes P :: ('*a*, '*b*, '*c*) *psi*

shows $guarded(\tau.(P))$
proof −
obtain *x*::*name* **where** $x \sharp P$ **and** $\tau.(P) = \lparen\!\lparen\nu x\rparen\!\rparen(((nameTerm\ x)\lparen\!\lparen\lambda*([])\ (nameTerm\ x)\rparen\!\rparen.\mathbf{0}) \parallel ((nameTerm\ x)\langle(nameTerm\ x)\rangle.P))$
by(*rule-tac C*=() **in** *tauActionUnfold*) *auto*

thus *?thesis* **by** *auto*
qed

lemma *tauChainBisim*:
fixes Ψ :: '*b*
and P :: ('*a*, '*b*, '*c*) *psi*
and P' :: ('*a*, '*b*, '*c*) *psi*
and P'' :: ('*a*, '*b*, '*c*) *psi*

assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$
and $\Psi \rhd P \sim P''$

obtains P''' **where** $\Psi \rhd P'' \Longrightarrow\hat{}_\tau P'''$ **and** $\Psi \rhd P' \sim P'''$
using *assms*
proof(*induct arbitrary*: *thesis rule*: *tauChainInduct*)
case(*TauBase P*)
thus *?case* **by** *auto*
next
case(*TauStep P P' P'''*)
from ‹$\Psi \rhd P \sim P''$› **obtain** P'''' **where** *P''Chain*: $\Psi \rhd P'' \Longrightarrow\hat{}_\tau P''''$ **and** $\Psi \rhd P' \sim P''''$
by(*rule-tac TauStep*) *auto*
from ‹$\Psi \rhd P' \sim P''''$› ‹$\Psi \rhd P' \longmapsto\tau \prec P'''$›
obtain P''''' **where** *P''''Trans*: $\Psi \rhd P'''' \longmapsto\tau \prec P'''''$ **and** $\Psi \rhd P''' \sim P'''''$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))
apply(*drule-tac simE*)
apply *auto*
apply(*drule-tac bisimE*(*4*))
by *blast*
from *P''Chain P''''Trans* **have** $\Psi \rhd P'' \Longrightarrow\hat{}_\tau P'''''$ **by**(*drule-tac tauActTauChain*) *auto*
with ‹$\Psi \rhd P''' \sim P'''''$› **show** *?case*
by(*metis TauStep*)
qed

lemma *tauChainStepCons*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and P' :: (*'a*, *'b*, *'c*) *psi*

assumes *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$

obtains P'' **where** $\Psi \rhd \tau.(P) \Longrightarrow_\tau P''$ **and** $\Psi \rhd P' \sim P''$
proof −
assume *Goal*: $\bigwedge P''.\ [\![\Psi \rhd \tau.(P) \Longrightarrow_\tau P''; \Psi \rhd P' \sim P'']\!] \Longrightarrow$ *thesis*
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
from *PChain* ‹$\Psi \rhd P \sim P''$› **obtain** P''' **where** *P''Chain*: $\Psi \rhd P'' \Longrightarrow\hat{}_\tau P'''$ **and** $\Psi \rhd P' \sim P'''$
by(*rule tauChainBisim*)
from *PTrans P''Chain* **have** $\Psi \rhd \tau.(P) \Longrightarrow_\tau P'''$ **by**(*drule-tac tauActTauStepChain*) *auto*
thus *?thesis* **using** ‹$\Psi \rhd P' \sim P'''$›
by(*rule Goal*)
qed

lemma *tauChainCons*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and P' :: $('a, 'b, 'c)$ *psi*

assumes $\Psi \rhd P \Longrightarrow\hat{}_\tau P'$

obtains P'' **where** $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau P''$ **and** $\Psi \rhd P' \sim P''$
proof −
assume *Goal*: $\bigwedge P''.\ [\![\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau P''; \Psi \rhd P' \sim P'']\!] \Longrightarrow$ *thesis*
from *assms* **obtain** P'' **where** $\Psi \rhd \tau.(P) \Longrightarrow_\tau P''$ **and** $\Psi \rhd P' \sim P''$
by(*rule tauChainStepCons*)
thus *?thesis* **by**(*rule-tac Goal*) *auto*
qed

lemma *weakTransitionTau*:
fixes Ψ :: $'b$
and Q :: $('a, 'b, 'c)$ *psi*
and P :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *action*
and P' :: $('a, 'b, 'c)$ *psi*

assumes *PTrans*: $\Psi : Q \rhd P \Longrightarrow\alpha \prec P'$
and $bn\ \alpha \sharp* \Psi$
and $bn\ \alpha \sharp* P$

obtains P'' **where** $\Psi : Q \rhd \tau.(P) \Longrightarrow\alpha \prec P''$ **and** $\Psi \rhd P' \sim P''$
proof −
assume *Goal*: $\bigwedge P''.\ [\![\Psi : Q \rhd \tau.(P) \Longrightarrow\alpha \prec P''; \Psi \rhd P' \sim P'']\!] \Longrightarrow$ *thesis*
from *PTrans* **obtain** P'' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau P''$
and *QimpP''*: *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P'') Ψ
and *P''Trans*: $\Psi \rhd P'' \longmapsto\alpha \prec P'$
by(*blast dest*: *weakTransitionE*)

from *PChain* **obtain** P''' **where** *tPChain*: $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau P'''$ **and** $\Psi \rhd P'' \sim P'''$ **by**(*rule tauChainCons*)
moreover from *QimpP''* ‹$\Psi \rhd P'' \sim P'''$› **have** *insertAssertion* (*extractFrame* Q) $\Psi \hookrightarrow_F$ *insertAssertion*(*extractFrame* P''') Ψ
by(*metis bisimE FrameStatEq-def FrameStatImpTrans*)
moreover from *tPChain* ‹$bn\ \alpha \sharp* P$› **have** $bn\ \alpha \sharp* P'''$ **by**(*force intro*: *tauChainFreshChain*)
with ‹$\Psi \rhd P'' \sim P'''$› *P''Trans* ‹$bn\ \alpha \sharp* \Psi$› **obtain** P'''' **where** $\Psi \rhd P''' \longmapsto\alpha \prec P''''$ **and** $\Psi \rhd P' \sim P''''$
by(*metis bisimE simE*)
ultimately show *?thesis*
by(*rule-tac Goal*) (*blast intro*: *weakTransitionI*)
qed

end

end

theory *Sum*
  imports *Semantics Close-Subst*
begin

context *env*
begin

abbreviation *sumAssertJudge* (‹- $\oplus_{-}$ -› [*150*, *50*, *50*] *150*)
  where (P::($'a$, $'b$, $'c$) *psi*) $\oplus_{\varphi}$ Q $\equiv$ *Cases* [(φ, P), (φ, Q)]

lemma *SumAssert1*:
  fixes Ψ :: $'b$
  and P :: ($'a$, $'b$, $'c$) *psi*
  and Q :: ($'a$, $'b$, $'c$) *psi*

  assumes $\Psi \rhd P \longmapsto Rs$
  and $\Psi \vdash \varphi$
  and *guarded* P

  shows $\Psi \rhd P \oplus_{\varphi} Q \longmapsto Rs$
proof −
  note ‹$\Psi \rhd P \longmapsto Rs$›
  moreover have (φ, P) *mem* [(φ, P), (φ, Q)] **by** *simp*
  ultimately show *?thesis* **using** ‹$\Psi \vdash \varphi$› ‹*guarded* P›
    by(*rule Case*)
qed

lemma *SumAssert2*:
  fixes Ψ :: $'b$
  and P :: ($'a$, $'b$, $'c$) *psi*
  and Q :: ($'a$, $'b$, $'c$) *psi*

  assumes $\Psi \rhd Q \longmapsto Rs$
  and $\Psi \vdash \varphi$
  and *guarded* Q

  shows $\Psi \rhd P \oplus_{\varphi} Q \longmapsto Rs$
proof −
  note ‹$\Psi \rhd Q \longmapsto Rs$›
  moreover have (φ, Q) *mem* [(φ, P), (φ, Q)] **by** *simp*
  ultimately show *?thesis* **using** ‹$\Psi \vdash \varphi$› ‹*guarded* Q›
    by(*rule Case*)
qed

lemma *sumAssertCases*[*consumes 2*, *case-names cSum1 cSum2*]:

```
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   φ :: 'c

  assumes Ψ ▷ P ⊕φ Q ⟼ Rs
  and     Ψ ⊢ φ
  and     rSum1: ⟦Ψ ▷ P ⟼ Rs; guarded P⟧ ⟹ Prop
  and     rSum2: ⟦Ψ ▷ Q ⟼ Rs; guarded Q⟧ ⟹ Prop

  shows Prop
proof −
  from ‹Ψ ▷ P ⊕φ Q ⟼ Rs› show ?thesis
  proof(induct rule: caseCases)
    case(cCase φ' P')
    from ‹(φ', P') mem [(φ, P), (φ, Q)]›
    have φ = φ' and D: P = P' ∨ Q = P' by auto
    from D show ?thesis
    proof(rule disjE)
      assume P = P'
      with ‹Ψ ▷ P' ⟼ Rs› ‹guarded P'› show ?case by(rule-tac rSum1) auto
    next
      assume Q = P'
      with ‹Ψ ▷ P' ⟼ Rs› ‹guarded P'› show ?case by(rule-tac rSum2) auto
    qed
  qed
qed

lemma sumElim[dest]:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi
  and   φ :: 'c

  assumes Ψ ▷ P ⊕φ Q ⟼ Rs
  and     ¬(Ψ ⊢ φ)

  shows False
using assms
by(induct rule: caseCases) auto

end

locale sum = env +
  fixes T :: 'c

  assumes Top: Ψ ⊢ T
  and     TopEqvt[eqvt]: ((p::name prm) • T) = T
  and     TopSubst[simp]: substCond T xvec Tvec = T
```

begin

abbreviation *topJudge* (‹⊤› *150*) **where** $\top \equiv T$
abbreviation *sumJudge* (**infixr** ‹⊕› *80*) **where** $P \oplus Q \equiv P \oplus_\top Q$

lemma *topSeqSubst*[*simp*]:
  shows (*substCond.seqSubst* T σ) = T
by(*induct* σ) *auto*

lemma *suppTop*:
  shows ((*supp*(⊤))::*name set*) = {}
by(*auto simp add*: *supp-def eqvts*)

lemma *freshTop*[*simp*]:
  fixes x :: *name*
  and *xvec* :: *name list*
  and *Xs* :: *name set*

  shows $x \sharp \top$ **and** *xvec* $\sharp*$ ⊤ **and** *Xs* $\sharp*$ ⊤
by(*auto simp add*: *fresh-def fresh-star-def suppTop*)

lemma *sumSubst*[*simp*]:
  fixes P :: ('*a*, '*b*, '*c*) *psi*
  and Q :: ('*a*, '*b*, '*c*) *psi*
  and *xvec* :: *name list*
  and *Tvec* :: '*a list*

  assumes *length xvec* = *length Tvec*
  and *distinct xvec*

  shows $(P \oplus Q)$[*xvec*::=*Tvec*] = (P[*xvec*::=*Tvec*] $\oplus$ Q[*xvec*::=*Tvec*])
by *auto*

lemma *sumSeqSubst*[*simp*]:
  fixes P :: ('*a*, '*b*, '*c*) *psi*
  and Q :: ('*a*, '*b*, '*c*) *psi*
  and σ :: (*name list* × '*a list*) *list*

  assumes *wellFormedSubst* σ

  shows $(P \oplus Q)$[<σ>] = ((P[<σ>]) $\oplus$ (Q[<σ>]))
using *assms*
by(*induct* σ *arbitrary*: P Q) *auto*

lemma *Sum1*:
  fixes Ψ :: '*b*
  and P :: ('*a*, '*b*, '*c*) *psi*
  and Q :: ('*a*, '*b*, '*c*) *psi*

```
  assumes Ψ ▷ P ⟼ Rs
  and     guarded P

  shows Ψ ▷ P ⊕ Q ⟼ Rs
proof −
  have Ψ ⊢ ⊤ by(rule Top)
  with ‹Ψ ▷ P ⟼ Rs› show ?thesis using ‹guarded P›
    by(rule SumAssert1)
qed

lemma Sum2:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi

  assumes Ψ ▷ Q ⟼ Rs
  and     guarded Q

  shows Ψ ▷ P ⊕ Q ⟼ Rs
proof −
  have Ψ ⊢ ⊤ by(rule Top)
  with ‹Ψ ▷ Q ⟼ Rs› show ?thesis using ‹guarded Q›
    by(rule SumAssert2)
qed

lemma sumCases[consumes 1, case-names cSum1 cSum2]:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi
  and   Q :: ('a, 'b, 'c) psi

  assumes Ψ ▷ P ⊕ Q ⟼ Rs
  and     rSum1: ⟦Ψ ▷ P ⟼ Rs; guarded P⟧ ⟹ Prop
  and     rSum2: ⟦Ψ ▷ Q ⟼ Rs; guarded Q⟧ ⟹ Prop

  shows Prop
proof −
  have Ψ ⊢ ⊤ by(rule Top)
  with ‹Ψ ▷ P ⊕ Q ⟼ Rs› show ?thesis
  proof(induct rule: sumAssertCases)
    case cSum1
    thus ?case by (rule rSum1)
  next
    case cSum2
    thus ?case by(rule rSum2)
  qed
qed

end
```

end

theory *Tau-Sim*
  imports *Tau Sum*
begin

nominal-datatype *'a prefix* =
  pInput 'a::fs-name name list 'a
| *pOutput 'a 'a*
| *pTau*

context *tau*
begin

nominal-primrec *bindPrefix* :: *'a prefix* ⇒ (*'a*, *'b*, *'c*) *psi* ⇒ (*'a*, *'b*, *'c*) *psi* (‹-·-› [*100*, *100*] *100*)
where
  bindPrefix (*pInput M xvec N*) *P* = *M*⦇λ∗*xvec N*⦈.*P*
| *bindPrefix* (*pOutput M N*) *P* = *M*⟨*N*⟩.*P*
| *bindPrefix* (*pTau*) *P* = τ.(*P*)
by(*rule TrueI*)+

lemma *bindPrefixEqvt*[*eqvt*]:
  fixes *p* :: *name prm*
  and α :: *'a prefix*
  and *P* :: (*'a*, *'b*, *'c*) *psi*

  shows (*p* · (α·*P*)) = (*p* · α)·(*p* · *P*)
by(*nominal-induct* α *rule*: *prefix.strong-inducts*) (*auto simp add*: *eqvts*)

lemma *prefixCases*[*consumes 1*, *case-names cInput cOutput cTau*]:
  fixes Ψ :: *'b*
  and α :: *'a prefix*
  and *P* :: (*'a*, *'b*, *'c*) *psi*
  and β :: *'a action*
  and *P'* :: (*'a*, *'b*, *'c*) *psi*

  assumes Ψ ⊳ α·*P* ⟼β ≺ *P'*
  and *rInput*: ⋀*M xvec N K Tvec*. ⟦Ψ ⊢ *M* ↔ *K*; *set xvec* ⊆ *supp N*; *length xvec* = *length Tvec*; *distinct xvec*⟧ ⟹
Prop (*pInput M xvec N*) (*K*⦇*N*[*xvec*::=*Tvec*]⦈) (*P*[*xvec*::=*Tvec*])
  and *rOutput*: ⋀*M N K*. Ψ ⊢ *M* ↔ *K* ⟹ *Prop* (*pOutput M N*) (*K*⟨*N*⟩) *P*
  and *rTau*: Ψ ⊳ *P* ∼ *P'* ⟹ *Prop* (*pTau*) (τ) *P'*

  shows *Prop* α β *P'*
using ‹Ψ ⊳ α·*P* ⟼β ≺ *P'*›
proof(*nominal-induct rule*: *prefix.strong-induct*)
  case(*pInput M xvec N*)

from ‹$\Psi \rhd (pInput\ M\ xvec\ N)\cdot P \longmapsto\beta \prec P'$› **have** $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).P \longmapsto\beta \prec P'$ **by** *simp*
thus *?case* **by**(*auto elim*: *inputCases intro*: *rInput*)
next
case(*pOutput M N*)
from ‹$\Psi \rhd (pOutput\ M\ N)\cdot P \longmapsto\beta \prec P'$› **have** $\Psi \rhd M\langle N\rangle.P \longmapsto\beta \prec P'$ **by** *simp*
thus *?case* **by**(*auto elim*: *outputCases intro*: *rOutput*)
next
case *pTau*
from ‹$\Psi \rhd (pTau)\cdot P \longmapsto\beta \prec P'$› **have** $\Psi \rhd \tau.(P) \longmapsto\beta \prec P'$ **by** *simp*
thus *?case*
by(*nominal-induct rule*: *action.strong-induct*) (*auto dest*: *tauActionE intro*: *rTau*)
qed

lemma *prefixTauCases*[*consumes 1*, *case-names cTau*]:
fixes α :: *'a prefix*
and P :: (*'a*, *'b*, *'c*) *psi*
and P' :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd \alpha\cdot P \longmapsto\tau \prec P'$
and *rTau*: $\Psi \rhd P \sim P' \Longrightarrow Prop\ (pTau)\ P'$

shows $Prop\ \alpha\ P'$
proof −
from ‹$\Psi \rhd \alpha\cdot P \longmapsto\tau \prec P'$› **obtain** β **where** $\Psi \rhd \alpha\cdot P \longmapsto\beta \prec P'$ **and** $\beta = \tau$
by *auto*
thus *?thesis*
by(*induct rule*: *prefixCases*) (*auto intro*: *rTau*)
qed

lemma *hennessySim1*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \rightsquigarrow$<*Rel*> Q
and *C1*: $\bigwedge\Psi\ P\ Q\ R.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in Rel]\!] \Longrightarrow (\Psi,\ P,\ R) \in Rel$

shows $\Psi \rhd \tau.(P) \rightsquigarrow$«*Rel*» Q
proof(*induct rule*: *weakCongSimI*)
case(*cTau Q'*)
from ‹$\Psi \rhd P \rightsquigarrow$<*Rel*> Q› ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$›
obtain P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'$ **and** *P'RelQ'*: $(\Psi,\ P',\ Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *PChain* **obtain** P'' **where** $\Psi \rhd \tau.(P) \Longrightarrow_\tau P''$ **and** $\Psi \rhd P' \sim P''$
by(*rule tauChainStepCons*)

thus *?case* **using** $P'RelQ'$ **by**(*metis bisimE C1*)
qed

lemma *hennessySim2*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes *PTrans*: $\Psi \rhd P \longmapsto\tau \prec P'$
and $P'RelQ$: $(\Psi,\ P',\ Q) \in Rel$
and *C1*: $\bigwedge\Psi\ P\ Q\ R.\ [\![(\Psi,\ P,\ Q) \in Rel;\ \Psi \rhd Q \sim R]\!] \Longrightarrow (\Psi,\ P,\ R) \in Rel$

shows $\Psi \rhd P \rightsquigarrow$«*Rel*» $\tau.(Q)$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* Q')
from ‹$\Psi \rhd \tau.(Q) \longmapsto\tau \prec Q'$› **have** $\Psi \rhd Q \sim Q'$ **by**(*blast dest*: *tauActionE*)
with *PTrans* $P'RelQ$ **show** *?case* **by**(*metis C1 tauActTauStepChain*)
qed

lemma *hennessySim3*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd P \rightsquigarrow$<*Rel*> Q
and *C1*: $\bigwedge Q'.\ \Psi \rhd Q \longmapsto\tau \prec Q' \Longrightarrow (\Psi,\ P,\ Q') \notin Rel$

shows $\Psi \rhd P \rightsquigarrow$«*Rel*» Q
proof(*induct rule*: *weakCongSimI*)
case(*cTau* Q')
from ‹$\Psi \rhd P \rightsquigarrow$<*Rel*> Q› ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$›
obtain P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_{\tau}\ P'$ **and** $P'RelQ'$: $(\Psi,\ P',\ Q') \in Rel$
by(*blast dest*: *weakSimE*)
with *C1* ‹$\Psi \rhd Q \longmapsto\tau \prec Q'$› **show** *?case*
by(*force simp add*: *rtrancl-eq-or-trancl*)
qed

lemma *tauLaw1SimLeft*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $\Psi \rhd P \rightsquigarrow$<*Rel*> Q
and *eqvt Rel*
and *C1*: $\bigwedge\Psi\ P\ Q\ R.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in Rel]\!] \Longrightarrow (\Psi,\ P,\ R) \in Rel$

shows $\Psi \rhd \tau.(P) \rightsquigarrow$<*Rel*> Q
proof(*induct rule*: *weakSimI2*)
case(*cAct* $\Psi'\ \alpha\ Q'$)

from ‹$bn\ \alpha\ \sharp*\ (\tau.(P))$› **have** $bn\ \alpha\ \sharp*\ P$ **by** *simp*
with ‹$\Psi \rhd P \rightsquigarrow$<*Rel*> Q› ‹$\Psi \rhd Q \longmapsto \alpha \prec Q'$› ‹$bn\ \alpha\ \sharp*\ \Psi$› ‹$\alpha \neq \tau$›
obtain $P''\ P'$ **where** *PTrans*: $\Psi : Q \rhd P \Longrightarrow \alpha \prec P''$ **and** P''*Chain*: $\Psi \otimes \Psi' \rhd P'' \Longrightarrow\hat{}_\tau\ P'$
and P'*RelQ'*: $(\Psi \otimes \Psi',\ P',\ Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *PTrans* ‹$bn\ \alpha\ \sharp*\ \Psi$› ‹$bn\ \alpha\ \sharp*\ P$› **obtain** P''' **where** $\Psi : Q \rhd \tau.(P) \Longrightarrow \alpha \prec P'''$ **and** $\Psi \rhd P'' \sim P'''$
by(*rule weakTransitionTau*)
moreover from ‹$\Psi \rhd P'' \sim P'''$› **have** $\Psi \otimes \Psi' \rhd P'' \sim P'''$ **by**(*rule bisimE*)
with P''*Chain* **obtain** P'''' **where** $\Psi \otimes \Psi' \rhd P''' \Longrightarrow\hat{}_\tau\ P''''$ **and** $\Psi \otimes \Psi' \rhd P' \sim P''''$
by(*rule tauChainBisim*)
ultimately show *?case* **using** ‹$(\Psi \otimes \Psi',\ P',\ Q') \in Rel$› **by**(*metis bisimE C1*)
next
case(*cTau* Q')
from ‹$\Psi \rhd P \rightsquigarrow$<*Rel*> Q› ‹$\Psi \rhd Q \longmapsto \tau \prec Q'$›
obtain P' **where** *PChain*: $\Psi \rhd P \Longrightarrow\hat{}_\tau\ P'$ **and** P'*RelQ'*: $(\Psi,\ P',\ Q') \in Rel$
by(*blast dest*: *weakSimE*)

from *PChain* **obtain** P'' **where** $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau\ P''$ **and** $\Psi \rhd P' \sim P''$
by(*rule tauChainCons*)
thus *?case* **using** P'*RelQ'* **by**(*metis bisimE C1*)
qed

lemma *tauLaw1SimRight*:
fixes $\Psi :: 'b$
and $P :: ('a,\ 'b,\ 'c)\ psi$
and $Q :: ('a,\ 'b,\ 'c)\ psi$

assumes *eqvt Rel*
and $(\Psi,\ P,\ Q) \in Rel$
and *C1*: $\bigwedge \Psi\ P\ Q\ R.\ [\![(\Psi,\ P,\ Q) \in Rel;\ \Psi \rhd Q \sim R]\!] \Longrightarrow (\Psi,\ P,\ R) \in Rel$

shows $\Psi \rhd P \rightsquigarrow$<*Rel*> $\tau.(Q)$
using ‹*eqvt Rel*›
proof(*induct rule*: *weakSimI*[**where** C=Q])
case(*cAct* $\Psi'\ \alpha\ Q'$)
hence *False* **by**(*cases rule*: *actionCases*[**where** α=α]) *auto*
thus *?case* **by** *simp*
next
case(*cTau* Q')
have $\Psi \rhd Q \Longrightarrow\hat{}_\tau\ Q$ **by** *simp*
moreover from ‹$\Psi \rhd \tau.(Q) \longmapsto \tau \prec Q'$› **have** $\Psi \rhd Q \sim Q'$ **by**(*rule tauActionE*)
with ‹$(\Psi,\ P,\ Q) \in Rel$› **have** $(\Psi,\ P,\ Q') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *tauLaw3SimLeft*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and *Q* :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a prefix*

assumes *eqvt Rel*
and $(\Psi, P, Q) \in Rel$
and *Subst*: $\bigwedge$*xvec Tvec*. *length xvec* = *length Tvec* $\Longrightarrow$ $(\Psi, P[xvec::=Tvec], Q[xvec::=Tvec]) \in Rel$
and *C1*: $\bigwedge \Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q; (\Psi, Q, R) \in Rel; \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi, P, S) \in Rel$
and *rExt*: $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in Rel \Longrightarrow (\Psi \otimes \Psi', P, Q) \in Rel$

shows $\Psi \rhd \alpha\cdot(\tau.(P)) \rightsquigarrow <Rel> \alpha\cdot Q$
using ‹*eqvt Rel*›
proof(*induct rule*: *weakSimI*[**where** *C*=*Q*])
case(*cAct* Ψ' β Q')
from ‹$\Psi \rhd \alpha\cdot Q \longmapsto\beta \prec Q'$› ‹$\beta \neq \tau$› **show** *?case*
proof(*induct rule*: *prefixCases*)
case(*cInput M xvec N K Tvec*)
note ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*›
moreover have *insertAssertion* (*extractFrame*($M(\!|\lambda{*}xvec\ N|\!).Q$)) $\Psi \hookrightarrow_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$ **by** *auto*
ultimately have $\Psi : (M(\!|\lambda{*}xvec\ N|\!).Q) \rhd (M(\!|\lambda{*}xvec\ N|\!).(\tau.(P))) \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P))[xvec::=Tvec]$
by(*rule weakInput*)
hence $\Psi : (M(\!|\lambda{*}xvec\ N|\!).Q) \rhd (M(\!|\lambda{*}xvec\ N|\!).(\tau.(P))) \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec \tau.(P[xvec::=Tvec])$
using ‹*length xvec* = *length Tvec*› ‹*distinct xvec*›
by *simp*

moreover obtain P' **where** *PTrans*: $\Psi \otimes \Psi' \rhd \tau.(P[xvec::=Tvec]) \longmapsto\tau \prec P'$ **and** $\Psi \otimes \Psi' \rhd (P[xvec::=Tvec]) \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \otimes \Psi' \rhd \tau.(P[xvec::=Tvec]) \Longrightarrow\hat{}_\tau\ P'$ **by**(*rule tauActTauChain*)
moreover from ‹*length xvec* = *length Tvec*› **have** $(\Psi, P[xvec::=Tvec], Q[xvec::=Tvec]) \in Rel$ **by**(*rule Subst*)
hence $(\Psi \otimes \Psi', P[xvec::=Tvec], Q[xvec::=Tvec]) \in Rel$ **by**(*rule rExt*)
with ‹$\Psi \otimes \Psi' \rhd P[xvec::=Tvec] \sim P'$› **have** $(\Psi \otimes \Psi', P', Q[xvec::=Tvec]) \in Rel$ **by**(*blast intro*: *bisimE C1 bisimReflexive*)
ultimately show *?case* **by** *fastforce*
next
case(*cOutput M N K*)
note ‹$\Psi \vdash M \leftrightarrow K$›
moreover have *insertAssertion* (*extractFrame* ($M\langle N\rangle.Q$)) $\Psi \hookrightarrow_F \langle\varepsilon, \Psi \otimes \mathbf{1}\rangle$ **by** *auto*

ultimately have Ψ : $M\langle N\rangle.Q \rhd M\langle N\rangle.(\tau.(P)) \Longrightarrow K\langle N\rangle \prec \tau.(P)$ **by**(*rule weakOutput*)
moreover obtain P' **where** *PTrans*: $\Psi \otimes \Psi' \rhd \tau.(P) \longmapsto\tau \prec P'$ **and** $\Psi \otimes \Psi' \rhd P \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \otimes \Psi' \rhd \tau.(P) \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauActTauChain*)
moreover from ‹$(\Psi, P, Q) \in Rel$› **have** $(\Psi \otimes \Psi', P, Q) \in Rel$ **by**(*rule rExt*)
with ‹$\Psi \otimes \Psi' \rhd P \sim P'$› **have** $(\Psi \otimes \Psi', P', Q) \in Rel$ **by**(*blast intro*: *bisimE C1 bisimReflexive*)
ultimately show *?case* **by** *fastforce*
next
case *cTau*
with ‹$\tau \neq \tau$› **show** *?case*
by *simp*
qed
next
case(*cTau* Q')
from ‹$\Psi \rhd \alpha\cdot Q \longmapsto\tau \prec Q'$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *tPTrans*: $\Psi \rhd \tau.(\tau.(P)) \longmapsto\tau \prec P'$ **and** $\Psi \rhd \tau.(P) \sim P'$ **using** *tauActionI*
by *auto*
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
from *PTrans* ‹$\Psi \rhd \tau.(P) \sim P'$› **obtain** P''' **where** *P'Trans*: $\Psi \rhd P' \longmapsto\tau \prec P'''$ **and** $\Psi \rhd P'' \sim P'''$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))
apply(*drule-tac simE*, *auto*)
by(*metis bisimE*)
from *tPTrans P'Trans* **have** $\Psi \rhd \tau.(\tau.(P)) \Longrightarrow\hat{}_\tau P'''$ **by**(*fastforce dest*: *tauAct-TauChain*)
moreover from ‹$(\Psi, P, Q) \in Rel$› ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P'' \sim P'''$› ‹$\Psi \rhd Q \sim Q'$› **have** $(\Psi, P''', Q') \in Rel$
by(*metis bisimTransitive C1 bisimSymmetric*)
ultimately show *?case* **by** *fastforce*
qed
qed

lemma *tauLaw3SimRight*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes *eqvt Rel*
and *Subst*: $\bigwedge\Psi$ *xvec Tvec*. *length xvec* = *length Tvec* $\Longrightarrow (\Psi, P[xvec::=Tvec], \tau.(Q[xvec::=Tvec])) \in Rel$

and *C1*: $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in Rel;\ \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi,\ P,\ S) \in Rel$
and $\bigwedge\Psi.\ (\Psi,\ P,\ \tau.(Q)) \in Rel$

shows $\Psi \rhd \alpha{\cdot}P \rightsquigarrow\!<Rel>\ \alpha{\cdot}(\tau.(Q))$
using ‹*eqvt Rel*›
proof(*induct rule*: *weakSimI*[**where** $C{=}Q$])
case(*cAct* $\Psi'\ \beta\ Q'$)
from ‹$\Psi \rhd \alpha{\cdot}(\tau.(Q)) \longmapsto\beta \prec Q'$› ‹$\beta \neq \tau$›
show *?case*
proof(*induct rule*: *prefixCases*)
case(*cInput M xvec N K Tvec*)
note ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec=length Tvec*›
moreover have *insertAssertion* (*extractFrame* ($M(\!|\lambda{*}xvec\ N|\!).(\tau.(Q))$)) $\Psi \hookrightarrow_F \langle\varepsilon,\ \Psi \otimes \mathbf{1}\rangle$ **by** *auto*
ultimately have $\Psi : (M(\!|\lambda{*}xvec\ N|\!).(\tau.(Q))) \rhd M(\!|\lambda{*}xvec\ N|\!).P \Longrightarrow K(\!|(N[xvec{::=}Tvec])|\!) \prec P[xvec{::=}Tvec]$
by(*rule weakInput*)
moreover have $\Psi \otimes \Psi' \rhd P[xvec{::=}Tvec] \Longrightarrow\hat{}_\tau\ P[xvec{::=}Tvec]$ **by** *auto*
ultimately show *?case* **using** *Subst* ‹*length xvec=length Tvec*› ‹*distinct xvec*›
by *fastforce*
next
case(*cOutput M N K*)
note ‹$\Psi \vdash M \leftrightarrow K$›
moreover have *insertAssertion* (*extractFrame* ($M\langle N\rangle.(\tau.(Q))$)) $\Psi \hookrightarrow_F \langle\varepsilon,\ \Psi \otimes \mathbf{1}\rangle$ **by** *auto*
ultimately have $\Psi : M\langle N\rangle.(\tau.(Q)) \rhd M\langle N\rangle.P \Longrightarrow K\langle N\rangle \prec P$ **by**(*rule weakOutput*)
moreover have $\Psi \otimes \Psi' \rhd P \Longrightarrow\hat{}_\tau\ P$ **by** *auto*
ultimately show *?case* **using** ‹$(\Psi \otimes \Psi',\ P,\ \tau.(Q)) \in Rel$› **by** *fastforce*
next
case *cTau*
with ‹$\tau \neq \tau$› **show** *?case* **by** *simp*
qed
next
case(*cTau* Q')
from ‹$\Psi \rhd \alpha{\cdot}(\tau.(Q)) \longmapsto\tau \prec Q'$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P'$ **and** $\Psi \rhd P \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau\ P'$ **by**(*rule tauActTauChain*)
moreover from ‹$\Psi \rhd P \sim P'$› ‹$\Psi \rhd \tau.(Q) \sim Q'$› ‹$(\Psi,\ P,\ \tau.(Q)) \in Rel$›
have $(\Psi,\ P',\ Q') \in Rel$ **by**(*metis bisimTransitive bisimSymmetric C1*)
ultimately show *?case* **by** *fastforce*
qed
qed

lemma *tauLaw3CongSimLeft*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $(\Psi,\ P,\ Q) \in Rel$
and *C1*: $\bigwedge \Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in Rel;\ \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi,\ P,\ S) \in Rel$
and *rExt*: $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi,\ P,\ Q) \in Rel \Longrightarrow (\Psi \otimes \Psi',\ P,\ Q) \in Rel$

shows $\Psi \rhd \alpha\cdot(\tau.(P)) \rightsquigarrow$«*Rel*» $\alpha\cdot Q$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* Q')
from ‹$\Psi \rhd \alpha\cdot Q \longmapsto\tau \prec Q'$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *tPTrans*: $\Psi \rhd \tau.(\tau.(P)) \longmapsto\tau \prec P'$ **and** $\Psi \rhd \tau.(P) \sim P'$ **using** *tauActionI*
by *auto*
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
from *PTrans* ‹$\Psi \rhd \tau.(P) \sim P'$› **obtain** P''' **where** *P'Trans*: $\Psi \rhd P' \longmapsto\tau \prec P'''$ **and** $\Psi \rhd P'' \sim P'''$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))
apply(*drule-tac simE*, *auto*)
by(*metis bisimE*)
from *tPTrans P'Trans* **have** $\Psi \rhd \tau.(\tau.(P)) \Longrightarrow_\tau P'''$ **by**(*fastforce dest*: *tauActTauStepChain*)
moreover from ‹$(\Psi,\ P,\ Q) \in Rel$› ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P'' \sim P'''$› ‹$\Psi \rhd Q \sim Q'$› **have** $(\Psi,\ P''',\ Q') \in Rel$
by(*metis bisimTransitive C1 bisimSymmetric*)
ultimately show *?case* **by** *fastforce*
qed
qed

lemma *tauLaw3CongSimRight*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*

assumes $(\Psi,\ P,\ Q) \in Rel$
and *C1*: $\bigwedge \Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in Rel;\ \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi,\ P,\ S) \in Rel$
and $\bigwedge \Psi.\ (\Psi,\ P,\ \tau.(Q)) \in Rel$

shows $\Psi \rhd \alpha\cdot P \rightsquigarrow$«*Rel*» $\alpha\cdot(\tau.(Q))$

proof(*induct rule*: *weakCongSimI*)
  case(*cTau* Q')
  from ‹$\Psi \rhd \alpha\cdot(\tau.(Q)) \longmapsto\tau \prec Q'$› **show** *?case*
  proof(*induct rule*: *prefixTauCases*)
    case *cTau*
    obtain P' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P'$ **and** $\Psi \rhd P \sim P'$ **using** *tauActionI*
      by *auto*
    from *PTrans* **have** $\Psi \rhd \tau.(P) \Longrightarrow_\tau P'$ **by**(*rule tauActTauStepChain*)
    moreover from ‹$\Psi \rhd P \sim P'$› ‹$\Psi \rhd \tau.(Q) \sim Q'$› ‹$(\Psi, P, \tau.(Q)) \in Rel$›
    have $(\Psi, P', Q') \in Rel$ **by**(*metis bisimTransitive bisimSymmetric C1*)
    ultimately show *?case* **by** *fastforce*
  qed
qed

end

locale *tauSum* = *tau* + *sum*
begin

lemma *tauLaw2SimLeft*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*

  assumes *Id*: $\bigwedge\Psi\ P.\ (\Psi, P, P) \in Rel$
  and *C1*: $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q; (\Psi, Q, R) \in Rel; \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi, P, S) \in Rel$

  shows $\Psi \rhd P \oplus \tau.(P) \rightsquigarrow$<*Rel*> $\tau.(P)$
proof(*induct rule*: *weakSimI2*)
  case(*cAct* $\Psi'\ \alpha\ P'$)
  thus *?case* **by**(*nominal-induct* α *rule*: *action.strong-inducts*) *auto*
next
  case(*cTau* P')
  from ‹$\Psi \rhd \tau.(P) \longmapsto\tau \prec P'$› **have** $\Psi \rhd P \sim P'$ **by**(*rule tauActionE*)
  obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
    by *auto*
  have *guarded*($\tau.(P)$) **by**(*rule guardedTau*)
  with *PTrans* **have** $\Psi \rhd P \oplus \tau.(P) \longmapsto\tau \prec P''$ **by**(*rule Sum2*)
  hence $\Psi \rhd P \oplus \tau.(P) \Longrightarrow\hat{}_\tau P''$ **by**(*rule tauActTauChain*)
  moreover from ‹$\Psi \rhd P \sim P''$› ‹$(\Psi, P, P) \in Rel$› ‹$\Psi \rhd P \sim P'$› **have** $(\Psi, P'', P') \in Rel$
    by(*metis C1 bisimE*)
  ultimately show *?case* **by** *blast*
qed

lemma *tauLaw2CongSimLeft*:
  fixes Ψ :: $'b$

and P :: ($'a$, $'b$, $'c$) *psi*

assumes *Id*: $\bigwedge\Psi\ P.\ (\Psi,\ P,\ P) \in Rel$
and *C1*: $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in Rel;\ \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi,\ P,\ S) \in Rel$

shows $\Psi \rhd P \oplus \tau.(P) \rightsquigarrow$«*Rel*» $\tau.(P)$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* P')
from ‹$\Psi \rhd \tau.(P) \longmapsto\tau \prec P'$› **have** $\Psi \rhd P \sim P'$ **by**(*rule tauActionE*)
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
have *guarded*($\tau.(P)$) **by**(*rule guardedTau*)
with *PTrans* **have** $\Psi \rhd P \oplus \tau.(P) \longmapsto\tau \prec P''$ **by**(*rule Sum2*)
hence $\Psi \rhd P \oplus \tau.(P) \Longrightarrow_\tau P''$ **by**(*rule tauActTauStepChain*)
moreover from ‹$\Psi \rhd P \sim P''$› ‹$(\Psi,\ P,\ P) \in Rel$› ‹$\Psi \rhd P \sim P'$› **have** $(\Psi,\ P'',\ P') \in Rel$
by(*metis C1 bisimE*)
ultimately show *?case* **by** *blast*
qed

lemma *tauLaw2SimRight*:
fixes Ψ :: $'b$
and P :: ($'a$, $'b$, $'c$) *psi*

assumes *C1*: $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi,\ P,\ Q) \in Rel$

shows $\Psi \rhd \tau.(P) \rightsquigarrow$<*Rel*> $P \oplus \tau.(P)$
proof(*induct rule*: *weakSimI2*)
case(*cAct* $\Psi'\ \alpha\ P'$)
from ‹$\Psi \rhd P \oplus \tau.(P) \longmapsto\alpha \prec P'$›
show *?case*
proof(*induct rule*: *sumCases*)
case *cSum1*
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau P''$ **by**(*rule tauActTauChain*)
moreover from ‹*guarded P*› **have** *insertAssertion* (*extractFrame P*) $\Psi \simeq_F \langle\varepsilon,\ \Psi \otimes \mathbf{1}\rangle$
by(*rule insertGuardedAssertion*)
with ‹$\Psi \rhd P \sim P''$› **have** *insertAssertion* (*extractFrame* P'') $\Psi \simeq_F \langle\varepsilon,\ \Psi \otimes \mathbf{1}\rangle$
by(*metis bisimE FrameStatEqTrans FrameStatEqSym*)
hence *insertAssertion* (*extractFrame*($P \oplus \tau.(P)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* P'') Ψ
by(*simp add*: *FrameStatEq-def*)
moreover from *PTrans* ‹*bn* $\alpha \sharp*$ ($\tau.(P)$)› **have** *bn* $\alpha \sharp*$ P'' **by**(*rule tauFreshChain-*

Derivative)
with ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P \longmapsto\alpha \prec P'$› ‹$bn\ \alpha \sharp* \Psi$›
obtain P''' **where** $P''Trans$: $\Psi \rhd P'' \longmapsto\alpha \prec P'''$ **and** $\Psi \rhd P''' \sim P'$
by(*metis bisimE simE*)
ultimately have $\Psi : (P \oplus \tau.(P)) \rhd \tau.(P) \Longrightarrow\alpha \prec P'''$
by(*rule-tac weakTransitionI*)
moreover have $\Psi \otimes \Psi' \rhd P''' \Longrightarrow\hat{}_\tau\ P'''$ **by** *auto*
moreover from ‹$\Psi \rhd P''' \sim P'$› **have** $\Psi \otimes \Psi' \rhd P''' \sim P'$ **by**(*rule bisimE*)
hence $(\Psi \otimes \Psi', P''', P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case *cSum2*
thus *?case* **using** ‹$\alpha \neq \tau$›
by(*nominal-induct* α *rule*: *action.strong-inducts*) *auto*
qed
next
case(*cTau* P')
from ‹$\Psi \rhd P \oplus \tau.(P) \longmapsto\tau \prec P'$›
show *?case*
proof(*induct rule*: *sumCases*)
case *cSum1*
obtain P'' **where** $PTrans$: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
moreover from ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P \longmapsto\tau \prec P'$›
obtain P''' **where** $P''Trans$: $\Psi \rhd P'' \longmapsto\tau \prec P'''$ **and** $\Psi \rhd P''' \sim P'$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))
by(*drule-tac simE*, *auto*)
ultimately have $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau\ P'''$ **by**(*auto dest*: *tauActTauChain rtrancl-into-rtrancl*)
moreover from ‹$\Psi \rhd P''' \sim P'$› **have** $(\Psi, P''', P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case *cSum2*
from ‹$\Psi \rhd \tau.(P) \longmapsto\tau \prec P'$› **have** $\Psi \rhd P \sim P'$ **by**(*rule tauActionE*)
obtain P'' **where** $PTrans$: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
hence $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau\ P''$ **by**(*rule-tac tauActTauChain*)
moreover from ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P \sim P'$› **have** $(\Psi, P'', P') \in Rel$
by(*metis C1 bisimTransitive bisimE*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *tauLaw2CongSimRight*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

assumes *C1*: $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi,\ P,\ Q) \in Rel$

shows $\Psi \rhd \tau.(P) \rightsquigarrow$«*Rel*» $P \oplus \tau.(P)$
proof(*induct rule*: *weakCongSimI*)
case(*cTau* P')
from ‹$\Psi \rhd P \oplus \tau.(P) \longmapsto\tau \prec P'$›
show *?case*
proof(*induct rule*: *sumCases*)
case *cSum1*
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
moreover from ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P \longmapsto\tau \prec P'$›
obtain P''' **where** P''*Trans*: $\Psi \rhd P'' \longmapsto\tau \prec P'''$ **and** $\Psi \rhd P''' \sim P'$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))
by(*drule-tac simE*) *auto*
ultimately have $\Psi \rhd \tau.(P) \Longrightarrow_\tau P'''$ **by**(*auto dest*: *tauActTauStepChain trancl-into-trancl*)
moreover from ‹$\Psi \rhd P''' \sim P'$› **have** $(\Psi,\ P''',\ P') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
next
case *cSum2*
from ‹$\Psi \rhd \tau.(P) \longmapsto\tau \prec P'$› **have** $\Psi \rhd P \sim P'$ **by**(*rule tauActionE*)
obtain P'' **where** *PTrans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
hence $\Psi \rhd \tau.(P) \Longrightarrow_\tau P''$ **by**(*rule-tac tauActTauStepChain*)
moreover from ‹$\Psi \rhd P \sim P''$› ‹$\Psi \rhd P \sim P'$› **have** $(\Psi,\ P'',\ P') \in Rel$
by(*metis C1 bisimTransitive bisimE*)
ultimately show *?case* **by** *blast*
qed
qed

lemma *tauLaw4SimLeft*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and M :: $'a$
and N :: $'a$

assumes $\bigwedge\Psi\ P.\ (\Psi,\ P,\ P) \in Rel$
and *C1*: $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi,\ P,\ Q) \in Rel$

shows $\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \rightsquigarrow$<*Rel*> $\alpha\cdot(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakSimI2*)
case(*cAct* $\Psi'\ \beta\ PQ$)
from ‹$\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \longmapsto\beta \prec PQ$› ‹$\beta \neq \tau$›
show *?case*

proof(*induct rule*: *prefixCases*)
case(*cInput M xvec N K Tvec*)
have $\Psi \rhd \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($\alpha{\cdot}(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q)$)) Ψ
using *insertTauAssertion Identity*
by(*nominal-induct* α *rule*: *prefix.strong-inducts*, *auto*)
(*rule FrameStatImpTrans*[**where** G=$\langle\varepsilon, \Psi\rangle$], *auto simp add*: *FrameStatEq-def AssertionStatEq-def*)
moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec*=*length Tvec*›
have $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P) \oplus Q)[xvec::=Tvec]$
by(*rule Input*)
hence $\Psi \rhd (M(\!|\lambda{*}xvec\ N|\!).P) \oplus M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P) \oplus Q)[xvec::=Tvec]$
by(*rule-tac Sum2*) *auto*
ultimately have $\Psi : (M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q)) \rhd (M(\!|\lambda{*}xvec\ N|\!).P) \oplus M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P) \oplus Q)[xvec::=Tvec]$
by(*rule-tac weakTransitionI*) *auto*
moreover have $\Psi \otimes \Psi' \rhd (\tau.(P) \oplus Q)[xvec::=Tvec] \Longrightarrow\hat{}_\tau\ (\tau.(P) \oplus Q)[xvec::=Tvec]$ **by** *auto*
ultimately show *?case* **using** ‹$(\Psi \otimes \Psi', (\tau.(P) \oplus Q)[xvec::=Tvec], (\tau.(P) \oplus Q)[xvec::=Tvec]) \in Rel$› **by** *fastforce*
next
case(*cOutput M N K*)
have $\Psi \rhd \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($\alpha{\cdot}(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q)$)) Ψ
using *insertTauAssertion Identity*
by(*nominal-induct* α *rule*: *prefix.strong-inducts*, *auto*)
(*rule FrameStatImpTrans*[**where** G=$\langle\varepsilon, \Psi\rangle$], *auto simp add*: *FrameStatEq-def AssertionStatEq-def*)

moreover from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \longmapsto K\langle N\rangle \prec (\tau.(P) \oplus Q)$
by(*rule Output*)
hence $\Psi \rhd M\langle N\rangle.P \oplus M\langle N\rangle.(\tau.(P) \oplus Q) \longmapsto K\langle N\rangle \prec (\tau.(P) \oplus Q)$ **by**(*rule-tac Sum2*) *auto*
ultimately have $\Psi : (M\langle N\rangle.(\tau.(P) \oplus Q)) \rhd M\langle N\rangle.P \oplus M\langle N\rangle.(\tau.(P) \oplus Q) \Longrightarrow K\langle N\rangle \prec (\tau.(P) \oplus Q)$
by(*rule-tac weakTransitionI*) *auto*
moreover have $\Psi \otimes \Psi' \rhd \tau.(P) \oplus Q \Longrightarrow\hat{}_\tau\ \tau.(P) \oplus Q$ **by** *auto*
ultimately show *?case* **using** ‹$(\Psi \otimes \Psi', \tau.(P) \oplus Q, \tau.(P) \oplus Q) \in Rel$› **by** *fastforce*
next
case *cTau*
from ‹$\tau \neq \tau$› **show** *?case*
by *simp*

qed
next
case(*cTau Q'*)
from ‹$\Psi \rhd \alpha\cdot(\tau.(P)) \oplus_{\top} Q \longmapsto \tau \prec Q'$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.(\tau.(P) \oplus Q) \longmapsto\tau \prec P'$ **and** $\Psi \rhd \tau.(P) \oplus Q \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \rhd (\tau.(P)) \oplus \tau.(\tau.(P) \oplus Q) \longmapsto\tau \prec P'$ **by**(*rule-tac Sum2*) (*auto intro*: *guardedTau*)
hence $\Psi \rhd (\tau.(P)) \oplus \tau.(\tau.(P) \oplus Q) \Longrightarrow\hat{}_{\tau} P'$ **by**(*rule tauActTauChain*)
moreover from ‹$\Psi \rhd \tau.(P) \oplus Q \sim P'$› ‹$\Psi \rhd (\tau.(P)) \oplus Q \sim Q'$› **have** $\Psi \rhd P' \sim Q'$ **by**(*metis bisimSymmetric bisimTransitive*)
hence $(\Psi, P', Q') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *fastforce*
qed
qed

lemma *tauLaw4CongSimLeft*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $Q :: ('a, 'b, 'c)\ psi$
and $M :: 'a$
and $N :: 'a$

assumes $\bigwedge\Psi\ P.\ (\Psi, P, P) \in Rel$
and $C1$: $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi, P, Q) \in Rel$

shows $\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \rightsquigarrow$«*Rel*» $\alpha\cdot(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakCongSimI*)
case(*cTau Q'*)
from ‹$\Psi \rhd \alpha\cdot(\tau.(P)) \oplus_{\top} Q \longmapsto \tau \prec Q'$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.(\tau.(P) \oplus Q) \longmapsto\tau \prec P'$ **and** $\Psi \rhd \tau.(P) \oplus Q \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \rhd (\tau.(P)) \oplus \tau.(\tau.(P) \oplus Q) \longmapsto\tau \prec P'$ **by**(*rule-tac Sum2*) (*auto intro*: *guardedTau*)
hence $\Psi \rhd (\tau.(P)) \oplus \tau.(\tau.(P) \oplus Q) \Longrightarrow_{\tau} P'$ **by**(*rule tauActTauStepChain*)
moreover from ‹$\Psi \rhd \tau.(P) \oplus Q \sim P'$› ‹$\Psi \rhd (\tau.(P)) \oplus Q \sim Q'$› **have** $\Psi \rhd P' \sim Q'$ **by**(*metis bisimSymmetric bisimTransitive*)
hence $(\Psi, P', Q') \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *fastforce*
qed
qed

lemma *tauLaw4SimRight*:

fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and Q :: $('a,\ 'b,\ 'c)$ *psi*
and α :: $'a$ *prefix*

assumes *C1*: $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi,\ P,\ Q) \in Rel$
and $\bigwedge\Psi\ P.\ (\Psi,\ P,\ P) \in Rel$

shows $\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \rightsquigarrow\!<Rel>\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakSimI2*)
case(*cAct* Ψ' β PQ)
from ‹$\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \longmapsto\beta \prec PQ$› **show** *?case*
proof(*induct rule*: *sumCases*)
case *cSum1*
from ‹$\Psi \rhd \alpha\cdot P \longmapsto\beta \prec PQ$› ‹$\beta \neq \tau$› **show** *?case*
proof(*induct rule*: *prefixCases*)
case(*cInput* M *xvec* N K *Tvec*)
have $\Psi \rhd M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($(M(\!|\lambda*xvec\ N|\!).P) \oplus M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q)$)) Ψ
by *auto*
moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp* N› ‹*length xvec* = *length Tvec*›
have $\Psi \rhd M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P) \oplus Q)[xvec::=Tvec]$ **by**(*rule Input*)
ultimately have Ψ : $((M(\!|\lambda*xvec\ N|\!).P) \oplus M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q)) \rhd M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q) \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P) \oplus Q)[xvec::=Tvec]$
by(*rule-tac weakTransitionI*) *auto*
with ‹*length xvec* = *length Tvec*› ‹*distinct xvec*›
have Ψ : $((M(\!|\lambda*xvec\ N|\!).P) \oplus M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q)) \rhd M(\!|\lambda*xvec\ N|\!).(\tau.(P) \oplus Q) \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P[xvec::=Tvec]) \oplus Q[xvec::=Tvec])$
by *auto*
moreover obtain P' **where** *PTrans*: $\Psi \otimes \Psi' \rhd \tau.(P[xvec::=Tvec]) \longmapsto\tau \prec P'$ **and** $\Psi \otimes \Psi' \rhd P[xvec::=Tvec] \sim P'$ **using** *tauActionI*
by *auto*
have *guarded*($\tau.(P[xvec::=Tvec])$) **by**(*rule guardedTau*)
with *PTrans* **have** $\Psi \otimes \Psi' \rhd (\tau.(P[xvec::=Tvec])) \oplus (Q[xvec::=Tvec]) \longmapsto\tau \prec P'$ **by**(*rule Sum1*)
hence $\Psi \otimes \Psi' \rhd (\tau.(P[xvec::=Tvec])) \oplus (Q[xvec::=Tvec]) \Longrightarrow\hat{}_\tau\ P'$ **by**(*rule tauActTauChain*)
moreover from ‹$\Psi \otimes \Psi' \rhd P[xvec::=Tvec] \sim P'$› **have** $(\Psi \otimes \Psi',\ P',\ P[xvec::=Tvec]) \in Rel$ **by**(*metis bisimE C1*)
ultimately show *?case* **by** *fastforce*
next
case(*cOutput* M N K)
have $\Psi \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ M\langle N\rangle.(\tau.(P) \oplus Q)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($M\langle N\rangle.P \oplus M\langle N\rangle.(\tau.(P) \oplus$

$Q))) \ \Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M\langle N\rangle.(\tau.(P) \oplus Q)$)) Ψ
by *auto*
moreover from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \longmapsto K\langle N\rangle \prec (\tau.(P) \oplus Q)$ **by**(*rule Output*)
ultimately have $\Psi : (M\langle N\rangle.P \oplus M\langle N\rangle.(\tau.(P) \oplus Q)) \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \Longrightarrow K\langle N\rangle \prec (\tau.(P) \oplus Q)$
by(*rule-tac weakTransitionI*) *auto*
moreover obtain P' **where** *PTrans*: $\Psi \otimes \Psi' \rhd \tau.(P) \longmapsto \tau \prec P'$ **and** $\Psi \otimes \Psi' \rhd P \sim P'$ **using** *tauActionI*
by *auto*
have *guarded*($\tau.(P)$) **by**(*rule guardedTau*)
with *PTrans* **have** $\Psi \otimes \Psi' \rhd (\tau.(P)) \oplus Q \longmapsto \tau \prec P'$ **by**(*rule Sum1*)
hence $\Psi \otimes \Psi' \rhd (\tau.(P)) \oplus Q \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauActTauChain*)
moreover from ‹$\Psi \otimes \Psi' \rhd P \sim P'$› **have** $(\Psi \otimes \Psi', P', P) \in$ *Rel* **by**(*metis bisimE C1*)
ultimately show *?case* **by** *fastforce*
next
case *cTau*
from ‹$\tau \neq \tau$› **show** *?case* **by** *simp*
qed
next
case *cSum2*
from ‹$\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \longmapsto \beta \prec PQ$› ‹$\beta \neq \tau$› **show** *?case*
proof(*induct rule*: *prefixCases*)
case(*cInput M xvec N K Tvec*)
have $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($(M(\!|\lambda{*}xvec\ N|\!).P) \oplus M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q)$)) Ψ
by *auto*
moreover from ‹$\Psi \vdash M \leftrightarrow K$› ‹*distinct xvec*› ‹*set xvec* $\subseteq$ *supp N*› ‹*length xvec* = *length Tvec*›
have $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P) \oplus Q)[xvec::=Tvec]$
by(*rule Input*)
with ‹*length xvec* = *length Tvec*› ‹*distinct xvec*›
have $\Psi \rhd M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \longmapsto K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P[xvec::=Tvec]) \oplus Q[xvec::=Tvec])$
by *simp*
ultimately have $\Psi : ((M(\!|\lambda{*}xvec\ N|\!).P) \oplus M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q)) \rhd M(\!|\lambda{*}xvec\ N|\!).(\tau.(P) \oplus Q) \Longrightarrow K(\!|(N[xvec::=Tvec])|\!) \prec (\tau.(P[xvec::=Tvec]) \oplus Q[xvec::=Tvec])$
by(*rule-tac weakTransitionI*) *auto*
moreover have $\Psi \otimes \Psi' \rhd \tau.(P[xvec::=Tvec]) \oplus (Q[xvec::=Tvec]) \Longrightarrow\hat{}_\tau \tau.(P[xvec::=Tvec]) \oplus (Q[xvec::=Tvec])$ **by** *auto*
ultimately show *?case* **using** ‹$(\Psi \otimes \Psi', \tau.(P[xvec::=Tvec]) \oplus (Q[xvec::=Tvec]), \tau.(P[xvec::=Tvec]) \oplus (Q[xvec::=Tvec])) \in$ *Rel*› ‹*length xvec* = *length Tvec*› ‹*distinct xvec*›

by *fastforce*
next
case(*cOutput M N K*)
have $\Psi \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau M\langle N\rangle.(\tau.(P) \oplus Q)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($M\langle N\rangle.P \oplus M\langle N\rangle.(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($M\langle N\rangle.(\tau.(P) \oplus Q)$)) Ψ
by *auto*
moreover from ‹$\Psi \vdash M \leftrightarrow K$› **have** $\Psi \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \longmapsto K\langle N\rangle \prec (\tau.(P) \oplus Q)$
by(*rule Output*)
ultimately have $\Psi : (M\langle N\rangle.P \oplus M\langle N\rangle.(\tau.(P) \oplus Q)) \rhd M\langle N\rangle.(\tau.(P) \oplus Q) \Longrightarrow K\langle N\rangle \prec (\tau.(P) \oplus Q)$
by(*rule-tac weakTransitionI*) *auto*
moreover have $\Psi \otimes \Psi' \rhd \tau.(P) \oplus Q \Longrightarrow\hat{}_\tau \tau.(P) \oplus Q$ **by** *auto*
ultimately show *?case* **using** ‹$(\Psi \otimes \Psi', \tau.(P) \oplus Q, \tau.(P) \oplus Q) \in Rel$› **by** *fastforce*
next
case *cTau*
from ‹$\tau \neq \tau$› **show** *?case* **by** *simp*
qed
qed
next
case(*cTau PQ*)
from ‹$\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \longmapsto \tau \prec PQ$› **show** *?case*
proof(*induct rule*: *sumCases*)
case *cSum1*
from ‹$\Psi \rhd \alpha\cdot P \longmapsto \tau \prec PQ$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \longmapsto \tau \prec P'$ **and** $\Psi \rhd (\tau.(P)) \oplus Q \sim P'$ **using** *tauActionI*
by *auto*
obtain P'' **where** *P'Trans*: $\Psi \rhd \tau.(P) \longmapsto \tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
from *P'Trans* **have** $\Psi \rhd \tau.(P) \oplus Q \longmapsto \tau \prec P''$ **by**(*rule-tac Sum1*) (*auto intro*: *guardedTau*)
with ‹$\Psi \rhd (\tau.(P)) \oplus Q \sim P'$› **obtain** P''' **where** *P''Trans*: $\Psi \rhd P' \longmapsto \tau \prec P'''$ **and** $\Psi \rhd P'' \sim P'''$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))
apply(*drule-tac simE*)
by(*auto dest*: *bisimE*)
from *PTrans P''Trans* **have** $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \Longrightarrow\hat{}_\tau P'''$ **by**(*fastforce dest*: *tauActTauChain*)
moreover from ‹$\Psi \rhd P \sim PQ$› ‹$\Psi \rhd P'' \sim P'''$› ‹$\Psi \rhd P \sim P''$› **have** $\Psi \rhd P''' \sim PQ$
by(*metis bisimSymmetric bisimTransitive*)
hence $(\Psi, P''', PQ) \in Rel$ **by**(*rule C1*)

ultimately show *?case* **by** *fastforce*
qed
next
case *cSum2*
from ‹$\Psi \rhd \alpha\cdot((\tau.(P)) \oplus Q) \longmapsto\tau \prec PQ$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \longmapsto\tau \prec P'$ **and** $\Psi \rhd (\tau.(P)) \oplus Q \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \Longrightarrow\hat{}_\tau P'$ **by**(*rule tauActTauChain*)
moreover from ‹$\Psi \rhd (\tau.(P)) \oplus Q \sim P'$› ‹$\Psi \rhd \tau.(P) \oplus Q \sim PQ$› **have** $\Psi \rhd P' \sim PQ$
by(*metis bisimSymmetric bisimTransitive*)
hence $(\Psi, P', PQ) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *fastforce*
qed
qed
qed

lemma *tauLaw4CongSimRight*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *prefix*

assumes *C1*: $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi, P, Q) \in Rel$

shows $\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \rightsquigarrow$«*Rel*»$\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakCongSimI*)
case(*cTau PQ*)
from ‹$\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \longmapsto\tau \prec PQ$› **show** *?case*
proof(*induct rule*: *sumCases*)
case *cSum1*
from ‹$\Psi \rhd \alpha\cdot P \longmapsto \tau \prec PQ$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \longmapsto\tau \prec P'$ **and** $\Psi \rhd (\tau.(P)) \oplus Q \sim P'$ **using** *tauActionI*
by *auto*
obtain P'' **where** *P'Trans*: $\Psi \rhd \tau.(P) \longmapsto\tau \prec P''$ **and** $\Psi \rhd P \sim P''$ **using** *tauActionI*
by *auto*
from *P'Trans* **have** $\Psi \rhd \tau.(P) \oplus Q\longmapsto\tau \prec P''$ **by**(*rule-tac Sum1*) (*auto intro*: *guardedTau*)
with ‹$\Psi \rhd (\tau.(P)) \oplus Q \sim P'$› **obtain** P''' **where** *P''Trans*: $\Psi \rhd P' \longmapsto\tau \prec P'''$ **and** $\Psi \rhd P'' \sim P'''$
apply(*drule-tac bisimE*(*4*))
apply(*drule-tac bisimE*(*2*))

apply(*drule-tac simE*)
by(*auto dest*: *bisimE*)
from *PTrans P''Trans* **have** $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \Longrightarrow_\tau P'''$ **by**(*fastforce dest*: *tauActTauStepChain*)
moreover from ‹$\Psi \rhd P \sim PQ$› ‹$\Psi \rhd P'' \sim P'''$› ‹$\Psi \rhd P \sim P''$› **have** $\Psi \rhd P''' \sim PQ$
by(*metis bisimSymmetric bisimTransitive*)
hence $(\Psi, P''', PQ) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *fastforce*
qed
next
case *cSum2*
from ‹$\Psi \rhd \alpha\cdot((\tau.(P)) \oplus Q) \longmapsto\tau \prec PQ$› **show** *?case*
proof(*induct rule*: *prefixTauCases*)
case *cTau*
obtain P' **where** *PTrans*: $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \longmapsto\tau \prec P'$ **and** $\Psi \rhd (\tau.(P)) \oplus Q \sim P'$ **using** *tauActionI*
by *auto*
from *PTrans* **have** $\Psi \rhd \tau.((\tau.(P)) \oplus Q) \Longrightarrow_\tau P'$ **by**(*rule tauActTauStepChain*)
moreover from ‹$\Psi \rhd (\tau.(P)) \oplus Q \sim P'$› ‹$\Psi \rhd \tau.(P) \oplus Q \sim PQ$› **have** $\Psi \rhd P' \sim PQ$
by(*metis bisimSymmetric bisimTransitive*)
hence $(\Psi, P', PQ) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *fastforce*
qed
qed
qed

end

end

theory *Tau-Stat-Imp*
imports *Tau-Sim Weaken-Stat-Imp*
begin

locale *weakTauLaws* = *weak* + *tau*
begin

lemma *tauLaw1StatImpLeft*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \lessapprox_w <Rel> Q$
and $(\Psi, \tau.(P), Q) \in Rel$

shows $\Psi \rhd \tau.(P) \lessapprox_w <Rel> Q$
proof −

have $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q$ **by** *simp*
moreover have *insertAssertion* (*extractFrame*(τ.(P))) $\Psi \simeq_F \langle\varepsilon, \Psi\rangle$ **by**(*rule insertTauAssertion*)
hence *insertAssertion* (*extractFrame*(τ.(P))) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q) Ψ
by(*metis FrameStatImpTrans FrameStatEq-def insertAssertionWeaken*)
ultimately show *?thesis* **using** ‹(Ψ, τ.(P), Q) $\in$ *Rel*›
by(*rule weakenStatImpI*)
qed

lemma *tauLaw1StatImpRight*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes $\Psi \rhd P \lessapprox$<*Rel*> Q
and *C1*: $\bigwedge\Psi\ P\ Q\ R.\ [\![(\Psi, P, Q) \in Rel; \Psi \rhd Q \sim R]\!] \Longrightarrow (\Psi, P, R) \in Rel'$

shows $\Psi \rhd P \lessapprox$<*Rel'*> τ.(Q)
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
from ‹$\Psi \rhd P \lessapprox$<*Rel*> Q› **obtain** Q' Q''
where *QChain*: $\Psi \rhd Q \Longrightarrow\hat{}_\tau Q'$ **and** *PImpQ'*: *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q') Ψ
and *Q'Chain*: $\Psi \otimes \Psi' \rhd Q' \Longrightarrow\hat{}_\tau Q''$ **and** *PRelQ''*: ($\Psi \otimes \Psi'$, P, Q'') $\in$ *Rel*
by(*rule weakStatImpE*)

obtain Q''' **where** *QTrans*: $\Psi \rhd \tau.(Q) \longmapsto\tau \prec Q'''$ **and** $\Psi \rhd Q \sim Q'''$ **using** *tauActionI* **by** *auto*

from ‹$\Psi \rhd Q \sim Q'''$› *QChain bisimE*(*2*) **obtain** Q'''' **where** *Q'''Chain*: $\Psi \rhd Q''' \Longrightarrow\hat{}_\tau Q''''$ **and** $\Psi \rhd Q' \sim Q''''$
by(*metis bisimE*(*4*) *simTauChain*)

from *QTrans Q'''Chain* **have** $\Psi \rhd \tau.(Q) \Longrightarrow\hat{}_\tau Q''''$ **by**(*drule-tac tauActTauChain*) *auto*
moreover from ‹$\Psi \rhd Q' \sim Q''''$› **have** *insertAssertion* (*extractFrame* Q') $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'''') Ψ
by(*metis bisimE FrameStatEq-def*)
with *PImpQ'* **have** *insertAssertion* (*extractFrame* P) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame* Q'''') Ψ
by(*rule FrameStatImpTrans*)
moreover from ‹$\Psi \rhd Q' \sim Q''''$› **have** $\Psi \otimes \Psi' \rhd Q' \sim Q''''$ **by**(*rule bisimE*)
then obtain Q''''' **where** *Q''''Chain*: $\Psi \otimes \Psi' \rhd Q'''' \Longrightarrow\hat{}_\tau Q'''''$ **and** $\Psi \otimes \Psi' \rhd Q'' \sim Q'''''$ **using** *Q'Chain bisimE*(*2*)
by(*metis bisimE*(*4*) *simTauChain*)
note *Q''''Chain*
moreover from ‹($\Psi \otimes \Psi'$, P, Q'') $\in$ *Rel*› ‹$\Psi \otimes \Psi' \rhd Q'' \sim Q'''''$› **have** ($\Psi \otimes \Psi'$, P, Q''''') $\in$ *Rel'*

by(*rule C1*)
ultimately show *?case* **by** *blast*
qed

end

context *tau* **begin**

lemma *tauLaw3StatImpLeft*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a prefix*

assumes *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi', \alpha\cdot(\tau.(P)), \alpha\cdot Q) \in Rel$

shows $\Psi \rhd \alpha\cdot(\tau.(P)) \lessapprox\!<Rel>\ \alpha\cdot Q$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd \alpha\cdot Q \Longrightarrow\hat{}_\tau\ \alpha\cdot Q$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($\alpha\cdot(\tau.(P))$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\alpha\cdot Q$)) Ψ **using** *insertTauAssertion*
by(*nominal-induct* α *rule*: *prefix.strong-inducts*) (*auto simp add*: *FrameStatEq-def intro*: *FrameStatImpTrans*)
moreover have $\Psi \otimes \Psi' \rhd \alpha\cdot Q \Longrightarrow\hat{}_\tau\ \alpha\cdot Q$ **by** *auto*
moreover have $(\Psi \otimes \Psi', \alpha\cdot(\tau.(P)), \alpha\cdot Q) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *tauLaw3StatImpRight*:
fixes Ψ :: *'b*
and P :: (*'a*, *'b*, *'c*) *psi*
and Q :: (*'a*, *'b*, *'c*) *psi*

assumes *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi', \alpha\cdot P, \alpha\cdot(\tau.(Q))) \in Rel$

shows $\Psi \rhd \alpha\cdot P \lessapprox\!<Rel>\ \alpha\cdot(\tau.(Q))$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd\ \alpha\cdot(\tau.(Q)) \Longrightarrow\hat{}_\tau\ \alpha\cdot(\tau.(Q))$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($\alpha\cdot P$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\alpha\cdot(\tau.(Q))$)) Ψ **using** *insertTauAssertion*
by(*nominal-induct* α *rule*: *prefix.strong-inducts*) (*auto simp add*: *FrameStatEq-def intro*: *FrameStatImpTrans*)
moreover have $\Psi \otimes \Psi' \rhd\ \alpha\cdot(\tau.(Q)) \Longrightarrow\hat{}_\tau\ \alpha\cdot(\tau.(Q))$ **by** *auto*
moreover have $(\Psi \otimes \Psi', \alpha\cdot P, \alpha\cdot(\tau.(Q))) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

end

context *tauSum* **begin**

lemma *tauLaw2StatImpLeft*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

assumes *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi', P \oplus \tau.(P), \tau.(P)) \in Rel$

shows $\Psi \rhd P \oplus \tau.(P) \lessapprox <Rel> \tau.(P)$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd \tau.(P) \Longrightarrow\hat{}_\tau\ \tau.(P)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($\tau.(P)$)) $\Psi \simeq_F \langle\varepsilon, \Psi\rangle$ **by**(*rule insertTauAssertion*)
hence *insertAssertion* (*extractFrame*($P \oplus \tau.(P)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\tau.(P)$)) Ψ **using** *Identity*
by(*rule-tac FrameStatImpTrans*) (*auto simp add*: *FrameStatEq-def Assertion-StatEq-def*)
moreover have $\Psi \otimes \Psi' \rhd \tau.(P) \Longrightarrow\hat{}_\tau\ \tau.(P)$ **by** *auto*
moreover have $(\Psi \otimes \Psi', P \oplus \tau.(P), \tau.(P)) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *tauLaw2StatImpRight*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

assumes *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi', \tau.(P), P \oplus \tau.(P)) \in Rel$

shows $\Psi \rhd \tau.(P) \lessapprox <Rel> P \oplus \tau.(P)$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd P \oplus \tau.(P) \Longrightarrow\hat{}_\tau\ P \oplus \tau.(P)$ **by** *auto*
moreover have *insertAssertion* (*extractFrame*($\tau.(P)$)) $\Psi \simeq_F \langle\varepsilon, \Psi\rangle$ **by**(*rule insertTauAssertion*)
hence *insertAssertion* (*extractFrame*($\tau.(P)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($P \oplus \tau.(P)$)) Ψ **using** *Identity*
by(*rule-tac FrameStatImpTrans*) (*auto simp add*: *FrameStatEq-def Assertion-StatEq-def*)
moreover have $\Psi \otimes \Psi' \rhd P \oplus \tau.(P) \Longrightarrow\hat{}_\tau\ P \oplus \tau.(P)$ **by** *auto*
moreover have $(\Psi \otimes \Psi', \tau.(P), P \oplus \tau.(P)) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *tauLaw4StatImpLeft*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

and Q :: $('a, 'b, 'c)$ *psi*
and M :: $'a$
and N :: $'a$

assumes *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi', \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q), \alpha\cdot(\tau.(P) \oplus Q)) \in Rel$

shows $\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \lessapprox\!<Rel>\ \alpha\cdot(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ \alpha\cdot(\tau.(P) \oplus Q)$ **by** *auto*
hence *insertAssertion* (*extractFrame*($\alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\alpha\cdot(\tau.(P) \oplus Q)$)) Ψ
using *insertTauAssertion Identity*
by(*nominal-induct* α *rule*: *prefix.strong-inducts*, *auto*)
(*rule FrameStatImpTrans*[**where** G=$\langle\varepsilon, \Psi\rangle$], *auto simp add*: *FrameStatEq-def AssertionStatEq-def*)
moreover have $\Psi \otimes \Psi' \rhd \alpha\cdot(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ \alpha\cdot(\tau.(P) \oplus Q)$ **by** *auto*
moreover have $(\Psi \otimes \Psi', \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q), \alpha\cdot(\tau.(P) \oplus Q)) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

lemma *tauLaw4StatImpRight*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*
and Q :: $('a, 'b, 'c)$ *psi*
and α :: $'a$ *prefix*

assumes *C1*: $\bigwedge\Psi'.\ (\Psi \otimes \Psi', \alpha\cdot(\tau.(P) \oplus Q), \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)) \in Rel$

shows $\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \lessapprox\!<Rel>\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakStatImpI*)
case(*cStatImp* Ψ')
have $\Psi \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$ **by** *auto*
hence *insertAssertion* (*extractFrame*($\alpha\cdot(\tau.(P) \oplus Q)$)) $\Psi \hookrightarrow_F$ *insertAssertion* (*extractFrame*($\alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$)) Ψ
using *insertTauAssertion Identity*
by(*nominal-induct* α *rule*: *prefix.strong-inducts*, *auto*)
(*rule FrameStatImpTrans*[**where** G=$\langle\varepsilon, \Psi\rangle$], *auto simp add*: *FrameStatEq-def AssertionStatEq-def*)
moreover have $\Psi \otimes \Psi' \rhd \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \Longrightarrow\hat{}_\tau\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$ **by** *auto*
moreover have $(\Psi \otimes \Psi', \alpha\cdot(\tau.(P) \oplus Q), \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)) \in Rel$ **by**(*rule C1*)
ultimately show *?case* **by** *blast*
qed

end

end

theory *Tau-Laws-Weak*
imports *Weaken-Bisimulation Weak-Congruence Tau-Sim Tau-Stat-Imp*
begin

context *weakTauLaws* **begin**

lemma *tauLaw1*:
fixes Ψ :: $'b$
and P :: $('a, 'b, 'c)$ *psi*

shows $\Psi \rhd \tau.(P) \approx P$
proof −
let $?X = \{(\Psi, \tau.(P), P) \mid \Psi\ P.\ True\}$ **let** $?Y = \{(\Psi, P, \tau.(P)) \mid \Psi\ P.\ True\}$
have $(\Psi, \tau.(P), P) \in ?X \cup ?Y$ **by** *auto*
moreover have *eqvt*$(?X \cup ?Y)$ **by**(*auto simp add*: *eqvt-def simp add*: *eqvts*)
ultimately have $\Psi \rhd \tau.(P) \approx_w P$
proof(*coinduct rule*: *weakenTransitiveCoinduct*)
case(*cStatImp* Ψ P Q)
show *?case*
proof(*cases* $(\Psi, P, Q) \in ?X$)
case *True*
{
fix Ψ P
have $\Psi \rhd P \lessapprox_w <(?X \cup ?Y \cup weakBisim)> P$ **by**(*auto simp add*: *weaken-StatImp-def intro*: *weakBisimReflexive*)
moreover have $(\Psi, \tau.(P), P) \in ?X \cup ?Y \cup weakBisim$ **by** *auto*
ultimately have $\Psi \rhd \tau.(P) \lessapprox_w <(?X \cup ?Y \cup weakBisim)> P$
by(*rule tauLaw1StatImpLeft*)
}
with ‹$(\Psi, P, Q) \in ?X$› **show** *?thesis* **by** *auto*
next
case *False*
from ‹$(\Psi, P, Q) \notin ?X$› ‹$(\Psi, P, Q) \in ?X \cup ?Y$› **have** $(\Psi, P, Q) \in ?Y$ **by** *auto*
{
fix Ψ P
have $\Psi \rhd P \lessapprox <weakBisim> P$ **using** *weakBisimReflexive*
by(*rule weakBisimE*)
moreover have $\bigwedge \Psi\ P\ Q\ R.\ [\![\Psi \rhd P \approx Q; \Psi \rhd Q \sim R]\!] \Longrightarrow (\Psi, P, R) \in ?X \cup ?Y \cup weakBisim$
by(*fastforce intro*: *weakBisimTransitive strongBisimWeakBisim*)
ultimately have $\Psi \rhd P \lessapprox <(?X \cup ?Y \cup weakBisim)> \tau.(P)$ **by**(*rule tauLaw1StatImpRight*)
moreover have $\bigwedge \Psi\ P\ Q\ \Psi'.\ [\![(\Psi, P, Q) \in ?X \cup ?Y \cup weakBisim; \Psi \simeq \Psi']\!] \Longrightarrow (\Psi', P, Q) \in ?X \cup ?Y \cup weakBisim$
by(*auto dest*: *statEqWeakBisim*)
ultimately have $\Psi \rhd P \lessapprox_w <(?X \cup ?Y \cup weakBisim)> \tau.(P)$ **by**(*rule*

weakStatImpWeakenStatImp)
}
with ‹$(\Psi,\ P,\ Q) \in ?Y$› **show** *?thesis* **by** *auto*
qed
next
case(*cSim* Ψ P Q)
let $?Z = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in ?X \cup ?Y \cup weakenBisim \wedge \Psi \rhd Q' \sim Q\}$
have *eqvt ?Z*
apply *auto*
apply(*auto simp add*: *eqvt-def eqvts*)
apply(*rule-tac x*=($p \cdot (\tau.(Q')$)) **in** *exI*)
apply(*auto intro*: *bisimClosed*)
apply(*simp add*: *eqvts*)
apply(*blast intro*: *bisimClosed weakBisimClosed*)
apply(*rule-tac x*=($p \cdot P'$) **in** *exI*)
apply(*auto intro*: *bisimClosed*)
apply(*rule-tac x*=$\tau.(p \cdot P')$ **in** *exI*)
apply *auto*
apply(*drule-tac p*=*p* **and** *Q*=*b* **in** *bisimClosed*)
apply(*simp add*: *eqvts*)
apply(*rule-tac x*=($p \cdot P'$) **in** *exI*)
apply(*auto intro*: *bisimClosed*)
apply(*rule-tac x*=$p \cdot Q'$ **in** *exI*)
apply *auto*
by(*blast intro*: *bisimClosed dest*: *weakBisimClosed*)+
show *?case*
proof(*cases* $(\Psi,\ P,\ Q) \in ?X$)
case *True*
{
fix P
have $\Psi \rhd P \rightsquigarrow <?Z> P$ **using** *weakenBisimEqWeakBisim* **by**(*blast intro*: *weakSimReflexive weakBisimReflexive bisimReflexive*)
moreover note ‹*eqvt ?Z*›
moreover have $\bigwedge \Psi\ P\ Q\ R.\ [\![\Psi \rhd P \sim Q;\ (\Psi,\ Q,\ R) \in ?Z]\!] \Longrightarrow (\Psi,\ P,\ R) \in ?Z$
by(*blast intro*: *bisimTransitive*)
ultimately have $\Psi \rhd \tau.(P) \rightsquigarrow <?Z> P$
by(*rule tauLaw1SimLeft*)
moreover have $\bigwedge \Psi\ P\ Q\ \Psi'.\ [\![(\Psi,\ P,\ Q) \in ?Z;\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi',\ P,\ Q) \in ?Z$
by *simp* (*blast intro*: *statEqWeakBisim statEqBisim*)
ultimately have $\Psi \rhd \tau.(P) \rightsquigarrow_w <?Z> P$ **by**(*rule weakSimWeakenSim*)
}
with ‹$(\Psi,\ P,\ Q) \in ?X$› **show** *?thesis* **by** *auto*
next
case *False*
from ‹$(\Psi,\ P,\ Q) \notin ?X$› ‹$(\Psi,\ P,\ Q) \in ?X \cup ?Y$› **have** $(\Psi,\ P,\ Q) \in ?Y$ **by** *auto*

moreover {
fix P
note ‹*eqvt ?Z*›
moreover have $(\Psi,\ P,\ P) \in ?Z$ **by** *simp* (*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover have $\bigwedge\Psi\ P\ Q\ R.\ [\![(\Psi,\ P,\ Q) \in ?Z;\ \Psi \rhd Q \sim R]\!] \Longrightarrow (\Psi,\ P,\ R) \in ?Z$
by(*blast intro*: *bisimTransitive*)
ultimately have $\Psi \rhd P \rightsquigarrow <?Z> \tau.(P)$ **by**(*rule tauLaw1SimRight*)
moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ [\![(\Psi,\ P,\ Q) \in ?Z;\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi',\ P,\ Q) \in ?Z$
by *simp* (*blast intro*: *statEqWeakBisim statEqBisim*)
ultimately have $\Psi \rhd P \rightsquigarrow_w <?Z> \tau.(P)$ **by**(*rule weakSimWeakenSim*)
}
ultimately show *?thesis* **by** *auto*
qed
next
case(*cExt* $\Psi\ P\ Q\ \Psi'$)
thus *?case* **by** *auto*
next
case(*cSym* $\Psi\ P\ Q$)
thus *?case* **by** *auto*
qed
thus *?thesis* **by** *simp*
qed

lemma *tauLaw3*:
fixes Ψ :: $'b$
and α :: $'a$ *prefix*
and P :: $('a,\ 'b,\ 'c)$ *psi*

shows $\Psi \rhd \alpha\cdot(\tau.(P)) \approx \alpha\cdot P$
proof −
let $?X = (\{(\Psi,\ \alpha\cdot(\tau.(P)),\ \alpha\cdot P) \mid \Psi\ \alpha\ P.\ True\})$
let $?Y = (\{(\Psi,\ \alpha\cdot P,\ \alpha\cdot(\tau.(P))) \mid \Psi\ \alpha\ P.\ True\})$

have $(\Psi,\ \alpha\cdot(\tau.(P)),\ \alpha\cdot P) \in ?X \cup ?Y$ **by** *blast*
moreover have $eqvt(?X \cup ?Y)$ **by**(*fastforce simp add*: *eqvt-def simp add*: *eqvts*)
ultimately have $\Psi \rhd \alpha\cdot(\tau.(P)) \approx_w \alpha\cdot P$
proof(*coinduct rule*: *weakenTransitiveCoinduct*)
case(*cStatImp* $\Psi\ P\ Q$)
show *?case*
proof(*cases* $(\Psi,\ P,\ Q) \in ?X$)
case *True*
{
fix $\Psi\ \alpha\ P$
have $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ \alpha\cdot(\tau.(P)),\ \alpha\cdot P) \in ?X \cup ?Y \cup weakenBisim$ **by** *auto*
hence $\Psi \rhd \alpha\cdot(\tau.(P)) \lessapprox <(?X \cup ?Y \cup weakenBisim)> \alpha\cdot P$ **by**(*rule tauLaw3StatImpLeft*)

moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ [\![(\Psi,\ P,\ Q) \in ?X \cup ?Y \cup \mathit{weakenBisim};\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi',\ P,\ Q) \in ?X \cup ?Y \cup \mathit{weakenBisim}$
by(*fastforce intro*: *statEqWeakBisim*)
ultimately have $\Psi \rhd \alpha\cdot(\tau.(P)) \lessapprox_w <(?X \cup ?Y \cup \mathit{weakenBisim})> \alpha\cdot P$ **by**(*rule weakStatImpWeakenStatImp*)
}
with ‹$(\Psi,\ P,\ Q) \in ?X$› **show** *?thesis* **by** *blast*
next
case *False*
{
fix $\Psi\ \alpha\ P$
have $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ \alpha\cdot P,\ \alpha\cdot(\tau.(P))) \in ?X \cup ?Y \cup \mathit{weakenBisim}$ **by** *auto*
hence $\Psi \rhd \alpha\cdot P \lessapprox <(?X \cup ?Y \cup \mathit{weakenBisim})> \alpha\cdot(\tau.(P))$ **by**(*rule tauLaw3StatImpRight*)
moreover have $\bigwedge\Psi\ P\ Q\ \Psi'.\ [\![(\Psi,\ P,\ Q) \in ?X \cup ?Y \cup \mathit{weakenBisim};\ \Psi \simeq \Psi']\!] \Longrightarrow (\Psi',\ P,\ Q) \in ?X \cup ?Y \cup \mathit{weakenBisim}$
by(*fastforce intro*: *statEqWeakBisim*)
ultimately have $\Psi \rhd \alpha\cdot P \lessapprox_w <(?X \cup ?Y \cup \mathit{weakenBisim})> \alpha\cdot(\tau.(P))$ **by**(*rule weakStatImpWeakenStatImp*)
}
moreover from ‹$(\Psi,\ P,\ Q) \notin ?X$› ‹$(\Psi,\ P,\ Q) \in ?X \cup ?Y$› **have** $(\Psi,\ P,\ Q) \in ?Y$ **by** *blast*
ultimately show *?thesis* **by** *auto*
qed
next
case(*cSim* $\Psi\ P\ Q$)
let $?Z = \{(\Psi,\ P,\ Q) \mid \Psi\ P\ Q.\ \exists P'\ Q'.\ \Psi \rhd P \sim P' \wedge (\Psi,\ P',\ Q') \in ?X \cup ?Y \cup \mathit{weakenBisim} \wedge \Psi \rhd Q' \sim Q\}$
have *eqvt ?Z*
apply(*clarsimp simp add*: *eqvt-def*)
apply(*elim disjE*)
apply(*rule-tac x=p* · P' **in** *exI*)
apply(*clarsimp simp add*: *bisimClosed eqvts*)
apply(*blast intro*: *bisimClosed eqvts*)
apply(*rule-tac x=p* · P' **in** *exI*)
apply(*clarsimp simp add*: *bisimClosed eqvts*)
apply(*rule-tac x=p* · $(\alpha\cdot(\tau.(P)))$ **in** *exI*)
apply(*clarsimp simp add*: *eqvts*)
apply(*rule conjI*)
apply(*rule disjI2*)
apply(*blast intro*: *bisimClosed eqvts*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*drule-tac p=p* **in** *bisimClosed*)
apply(*simp add*: *eqvts*)
by(*blast dest*: *bisimClosed weakBisimClosed*)
show *?case*
proof(*cases* $(\Psi,\ P,\ Q) \in ?X$)
case *True*
note ‹$(\Psi,\ P,\ Q) \in ?X$›

moreover {
fix Ψ P α
note ‹*eqvt ?Z*›
moreover have $(\Psi, P, P) \in ?Z$ **using** *weakenBisimEqWeakBisim* **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover have $\bigwedge$*xvec Tvec. length xvec* = *length Tvec* $\Longrightarrow (\Psi, P[xvec::=Tvec], P[xvec::=Tvec]) \in ?Z$
using *weakenBisimEqWeakBisim* **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover have $\bigwedge \Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q; (\Psi, Q, R) \in ?Z; \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi, P, S) \in ?Z$ **by**(*blast intro*: *bisimTransitive*)
moreover have $\bigwedge \Psi\ P\ Q\ \Psi'.\ (\Psi, P, Q) \in ?Z \Longrightarrow (\Psi \otimes \Psi', P, Q) \in ?Z$ **by**(*blast dest*: *weakenBisimE*(*3*) *bisimE*(*3*))
ultimately have $\Psi \rhd \alpha\cdot(\tau.(P)) \rightsquigarrow <?Z> \alpha\cdot P$ **by**(*rule tauLaw3SimLeft*)
moreover have $\bigwedge \Psi\ P\ Q\ \Psi'.\ [\![(\Psi, P, Q) \in ?Z; \Psi \simeq \Psi']\!] \Longrightarrow (\Psi', P, Q) \in ?Z$ **by** *simp* (*blast dest*: *statEqWeakBisim statEqBisim*)
ultimately have $\Psi \rhd \alpha\cdot(\tau.(P)) \rightsquigarrow_w <?Z> \alpha\cdot P$ **by**(*rule weakSimWeakenSim*)
}
ultimately show *?thesis* **by** *auto*
next
case *False*
from ‹$(\Psi, P, Q) \notin ?X$› ‹$(\Psi, P, Q) \in ?X \cup ?Y$› **have** $(\Psi, P, Q) \in ?Y$ **by** *blast*
moreover {
fix Ψ P α
note ‹*eqvt ?Z*›
moreover have $\bigwedge \Psi$ *xvec Tvec. length xvec*=*length Tvec* $\Longrightarrow (\Psi, P[xvec::=Tvec], \tau.(P[xvec::=Tvec])) \in ?Z$
by *simp* (*blast intro*: *weakBisimE*(*4*) *bisimReflexive tauLaw1*)
moreover have $\bigwedge \Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q; (\Psi, Q, R) \in ?Z; \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi, P, S) \in ?Z$ **by**(*blast intro*: *bisimTransitive*)
moreover have $\bigwedge \Psi.\ (\Psi, P, \tau.(P)) \in ?Z$ **by** *simp* (*blast intro*: *weakBisimE*(*4*) *bisimReflexive tauLaw1*)
ultimately have $\Psi \rhd \alpha\cdot P \rightsquigarrow <?Z> \alpha\cdot(\tau.(P))$ **by**(*rule tauLaw3SimRight*)
moreover have $\bigwedge \Psi\ P\ Q\ \Psi'.\ [\![(\Psi, P, Q) \in ?Z; \Psi \simeq \Psi']\!] \Longrightarrow (\Psi', P, Q) \in ?Z$ **by** *simp* (*blast dest*: *statEqWeakBisim statEqBisim*)
ultimately have $\Psi \rhd \alpha\cdot P \rightsquigarrow_w <?Z> \alpha\cdot(\tau.(P))$ **by**(*rule weakSimWeakenSim*)
}
ultimately show *?thesis* **by** *auto*
qed
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by** *auto*
next
case(*cSym* Ψ P Q)
thus *?case* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed

lemma *tauLaw3PsiCong*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*

  shows $\Psi \rhd \alpha\cdot(\tau.(P)) \doteq \alpha\cdot P$
proof(*induct rule*: *weakPsiCongI*)
  case *cWeakBisim*
  show *?case* **by**(*rule tauLaw3*)
next
  case *cSimLeft*
  have $\Psi \rhd P \approx P$ **by**(*rule weakBisimReflexive*)
  moreover have $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ \Psi \rhd Q \approx R;\ \Psi \rhd R \sim S]\!] \Longrightarrow \Psi \rhd P \approx S$
    by(*blast intro*: *weakBisimTransitive strongBisimWeakBisim*)
  ultimately show *?case* **using** *weakBisimE*(*3*) **by**(*rule tauLaw3CongSimLeft*)
next
  case *cSimRight*
  have $\Psi \rhd P \approx P$ **by**(*rule weakBisimReflexive*)
  moreover have $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ \Psi \rhd Q \approx R;\ \Psi \rhd R \sim S]\!] \Longrightarrow \Psi \rhd P \approx S$
    by(*blast intro*: *weakBisimTransitive strongBisimWeakBisim*)
  ultimately show *?case* **using** *tauLaw1*[*THEN weakBisimE*(*4*)]
    by(*rule tauLaw3CongSimRight*)
qed

lemma *tauLaw3Cong*:
  fixes Ψ :: $'b$
  and P :: $('a, 'b, 'c)$ *psi*

  shows $\alpha\cdot(\tau.(P)) \doteq_c \alpha\cdot P$
proof(*induct rule*: *weakCongI*)
  case(*cWeakPsiCong* Ψ σ)
  show *?case*
  proof(*nominal-induct* α *rule*: *prefix.strong-inducts*)
  next
    case(*pInput M yvec N*)
    obtain p **where** *set* $p \subseteq$ *set yvec* $\times$ *set*($p \cdot$ *yvec*) **and** ($p \cdot$ *yvec*) $\sharp*$ N **and** ($p \cdot$ *yvec*) $\sharp*$ P **and** ($p \cdot$ *yvec*) $\sharp*$ σ
      by(*rule-tac xvec=yvec* **and** $c=(N, P, \sigma)$ **in** *name-list-avoiding*) *auto*
      thus *?case* **using** ‹*wellFormedSubst* σ› *tauLaw3PsiCong*[**where** α=*pInput* (*substTerm.seqSubst M* σ) ($p \cdot$ *yvec*) (*substTerm.seqSubst* ($p \cdot N$) σ)]
      by(*simp add*: *inputChainAlpha′ eqvts*)
  next
    case(*pOutput M N*)
    thus *?case* **using** ‹*wellFormedSubst* σ› *tauLaw3PsiCong*[**where** α=*pOutput* (*substTerm.seqSubst M* σ) (*substTerm.seqSubst N* σ)]
      by *simp*
  next

```
    case pTau
    thus ?case using ‹wellFormedSubst σ› tauLaw3PsiCong[where α=pTau]
      by simp
  qed
qed

end

end

theory Tau-Laws-No-Weak
  imports Tau-Sim Tau-Stat-Imp
begin

context tauSum
begin

lemma tauLaw2:
  fixes Ψ :: 'b
  and   P :: ('a, 'b, 'c) psi

  shows Ψ ▷ P ⊕ τ.(P) ≈ τ.(P)
proof −
  let ?X = {(Ψ, P ⊕ τ.(P), τ.(P)) | Ψ P. True}
  let ?Y = {(Ψ, τ.(P), P ⊕ τ.(P)) | Ψ P. True}

  have (Ψ, P ⊕ τ.(P), τ.(P)) ∈ ?X ∪ ?Y by auto
  moreover have eqvt(?X ∪ ?Y) by(auto simp add: eqvt-def eqvts)
  ultimately show ?thesis
  proof(coinduct rule: weakTransitiveCoinduct)
    case(cStatImp Ψ P Q)
    show ?case
    proof(cases (Ψ, P, Q) ∈ ?X)
      case True
      note ‹(Ψ, P, Q) ∈ ?X›
      moreover {
        fix Ψ P
        have ⋀Ψ'. (Ψ ⊗ Ψ', P ⊕ τ.(P), τ.(P)) ∈ ?X ∪ ?Y ∪ weakBisim by auto
          hence Ψ ▷ P ⊕ τ.(P) ≲<(?X ∪ ?Y ∪ weakBisim)> τ.(P) by(rule
tauLaw2StatImpLeft)
      }
      ultimately show ?thesis by blast
    next
      case False
      from ‹(Ψ, P, Q) ∉ ?X› ‹(Ψ, P, Q) ∈ ?X ∪ ?Y› have (Ψ, P, Q) ∈ ?Y by
auto
      moreover {
        fix Ψ P
        have ⋀Ψ'. (Ψ ⊗ Ψ', τ.(P), P ⊕ τ.(P)) ∈ ?X ∪ ?Y ∪ weakBisim by auto
```

hence $\Psi \rhd \tau.(P) \lessapprox <(?X \cup ?Y \cup weakBisim)> P \oplus \tau.(P)$ **by**(*rule tauLaw2StatImpRight*)
}
ultimately show *?thesis* **by** *blast*
qed
next
case(*cSim* Ψ P Q)
let $?Z = \{(\Psi, P, S) \mid \Psi\ P\ S.\ \exists Q\ R.\ \Psi \rhd P \sim Q \wedge (\Psi, Q, R) \in ?X \cup ?Y \cup weakBisim \wedge \Psi \rhd R \sim S\}$
show *?case*

proof(*cases* $(\Psi, P, Q) \in ?X$)
case *True*
note ‹$(\Psi, P, Q) \in ?X$›
moreover {
fix Ψ P
have $\bigwedge\Psi\ P.\ (\Psi, P, P) \in ?Z$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover have $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q; (\Psi, Q, R) \in ?Z; \Psi \rhd R \sim S]\!] \Longrightarrow (\Psi, P, S) \in ?Z$
by(*blast intro*: *bisimTransitive dest*: *bisimE*(*4*))
ultimately have $\Psi \rhd P \oplus \tau.(P) \rightsquigarrow <?Z> \tau.(P)$ **by**(*rule tauLaw2SimLeft*)
}
ultimately show *?thesis* **by** *blast*
next
case *False*
from ‹$(\Psi, P, Q) \notin ?X$› ‹$(\Psi, P, Q) \in ?X \cup ?Y$› **have** $(\Psi, P, Q) \in ?Y$ **by** *auto*
moreover {
fix Ψ P
have $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi, P, Q) \in ?Z$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
hence $\Psi \rhd \tau.(P) \rightsquigarrow <?Z> P \oplus \tau.(P)$ **by**(*rule tauLaw2SimRight*)
}
ultimately show *?thesis* **by** *blast*
qed
next
case(*cExt* Ψ P Q Ψ')
thus *?case* **by** *auto*
next
case(*cSym* Ψ P Q)
thus *?case* **by** *auto*
qed
qed

lemma *tauLawPsiCong2*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c)\ psi$

shows $\Psi \rhd P \oplus \tau.(P) \doteq \tau.(P)$
proof(*induct rule*: *weakPsiCongI*)
case *cWeakBisim*
thus *?case* **by**(*rule tauLaw2*)
next
case *cSimLeft*
note *weakBisimReflexive*
moreover have $\bigwedge\Psi\ P\ Q\ R\ S.\ [\![\Psi \rhd P \sim Q;\ \Psi \rhd Q \approx R;\ \Psi \rhd R \sim S]\!] \Longrightarrow \Psi \rhd P \approx S$
by(*metis weakBisimTransitive strongBisimWeakBisim*)
ultimately show *?case* **by**(*rule tauLaw2CongSimLeft*)
next
case *cSimRight*
from *strongBisimWeakBisim* **show** *?case* **by**(*rule tauLaw2CongSimRight*)
qed

lemma *tauLawCong2*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*

shows $P \oplus \tau.(P) \doteq_c \tau.(P)$
using *tauLawPsiCong2*
by(*rule-tac weakCongI*) *auto*

lemma *tauLaw4*:
fixes Ψ :: $'b$
and P :: $('a,\ 'b,\ 'c)$ *psi*
and α :: $'a$ *prefix*

shows $\Psi \rhd \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q) \approx \alpha{\cdot}(\tau.(P) \oplus Q)$
proof −
let $?X = \{(\Psi,\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q),\ \alpha{\cdot}(\tau.(P) \oplus Q)) \mid \Psi\ P\ \alpha\ Q.\ True\}$
let $?Y = \{(\Psi,\ \alpha{\cdot}(\tau.(P) \oplus Q),\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q)) \mid \Psi\ P\ \alpha\ Q.\ True\}$
have $(\Psi,\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q),\ \alpha{\cdot}(\tau.(P) \oplus Q)) \in ?X \cup ?Y$ **by** *auto*
moreover have $eqvt(?X \cup ?Y)$ **by**(*fastforce simp add*: *eqvt-def eqvts*)
ultimately show *?thesis*
proof(*coinduct rule*: *weakTransitiveCoinduct*)
case(*cStatImp* $\Psi\ P\ Q$)
show *?case*
proof(*cases* $(\Psi,\ P,\ Q) \in ?X$)
case *True*
note ‹$(\Psi,\ P,\ Q) \in ?X$›
moreover {
fix $\Psi\ \alpha\ P\ Q$
have $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q),\ \alpha{\cdot}(\tau.(P) \oplus Q)) \in ?X \cup ?Y \cup weakBisim$ **by** *blast*
hence $\Psi \rhd\ \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q) \lessapprox <(?X \cup ?Y \cup weakBisim)>\ \alpha{\cdot}(\tau.(P) \oplus Q)$ **by**(*rule tauLaw4StatImpLeft*)
}

ultimately show *?thesis* **by** *blast*
next
case *False*
from ‹$(\Psi,\ P,\ Q) \notin ?X$› ‹$(\Psi,\ P,\ Q) \in ?X \cup ?Y$› **have** $(\Psi,\ P,\ Q) \in ?Y$ **by** *blast*
moreover {
fix $\Psi\ \alpha\ P\ Q$
have $\bigwedge\Psi'.\ (\Psi \otimes \Psi',\ \alpha\cdot(\tau.(P) \oplus Q),\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)) \in ?X \cup ?Y \cup weakBisim$ **by** *auto*
hence $\Psi \rhd\ \alpha\cdot(\tau.(P) \oplus Q) \lessapprox <(?X \cup ?Y \cup weakBisim)> \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$ **by**(*rule tauLaw4StatImpRight*)
}
ultimately show *?thesis* **by** *blast*
qed
next
case(*cSim* $\Psi\ P\ Q$)
let $?Z = \{(\Psi,\ P,\ S) \mid \Psi\ P\ S.\ \exists\, Q\ R.\ \Psi \rhd P \sim Q \wedge (\Psi,\ Q,\ R) \in ?X \cup ?Y \cup weakBisim \wedge \Psi \rhd R \sim S\}$
show *?case*
proof(*cases* $(\Psi,\ P,\ Q) \in ?X$)
case *True*
note ‹$(\Psi,\ P,\ Q) \in ?X$›
moreover {
fix $\Psi\ P\ \alpha\ Q$
have $\bigwedge\Psi\ P.\ (\Psi,\ P,\ P) \in ?Z$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover have $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi,\ P,\ Q) \in ?Z$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
ultimately have $\Psi \rhd\ \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q) \rightsquigarrow <?Z>\ \alpha\cdot(\tau.(P) \oplus Q)$ **by**(*rule tauLaw4SimLeft*)
}
ultimately show *?thesis* **by** *blast*
next
case *False*
from ‹$(\Psi,\ P,\ Q) \notin ?X$› ‹$(\Psi,\ P,\ Q) \in ?X \cup ?Y$› **have** $(\Psi,\ P,\ Q) \in ?Y$ **by** *blast*
moreover {
fix $\Psi\ P\ \alpha\ Q$
have $\bigwedge\Psi\ P\ Q.\ \Psi \rhd P \sim Q \Longrightarrow (\Psi,\ P,\ Q) \in ?Z$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
moreover have $\bigwedge\Psi\ P.\ (\Psi,\ P,\ P) \in ?Z$ **by**(*blast intro*: *weakBisimReflexive bisimReflexive*)
ultimately have $\Psi \rhd \alpha\cdot(\tau.(P) \oplus Q) \rightsquigarrow <?Z> \alpha\cdot P \oplus \alpha\cdot(\tau.(P) \oplus Q)$ **by**(*rule tauLaw4SimRight*)
}
ultimately show *?thesis* **by** *blast*
qed
next
case(*cExt* $\Psi\ P\ Q\ \Psi'$)

thus *?case* **by** *auto*
next
case(*cSym* Ψ *P Q*)
thus *?case* **by** *blast*
qed
qed

lemma *tauLaw4PsiCong*:
fixes Ψ :: *'b*
and *P* :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a prefix*
and *Q* :: (*'a*, *'b*, *'c*) *psi*

shows $\Psi \rhd \alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q) \doteq \alpha{\cdot}(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakPsiCongI*)
case *cWeakBisim*
show *?case* **by**(*rule tauLaw4*)
next
case *cSimLeft*
from *weakBisimReflexive strongBisimWeakBisim* **show** *?case* **by**(*rule tauLaw4CongSimLeft*)
next
case *cSimRight*
from *strongBisimWeakBisim* **show** *?case* **by**(*rule tauLaw4CongSimRight*)
qed

lemma *tauLaw4Cong*:
fixes *P* :: (*'a*, *'b*, *'c*) *psi*
and α :: *'a prefix*
and *Q* :: (*'a*, *'b*, *'c*) *psi*

shows $\alpha{\cdot}P \oplus \alpha{\cdot}(\tau.(P) \oplus Q) \doteq_c \alpha{\cdot}(\tau.(P) \oplus Q)$
proof(*induct rule*: *weakCongI*)
case(*cWeakPsiCong* Ψ σ)
show *?case*
proof(*nominal-induct* α *rule*: *prefix.strong-inducts*)
next
case(*pInput M yvec N*)
obtain *p* **where** *set p* $\subseteq$ *set yvec* $\times$ *set*(*p* $\cdot$ *yvec*) **and** (*p* $\cdot$ *yvec*) $\sharp*$ *N* **and** (*p* $\cdot$ *yvec*) $\sharp*$ *P* **and** (*p* $\cdot$ *yvec*) $\sharp*$ *Q* **and** (*p* $\cdot$ *yvec*) $\sharp*$ σ
by(*rule-tac xvec=yvec* **and** *c*=(*N*, *P*, *Q*, σ) **in** *name-list-avoiding*) *auto*
thus *?case* **using** ‹*wellFormedSubst* σ› *tauLaw4PsiCong*[**where** α=*pInput* (*substTerm.seqSubst M* σ) (*p* $\cdot$ *yvec*) (*substTerm.seqSubst* (*p* $\cdot$ *N*) σ)]
by(*simp add*: *inputChainAlpha'* *eqvts*)
next
case(*pOutput M N*)
thus *?case* **using** ‹*wellFormedSubst* σ› *tauLaw4PsiCong*[**where** α=*pOutput* (*substTerm.seqSubst M* σ) (*substTerm.seqSubst N* σ)]
by *simp*

```
  next
    case pTau
    thus ?case using ‹wellFormedSubst σ› tauLaw4PsiCong[where α=pTau]
      by simp
  qed
qed

end

end
```

References

[1] J. Bengtson. *Formalising process calculi*, volume 94. Uppsala Dissertations from the Faculty of Science and Technology, 2010.

[2] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, pages 39–48. IEEE Computer Society, 2009.

[3] J. Bengtson and J. Parrow. Psi-calculi in isabelle. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2009.

[4] M. Johansson, J. Bengtson, J. Parrow, and B. Victor. Weak equivalences in psi-calculi. In *LICS*, pages 322–331. IEEE Computer Society, 2010.