

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
    {proof}

  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)
    {proof}

  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 ≡ pi2 ==> pi1 · Xs = pi2 · Xs
⟨proof⟩

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

shows pt TYPE('a set) TYPE('x)
⟨proof⟩

lemma pt-ball-eqvt:
fixes pi :: 'a prm
and   Xs :: 'b set
and   P :: 'b ⇒ bool

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

shows (pi · (forall x ∈ Xs. P x)) = (forall x ∈ (pi · Xs). pi · P (rev pi · x))
⟨proof⟩

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 · (Xs × Ys)) = (((p · Xs) × (p · Ys))::((b × 'c) set))
⟨proof⟩

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** $(X :: 'b) \in Xs$ **and** $x \in supp\ X$
 $\langle proof \rangle$

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

shows $supp\ (Xs \times \{\}) = (\{\} :: 'a set)$
and $supp\ (\{\} \times Xs) = (\{\} :: 'a set)$
 $\langle proof \rangle$

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 $a: Xs \neq \{\}$
and $b: Ys \neq \{\}$

shows $((supp\ (Xs \times Ys)) :: 'a set) = ((supp\ Xs) \cup (supp\ Ys))$
 $\langle proof \rangle$

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 $a: Xs \neq \{\}$
and $b: Ys \neq \{\}$

shows $(x \notin (Xs \times Ys)) = (x \notin Xs \wedge x \notin Ys)$
 $\langle proof \rangle$

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 $(Zs \#* (Xs \times Ys)) = (Zs \#* Xs \wedge Zs \#* Ys)$
and $(xvec \#* (Xs \times Ys)) = (xvec \#* Xs \wedge xvec \#* Ys)$
 $\langle proof \rangle$

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 $p \cdot q \cdot P = q \cdot p \cdot P$
 $\langle proof \rangle$

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)
```

$\langle proof \rangle$

```

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
  ⟨proof⟩

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⟩

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⟩

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 # l
  and    p mem l

  shows x # p
  ⟨proof⟩

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 #* l ==> xvec #* p
and Xs #* l ==> Xs #* p
⟨proof⟩

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

shows (A #* (B @ C)) = ((A #* B) ∧ (A #* C))
⟨proof⟩

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

shows ((Xs ∪ Ys) #* C) = ((Xs #* C) ∧ (Ys #* C))
⟨proof⟩

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

shows xvec #* (supp C - set xvec)
⟨proof⟩

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 #* p; Xs #* C] ==> Xs #* (p · C)
and [xvec #* p; xvec #* C] ==> xvec #* (p · C)
⟨proof⟩

lemma freshSets[simp]:
fixes x :: name
and y :: name
and xvec :: name list
and X :: name set

```

```

and    $C \quad :: \text{'a}$ 

shows ( $[]::\text{name list}$ )  $\sharp*$   $C$ 
and   ( $[]::\text{name list}$ )  $\sharp*$   $[y].C$ 
and   ( $\{\}::\text{name set}$ )  $\sharp*$   $C$ 
and   ( $\{\}::\text{name set}$ )  $\sharp*$   $[y].C$ 
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    $((\text{insert } x X) \sharp* C) = (x \sharp C \wedge X \sharp* C)$ 
and    $((\text{insert } x X) \sharp* ([y].C)) = (x \sharp ([y].C) \wedge X \sharp* ([y].C))$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{freshStarAtom}[simp]$ :  $(xvec::\text{name list}) \sharp* (x::\text{name}) = x \sharp xvec$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{name-list-set-fresh}[simp]$ :
fixes  $xvec :: \text{name list}$ 
and    $x \quad :: \text{'a::fs-name}$ 

shows  $(\text{set } xvec) \sharp* x = xvec \sharp* x$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{name-list-supp}$ :
fixes  $xvec :: \text{name list}$ 

shows  $\text{set } xvec = \text{supp } xvec$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{abs-fresh-list-star}$ :
fixes  $xvec :: \text{name list}$ 
and    $a \quad :: \text{name}$ 
and    $P \quad :: \text{'a::fs-name}$ 

shows  $(xvec \sharp* [a].P) = ((\text{set } xvec) - \{a\}) \sharp* P$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{abs-fresh-set-star}$ :
fixes  $X :: \text{name set}$ 
and    $a :: \text{name}$ 
and    $P :: \text{'a::fs-name}$ 

shows  $(X \sharp* [a].P) = (X - \{a\}) \sharp* P$ 
 $\langle \text{proof} \rangle$ 

lemmas  $\text{abs-fresh-star} = \text{abs-fresh-list-star} \text{ } \text{abs-fresh-set-star}$ 

lemma  $\text{abs-fresh-list-star}'[simp]$ :
fixes  $xvec :: \text{name list}$ 
and    $a \quad :: \text{name}$ 

```

```

and    $P \quad :: \text{'}a::\text{fs-name}$ 

assumes  $a \notin xvec$ 

shows  $xvec \#* [a].P = xvec \#* P$ 
 $\langle proof \rangle$ 

lemma  $\text{freshChainSym}[\text{simp}]:$ 
  fixes  $xvec :: \text{name list}$ 
  and    $yvec :: \text{name list}$ 

  shows  $xvec \#* yvec = yvec \#* xvec$ 
 $\langle proof \rangle$ 

lemmas [ $\text{eqvt}$ ] =  $\text{perm-cart-prod}[\text{OF pt-name-inst}, \text{OF pt-name-inst}, \text{OF at-name-inst}]$ 

lemma  $\text{name-set-avoiding}:$ 
  fixes  $c :: \text{'a::fs-name}$ 
  and    $X :: \text{name set}$ 

  assumes  $\text{finite } X$ 
  and    $\bigwedge_{pi::\text{name}} \text{prm}. \llbracket (pi \cdot X) \#* c; \text{distinctPerm } pi; \text{set } pi \subseteq X \times (pi \cdot X) \rrbracket$ 
 $\implies \text{thesis}$ 

  shows  $\text{thesis}$ 
 $\langle proof \rangle$ 

lemmas  $\text{simps}[\text{simp}] = \text{fresh-atm fresh-prod}$ 
   $\text{pt3}[\text{OF pt-name-inst}, \text{OF at-ds1}, \text{OF at-name-inst}]$ 
   $\text{pt-fresh-fresh}[\text{OF pt-name-inst}, \text{OF at-name-inst}]$ 
   $\text{pt-rev-pi}[\text{OF pt-name-inst}, \text{OF at-name-inst}]$ 
   $\text{pt-pi-rev}[\text{OF pt-name-inst}, \text{OF at-name-inst}]$ 

lemmas  $\text{name-supp-cart-prod} = \text{supp-cart-prod}[\text{OF pt-name-inst}, \text{OF pt-name-inst},$ 
 $\text{OF fs-name-inst}, \text{OF fs-name-inst}, \text{OF at-name-inst}]$ 
lemmas  $\text{name-fresh-cart-prod} = \text{fresh-cart-prod}[\text{OF pt-name-inst}, \text{OF pt-name-inst},$ 
 $\text{OF fs-name-inst}, \text{OF fs-name-inst}, \text{OF at-name-inst}]$ 
lemmas  $\text{name-fresh-star-cart-prod} = \text{fresh-star-cart-prod}[\text{OF pt-name-inst}, \text{OF pt-name-inst},$ 
 $\text{OF fs-name-inst}, \text{OF fs-name-inst}, \text{OF at-name-inst}]$ 

lemmas  $\text{name-swap-bij}[\text{simp}] = \text{pt-swap-bij}[\text{OF pt-name-inst}, \text{OF at-name-inst}]$ 
lemmas  $\text{name-swap} = \text{pt-swap}[\text{OF pt-name-inst}, \text{OF at-name-inst}]$ 
lemmas  $\text{name-set-fresh-fresh}[\text{simp}] = \text{pt-freshs-freshs}[\text{OF pt-name-inst}, \text{OF at-name-inst}]$ 
lemmas  $\text{list-fresh}[\text{simp}] = \text{fresh-list-nil} \text{ fresh-list-cons} \text{ fresh-list-append}$ 

definition  $\text{eqvt} :: \text{'a::fs-name set} \Rightarrow \text{bool where}$ 
 $\text{eqvt } X \equiv \forall x \in X. \forall p::\text{name} \text{ prm}. p \cdot x \in X$ 

```

```

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')
  ⟨proof⟩

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

  assumes eqvt X

  shows ([(x, y)] · X) = X
  ⟨proof⟩

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

  assumes eqvt X
  and     x ∈ X

  shows (p · x) ∈ X
  ⟨proof⟩

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

  shows xvec #* []
  and   Xs #* []
  ⟨proof⟩

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 #* (x#xs)) = ((xvec #* x) ∧ xvec #* xs)
  and   (Xs #* (x#xs)) = ((Xs #* x) ∧ (Xs #* xs))
  ⟨proof⟩

```

```

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

  shows (X #* (x, y)) = (X #* x ∧ X #* y)
  and (xvec #* (x, y)) = (xvec #* x ∧ xvec #* y)
  ⟨proof⟩

lemma name-list-avoiding:
  fixes c :: 'a::fs-name
  and xvec :: name list

  assumes ⋀ pi::name prm. [(pi · xvec) #* c; distinctPerm pi; set pi ⊆ (set xvec)
  × (set (pi · xvec))] ==> thesis

  shows thesis
  ⟨proof⟩

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

  shows distinctPerm([]::name prm)
  and (distinctPerm((a, b) # p)) = (distinctPerm p ∧ a ≠ b ∧ a # p ∧ b # p)
  ⟨proof⟩

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

  shows (p · (map f lst)) = map (p · f) (p · lst)
  ⟨proof⟩

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
  ⟨proof⟩

⟨ML⟩

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

```

```

shows ( $p \cdot (\text{distinct } xs)$ ) =  $\text{distinct } (p \cdot xs)$ 
⟨proof⟩

lemma  $\text{distinctClosed}[\text{simp}]$ :
  fixes  $p :: \text{name prm}$ 
  and  $xs :: 'a::\text{pt-name list}$ 

  shows  $\text{distinct } (p \cdot xs) = \text{distinct } xs$ 
  ⟨proof⟩

lemma  $\text{lengthEqvt}[\text{eqvt}]$ :
  fixes  $p :: \text{name prm}$ 
  and  $xs :: 'a::\text{pt-name list}$ 

  shows  $p \cdot (\text{length } xs) = \text{length } (p \cdot xs)$ 
  ⟨proof⟩

lemma  $\text{lengthClosed}[\text{simp}]$ :
  fixes  $p :: \text{name prm}$ 
  and  $xs :: 'a::\text{pt-name list}$ 

  shows  $\text{length } (p \cdot xs) = \text{length } xs$ 
  ⟨proof⟩

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

  shows ( $p \cdot (S \subseteq T)$ ) =  $((p \cdot S) \subseteq (p \cdot T))$ 
  ⟨proof⟩

lemma  $\text{subsetClosed}[\text{simp}]$ :
  fixes  $p :: \text{name prm}$ 
  and  $S :: ('a::\text{pt-name}) \text{ set}$ 
  and  $T :: ('a::\text{pt-name}) \text{ set}$ 

  shows  $((p \cdot S) \subseteq (p \cdot T)) = (S \subseteq T)$ 
  ⟨proof⟩

lemma  $\text{subsetClosed}'[\text{simp}]$ :
  fixes  $p :: \text{name prm}$ 
  and  $xvec :: \text{name list}$ 
  and  $P :: 'a::\text{fs-name}$ 

  shows ( $\text{set } (p \cdot xvec) \subseteq \text{supp } (p \cdot P)$ ) =  $(\text{set } xvec \subseteq \text{supp } P)$ 
  ⟨proof⟩

```

```

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))
  (proof)

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)

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

  shows ((p · x, p · y) mem (p · xs)) = ((x, y) mem xs)
  (proof)

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

  assumes x # p

  shows p · x = x
  (proof)

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

  assumes xvec #* p

  shows p · xvec = xvec
  and rev p · xvec = xvec
  (proof)

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

```

```

shows (xvec@yvec) #* C = ((xvec #* C) ∧ (yvec #* C))
⟨proof⟩

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

  assumes set xvec ⊆ set yvec
  and     yvec #* C

  shows xvec #* C
⟨proof⟩

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

  assumes distinctPerm p

  shows (p · (p · T)) = T
⟨proof⟩

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
⟨proof⟩

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

  shows (p · (composePerm xvec yvec)) = composePerm (p · xvec) (p · yvec)

```

$\langle proof \rangle$

abbreviation

composePermJudge ($\langle [-] \cdot_v \rightarrow [80, 80, 80] 80 \rangle$) **where** $[xvec\ yvec] \cdot_v p \equiv (composePerm\ xvec\ yvec) \cdot p$

abbreviation

composePermInvJudge ($\langle [-]^- \cdot_v \rightarrow [80, 80, 80] 80 \rangle$) **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 $((composePerm\ xvec\ yvec) @\ perm) \cdot p = [xvec\ yvec] \cdot_v (perm \cdot p)$

$\langle proof \rangle$

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 $(p \cdot ([xvec\ yvec]^- \cdot_v x)) = [(p \cdot xvec)\ (p \cdot yvec)]^- \cdot_v (p \cdot x)$

$\langle proof \rangle$

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 $(([xvec\ yvec]^- \cdot_v p) = ([xvec\ yvec]^- \cdot_v q)) = (p = q)$

$\langle proof \rangle$

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  $((xvec1 @ xvec2) (yvec1 @ yvec2))^- \cdot_v p = [xvec2\ yvec2]^- \cdot_v [xvec1\ yvec1]^- \cdot_v p$ 
 $\langle proof \rangle$ 

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

assumes length xvec = length yvec
and x \notin xvec
and x \notin yvec

shows [xvec\ yvec] \cdot_v x = x
 $\langle proof \rangle$ 

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

assumes length xvec = length yvec
and x \notin xvec
and x \notin yvec

shows [xvec\ yvec]^- \cdot_v x = x
 $\langle proof \rangle$ 

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

assumes x \notin xvec
and x \notin yvec
and length xvec = length yvec

shows x \notin [xvec\ yvec] \cdot_v p = x \notin p
and x \notin [xvec\ yvec]^- \cdot_v p = x \notin p
 $\langle proof \rangle$ 

lemma chainFreshFresh:
fixes x :: name
and y :: name
and xvec :: name list

```

```

and    $p :: 'a::pt\text{-name}$ 

assumes  $x \notin xvec$ 
and    $y \notin xvec$ 

shows  $xvec \#* [(x, y)] \cdot p = (xvec \#* p)$ 
 $\langle proof \rangle$ 

lemma permChainFreshFresh:
  fixes  $xvec :: name\ list$ 
  and    $yvec :: name\ list$ 
  and    $p :: 'a::pt\text{-name}$ 

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

  shows  $[xvec\ yvec] \cdot_v p = p$ 
  and    $[xvec\ yvec]^- \cdot_v p = p$ 
 $\langle proof \rangle$ 

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

  shows  $x \notin set\ xvec = x \notin xvec$ 
 $\langle proof \rangle$ 

lemma calcChain:
  fixes  $xvec :: name\ list$ 
  and    $yvec :: name\ list$ 

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

  shows  $[xvec\ yvec] \cdot_v xvec = yvec$ 
 $\langle proof \rangle$ 

lemma freshChainPerm:
  fixes  $xvec :: name\ list$ 
  and    $yvec :: name\ list$ 
  and    $x :: name$ 
  and    $C :: 'a::pt\text{-name}$ 

  assumes  $length\ xvec = length\ yvec$ 
  and    $yvec \#* C$ 
  and    $xvec \#* yvec$ 

```

```

and       $x \text{ mem } xvec$ 
and       $\text{distinct } yvec$ 

shows  $x \notin [xvec \ yvec] \cdot_v C$ 
 $\langle proof \rangle$ 

lemma  $\text{memFreshSimp[simp]}:$ 
fixes  $y :: \text{name}$ 
and  $yvec :: \text{name list}$ 

shows  $(\neg(y \text{ mem } yvec)) = y \notin yvec$ 
 $\langle proof \rangle$ 

lemma  $\text{freshChainPerm}':$ 
fixes  $xvec :: \text{name list}$ 
and  $yvec :: \text{name list}$ 
and  $p :: 'a::\text{pt-name}$ 

assumes  $\text{length } xvec = \text{length } yvec$ 
and  $yvec \nexists p$ 
and  $xvec \nexists yvec$ 
and  $\text{distinct } yvec$ 

shows  $xvec \nexists ([xvec \ yvec] \cdot_v p)$ 
 $\langle proof \rangle$ 

lemma  $\text{permSym}:$ 
fixes  $x :: \text{name}$ 
and  $y :: \text{name}$ 
and  $xvec :: \text{name list}$ 
and  $yvec :: \text{name list}$ 
and  $p :: 'a::\text{pt-name}$ 

assumes  $x \notin xvec$ 
and  $x \notin yvec$ 
and  $y \notin xvec$ 
and  $y \notin yvec$ 
and  $\text{length } xvec = \text{length } yvec$ 

shows  $((x, y) \cdot [xvec \ yvec] \cdot_v p) = [xvec \ yvec] \cdot_v ((x, y) \cdot p)$ 
 $\langle proof \rangle$ 

lemma  $\text{distinctPermClosed[simp]}:$ 
fixes  $p :: \text{name prm}$ 
and  $q :: \text{name prm}$ 

assumes  $\text{distinctPerm } p$ 

shows  $\text{distinctPerm}(q \cdot p)$ 

```

$\langle proof \rangle$

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 $Xs \#* x$
and $Ys \#* x$

shows $x \# (p \cdot C) = x \# C$

$\langle proof \rangle$

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 $Xs \#* xvec$
and $Ys \#* xvec$

shows $xvec \#* (p \cdot C) = xvec \#* C$

$\langle proof \rangle$

lemma *permStarFresh*:

fixes $xvec :: name list$
and $p :: name prm$
and $T :: 'a::pt-name$

assumes $xvec \#* p$

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

$\langle proof \rangle$

lemma *swapStarFresh*:

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

assumes $x \# p$

shows $x \# (p \cdot T) = x \# T$

$\langle proof \rangle$

```

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
  ⟨proof⟩

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⟩

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 #* (set yvec) = xvec #* yvec
  and Xs #* (set yvec) = Xs #* yvec
  and x # (set yvec) = x # yvec
  and set xvec #* Xs = xvec #* Xs
  ⟨proof⟩

end

theory Subst-Term
  imports Chain
begin

```

```

locale substType =
  fixes subst :: 'a::fs-name  $\Rightarrow$  name list  $\Rightarrow$  'b::fs-name list  $\Rightarrow$  'a ( $\langle -[-::=] \rangle$  [80, 80 ,80] 130)

  assumes eq[eqvt]:  $\bigwedge p::name\;prm.\;(p \cdot (M[xvec ::= Tvec])) = ((p \cdot M)[(p \cdot xvec)::=(p \cdot Tvec)])$ 

  and subst3:  $\bigwedge xvec\;Tvec\;T\;x.\;\llbracket length\;xvec = length\;Tvec; distinct\;xvec; set(xvec) \subseteq supp(T); (x::name) \notin T[xvec ::= Tvec] \rrbracket \implies x \notin Tvec$ 

  and renaming:  $\bigwedge xvec\;Tvec\;T.$   $\llbracket length\;xvec = length\;Tvec; (set\;p) \subseteq set\;xvec \times set\;(p \cdot xvec); distinctPerm\;p; (p \cdot xvec) \#*\;T \rrbracket \implies T[xvec ::= Tvec] = (p \cdot T)[(p \cdot xvec)::=Tvec]$ 

begin

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

  shows ( $supp(M[xvec ::= Tvec])::name\;set$ )  $\subseteq ((supp\;M) \cup (supp\;xvec) \cup (supp\;Tvec))$ 
   $\langle proof \rangle$ 

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

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

  shows x  $\notin M[xvec ::= Tvec]$ 
   $\langle proof \rangle$ 

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

  assumes yvec  $\#*\;M$ 
  and yvec  $\#*\;xvec$ 
  and yvec  $\#*\;Tvec$ 

  shows yvec  $\#*\;M[xvec ::= Tvec]$ 

```

$\langle proof \rangle$

lemma $fs[simp]$: $finite ((supp subst)::name\ set)$
 $\langle proof \rangle$

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 $set\ xvec \subseteq supp\ T$
 and $Xs \sharp* T[xvec::=Tvec]$

shows $Xs \sharp* Tvec$

$\langle proof \rangle$

lemma $subst4Chain$:

fixes $xvec :: name\ list$
 and $Tvec :: 'b\ list$
 and $T :: 'a$

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

shows $xvec \sharp* T[xvec::=Tvec]$

$\langle proof \rangle$

definition $seqSubst :: 'a \Rightarrow (name\ list \times 'b\ list)\ list \Rightarrow 'a\ ([\cdot, \cdot])\ [80, 80]\ 130$
where $M[\langle \sigma \rangle] \equiv foldl (\lambda N. \lambda(xvec, Tvec). N[xvec::=Tvec]) M \sigma$

lemma $seqSubstNil[simp]$:

$seqSubst\ M\ [] = M$

$\langle proof \rangle$

lemma $seqSubstCons[simp]$:

shows $seqSubst\ M\ ((xvec, Tvec)\#\sigma) = seqSubst(M[xvec::=Tvec])\ \sigma$
 $\langle proof \rangle$

lemma $seqSubstTermAppend[simp]$:

shows $seqSubst\ M\ (\sigma @ \sigma') = seqSubst\ (seqSubst\ M\ \sigma)\ \sigma'$
 $\langle proof \rangle$

definition $wellFormedSubst :: (('d::fs-name)\ list \times ('e::fs-name)\ list)\ list \Rightarrow bool$
where $wellFormedSubst\ \sigma = ((filter\ (\lambda(xvec, Tvec). \neg(length\ xvec = length\ Tvec \wedge distinct\ xvec))\ \sigma) = [])$

```

lemma wellFormedSubstEqvt[eqvt]:
  fixes  $\sigma :: (('d::fs-name) list \times ('e::fs-name) list) list$ 
  and  $p :: name\; prm$ 

  shows  $p \cdot (wellFormedSubst \sigma) = wellFormedSubst(p \cdot \sigma)$ 
   $\langle proof \rangle$ 

lemma wellFormedSimp[simp]:
  fixes  $\sigma :: (('d::fs-name) list \times ('e::fs-name) list) list$ 
  and  $p :: name\; prm$ 

  shows  $wellFormedSubst(p \cdot \sigma) = wellFormedSubst \sigma$ 
   $\langle proof \rangle$ 

lemma wellFormedNil[simp]:
   $wellFormedSubst []$ 
   $\langle proof \rangle$ 

lemma wellFormedCons[simp]:
  shows  $wellFormedSubst((xvec, Tvec)\#\sigma) = (length\; xvec = length\; Tvec \wedge distinct\; xvec \wedge wellFormedSubst \sigma)$ 
   $\langle proof \rangle$ 

lemma wellFormedAppend[simp]:
  fixes  $\sigma :: (('d::fs-name) list \times ('e::fs-name) list) list$ 
  and  $\sigma' :: (('d::fs-name) list \times ('e::fs-name) list) list$ 

  shows  $wellFormedSubst(\sigma @ \sigma') = (wellFormedSubst \sigma \wedge wellFormedSubst \sigma')$ 
   $\langle proof \rangle$ 

lemma seqSubst2[intro]:
  fixes  $\sigma :: (name\; list \times 'b\; list) list$ 
  and  $T :: 'a$ 
  and  $x :: name$ 

  assumes  $x \# \sigma$ 
  and  $x \# T$ 

  shows  $x \# T[\langle \sigma \rangle]$ 
   $\langle proof \rangle$ 

lemma seqSubst2Chain[intro]:
  fixes  $\sigma :: (name\; list \times 'b\; list) list$ 
  and  $T :: 'a$ 
  and  $xvec :: name\; list$ 

  assumes  $xvec \#* \sigma$ 
  and  $xvec \#* T$ 

```

```

shows xvec #* T[< $\sigma$ >]
⟨proof⟩

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
  ⟨(⊥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    $\Psi :: 'b$ 
and    $\Phi :: 'c$ 

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

and    $X \#* \perp_c$ 
and    $X \#* \Box \Phi \Rightarrow P C = (X \#* \Phi \wedge X \#* P \wedge X \#* C)$ 

```

$\langle proof \rangle$

lemma $\psiFreshVec[simp]$:

fixes $xvec :: name list$

```

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

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

and    $xvec \#* \perp_c$ 
and    $xvec \#* \Box \Phi \Rightarrow P C = (xvec \#* \Phi \wedge xvec \#* P \wedge xvec \#* C)$ 

```

$\langle proof \rangle$

fun $\psiCases :: ('c::fs-name \times ('a::fs-name, 'b::fs-name, 'c) \psi) list \Rightarrow ('a, 'b, 'c)$

psiCase

where

```

base:  $\psiCases [] = \perp_c$ 
| step:  $\psiCases ((\Phi, P)\#xs) = Cond \Phi P (\psiCases xs)$ 

```

lemma $\psiCasesEqvt[eqvt]$:

fixes $p :: name prm$

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

```

shows ( $p \cdot (\psi\text{Cases } Cs) = \psi\text{Cases}(p \cdot Cs)$ 
 $\langle proof \rangle$ 

lemma  $\psi\text{CasesFresh}[simp]$ :
  fixes  $x :: name$ 
  and  $Cs :: ('c::fs-name \times ('a::fs-name, 'b::fs-name, 'c) psi) list$ 

  shows  $x \# \psi\text{Cases } Cs = x \# Cs$ 
 $\langle proof \rangle$ 

lemma  $\psi\text{CasesFreshChain}[simp]$ :
  fixes  $xvec :: name list$ 
  and  $Cs :: ('c::fs-name \times ('a::fs-name, 'b::fs-name, 'c) psi) list$ 
  and  $Xs :: name set$ 

  shows  $(xvec \#* \psi\text{Cases } Cs) = xvec \#* Cs$ 
  and  $(Xs \#* \psi\text{Cases } Cs) = Xs \#* Cs$ 
 $\langle proof \rangle$ 

abbreviation
 $\psi\text{CasesJudge} (\langle Cases \rightarrow [80] 80 \rangle \text{ where } Cases Cs \equiv \text{Case}(\psi\text{Cases } Cs))$ 

primrec  $\text{resChain} :: name list \Rightarrow ('a::fs-name, 'b::fs-name, 'c::fs-name) psi \Rightarrow ('a, 'b, 'c) psi \text{ where}$ 
   $\text{base: } \text{resChain} [] P = P$ 
   $\mid \text{step: } \text{resChain} (x \# xs) P = (\nu x)(\text{resChain } xs P)$ 

notation  $\text{resChain} (\langle (\nu *-) \rightarrow [80, 80] 80 \rangle)$ 

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

  shows  $perm \cdot ((\nu * xvec) P) = (\nu * (perm \cdot xvec))(perm \cdot P)$ 
 $\langle proof \rangle$ 

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

  shows  $\text{supp}((\nu * xvec) P) = (\text{supp } P) - \text{set } xvec$ 
 $\langle proof \rangle$ 

lemma  $\text{resChainFresh}$ :
  fixes  $x :: name$ 
  and  $xvec :: name list$ 
  and  $P :: ('a::fs-name, 'b::fs-name, 'c::fs-name) psi$ 
```

```

shows  $x \notin (\nu*xvec)P = (x \in set\ xvec \vee x \notin P)$ 
⟨proof⟩

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  $Xs \#* ((\nu*xvec)P) = (\forall x \in Xs. x \in set\ xvec \vee x \notin P)$ 
  and  $yvec \#* ((\nu*xvec)P) = (\forall x \in (set\ yvec). x \in set\ xvec \vee x \notin P)$ 
⟨proof⟩

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  $Xs \#* xvec \implies Xs \#* ((\nu*xvec)P) = (Xs \#* P)$ 
  and  $yvec \#* xvec \implies yvec \#* ((\nu*xvec)P) = (yvec \#* P)$ 
  and  $xvec \#* ((\nu*xvec)P)$ 
⟨proof⟩

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

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

  shows  $(\nu*xvec)P = (\nu*(p \cdot xvec))((p \cdot P))$ 
⟨proof⟩

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

  shows  $(\nu*(xvec @ yvec))P = (\nu*xvec)((\nu*yvec)P)$ 
⟨proof⟩

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  $((\nu*xvec)(P \parallel Q)) = P' \parallel Q' \implies (P = P' \wedge Q = Q')$ 
and    $(P \parallel Q = (\nu*xvec)(P' \parallel Q')) \implies (P = P' \wedge Q = Q')$ 
 $\langle proof \rangle$ 

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

abbreviation
   $inputChainJudge (\cdot(\lambda*- -)\cdot \rightarrow [80, 80, 80, 80] 80) \text{ where } M(\lambda*xvec N).P \equiv$ 
 $M((inputChain xvec N P))$ 

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  $p \cdot (inputChain xvec N P) = inputChain (p \cdot xvec) (p \cdot N) (p \cdot P)$ 
 $\langle proof \rangle$ 

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  $x \notin (inputChain xvec N P) = (x \in set xvec \vee (x \notin N \wedge x \notin P))$ 
 $\langle proof \rangle$ 

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

  shows  $xvec \#* (inputChain xvec N P)$ 
 $\langle proof \rangle$ 

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  $Xs \#* (inputChain xvec N P) = (\forall x \in Xs. x \in set xvec \vee (x \notin N \wedge x \notin P))$ 
 $\langle proof \rangle$ 

```

```

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

  assumes XsFreshP: Xs #* (inputChain xvec N P)
  and   YsFreshN: Ys #* N
  and   YsFreshP: Ys #* P
  and   S: set p ⊆ Xs × Ys

  shows (inputChain xvec N P) = (inputChain (p · xvec) (p · N) (p · P))
  ⟨proof⟩

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⟩

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⟩

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⟩

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⟩

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⟩

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⟩

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)
⟨proof⟩

lemma inputChainInject[simp]:
shows (inputChain xvec M P = inputChain xvec N Q) = ((M = N) ∧ (P = Q))
⟨proof⟩

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⟩

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⟩

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)
⟨proof⟩

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⟨N⟩.P) = True
  | guarded (M(I)) = True
  | guarded (Case C) = guarded'' C
  | guarded (P ∥ Q) = ((guarded P) ∧ (guarded Q))
  | guarded ((νx)P) = (guarded P)
  | guarded ({Ψ}) = False
  | guarded (!P) = guarded P

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

  | guarded'' (⊥c) = True
  | guarded'' (□φ ⇒ P C) = (guarded P ∧ guarded'' C)
  ⟨proof⟩

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 · (guarded P)) = guarded (p · P)
  and (p · (guarded' I)) = guarded' (p · I)
  and (p · (guarded'' C)) = guarded'' (p · C)
  ⟨proof⟩

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 · P)
  ⟨proof⟩

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  $\Rightarrow$  name list  $\Rightarrow$  'a list  $\Rightarrow$  ('a, 'b, 'c) psi
  and subs' :: ('a::fs-name, 'b::fs-name, 'c::fs-name) input  $\Rightarrow$  name list  $\Rightarrow$  'a list  $\Rightarrow$  ('a, 'b, 'c) input
  and subs'' :: ('a::fs-name, 'b::fs-name, 'c::fs-name) psiCase  $\Rightarrow$  name list  $\Rightarrow$  'a list
     $\Rightarrow$  ('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)
  | [ $y \notin xvec; y \notin Tvec \implies (\text{subs } (\{\nu y\} P) xvec Tvec) = (\{\nu y\} (\text{subs } P xvec Tvec))$ ]
  | (subs (\{\Psi\}) xvec Tvec) =  $\{\text{substAssert } \Psi xvec Tvec\}$ 
  | (subs (!P) xvec Tvec) =  $!(\text{subs } P xvec Tvec)$ 
  | (subs' ((Trm M P)::('a::fs-name, 'b::fs-name, 'c::fs-name) input) xvec Tvec) =
     $(\{\text{substTerm } M xvec Tvec\}.(\text{subs } P xvec Tvec))$ 
  | [ $y \notin xvec; y \notin Tvec \implies (\text{subs}' (\nu y I) xvec Tvec) = (\nu y (\text{subs}' I xvec Tvec))$ ]
  | (subs'' ( $\perp_c$ ::('a::fs-name, 'b::fs-name, 'c::fs-name) psiCase) xvec Tvec) =  $\perp_c$ 
  | (subs'' ( $\Box \Phi \Rightarrow P C$ ) xvec Tvec) =  $(\Box (\text{substCond } \Phi xvec Tvec) \Rightarrow (\text{subs } P xvec Tvec))$ 
    ⟨proof⟩

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 · (subs P xvec Tvec)) = subs (p · P) (p · xvec) (p · Tvec)
  and (p · (subs' I xvec Tvec)) = subs' (p · I) (p · xvec) (p · Tvec)
  and (p · (subs'' C xvec Tvec)) = subs'' (p · C) (p · xvec) (p · Tvec)
  ⟨proof⟩

lemma subst2[intro]:
  fixes xvec :: name list
  and Tvec :: 'a list
  and x :: name
  and P :: ('a, 'b, 'c) psi

```

```

and    $I :: ('a, 'b, 'c) \text{ input}$ 
and    $C :: ('a, 'b, 'c) \text{ psiCase}$ 

assumes  $x \notin Tvec$ 
and      $x \notin xvec$ 

shows  $x \notin P \implies x \notin (\text{subs } P \text{ } xvec \text{ } Tvec)$ 
and    $x \notin I \implies x \notin (\text{subs}' \text{ } I \text{ } xvec \text{ } Tvec)$ 
and    $x \notin C \implies x \notin (\text{subs}'' \text{ } C \text{ } xvec \text{ } Tvec)$ 
⟨proof⟩

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

assumes  $Xs \nsubseteq xvec$ 
and      $Xs \nsubseteq Tvec$ 

shows  $Xs \nsubseteq P \implies Xs \nsubseteq (\text{subs } P \text{ } xvec \text{ } Tvec)$ 
and    $Xs \nsubseteq I \implies Xs \nsubseteq (\text{subs}' \text{ } I \text{ } xvec \text{ } Tvec)$ 
and    $Xs \nsubseteq C \implies Xs \nsubseteq (\text{subs}'' \text{ } C \text{ } xvec \text{ } Tvec)$ 
⟨proof⟩

lemma renaming:
fixes  $xvec :: \text{name list}$ 
and    $Tvec :: 'a \text{ list}$ 
and    $p :: \text{name prm}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $I :: ('a, 'b, 'c) \text{ input}$ 
and    $C :: ('a, 'b, 'c) \text{ psiCase}$ 

assumes  $\text{length } xvec = \text{length } Tvec$ 
and      $\text{set } p \subseteq \text{set } xvec \times \text{set } (p \cdot xvec)$ 
and      $\text{distinctPerm } p$ 

shows  $\llbracket (p \cdot xvec) \nsubseteq P \rrbracket \implies (\text{subs } P \text{ } xvec \text{ } Tvec) = \text{subs } (p \cdot P) \text{ } (p \cdot xvec) \text{ } Tvec$ 
and    $\llbracket (p \cdot xvec) \nsubseteq I \rrbracket \implies (\text{subs}' \text{ } I \text{ } xvec \text{ } Tvec) = \text{subs}' (p \cdot I) \text{ } (p \cdot xvec) \text{ } Tvec$ 
and    $\llbracket (p \cdot xvec) \nsubseteq C \rrbracket \implies (\text{subs}'' \text{ } C \text{ } xvec \text{ } Tvec) = \text{subs}'' (p \cdot C) \text{ } (p \cdot xvec) \text{ } Tvec$ 
⟨proof⟩

lemma subst4Chain:
fixes  $xvec :: \text{name list}$ 
and    $Tvec :: 'a \text{ list}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $I :: ('a, 'b, 'c) \text{ input}$ 

```

```

and    $C :: ('a, 'b, 'c) \psiCase$ 

assumes  $\text{length } xvec = \text{length } Tvec$ 
and     $\text{distinct } xvec$ 
and     $xvec \#* Tvec$ 

shows  $xvec \#* (\text{subs } P xvec Tvec)$ 
and    $xvec \#* (\text{subs}' I xvec Tvec)$ 
and    $xvec \#* (\text{subs}'' C xvec Tvec)$ 
⟨proof⟩

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

assumes  $\text{length } xvec = \text{length } Tvec$ 
and     $\text{distinct } xvec$ 

shows  $\text{guarded } P \implies \text{guarded}(\text{subs } P xvec Tvec)$ 
and    $\text{guarded}' I \implies \text{guarded}'(\text{subs}' I xvec Tvec)$ 
and    $\text{guarded}'' C \implies \text{guarded}''(\text{subs}'' C xvec Tvec)$ 
⟨proof⟩

definition  $\text{seqSubs} :: ('a, 'b, 'c) \psi \Rightarrow (\text{name list} \times 'a \text{list}) \text{list} \Rightarrow ('a, 'b, 'c) \psi$ 
 $(\langle \cdot \rangle, [80, 80], 130)$ 
where  $P[\langle \sigma \rangle] \equiv \text{foldl } (\lambda Q. \lambda(xvec, Tvec). \text{subs } Q xvec Tvec) P \sigma$ 

definition  $\text{seqSubs}' :: ('a, 'b, 'c) \text{input} \Rightarrow (\text{name list} \times 'a \text{list}) \text{list} \Rightarrow ('a, 'b, 'c)$ 
 $\text{input}$ 
where  $\text{seqSubs}' I \sigma \equiv \text{foldl } (\lambda Q. \lambda(xvec, Tvec). \text{subs}' Q xvec Tvec) I \sigma$ 

definition  $\text{seqSubs}'' :: ('a, 'b, 'c) \psiCase \Rightarrow (\text{name list} \times 'a \text{list}) \text{list} \Rightarrow ('a, 'b,$ 
 $'c) \psiCase$ 
where  $\text{seqSubs}'' C \sigma \equiv \text{foldl } (\lambda Q. \lambda(xvec, Tvec). \text{subs}'' Q xvec Tvec) C \sigma$ 

lemma  $\text{substInputChain[simp]}:$ 
fixes  $xvec :: \text{name list}$ 
and    $N :: 'a$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $yvec :: \text{name list}$ 
and    $Tvec :: 'a \text{list}$ 

assumes  $xvec \#* yvec$ 
and    $xvec \#* Tvec$ 

shows  $\text{subs}' (\text{inputChain } xvec N P) yvec Tvec = \text{inputChain } xvec (\text{substTerm } N$ 

```

```

yvec Tvec) (subs P yvec Tvec)
⟨proof⟩

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)
⟨proof⟩

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

  shows (subs'' (psiCases Cs) xvec Tvec) = psiCases(caseListSubst Cs xvec Tvec)
⟨proof⟩

lemma seqSubstSimps[simp]:
  shows seqSubs (0) σ = 0
  and (seqSubs (M⟨N⟩.P) σ) = (substTerm.seqSubst M σ)⟨(substTerm.seqSubst
N σ)⟩.(seqSubs P σ)
  and (seqSubs (M(I) σ) = (substTerm.seqSubst M σ)⟨(seqSubs' I σ)

  and (seqSubs (Case C) σ) = (Case (seqSubs'' C σ))
  and (seqSubs (P || Q) σ) = (seqSubs P σ) || (seqSubs Q σ)
  and [y # σ] ⇒ (seqSubs ((νy)P) σ) = ((νy)(seqSubs P σ))
  and (seqSubs ({Ψ}) σ) = {((substAssert.seqSubst Ψ σ))}
  and (seqSubs (!P) σ) = !(seqSubs P σ)

  and (seqSubs' ((Trm M P)::('a::fs-name, 'b::fs-name, 'c::fs-name) input) σ) =
()⟨(substTerm.seqSubst M σ).(seqSubs P σ)⟩
  and [y # σ] ⇒ (seqSubs' (ν y I) σ) = (ν y (seqSubs' I σ))

  and (seqSubs'' (⊥c::('a::fs-name, 'b::fs-name, 'c::fs-name) psiCase) σ) = ⊥c
  and (seqSubs'' (□Φ ⇒ P C) σ) = (□(substCond.seqSubst Φ σ) ⇒ (seqSubs P
σ)) (seqSubs'' C σ))
⟨proof⟩

lemma seqSubsNil[simp]:
  seqSubs P [] = P

```

$\langle proof \rangle$

lemma *seqSubsCons*[simp]:

shows *seqSubs P ((xvec, Tvec) # σ) = seqSubs(subs P xvec Tvec) σ*
 $\langle proof \rangle$

lemma *seqSubsTermAppend*[simp]:

shows *seqSubs P (σ @ σ') = seqSubs (seqSubs P σ) σ'*
 $\langle proof \rangle$

fun *caseListSeqSubst* :: $('c \times ('a, 'b, 'c) \psi) \text{ list} \Rightarrow (\text{name list} \times 'a \text{ list}) \text{ list} \Rightarrow ('c \times ('a, 'b, 'c) \psi) \text{ list}$

where

caseListSeqSubst [] - = []
| *caseListSeqSubst ((φ, P) # Cs) σ = (substCond.seqSubst φ σ, (seqSubs P σ)) # (caseListSeqSubst Cs σ)*

lemma *seqSubstCases*[simp]:

fixes *Cs :: ('c × ('a, 'b, 'c) ψ) list*
 and $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$

shows *seqSubs (Cases Cs) σ = Cases(caseListSeqSubst Cs σ)*
 $\langle proof \rangle$

lemma *seqSubstCases'*[simp]:

fixes *Cs :: ('c × ('a, 'b, 'c) ψ) list*
 and $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$

shows *(seqSubs'' (ψCases Cs) σ) = ψCases(caseListSeqSubst Cs σ)*
 $\langle proof \rangle$

lemma *seqSubstEqvt*[eqvt]:

fixes *P :: ('a, 'b, 'c) ψ*
 and $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$
 and $p :: \text{name prm}$

shows *(p · (P[<σ>])) = (p · P)[<(p · σ)>]*
 $\langle proof \rangle$

lemma *guardedSeqSubst*:

assumes *guarded P*
 and *wellFormedSubst σ*

shows *guarded(seqSubs P σ)*
 $\langle proof \rangle$

end

lemma *inter-eqvt*:

```

shows ( $pi :: name\ perm$ )  $\cdot ((X :: name\ set) \cap Y) = (pi \cdot X) \cap (pi \cdot Y)$ 
 $\langle proof \rangle$ 

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

  shows  $p \cdot (X - Y) = (p \cdot X) - (p \cdot Y)$ 
 $\langle proof \rangle$ 

lemma perm-singleton[simp]:
  shows ( $p :: name\ perm$ )  $\cdot \{(x :: name)\} = \{p \cdot x\}$ 
 $\langle proof \rangle$ 

end

theory Frame
  imports Agent
begin

lemma permLength[simp]:
  fixes  $p :: name\ perm$ 
  and  $xvec :: 'a :: pt-name\ list$ 

  shows  $length(p \cdot xvec) = length\ xvec$ 
 $\langle proof \rangle$ 

nominal-datatype 'assertion frame =
  FAassert 'assertion::fs-name
  | FRes «name» ('assertion frame) ( $\langle \langle \nu \cdot \rangle \rangle [80, 80] 80$ )
 $\langle proof \rangle$ 

primrec frameResChain :: name list  $\Rightarrow$  ('a::fs-name) frame  $\Rightarrow$  'a frame where
  base: frameResChain []  $F = F$ 
  | step: frameResChain (x#xs)  $F = (\nu x)(frameResChain xs F)$ 

  notation frameResChain ( $\langle \langle \nu^* \cdot \rangle \rangle [80, 80] 80$ )
  notation FAassert ( $\langle \langle \varepsilon, \cdot \rangle \rangle [80] 80$ )
  abbreviation FAassertJudge ( $\langle \langle \cdot, \cdot \rangle \rangle [80, 80] 80$ ) where  $\langle A_F, \Psi_F \rangle \equiv frameResChain A_F (FAassert \Psi_F)$ 

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

  shows perm  $\cdot (\langle \langle \nu^* xvec \rangle \rangle F) = (\nu^*(perm \cdot xvec)) \cdot (perm \cdot F)$ 
 $\langle proof \rangle$ 

```

```

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

  shows x # (ν*xvec)F = (x ∈ set xvec ∨ x # F)
  ⟨proof⟩

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

  shows Xs #* ((ν*xvec)F) = (∀x∈Xs. x ∈ set xvec ∨ x # F)
  ⟨proof⟩

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

  assumes xvecFreshF: (p • xvec) #* F
  and S: set p ⊆ set xvec × set (p • xvec)

  shows (ν*xvec)F = (ν*(p • xvec))(p • F)
  ⟨proof⟩

lemma frameChainAlpha':
  fixes p :: name prm
  and AP :: name list
  and ΨP :: 'a::fs-name

  assumes (p • AP) #* ΨP
  and S: set p ⊆ set AP × set (p • AP)

  shows ⟨AP, ΨP⟩ = ⟨(p • AP), p • ΨP⟩
  ⟨proof⟩

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

  assumes y # F

  shows (νx)F = (νy)([(x, y)] • F)
  ⟨proof⟩

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
⟨proof⟩

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⟩

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
⟨proof⟩

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
⟨proof⟩

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       $\text{supp } \Psi = (\{\} :: \text{name set})$ 

shows  $\Psi = \Psi'$ 
⟨proof⟩

lemma frameChainEq:
fixes  $xvec :: \text{name list}$ 
and  $\Psi :: 'a::\text{fs-name}$ 
and  $yvec :: \text{name list}$ 
and  $\Psi' :: 'a::\text{fs-name}$ 

assumes  $\langle xvec, \Psi \rangle = \langle yvec, \Psi' \rangle$ 
and  $xvec \#* yvec$ 

obtains  $p$  where  $(\text{set } p) \subseteq (\text{set } xvec) \times \text{set } (yvec)$  and  $\text{distinctPerm } p$  and  $\Psi' = p \cdot \Psi$ 
⟨proof⟩

lemma frameChainEq':
fixes  $xvec :: \text{name list}$ 
and  $\Psi :: 'a::\text{fs-name}$ 
and  $yvec :: \text{name list}$ 
and  $\Psi' :: 'a::\text{fs-name}$ 

assumes  $\langle xvec, \Psi \rangle = \langle yvec, \Psi' \rangle$ 
and  $xvec \#* yvec$ 
and  $\text{distinct } xvec$ 
and  $\text{distinct } yvec$ 

obtains  $p$  where  $(\text{set } p) \subseteq (\text{set } xvec) \times \text{set } (p \cdot xvec)$  and  $\text{distinctPerm } p$  and
 $yvec = p \cdot xvec$  and  $\Psi' = p \cdot \Psi$ 
⟨proof⟩

lemma frameEq[simp]:
fixes  $A_F :: \text{name list}$ 
and  $\Psi :: 'a::\text{fs-name}$ 
and  $\Psi' :: 'a$ 

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⟩

lemma distinctFrame:
fixes  $A_F :: \text{name list}$ 
and  $\Psi_F :: 'a::\text{fs-name}$ 
and  $C :: 'b::\text{fs-name}$ 

assumes  $A_F \#* C$ 

```

```

obtains  $A_F'$  where  $\langle A_F, \Psi_F \rangle = \langle A_F', \Psi_F \rangle$  and distinct  $A_F'$  and  $A_F' \#* C$ 
⟨proof⟩

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

  obtains  $A_F \Psi_F$  where  $F = \langle A_F, \Psi_F \rangle$  and distinct  $A_F$  and  $A_F \#* C$ 
⟨proof⟩

locale assertionAux =
  fixes SCompose :: ' $b::fs-name \Rightarrow 'b \Rightarrow 'b$ ' (infixr  $\langle\otimes\rangle$  80)
  and SImp :: ' $b \Rightarrow 'c::fs-name \Rightarrow bool$  ( $\langle\cdot \vdash \cdot\rangle [70, 70]$ ) 70)
  and SBottom :: ' $b$ ' ( $\langle\perp\rangle 90$ )
  and SChanEq :: ' $a::fs-name \Rightarrow 'a \Rightarrow 'c$ ' ( $\langle\cdot \leftrightarrow \cdot\rangle [80, 80]$ ) 80)

assumes statEqvt[eqvt]:  $\bigwedge p::name prm. p \cdot (\Psi \vdash \Phi) = (p \cdot \Psi) \vdash (p \cdot \Phi)$ 
and statEqvt'[eqvt]:  $\bigwedge p::name prm. p \cdot (\Psi \otimes \Psi') = (p \cdot \Psi) \otimes (p \cdot \Psi')$ 
and statEqvt''[eqvt]:  $\bigwedge p::name prm. p \cdot (M \leftrightarrow N) = (p \cdot M) \leftrightarrow (p \cdot N)$ 
and permBottom[eqvt]:  $\bigwedge p::name prm. (p \cdot SBottom) = SBottom$ 

begin

lemma statClosed:
  fixes  $\Psi :: 'b$ 
  and  $\varphi :: 'c$ 
  and  $p :: name prm$ 

  assumes  $\Psi \vdash \varphi$ 

  shows  $(p \cdot \Psi) \vdash (p \cdot \varphi)$ 
⟨proof⟩

lemma compSupp:
  fixes  $\Psi :: 'b$ 
  and  $\Psi' :: 'b$ 

  shows  $(supp(\Psi \otimes \Psi')::name set) \subseteq ((supp \Psi) \cup (supp \Psi'))$ 
⟨proof⟩

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

  shows  $(supp(M \leftrightarrow N)::name set) \subseteq ((supp M) \cup (supp N))$ 
⟨proof⟩

lemma freshComp[intro]:
  fixes  $x :: name$ 

```

```

and    $\Psi :: 'b$ 
and    $\Psi' :: 'b$ 

assumes  $x \notin \Psi$ 
and      $x \notin \Psi'$ 

shows  $x \notin \Psi \otimes \Psi'$ 
⟨proof⟩

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

  shows  $\llbracket xvec \notin \Psi; xvec \notin \Psi' \rrbracket \implies xvec \notin (\Psi \otimes \Psi')$ 
  and    $\llbracket Xs \notin \Psi; Xs \notin \Psi' \rrbracket \implies Xs \notin (\Psi \otimes \Psi')$ 
⟨proof⟩

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

  assumes  $x \notin M$ 
  and      $x \notin N$ 

  shows  $x \notin M \leftrightarrow N$ 
⟨proof⟩

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

  shows  $\llbracket xvec \notin M; xvec \notin N \rrbracket \implies xvec \notin (M \leftrightarrow N)$ 
  and    $\llbracket Xs \notin M; Xs \notin N \rrbracket \implies Xs \notin (M \leftrightarrow N)$ 
⟨proof⟩

lemma suppBottom[simp]:
  shows  $((supp SBottom)::name set) = \{\}$ 
⟨proof⟩

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

  shows  $x \notin \perp$ 
⟨proof⟩

```

```

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

  shows xvec #* ( $\perp$ )
  and   Xs   #* ( $\perp$ )
  {proof}

lemma chanEqClosed:
  fixes  $\Psi$  :: '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}

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

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

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

  shows  $\Psi' \vdash \Phi$ 
  {proof}

lemma statEqEnt:
  fixes  $\Psi$  :: 'b
  and    $\Psi'$  :: 'b
  and    $\Phi$  :: 'c

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

  shows  $\Psi' \vdash \Phi$ 
  {proof}

```

```

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>

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

  assumes  $\Psi \simeq \Psi'$ 

  shows  $(p \cdot \Psi) \simeq (p \cdot \Psi')$ 
  <proof>

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

  shows  $(p \cdot (\Psi \hookrightarrow \Psi')) = ((p \cdot \Psi) \hookrightarrow (p \cdot \Psi'))$ 
  <proof>

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

  shows  $(p \cdot (\Psi \simeq \Psi')) = ((p \cdot \Psi) \simeq (p \cdot \Psi'))$ 
  <proof>

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

  shows  $\Psi \hookrightarrow \Psi$ 
  <proof>

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

  shows  $\Psi \simeq \Psi$ 
  <proof>

```

```

lemma AssertionStatEqSym:
  fixes  $\Psi$  :: ' $b$ 
  and  $\Psi'$  :: ' $b$ 
  assumes  $\Psi \simeq \Psi'$ 
  shows  $\Psi' \simeq \Psi$ 
   $\langle proof \rangle$ 

lemma AssertionStatImpTrans:
  fixes  $\Psi$  :: ' $b$ 
  and  $\Psi'$  :: ' $b$ 
  and  $\Psi''$  :: ' $b$ 
  assumes  $\Psi \hookrightarrow \Psi'$ 
  and  $\Psi' \hookrightarrow \Psi''$ 
  shows  $\Psi \hookrightarrow \Psi''$ 
   $\langle proof \rangle$ 

lemma AssertionStatEqTrans:
  fixes  $\Psi$  :: ' $b$ 
  and  $\Psi'$  :: ' $b$ 
  and  $\Psi''$  :: ' $b$ 
  assumes  $\Psi \simeq \Psi'$ 
  and  $\Psi' \simeq \Psi''$ 
  shows  $\Psi \simeq \Psi''$ 
   $\langle proof \rangle$ 

definition
   $FrameImp :: 'b :: fs\text{-}name$  frame  $\Rightarrow 'c \Rightarrow bool$  (infixl  $\vdash_F$  70)
  where  $(F \vdash_F \Phi) = (\exists A_F \Psi_F. F = \langle A_F, \Psi_F \rangle \wedge A_F \not\models \Phi \wedge (\Psi_F \vdash \Phi))$ 

lemma frameImpI:
  fixes  $F$  :: ' $b$  frame
  and  $\varphi$  :: ' $c$ 
  and  $A_F :: name\ list$ 
  and  $\Psi_F :: 'b$ 
  assumes  $F = \langle A_F, \Psi_F \rangle$ 
  and  $A_F \not\models \varphi$ 
  and  $\Psi_F \vdash \varphi$ 
  shows  $F \vdash_F \varphi$ 
   $\langle proof \rangle$ 

lemma frameImpAlphaEnt:

```

```

fixes  $A_F :: \text{name list}$ 
and  $\Psi_F :: 'b$ 
and  $A_{F'} :: \text{name list}$ 
and  $\Psi_{F'} :: 'b$ 
and  $\varphi :: 'c$ 

assumes  $\langle A_F, \Psi_F \rangle = \langle A_{F'}, \Psi_{F'} \rangle$ 
and  $A_F \#* \varphi$ 
and  $A_{F'} \#* \varphi$ 
and  $\Psi_{F'} \vdash \varphi$ 

shows  $\Psi_F \vdash \varphi$ 
⟨proof⟩

lemma frameImpEAux:
fixes  $F :: 'b \text{ frame}$ 
and  $\Phi :: 'c$ 

assumes  $F \vdash_F \Phi$ 
and  $F = \langle A_F, \Psi_F \rangle$ 
and  $A_F \#* \Phi$ 

shows  $\Psi_F \vdash \Phi$ 
⟨proof⟩

lemma frameImpE:
fixes  $F :: 'b \text{ frame}$ 
and  $\Phi :: 'c$ 

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

shows  $\Psi_F \vdash \Phi$ 
⟨proof⟩

lemma frameImpClosed:
fixes  $F :: 'b \text{ frame}$ 
and  $\Phi :: 'c$ 
and  $p :: \text{name prm}$ 

assumes  $F \vdash_F \Phi$ 

shows  $(p \cdot F) \vdash_F (p \cdot \Phi)$ 
⟨proof⟩

lemma frameImpEqvt[eqvt]:
fixes  $F :: 'b \text{ frame}$ 
and  $\Phi :: 'c$ 
and  $p :: \text{name prm}$ 

```

shows $(p \cdot (F \vdash_F \Phi)) = (p \cdot F) \vdash_F (p \cdot \Phi)$
 $\langle proof \rangle$

lemma *frameImpEmpty*[simp]:
fixes $\Psi :: 'b$
and $\varphi :: 'c$

shows $\langle \varepsilon, \Psi \rangle \vdash_F \varphi = \Psi \vdash \varphi$
 $\langle proof \rangle$

definition

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

definition

$FrameStatEq :: 'b frame \Rightarrow 'b frame \Rightarrow \text{bool}$ (**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 $p :: \text{name prm}$

assumes $F \hookrightarrow_F G$

shows $(p \cdot F) \hookrightarrow_F (p \cdot G)$
 $\langle proof \rangle$

lemma *FrameStatEqClosed*:

fixes $F :: 'b frame$
and $G :: 'b frame$
and $p :: \text{name prm}$

assumes $F \simeq_F G$

shows $(p \cdot F) \simeq_F (p \cdot G)$
 $\langle proof \rangle$

lemma *FrameStatImpEqvt*[eqvt]:

fixes $F :: 'b frame$
and $G :: 'b frame$
and $p :: \text{name prm}$

shows $(p \cdot (F \hookrightarrow_F G)) = ((p \cdot F) \hookrightarrow_F (p \cdot G))$
 $\langle proof \rangle$

lemma *FrameStatEqEqvt*[eqvt]:
fixes $F :: 'b frame$

```

and    $G :: 'b \text{ frame}$ 
and    $p :: \text{name} \text{ prm}$ 

shows  $(p \cdot (F \simeq_F G)) = ((p \cdot F) \simeq_F (p \cdot G))$ 
 $\langle proof \rangle$ 

lemma FrameStatImpRefl[simp]:
fixes  $F :: 'b \text{ frame}$ 

shows  $F \hookrightarrow_F F$ 
 $\langle proof \rangle$ 

lemma FrameStatEqRefl[simp]:
fixes  $F :: 'b \text{ frame}$ 

shows  $F \simeq_F F$ 
 $\langle proof \rangle$ 

lemma FrameStatEqSym:
fixes  $F :: 'b \text{ frame}$ 
and    $G :: 'b \text{ frame}$ 

assumes  $F \simeq_F G$ 

shows  $G \simeq_F F$ 
 $\langle proof \rangle$ 

lemma FrameStatImpTrans:
fixes  $F :: 'b \text{ frame}$ 
and    $G :: 'b \text{ frame}$ 
and    $H :: 'b \text{ frame}$ 

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

shows  $F \hookrightarrow_F H$ 
 $\langle proof \rangle$ 

lemma FrameStatEqTrans:
fixes  $F :: 'b \text{ frame}$ 
and    $G :: 'b \text{ frame}$ 
and    $H :: 'b \text{ frame}$ 

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

shows  $F \simeq_F H$ 
 $\langle proof \rangle$ 

```

```

lemma fsCompose[simp]: finite((supp SCompose)::name set)
⟨proof⟩

nominal-primrec
  insertAssertion :: 'b frame ⇒ 'b ⇒ 'b frame
where
  insertAssertion (FAssert Ψ) Ψ' = FAssert (Ψ' ⊗ Ψ)
  |  $x \notin \Psi' \implies \text{insertAssertion} ((\nu x)F) \Psi' = (\nu x)(\text{insertAssertion} F \Psi')$ 
⟨proof⟩

lemma insertAssertionEqvt[eqvt]:
  fixes  $p$  :: name prm
  and  $F$  :: 'b frame
  and  $\Psi$  :: 'b

  shows  $p \cdot (\text{insertAssertion} F \Psi) = \text{insertAssertion} (p \cdot F) (p \cdot \Psi)$ 
⟨proof⟩

nominal-primrec
  mergeFrame :: 'b frame ⇒ 'b frame ⇒ 'b frame
where
  mergeFrame (FAssert Ψ) G = insertAssertion G Ψ
  |  $x \notin G \implies \text{mergeFrame} ((\nu x)F) G = (\nu x)(\text{mergeFrame} F G)$ 
⟨proof⟩

notation mergeFrame (infixr ⟨ $\otimes_F$ ⟩ 80)

abbreviation
  frameBottomJudge (⟨ $\perp_F$ ⟩) where  $\perp_F \equiv (\text{FAssert} SBottom)$ 

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

  shows  $p \cdot (\text{mergeFrame} F G) = \text{mergeFrame} (p \cdot F) (p \cdot G)$ 
⟨proof⟩

nominal-primrec
  extractFrame :: ('a, 'b, 'c) psi ⇒ 'b frame
  and extractFrame' :: ('a, 'b, 'c) input ⇒ 'b frame
  and extractFrame'' :: ('a, 'b, 'c) psiCase ⇒ 'b frame

where
  extractFrame (0) = ⟨ε, ⊥⟩
  | extractFrame (M(I)) = ⟨ε, ⊥⟩
  | extractFrame (M⟨N⟩.P) = ⟨ε, ⊥⟩
  | extractFrame (Case C) = ⟨ε, ⊥⟩

```

```

| extractFrame ( $P \parallel Q$ ) = ( $\text{extractFrame } P$ )  $\otimes_F$  ( $\text{extractFrame } Q$ )
| extractFrame (( $\{\Psi\}::('a, 'b, 'c) \psi$ )) =  $\langle \varepsilon, \Psi \rangle$ 

| extractFrame (( $\nu x$ ) $P$ ) = ( $\nu x$ ) $(\text{extractFrame } P)$ 
| extractFrame (! $P$ ) =  $\langle \varepsilon, \perp \rangle$ 

| extractFrame' (( $\text{Trm } M P$ ) $::('a::fs-name, 'b::fs-name, 'c::fs-name) \text{ input}$ ) =  $\langle \varepsilon, \perp \rangle$ 

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

| extractFrame'' ( $\perp_c::('a::fs-name, 'b::fs-name, 'c::fs-name) \psi$ ) =  $\langle \varepsilon, \perp \rangle$ 
| extractFrame'' ( $\Box \Phi \Rightarrow P C$ ) =  $\langle \varepsilon, \perp \rangle$ 
⟨proof⟩

lemmas extractFrameSimps = extractFrame-extractFrame'-extractFrame''.simp

lemma extractFrameEqvt[eqvt]:
  fixes  $p :: \text{name}$   $\text{prm}$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $I :: ('a, 'b, 'c) \text{ input}$ 
  and  $C :: ('a, 'b, 'c) \psi$ 
  shows  $p \cdot (\text{extractFrame } P) = \text{extractFrame } (p \cdot P)$ 
  and  $p \cdot (\text{extractFrame}' I) = \text{extractFrame}' (p \cdot I)$ 
  and  $p \cdot (\text{extractFrame}'' C) = \text{extractFrame}'' (p \cdot C)$ 
⟨proof⟩

lemma insertAssertionFresh[intro]:
  fixes  $F :: 'b \text{ frame}$ 
  and  $\Psi :: 'b$ 
  and  $x :: \text{name}$ 
  assumes  $x \notin F$ 
  and  $x \notin \Psi$ 
  shows  $x \notin (\text{insertAssertion } F \Psi)$ 
⟨proof⟩

lemma insertAssertionFreshChain[intro]:
  fixes  $F :: 'b \text{ frame}$ 
  and  $\Psi :: 'b$ 
  and  $xvec :: \text{name list}$ 
  and  $Xs :: \text{name set}$ 
  shows  $\llbracket xvec \#* F; xvec \#* \Psi \rrbracket \implies xvec \#* (\text{insertAssertion } F \Psi)$ 
  and  $\llbracket Xs \#* F; Xs \#* \Psi \rrbracket \implies Xs \#* (\text{insertAssertion } F \Psi)$ 
⟨proof⟩

lemma mergeFrameFresh[intro]:

```

```

fixes F :: 'b frame
and G :: 'b frame
and x :: name

shows [x # F; x # G]  $\implies$  x # (mergeFrame F G)
⟨proof⟩

```

```

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

shows [xvec #* F; xvec #* G]  $\implies$  xvec #* (mergeFrame F G)
and [Xs #* F; Xs #* G]  $\implies$  Xs #* (mergeFrame F G)
⟨proof⟩

```

```

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 # P  $\implies$  x # extractFrame P
and x # I  $\implies$  x # extractFrame' I
and x # C  $\implies$  x # extractFrame'' C
⟨proof⟩

```

```

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 #* P  $\implies$  xvec #* extractFrame P
and xvec #* I  $\implies$  xvec #* extractFrame' I
and xvec #* C  $\implies$  xvec #* extractFrame'' C
and Xs #* P  $\implies$  Xs #* extractFrame P
and Xs #* I  $\implies$  Xs #* extractFrame' I
and Xs #* C  $\implies$  Xs #* extractFrame'' C
⟨proof⟩

```

```

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  $P \implies x \notin (\text{extractFrame } P)$ 
and   guarded'  $I \implies x \notin (\text{extractFrame}' I)$ 
and   guarded''  $C \implies x \notin (\text{extractFrame}'' C)$ 
⟨proof⟩

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

shows (xvec #* (( $\nu$ *yvec)F)) = ( $\forall x \in \text{set } xvec. x \in \text{set } yvec \vee x \notin F$ )
⟨proof⟩

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

shows xvec #* (FAssert Ψ) = xvec #* Ψ
and   Xs #* (FAssert Ψ) = Xs #* Ψ
⟨proof⟩

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

assumes xvec #* yvec

shows xvec #* (( $\nu$ *yvec)F) = xvec #* F
⟨proof⟩

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

assumes x # xvec

shows x # (( $\nu$ *xvec)F) = x # F
⟨proof⟩

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

shows xvec #* ( $\perp_F$ )

```

```

and    $Xs \#* (\perp_F)$ 
⟨proof⟩

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

shows  $xvec \#* (SBottom)$ 
and    $Xs \#* (SBottom)$ 
⟨proof⟩

lemma  $freshFrameDest[dest]$ :
fixes  $A_F :: name list$ 
and    $\Psi_F :: 'b$ 
and    $xvec :: name list$ 

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

shows  $xvec \#* A_F \implies xvec \#* \Psi_F$ 
and    $A_F \#* xvec \implies xvec \#* \Psi_F$ 
⟨proof⟩

lemma  $insertAssertionSimps[simp]$ :
fixes  $A_F :: name list$ 
and    $\Psi_F :: 'b$ 
and    $\Psi :: 'b$ 

assumes  $A_F \#* \Psi$ 

shows  $insertAssertion (\langle A_F, \Psi_F \rangle) \Psi = \langle A_F, \Psi \otimes \Psi_F \rangle$ 
⟨proof⟩

lemma  $mergeFrameSimps[simp]$ :
fixes  $A_F :: name list$ 
and    $\Psi_F :: 'b$ 
and    $\Psi :: 'b$ 

assumes  $A_F \#* \Psi$ 

shows  $(\langle A_F, \Psi_F \rangle) \otimes_F \langle \varepsilon, \Psi \rangle = \langle A_F, \Psi_F \otimes \Psi \rangle$ 
⟨proof⟩

lemma  $mergeFrames[simp]$ :
fixes  $A_F :: name list$ 
and    $\Psi_F :: 'b$ 
and    $A_G :: name list$ 
and    $\Psi_G :: 'b$ 

assumes  $A_F \#* A_G$ 

```

```

and       $A_F \#* \Psi_G$ 
and       $A_G \#* \Psi_F$ 

shows  $(\langle A_F, \Psi_F \rangle) \otimes_F (\langle A_G, \Psi_G \rangle) = (\langle (A_F @ A_G), \Psi_F \otimes \Psi_G \rangle)$ 
 $\langle proof \rangle$ 

lemma frameImpResFreshLeft:
  fixes  $F :: 'b \text{ frame}$ 
  and    $x :: \text{name}$ 

  assumes  $x \notin F$ 

  shows  $(\nu x)F \hookrightarrow_F F$ 
 $\langle proof \rangle$ 

lemma frameImpResFreshRight:
  fixes  $F :: 'b \text{ frame}$ 
  and    $x :: \text{name}$ 

  assumes  $x \notin F$ 

  shows  $F \hookrightarrow_F (\nu x)F$ 
 $\langle proof \rangle$ 

lemma frameResFresh:
  fixes  $F :: 'b \text{ frame}$ 
  and    $x :: \text{name}$ 

  assumes  $x \notin F$ 

  shows  $(\nu x)F \simeq_F F$ 
 $\langle proof \rangle$ 

lemma frameImpResPres:
  fixes  $F :: 'b \text{ frame}$ 
  and    $G :: 'b \text{ frame}$ 
  and    $x :: \text{name}$ 

  assumes  $F \hookrightarrow_F G$ 

  shows  $(\nu x)F \hookrightarrow_F (\nu x)G$ 
 $\langle proof \rangle$ 

lemma frameResPres:
  fixes  $F :: 'b \text{ frame}$ 
  and    $G :: 'b \text{ frame}$ 
  and    $x :: \text{name}$ 

  assumes  $F \simeq_F G$ 

```

```

shows  $(\nu x)F \simeq_F (\nu x)G$ 
⟨proof⟩

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⟩

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

shows  $(\nu x)((\nu y)F) \simeq_F (\nu y)((\nu x)F)$ 
⟨proof⟩

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

shows  $(\nu x)((\nu*xvec)F) \hookrightarrow_F (\nu*xvec)((\nu x)F)$ 
⟨proof⟩

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

shows  $(\nu*xvec)((\nu x)F) \hookrightarrow_F (\nu x)((\nu*xvec)F)$ 
⟨proof⟩

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

shows  $(\nu x)((\nu*xvec)F) \simeq_F (\nu*xvec)((\nu x)F)$ 
⟨proof⟩

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

```

```

shows  $(\nu*xvec)(\nu*yvec)F \hookrightarrow_F (\nu*yvec)(\nu*xvec)F$ 
⟨proof⟩

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

shows  $(\nu*xvec)(\nu*yvec)F \simeq_F (\nu*yvec)(\nu*xvec)F$ 
⟨proof⟩

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

shows  $(\varepsilon, \Psi) \hookrightarrow_F (\varepsilon, \Psi') = (\Psi \hookrightarrow \Psi')$ 
⟨proof⟩

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

shows  $(\varepsilon, \Psi) \simeq_F (\varepsilon, \Psi') = (\Psi \simeq \Psi')$ 
⟨proof⟩

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

shows extractFrame( $\nu*xvec)P \hookrightarrow_F (\nu*xvec)(extractFrame P)$ 
⟨proof⟩

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

shows extractFrame( $\nu*xvec)P \simeq_F (\nu*xvec)(extractFrame P)$ 
⟨proof⟩

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

assumes xvec #* Ψ

shows insertAssertion(extractFrame( $\nu*xvec)P)) \Psi \hookrightarrow_F (\nu*xvec)(insertAssertion(extractFrame P) \Psi)$ 
```

$\langle proof \rangle$

lemma *insertAssertionExtractFrameFresh*:

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

assumes *xvec* $\sharp*$ Ψ

shows *insertAssertion*(*extractFrame*(($\nu*xvec$)*P*)) $\Psi \simeq_F (\nu*xvec)$ (*insertAssertion*(*extractFrame P*) Ψ)

$\langle proof \rangle$

lemma *frameImpResChainPres*:

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

assumes *F* $\hookrightarrow_F G$

shows ($\nu*xvec$)*F* $\hookrightarrow_F (\nu*xvec)*G*$

$\langle proof \rangle$

lemma *frameResChainPres*:

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

assumes *F* $\simeq_F G$

shows ($\nu*xvec$)*F* $\simeq_F (\nu*xvec)*G*$

$\langle proof \rangle$

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* $\sharp*$ Ψ

and *distinct A_F*

obtains Ψ_F **where** *F* = $\langle A_F, \Psi_F \rangle$ **and** $\Psi' = \Psi \otimes \Psi_F$

$\langle proof \rangle$

lemma *mergeFrameE*:

fixes *F* :: '*b frame*

```

and    $G :: 'b \text{ frame}$ 
and    $A_{FG} :: \text{name list}$ 
and    $\Psi_{FG} :: 'b$ 

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

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  $A_F \nparallel \Psi_G$  and  $A_G \nparallel \Psi_F$ 
           $\langle \text{proof} \rangle$ 

lemma  $\text{mergeFrameRes1[simp]}$ :
fixes  $A_F :: \text{name list}$ 
and    $\Psi_F :: 'b$ 
and    $x :: \text{name}$ 
and    $A_G :: \text{name list}$ 
and    $\Psi_G :: 'b$ 

assumes  $A_F \nparallel \Psi_G$ 
and    $A_F \nparallel A_G$ 
and    $x \notin A_F$ 
and    $x \notin \Psi_F$ 
and    $A_G \nparallel \Psi_F$ 

shows  $(\langle A_F, \Psi_F \rangle) \otimes_F ((\nu x)(\langle A_G, \Psi_G \rangle)) = (((A_F @ x \# A_G), \Psi_F \otimes \Psi_G))$ 
           $\langle \text{proof} \rangle$ 

lemma  $\text{mergeFrameRes2[simp]}$ :
fixes  $A_F :: \text{name list}$ 
and    $\Psi_F :: 'b$ 
and    $x :: \text{name}$ 
and    $A_G :: \text{name list}$ 
and    $\Psi_G :: 'b$ 

assumes  $A_F \nparallel \Psi_G$ 
and    $A_G \nparallel A_F$ 
and    $x \notin A_F$ 
and    $x \notin \Psi_F$ 
and    $A_G \nparallel \Psi_F$ 

shows  $(\langle A_F, \Psi_F \rangle) \otimes_F ((\nu x)(\langle A_G, \Psi_G \rangle)) = (((A_F @ x \# A_G), \Psi_F \otimes \Psi_G))$ 
           $\langle \text{proof} \rangle$ 

lemma  $\text{insertAssertionResChain[simp]}$ :
fixes  $xvec :: \text{name list}$ 
and    $F :: 'b \text{ frame}$ 
and    $\Psi :: 'b$ 

```

```

assumes xvec #* Ψ

shows insertAssertion ((|ν*xvec|)F) Ψ = (|ν*xvec|)(insertAssertion F Ψ)
⟨proof⟩

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

shows extractFrame((|ν*xvec|)P) = (|ν*xvec|)(extractFrame P)
⟨proof⟩

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

assumes xvec #* F

shows (|ν*xvec|)F ≈_F F
⟨proof⟩

end

locale assertion = assertionAux SCompose SImp SBottom SChanEq
for SCompose :: 'b::fs-name ⇒ 'b ⇒ 'b
and SImp :: 'b ⇒ 'c::fs-name ⇒ bool
and SBottom :: 'b
and SChanEq :: 'a::fs-name ⇒ 'a ⇒ 'c +
assumes chanEqSym: SImp Ψ (SChanEq M N) ⇒ SImp Ψ (SChanEq N M)
and chanEqTrans: [SImp Ψ (SChanEq M N); SImp Ψ (SChanEq N L)] ⇒
SImp Ψ (SChanEq M L)
and Composition: assertionAux.AssertionStatEq SImp Ψ Ψ' ⇒ assertionAux.AssertionStatEq SImp (SCompose Ψ Ψ') (SCompose Ψ' Ψ')
and Identity: assertionAux.AssertionStatEq SImp (SCompose Ψ SBottom) Ψ
and Associativity: assertionAux.AssertionStatEq SImp (SCompose (SCompose Ψ Ψ') Ψ'') (SCompose Ψ (SCompose Ψ' Ψ''))
and Commutativity: assertionAux.AssertionStatEq SImp (SCompose Ψ Ψ') (SCompose Ψ' Ψ)

begin

notation SCompose (infixr ⟨⊗⟩ 90)
notation SImp (⟨- ⊢ -⟩ [85, 85] 85)
notation SChanEq (⟨- ↔ -⟩ [90, 90] 90)
notation SBottom (⟨⊥⟩ 90)

```

```

lemma compositionSym:
  fixes  $\Psi$  :: 'b
  and    $\Psi'$  :: 'b
  and    $\Psi''$  :: 'b

  assumes  $\Psi \simeq \Psi'$ 

  shows  $\Psi'' \otimes \Psi \simeq \Psi'' \otimes \Psi'$ 
   $\langle proof \rangle$ 

lemma Composition':
  fixes  $\Psi$  :: 'b
  and    $\Psi'$  :: 'b
  and    $\Psi''$  :: 'b
  and    $\Psi'''$  :: 'b

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

  shows  $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi'''$ 
   $\langle proof \rangle$ 

lemma composition':
  fixes  $\Psi$  :: 'b
  and    $\Psi'$  :: 'b
  and    $\Psi''$  :: 'b
  and    $\Psi'''$  :: 'b

  assumes  $\Psi \simeq \Psi'$ 

  shows  $(\Psi \otimes \Psi'') \otimes \Psi''' \simeq (\Psi' \otimes \Psi'') \otimes \Psi'''$ 
   $\langle proof \rangle$ 

lemma associativitySym:
  fixes  $\Psi$  :: 'b
  and    $\Psi'$  :: 'b
  and    $\Psi''$  :: 'b

  shows  $(\Psi \otimes \Psi') \otimes \Psi'' \simeq (\Psi \otimes \Psi'') \otimes \Psi'$ 
   $\langle proof \rangle$ 

lemma frameIntAssociativity:
  fixes  $A_F$  :: name list
  and    $\Psi$  :: 'b
  and    $\Psi'$  :: 'b
  and    $\Psi''$  :: 'b

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

```

$\langle proof \rangle$

lemma *frameIntCommutativity*:

fixes $A_F :: name\ list$
 and $\Psi :: 'b$
 and $\Psi' :: 'b$

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

lemma *frameIntIdentity*:

fixes $A_F :: name\ list$
 and $\Psi_F :: 'b$

shows $\langle A_F, \Psi_F \otimes SBottom \rangle \simeq_F \langle A_F, \Psi_F \rangle$
 $\langle proof \rangle$

lemma *frameIntComposition*:

fixes $\Psi :: 'b$
 and $\Psi' :: 'b$
 and $A_F :: name\ list$
 and $\Psi_F :: 'b$

assumes $\Psi \simeq \Psi'$

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

lemma *frameIntCompositionSym*:

fixes $\Psi :: 'b$
 and $\Psi' :: 'b$
 and $A_F :: name\ list$
 and $\Psi_F :: 'b$

assumes $\Psi \simeq \Psi'$

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

lemma *frameCommutativity*:

fixes $F :: 'b\ frame$
 and $G :: 'b\ frame$

shows $F \otimes_F G \simeq_F G \otimes_F F$
 $\langle proof \rangle$

lemma *frameScopeExt*:

fixes $x :: name$
 and $F :: 'b\ frame$

```

and     $G :: 'b \text{ frame}$ 
assumes  $x \notin F$ 
shows  $(\nu x)(F \otimes_F G) \simeq_F F \otimes_F ((\nu x)G)$ 
 $\langle proof \rangle$ 

lemma insertDoubleAssertionStatEq:
fixes  $F :: 'b \text{ frame}$ 
and  $\Psi :: 'b$ 
and  $\Psi' :: 'b$ 

shows insertAssertion(insertAssertion F Ψ) Ψ' ≈F (insertAssertion F) (Ψ ⊗ Ψ')
 $\langle proof \rangle$ 

lemma guardedStatEq:
fixes  $P :: ('a, 'b, 'c) \psi$ 
and  $I :: ('a, 'b, 'c) \text{ input}$ 
and  $C :: ('a, 'b, 'c) \psiCase$ 
and  $A_P :: \text{name list}$ 
and  $\Psi_P :: 'b$ 

shows  $\llbracket \text{guarded } P; \text{ extractFrame } P = \langle A_P, \Psi_P \rangle \rrbracket \implies \Psi_P \simeq \perp \wedge \text{supp } \Psi_P = (\{\} :: \text{name set})$ 
and  $\llbracket \text{guarded}' I; \text{ extractFrame}' I = \langle A_P, \Psi_P \rangle \rrbracket \implies \Psi_P \simeq \perp \wedge \text{supp } \Psi_P = (\{\} :: \text{name set})$ 
and  $\llbracket \text{guarded}'' C; \text{ extractFrame}'' C = \langle A_P, \Psi_P \rangle \rrbracket \implies \Psi_P \simeq \perp \wedge \text{supp } \Psi_P = (\{\} :: \text{name set})$ 
 $\langle proof \rangle$ 

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 ( $\langle \cdot \leftarrow \cdot \rangle [110, 110] 110$ )
  | BStep «name» ('a, 'b, 'c) boundOutput
    ( $\langle (\nu \cdot) \cdot \rangle [110, 110] 110$ )

primrec BOresChain :: name list  $\Rightarrow$  ('a::fs-name, 'b::fs-name, 'c::fs-name) boundOutput  $\Rightarrow$ 
  ('a, 'b, 'c) boundOutput where
  Base: BOresChain [] B = B
  | Step: BOresChain (x#xs) B =  $(\nu x)(BOresChain xs B)$ 

```

abbreviation

BOresChainJudge ($\langle (\nu * \text{-}) \rightarrow [80, 80] \rangle 80$) **where** $(\nu * xvec)B \equiv BOresChain xvec B$

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 $((\nu * xvec)B) = (\nu * (\text{perm} \cdot xvec))(\text{perm} \cdot B)$
{proof}

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 $((\nu * xvec)B = (\nu * xvec)B') = (B = B')$
{proof}

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 $(Xs \#* (N \prec' P)) = ((Xs \#* N) \wedge (Xs \#* P))$
and $(xvec \#* (N \prec' P)) = ((xvec \#* N) \wedge (xvec \#* P))$
{proof}

lemma *boundOutputFresh*:

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

shows $(x \notin (\nu * xvec)B) = (x \in \text{set } xvec \vee x \notin B)$
{proof}

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  $Xs \#* ((\nu*xvec)B) = (\forall x \in Xs. x \in set\ xvec \vee x \notin B)$ 
and    $yvec \#* ((\nu*xvec)B) = (\forall x \in (set\ yvec). x \in set\ xvec \vee x \notin B)$ 
and    $Xs \#* ((\nu x)B) = Xs \#* [x].B$ 
and    $xvec \#* ((\nu x)B) = xvec \#* [x].B$ 
(proof)

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

shows  $(supp((\nu*xvec)B)::name\ set) = (supp\ B) - (supp\ xvec)$ 
(proof)

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  $Xs \#* xvec \implies (Xs \#* ((\nu*xvec)B)) = (Xs \#* B)$ 
and    $yvec \#* xvec \implies yvec \#* ((\nu*xvec)B) = yvec \#* B$ 
and    $xvec \#* ((\nu*xvec)B)$ 
and    $x \notin xvec \implies x \notin ((\nu*xvec)B) = x \notin B$ 
(proof)

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  $xvecFreshB: (p \cdot xvec) \#* B$ 
and    $S: set\ p \subseteq set\ xvec \times set\ (p \cdot xvec)$ 
and    $(set\ xvec) \subseteq (set\ yvec)$ 

shows  $((\nu*yvec)B) = ((\nu*(p \cdot yvec))(p \cdot B))$ 
(proof)

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  $\#*$  B
and      S: set p  $\subseteq$  set xvec  $\times$  set yvec
and      yvec  $\#*$  (( $\nu$ *zvec)B)

shows (( $\nu$ *zvec)B) = (( $\nu$ *(p + zvec))|(p + B))
⟨proof⟩

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 + xvec)  $\#*$  M
and   (p + xvec)  $\#*$  P
and   set p  $\subseteq$  set xvec  $\times$  set (p + xvec)
and   (set xvec)  $\subseteq$  (set yvec)

shows (( $\nu$ *yvec)M  $\prec'$  P) = (( $\nu$ *(p + yvec))|(p + M)  $\prec'$  (p + P))
⟨proof⟩

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  $\#$  N
and   y  $\#$  P
and   x  $\in$  (set xvec)

shows ( $\nu$ *xvec)N  $\prec'$  P = ( $\nu$ *([(x, y)] + xvec))|([(x, y)] + N)  $\prec'$  ([(x, y)] + P)
⟨proof⟩

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

assumes y  $\#$  B

shows ( $\nu$ x)B = ( $\nu$ y)|([(x, y)] + B)
⟨proof⟩

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    $x \notin B$ 

shows  $y \notin C$ 
⟨proof⟩

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  $y \in supp C$ 
⟨proof⟩

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    $xvec \neq yvec$ 
and    $length xvec = length yvec$ 

shows  $\exists p. (set p) \subseteq (set xvec) \times set (yvec) \wedge distinctPerm p \wedge B = p \cdot B' \wedge$ 
 $(set (map fst p)) \subseteq (supp B) \wedge xvec \neq B' \wedge yvec \neq B$ 
⟨proof⟩

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⟩

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 #* yvec

shows  $\exists p.$  (set p)  $\subseteq$  (set xvec)  $\times$  set (yvec)  $\wedge$  distinctPerm p  $\wedge$  M = p  $\cdot$  N  $\wedge$ 
P = p  $\cdot$  Q  $\wedge$  xvec #* N  $\wedge$  xvec #* Q  $\wedge$  yvec #* M  $\wedge$  yvec #* P
⟨proof⟩

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 #* 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 #* N and xvec #* Q
and (p  $\cdot$  xvec) #* M and (p  $\cdot$  xvec) #* P
⟨proof⟩

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 #* yvec
and x #* xvec
and y #* xvec
and y #* yvec
and xvec #* yvec

```

and $x \in \text{supp } M$

shows $y \in \text{supp } N$
 $\langle \text{proof} \rangle$

lemma *boundOutputChainOpenIH*:

fixes $xvec :: \text{name list}$

and $x :: \text{name}$

and $B :: ('a::fs-name, 'b::fs-name, 'c::fs-name) \text{ boundOutput}$

and $yvec :: \text{name list}$

and $y :: \text{name}$

and $B' :: ('a, 'b, 'c) \text{ boundOutput}$

assumes $\text{Eq}: (\nu*xvec)(\nu x)B = (\nu*yvec)(\nu y)B'$

and $L: \text{length } xvec = \text{length } yvec$

and $xFreshB': x \notin B'$

and $xFreshxvec: x \notin xvec$

and $xFreshyvec: x \notin yvec$

shows $(\nu*xvec)B = (\nu*yvec)[(x, y)] \cdot B'$
 $\langle \text{proof} \rangle$

lemma *boundOutputPar1Dest*:

fixes $xvec :: \text{name list}$

and $M :: 'a::fs-name$

and $P :: ('a, 'b::fs-name, 'c::fs-name) \psi$

and $yvec :: \text{name list}$

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 $xvec \#* R$

and $yvec \#* R$

obtains T **where** $P = T \parallel R$ **and** $(\nu*xvec)M \prec' T = (\nu*yvec)N \prec' Q$
 $\langle \text{proof} \rangle$

lemma *boundOutputPar1Dest'*:

fixes $xvec :: \text{name list}$

and $M :: 'a::fs-name$

and $P :: ('a, 'b::fs-name, 'c::fs-name) \psi$

and $yvec :: \text{name list}$

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 $xvec \#* yvec$

```

obtains T p where set p ⊆ set xvec × set yvec and P = T || (p · R) and
(ℓν*xvec)M ≺' T = (ℓν*yvec)N ≺' Q
⟨proof⟩

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 (ℓν*xvec)M ≺' P = (ℓν*yvec)N ≺' (Q || R)
and xvec #* Q
and yvec #* Q

obtains T where P = Q || T and (ℓν*xvec)M ≺' T = (ℓν*yvec)N ≺' R
⟨proof⟩

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 (ℓν*xvec)M ≺' P = (ℓν*yvec)N ≺' (Q || R)
and xvec #* yvec

obtains T p where set p ⊆ set xvec × set yvec and P = (p · Q) || T and
(ℓν*xvec)M ≺' T = (ℓν*yvec)N ≺' R
⟨proof⟩

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

shows (ℓν*(xvec@yvec))B = (ℓν*xvec)((ℓν*yvec)B)
⟨proof⟩

lemma openInjectAuxAuxAux:
fixes x :: name
and xvec :: name list

```

shows $\exists y \text{ yvec. } x \# xvec = yvec @ [y] \wedge \text{length } xvec = \text{length } yvec$
 $\langle proof \rangle$

lemma *openInjectAuxAux*:

fixes $xvec1 :: \text{name list}$
and $xvec2 :: \text{name list}$
and $yvec :: \text{name list}$

assumes $\text{length}(xvec1 @ xvec2) = \text{length } yvec$

shows $\exists yvec1 yvec2. yvec = yvec1 @ yvec2 \wedge \text{length } xvec1 = \text{length } yvec1 \wedge \text{length } xvec2 = \text{length } yvec2$
 $\langle proof \rangle$

lemma *openInjectAux*:

fixes $xvec1 :: \text{name list}$
and $x :: \text{name}$
and $xvec2 :: \text{name list}$
and $yvec :: \text{name list}$

assumes $\text{length}(xvec1 @ x # xvec2) = \text{length } yvec$

shows $\exists yvec1 y yvec2. yvec = yvec1 @ y \# yvec2 \wedge \text{length } xvec1 = \text{length } yvec1 \wedge \text{length } xvec2 = \text{length } yvec2$
 $\langle proof \rangle$

lemma *boundOutputOpenDest*:

fixes $yvec :: \text{name list}$
and $M :: 'a::fs-name$
and $P :: ('a, 'b::fs-name, 'c::fs-name) \psi$
and $xvec1 :: \text{name list}$
and $x :: \text{name}$
and $xvec2 :: \text{name list}$
and $N :: 'a$
and $Q :: ('a, 'b, 'c) \psi$

assumes $\text{Eq}: (\nu*(xvec1 @ x # xvec2)) M \prec' P = (\nu*yvec) N \prec' Q$

and $x \notin xvec1$
and $x \notin yvec$
and $x \notin N$
and $x \notin Q$
and $\text{distinct } yvec$

obtains $yvec1 y yvec2$ **where** $yvec = yvec1 @ y \# yvec2$ **and** $\text{length } xvec1 = \text{length } yvec1$ **and** $\text{length } xvec2 = \text{length } yvec2$

and $(\nu*(xvec1 @ xvec2)) M \prec' P = (\nu*(yvec1 @ yvec2))([(x, y)] \cdot N) \prec'([(x, y)] \cdot Q)$
 $\langle proof \rangle$

```

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  $\notin$  xvec1
  and x  $\notin$  yvec
  and x  $\notin$  N
  and x  $\notin$  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}

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  $\notin$  xvec
  and z  $\notin$  yvec

  obtains R where P =  $(\nu z) R$  and  $(\nu*xvec) M \prec' R = (\nu*yvec) N \prec' Q$ 
  {proof}

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       $(\langle - \rangle) [90, 90] 90$ 
  | Out 'a::fs-name name list 'a  $(\langle - \rangle) (\nu * \langle - \rangle) [90, 90, 90] 90$ 
  | Tau            $(\langle \tau \rangle 90)$ 

```

```

nominal-primrec bn :: ('a::fs-name) action  $\Rightarrow$  name list
  where
    bn ( $M(N)$ ) = []
  | bn ( $M(\nu*xvec)(N)$ ) = xvec
  | bn ( $\tau$ ) = []
  ⟨proof⟩

lemma bnEqvt[eqvt]:
  fixes p :: name prm
  and   α :: ('a::fs-name) action

  shows (p · bn α) = bn(p · α)
  ⟨proof⟩

nominal-primrec create-residual :: ('a::fs-name) action  $\Rightarrow$  ('a, 'b::fs-name, 'c::fs-name)
psi  $\Rightarrow$  ('a, 'b, 'c) residual ( $\langle\cdot\rangle$  [80, 80] 80)
  where
    ( $M(N)$ )  $\prec P = RIn M N P$ 
  |  $M(\nu*xvec)(N) \prec P = ROut M ((\nu*xvec)(N \prec' P))$ 
  |  $\tau \prec P = (RTau P)$ 
  ⟨proof⟩

nominal-primrec subject :: ('a::fs-name) action  $\Rightarrow$  'a option
  where
    subject ( $M(N)$ ) = Some M
  | subject ( $M(\nu*xvec)(N)$ ) = Some M
  | subject ( $\tau$ ) = None
  ⟨proof⟩

nominal-primrec object :: ('a::fs-name) action  $\Rightarrow$  'a option
  where
    object ( $M(N)$ ) = Some N
  | object ( $M(\nu*xvec)(N)$ ) = Some N
  | object ( $\tau$ ) = None
  ⟨proof⟩

lemma optionFreshChain[simp]:
  fixes xvec :: name list
  and   X   :: name set

  shows xvec #* (Some x) = xvec #* x
  and   X #* (Some x) = X #* x
  and   xvec #* None
  and   X #* None
  ⟨proof⟩

lemmas [simp] = fresh-some fresh-none

lemma actionFresh[simp]:

```

```

fixes x :: name
and    $\alpha$  :: ('a::fs-name) action

shows ( $x \# \alpha$ ) = ( $x \# (\text{subject } \alpha)$   $\wedge$   $x \# (\text{bn } \alpha)$   $\wedge$   $x \# (\text{object } \alpha)$ )
⟨proof⟩

lemma actionFreshChain[simp]:
fixes X :: name set
and    $\alpha$  :: ('a::fs-name) action
and   xvec :: name list

shows ( $X \#* \alpha$ ) = ( $X \#* (\text{subject } \alpha)$   $\wedge$   $X \#* (\text{bn } \alpha)$   $\wedge$   $X \#* (\text{object } \alpha)$ )
and   ( $xvec \#* \alpha$ ) = ( $xvec \#* (\text{subject } \alpha)$   $\wedge$   $xvec \#* (\text{bn } \alpha)$   $\wedge$   $xvec \#* (\text{object } \alpha)$ )
⟨proof⟩

lemma subjectEqvt[eqvt]:
fixes p :: name prm
and    $\alpha$  :: ('a::fs-name) action

shows ( $p \cdot \text{subject } \alpha$ ) =  $\text{subject}(p \cdot \alpha)$ 
⟨proof⟩

lemma objectEqvt[eqvt]:
fixes p :: name prm
and    $\alpha$  :: ('a::fs-name) action

shows ( $p \cdot \text{object } \alpha$ ) =  $\text{object}(p \cdot \alpha)$ 
⟨proof⟩

lemma create-residualEqvt[eqvt]:
fixes p :: name prm
and    $\alpha$  :: ('a::fs-name) action
and   P :: ('a, 'b::fs-name, 'c::fs-name) psi

shows ( $p \cdot (\alpha \prec P)$ ) = ( $p \cdot \alpha$ )  $\prec$  ( $p \cdot P$ )
⟨proof⟩

lemma residualFresh:
fixes x :: name
and    $\alpha$  :: 'a::fs-name action
and   P :: ('a, 'b::fs-name, 'c::fs-name) psi

shows ( $x \# (\alpha \prec P)$ ) = ( $x \# (\text{subject } \alpha)$   $\wedge$  ( $x \in (\text{set}(\text{bn}(\alpha)))$   $\vee$  ( $x \# \text{object}(\alpha)$   $\wedge$ 
 $x \# P$ )))
⟨proof⟩

lemma residualFresh2[simp]:
fixes x :: name
and    $\alpha$  :: ('a::fs-name) action

```

```

and    $P :: ('a, 'b::fs-name, 'c::fs-name) \psi$ 

assumes  $x \notin \alpha$ 
and     $x \notin P$ 

shows  $x \notin \alpha \prec P$ 
⟨proof⟩

lemma residualFreshChain2[simp]:
fixes  $xvec :: name list$ 
and    $X :: name set$ 
and    $\alpha :: ('a::fs-name) action$ 
and    $P :: ('a, 'b::fs-name, 'c::fs-name) \psi$ 

shows  $\llbracket xvec \#* \alpha; xvec \#* P \rrbracket \implies xvec \#* (\alpha \prec P)$ 
and    $\llbracket X \#* \alpha; X \#* P \rrbracket \implies X \#* (\alpha \prec P)$ 
⟨proof⟩

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  $x \notin (M(N) \prec P) = (x \notin M \wedge x \notin N \wedge x \notin P)$ 
and    $x \notin (M(\nu*xvec)\langle N \rangle \prec P) = (x \notin M \wedge x \notin (\nu*xvec)(N \prec' P))$ 
and    $x \notin (\tau \prec P) = (x \notin P)$ 
⟨proof⟩

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⟩

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 $xvec \#* (M(N) \prec P) = (xvec \#* M \wedge xvec \#* N \wedge xvec \#* P)$
and $xvec \#* (M(\nu*yvec)\langle N \rangle \prec P) = (xvec \#* M \wedge xvec \#* ((\nu*yvec)(N \prec' P)))$
and $xvec \#* (\tau \prec P) = (xvec \#* P)$
and $X \#* (M(N) \prec P) = (X \#* M \wedge X \#* N \wedge X \#* P)$
and $X \#* (M(\nu*yvec)\langle N \rangle \prec P) = (X \#* M \wedge X \#* ((\nu*yvec)(N \prec' P)))$
and $X \#* (\tau \prec P) = (X \#* P)$
 $\langle proof \rangle$

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 $xvec \#* (RIn M N P) = (xvec \#* M \wedge xvec \#* N \wedge xvec \#* P)$
and $xvec \#* (ROut M B) = (xvec \#* M \wedge xvec \#* B)$
and $xvec \#* (RTau P) = (xvec \#* P)$
and $X \#* (RIn M N P) = (X \#* M \wedge X \#* N \wedge X \#* P)$
and $X \#* (ROut M B) = (X \#* M \wedge X \#* B)$
and $X \#* (RTau P) = (X \#* P)$
 $\langle proof \rangle$

lemma *freshResidual3[dest]*:
fixes $x :: name$
and $\alpha :: ('a::fs-name) action$
and $P :: ('a, 'b::fs-name, 'c::fs-name) psi$

assumes $x \# bn \alpha$
and $x \# \alpha \prec P$

shows $x \# \alpha \text{ and } x \# P$
 $\langle proof \rangle$

lemma *freshResidualChain3[dest]*:
fixes $xvec :: name list$
and $\alpha :: ('a::fs-name) action$
and $P :: ('a, 'b::fs-name, 'c::fs-name) psi$

assumes $xvec \#* (\alpha \prec P)$
and $xvec \#* bn \alpha$

shows $xvec \#* \alpha \text{ and } xvec \#* P$
 $\langle proof \rangle$

lemma *freshResidual4[dest]*:

```

fixes x :: name
and   α :: ('a::fs-name) action
and   P :: ('a, 'b::fs-name, 'c::fs-name) psi

assumes x # α ⊢ P

shows x # subject α
⟨proof⟩

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 #* (α ⊢ P)

shows xvec #* subject α
⟨proof⟩

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 · xvec) #* N
and   (p · xvec) #* P
and   set p ⊆ set xvec × set(p · xvec)
and   set xvec ⊆ set yvec

shows M(ν*yvec)(N) ⊢ P = M(ν*(p · yvec))(p · N) ⊢ (p · P)
⟨proof⟩

lemmas[simp del] = create-residual.simps

lemma residualInject'':
assumes bn α = bn β

shows (α ⊢ P = β ⊢ Q) = (α = β ∧ P = Q)
⟨proof⟩

lemmas residualInject = residual.inject create-residual.simps residualInject' residualInject''

lemma bnFreshResidual[simp]:
fixes α :: ('a::fs-name) action

```

shows $(bn \alpha) \#* (\alpha \prec P) = bn \alpha \#* (subject \alpha)$
 $\langle proof \rangle$

lemma *actionCases*[case-names *cInput cOutput cTau*]:
fixes $\alpha :: ('a::fs-name) action$

assumes $\bigwedge M N. \alpha = M(N) \implies Prop$
and $\bigwedge M xvec N. \alpha = M(\nu*xvec)(N) \implies Prop$
and $\alpha = \tau \implies Prop$

shows *Prop*
 $\langle proof \rangle$

lemma *actionPar1Dest*:

fixes $\alpha :: ('a::fs-name) action$
and $P :: ('a, 'b::fs-name, 'c::fs-name) psi$
and $\beta :: ('a::fs-name) action$
and $Q :: ('a, 'b, 'c) psi$
and $R :: ('a, 'b, 'c) psi$

assumes $\alpha \prec P = \beta \prec (Q \parallel R)$
and $bn \alpha \#* bn \beta$

obtains $T p$ **where** $set p \subseteq set(bn \alpha) \times set(bn \beta)$ **and** $P = T \parallel (p \cdot R)$ **and**
 $\alpha \prec T = \beta \prec Q$
 $\langle proof \rangle$

lemma *actionPar2Dest*:

fixes $\alpha :: ('a::fs-name) action$
and $P :: ('a, 'b::fs-name, 'c::fs-name) psi$
and $\beta :: ('a::fs-name) action$
and $Q :: ('a, 'b, 'c) psi$
and $R :: ('a, 'b, 'c) psi$

assumes $\alpha \prec P = \beta \prec (Q \parallel R)$
and $bn \alpha \#* bn \beta$

obtains $T p$ **where** $set p \subseteq set(bn \alpha) \times set(bn \beta)$ **and** $P = (p \cdot Q) \parallel T$ **and**
 $\alpha \prec T = \beta \prec R$
 $\langle proof \rangle$

lemma *actionScopeDest*:

fixes $\alpha :: ('a::fs-name) action$
and $P :: ('a, 'b::fs-name, 'c::fs-name) psi$
fixes $\beta :: ('a::fs-name) action$
and $x :: name$
and $Q :: ('a, 'b, 'c) psi$

assumes $\alpha \prec P = \beta \prec (\nu x) Q$

and $x \# bn \alpha$
and $x \# bn \beta$

obtains R **where** $P = (\nu x)R$ **and** $\alpha \prec R = \beta \prec Q$
 $\langle proof \rangle$

abbreviation

outputJudge ($\langle -\langle - \rangle \rangle [110, 110] 110$) **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)  $\Rightarrow$  name list  $\Rightarrow$  'a::fs-name list  $\Rightarrow$  'a
and substAssert :: ('b::fs-name)  $\Rightarrow$  name list  $\Rightarrow$  'a::fs-name list  $\Rightarrow$  'b
and substCond :: ('c::fs-name)  $\Rightarrow$  name list  $\Rightarrow$  'a::fs-name list  $\Rightarrow$  'c
and SCompose' :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b
and SImp' :: 'b  $\Rightarrow$  'c  $\Rightarrow$  bool
and SBOTTOM' :: 'b
and SChaneq' :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'c
begin
notation SCompose' (infixr  $\langle \otimes \rangle$  90)
notation SImp' ( $\langle - \vdash \rightarrow [85, 85] 85$ )
notation FrameImp ( $\langle - \vdash_F \rightarrow [85, 85] 85$ )
abbreviation
  FBOTTOMJudge ( $\langle \perp_F \rangle 90$ ) where  $\perp_F \equiv (F\text{Assert } SBOTTOM')$ 
notation SChaneq' ( $\langle - \leftrightarrow \rightarrow [90, 90] 90$ )
notation substTerm ( $\langle -[-::=-] \rangle [100, 100, 100] 100$ )
notation subs ( $\langle -[-::=-] \rangle [100, 100, 100] 100$ )
notation AssertionStatEq ( $\langle - \simeq \rightarrow [80, 80] 80$ )
notation FrameStatEq ( $\langle - \simeq_F \rightarrow [80, 80] 80$ )
notation SBOTTOM' ( $\langle 1 \rangle 190$ )
abbreviation insertAssertion' ( $\langle insertAssertion \rangle$ ) where insertAssertion'  $\equiv$  assertionAux.insertAssertion ( $\otimes$ )
inductive semantics :: 'b  $\Rightarrow$  ('a, 'b, 'c) psi  $\Rightarrow$  ('a, 'b, 'c) residual  $\Rightarrow$  bool
          ( $\langle - \triangleright - \mapsto \rightarrow [50, 50, 50] 50$ )
where
  cInput:  $\llbracket \Psi \vdash M \leftrightarrow K; distinct xvec; set xvec \subseteq supp N; xvec \#* Tvec;$ 
           $length xvec = length Tvec;$ 
           $xvec \#* \Psi; xvec \#* M; xvec \#* K \rrbracket \implies \Psi \triangleright M(\lambda*xvec N).P \mapsto$ 
 $K(\llbracket N[xvec:=Tvec] \rrbracket) \prec P[xvec:=Tvec]$ 
  | Output:  $\llbracket \Psi \vdash M \leftrightarrow K \rrbracket \implies \Psi \triangleright M\langle N \rangle.P \mapsto K\langle N \rangle \prec P$ 
  | Case:  $\llbracket \Psi \triangleright P \mapsto Rs; (\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; guarded P \rrbracket \implies \Psi \triangleright Cases Cs$ 
           $\mapsto Rs$ 
  | cPar1:  $\llbracket (\Psi \otimes \Psi_Q) \triangleright P \mapsto \alpha \prec P'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$ 
             $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* P'; distinct(bn \alpha);$ 
             $bn \alpha \#* \Psi; bn \alpha \#* \Psi_Q; bn \alpha \#* Q; bn \alpha \#* P; bn \alpha \#* (\text{subject } \alpha) \rrbracket \implies$ 

```

$\Psi \triangleright P \parallel Q \xrightarrow{\alpha} \prec (P' \parallel Q)$
| $cPar2$: $\llbracket (\Psi \otimes \Psi_P) \triangleright Q \xrightarrow{\alpha} \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* Q'; distinct(bn \alpha);$
 $bn \alpha \#* \Psi; bn \alpha \#* \Psi_P; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* (subject \alpha) \rrbracket \implies$
 $\Psi \triangleright P \parallel Q \xrightarrow{\alpha} \prec (P \parallel Q')$
| $cComm1$: $\llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(\nu*xvec)} \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(\nu*xvec)} \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle;$
 $distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* N; A_Q \#* P';$
 $A_Q \#* Q; A_Q \#* K; A_Q \#* Q'; A_Q \#* xvec; distinct xvec;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M;$
 $xvec \#* Q; xvec \#* K \rrbracket \implies$
 $\Psi \triangleright P \parallel Q \xrightarrow{\tau} \prec (\nu*xvec)(P' \parallel Q')$
| $cComm2$: $\llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(\nu*xvec)} \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(\nu*xvec)} \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* N; A_Q \#* P';$
 $A_Q \#* Q; A_Q \#* K; A_Q \#* Q'; A_Q \#* xvec; distinct xvec;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M;$
 $xvec \#* Q; xvec \#* K \rrbracket \implies$
 $\Psi \triangleright P \parallel Q \xrightarrow{\tau} \prec (\nu*xvec)(P' \parallel Q')$
| $cOpen$: $\llbracket \Psi \triangleright P \xrightarrow{M(\nu*(xvec@yvec))} \prec P'; x \in supp N; x \# xvec; x \# yvec; x \# M; x \# \Psi;$
 $distinct xvec; distinct yvec;$
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* yvec; yvec \#* \Psi; yvec \#* P;$
 $yvec \#* M \rrbracket \implies$
 $\Psi \triangleright (\nu x)P \xrightarrow{M(\nu*(xvec@x#yvec))} \prec P'$
| $cScope$: $\llbracket \Psi \triangleright P \xrightarrow{\alpha} \prec P'; x \# \Psi; x \# \alpha; bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* (subject \alpha); distinct(bn \alpha) \rrbracket \implies \Psi \triangleright (\nu x)P \xrightarrow{\alpha} \prec ((\nu x)P')$
| $Bang$: $\llbracket \Psi \triangleright P \parallel !P \xrightarrow{} Rs; guarded P \rrbracket \implies \Psi \triangleright !P \xrightarrow{} Rs$

abbreviation

semanticsBottomJudge ($\leftarrow \mapsto [50, 50] 50$) **where** $P \mapsto Rs \equiv \mathbf{1} \triangleright P \mapsto Rs$

equivariance *env.semantics*

nominal-inductive2 *env.semantics*

avoids *cInput*: set *xvec*

- | $cPar1$: set $A_Q \cup set(bn \alpha)$
- | $cPar2$: set $A_P \cup set(bn \alpha)$

```

| cComm1: set AP ∪ set AQ ∪ set xvec
| cComm2: set AP ∪ set AQ ∪ set xvec
| cOpen: {x} ∪ set xvec ∪ set yvec
| cScope: {x} ∪ set(bn α)
⟨proof⟩

```

```

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 × ('a, 'b, 'c) psi) list
and Ψ' :: 'b

shows Ψ ▷ 0 → Rs ==> False
and Ψ ▷ M(λ*xvec N).P → K(ν*yvec)(N') ⊢ P' ==> False
and Ψ ▷ M(λ*xvec N).P → τ ⊢ P' ==> False
and Ψ ▷ M(N).P → K(N') ⊢ P' ==> False
and Ψ ▷ M(N).P → τ ⊢ P' ==> False
and Ψ ▷ {Ψ'} → Rs ==> False
⟨proof⟩

```

```

lemma residualEq:
fixes α :: 'a action
and P :: ('a, 'b, 'c) psi
and β :: 'a action
and Q :: ('a, 'b, 'c) psi

assumes α ⊢ P = β ⊢ Q
and bn α #* (bn β)
and distinct(bn α)
and distinct(bn β)
and bn α #* (α ⊢ P)
and bn β #* (β ⊢ Q)

```

```

obtains p where set p ⊆ set(bn α) × set(bn(p · α)) and distinctPerm p and
β = p · α and Q = p · P and bn α #* β and bn α #* Q and bn(p · α) #* α and
bn(p · α) #* P
⟨proof⟩

```

```

lemma semanticsInduct[consumes 3, case-names cAlpha cInput cOutput cCase
cPar1 cPar2 cComm1 cComm2 cOpen cScope cBang]:
fixes Ψ :: 'b

```

and $P :: ('a, 'b, 'c) \text{ psi}$
and $\alpha :: 'a \text{ action}$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $\text{Prop} :: 'd::\text{fs-name} \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \text{ psi} \Rightarrow 'a \text{ action} \Rightarrow ('a, 'b, 'c) \text{ psi} \Rightarrow \text{bool}$
and $C :: 'd::\text{fs-name}$

assumes $\Psi \triangleright P \xrightarrow{\alpha} P'$
and $\text{bn } \alpha \#* (\text{subject } \alpha)$
and $\text{distinct}(\text{bn } \alpha)$
and $rAlpha: \bigwedge \Psi P \alpha P' p C. [\text{bn } \alpha \#* \Psi; \text{bn } \alpha \#* P; \text{bn } \alpha \#* (\text{subject } \alpha); \text{bn } \alpha \#* C; \text{bn } \alpha \#* (\text{bn}(p \cdot \alpha)); \text{set } p \subseteq \text{set}(\text{bn } \alpha) \times \text{set}(\text{bn}(p \cdot \alpha)); \text{distinctPerm } p; (\text{bn}(p \cdot \alpha)) \#* \alpha; (\text{bn}(p \cdot \alpha)) \#* P'; \text{Prop } C \Psi P \alpha$
 $P] \implies$
 $\text{Prop } C \Psi P (p \cdot \alpha) (p \cdot P')$

and $rInput: \bigwedge \Psi M K xvec N Tvec P C.$
 $[\Psi \vdash M \leftrightarrow K; \text{distinct } xvec; \text{set } xvec \subseteq \text{supp } N; \text{length } xvec = \text{length } Tvec; xvec \#* \Psi; xvec \#* M; xvec \#* K; xvec \#* C] \implies$
 $\text{Prop } C \Psi (M(\lambda*xvec N).P)$
 $(K(N[xvec:=Tvec])) (P[xvec:=Tvec])$

and $rOutput: \bigwedge \Psi M K N P C. [\Psi \vdash M \leftrightarrow K] \implies \text{Prop } C \Psi (M(N).P)$
 $(K(N)) P$

and $rCase: \bigwedge \Psi P \alpha P' \varphi Cs C. [\Psi \triangleright P \xrightarrow{\alpha} P'; \bigwedge C. \text{Prop } C \Psi P \alpha P'; (\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; \text{guarded } P] \implies$
 $\text{Prop } C \Psi (\text{Cases } Cs) \alpha P'$

and $rPar1: \bigwedge \Psi_Q P \alpha P' A_Q Q C.$
 $[\Psi \otimes \Psi_Q \triangleright P \xrightarrow{\alpha} P'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P \alpha P'; A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* P'; A_Q \#* C;$
 $\text{distinct}(\text{bn } \alpha); \text{bn } \alpha \#* Q;$
 $\text{bn } \alpha \#* \Psi; \text{bn } \alpha \#* \Psi_Q; \text{bn } \alpha \#* P; \text{bn } \alpha \#* \text{subject } \alpha; \text{bn } \alpha \#* C]$
 \implies
 $\text{Prop } C \Psi (P \parallel Q) \alpha (P' \parallel Q)$

and $rPar2: \bigwedge \Psi_P Q \alpha Q' A_P P C.$
 $[\Psi \otimes \Psi_P \triangleright Q \xrightarrow{\alpha} Q'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q \alpha Q'; A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* Q'; A_P \#* C;$
 $\text{distinct}(\text{bn } \alpha); \text{bn } \alpha \#* Q;$
 $\text{bn } \alpha \#* \Psi; \text{bn } \alpha \#* \Psi_P; \text{bn } \alpha \#* P; \text{bn } \alpha \#* \text{subject } \alpha; \text{bn } \alpha \#* C]$
 \implies
 $\text{Prop } C \Psi (P \parallel Q) \alpha (P \parallel Q')$

and $rComm1: \bigwedge \Psi_Q P M N P' A_P \Psi_P Q K xvec Q' A_Q C.$
 $[\Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(N)} P'; \bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P (M(N))$
 $P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$

$\Psi \otimes \Psi_P \triangleright Q \longrightarrow K(\nu*xvec)\langle N \rangle \prec Q'; \bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q$
 $(K(\nu*xvec)\langle N \rangle) Q';$
 $\quad \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\quad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$
 $\quad A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $\quad A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $\quad A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $\quad \text{distinct } xvec;$
 $\quad A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $\quad xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$
 $\quad \text{Prop } C \Psi (P \parallel Q) (\tau) ((\nu*xvec)(P' \parallel Q'))$
and $rComm2: \bigwedge \Psi \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \longrightarrow M(\nu*xvec)\langle N \rangle \prec P'; \bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P$
 $(M(\nu*xvec)\langle N \rangle) P';$
 $\quad \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\quad \Psi \otimes \Psi_P \triangleright Q \longrightarrow K(N) \prec Q'; \bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q (K(N))$
 $Q';$
 $\quad \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\quad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$
 $\quad A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $\quad A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $\quad A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $\quad \text{distinct } xvec;$
 $\quad A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $\quad xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$
 $\quad \text{Prop } C \Psi (P \parallel Q) (\tau) ((\nu*xvec)(P' \parallel Q'))$
and $rOpen: \bigwedge \Psi P M xvec yvec N P' x C.$
 $\llbracket \Psi \triangleright P \longrightarrow M(\nu*(xvec@yvec))\langle N \rangle \prec P'; x \in \text{supp } N; \bigwedge C. \text{Prop } C$
 $\Psi P (M(\nu*(xvec@yvec))\langle N \rangle) P';$
 $\quad x \#* \Psi; x \#* M; x \#* xvec; x \#* yvec; xvec \#* \Psi; xvec \#* P; xvec \#* M;$
 $\quad \text{distinct } xvec; \text{distinct } yvec;$
 $\quad yvec \#* \Psi; yvec \#* P; yvec \#* M; yvec \#* C; x \#* C; xvec \#* C] \implies$
 $\quad \text{Prop } C \Psi ((\nu x)P) (M(\nu*(xvec@x#yvec))\langle N \rangle) P'$
and $rScope: \bigwedge \Psi P \alpha P' x C.$
 $\llbracket \Psi \triangleright P \longrightarrow \alpha \prec P'; \bigwedge C. \text{Prop } C \Psi P \alpha P';$
 $\quad x \#* \Psi; x \#* \alpha; bn \alpha \#* \Psi;$
 $\quad bn \alpha \#* P; bn \alpha \#* (\text{subject } \alpha); x \#* C; bn \alpha \#* C; \text{distinct}(bn \alpha) \rrbracket$
 \implies
 $\quad \text{Prop } C \Psi ((\nu x)P) \alpha ((\nu x)P')$
and $rBang: \bigwedge \Psi P \alpha P' C.$
 $\llbracket \Psi \triangleright P \parallel !P \longrightarrow \alpha \prec P'; \text{guarded } P; \bigwedge C. \text{Prop } C \Psi (P \parallel !P) \alpha$
 $P \rrbracket \implies$
 $\quad \text{Prop } C \Psi (!P) \alpha P'$

shows $\text{Prop } C \Psi P \alpha P'$
 $\langle proof \rangle$

lemma *outputInduct*[consumes 1, case-names *cOutput* *cCase* *cPar1* *cPar2* *cOpen* *cScope* *cBang*]:

```

fixes  $\Psi$  :: 'b
and  $P$  :: ('a, 'b, 'c) psi
and  $M$  :: 'a
and  $B$  :: ('a, 'b, 'c) boundOutput
and  $Prop :: 'd::fs-name \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow$ 
      ' $a \Rightarrow ('a, 'b, 'c)$  boundOutput  $\Rightarrow$  bool
and  $C :: 'd::fs-name$ 

assumes  $\Psi \triangleright P \longmapsto ROut M B$ 
and  $rOutput: \bigwedge \Psi M K N P C. [\Psi \vdash M \leftrightarrow K] \implies Prop C \Psi (M\langle N \rangle.P) K$ 
 $(N \prec' P)$ 
and  $rCase: \bigwedge \Psi P M B \varphi Cs C.$ 
 $[\Psi \triangleright P \longmapsto (ROut M B); \bigwedge C. Prop C \Psi P M B; (\varphi, P) \text{ mem } Cs;$ 
 $\Psi \vdash \varphi; \text{guarded } P] \implies$ 
 $Prop C \Psi (\text{Cases } Cs) M B$ 
and  $rPar1: \bigwedge \Psi_Q P M xvec N P' A_Q Q C.$ 
 $[\Psi \otimes \Psi_Q \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'; \text{extractFrame } Q = \langle A_Q,$ 
 $\Psi_Q \rangle; \text{distinct } A_Q;$ 
 $\bigwedge C. Prop C (\Psi \otimes \Psi_Q) P M ((\nu*xvec)\langle N \rangle \prec' P');$ 
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* M;$ 
 $A_Q \#* xvec; A_Q \#* N; A_Q \#* P'; A_Q \#* C; xvec \#* Q;$ 
 $xvec \#* \Psi; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M; xvec \#* C] \implies$ 
 $Prop C \Psi (P \parallel Q) M ((\nu*xvec)\langle N \rangle \prec' (P' \parallel Q))$ 
and  $rPar2: \bigwedge \Psi_P Q M xvec N Q' A_P P C.$ 
 $[\Psi \otimes \Psi_P \triangleright Q \longmapsto M(\nu*xvec)\langle N \rangle \prec Q'; \text{extractFrame } P = \langle A_P,$ 
 $\Psi_P \rangle; \text{distinct } A_P;$ 
 $\bigwedge C. Prop C (\Psi \otimes \Psi_P) Q M ((\nu*xvec)\langle N \rangle \prec' Q');$ 
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* M;$ 
 $A_P \#* xvec; A_P \#* N; A_P \#* Q'; A_P \#* C; xvec \#* P;$ 
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* Q; xvec \#* M; xvec \#* C] \implies$ 
 $Prop C \Psi (P \parallel Q) M ((\nu*xvec)\langle N \rangle \prec' (P \parallel Q'))$ 
and  $rOpen: \bigwedge \Psi P M xvec yvec N P' x C.$ 
 $[\Psi \triangleright P \longmapsto M(\nu*(xvec@yvec))\langle N \rangle \prec P'; x \in \text{supp } N; \bigwedge C. Prop C$ 
 $\Psi P M ((\nu*(xvec@yvec))\langle N \rangle \prec' P');$ 
 $x \# \Psi; x \# M; x \# xvec; x \# yvec; xvec \#* \Psi; xvec \#* P; xvec \#* M;$ 
 $xvec \#* yvec; yvec \#* \Psi; yvec \#* P; yvec \#* M; yvec \#* C; x \# C;$ 
 $xvec \#* C] \implies$ 
 $Prop C \Psi ((\nu x)P) M ((\nu*(xvec@x#yvec))\langle N \rangle \prec' P')$ 
and  $rScope: \bigwedge \Psi P M xvec N P' x C.$ 
 $[\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'; \bigwedge C. Prop C \Psi P M ((\nu*xvec)\langle N$ 
 $\prec' P'));$ 
 $x \# \Psi; x \# M; x \# xvec; x \# N; xvec \#* \Psi; xvec \#* P; xvec \#* M;$ 
 $x \# C; xvec \#* C] \implies$ 
 $Prop C \Psi ((\nu x)P) M ((\nu*xvec)\langle N \rangle \prec' (\nu x)P')$ 
and  $rBang: \bigwedge \Psi P M B C.$ 
 $[\Psi \triangleright P \parallel !P \longmapsto (ROut M B); \text{guarded } P; \bigwedge C. Prop C \Psi (P \parallel$ 
 $!P) M B] \implies$ 
```

```


$$\text{Prop } C \Psi (!P) M B$$

shows Prop C Ψ P M B
⟨proof⟩

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 ↦ $\alpha$  ↻ P'
  and bn α #* subject α
  and distinct(bn α)
  and y ∈ set(bn α)

  shows y ∈ supp(object α)
⟨proof⟩

lemma alphaBoundOutputChain':
  fixes yvec :: name list
  and xvec :: name list
  and B :: ('a, 'b, 'c) boundOutput

  assumes length xvec = length yvec
  and yvec #* B
  and yvec #* xvec
  and distinct yvec

  shows (λxvec|B = (λyvec|([xvec yvec] ·v B)
⟨proof⟩

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 #* N
  and yvec #* P
  and yvec #* xvec
  and distinct yvec

  shows (λxvec|(N ↻' P) = (λyvec|(([xvec yvec] ·v N) ↻' ([xvec yvec] ·v P))
⟨proof⟩

```

```

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 α ⊲ P = β ⊲ Q
  and distinct(bn α)
  and ⋀x. x ∈ set(bn α) ==> x ∈ supp(object α)
  and bn α #* bn β
  and bn α #* (object β)
  and bn α #* Q

  shows distinct(bn β)
  ⟨proof⟩

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)
  ⟨proof⟩

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 Ψ ▷ M(λ*xvec N).P ↣ Rs

  shows distinct xvec
  ⟨proof⟩

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 $P' :: ('a, 'b, 'c) \text{ psi}$
and $\text{Prop} :: 'd::\text{fs-name} \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \text{ psi} \Rightarrow$
 $'a \Rightarrow \text{name list} \Rightarrow 'a \Rightarrow ('a, 'b, 'c) \text{ psi} \Rightarrow \text{bool}$
and $C :: 'd::\text{fs-name}$

assumes $\Psi \triangleright P \xrightarrow{\longrightarrow M(\nu*xvec)\langle N \rangle} \prec P'$
and $xvec \#* M$
and $rAlpha: \bigwedge \Psi P M xvec N P' p C. [\![xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* C; xvec \#* (p \cdot xvec); set p \subseteq set xvec \times set(p \cdot xvec); distinctPerm p; (p \cdot xvec) \#* N; (p \cdot xvec) \#* P'; Prop C \Psi P M xvec N P]\!] \Rightarrow$
 $Prop C \Psi P M (p \cdot xvec) (p \cdot N) (p \cdot P')$
and $rOutput: \bigwedge \Psi M K N P C. [\![\Psi \vdash M \leftrightarrow K]\!] \Rightarrow Prop C \Psi (M\langle N \rangle.P) K ([] N P$
and $rCase: \bigwedge \Psi P M xvec N P' \varphi Cs C. [\![\Psi \triangleright P \xrightarrow{\longrightarrow M(\nu*xvec)\langle N \rangle} \prec P'; \bigwedge C. Prop C \Psi P M xvec N P'; (\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; \text{guarded } P]\!] \Rightarrow$
 $Prop C \Psi (\text{Cases } Cs) M xvec N P'$
and $rPar1: \bigwedge \Psi \Psi_Q P M xvec N P' A_Q Q C.$
 $[\![\Psi \otimes \Psi_Q \triangleright P \xrightarrow{\longrightarrow M(\nu*xvec)\langle N \rangle} \prec P'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;\bigwedge C. Prop C (\Psi \otimes \Psi_Q) P M xvec N P'; A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* M;$
 $A_Q \#* xvec; A_Q \#* N; A_Q \#* P'; A_Q \#* C; xvec \#* Q; xvec \#* \Psi; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M; xvec \#* C]\!] \Rightarrow$
 $Prop C \Psi (P \parallel Q) M xvec N (P' \parallel Q)$
and $rPar2: \bigwedge \Psi \Psi_P Q M xvec N Q' A_P P C.$
 $[\![\Psi \otimes \Psi_P \triangleright Q \xrightarrow{\longrightarrow M(\nu*xvec)\langle N \rangle} \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;\bigwedge C. Prop C (\Psi \otimes \Psi_P) Q M xvec N Q'; A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* M;$
 $A_P \#* xvec; A_P \#* N; A_P \#* Q'; A_P \#* C; xvec \#* Q; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* C]\!] \Rightarrow$
 $Prop C \Psi (P \parallel Q) M xvec N (P \parallel Q')$
and $rOpen: \bigwedge \Psi P M xvec yvec N P' x C.$
 $[\![\Psi \triangleright P \xrightarrow{\longrightarrow M(\nu*(xvec@yvec))\langle N \rangle} \prec P'; x \in supp N; \bigwedge C. Prop C \Psi P M (xvec@yvec) N P'; x \# \Psi; x \# M; x \# xvec; x \# yvec; xvec \#* \Psi; xvec \#* P; xvec \#* M; yvec \#* \Psi; yvec \#* P; yvec \#* M; yvec \#* C; x \# C; xvec \#* C]\!] \Rightarrow$
 $Prop C \Psi ((\nu x)P) M (xvec@x\#yvec) N P'$
and $rScope: \bigwedge \Psi P M xvec N P' x C.$
 $[\![\Psi \triangleright P \xrightarrow{\longrightarrow M(\nu*xvec)\langle N \rangle} \prec P'; \bigwedge C. Prop C \Psi P M xvec N P'; x \# \Psi; x \# M; x \# xvec; x \# N; xvec \#* \Psi; xvec \#* P; xvec \#* M; x \# C; xvec \#* C]\!] \Rightarrow$
 $Prop C \Psi ((\nu x)P) M xvec N ((\nu x)P')$
and $rBang: \bigwedge \Psi P M xvec N P' C.$
 $[\![\Psi \triangleright P \parallel !P \xrightarrow{\longrightarrow M(\nu*xvec)\langle N \rangle} \prec P'; \text{guarded } P; \bigwedge C. Prop C \Psi (P \parallel !P) M xvec N P]\!] \Rightarrow$
 $Prop C \Psi (!P) M xvec N P'$

shows $\text{Prop } C \Psi P M xvec N P'$
 $\langle \text{proof} \rangle$

lemma $\text{inputInduct}[\text{consumes } 1, \text{ case-names } cInput \ cCase \ cPar1 \ cPar2 \ cScope \ cBang]$:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $M :: 'a$
and $N :: 'a$
and $P' :: ('a, 'b, 'c) \psi$
and $\text{Prop} :: 'd:\text{fs-name} \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a, 'b, 'c) \psi \Rightarrow \text{bool}$
and $C :: 'd:\text{fs-name}$

assumes $\text{Trans}: \Psi \triangleright P \rightarrowtail M(N) \prec P'$

and $rInput: \bigwedge \Psi M K xvec N Tvec P C.$

$\llbracket \Psi \vdash M \leftrightarrow K; \text{distinct } xvec; \text{set } xvec \subseteq \text{supp } N;$
 $\text{length } xvec = \text{length } Tvec; xvec \not\models \Psi;$
 $xvec \not\models M; xvec \not\models K; xvec \not\models C \rrbracket \implies$
 $\text{Prop } C \Psi (M(\lambda*xvec N).P)$

$K(N[xvec::=Tvec]) (P[xvec::=Tvec])$

and $rCase: \bigwedge \Psi P M N P' \varphi Cs C. \llbracket \Psi \triangleright P \rightarrowtail M(N) \prec P'; \bigwedge C. \text{Prop } C \Psi P M N P'; (\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; \text{guarded } P \rrbracket \implies$

$\text{Prop } C \Psi (\text{Cases } Cs) M N P'$

and $rPar1: \bigwedge \Psi \Psi_Q P M N P' A_Q Q C.$

$\llbracket \Psi \otimes \Psi_Q \triangleright P \rightarrowtail M(N) \prec P'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$

$\text{distinct } A_Q;$

$\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P M N P'; \text{distinct } A_Q;$
 $A_Q \not\models P; A_Q \not\models Q; A_Q \not\models \Psi; A_Q \not\models M; A_Q \not\models N;$
 $A_Q \not\models P'; A_Q \not\models C \rrbracket \implies$

$\text{Prop } C \Psi (P \parallel Q) M N (P' \parallel Q)$

and $rPar2: \bigwedge \Psi \Psi_P Q M N Q' A_P P C.$

$\llbracket \Psi \otimes \Psi_P \triangleright Q \rightarrowtail M(N) \prec Q'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$

$\text{distinct } A_P;$

$\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q M N Q'; \text{distinct } A_P;$
 $A_P \not\models P; A_P \not\models Q; A_P \not\models \Psi; A_P \not\models M; A_P \not\models N;$
 $A_P \not\models Q'; A_P \not\models C \rrbracket \implies$

$\text{Prop } C \Psi (P \parallel Q) M N (P \parallel Q')$

and $rScope: \bigwedge \Psi P M N P' x C.$

$\llbracket \Psi \triangleright P \rightarrowtail M(N) \prec P'; \bigwedge C. \text{Prop } C \Psi P M N P'; x \notin \Psi; x \notin M; x \notin N; x \notin C \rrbracket \implies$

$\text{Prop } C \Psi ((\nu x)P) M N ((\nu x)P')$

and $rBang: \bigwedge \Psi P M N P' C.$

$\llbracket \Psi \triangleright P \parallel !P \rightarrowtail M(N) \prec P'; \text{guarded } P; \bigwedge C. \text{Prop } C \Psi (P \parallel !P) M N P' \rrbracket \implies \text{Prop } C \Psi (!P) M N P'$

shows $\text{Prop } C \Psi P M N P'$

$\langle \text{proof} \rangle$

lemma $\text{tauInduct}[\text{consumes } 1, \text{ case-names } cCase \ cPar1 \ cPar2 \ cComm1 \ cComm2]$

cScope cBang]:

```

fixes  $\Psi$  :: ' $b$ 
and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
and  $Rs$  :: (' $a$ , ' $b$ , ' $c$ ) residual
and  $Prop :: 'd::fs-name \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow$ 
      (' $a$ , ' $b$ , ' $c$ )  $\psi \Rightarrow \text{bool}$ 
and  $C :: 'd::fs-name$ 

assumes Trans:  $\Psi \triangleright P \xrightarrow{\tau} \prec P'$ 
and rCase:  $\bigwedge \Psi P P' \varphi Cs C. [\Psi \triangleright P \xrightarrow{\tau} \prec P'; \bigwedge C. Prop C \Psi P P'; (\varphi,$ 
 $P) \text{ mem } Cs; \Psi \vdash \varphi; \text{ guarded } P] \implies$ 
       $Prop C \Psi (\text{Cases } Cs) P'$ 
and rPar1:  $\bigwedge \Psi \Psi_Q P P' A_Q Q C.$ 
       $[\Psi \otimes \Psi_Q \triangleright P \xrightarrow{\tau} \prec P'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$ 
       $\bigwedge C. Prop C (\Psi \otimes \Psi_Q) P P';$ 
       $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi;$ 
       $A_Q \#* P'; A_Q \#* C] \implies$ 
       $Prop C \Psi (P \parallel Q) (P' \parallel Q)$ 
and rPar2:  $\bigwedge \Psi \Psi_P Q Q' A_P P C.$ 
       $[\Psi \otimes \Psi_P \triangleright Q \xrightarrow{\tau} \prec Q'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$ 
       $\bigwedge C. Prop C (\Psi \otimes \Psi_P) Q Q';$ 
       $A_P \#* P; A_P \#* Q; A_P \#* \Psi;$ 
       $A_P \#* Q'; A_P \#* C] \implies$ 
       $Prop C \Psi (P \parallel Q) (P \parallel Q')$ 
and rComm1:  $\bigwedge \Psi \Psi_Q P M N P' A_P \Psi_P Q K xvec Q' A_Q C.$ 
       $[\Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(N)} \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$ 
 $\text{distinct } A_P;$ 
       $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(\nu*xvec)(N)} \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$ 
 $\text{distinct } A_Q;$ 
       $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$ 
       $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$ 
       $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$ 
       $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$ 
       $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$ 
 $M;$ 
       $xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$ 
       $Prop C \Psi (P \parallel Q) ((\nu*xvec)(P' \parallel Q'))$ 
and rComm2:  $\bigwedge \Psi \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q C.$ 
       $[\Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(\nu*xvec)(N)} \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$ 
 $\text{distinct } A_P;$ 
       $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(N)} \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$ 
 $\text{distinct } A_Q;$ 
       $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$ 
       $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$ 
       $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$ 
       $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$ 
       $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$ 

```

M;

$$xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$$

$$\text{Prop } C \Psi (P \parallel Q) ((\nu*xvec)(P' \parallel Q'))$$

and $rScope: \bigwedge \Psi P P' x C.$

$$[\Psi \triangleright P \xrightarrow{\tau} P'; \bigwedge C. \text{Prop } C \Psi P P'; x \notin \Psi; x \notin C] \implies$$

$$\text{Prop } C \Psi ((\nu x)P) ((\nu x)P')$$

and $rBang: \bigwedge \Psi P P' C.$

$$[\Psi \triangleright P \parallel !P \xrightarrow{\tau} P'; \text{guarded } P; \bigwedge C. \text{Prop } C \Psi (P \parallel !P) P'] \implies$$

$$\text{Prop } C \Psi (!P) P'$$

shows $\text{Prop } C \Psi P P'$

(proof)

lemma *semanticsFrameInduct*[consumes 3, case-names *cAlpha* *cInput* *cOutput* *cCase* *cPar1* *cPar2* *cComm1* *cComm2* *cOpen* *cScope* *cBang*]:

fixes $\Psi :: 'b$

and $P :: ('a, 'b, 'c) \psi$

and $Rs :: ('a, 'b, 'c) \text{ residual}$

and $A_P :: \text{name list}$

and $\Psi_P :: 'b$

and $\text{Prop} :: 'd::fs-name \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow ('a, 'b, 'c) \text{ residual} \Rightarrow \text{name list} \Rightarrow 'b \Rightarrow \text{bool}$

and $C :: 'd::fs-name$

assumes *Trans*: $\Psi \triangleright P \xrightarrow{} Rs$

and $\text{FrP}: \text{extractFrame } P = \langle A_P, \Psi_P \rangle$

and $\text{distinct } A_P$

and $rAlpha: \bigwedge \Psi P A_P \Psi_P p Rs C. [A_P \#* \Psi; A_P \#* P; A_P \#* (p \cdot A_P); A_P \#* Rs; A_P \#* C; set p \subseteq \text{set } A_P \times \text{set}(p \cdot A_P); \text{distinctPerm } p; \text{Prop } C \Psi P Rs A_P \Psi_P] \implies \text{Prop } C \Psi P Rs (p \cdot A_P) (p \cdot \Psi_P)$

and $rInput: \bigwedge \Psi M K xvec N Tvec P C.$

$$[\Psi \vdash M \leftrightarrow K; \text{distinct } xvec; \text{set } xvec \subseteq \text{supp } N; \text{length } xvec = \text{length } Tvec; xvec \#* \Psi; xvec \#* M; xvec \#* K; xvec \#* C] \implies$$

$$\text{Prop } C \Psi (M(\lambda*xvec N).P)$$

$$(K((N[xvec:=Tvec])) \prec (P[xvec:=Tvec])) ([])(\mathbf{1})$$

and $rOutput: \bigwedge \Psi M K N P C. \Psi \vdash M \leftrightarrow K \implies \text{Prop } C \Psi (M\langle N \rangle.P) (K\langle N \rangle \prec P) ([])(\mathbf{1})$

and $rCase: \bigwedge \Psi P Rs \varphi Cs A_P \Psi_P C. [\Psi \triangleright P \xrightarrow{} Rs; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P; \bigwedge C. \text{Prop } C \Psi P Rs A_P \Psi_P; (\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; \text{guarded } P; \Psi_P \simeq \mathbf{1}; (\text{supp } \Psi_P) = (\{\}::\text{name set}); A_P \#* \Psi; A_P \#* P; A_P \#* Rs; A_P \#* C] \implies$

$$\text{Prop } C \Psi (\text{Cases } Cs) Rs ([])(\mathbf{1})$$

and $rPar1: \bigwedge \Psi_Q P \alpha P' A_Q Q A_P \Psi_P C.$

$$[\Psi \otimes \Psi_Q \triangleright P \xrightarrow{\alpha} P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$$

$\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P (\alpha \prec P') A_P \Psi_P; \text{distinct}(bn \alpha);$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* P'; A_P \#* A_Q; A_P$
 $\#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* P'; A_Q \#* \Psi_P;$
 $bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* \text{subject } \alpha; bn \alpha \#* \Psi_P;$
 $bn \alpha \#* \Psi_Q;$
 $[A_P \#* C; A_Q \#* C; bn \alpha \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) (\alpha \prec (P' \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rPar2: \bigwedge \Psi \Psi_P Q \alpha Q' A_P P A_Q \Psi_Q C.$
 $[\Psi \otimes \Psi_P \triangleright Q \mapsto \alpha \prec Q';$
 $\text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q (\alpha \prec Q') A_Q \Psi_Q; \text{distinct}(bn \alpha);$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* Q'; A_P \#* A_Q; A_P$
 $\#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* Q'; A_Q \#* \Psi_P;$
 $bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* \text{subject } \alpha; bn \alpha \#* \Psi_P;$
 $bn \alpha \#* \Psi_Q;$
 $[A_P \#* C; A_Q \#* C; bn \alpha \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) (\alpha \prec (P \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rComm1: \bigwedge \Psi \Psi_Q P M N P' A_P \Psi_P Q K xvec Q' A_Q C.$
 $[\Psi \otimes \Psi_Q \triangleright P \mapsto M(N) \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$
 $\text{distinct } A_P;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P ((M(N)) \prec P') A_P \Psi_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K(\nu * xvec)(N) \prec Q'; \text{extractFrame } Q = \langle A_Q,$
 $\Psi_Q \rangle; \text{distinct } A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q (K(\nu * xvec)(N) \prec Q') A_Q \Psi_Q; \text{distinct}$
 $xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $xvec \#* Q; xvec \#* K; A_P \#* C; xvec \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) (\tau \prec (\nu * xvec)(P' \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rComm2: \bigwedge \Psi \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q C.$
 $[\Psi \otimes \Psi_Q \triangleright P \mapsto M(\nu * xvec)(N) \prec P'; \text{extractFrame } P = \langle A_P,$
 $\Psi_P \rangle; \text{distinct } A_P;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P (M(\nu * xvec)(N) \prec P') A_P \Psi_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K(N) \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$
 $\text{distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q (K(N) \prec Q') A_Q \Psi_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; \text{distinct } xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$

$xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$
 $Prop\ C\ \Psi\ (P \parallel Q)\ (\tau \prec (\nu*xvec)(P' \parallel Q'))\ (A_P @ A_Q)\ (\Psi_P \otimes \Psi_Q)$
and $rOpen: \bigwedge \Psi\ P\ M\ xvec\ yvec\ N\ P'\ x\ A_P\ \Psi_P\ C.$
 $\llbracket \Psi \triangleright P \longmapsto M(\nu*(xvec@yvec))\langle N \rangle \prec P'; extractFrame\ P = \langle A_P, \Psi_P \rangle; distinct\ A_P;$
 $\bigwedge C.\ Prop\ C\ \Psi\ P\ (M(\nu*(xvec@yvec))\langle N \rangle \prec P')\ A_P\ \Psi_P; x \in supp\ N; x \# \Psi; x \# M;$
 $x \# A_P; x \# xvec; x \# yvec; A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#*$
 $N; A_P \#* P';$
 $A_P \#* xvec; A_P \#* yvec; xvec \#* yvec; distinct\ xvec; distinct\ yvec;$
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* \Psi_P; yvec \#* \Psi_P;$
 $yvec \#* \Psi; yvec \#* P; yvec \#* M; A_P \#* C; x \# C; xvec \#* C; yvec$
 $\#* C] \implies$
 $Prop\ C\ \Psi\ ((\nu x)P)\ (M(\nu*(xvec@x#yvec))\langle N \rangle \prec P')\ (x\#A_P)\ \Psi_P$
and $rScope: \bigwedge \Psi\ P\ \alpha\ P'\ x\ A_P\ \Psi_P\ C.$
 $\llbracket \Psi \triangleright P \longmapsto \alpha \prec P'; extractFrame\ P = \langle A_P, \Psi_P \rangle; distinct\ A_P;$
 $\bigwedge C.\ Prop\ C\ \Psi\ P\ (\alpha \prec P')\ A_P\ \Psi_P;$
 $x \# \Psi; x \# \alpha; x \# A_P; A_P \#* \Psi; A_P \#* P;$
 $A_P \#* \alpha; A_P \#* P'; distinct(bn\ \alpha);$
 $bn\ \alpha \#* \Psi; bn\ \alpha \#* P; bn\ \alpha \#* subject\ \alpha; bn\ \alpha \#* \Psi_P;$
 $A_P \#* C; x \# C; bn\ \alpha \#* C] \implies$
 $Prop\ C\ \Psi\ ((\nu x)P)\ (\alpha \prec ((\nu x)P'))\ (x\#A_P)\ \Psi_P$
and $rBang: \bigwedge \Psi\ P\ Rs\ A_P\ \Psi_P\ C.$
 $\llbracket \Psi \triangleright P \parallel !P \longmapsto Rs; guarded\ P; extractFrame\ P = \langle A_P, \Psi_P \rangle;$
 $distinct\ A_P;$
 $\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ Rs\ A_P\ (\Psi_P \otimes \mathbf{1}); \Psi_P \simeq \mathbf{1}; supp\ \Psi_P =$
 $(\{\}::name\ set);$
 $A_P \#* \Psi; A_P \#* P; A_P \#* Rs; A_P \#* C] \implies Prop\ C\ \Psi\ (!P)\ Rs$
 $([])\ (\mathbf{1})$
shows $Prop\ C\ \Psi\ P\ Rs\ A_P\ \Psi_P$
 $\langle proof \rangle$

lemma *semanticsFrameInduct*'[consumes 5, case-names cAlpha cFrameAlpha cIn-put cOutput cCase cPar1 cPar2 cComm1 cComm2 cOpen cScope cBang]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Rs :: ('a, 'b, 'c) residual$
and $A_P :: name\ list$
and $\Psi_P :: 'b$
and $Prop :: 'd::fs-name \Rightarrow 'b \Rightarrow ('a, 'b, 'c) psi \Rightarrow 'a\ action \Rightarrow$
 $('a, 'b, 'c) psi \Rightarrow name\ list \Rightarrow 'b \Rightarrow bool$
and $C :: 'd::fs-name$

assumes *Trans*: $\Psi \triangleright P \longmapsto \alpha \prec P'$
and $FrP: extractFrame\ P = \langle A_P, \Psi_P \rangle$
and $distinct\ A_P$
and $bn\ \alpha \#* subject\ \alpha$
and $distinct(bn\ \alpha)$
and $rAlpha: \bigwedge \Psi\ P\ \alpha\ P'\ p\ A_P\ \Psi_P\ C. \llbracket bn\ \alpha \#* \Psi; bn\ \alpha \#* P; bn\ \alpha \#* subject$

$\alpha; bn \alpha \#* \Psi_P;$
 $A_P \#* \alpha; A_P \#* P'; A_P \#* C;$ $bn \alpha \#* C; bn \alpha \#* (p \cdot \alpha); A_P \#* \Psi; A_P \#* P;$
 $p;$ $set p \subseteq set(bn \alpha) \times set(bn(p \cdot \alpha)); distinctPerm$
 $P' A_P \Psi_P] \implies$ $bn(p \cdot \alpha) \#* \alpha; (bn(p \cdot \alpha)) \#* P'; Prop C \Psi P \alpha$
 $Prop C \Psi P (p \cdot \alpha) (p \cdot P') A_P \Psi_P$
and $rFrameAlpha: \bigwedge \Psi P A_P \Psi_P p \alpha P' C. [A_P \#* \Psi; A_P \#* P; A_P \#* (p \cdot$
 $A_P); A_P \#* \alpha; A_P \#* P'; A_P \#* C;$ $set p \subseteq set A_P \times set(p \cdot A_P); distinctPerm$
 $p; A_P \#* subject \alpha;$ $Prop C \Psi P \alpha P' A_P \Psi_P] \implies Prop C \Psi P$
 $\alpha P' (p \cdot A_P) (p \cdot \Psi_P)$
and $rInput: \bigwedge \Psi M K xvec N Tvec P C.$
 $\llbracket \Psi \vdash M \leftrightarrow K; distinct xvec; set xvec \subseteq supp N;$
 $length xvec = length Tvec; xvec \#* \Psi;$
 $xvec \#* M; xvec \#* K; xvec \#* C \rrbracket \implies$
 $Prop C \Psi (M(\lambda*xvec N).P)$
 $(K(N[xvec:=Tvec])) (P[xvec:=Tvec]) ([])(\mathbf{1})$
and $rOutput: \bigwedge \Psi M K N P C. \Psi \vdash M \leftrightarrow K \implies Prop C \Psi (M(N).P) (K(N))$
 $P ([])(\mathbf{1})$
and $rCase: \bigwedge \Psi P \alpha P' \varphi Cs A_P \Psi_P C. [\Psi \triangleright P \mapsto \alpha \prec P'; extractFrame$
 $P = \langle A_P, \Psi_P \rangle; distinct A_P; \bigwedge C. Prop C \Psi P \alpha P' A_P \Psi_P;$
 $(\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; guarded P; \Psi_P \simeq \mathbf{1};$
 $(supp \Psi_P) = (\{\}::name set);$
 $A_P \#* \Psi; A_P \#* P; A_P \#* \alpha; A_P \#* P'; A_P \#*$
 $C \rrbracket \implies Prop C \Psi (Cases Cs) \alpha P' ([])(\mathbf{1})$
and $rPar1: \bigwedge \Psi \Psi_Q P \alpha P' A_Q Q A_P \Psi_P C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \mapsto \alpha \prec P';$
 $extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $\bigwedge C. Prop C (\Psi \otimes \Psi_Q) P \alpha P' A_P \Psi_P;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* P'; A_P \#* A_Q; A_P$
 $\#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* P'; A_Q \#* \Psi_P;$
 $bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* subject \alpha; bn \alpha \#* \Psi_P;$
 $bn \alpha \#* \Psi_Q;$
 $A_P \#* C; A_Q \#* C; bn \alpha \#* C \rrbracket \implies$
 $Prop C \Psi (P \parallel Q) \alpha (P' \parallel Q) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rPar2: \bigwedge \Psi \Psi_P Q \alpha Q' A_P P A_Q \Psi_Q C.$
 $\llbracket \Psi \otimes \Psi_P \triangleright Q \mapsto \alpha \prec Q';$
 $extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $\bigwedge C. Prop C (\Psi \otimes \Psi_P) Q \alpha Q' A_Q \Psi_Q;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* Q'; A_P \#* A_Q; A_P$
 $\#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* Q'; A_Q \#* \Psi_P;$
 $bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* subject \alpha; bn \alpha \#* \Psi_P;$

$bn \alpha \#* \Psi_Q;$
 $A_P \#* C; A_Q \#* C; bn \alpha \#* C] \implies$
 $Prop C \Psi (P \parallel Q) \alpha (P \parallel Q') (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rComm1: \bigwedge \Psi \Psi_Q P M N P' A_P \Psi_P Q K xvec Q' A_Q C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \longmapsto M(N) \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle;$
distinct $A_P;$
 $\bigwedge C. Prop C (\Psi \otimes \Psi_Q) P (M(N)) P' A_P \Psi_P;$
 $\Psi \otimes \Psi_P \triangleright Q \longmapsto K(\nu*xvec)\langle N \rangle \prec Q'; extractFrame Q = \langle A_Q,$
 $\Psi_Q \rangle; distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; distinct xvec;$
 $\bigwedge C. Prop C (\Psi \otimes \Psi_P) Q (K(\nu*xvec)\langle N \rangle) Q' A_Q \Psi_Q;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$
 $Prop C \Psi (P \parallel Q) (\tau) ((\nu*xvec)(P' \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rComm2: \bigwedge \Psi \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'; extractFrame P = \langle A_P,$
 $\Psi_P \rangle; distinct A_P;$
 $\bigwedge C. Prop C (\Psi \otimes \Psi_Q) P (M(\nu*xvec)\langle N \rangle) P' A_P \Psi_P;$
 $\Psi \otimes \Psi_P \triangleright Q \longmapsto K(N) \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle;$
distinct $A_Q;$
 $\bigwedge C. Prop C (\Psi \otimes \Psi_P) Q (K(N)) Q' A_Q \Psi_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; distinct xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $xvec \#* Q; xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C] \implies$
 $Prop C \Psi (P \parallel Q) (\tau) ((\nu*xvec)(P' \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rOpen: \bigwedge \Psi P M xvec yvec N P' x A_P \Psi_P y C.$
 $\llbracket \Psi \triangleright P \longmapsto M(\nu*(xvec@yvec))\langle N \rangle \prec P'; extractFrame P = \langle A_P,$
 $\Psi_P \rangle; distinct A_P;$
 $\bigwedge C. Prop C \Psi P (M(\nu*(xvec@yvec))\langle N \rangle) P' A_P \Psi_P; x \in supp$
 $N; x \# \Psi; x \# M;$
 $x \# A_P; x \# xvec; x \# yvec; A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#*$
 $N; A_P \#* P';$
 $A_P \#* xvec; A_P \#* yvec; xvec \#* yvec; distinct xvec; distinct yvec;$
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* \Psi_P;$
 $yvec \#* \Psi; yvec \#* P; yvec \#* M; A_P \#* C; x \# C; xvec \#* C; yvec$
 $\#* C;$
 $y \neq x; y \# \Psi; y \# P; y \# M; y \# xvec; y \# yvec; y \# N; y \# P'; y \#$
 $A_P; y \# \Psi_P; y \# C] \implies$
 $Prop C \Psi ((\nu x)P) (M(\nu*(xvec@y#yvec))\langle ((x, y) \cdot N) \rangle) ((x,$
 $y) \cdot P') (x \# A_P) \Psi_P$
and $rScope: \bigwedge \Psi P \alpha P' x A_P \Psi_P C.$

$\llbracket \Psi \triangleright P \xrightarrow{\alpha} P' ; extractFrame P = \langle A_P, \Psi_P \rangle ; distinct A_P ;$
 $\wedge C. Prop C \Psi P \alpha P' A_P \Psi_P ;$
 $x \notin \Psi; x \notin \alpha; x \notin A_P; A_P \#* \Psi; A_P \#* P;$
 $A_P \#* \alpha; A_P \#* P' ;$
 $bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* subject \alpha; bn \alpha \#* \Psi_P;$
 $A_P \#* C; x \notin C; bn \alpha \#* C \rrbracket \implies$
 $Prop C \Psi ((\nu x)P) \alpha ((\nu x)P') (x \# A_P) \Psi_P$
and $rBang: \wedge \Psi P \alpha P' A_P \Psi_P C.$
 $\llbracket \Psi \triangleright P \parallel !P \xrightarrow{\alpha} P' ; guarded P; extractFrame P = \langle A_P, \Psi_P \rangle ;$
 $distinct A_P ;$
 $\wedge C. Prop C \Psi (P \parallel !P) \alpha P' A_P (\Psi_P \otimes \mathbf{1}); \Psi_P \simeq \mathbf{1}; supp \Psi_P$
 $= (\{\}::name set);$
 $A_P \#* \Psi; A_P \#* P; A_P \#* \alpha; A_P \#* P'; A_P \#* C \rrbracket \implies Prop C \Psi$
 $(!P) \alpha P' (\mathbb{I}) (\mathbf{1})$
shows $Prop C \Psi P \alpha P' A_P \Psi_P$
 $\langle proof \rangle$

lemma $inputFrameInduct[consumes \ 3, \ case-names \ cAlpha \ cInput \ cCase \ cPar1$
 $cPar2 \ cScope \ cBang]:$
fixes $\Psi :: 'b$
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 $Trans: \Psi \triangleright P \xrightarrow{} M(N) \prec P'$
and $FrP: extractFrame P = \langle A_P, \Psi_P \rangle$
and $distinct A_P$
and $rAlpha: \wedge \Psi P M N P' A_P \Psi_P p \ C. \llbracket A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P$
 $\#* N; A_P \#* P'; A_P \#* (p \cdot A_P); A_P \#* C;$
 $set \ p \subseteq set A_P \times set(p \cdot A_P); distinctPerm p;$
 $Prop C \Psi P M N P' A_P \Psi_P \rrbracket \implies Prop C \Psi$
 $P M N P' (p \cdot A_P) (p \cdot \Psi_P)$
and $rInput: \wedge \Psi M K xvec N Tvec P \ C.$
 $\llbracket \Psi \vdash M \leftrightarrow K; distinct xvec; set xvec \subseteq supp N;$
 $length xvec = length Tvec; xvec \#* \Psi;$
 $xvec \#* M; xvec \#* K; xvec \#* C \rrbracket \implies$
 $Prop C \Psi (M(\lambda*xvec N).P)$
 $K (N[xvec:=Tvec]) (P[xvec:=Tvec]) (\mathbb{I}) (\mathbf{1})$
and $rCase: \wedge \Psi P M N P' \varphi \ Cs \ A_P \ \Psi_P \ C. \llbracket \Psi \triangleright P \xrightarrow{} M(N) \prec P';$
 $extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P; \wedge C. Prop C \Psi P M N P' A_P \Psi_P;$
 $(\varphi, P) \ mem \ Cs; \Psi \vdash \varphi; guarded P; \Psi_P \simeq \mathbf{1};$
 $(supp \Psi_P) = (\{\}::name set);$
 $A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* N; A_P$
 $\#* P'; A_P \#* C \rrbracket \implies Prop C \Psi (Cases \ Cs) M N P' (\mathbb{I}) (\mathbf{1})$
and $rPar1: \wedge \Psi_Q P M N P' A_Q Q A_P \Psi_P C.$

$\llbracket \Psi \otimes \Psi_Q \triangleright P \longmapsto M(N) \prec P';$
 $\text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{ distinct } A_P;$
 $\text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{ distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P M N P' A_P \Psi_P;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#*$
 $A_Q; A_P \#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* M; A_Q \#* N; A_Q \#* P'; A_Q$
 $\#* \Psi_P;$
 $A_P \#* C; A_Q \#* C \rrbracket \implies$
 $\text{Prop } C \Psi (P \parallel Q) M N (P' \parallel Q) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rPar2: \bigwedge \Psi \Psi_P Q M N Q' A_P P A_Q \Psi_Q C.$
 $\llbracket \Psi \otimes \Psi_P \triangleright Q \longmapsto M(N) \prec Q';$
 $\text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{ distinct } A_P;$
 $\text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{ distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q M N Q' A_Q \Psi_Q;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* M; A_P \#* N; A_P \#* Q'; A_P$
 $\#* A_Q; A_P \#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* M; A_Q \#* N; A_Q \#* Q'; A_Q$
 $\#* \Psi_P;$
 $A_P \#* C; A_Q \#* C \rrbracket \implies$
 $\text{Prop } C \Psi (P \parallel Q) M N (P \parallel Q') (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rScope: \bigwedge \Psi P M N P' x A_P \Psi_P C.$
 $\llbracket \Psi \triangleright P \longmapsto M(N) \prec P'; \text{ extractFrame } P = \langle A_P, \Psi_P \rangle; \text{ distinct } A_P;$
 $\bigwedge C. \text{Prop } C \Psi P M N P' A_P \Psi_P; x \# \Psi; x \# M; x \# N;$
 $x \# A_P; A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* C; x \# C \rrbracket \implies$
 $\text{Prop } C \Psi ((\nu x) P) M N ((\nu x) P) (x \# A_P) \Psi_P$
and $rBang: \bigwedge \Psi P M N P' A_P \Psi_P C.$
 $\llbracket \Psi \triangleright P \parallel !P \longmapsto M(N) \prec P'; \text{ guarded } P; \text{ extractFrame } P = \langle A_P, \Psi_P \rangle; \text{ distinct } A_P;$
 $\bigwedge C. \text{Prop } C \Psi (P \parallel !P) M N P' A_P (\Psi_P \otimes \mathbf{1}); \Psi_P \simeq \mathbf{1}; (\text{supp } \Psi_P) = (\{\} :: \text{name set});$
 $A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* C \rrbracket \implies$
 $\text{Prop } C \Psi (!P) M N P' (\mathbb{0}) (\mathbf{1})$
shows $\text{Prop } C \Psi P M N P' A_P \Psi_P$
 $\langle proof \rangle$

lemma $\text{outputFrameInduct}[\text{consumes } 3, \text{ case-names } cAlpha \text{ } cOutput \text{ } cCase \text{ } cPar1$
 $cPar2 \text{ } cOpen \text{ } cScope \text{ } cBang]:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $M :: 'a$
and $B :: ('a, 'b, 'c) \text{ boundOutput}$
and $A_P :: \text{name list}$
and $\Psi_P :: 'b$
and $\text{Prop} :: 'd :: \text{fs-name} \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \text{ psi} \Rightarrow$
 $'a \Rightarrow ('a, 'b, 'c) \text{ boundOutput} \Rightarrow \text{name list} \Rightarrow 'b \Rightarrow \text{bool}$
and $C :: 'd :: \text{fs-name}$

assumes *Trans*: $\Psi \triangleright P \longrightarrow ROut M B$
and FrP : $extractFrame P = \langle A_P, \Psi_P \rangle$
and $distinct A_P$
and $rAlpha$: $\bigwedge \Psi P M A_P \Psi_P p B C. \llbracket A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* (p \cdot A_P); A_P \#* B; A_P \#* C; set p \subseteq set A_P \times set(p \cdot A_P); distinctPerm p; Prop C \Psi P M B A_P \Psi_P \rrbracket \implies Prop C \Psi P M B$
 $(p \cdot A_P) (p \cdot \Psi_P)$
and $rOutput$: $\bigwedge \Psi M K N P C. \Psi \vdash M \leftrightarrow K \implies Prop C \Psi (M\langle N \rangle.P) K (N \prec' P) (\square) (1)$
and $rCase$: $\bigwedge \Psi P M B \varphi Cs A_P \Psi_P C. \llbracket \Psi \triangleright P \longrightarrow (ROut M B); extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P; \bigwedge C. Prop C \Psi P M B A_P \Psi_P; (\varphi, P) mem Cs; \Psi \vdash \varphi; guarded P; \Psi_P \simeq 1; (supp \Psi_P) = (\{\}::name set); A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* B; A_P \#* C \rrbracket \implies Prop C \Psi (Cases Cs) M B (\square) (1)$
and $rPar1$: $\bigwedge \Psi_Q P M xvec N P' A_Q Q A_P \Psi_P C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \longrightarrow M(\nu*xvec)\langle N \rangle \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q; \bigwedge C. Prop C (\Psi \otimes \Psi_Q) P M ((\nu*xvec)N \prec' P') A_P \Psi_P; A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* M; A_P \#* xvec; A_P \#* N; A_P \#* P'; A_P \#* A_Q; A_P \#* \Psi_Q; A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* M; A_Q \#* xvec; A_Q \#* N; A_Q \#* P'; A_Q \#* \Psi_P; xvec \#* \Psi; xvec \#* P; xvec \#* Q; xvec \#* M; xvec \#* \Psi_P; xvec \#* \Psi_Q; A_P \#* C; A_Q \#* C; xvec \#* C \rrbracket \implies Prop C \Psi (P \parallel Q) M ((\nu*xvec)N \prec' (P' \parallel Q)) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rPar2$: $\bigwedge \Psi \Psi_P Q M xvec N Q' A_P P A_Q \Psi_Q C.$
 $\llbracket \Psi \otimes \Psi_P \triangleright Q \longrightarrow M(\nu*xvec)\langle N \rangle \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q; \bigwedge C. Prop C (\Psi \otimes \Psi_P) Q M ((\nu*xvec)N \prec' Q') A_Q \Psi_Q; A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* M; A_P \#* xvec; A_P \#* N; A_P \#* Q'; A_P \#* A_Q; A_P \#* \Psi_Q; A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* M; A_Q \#* xvec; A_Q \#* N; A_Q \#* Q'; A_Q \#* \Psi_P; xvec \#* \Psi; xvec \#* P; xvec \#* Q; xvec \#* M; xvec \#* \Psi_P; xvec \#* \Psi_Q; A_P \#* C; A_Q \#* C; xvec \#* C \rrbracket \implies Prop C \Psi (P \parallel Q) M ((\nu*xvec)N \prec' (P \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rOpen$: $\bigwedge \Psi P M xvec yvec N P' x A_P \Psi_P C.$
 $\llbracket \Psi \triangleright P \longrightarrow M(\nu*(xvec @ yvec))\langle N \rangle \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P; \bigwedge C. Prop C \Psi P M ((\nu*(xvec @ yvec))N \prec' P') A_P \Psi_P; x \in supp N; x \# \Psi; x \# M; x \# A_P; x \# xvec; x \# yvec; A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P' \rrbracket$

$A_P \#* xvec; A_P \#* yvec;$
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* \Psi_P;$
 $yvec \#* \Psi; yvec \#* P; yvec \#* M; A_P \#* C; x \#* C; xvec \#* C; yvec$
 $\#* C] \implies$
 $Prop\ C\ \Psi\ ((\nu x)P)\ M\ ((\nu*(xvec@x\#yvec))N \prec' P')\ (x\#A_P)\ \Psi_P$
and $rScope: \bigwedge \Psi\ P\ M\ xvec\ N\ P'\ x\ A_P\ \Psi_P\ C.$
 $\llbracket \Psi \triangleright P \mapsto M(\nu*xvec)\langle N \rangle \prec P'; extractFrame\ P = \langle A_P, \Psi_P \rangle;$
distinct A_P ;
 $\bigwedge C.\ Prop\ C\ \Psi\ P\ M\ ((\nu*xvec)N \prec' P')\ A_P\ \Psi_P;$
 $x \#* \Psi; x \#* M; x \#* xvec; x \#* N; x \#* A_P; A_P \#* \Psi; A_P \#* P;$
 $A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* xvec;$
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* \Psi_P;$
 $A_P \#* C; x \#* C; xvec \#* C] \implies$
 $Prop\ C\ \Psi\ ((\nu x)P)\ M\ ((\nu*xvec)N \prec' ((\nu x)P'))\ (x\#A_P)\ \Psi_P$
and $rBang: \bigwedge \Psi\ P\ M\ B\ A_P\ \Psi_P\ C.$
 $\llbracket \Psi \triangleright P \parallel !P \mapsto ROut\ M\ B; guarded\ P; extractFrame\ P = \langle A_P, \Psi_P \rangle; distinct\ A_P;$
 $\bigwedge C.\ Prop\ C\ \Psi\ (P \parallel !P)\ M\ B\ A_P\ (\Psi_P \otimes \mathbf{1}); \Psi_P \simeq \mathbf{1}; supp\ \Psi_P$
 $= (\{\}::name\ set);$
 $A_P \#* \Psi; A_P \#* P; A_P \#* M; A_P \#* C] \implies Prop\ C\ \Psi\ (!P)\ M$
 $B\ (\parallel)\ (\mathbf{1})$
shows $Prop\ C\ \Psi\ P\ M\ B\ A_P\ \Psi_P$
 $\langle proof \rangle$

lemma $tauFrameInduct$ [consumes 3, case-names $cAlpha$ $cCase$ $cPar1$ $cPar2$ $cComm1$ $cComm2$ $cScope$ $cBang$]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $P' :: ('a, 'b, 'c) psi$
and $Prop :: 'd::fs-name \Rightarrow 'b \Rightarrow ('a, 'b, 'c) psi \Rightarrow ('a, 'b, 'c) psi \Rightarrow name\ list \Rightarrow 'b \Rightarrow bool$
and $C :: 'd::fs-name$

assumes $Trans: \Psi \triangleright P \mapsto \tau \prec P'$
and $FrP: extractFrame\ P = \langle A_P, \Psi_P \rangle$
and $distinct\ A_P$
and $rAlpha: \bigwedge \Psi\ P\ P'\ A_P\ \Psi_P\ p\ C. \llbracket A_P \#* \Psi; A_P \#* P; A_P \#* P'; A_P \#* (p \cdot A_P); A_P \#* C;$
 $set\ p \subseteq set\ A_P \times set\ (p \cdot A_P); distinctPerm\ p;$
 $Prop\ C\ \Psi\ P\ P'\ A_P\ \Psi_P] \implies Prop\ C\ \Psi\ P\ P'\ (p \cdot A_P)\ (p \cdot \Psi_P)$
and $rCase: \bigwedge \Psi\ P\ P'\ \varphi\ Cs\ A_P\ \Psi_P\ C. \llbracket \Psi \triangleright P \mapsto \tau \prec P'; extractFrame\ P = \langle A_P, \Psi_P \rangle; distinct\ A_P; \bigwedge C.\ Prop\ C\ \Psi\ P\ P'\ A_P\ \Psi_P;$
 $(\varphi, P) \ mem\ Cs; \Psi \vdash \varphi; guarded\ P; \Psi_P \simeq \mathbf{1};$
 $(supp\ \Psi_P) = (\{\}::name\ set);$
 $A_P \#* \Psi; A_P \#* P; A_P \#* P'; A_P \#* C] \implies$
 $Prop\ C\ \Psi\ (Cases\ Cs)\ P' (\parallel)\ (\mathbf{1})$
and $rPar1: \bigwedge \Psi_Q\ P\ P'\ A_Q\ Q\ A_P\ \Psi_P\ C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \mapsto \tau \prec P';$

$\text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_Q) P P' A_P \Psi_P;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* P'; A_P \#* A_Q; A_P \#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* P'; A_Q \#* \Psi_P;$
 $A_P \#* C; A_Q \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) (P' \parallel Q) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rPar2: \bigwedge \Psi \Psi_P Q Q' A_P P A_Q \Psi_Q C.$
 $\llbracket \Psi \otimes \Psi_P \triangleright Q \xrightarrow{\tau} Q' ;$
 $\text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\bigwedge C. \text{Prop } C (\Psi \otimes \Psi_P) Q Q' A_Q \Psi_Q;$
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* Q'; A_P \#* A_Q; A_P \#* \Psi_Q;$
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* Q'; A_Q \#* \Psi_P;$
 $A_P \#* C; A_Q \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) (P \parallel Q') (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rComm1: \bigwedge \Psi \Psi_Q P M N P' A_P \Psi_P Q K xvec Q' A_Q C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(N)} \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$
 $\text{distinct } A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(\nu * xvec)(N)} \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; \text{distinct } xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $xvec \#* Q; xvec \#* K; A_P \#* C; xvec \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) ((\nu * xvec)(P' \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rComm2: \bigwedge \Psi \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q C.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(\nu * xvec)(N)} \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$
 $\text{distinct } A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(N)} \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$
 $\text{distinct } A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; \text{distinct } xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P';$
 $A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* K; A_Q \#* Q';$
 $A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#*$
 $M;$
 $xvec \#* Q; xvec \#* K; A_P \#* C; xvec \#* C] \implies$
 $\text{Prop } C \Psi (P \parallel Q) ((\nu * xvec)(P' \parallel Q')) (A_P @ A_Q) (\Psi_P \otimes \Psi_Q)$
and $rScope: \bigwedge \Psi P P' x A_P \Psi_P C.$
 $\llbracket \Psi \triangleright P \xrightarrow{\tau} \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\bigwedge C. \text{Prop } C \Psi P P' A_P \Psi_P; x \# \Psi;$
 $x \# A_P; A_P \#* \Psi; A_P \#* P; A_P \#* P';$
 $A_P \#* C; x \# C] \implies$
 $\text{Prop } C \Psi ((\nu x)P) ((\nu x)P') (x \# A_P) \Psi_P$
and $rBang: \bigwedge \Psi P P' A_P \Psi_P C.$

$\llbracket \Psi \triangleright P \parallel !P \xrightarrow{\tau} P'; \text{guarded } P; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P; \wedge C. \text{Prop } C \Psi (P \parallel !P) P' A_P (\Psi_P \otimes \mathbf{1}); \Psi_P \simeq \mathbf{1}; \text{supp } \Psi_P = (\{\} :: \text{name set}); A_P \#* \Psi; A_P \#* P; A_P \#* P'; A_P \#* C \rrbracket \implies \text{Prop } C \Psi (!P) P'$
 $([])(\mathbf{1})$
shows $\text{Prop } C \Psi P P' A_P \Psi_P$
 $\langle \text{proof} \rangle$

lemma *inputFreshDerivative*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $M :: 'a$
and $N :: 'a$
and $P' :: ('a, 'b, 'c) \psi$
and $x :: \text{name}$

assumes $\Psi \triangleright P \xrightarrow{M(N)} \prec P'$
and $x \notin P$
and $x \notin N$

shows $x \notin P'$
 $\langle \text{proof} \rangle$

lemma *inputFreshChainDerivative*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $M :: 'a$
and $N :: 'a$
and $P' :: ('a, 'b, 'c) \psi$
and $xvec :: \text{name list}$

assumes $\Psi \triangleright P \xrightarrow{M(N)} \prec P'$
and $xvec \#* P$
and $xvec \#* N$

shows $xvec \#* P'$
 $\langle \text{proof} \rangle$

lemma *outputFreshDerivative*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $M :: 'a$
and $xvec :: \text{name list}$
and $N :: 'a$
and $P' :: ('a, 'b, 'c) \psi$
and $x :: \text{name}$

assumes $\Psi \triangleright P \xrightarrow{M(\nu*xvec)} \langle N \rangle \prec P'$

```

and       $xvec \#* M$ 
and       $distinct\ xvec$ 
and       $x \# P$ 
and       $x \# xvec$ 

shows  $x \# N$ 
and       $x \# P'$ 
(proof)

lemma outputFreshChainDerivative:
  fixes  $\Psi :: 'b$ 
  and       $P :: ('a, 'b, 'c) \psi$ 
  and       $M :: 'a$ 
  and       $xvec :: name\ list$ 
  and       $N :: 'a$ 
  and       $P' :: ('a, 'b, 'c) \psi$ 
  and       $yvec :: name\ list$ 

  assumes  $\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
  and       $xvec \#* M$ 
  and       $distinct\ xvec$ 
  and       $yvec \#* P$ 
  and       $yvec \#* xvec$ 

  shows  $yvec \#* N$ 
  and       $yvec \#* P'$ 
(proof)

lemma tauFreshDerivative:
  fixes  $\Psi :: 'b$ 
  and       $P :: ('a, 'b, 'c) \psi$ 
  and       $P' :: ('a, 'b, 'c) \psi$ 
  and       $x :: name$ 

  assumes  $\Psi \triangleright P \longmapsto \tau \prec P'$ 
  and       $x \# P$ 

  shows  $x \# P'$ 
(proof)

lemma tauFreshChainDerivative:
  fixes  $\Psi :: 'b$ 
  and       $P :: ('a, 'b, 'c) \psi$ 
  and       $M :: 'a$ 
  and       $N :: 'a$ 
  and       $P' :: ('a, 'b, 'c) \psi$ 
  and       $xvec :: name\ list$ 

  assumes  $\Psi \triangleright P \longmapsto \tau \prec P'$ 

```

```

and       $xvec \#* P$ 

shows  $xvec \#* P'$ 
⟨proof⟩

lemma freeFreshDerivative:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $\alpha :: 'a action$ 
and  $P' :: ('a, 'b, 'c) psi$ 
and  $x :: name$ 

assumes  $\Psi \triangleright P \longmapsto \alpha \prec P'$ 
and  $bn \alpha \#* subject \alpha$ 
and  $distinct(bn \alpha)$ 
and  $x \# \alpha$ 
and  $x \# P$ 

shows  $x \# P'$ 
⟨proof⟩

lemma freeFreshChainDerivative:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $\alpha :: 'a action$ 
and  $P' :: ('a, 'b, 'c) psi$ 
and  $xvec :: name list$ 

assumes  $\Psi \triangleright P \longmapsto \alpha \prec P'$ 
and  $bn \alpha \#* subject \alpha$ 
and  $distinct(bn \alpha)$ 
and  $xvec \#* P$ 
and  $xvec \#* \alpha$ 

shows  $xvec \#* P'$ 
⟨proof⟩

lemma Input:
fixes  $\Psi :: 'b$ 
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 $\Psi \triangleright M(\lambda*xvec\ N).P \longmapsto K(N[xvec:=Tvec]) \prec P[xvec:=Tvec]$
 $\langle proof \rangle$

lemma *residualAlpha*:

fixes $p :: name\ prm$
and $\alpha :: 'a\ action$
and $P :: ('a, 'b, 'c)\ psi$

assumes $bn(p \cdot \alpha) \#* object\ \alpha$
and $bn(p \cdot \alpha) \#* P$
and $bn\ \alpha \#* subject\ \alpha$
and $bn(p \cdot \alpha) \#* subject\ \alpha$
and $set\ p \subseteq set(bn\ \alpha) \times set(bn(p \cdot \alpha))$

shows $\alpha \prec P = (p \cdot \alpha) \prec (p \cdot P)$
 $\langle proof \rangle$

lemma *Par1*:

fixes $\Psi :: 'b$
and $\Psi_Q :: 'b$
and $P :: ('a, 'b, 'c)\ psi$
and $\alpha :: 'a\ action$
and $P' :: ('a, 'b, 'c)\ psi$
and $A_Q :: name\ list$
and $Q :: ('a, 'b, 'c)\ psi$

assumes $Trans: \Psi \otimes \Psi_Q \triangleright P \longmapsto \alpha \prec P'$
and $extractFrame\ Q = \langle A_Q, \Psi_Q \rangle$
and $bn\ \alpha \#* Q$
and $A_Q \#* \Psi$
and $A_Q \#* P$
and $A_Q \#* \alpha$

shows $\Psi \triangleright P \parallel Q \longmapsto \alpha \prec (P' \parallel Q)$
 $\langle proof \rangle$

lemma *Par2*:

fixes $\Psi :: 'b$
and $\Psi_P :: 'b$
and $Q :: ('a, 'b, 'c)\ psi$
and $\alpha :: 'a\ action$
and $Q' :: ('a, 'b, 'c)\ psi$
and $A_P :: name\ list$
and $P :: ('a, 'b, 'c)\ psi$

assumes $Trans: \Psi \otimes \Psi_P \triangleright Q \longmapsto \alpha \prec Q'$
and $extractFrame\ P = \langle A_P, \Psi_P \rangle$
and $bn\ \alpha \#* P$

and $A_P \#* \Psi$
and $A_P \#* Q$
and $A_P \#* \alpha$

shows $\Psi \triangleright P \parallel Q \xrightarrow{\alpha} \prec (P \parallel Q)$
 $\langle proof \rangle$

lemma *Open*:

fixes $\Psi :: 'b$
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*: $\Psi \triangleright P \xrightarrow{M(\nu*(xvec @ yvec))} \langle N \rangle \prec P'$
and $x \in supp\ N$
and $x \notin \Psi$
and $x \notin M$
and $x \notin xvec$
and $x \notin yvec$

shows $\Psi \triangleright (\nu x)P \xrightarrow{M(\nu*(xvec @ x \# yvec))} \langle N \rangle \prec P'$
 $\langle proof \rangle$

lemma *Scope*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a\ action$
and $P' :: ('a, 'b, 'c) \psi$
and $x :: name$

assumes $\Psi \triangleright P \xrightarrow{\alpha} \prec P'$
and $x \notin \Psi$
and $x \notin \alpha$

shows $\Psi \triangleright (\nu x)P \xrightarrow{\alpha} \prec (\nu x)P'$
 $\langle proof \rangle$

lemma *inputSwapFrameSubject*:

fixes $\Psi :: 'b$
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 $\Psi \triangleright P \longmapsto M(N) \prec P'$
and $x \notin P$
and $y \notin P$

shows $((x, y)] \cdot \Psi) \triangleright P \longmapsto ((x, y)] \cdot M)(N) \prec P'$
 $\langle proof \rangle$

lemma *inputPermFrameSubject*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $M :: 'a$
and $N :: 'a$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $p :: \text{name} \text{ prm}$
and $Xs :: \text{name set}$
and $Ys :: \text{name set}$

assumes $\Psi \triangleright P \longmapsto M(N) \prec P'$
and $S: \text{set } p \subseteq Xs \times Ys$
and $Xs \nsubseteq P$
and $Ys \nsubseteq P$

shows $(p \cdot \Psi) \triangleright P \longmapsto (p \cdot M)(N) \prec P'$
 $\langle proof \rangle$

lemma *inputSwapSubject*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $M :: 'a$
and $N :: 'a$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $x :: \text{name}$
and $y :: \text{name}$

assumes $\Psi \triangleright P \longmapsto M(N) \prec P'$
and $x \notin P$
and $y \notin P$
and $x \notin \Psi$
and $y \notin \Psi$

shows $\Psi \triangleright P \longmapsto ((x, y)] \cdot M)(N) \prec P'$
 $\langle proof \rangle$

lemma *inputPermSubject*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $M :: 'a$
and $N :: 'a$

```

and    $P' :: ('a, 'b, 'c) \psi$ 
and    $p :: \text{name prm}$ 
and    $Xs :: \text{name set}$ 
and    $Ys :: \text{name set}$ 

assumes  $\Psi \triangleright P \longmapsto M(N) \prec P'$ 
and    $S: \text{set } p \subseteq Xs \times Ys$ 
and    $Xs \#* P$ 
and    $Ys \#* P$ 
and    $Xs \#* \Psi$ 
and    $Ys \#* \Psi$ 

```

shows $\Psi \triangleright P \longmapsto (p \cdot M)(N) \prec P'$
 $\langle \text{proof} \rangle$

lemma *inputSwapFrame*:

```

fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $M :: 'a$ 
and    $N :: 'a$ 
and    $P' :: ('a, 'b, 'c) \psi$ 
and    $x :: \text{name}$ 
and    $y :: \text{name}$ 

```

```

assumes  $\Psi \triangleright P \longmapsto M(N) \prec P'$ 
and    $x \# P$ 
and    $y \# P$ 
and    $x \# M$ 
and    $y \# M$ 

```

shows $([(x, y)] \cdot \Psi) \triangleright P \longmapsto M(N) \prec P'$
 $\langle \text{proof} \rangle$

lemma *inputPermFrame*:

```

fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $M :: 'a$ 
and    $N :: 'a$ 
and    $P' :: ('a, 'b, 'c) \psi$ 
and    $p :: \text{name prm}$ 
and    $Xs :: \text{name set}$ 
and    $Ys :: \text{name set}$ 

```

```

assumes  $\Psi \triangleright P \longmapsto M(N) \prec P'$ 
and    $S: \text{set } p \subseteq Xs \times Ys$ 
and    $Xs \#* P$ 
and    $Ys \#* P$ 
and    $Xs \#* M$ 
and    $Ys \#* M$ 

```

shows $(p \cdot \Psi) \triangleright P \longmapsto M(N) \prec P'$
 $\langle proof \rangle$

```
lemma inputAlpha:
  fixes  $\Psi$  :: ' $b$ 
  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  $\Psi \triangleright P \longmapsto M(N) \prec P'$ 
  and  $set p \subseteq (set xvec) \times (set (p \cdot xvec))$ 
  and  $distinctPerm p$ 
  and  $xvec \#* P$ 
  and  $(p \cdot xvec) \#* P$ 
```

shows $\Psi \triangleright P \longmapsto M((p \cdot N)) \prec (p \cdot P')$
 $\langle proof \rangle$

```
lemma frameFresh[dest]:
  fixes  $x$  :: name
  and  $A_F$  :: name list
  and  $\Psi_F$  :: ' $b$ 

  assumes  $x \notin A_F$ 
  and  $x \notin \langle A_F, \Psi_F \rangle$ 
```

shows $x \notin \Psi_F$
 $\langle proof \rangle$

```
lemma outputSwapFrameSubject:
  fixes  $\Psi$  :: ' $b$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $M$  :: ' $a$ 
  and  $xvec$  :: name list
  and  $N$  :: ' $a$ 
  and  $x$  :: name
  and  $y$  :: name
```

```
assumes  $\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
and  $xvec \#* M$ 
and  $x \notin P$ 
and  $y \notin P$ 
```

shows $((x, y) \cdot \Psi) \triangleright P \longmapsto ((x, y) \cdot M)(\nu*xvec)\langle N \rangle \prec P'$
 $\langle proof \rangle$

```

lemma outputPermFrameSubject:
  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
  and  $p$  :: name prm
  and  $yvec$  :: name list
  and  $zvec$  :: name list

  assumes  $\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
  and  $S: set p \subseteq set yvec \times set zvec$ 
  and  $yvec \#* P$ 
  and  $zvec \#* P$ 

  shows  $(p \cdot \Psi) \triangleright P \longmapsto (p \cdot M)(\nu*xvec)\langle N \rangle \prec P'$ 
  (proof)

lemma outputSwapSubject:
  fixes  $\Psi$  :: 'b
  and  $P$  :: ('a, 'b, 'c) psi
  and  $M$  :: 'a
  and  $B$  :: ('a, 'b, 'c) boundOutput
  and  $x$  :: name
  and  $y$  :: name

  assumes  $\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
  and  $xvec \#* M$ 
  and  $x \# P$ 
  and  $y \# P$ 
  and  $x \# \Psi$ 
  and  $y \# \Psi$ 

  shows  $\Psi \triangleright P \longmapsto [(x, y)] \cdot M)(\nu*xvec)\langle N \rangle \prec P'$ 
  (proof)

lemma outputPermSubject:
  fixes  $\Psi$  :: '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  $\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
  and  $S: set p \subseteq set yvec \times set zvec$ 

```

```

and      yvec #* P
and      zvec #* P
and      yvec #* Ψ
and      zvec #* Ψ

shows Ψ ⊢ P ↣ (p · M)(ν*xvec)⟨N⟩ ⊸ P'
⟨proof⟩

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 Ψ ⊢ P ↣ M(ν*xvec)⟨N⟩ ⊸ P'
and   xvec #* M
and   x # P
and   y # P
and   x # M
and   y # M

shows ([x, y] · Ψ) ⊢ P ↣ M(ν*xvec)⟨N⟩ ⊸ P'
⟨proof⟩

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 Ψ ⊢ P ↣ M(ν*xvec)⟨N⟩ ⊸ P'
and   S: set p ⊆ set yvec × set zvec
and   yvec #* P
and   zvec #* P
and   yvec #* M
and   zvec #* M

shows (p · Ψ) ⊢ P ↣ M(ν*xvec)⟨N⟩ ⊸ P'
⟨proof⟩

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 AP :: name list
and ΨP :: 'b
and Q :: ('a, 'b, 'c) psi
and K :: 'a
and xvec :: name list
and Q' :: ('a, 'b, 'c) psi
and AQ :: name list

assumes Ψ ⊢ P ┤→ M(|N|) ⊲ P'
and extractFrame P = ⟨AP, ΨP⟩
and Ψ ⊢ Q ┤→ K(|ν*xvec|)⟨N⟩ ⊲ Q'
and extractFrame Q = ⟨AQ, ΨQ⟩
and Ψ ⊢ ΨP ⊗ ΨQ ⊢ M ↔ K
and AP #* Ψ
and AP #* P
and AP #* Q
and AP #* M
and AP #* AQ
and AQ #* Ψ
and AQ #* P
and AQ #* Q
and AQ #* K
and xvec #* P

```

shows Ψ ⊢ P || Q ┤→τ ⊲ (|ν*xvec|)(P' || Q')
(proof)

lemma Comm2:

```

fixes Ψ :: 'b
and ΨQ :: 'b
and P :: ('a, 'b, 'c) psi
and M :: 'a
and xvec :: name list
and N :: 'a
and P' :: ('a, 'b, 'c) psi
and AP :: name list
and ΨP :: 'b
and Q :: ('a, 'b, 'c) psi
and K :: 'a
and Q' :: ('a, 'b, 'c) psi
and AQ :: name list

assumes Ψ ⊢ P ┤→ M(|ν*xvec|)⟨N⟩ ⊲ P'
and extractFrame P = ⟨AP, ΨP⟩
and Ψ ⊢ Q ┤→ K(|N|) ⊲ Q'
and extractFrame Q = ⟨AQ, ΨQ⟩

```

```

and    $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$ 
and    $A_P \#* \Psi$ 
and    $A_P \#* P$ 
and    $A_P \#* Q$ 
and    $A_P \#* M$ 
and    $A_P \#* A_Q$ 
and    $A_Q \#* \Psi$ 
and    $A_Q \#* P$ 
and    $A_Q \#* Q$ 
and    $A_Q \#* K$ 
and    $xvec \#* Q$ 

shows  $\Psi \triangleright P \parallel Q \xrightarrow{\tau} \prec (\nu * xvec)(P' \parallel Q')$ 
 $\langle proof \rangle$ 

lemma semanticsCasesAux[consumes 1, case-names cInput cOutput cCase cPar1
cPar2 cComm1 cComm2 cOpen cScope cBang]:
fixes  $\Psi :: 'b$ 
and    $cP :: ('a, 'b, 'c) psi$ 
and    $cRs :: ('a, 'b, 'c) residual$ 
and    $C :: 'd::fs-name$ 
and    $x :: name$ 

assumes  $\Psi \triangleright cP \mapsto cRs$ 
and    $rInput: \bigwedge M K xvec N Tvec P. [cP = M(\lambda * xvec N).P; cRs = K((N[xvec:=Tvec]))]$ 
 $\prec P[xvec:=Tvec];$ 
 $\quad \quad \quad \Psi \vdash M \leftrightarrow K; distinct xvec; set xvec \subseteq supp N;$ 
 $length xvec = length Tvec;$ 
 $\quad \quad \quad xvec \#* Tvec; xvec \#* \Psi; xvec \#* M; xvec \#* K;$ 
 $xvec \#* C] \implies Prop$ 
and    $rOutput: \bigwedge M K N P. [cP = M\langle N \rangle.P; cRs = K\langle N \rangle \prec P; \Psi \vdash M \leftrightarrow K]$ 
 $\implies Prop$ 
and    $rCase: \bigwedge Cs P \varphi. [cP = Cases Cs; \Psi \triangleright P \mapsto cRs; (\varphi, P) mem Cs; \Psi$ 
 $\vdash \varphi; guarded P] \implies Prop$ 

and    $rPar1: \bigwedge \Psi_Q P \alpha P' Q A_Q. [cP = P \parallel Q; cRs = \alpha \prec (P' \parallel Q);$ 
 $(\Psi \otimes \Psi_Q) \triangleright P \mapsto (\alpha \prec P'); extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct$ 
 $A_Q;$ 
 $A_Q \#* P; A_Q \#* Q; A_Q \#* \Psi; A_Q \#* \alpha; A_Q \#* C; A_Q \#* P'; bn \alpha \#*$ 
 $\Psi; bn \alpha \#* \Psi_Q;$ 
 $bn \alpha \#* Q; bn \alpha \#* P; bn \alpha \#* subject \alpha; bn \alpha \#* C; distinct(bn \alpha)]$ 
 $\implies$ 
 $Prop$ 
and    $rPar2: \bigwedge \Psi_P Q \alpha Q' P A_P. [cP = P \parallel Q; cRs = \alpha \prec (P \parallel Q');$ 
 $(\Psi \otimes \Psi_P) \triangleright Q \mapsto \alpha \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct$ 
 $A_P;$ 
 $A_P \#* P; A_P \#* Q; A_P \#* \Psi; A_P \#* \alpha; A_P \#* C;$ 
 $A_P \#* Q'; bn \alpha \#* \Psi; bn \alpha \#* \Psi_P; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* subject$ 
 $\alpha; bn \alpha \#* C; distinct(bn \alpha)] \implies Prop$ 

```

and $rComm1: \bigwedge \Psi_Q P M N P' A_P \Psi_P Q K xvec Q' A_Q.$
 $\llbracket cP = P \parallel Q; cRs = \tau \prec (\nu*xvec)P' \parallel Q';$
 $\Psi \otimes \Psi_Q \triangleright P \mapsto M(N) \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle;$
 $distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K(\nu*xvec)(N) \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle;$
 $distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#*$
 $M; A_P \#* N;$
 $A_P \#* P'; A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi;$
 $A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* Q'; A_Q$
 $\#* xvec;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M; xvec \#*$
 $Q;$
 $xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C; distinct xvec] \implies Prop$

and $rComm2: \bigwedge \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q.$
 $\llbracket cP = P \parallel Q; cRs = \tau \prec (\nu*xvec)P' \parallel Q';$
 $\Psi \otimes \Psi_Q \triangleright P \mapsto M(\nu*xvec)(N) \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle;$
 $distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K(N) \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle;$
 $distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#*$
 $M; A_P \#* N;$
 $A_P \#* P'; A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi;$
 $A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* Q'; A_Q$
 $\#* xvec;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M; xvec \#*$
 $Q;$
 $xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C; distinct xvec] \implies Prop$

and $rOpen: \bigwedge P M xvec yvec N P' x.$
 $\llbracket cP = (\nu x)P; cRs = M(\nu*(xvec @ x \# yvec)) \langle N \rangle \prec P';$
 $\Psi \triangleright P \mapsto M(\nu*(xvec @ yvec)) \langle N \rangle \prec P'; x \in supp N; x \# xvec; x \#$
 $yvec; x \# M; x \# \Psi; distinct xvec; distinct yvec;$
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* yvec; yvec \#* \Psi; yvec \#* P;$
 $yvec \#* M; xvec \#* C; x \# C; yvec \#* C] \implies$
 $Prop$

and $rScope: \bigwedge P \alpha P' x. [cP = (\nu x)P; cRs = \alpha \prec (\nu x)P';$
 $\Psi \triangleright P \mapsto \alpha \prec P'; x \# \Psi; x \# \alpha; x \# C; bn \alpha \#* \Psi; bn \alpha$
 $\#* P; bn \alpha \#* subject \alpha; bn \alpha \#* C; distinct(bn \alpha)] \implies Prop$

and $rBang: \bigwedge P. [cP = !P; \Psi \triangleright P \parallel !P \mapsto cRs; guarded P] \implies Prop$

shows $Prop$
 $\langle proof \rangle$

nominal-primrec

$inputLength :: ('a::fs-name, 'b::fs-name, 'c::fs-name) psi \Rightarrow nat$
and $inputLength' :: ('a::fs-name, 'b::fs-name, 'c::fs-name) input \Rightarrow nat$
and $inputLength'' :: ('a::fs-name, 'b::fs-name, 'c::fs-name) psiCase \Rightarrow nat$

```

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

| inputLength' (Trm M P) = 0
| inputLength' ( $\nu y I$ ) = 1 + (inputLength' I)

| inputLength'' ( $\perp_c$ ) = 0
| inputLength'' ( $\Box \Phi \Rightarrow P C$ ) = 0
⟨proof⟩

nominal-primrec boundOutputLength :: ('a, 'b, 'c) boundOutput  $\Rightarrow$  nat
where
boundOutputLength (BOut M P) = 0
| boundOutputLength (BStep x B) = (boundOutputLength B) + 1
⟨proof⟩

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
⟨proof⟩

lemma inputLengthProc[simp]:
shows inputLength(M( $\lambda * xvec N$ ).P) = length xvec
⟨proof⟩

lemma boundOutputLengthSimp[simp]:
shows residualLength(M( $\nu * xvec$ ). $\langle N \rangle \prec P$ ) = length xvec
⟨proof⟩

lemma boundOutputLengthSimp2[simp]:
shows residualLength( $\alpha \prec P$ ) = length(bn α)
⟨proof⟩

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 #* yvec
and    distinct xvec
and    distinct yvec

obtains p where set p ⊆ set xvec × set(p · xvec) and distinctPerm p and yvec
= p · xvec
⟨proof⟩

lemma distinctAppend[simp]:
fixes xvec :: name list
and    yvec :: name list

shows (set xvec ∩ set yvec = {}) = xvec #* yvec
⟨proof⟩

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
⟨proof⟩

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⟩

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 $\Psi \triangleright cP \mapsto cRs$
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 $rInput: \bigwedge M\ K\ N\ Tvec\ P. ([xvec1 \#* \Psi; xvec1 \#* cP; xvec1 \#* cRs] \implies cP = M(\lambda*xvec1\ N).P \wedge cRs = K((N[xvec1::=Tvec])) \prec P[xvec1::=Tvec] \wedge \Psi \vdash M \leftrightarrow K \wedge distinct\ xvec1 \wedge set\ xvec1 \subseteq supp\ N \wedge length\ xvec1 = length\ Tvec \wedge xvec1 \#* Tvec \wedge xvec1 \#* \Psi \wedge xvec1 \#* M \wedge xvec1 \#* K) \implies Prop$
and $rOutput: \bigwedge M\ K\ N\ P. [cP = M\langle N \rangle.P; cRs = K\langle N \rangle \prec P; \Psi \vdash M \leftrightarrow K] \implies Prop$
and $rCase: \bigwedge Cs\ P\ \varphi. [cP = Cases\ Cs; \Psi \triangleright P \mapsto cRs; (\varphi, P) mem\ Cs; \Psi \vdash \varphi; guarded\ P] \implies Prop$
and $rPar1: \bigwedge \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q. ([xvec2 \#* \Psi; xvec2 \#* cP; xvec2 \#* cRs] \implies cP = P \parallel Q \wedge cRs = \alpha \prec (P' \parallel Q) \wedge xvec2 = bn\ \alpha \wedge \Psi \otimes \Psi_Q \triangleright P \mapsto \alpha \prec P' \wedge extractFrame\ Q = \langle A_Q, \Psi_Q \rangle \wedge distinct\ A_Q \wedge A_Q \#* P \wedge A_Q \#* Q \wedge A_Q \#* \Psi \wedge A_Q \#* \alpha \wedge A_Q \#* P' \wedge A_Q \#* C) \implies Prop$
and $rPar2: \bigwedge \Psi_P\ Q\ \alpha\ Q'\ P\ A_P. ([xvec3 \#* \Psi; xvec3 \#* cP; xvec3 \#* cRs] \implies cP = P \parallel Q \wedge cRs = \alpha \prec (P \parallel Q') \wedge xvec3 = bn\ \alpha \wedge \Psi \otimes \Psi_P \triangleright Q \mapsto \alpha \prec Q' \wedge extractFrame\ P = \langle A_P, \Psi_P \rangle \wedge distinct\ A_P \wedge A_P \#* P \wedge A_P \#* Q \wedge A_P \#* \Psi \wedge A_P \#* \alpha \wedge A_P \#* Q' \wedge A_P \#* C) \implies Prop$
and $rComm1: \bigwedge \Psi_Q\ P\ M\ N\ P'\ A_P\ \Psi_P\ Q\ K\ xvec\ Q'\ A_Q. [cP = P \parallel Q; cRs = \tau \prec (\nu*xvec)P' \parallel Q'; \Psi \otimes \Psi_Q \triangleright P \mapsto M(N) \prec P'; extractFrame\ P = \langle A_P, \Psi_P \rangle; distinct\ A_P; \Psi \otimes \Psi_P \triangleright Q \mapsto K(\nu*xvec)\langle N \rangle \prec Q'; extractFrame\ Q = \langle A_Q, \Psi_Q \rangle; distinct\ A_Q; A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* Q'; A_Q \#* xvec; xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M; xvec \#* Q;$

$xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C; distinct xvec] \implies Prop$
and $rComm2: \bigwedge \Psi_Q P M xvec N P' A_P \Psi_P Q K Q' A_Q.$
 $\llbracket cP = P \parallel Q; cRs = \tau \prec (\nu*xvec)P' \parallel Q';$
 $\Psi \otimes \Psi_Q \triangleright P \mapsto M(\nu*xvec)\langle N \rangle \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K\langle N \rangle \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#*$
 $M; A_P \#* N;$
 $A_P \#* P'; A_P \#* Q; A_P \#* Q'; A_P \#* A_Q; A_P \#* xvec; A_Q \#* \Psi;$
 $A_Q \#* \Psi_P;$
 $A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* Q'; A_Q$
 $\#* xvec;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* \Psi_Q; xvec \#* P; xvec \#* M; xvec \#*$
 $Q;$
 $xvec \#* K; A_P \#* C; A_Q \#* C; xvec \#* C; distinct xvec] \implies Prop$
and $rOpen: \bigwedge P M xvec y yvec N P'.$
 $\llbracket [xvec4 \#* \Psi; xvec4 \#* cP; xvec4 \#* cRs; x1 \# \Psi; x1 \# cP; x1 \#$
 $cRs; x1 \# xvec4] \implies$
 $cP = (\nu x1)P \wedge cRs = M(\nu*(xvec@x1#yvec))\langle N \rangle \prec P' \wedge$
 $xvec4 = xvec@y#yvec \wedge$
 $\Psi \triangleright P \mapsto M(\nu*(xvec@yvec))\langle N \rangle \prec P' \wedge x1 \in supp N \wedge x1 \#$
 $xvec \wedge x1 \# yvec \wedge$
 $distinct xvec \wedge distinct yvec \wedge xvec \#* \Psi \wedge xvec \#* P \wedge xvec \#* M$
 $\wedge xvec \#* yvec \wedge$
 $yvec \#* \Psi \wedge yvec \#* P \wedge yvec \#* M) \implies Prop$
and $rScope: \bigwedge P \alpha P'. ([xvec5 \#* \Psi; xvec5 \#* cP; xvec5 \#* cRs; x2 \# \Psi; x2 \#$
 $cP; x2 \# cRs; x2 \# xvec5] \implies$
 $cP = (\nu x2)P \wedge cRs = \alpha \prec (\nu x2)P' \wedge xvec5 = bn \alpha \wedge$
 $\Psi \triangleright P \mapsto \alpha \prec P' \wedge x2 \# \Psi \wedge x2 \# \alpha \wedge bn \alpha \#* subject \alpha$
 $\wedge distinct(bn \alpha)) \implies Prop$
and $rBang: \bigwedge P. [cP = !P; \Psi \triangleright P \parallel !P \mapsto cRs; guarded P] \implies Prop$
shows $Prop$
 $\langle proof \rangle$

lemma $parCases[consumes 5, case-names cPar1 cPar2 cComm1 cComm2]:$

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Q :: ('a, 'b, 'c) psi$
and $\alpha :: 'a action$
and $T :: ('a, 'b, 'c) psi$
and $C :: 'd::fs-name$

assumes $Trans: \Psi \triangleright P \parallel Q \mapsto \alpha \prec T$
and $bn \alpha \#* \Psi$
and $bn \alpha \#* P$
and $bn \alpha \#* Q$
and $bn \alpha \#* subject \alpha$
and $rPar1: \bigwedge P' A_Q \Psi_Q. [\Psi \otimes \Psi_Q \triangleright P \mapsto \alpha \prec P'; extractFrame Q = \langle A_Q,$

$\Psi_Q\rangle$; distinct A_Q ;

$$A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* \alpha; A_Q \#* P'; A_Q \#* C] \implies \text{Prop } \alpha (P' \parallel Q)$$

and $rPar2: \bigwedge Q' A_P \Psi_P$. $[\Psi \otimes \Psi_P \triangleright Q \mapsto \alpha \prec Q'$; extractFrame $P = \langle A_P, \Psi_P \rangle$; distinct A_P ;

$$A_P \#* \Psi; A_P \#* P; A_P \#* Q; A_P \#* \alpha; A_P \#* Q'; A_P \#* C] \implies \text{Prop } \alpha (P \parallel Q')$$

and $rComm1: \bigwedge \Psi_Q M N P' A_P \Psi_P K xvec Q' A_Q$.

$$[\Psi \otimes \Psi_Q \triangleright P \mapsto M(N) \prec P'$$
; extractFrame $P = \langle A_P, \Psi_P \rangle$; distinct A_P ;
$$\Psi \otimes \Psi_P \triangleright Q \mapsto K(\nu*xvec)\langle N \rangle \prec Q'$$
; extractFrame $Q = \langle A_Q, \Psi_Q \rangle$;

distinct A_Q ;

$$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$$
; distinct $xvec$;
$$A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$$

$$A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$$

$$xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q; xvec \#* \Psi_Q; xvec \#* C] \implies$$

$$\text{Prop } (\tau) ((\nu*xvec)(P' \parallel Q'))$$

and $rComm2: \bigwedge \Psi_Q M xvec N P' A_P \Psi_P K Q' A_Q$.

$$[\Psi \otimes \Psi_Q \triangleright P \mapsto M(\nu*xvec)\langle N \rangle \prec P'$$
; extractFrame $P = \langle A_P, \Psi_P \rangle$;

distinct A_P ;

$$\Psi \otimes \Psi_P \triangleright Q \mapsto K(N) \prec Q'$$
; extractFrame $Q = \langle A_Q, \Psi_Q \rangle$; distinct A_Q ;
$$\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K$$
; distinct $xvec$;
$$A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$$

$$A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$$

$$xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q; xvec \#* \Psi_Q; xvec \#* C] \implies$$

$$\text{Prop } (\tau) ((\nu*xvec)(P' \parallel Q'))$$

shows $\text{Prop } \alpha T$
 $\langle proof \rangle$

lemma $\text{parInputCases}[\text{consumes } 1, \text{ case-names } cPar1 \text{ } cPar2]$:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $M :: 'a$
and $N :: 'a$
and $R :: ('a, 'b, 'c) \text{ psi}$
and $C :: 'd::fs-name$

assumes $\text{Trans}: \Psi \triangleright P \parallel Q \mapsto M(N) \prec R$
and $rPar1: \bigwedge P' A_Q \Psi_Q$. $[\Psi \otimes \Psi_Q \triangleright P \mapsto M(N) \prec P'$; extractFrame $Q = \langle A_Q, \Psi_Q \rangle$; distinct A_Q ;

$$A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* M; A_Q \#* N; A_Q \#* C] \implies \text{Prop } (P' \parallel Q)$$

and $rPar2: \bigwedge Q' A_P \Psi_P. [\Psi \otimes \Psi_P \triangleright Q \xrightarrow{M(N)} \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $A_P \#* \Psi; A_P \#* P; A_P \#* Q; A_P \#* M; A_P \#* N; A_P \#* C]$
 $\implies Prop(P \parallel Q')$
shows $Prop R$
 $\langle proof \rangle$

lemma $parOutputCases[consumes 5, case-names cPar1 cPar2]:$

fixes $\Psi :: 'b$
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 $Trans: \Psi \triangleright P \parallel Q \xrightarrow{M(\nu*xvec)} \langle N \rangle \prec R$
and $xvec \#* \Psi$
and $xvec \#* P$
and $xvec \#* Q$
and $xvec \#* M$
and $rPar1: \bigwedge P' A_Q \Psi_Q. [\Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(\nu*xvec)} \langle N \rangle \prec P'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* M; A_Q \#* xvec; A_Q \#* N;$
 $A_Q \#* C; A_Q \#* xvec; distinct xvec] \implies Prop(P' \parallel Q)$
and $rPar2: \bigwedge Q' A_P \Psi_P. [\Psi \otimes \Psi_P \triangleright Q \xrightarrow{M(\nu*xvec)} \langle N \rangle \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $A_P \#* \Psi; A_P \#* P; A_P \#* Q; A_P \#* M; A_P \#* xvec; A_P \#* N;$
 $A_P \#* C; A_P \#* xvec; distinct xvec] \implies Prop(P \parallel Q')$
shows $Prop R$
 $\langle proof \rangle$

lemma $theEqvt[eqvt-force]:$

fixes $p :: name prm$
and $\alpha :: 'a action$

assumes $\alpha \neq \tau$

shows $(p \cdot the(subject \alpha)) = the(p \cdot (subject \alpha))$
 $\langle proof \rangle$

lemma $theSubjectFresh[simp]:$

fixes $\alpha :: 'a action$
and $x :: name$

assumes $\alpha \neq \tau$

shows $x \# the(subject \alpha) = x \# subject \alpha$

$\langle proof \rangle$

```
lemma theSubjectFreshChain[simp]:
  fixes  $\alpha :: 'a action$ 
  and  $xvec :: name list$ 
```

```
assumes  $\alpha \neq \tau$ 
```

shows $xvec \#* the(subject \alpha) = xvec \#* subject \alpha$

$\langle proof \rangle$

```
lemma obtainPrefix:
```

```
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $\alpha :: 'a action$ 
and  $P' :: ('a, 'b, 'c) psi$ 
and  $A_P :: name list$ 
and  $\Psi_P :: 'b$ 
and  $B :: name list$ 
```

```
assumes  $\Psi \triangleright P \longmapsto \alpha \prec P'$ 
```

```
and  $extractFrame P = \langle A_P, \Psi_P \rangle$ 
and  $distinct A_P$ 
and  $bn \alpha \#* subject \alpha$ 
and  $distinct(bn \alpha)$ 
and  $\alpha \neq \tau$ 
and  $B \#* P$ 
and  $A_P \#* \Psi$ 
and  $A_P \#* B$ 
and  $A_P \#* P$ 
and  $A_P \#* subject \alpha$ 
```

obtains M **where** $\Psi \otimes \Psi_P \vdash the(subject \alpha) \leftrightarrow M$ **and** $B \#* M$

$\langle proof \rangle$

```
lemma inputRenameSubject:
```

```
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $M :: 'a$ 
and  $N :: 'a$ 
and  $P' :: ('a, 'b, 'c) psi$ 
and  $A_P :: name list$ 
and  $\Psi_P :: 'b$ 
```

```
assumes  $\Psi \triangleright P \longmapsto M(N) \prec P'$ 
```

```
and  $extractFrame P = \langle A_P, \Psi_P \rangle$ 
and  $distinct A_P$ 
and  $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$ 
and  $A_P \#* \Psi$ 
```

```

and       $A_P \#* P$ 
and       $A_P \#* M$ 
and       $A_P \#* K$ 

shows  $\Psi \triangleright P \longmapsto K(N) \prec P'$ 
(proof)

```

lemma *outputRenameSubject*:

```

fixes  $\Psi :: 'b$ 
and     $P :: ('a, 'b, 'c) \psi$ 
and     $M :: 'a$ 
and     $xvec :: name list$ 
and     $N :: 'a$ 
and     $P' :: ('a, 'b, 'c) \psi$ 
and     $A_P :: name list$ 
and     $\Psi_P :: 'b$ 

assumes  $\Psi \triangleright P \longmapsto M(\nu*xvec)(N) \prec P'$ 
and     $extractFrame P = \langle A_P, \Psi_P \rangle$ 
and     $distinct A_P$ 
and     $\Psi \otimes \Psi_P \vdash M \leftrightarrow K$ 
and     $A_P \#* \Psi$ 
and     $A_P \#* P$ 
and     $A_P \#* M$ 
and     $A_P \#* K$ 

```

```

shows  $\Psi \triangleright P \longmapsto K(\nu*xvec)(N) \prec P'$ 
(proof)

```

lemma *parCasesSubject*[consumes 7, case-names *cPar1* *cPar2* *cComm1* *cComm2*]:

```

fixes  $\Psi :: 'b$ 
and     $P :: ('a, 'b, 'c) \psi$ 
and     $Q :: ('a, 'b, 'c) \psi$ 
and     $\alpha :: 'a action$ 
and     $R :: ('a, 'b, 'c) \psi$ 
and     $C :: 'd::fs-name$ 
and     $yvec :: name list$ 

```

```

assumes Trans:  $\Psi \triangleright P \parallel Q \longmapsto \alpha \prec R$ 
and       $bn \alpha \#* \Psi$ 
and       $bn \alpha \#* P$ 
and       $bn \alpha \#* Q$ 
and       $bn \alpha \#* subject \alpha$ 
and       $yvec \#* P$ 
and       $yvec \#* Q$ 
and       $rPar1: \bigwedge P' A_Q \Psi_Q. [\Psi \otimes \Psi_Q \triangleright P \longmapsto \alpha \prec P'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$ 
                    $A_Q \#* \Psi; A_Q \#* P; A_Q \#* \alpha; A_Q \#* C] \implies Prop \alpha (P' \parallel Q)$ 
and       $rPar2: \bigwedge Q' A_P \Psi_P. [\Psi \otimes \Psi_P \triangleright Q \longmapsto \alpha \prec Q'; extractFrame P = \langle A_P,$ 

```

$\Psi_P \rangle; \text{distinct } A_P;$
 $A_P \#* \Psi; A_P \#* Q; A_P \#* \alpha; A_P \#* C] \implies \text{Prop } \alpha (P \parallel Q')$
and $rComm1: \bigwedge \Psi_Q M N P' A_P \Psi_P K xvec Q' A_Q.$
 $[\Psi \otimes \Psi_Q \triangleright P \mapsto M(N) \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle; \text{distinct } A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K(\nu*xvec)\langle N \rangle \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$
 $\text{distinct } A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; yvec \#* M; yvec \#* K; \text{distinct } xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P$
 $\#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#*$
 $Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q;$
 $xvec \#* \Psi_Q; xvec \#* C] \implies$
 $\text{Prop } (\tau) ((\nu*xvec)(P' \parallel Q'))$
and $rComm2: \bigwedge \Psi_Q M xvec N P' A_P \Psi_P K Q' A_Q.$
 $[\Psi \otimes \Psi_Q \triangleright P \mapsto M(\nu*xvec)\langle N \rangle \prec P'; \text{extractFrame } P = \langle A_P, \Psi_P \rangle;$
 $\text{distinct } A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \mapsto K(N) \prec Q'; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle; \text{distinct } A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; yvec \#* M; yvec \#* K; \text{distinct } xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P$
 $\#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#*$
 $Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q;$
 $xvec \#* \Psi_Q; xvec \#* C] \implies$
 $\text{Prop } (\tau) ((\nu*xvec)(P' \parallel Q'))$

shows $\text{Prop } \alpha R$
 $\langle proof \rangle$

lemma $\text{inputCases}[\text{consumes 1, case-names cInput}]:$
fixes $\Psi :: 'b$
and $M :: 'a$
and $xvec :: \text{name list}$
and $N :: 'a$
and $P :: ('a, 'b, 'c) \text{psi}$
and $\alpha :: 'a \text{ action}$
and $P' :: ('a, 'b, 'c) \text{psi}$

assumes $\text{Trans}: \Psi \triangleright M(\lambda*xvec N).P \mapsto \alpha \prec P'$
and $rInput: \bigwedge K \text{Tvec}. [\Psi \vdash M \leftrightarrow K; \text{set } xvec \subseteq \text{supp } N; \text{length } xvec = \text{length } Tvec; \text{distinct } xvec] \implies \text{Prop } (K(N[xvec:=Tvec])) (P[xvec:=Tvec])$

shows $\text{Prop } \alpha P'$
 $\langle proof \rangle$

lemma $\text{outputCases}[\text{consumes 1, case-names cOutput}]:$
fixes $\Psi :: 'b$
and $M :: 'a$

```

and    $N :: 'a$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $\alpha :: 'a \text{ action}$ 
and    $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright M\langle N \rangle.P \xrightarrow{\alpha} P'$ 
and    $\bigwedge K. \Psi \vdash M \leftrightarrow K \implies \text{Prop}(K\langle N \rangle) P$ 

shows  $\text{Prop } \alpha \ P'$ 
 $\langle \text{proof} \rangle$ 

lemma caseCases[consumes 1, case-names cCase]:
fixes  $\Psi :: 'b$ 
and    $Cs :: ('c \times ('a, 'b, 'c) \psi) \text{ list}$ 
and    $\alpha :: 'a \text{ action}$ 
and    $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\text{Trans}: \Psi \triangleright (\text{Cases } Cs) \mapsto Rs$ 
and    $rCase: \bigwedge \varphi P. [\Psi \triangleright P \mapsto Rs; (\varphi, P) \text{ mem } Cs; \Psi \vdash \varphi; \text{guarded } P] \implies$ 
 $Prop$ 

shows  $\text{Prop}$ 
 $\langle \text{proof} \rangle$ 

lemma resCases[consumes 7, case-names cOpen cRes]:
fixes  $\Psi :: 'b$ 
and    $x :: \text{name}$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $\alpha :: 'a \text{ action}$ 
and    $P' :: ('a, 'b, 'c) \psi$ 
and    $C :: 'd::fs\text{-name}$ 

assumes  $\text{Trans}: \Psi \triangleright (\nu x)P \xrightarrow{\alpha} P'$ 
and    $x \notin \Psi$ 
and    $x \notin \alpha$ 
and    $x \notin P'$ 
and    $bn \alpha \notin \Psi$ 
and    $bn \alpha \notin P$ 
and    $bn \alpha \notin \text{subject } \alpha$ 
and    $rOpen: \bigwedge M xvec yvec y N P'. [\Psi \triangleright P \mapsto M(\nu*(xvec@yvec)) \langle ((x, y)] \cdot N \rangle \prec ((x, y)] \cdot P'); y \in \text{supp } N;$ 
 $x \notin N; x \notin P'; x \neq y; y \notin xvec; y \notin yvec; y \notin M;$ 
 $\text{distinct } xvec; \text{ distinct } yvec;$ 
 $xvec \notin \Psi; y \notin \Psi; yvec \notin \Psi; xvec \notin P; y \notin P;$ 
 $yvec \notin P; xvec \notin M; y \notin M;$ 
 $yvec \notin M; xvec \notin yvec] \implies$ 
 $\text{Prop}(M(\nu*(xvec@y#yvec)) \langle N \rangle) P'$ 
and    $rScope: \bigwedge P'. [\Psi \triangleright P \xrightarrow{\alpha} P'] \implies \text{Prop } \alpha ((\nu x)P')$ 

```

shows $\text{Prop } \alpha \ P'$
 $\langle \text{proof} \rangle$

```

lemma resCases'[consumes  $\gamma$ , case-names  $cOpen\ cRes$ ]:
  fixes  $\Psi :: 'b$ 
  and  $x :: \text{name}$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $\alpha :: 'a \text{ action}$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  and  $C :: 'd::\text{fs-name}$ 

  assumes Trans:  $\Psi \triangleright (\nu x)P \longmapsto \alpha \prec P'$ 
  and  $x \notin \Psi$ 
  and  $x \notin \alpha$ 
  and  $x \notin P'$ 
  and  $bn \alpha \notin \Psi$ 
  and  $bn \alpha \notin P$ 
  and  $bn \alpha \notin \text{subject } \alpha$ 
  and rOpen:  $\bigwedge M \ xvec \ yvec \ y \ N \ P'. [\Psi \triangleright ((x, y)] \cdot P) \longmapsto M(\nu*(xvec @ yvec)) \langle N \rangle$ 
   $\prec P'; y \in \text{supp } N;$ 
   $x \notin N; x \notin P'; x \neq y; y \notin xvec; y \notin yvec; y \notin M;$ 
  distinct xvec; distinct yvec;
   $xvec \notin \Psi; y \notin \Psi; yvec \notin \Psi; xvec \notin P; y \notin P;$ 
   $yvec \notin P; xvec \notin M; y \notin M;$ 
   $yvec \notin M; xvec \notin yvec] \implies$ 
  Prop  $(M(\nu*(xvec @ y#yvec)) \langle N \rangle) \ P'$ 
  and rScope:  $\bigwedge P'. [\Psi \triangleright P \longmapsto \alpha \prec P] \implies \text{Prop } \alpha ((\nu x)P')$ 

  shows  $\text{Prop } \alpha \ P'$ 
   $\langle \text{proof} \rangle$ 

```

abbreviation
 $\text{statImpJudge} (\leftarrow \hookrightarrow [80, 80] 80)$
where $\Psi \hookrightarrow \Psi' \equiv \text{AssertionStatImp } \Psi \ \Psi'$

```

lemma statEqTransition:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Rs :: ('a, 'b, 'c) \text{ residual}$ 
  and  $\Psi' :: 'b$ 

  assumes  $\Psi \triangleright P \longmapsto Rs$ 
  and  $\Psi \simeq \Psi'$ 

  shows  $\Psi' \triangleright P \longmapsto Rs$ 
   $\langle \text{proof} \rangle$ 

lemma actionPar1Dest':
  fixes  $\alpha :: ('a::\text{fs-name}) \text{ action}$ 

```

```

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 α ⊢ P = β ⊢ (Q || R)
and bn α #* R
and bn β #* R

obtains T where P = T || R and α ⊢ T = β ⊢ Q
⟨proof⟩

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 α ⊢ P = β ⊢ (Q || R)
  and bn α #* Q
  and bn β #* Q

  obtains T where P = Q || T and α ⊢ T = β ⊢ R
⟨proof⟩

lemma expandNonTauFrame:
  fixes Ψ :: 'b
  and P :: ('a, 'b, 'c) psi
  and α :: 'a action
  and P' :: ('a, 'b, 'c) psi
  and AP :: name list
  and ΨP :: 'b
  and C :: 'd::fs-name
  and C' :: 'e::fs-name

  assumes Trans: Ψ ⊢ P ↦ α ⊢ P'
  and extractFrame P = ⟨AP, ΨP⟩
  and distinct AP
  and bn α #* subject α
  and AP #* P
  and AP #* α
  and AP #* C
  and AP #* C'
  and bn α #* P
  and bn α #* C'
  and α ≠ τ

  obtains p Ψ' AP' ΨP' where set p ⊆ set(bn α) × set(bn(p · α)) and (p · ΨP)

```

$\otimes \Psi' \simeq \Psi_P'$ and $distinctPerm p$ and
 $extractFrame P' = \langle A_{P'}, \Psi_{P'} \rangle$ and $A_{P'} \#* P'$ and $A_{P'} \#*$
 α and $A_{P'} \#* (p \cdot \alpha)$ and
 $A_{P'} \#* C$ and $(bn(p \cdot \alpha)) \#* C'$ and $(bn(p \cdot \alpha)) \#* \alpha$ and
 $(bn(p \cdot \alpha)) \#* P'$ and $distinct A_{P'}$
 $\langle proof \rangle$

lemma *expandTauFrame*:

```

fixes  $\Psi$  :: 'b
and  $P$  :: ('a, 'b, 'c) psi
and  $P'$  :: ('a, 'b, 'c) psi
and  $A_P$  :: name list
and  $\Psi_P$  :: 'b
and  $C$  :: 'd::fs-name

assumes  $\Psi \triangleright P \longmapsto \tau \prec P'$ 
and  $extractFrame P = \langle A_P, \Psi_P \rangle$ 
and  $distinct A_P$ 
and  $A_P \#* P$ 
and  $A_P \#* C$ 

```

$\text{obtains } \Psi' A_{P'} \Psi_{P'}$ where $extractFrame P' = \langle A_{P'}, \Psi_{P'} \rangle$ and $\Psi_P \otimes \Psi' \simeq$
 $\Psi_{P'}$ and $A_{P'} \#* C$ and $A_{P'} \#* P'$ and $distinct A_{P'}$
 $\langle proof \rangle$

lemma *expandFrame*:

```

fixes  $\Psi$  :: 'b
and  $P$  :: ('a, 'b, 'c) psi
and  $\alpha$  :: 'a action
and  $P'$  :: ('a, 'b, 'c) psi
and  $A_P$  :: name list
and  $\Psi_P$  :: 'b
and  $C$  :: 'd::fs-name
and  $C'$  :: 'e::fs-name

```

```

assumes Trans:  $\Psi \triangleright P \longmapsto \alpha \prec P'$ 
and  $extractFrame P = \langle A_P, \Psi_P \rangle$ 
and  $distinct A_P$ 
and  $bn \alpha \#* subject \alpha$ 
and  $distinct(bn \alpha)$ 
and  $A_P \#* \alpha$ 
and  $A_P \#* P$ 
and  $A_P \#* C$ 
and  $A_P \#* C'$ 
and  $bn \alpha \#* P$ 
and  $bn \alpha \#* C'$ 

```

$\text{obtains } p \Psi' A_{P'} \Psi_{P'}$ 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

$\text{extractFrame } P' = \langle A_{P'}, \Psi_{P'} \rangle$ **and** $A_{P'} \#* P'$ **and** $A_{P'} \#*$
 α **and** $A_{P'} \#* (p \cdot \alpha)$ **and**
 $A_{P'} \#* C$ **and** $(bn(p \cdot \alpha)) \#* C'$ **and** $(bn(p \cdot \alpha)) \#* \alpha$ **and**
 $(bn(p \cdot \alpha)) \#* P'$ **and** $\text{distinct } A_{P'}$
 $\langle \text{proof} \rangle$

abbreviation

$\text{frameImpJudge } (\leftarrow \hookrightarrow_F \rightarrow [80, 80] 80)$
where $F \hookrightarrow_F G \equiv \text{FrameStatImp } F G$

lemma $\text{FrameStatEqImpCompose}:$
fixes $F :: 'b \text{ frame}$
and $G :: 'b \text{ frame}$
and $H :: 'b \text{ frame}$
and $I :: 'b \text{ frame}$

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

shows $F \hookrightarrow_F I$
 $\langle \text{proof} \rangle$

lemma $\text{transferNonTauFrame}:$
fixes $\Psi_F :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\alpha :: 'a \text{ action}$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $A_F :: \text{name list}$
and $A_G :: \text{name list}$
and $\Psi_G :: 'b$

assumes $\Psi_F \triangleright P \longmapsto \alpha \prec P'$
and $\text{extractFrame } P = \langle A_P, \Psi_P \rangle$
and $\text{distinct } A_P$
and $\text{distinct}(bn \alpha)$
and $\langle A_F, \Psi_F \otimes \Psi_P \rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \rangle$
and $A_F \#* P$
and $A_G \#* P$
and $A_F \#* \text{subject } \alpha$
and $A_G \#* \text{subject } \alpha$
and $A_P \#* A_F$
and $A_P \#* A_G$
and $A_P \#* \Psi_F$
and $A_P \#* \Psi_G$
and $\alpha \neq \tau$

shows $\Psi_G \triangleright P \longmapsto \alpha \prec P'$
 $\langle \text{proof} \rangle$

```

lemma transferTauFrame:
  fixes  $\Psi_F :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $P' :: ('a, 'b, 'c) \text{psi}$ 
  and  $A_F :: \text{name list}$ 
  and  $A_G :: \text{name list}$ 
  and  $\Psi_G :: 'b$ 

  assumes  $\Psi_F \triangleright P \xrightarrow{\tau} P'$ 
  and  $\text{extractFrame } P = \langle A_P, \Psi_P \rangle$ 
  and  $\text{distinct } A_P$ 
  and  $\langle A_F, \Psi_F \otimes \Psi_P \rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \rangle$ 
  and  $A_F \#* P$ 
  and  $A_G \#* P$ 
  and  $A_P \#* A_F$ 
  and  $A_P \#* A_G$ 
  and  $A_P \#* \Psi_F$ 
  and  $A_P \#* \Psi_G$ 

  shows  $\Psi_G \triangleright P \xrightarrow{\tau} P'$ 
   $\langle \text{proof} \rangle$ 

lemma transferFrame:
  fixes  $\Psi_F :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $\alpha :: 'a \text{ action}$ 
  and  $P' :: ('a, 'b, 'c) \text{psi}$ 
  and  $A_F :: \text{name list}$ 
  and  $A_G :: \text{name list}$ 
  and  $\Psi_G :: 'b$ 

  assumes  $\Psi_F \triangleright P \xrightarrow{\alpha} P'$ 
  and  $\text{extractFrame } P = \langle A_P, \Psi_P \rangle$ 
  and  $\text{distinct } A_P$ 
  and  $\langle A_F, \Psi_F \otimes \Psi_P \rangle \hookrightarrow_F \langle A_G, \Psi_G \otimes \Psi_P \rangle$ 
  and  $A_F \#* P$ 
  and  $A_G \#* P$ 
  and  $A_F \#* \text{subject } \alpha$ 
  and  $A_G \#* \text{subject } \alpha$ 
  and  $A_P \#* A_F$ 
  and  $A_P \#* A_G$ 
  and  $A_P \#* \Psi_F$ 
  and  $A_P \#* \Psi_G$ 

  shows  $\Psi_G \triangleright P \xrightarrow{\alpha} P'$ 
   $\langle \text{proof} \rangle$ 

```

lemma parCasesInputFrame[consumes 7, case-names cPar1 cPar2]:

```

fixes  $\Psi$  :: ' $b$ 
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:  $\Psi \triangleright P \parallel Q \xrightarrow{M(N)} \prec T$ 
and  $\text{extractFrame}(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ} \rangle$ 
and  $\text{distinct } A_{PQ}$ 
and  $A_{PQ} \#* \Psi$ 
and  $A_{PQ} \#* P$ 
and  $A_{PQ} \#* Q$ 
and  $A_{PQ} \#* M$ 
and  $rPar1: \bigwedge P' A_P \Psi_P A_Q \Psi_Q. [\Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(N)} \prec P'; \text{extractFrame}$ 
 $P = \langle A_P, \Psi_P \rangle; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$ 
 $\text{distinct } A_P; \text{distinct } A_Q; A_P \#* \Psi; A_P \#* P; A_P \#*$ 
 $Q; A_P \#* M; A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* M;$ 
 $A_P \#* \Psi_Q; A_Q \#* \Psi_P; A_P \#* A_Q; A_{PQ} = A_P @ A_Q;$ 
 $\Psi_{PQ} = \Psi_P \otimes \Psi_Q] \implies \text{Prop } (P' \parallel Q)$ 
and  $rPar2: \bigwedge Q' A_P \Psi_P A_Q \Psi_Q. [\Psi \otimes \Psi_P \triangleright Q \xrightarrow{M(N)} \prec Q'; \text{extractFrame}$ 
 $P = \langle A_P, \Psi_P \rangle; \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle;$ 
 $\text{distinct } A_P; \text{distinct } A_Q; A_P \#* \Psi; A_P \#* P; A_P \#*$ 
 $Q; A_P \#* M; A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* M;$ 
 $A_P \#* \Psi_Q; A_Q \#* \Psi_P; A_P \#* A_Q; A_{PQ} = A_P @ A_Q;$ 
 $\Psi_{PQ} = \Psi_P \otimes \Psi_Q] \implies \text{Prop } (P \parallel Q')$ 
shows Prop  $T$ 
(proof)

```

lemma *parCasesOutputFrame*[consumes 11, case-names *cPar1 cPar2*]:

```

fixes  $\Psi$  :: ' $b$ 
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:  $\Psi \triangleright P \parallel Q \xrightarrow{M(\nu*xvec)} \langle N \rangle \prec T$ 
and  $xvec \#* \Psi$ 
and  $xvec \#* P$ 
and  $xvec \#* Q$ 
and  $xvec \#* M$ 
and  $\text{extractFrame}(P \parallel Q) = \langle A_{PQ}, \Psi_{PQ} \rangle$ 
and  $\text{distinct } A_{PQ}$ 
and  $A_{PQ} \#* \Psi$ 

```

```

and       $A_{PQ} \#* P$ 
and       $A_{PQ} \#* Q$ 
and       $A_{PQ} \#* M$ 
and       $rPar1: \bigwedge P' A_P \Psi_P A_Q \Psi_Q. [\Psi \otimes \Psi_Q \triangleright P \longrightarrow M(\nu*xvec)\langle N \rangle \prec P';$ 
extractFrame  $P = \langle A_P, \Psi_P \rangle$ ; extractFrame  $Q = \langle A_Q, \Psi_Q \rangle$ ;
distinct  $A_P$ ; distinct  $A_Q$ ;  $A_P \#* \Psi$ ;  $A_P \#* P$ ;  $A_P \#*$ 
 $Q$ ;  $A_P \#* M$ ;  $A_Q \#* \Psi$ ;  $A_Q \#* P$ ;  $A_Q \#* Q$ ;  $A_Q \#* M$ ;
 $A_P \#* \Psi_Q$ ;  $A_Q \#* \Psi_P$ ;  $A_P \#* A_Q$ ;  $A_{PQ} = A_P @ A_Q$ ;
 $\Psi_{PQ} = \Psi_P \otimes \Psi_Q] \implies Prop(P' \parallel Q)$ 
and       $rPar2: \bigwedge Q' A_P \Psi_P A_Q \Psi_Q. [\Psi \otimes \Psi_P \triangleright Q \longrightarrow M(\nu*xvec)\langle N \rangle \prec Q';$ 
extractFrame  $P = \langle A_P, \Psi_P \rangle$ ; extractFrame  $Q = \langle A_Q, \Psi_Q \rangle$ ;
distinct  $A_P$ ; distinct  $A_Q$ ;  $A_P \#* \Psi$ ;  $A_P \#* P$ ;  $A_P \#*$ 
 $Q$ ;  $A_P \#* M$ ;  $A_Q \#* \Psi$ ;  $A_Q \#* P$ ;  $A_Q \#* Q$ ;  $A_Q \#* M$ ;
 $A_P \#* \Psi_Q$ ;  $A_Q \#* \Psi_P$ ;  $A_P \#* A_Q$ ;  $A_{PQ} = A_P @ A_Q$ ;
 $\Psi_{PQ} = \Psi_P \otimes \Psi_Q] \implies Prop(P \parallel Q')$ 
shows  $Prop T$ 
 $\langle proof \rangle$ 

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  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) psi$ 
  and  $Rs :: ('a, 'b, 'c) residual$ 
  and  $Prop :: 'd::fs-name \Rightarrow 'b \Rightarrow ('a, 'b, 'c) psi \Rightarrow ('a, 'b, 'c) residual \Rightarrow bool$ 
  and  $C :: 'd$ 

  assumes  $\Psi \triangleright !P \longrightarrow Rs$ 
  and  $rPar1: \bigwedge \alpha P' C. [\Psi \triangleright P \longrightarrow \alpha \prec P'; bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#*$ 
  subject  $\alpha$ ;  $bn \alpha \#* C$ ; distinct( $bn \alpha$ ) $] \implies Prop C \Psi (P \parallel !P) (\alpha \prec (P' \parallel !P))$ 
  and  $rPar2: \bigwedge \alpha P' C. [\Psi \triangleright !P \longrightarrow \alpha \prec P'; bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#*$ 
  subject  $\alpha$ ;  $bn \alpha \#* C$ ; distinct( $bn \alpha$ );
 $\bigwedge C. Prop C \Psi (!P) (\alpha \prec P')] \implies Prop C \Psi (P \parallel !P) (\alpha$ 
 $\prec (P \parallel P'))$ 
  and  $rComm1: \bigwedge M N P' K xvec P'' C. [\Psi \triangleright P \longrightarrow M(N) \prec P'; \Psi \triangleright !P$ 
 $\longrightarrow K(\nu*xvec)\langle N \rangle \prec P''; \bigwedge C. Prop C \Psi (!P) (K(\nu*xvec)\langle N \rangle \prec P''); \Psi \vdash M \leftrightarrow K;$ 
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* K;$ 
 $xvec \#* C$ ; distinct  $xvec$  $] \implies Prop C \Psi (P \parallel !P) (\tau \prec (\nu*xvec)(P' \parallel P''))$ 
  and  $rComm2: \bigwedge M xvec N P' K P'' C. [\Psi \triangleright P \longrightarrow M(\nu*xvec)\langle N \rangle \prec P'; \Psi$ 
 $\triangleright !P \longrightarrow K(\nu*xvec)\langle N \rangle \prec P''; \bigwedge C. Prop C \Psi (!P) (K(\nu*xvec)\langle N \rangle \prec P''); \Psi \vdash M \leftrightarrow K;$ 
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* K;$ 
 $xvec \#* C$ ; distinct  $xvec$  $] \implies Prop C \Psi (P \parallel !P) (\tau \prec (\nu*xvec)(P' \parallel P''))$ 
  and  $rBang: \bigwedge Rs C. [\Psi \triangleright P \parallel !P \longrightarrow Rs; \bigwedge C. Prop C \Psi (P \parallel !P) Rs;$ 
guarded  $P]$  $\implies Prop C \Psi (!P) Rs$ 
shows  $Prop C \Psi (!P) Rs$ 

```

$\langle proof \rangle$

```

lemma bangInputInduct[consumes 1, case-names cPar1 cPar2 cBang]:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $M :: 'a$ 
  and  $N :: 'a$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  and  $Prop :: 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow 'a \Rightarrow ('a, 'b, 'c) \psi \Rightarrow bool$ 

  assumes  $\Psi \triangleright !P \mapsto M(N) \prec P'$ 
  and  $rPar1: \bigwedge P'. \Psi \triangleright P \mapsto M(N) \prec P' \Rightarrow Prop \Psi (P \parallel !P) M N (P' \parallel !P)$ 
  and  $rPar2: \bigwedge P'. [\Psi \triangleright !P \mapsto M(N) \prec P'; Prop \Psi (!P) M N P'] \Rightarrow Prop \Psi (P \parallel !P) M N (P \parallel P')$ 
  and  $rBang: \bigwedge P'. [\Psi \triangleright P \parallel !P \mapsto M(N) \prec P'; Prop \Psi (P \parallel !P) M N P'; guarded P] \Rightarrow Prop \Psi (!P) M N P'$ 
  shows  $Prop \Psi (!P) M N P'$ 
 $\langle proof \rangle$ 

```

```

lemma bangOutputInduct[consumes 1, case-names cPar1 cPar2 cBang]:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $M :: 'a$ 
  and  $xvec :: name list$ 
  and  $N :: 'a$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  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  $\Psi \triangleright !P \mapsto M(\nu*xvec)(N) \prec P'$ 
  and  $rPar1: \bigwedge xvec N P' C. [\Psi \triangleright P \mapsto M(\nu*xvec)(N) \prec P'; xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* C; distinct xvec] \Rightarrow Prop C \Psi (P \parallel !P) M ((\nu*xvec)N \prec' (P' \parallel !P))$ 
  and  $rPar2: \bigwedge xvec N P' C. [\Psi \triangleright !P \mapsto M(\nu*xvec)(N) \prec P'; \bigwedge C. Prop C \Psi (!P) M ((\nu*xvec)N \prec' P'); xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* C; distinct xvec] \Rightarrow Prop C \Psi (P \parallel !P) M ((\nu*xvec)N \prec' (P \parallel P'))$ 
  and  $rBang: \bigwedge B C. [\Psi \triangleright P \parallel !P \mapsto (ROut M B); \bigwedge C. Prop C \Psi (P \parallel !P) M B; guarded P] \Rightarrow Prop C \Psi (!P) M B$ 

  shows  $Prop C \Psi (!P) M ((\nu*xvec)N \prec' P')$ 
 $\langle proof \rangle$ 

```

```

lemma bangTauInduct[consumes 1, case-names cPar1 cPar2 cComm1 cComm2 cBang]:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

```

```

and  $P' :: ('a, 'b, 'c) \psi$ 
and  $\text{Prop} :: 'd::\text{fs-name} \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow ('a, 'b, 'c) \psi \Rightarrow \text{bool}$ 
and  $C :: 'd$ 

assumes  $\Psi \triangleright !P \xrightarrow{\tau} \prec P'$ 
and  $rPar1: \bigwedge P' C. \Psi \triangleright P \xrightarrow{\tau} \prec P' \implies \text{Prop } C \Psi (P \parallel !P) (P' \parallel !P)$ 
and  $rPar2: \bigwedge P' C. [\Psi \triangleright !P \xrightarrow{\tau} \prec P'; \bigwedge C. \text{Prop } C \Psi (!P) P] \implies \text{Prop } C \Psi (P \parallel !P) (P \parallel P')$ 
and  $rComm1: \bigwedge M N P' K xvec P'' C. [\Psi \triangleright P \xrightarrow{M(N)} \prec P'; \Psi \triangleright !P \xrightarrow{K(\nu*xvec)(N)} \prec P''; \Psi \vdash M \leftrightarrow K;$ 
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* K;$ 
 $xvec \#* C] \implies \text{Prop } C \Psi (P \parallel !P) ((\nu*xvec)(P' \parallel P''))$ 
and  $rComm2: \bigwedge M N P' K xvec P'' C. [\Psi \triangleright P \xrightarrow{M(\nu*xvec)(N)} \prec P'; \Psi \triangleright !P \xrightarrow{K(N)} \prec P''; \Psi \vdash M \leftrightarrow K;$ 
 $xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* K;$ 
 $xvec \#* C] \implies \text{Prop } C \Psi (P \parallel !P) ((\nu*xvec)(P' \parallel P''))$ 
and  $rBang: \bigwedge P' C. [\Psi \triangleright P \parallel !P \xrightarrow{\tau} \prec P'; \bigwedge C. \text{Prop } C \Psi (P \parallel !P) P';$ 
 $\text{guarded } P] \implies \text{Prop } C \Psi (!P) P'$ 

shows  $\text{Prop } C \Psi (!P) P'$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{bangInduct}'[\text{consumes 2, case-names cAlpha cPar1 cPar2 cComm1 cComm2 cBang}]:$ 
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $\alpha :: 'a \text{ action}$ 
and  $P' :: ('a, 'b, 'c) \psi$ 
and  $\text{Prop} :: 'd::\text{fs-name} \Rightarrow 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow 'a \text{ action} \Rightarrow ('a, 'b, 'c) \psi$ 
 $\Rightarrow \text{bool}$ 
and  $C :: 'd::\text{fs-name}$ 

assumes  $\Psi \triangleright !P \xrightarrow{\alpha} \prec P'$ 
and  $bn \alpha \#* \text{subject } \alpha$ 
and  $rAlpha: \bigwedge \alpha P' p C. [bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* \text{subject } \alpha; bn \alpha \#*$ 
 $C;$ 
 $\text{set } p \subseteq \text{set}(bn \alpha) \times \text{set}(bn(p \cdot \alpha)); \text{distinctPerm } p;$ 
 $bn(p \cdot \alpha) \#* \alpha; bn(p \cdot \alpha) \#* P'; \text{Prop } C \Psi (P \parallel !P) \alpha$ 
 $P] \implies$ 
 $\text{Prop } C \Psi (P \parallel !P) (p \cdot \alpha) (p \cdot P')$ 
and  $rPar1: \bigwedge \alpha P' C.$ 
 $[\Psi \triangleright P \xrightarrow{\alpha} \prec P'; bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* \text{subject } \alpha; bn \alpha$ 
 $\#* C; \text{distinct}(bn \alpha)] \implies$ 
 $\text{Prop } C \Psi (P \parallel !P) \alpha (P' \parallel !P)$ 
and  $rPar2: \bigwedge \alpha P' C.$ 
 $[\Psi \triangleright !P \xrightarrow{\alpha} \prec P'; \bigwedge C. \text{Prop } C \Psi (!P) \alpha P';$ 
 $bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* \text{subject } \alpha; bn \alpha \#* C; \text{distinct}(bn$ 
 $\alpha)] \implies$ 
 $\text{Prop } C \Psi (P \parallel !P) \alpha (P \parallel P')$ 

```

and $rComm1: \bigwedge M N P' K xvec P'' C. [\Psi \triangleright P \xrightarrow{M(N)} \prec P'; \Psi \triangleright !P \xrightarrow{K(\nu*xvec)(N)} \prec P''; \bigwedge C. Prop C \Psi (!P) (K(\nu*xvec)(N)) P''; \Psi \vdash M \leftrightarrow K;$

$xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* C; distinct xvec] \implies Prop C \Psi (P \parallel !P) (\tau) ((\nu*xvec)(P' \parallel P''))$

and $rComm2: \bigwedge M xvec N P' K P'' C. [\Psi \triangleright P \xrightarrow{M(\nu*xvec)(N)} \prec P'; \Psi \triangleright !P \xrightarrow{K(N)} \prec P''; \bigwedge C. Prop C \Psi (!P) (K(N)) P''; \Psi \vdash M \leftrightarrow K;$

$xvec \#* \Psi; xvec \#* P; xvec \#* M; xvec \#* K;$

$xvec \#* C; distinct xvec] \implies Prop C \Psi (P \parallel !P) (\tau) ((\nu*xvec)(P' \parallel P''))$

and $rBang: \bigwedge \alpha P' C.$

$[\Psi \triangleright P \parallel !P \xrightarrow{\alpha} \prec P'; guarded P; \bigwedge C. Prop C \Psi (P \parallel !P) \alpha$

$P'; guarded P; distinct(bn \alpha)] \implies Prop C \Psi (!P) \alpha P'$

shows $Prop C \Psi (!P) \alpha P'$

$\langle proof \rangle$

lemma $comm1Aux:$

fixes $\Psi :: 'b$

and $\Psi_Q :: 'b$

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 $\Psi_R :: 'b$

and $P :: ('a, 'b, 'c) psi$

and $M :: 'a$

and $L :: 'a$

and $P' :: ('a, 'b, 'c) psi$

and $A_P :: name list$

and $\Psi_P :: 'b$

and $A_Q :: name list$

assumes $RTrans: \Psi \otimes \Psi_Q \triangleright R \xrightarrow{K(\nu*xvec)(N)} \prec R'$

and $FrR: extractFrame R = \langle A_R, \Psi_R \rangle$

and $PTrans: \Psi \otimes \Psi_R \triangleright P \xrightarrow{M(L)} \prec P'$

and $MeqK: \Psi \otimes \Psi_Q \otimes \Psi_R \vdash M \leftrightarrow K$

and $PeqQ: \langle A_Q, (\Psi \otimes \Psi_Q) \otimes \Psi_R \rangle \xrightarrow{F} \langle A_P, (\Psi \otimes \Psi_P) \otimes \Psi_R \rangle$

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 $A_R \#* A_P$

and $A_R \#* A_Q$

and $A_R \#* \Psi$

and $A_R \#* P$

and $A_R \#* Q$

and $A_R \#* R$

```

and  $A_R \#* K$ 
and  $A_P \#* \Psi$ 
and  $A_P \#* R$ 
and  $A_P \#* P$ 
and  $A_P \#* M$ 
and  $A_Q \#* R$ 
and  $A_Q \#* M$ 

```

obtains K' **where** $\Psi \otimes \Psi_P \triangleright R \longmapsto K'(\nu*xvec)\langle N \rangle \prec R'$ **and** $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$ **and** $A_R \#* K'$
(proof)

lemma *comm2Aux*:

```

fixes  $\Psi :: 'b$ 
and  $\Psi_Q :: 'b$ 
and  $R :: ('a, 'b, 'c) \text{ psi}$ 
and  $K :: 'a$ 
and  $N :: 'a$ 
and  $R' :: ('a, 'b, 'c) \text{ psi}$ 
and  $A_R :: \text{name list}$ 
and  $\Psi_R :: 'b$ 
and  $P :: ('a, 'b, 'c) \text{ psi}$ 
and  $M :: 'a$ 
and  $xvec :: \text{name list}$ 
and  $P' :: ('a, 'b, 'c) \text{ psi}$ 
and  $A_P :: \text{name list}$ 
and  $\Psi_P :: 'b$ 
and  $A_Q :: \text{name list}$ 

```

```

assumes  $RTrans: \Psi \otimes \Psi_Q \triangleright R \longmapsto K\langle N \rangle \prec R'$ 
and  $FrR: \text{extractFrame } R = \langle A_R, \Psi_R \rangle$ 
and  $PTrans: \Psi \otimes \Psi_R \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
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  $FrP: \text{extractFrame } P = \langle A_P, \Psi_P \rangle$ 
and  $FrQ: \text{extractFrame } Q = \langle A_Q, \Psi_Q \rangle$ 
and  $\text{distinct } A_P$ 
and  $\text{distinct } A_R$ 
and  $A_R \#* A_P$ 
and  $A_R \#* A_Q$ 
and  $A_R \#* \Psi$ 
and  $A_R \#* P$ 
and  $A_R \#* Q$ 
and  $A_R \#* R$ 
and  $A_R \#* K$ 
and  $A_P \#* \Psi$ 
and  $A_P \#* R$ 
and  $A_P \#* P$ 
and  $A_P \#* M$ 

```

```

and       $A_Q \#* R$ 
and       $A_Q \#* M$ 
and       $A_R \#* xvec$ 
and       $xvec \#* M$ 

obtains  $K'$  where  $\Psi \otimes \Psi_P \triangleright R \longmapsto K'(\|N\|) \prec R'$  and  $\Psi \otimes \Psi_P \otimes \Psi_R \vdash M \leftrightarrow K'$  and  $A_R \#* K'$ 
⟨proof⟩

end

end

theory Simulation
imports Semantics
begin

context env begin

definition
simulation :: 'b ⇒ ('a, 'b, 'c) psi ⇒
    ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set ⇒
    ('a, 'b, 'c) psi ⇒ bool (⟨- ∘ - ↪ [-] → [80, 80, 80, 80] 80)

where
 $\Psi \triangleright P \rightsquigarrow_{[Rel]} Q \equiv \forall \alpha. Q'. \Psi \triangleright Q \longmapsto^\alpha \prec Q' \longrightarrow bn \alpha \#* \Psi \longrightarrow bn \alpha \#* P \longrightarrow (\exists P'. \Psi \triangleright P \longmapsto^\alpha \prec P' \wedge (\Psi, P', Q') \in Rel)$ 

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  $\Psi :: 'b$ 
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  $C :: 'd::fs-name$ 

assumes Eqvt: eqvt Rel
and Sim:  $\bigwedge \alpha. Q'. [\Psi \triangleright Q \longmapsto^\alpha \prec Q'; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* \Psi;$ 
distinct(bn α);
 $bn \alpha \#* (\text{subject } \alpha); bn \alpha \#* C] \implies \exists P'. \Psi \triangleright P \longmapsto^\alpha \prec P'$ 
 $\wedge (\Psi, P', Q') \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow_{[Rel]} Q$ 
⟨proof⟩

lemma simI2[case-names cSim]:
fixes  $\Psi :: 'b$ 

```

```

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 Sim:  $\bigwedge \alpha Q'. [\Psi \triangleright Q \mapsto \alpha \prec Q'; bn \alpha \#* P; bn \alpha \#* \Psi; distinct(bn \alpha)] \implies \exists P'. \Psi \triangleright P \mapsto \alpha \prec P' \wedge (\Psi, P', Q') \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow [Rel] Q$ 
⟨proof⟩

```

```

lemma simICChainFresh[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 xvec :: name list
and C :: 'd::fs-name

assumes Eqvt: eqvt Rel
and xvec #* Ψ
and xvec #* P
and xvec #* Q
and Sim:  $\bigwedge \alpha Q'. [\Psi \triangleright Q \mapsto \alpha \prec Q'; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* \Psi;$ 
 $bn \alpha \#* subject \alpha; distinct(bn \alpha); bn \alpha \#* C; xvec \#* \alpha; xvec \#* Q] \implies$ 
 $\exists P'. \Psi \triangleright P \mapsto \alpha \prec P' \wedge (\Psi, P', Q') \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow [Rel] Q$ 
⟨proof⟩

```

```

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 #* Ψ
and x #* P
and x #* Q
and Sim:  $\bigwedge \alpha Q'. [\Psi \triangleright Q \mapsto \alpha \prec Q'; bn \alpha \#* P; bn \alpha \#* Q; bn \alpha \#* \Psi;$ 
 $bn \alpha \#* subject \alpha; distinct(bn \alpha); bn \alpha \#* C; x \#* \alpha; x \#* Q] \implies$ 
 $\exists P'. \Psi \triangleright P \mapsto \alpha \prec P' \wedge (\Psi, P', Q') \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow [Rel] Q$ 
⟨proof⟩

```

```

lemma simE:
  fixes F :: '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 Ψ ⊢ P ~>[Rel] Q

  shows ⋀α Q'. [Ψ ⊢ Q ↪α Q'; bn α #: Ψ; bn α #: P] ⇒ ∃P'. Ψ ⊢ P
  ↪α P' ∧ (Ψ, P', Q') ∈ Rel
  ⟨proof⟩

lemma simClosedAux:
  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 p :: name prm

  assumes EqvtRel: eqvt Rel
  and PSimQ: Ψ ⊢ P ~>[Rel] Q

  shows (p · Ψ) ⊢ (p · P) ~>[Rel] (p · Q)
  ⟨proof⟩

lemma simClosed:
  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 p :: name prm

  assumes EqvtRel: eqvt Rel

  shows Ψ ⊢ P ~>[Rel] Q ⇒ (p · Ψ) ⊢ (p · P) ~>[Rel] (p · Q)
  and P ~>[Rel] Q ⇒ (p · P) ~>[Rel] (p · Q)
  ⟨proof⟩

lemma reflexive:
  fixes Rel :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set
  and Ψ :: 'b
  and P :: ('a, 'b, 'c) psi

  assumes {(\Psi, P, P) | Ψ P. True} ⊆ Rel

  shows Ψ ⊢ P ~>[Rel] P
  ⟨proof⟩

lemma transitive:

```

```

fixes  $\Psi$  :: ' $b$ 
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  $R$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
and  $Rel''$  :: (' $b$  × (' $a$ , ' $b$ , ' $c$ )  $\psi$  × (' $a$ , ' $b$ , ' $c$ )  $\psi$ ) set

assumes  $PSimQ$ :  $\Psi \triangleright P \rightsquigarrow[Rel] Q$ 
and  $QSimR$ :  $\Psi \triangleright Q \rightsquigarrow[Rel'] R$ 
and  $Eqvt$ :  $Eqvt Rel''$ 
and  $Set$ :  $\{(\Psi, P, R) \mid \Psi \triangleright P \ R. \exists Q. (\Psi, P, Q) \in Rel \wedge (\Psi, Q, R) \in Rel'\} \subseteq Rel''$ 

shows  $\Psi \triangleright P \rightsquigarrow[Rel''] R$ 
⟨proof⟩

lemma  $statEqSim$ :
fixes  $\Psi$  :: ' $b$ 
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  $\Psi'$  :: ' $b$ 

assumes  $PSimQ$ :  $\Psi \triangleright P \rightsquigarrow[Rel] Q$ 
and  $Eqvt Rel'$ 
and  $\Psi \simeq \Psi'$ 
and  $C1$ :  $\bigwedge \Psi'' R S \Psi'''$ .  $\llbracket (\Psi'', R, S) \in Rel; \Psi'' \simeq \Psi''' \rrbracket \implies (\Psi''', R, S) \in Rel'$ 

shows  $\Psi' \triangleright P \rightsquigarrow[Rel'] Q$ 
⟨proof⟩

lemma  $monotonic$ :
fixes  $\Psi$  :: ' $b$ 
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  $\Psi \triangleright P \rightsquigarrow[A] Q$ 
and  $A \subseteq B$ 

shows  $\Psi \triangleright P \rightsquigarrow[B] Q$ 
⟨proof⟩

end

end

```

```

theory Tau-Chain
imports Semantics
begin

context env begin

abbreviation tauChain :: "'b ⇒ ('a, 'b, 'c) psi ⇒ ('a, 'b, 'c) psi ⇒ bool (‐▷ - 
⇒ ^τ → [80, 80, 80] 80)
where Ψ ▷ P ⇒ ^τ P' ≡ (P, P') ∈ {(P, P'). Ψ ▷ P ↳ τ < P'} ^*
abbreviation tauStepChain :: "'b ⇒ ('a, 'b, 'c) psi ⇒ ('a, 'b, 'c) psi ⇒ bool (‐▷ - 
- ⇒ ^τ → [80, 80, 80] 80)
where Ψ ▷ P ⇒ ^τ P' ≡ (P, P') ∈ {(P, P'). Ψ ▷ P ↳ τ < P'} ^
abbreviation tauContextChain :: ('a, 'b, 'c) psi ⇒ ('a, 'b, 'c) psi ⇒ bool (‐⇒ ^τ 
→ [80, 80] 80)
where P ⇒ ^τ P' ≡ 1 ▷ P ⇒ ^τ P'
abbreviation tauContextStepChain :: ('a, 'b, 'c) psi ⇒ ('a, 'b, 'c) psi ⇒ bool (‐⇒ ^τ 
→ [80, 80] 80)
where P ⇒ ^τ P' ≡ 1 ▷ P ⇒ ^τ P'

lemmas tauChainInduct[consumes 1, case-names TauBase TauStep] = rtrancl.induct[of 
- - {(P, P'). Ψ ▷ P ↳ τ < P'}, simplified] for Ψ
lemmas tauStepChainInduct[consumes 1, case-names TauBase TauStep] = trancl.induct[of 
- - {(P, P'). Ψ ▷ P ↳ τ < P'}, simplified] for Ψ

lemma tauActTauStepChain:
fixes Ψ :: 'b
and P :: ('a, 'b, 'c) psi
and P' :: ('a, 'b, 'c) psi

assumes Ψ ▷ P ↳ τ < P'

shows Ψ ▷ P ⇒ ^τ P'
⟨proof⟩

lemma tauActTauChain:
fixes Ψ :: 'b
and P :: ('a, 'b, 'c) psi
and P' :: ('a, 'b, 'c) psi

assumes Ψ ▷ P ↳ τ < P'

shows Ψ ▷ P ⇒ ^τ P'
⟨proof⟩

lemma tauStepChainEqvt[eqvt]:

```

```

fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 
and  $p :: name prm$ 

assumes  $\Psi \triangleright P \Rightarrow_{\tau} P'$ 

shows  $(p \cdot \Psi) \triangleright (p \cdot P) \Rightarrow_{\tau} (p \cdot P')$ 
⟨proof⟩

lemma tauChainEqvt[eqvt]:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 
and  $p :: name prm$ 

assumes  $\Psi \triangleright P \Rightarrow^{\hat{\tau}} P'$ 

shows  $(p \cdot \Psi) \triangleright (p \cdot P) \Rightarrow^{\hat{\tau}} (p \cdot P')$ 
⟨proof⟩

lemma tauStepChainEqvt'[eqvt]:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 
and  $p :: name prm$ 

shows  $(p \cdot (\Psi \triangleright P \Rightarrow_{\tau} P')) = (p \cdot \Psi) \triangleright (p \cdot P) \Rightarrow_{\tau} (p \cdot P')$ 
⟨proof⟩

lemma tauChainEqvt'[eqvt]:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 
and  $p :: name prm$ 

shows  $(p \cdot (\Psi \triangleright P \Rightarrow^{\hat{\tau}} P')) = (p \cdot \Psi) \triangleright (p \cdot P) \Rightarrow^{\hat{\tau}} (p \cdot P')$ 
⟨proof⟩

lemma tauStepChainFresh:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 
and  $x :: name$ 

assumes  $\Psi \triangleright P \Rightarrow_{\tau} P'$ 
and  $x \notin P$ 

shows  $x \notin P'$ 

```

$\langle proof \rangle$

```
lemma tauChainFresh:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  and  $x :: name$ 

  assumes  $\Psi \triangleright P \Rightarrow^{\hat{\tau}} P'$ 
  and  $x \notin P$ 
```

shows $x \notin P'$

$\langle proof \rangle$

```
lemma tauStepChainFreshChain:
```

```
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  and  $xvec :: name list$ 
```

```
  assumes  $\Psi \triangleright P \Rightarrow_{\tau} P'$ 
  and  $xvec \notin P$ 
```

shows $xvec \notin P'$

$\langle proof \rangle$

```
lemma tauChainFreshChain:
```

```
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  and  $xvec :: name list$ 
```

```
  assumes  $\Psi \triangleright P \Rightarrow^{\hat{\tau}} P'$ 
  and  $xvec \notin P$ 
```

shows $xvec \notin P'$

$\langle proof \rangle$

```
lemma tauStepChainCase:
```

```
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $P' :: ('a, 'b, 'c) \psi$ 
  and  $\varphi :: 'c$ 
  and  $Cs :: ('c \times ('a, 'b, 'c) \psi) list$ 
```

```
  assumes  $\Psi \triangleright P \Rightarrow_{\tau} P'$ 
  and  $(\varphi, P) \text{ mem } Cs$ 
  and  $\Psi \vdash \varphi$ 
  and  $\text{guarded } P$ 
```

shows $\Psi \triangleright (\text{Cases } Cs) \implies_{\tau} P'$
 $\langle proof \rangle$

lemma *tauStepChainResPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $x :: \text{name}$

assumes $\Psi \triangleright P \implies_{\tau} P'$
and $x \notin \Psi$

shows $\Psi \triangleright (\nu x)P \implies_{\tau} (\nu x)P'$
 $\langle proof \rangle$

lemma *tauChainResPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $x :: \text{name}$

assumes $\Psi \triangleright P \implies_{\tau} P'$
and $x \notin \Psi$

shows $\Psi \triangleright (\nu x)P \implies_{\tau} (\nu x)P'$
 $\langle proof \rangle$

lemma *tauStepChainResChainPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $xvec :: \text{name list}$

assumes $\Psi \triangleright P \implies_{\tau} P'$
and $xvec \notin \Psi$

shows $\Psi \triangleright (\nu * xvec)P \implies_{\tau} (\nu * xvec)P'$
 $\langle proof \rangle$

lemma *tauChainResChainPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $P' :: ('a, 'b, 'c) \text{ psi}$
and $xvec :: \text{name list}$

assumes $\Psi \triangleright P \implies_{\tau} P'$
and $xvec \notin \Psi$

shows $\Psi \triangleright (\nu*xvec)P \implies_{\tau} (\nu*xvec)P'$
 $\langle proof \rangle$

lemma *tauStepChainPar1*:

```

fixes  $\Psi :: 'b$ 
and  $\Psi_Q :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $P' :: ('a, 'b, 'c) psi$ 
and  $Q :: ('a, 'b, 'c) psi$ 
and  $A_Q :: name list$ 

assumes  $\Psi \otimes \Psi_Q \triangleright P \implies_{\tau} P'$ 
and  $extractFrame Q = \langle A_Q, \Psi_Q \rangle$ 
and  $A_Q \#* \Psi$ 
and  $A_Q \#* P$ 

shows  $\Psi \triangleright P \parallel Q \implies_{\tau} P' \parallel Q$ 
 $\langle proof \rangle$ 
```

lemma *tauChainPar1*:

```

fixes  $\Psi :: 'b$ 
and  $\Psi_Q :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $P' :: ('a, 'b, 'c) psi$ 
and  $Q :: ('a, 'b, 'c) psi$ 
and  $A_Q :: name list$ 

assumes  $\Psi \otimes \Psi_Q \triangleright P \implies_{\tau} P'$ 
and  $extractFrame Q = \langle A_Q, \Psi_Q \rangle$ 
and  $A_Q \#* \Psi$ 
and  $A_Q \#* P$ 

shows  $\Psi \triangleright P \parallel Q \implies_{\tau} P' \parallel Q$ 
 $\langle proof \rangle$ 
```

lemma *tauStepChainPar2*:

```

fixes  $\Psi :: 'b$ 
and  $\Psi_P :: 'b$ 
and  $Q :: ('a, 'b, 'c) psi$ 
and  $Q' :: ('a, 'b, 'c) psi$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $A_P :: name list$ 

assumes  $\Psi \otimes \Psi_P \triangleright Q \implies_{\tau} Q'$ 
and  $extractFrame P = \langle A_P, \Psi_P \rangle$ 
and  $A_P \#* \Psi$ 
and  $A_P \#* Q$ 

shows  $\Psi \triangleright P \parallel Q \implies_{\tau} P \parallel Q'$ 
```

$\langle proof \rangle$

```
lemma tauChainPar2:
  fixes  $\Psi$  :: ' $b$ 
  and  $\Psi_P$  :: ' $b$ 
  and  $Q$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $Q'$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $A_P$  :: name list

  assumes  $\Psi \otimes \Psi_P \triangleright Q \Rightarrow \hat{\tau} Q'$ 
  and   extractFrame  $P = \langle A_P, \Psi_P \rangle$ 
  and    $A_P \#* \Psi$ 
  and    $A_P \#* Q$ 

  shows  $\Psi \triangleright P \parallel Q \Rightarrow \hat{\tau} P \parallel Q'$ 
   $\langle proof \rangle$ 
```

```
lemma tauStepChainBang:
  fixes  $\Psi$  :: ' $b$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $P'$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 

  assumes  $\Psi \triangleright P \parallel !P \Rightarrow_{\tau} P'$ 
  and   guarded  $P$ 

  shows  $\Psi \triangleright !P \Rightarrow_{\tau} P'$ 
   $\langle proof \rangle$ 
```

```
lemma tauStepChainStatEq:
  fixes  $\Psi$  :: ' $b$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $P'$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $\Psi' :: 'b$ 

  assumes  $\Psi \triangleright P \Rightarrow_{\tau} P'$ 
  and    $\Psi \simeq \Psi'$ 

  shows  $\Psi' \triangleright P \Rightarrow_{\tau} P'$ 
   $\langle proof \rangle$ 
```

```
lemma tauChainStatEq:
  fixes  $\Psi$  :: ' $b$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $P'$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $\Psi' :: 'b$ 

  assumes  $\Psi \triangleright P \Rightarrow \hat{\tau} P'$ 
  and    $\Psi \simeq \Psi'$ 
```

shows $\Psi' \triangleright P \Rightarrow^{\hat{\tau}} P'$

$\langle proof \rangle$

definition $weakTransition :: 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow ('a, 'b, 'c) \psi \Rightarrow 'a action \Rightarrow ('a, 'b, 'c) \psi \Rightarrow bool (\dashleftarrow : - \triangleright - \Rightarrow - \prec - [80, 80, 80, 80, 80] 80)$

where

$\Psi : Q \triangleright P \Rightarrow \alpha \prec P' \equiv \exists P''. \Psi \triangleright P \Rightarrow^{\hat{\tau}} P'' \wedge (insertAssertion(extractFrame Q) \Psi) \hookrightarrow_F (insertAssertion(extractFrame P'') \Psi) \wedge \Psi \triangleright P'' \mapsto \alpha \prec P'$

lemma $weakTransitionI$:

fixes $\Psi :: 'b$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$
and $P'' :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a action$
and $P' :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \Rightarrow^{\hat{\tau}} P''$
and $insertAssertion(extractFrame Q) \Psi \hookrightarrow_F insertAssertion(extractFrame P'') \Psi$
and $\Psi \triangleright P'' \mapsto \alpha \prec P'$

shows $\Psi : Q \triangleright P \Rightarrow \alpha \prec P'$

$\langle proof \rangle$

lemma $weakTransitionE$:

fixes $\Psi :: 'b$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a action$
and $P' :: ('a, 'b, 'c) \psi$

assumes $\Psi : Q \triangleright P \Rightarrow \alpha \prec P'$

obtains P'' **where** $\Psi \triangleright P \Rightarrow^{\hat{\tau}} P''$ **and** $insertAssertion(extractFrame Q) \Psi \hookrightarrow_F insertAssertion(extractFrame P'') \Psi$

and $\Psi \triangleright P'' \mapsto \alpha \prec P'$

$\langle proof \rangle$

lemma $weakTransitionClosed[eqvt]$:

fixes $\Psi :: 'b$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a action$
and $P' :: ('a, 'b, 'c) \psi$
and $p :: name prm$

```

assumes  $\Psi : Q \triangleright P \implies \alpha \prec P'$ 

shows  $(p \cdot \Psi) : (p \cdot Q) \triangleright (p \cdot P) \implies (p \cdot \alpha) \prec (p \cdot P')$ 
⟨proof⟩

lemma weakOutputAlpha:
  fixes  $\Psi :: 'b$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $M :: 'a$ 
  and  $xvec :: \text{name list}$ 
  and  $N :: 'a$ 
  and  $P' :: ('a, 'b, 'c) \text{ psi}$ 
  and  $p :: \text{name prm}$ 
  and  $yvec :: \text{name list}$ 

assumes PTrans:  $\Psi : Q \triangleright P \implies M(\nu * (p \cdot xvec)) \langle (p \cdot N) \rangle \prec P'$ 
and  $S : \text{set } p \subseteq \text{set } xvec \times \text{set}(p \cdot xvec)$ 
and  $\text{distinctPerm } p$ 
and  $xvec \#* P$ 
and  $xvec \#* (p \cdot xvec)$ 
and  $(p \cdot xvec) \#* M$ 
and  $\text{distinct } xvec$ 

shows  $\Psi : Q \triangleright P \implies M(\nu * xvec) \langle N \rangle \prec (p \cdot P')$ 
⟨proof⟩

lemma weakFreshDerivative:
  fixes  $\Psi :: 'b$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\alpha :: 'a \text{ action}$ 
  and  $P' :: ('a, 'b, 'c) \text{ psi}$ 
  and  $x :: \text{name}$ 

  assumes PTrans:  $\Psi : Q \triangleright P \implies \alpha \prec P'$ 
  and  $x \# P$ 
  and  $x \# \alpha$ 
  and  $\text{bn } \alpha \#* \text{subject } \alpha$ 
  and  $\text{distinct}(\text{bn } \alpha)$ 

shows  $x \# P'$ 
⟨proof⟩

lemma weakFreshChainDerivative:
  fixes  $\Psi :: 'b$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\alpha :: 'a \text{ action}$ 
```

```

and    $P' :: ('a, 'b, 'c) \psi$ 
and    $yvec :: name list$ 

assumes  $PTrans: \Psi : Q \triangleright P \implies_{\alpha} \prec P'$ 
and    $yvec \#* P$ 
and    $yvec \#* \alpha$ 
and    $bn \alpha \#* subject \alpha$ 
and    $distinct(bn \alpha)$ 

shows  $yvec \#* P'$ 
⟨proof⟩

lemma weakInputFreshDerivative:
fixes  $\Psi :: 'b$ 
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  $PTrans: \Psi : Q \triangleright P \implies M(N) \prec P'$ 
and    $x \notin P$ 
and    $x \notin N$ 

shows  $x \notin P'$ 
⟨proof⟩

lemma weakInputFreshChainDerivative:
fixes  $\Psi :: 'b$ 
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  $PTrans: \Psi : Q \triangleright P \implies M(N) \prec P'$ 
and    $xvec \#* P$ 
and    $xvec \#* N$ 

shows  $xvec \#* P'$ 
⟨proof⟩

lemma weakOutputFreshDerivative:
fixes  $\Psi :: 'b$ 
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:  $\Psi : Q \triangleright P \implies M(\nu*xvec)\langle N \rangle \prec P'$ 
and   x  $\notin$  P
and   x  $\notin$  xvec
and   xvec  $\notin$  M
and   distinct xvec

shows x  $\notin$  N
and   x  $\notin$  P'
(proof)

lemma weakOutputFreshChainDerivative:
  fixes  $\Psi$  :: '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:  $\Psi : Q \triangleright P \implies M(\nu*xvec)\langle N \rangle \prec P'$ 
  and   yvec  $\notin$  P
  and   xvec  $\notin$  yvec
  and   xvec  $\notin$  M
  and   distinct xvec

  shows yvec  $\notin$  N
  and   yvec  $\notin$  P'
(proof)

lemma weakOutputPermSubject:
  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
  and   p    :: name prm
  and   yvec :: name list
  and   zvec :: name list

  assumes PTrans:  $\Psi : Q \triangleright P \implies M(\nu*xvec)\langle N \rangle \prec P'$ 
  and   S: set p  $\subseteq$  set yvec  $\times$  set zvec
  and   yvec  $\notin$   $\Psi$ 

```

```

and      zvec #*  $\Psi$ 
and      yvec #*  $P$ 
and      zvec #*  $P$ 

shows  $\Psi : Q \triangleright P \implies (p \cdot M)(\nu*xvec)\langle N \rangle \prec P'$ 
(proof)

lemma weakInputPermSubject:
fixes  $\Psi :: 'b$ 
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       $p :: name \text{ prm}$ 
and      yvec :: name list
and      zvec :: name list

assumes PTrans:  $\Psi : Q \triangleright P \implies M\langle N \rangle \prec P'$ 
and       $S : set p \subseteq set yvec \times set zvec$ 
and      yvec #*  $\Psi$ 
and      zvec #*  $\Psi$ 
and      yvec #*  $P$ 
and      zvec #*  $P$ 

shows  $\Psi : Q \triangleright P \implies (p \cdot M)\langle N \rangle \prec P'$ 
(proof)

lemma weakInput:
fixes  $\Psi :: 'b$ 
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       $Qeq\Psi : insertAssertion (extractFrame Q) \Psi \hookrightarrow_F \langle \varepsilon, \Psi \otimes \mathbf{1} \rangle$ 

shows  $\Psi : Q \triangleright M(\lambda*xvec N).P \implies K\langle (N[xvec:=Tvec]) \rangle \prec P[xvec:=Tvec]$ 
(proof)

lemma weakOutput:
fixes  $\Psi :: 'b$ 

```

```

and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $M :: 'a$ 
and    $K :: 'a$ 
and    $N :: 'a$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\Psi \vdash M \leftrightarrow K$ 
and      $Q \in \Psi : \text{insertAssertion}(\text{extractFrame } Q) \Psi \hookrightarrow_F \langle \varepsilon, \Psi \otimes \mathbf{1} \rangle$ 

shows  $\Psi : Q \triangleright M\langle N \rangle.P \implies K\langle N \rangle \prec P$ 
         $\langle \text{proof} \rangle$ 

lemma insertGuardedAssertion:
fixes  $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes guarded P

shows  $\text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \langle \varepsilon, \Psi \otimes \mathbf{1} \rangle$ 
         $\langle \text{proof} \rangle$ 

lemma weakCase:
fixes  $\Psi :: 'b$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $\alpha :: 'a \text{ action}$ 
and    $P' :: ('a, 'b, 'c) \text{ psi}$ 
and    $R :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $P\text{Trans} : \Psi : Q \triangleright P \implies \alpha \prec P'$ 
and      $(\varphi, P) \in \text{mem } CsP$ 
and      $\Psi \vdash \varphi$ 
and     guarded P
and      $R \in \text{Imp } Q : \text{insertAssertion}(\text{extractFrame } R) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } Q) \Psi$ 
and      $R \in \text{Imp } R : \text{insertAssertion}(\text{extractFrame } R) \Psi \hookrightarrow_F \langle \varepsilon, \Psi \rangle$ 

shows  $\Psi : R \triangleright \text{Cases } CsP \implies \alpha \prec P'$ 
         $\langle \text{proof} \rangle$ 

lemma weakOpen:
fixes  $\Psi :: 'b$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $M :: 'a$ 
and    $xvec :: \text{name list}$ 
and    $yvec :: \text{name list}$ 
and    $N :: 'a$ 
and    $P' :: ('a, 'b, 'c) \text{ psi}$ 

```

assumes $PTrans: \Psi : Q \triangleright P \implies M(\nu*(xvec@yvec))\langle N \rangle \prec P'$

and $x \in supp N$

and $x \notin \Psi$

and $x \notin M$

and $x \notin xvec$

and $x \notin yvec$

shows $\Psi : (\nu x)Q \triangleright (\nu x)P \implies M(\nu*(xvec@x#yvec))\langle N \rangle \prec P'$
 $\langle proof \rangle$

lemma *weakScope*:

fixes $\Psi :: 'b$

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

and $P :: ('a, 'b, 'c) psi$

and $\alpha :: 'a action$

and $P' :: ('a, 'b, 'c) psi$

assumes $PTrans: \Psi : Q \triangleright P \implies \alpha \prec P'$

and $x \notin \Psi$

and $x \notin \alpha$

shows $\Psi : (\nu x)Q \triangleright (\nu x)P \implies \alpha \prec (\nu x)P'$

$\langle proof \rangle$

lemma *weakPar1*:

fixes $\Psi :: 'b$

and $R :: ('a, 'b, 'c) psi$

and $P :: ('a, 'b, 'c) psi$

and $\alpha :: 'a action$

and $P' :: ('a, 'b, 'c) psi$

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

and $A_Q :: name list$

and $\Psi_Q :: 'b$

assumes $PTrans: \Psi \otimes \Psi_Q : R \triangleright P \implies \alpha \prec P'$

and $FrQ: extractFrame Q = \langle A_Q, \Psi_Q \rangle$

and $bn \alpha \#* Q$

and $A_Q \#* \Psi$

and $A_Q \#* P$

and $A_Q \#* \alpha$

and $A_Q \#* R$

shows $\Psi : R \parallel Q \triangleright P \parallel Q \implies \alpha \prec P' \parallel Q$

$\langle proof \rangle$

lemma *weakPar2*:

fixes $\Psi :: 'b$

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:  $\Psi \otimes \Psi_P : R \triangleright Q \xrightarrow{\alpha} \prec Q'$ 
and FrP: extractFrame P = ⟨A_P, Ψ_P⟩
and bn α #* P
and A_P #* Ψ
and A_P #* Q
and A_P #* α
and A_P #* R

```

shows $\Psi : P \parallel R \triangleright P \parallel Q \xrightarrow{\alpha} \prec P \parallel Q'$
(proof)

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:  $\Psi \otimes \Psi_Q : R \triangleright P \xrightarrow{M(N)} \prec P'$ 
and FrR: extractFrame R = ⟨A_R, Ψ_R⟩
and QTrans:  $\Psi \otimes \Psi_R \triangleright Q \xrightarrow{K(\nu*xvec)} \langle N \rangle \prec Q'$ 
and FrQ: extractFrame Q = ⟨A_Q, Ψ_Q⟩
and MeqK:  $\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$ 
and A_R #* Ψ
and A_R #* P
and A_R #* Q
and A_R #* R
and A_R #* M
and A_R #* A_Q
and A_Q #* Ψ
and A_Q #* P
and A_Q #* Q
and A_Q #* R
and A_Q #* K
and xvec #* P

```

shows $\Psi \triangleright P \parallel Q \xrightarrow{\tau} (\langle \nu*xvec \rangle (P' \parallel Q'))$
(proof)

```

lemma weakComm2:
  fixes  $\Psi$  :: 'b
  and  $R$  :: ('a, 'b, 'c) psi
  and  $P$  :: ('a, 'b, 'c) psi
  and  $\alpha$  :: 'a action
  and  $P'$  :: ('a, 'b, 'c) psi
  and  $Q$  :: ('a, 'b, 'c) psi
  and  $A_Q$  :: name list
  and  $\Psi_Q$  :: 'b

  assumes PTrans:  $\Psi \otimes \Psi_Q : R \triangleright P \implies M(\nu*xvec)(N) \prec P'$ 
  and FrR: extractFrame R =  $\langle A_R, \Psi_R \rangle$ 
  and QTrans:  $\Psi \otimes \Psi_R \triangleright Q \longmapsto K(N) \prec Q'$ 
  and FrQ: extractFrame Q =  $\langle A_Q, \Psi_Q \rangle$ 
  and MeqK:  $\Psi \otimes \Psi_R \otimes \Psi_Q \vdash M \leftrightarrow K$ 
  and  $A_R \#* \Psi$ 
  and  $A_R \#* P$ 
  and  $A_R \#* Q$ 
  and  $A_R \#* R$ 
  and  $A_R \#* M$ 
  and  $A_R \#* A_Q$ 
  and  $A_Q \#* \Psi$ 
  and  $A_Q \#* P$ 
  and  $A_Q \#* Q$ 
  and  $A_Q \#* R$ 
  and  $A_Q \#* K$ 
  and  $xvec \#* Q$ 
  and  $xvec \#* M$ 
  and  $xvec \#* A_Q$ 
  and  $xvec \#* A_R$ 

  shows  $\Psi \triangleright P \parallel Q \implies_{\tau} ((\nu*xvec)(P' \parallel Q'))$ 
  (proof)

lemma frameImpIntComposition:
  fixes  $\Psi$  :: 'b
  and  $\Psi'$  :: 'b
  and  $A_F$  :: name list
  and  $\Psi_F$  :: '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)

lemma insertAssertionStatImp:
  fixes  $F$  :: 'b frame
  and  $\Psi$  :: 'b

```

```

and    $G :: 'b \text{ frame}$ 
and    $\Psi' :: 'b$ 

assumes  $\text{FeqG: } \text{insertAssertion } F \Psi \hookrightarrow_F \text{insertAssertion } G \Psi$ 
and    $\Psi \simeq \Psi'$ 

shows  $\text{insertAssertion } F \Psi' \hookleftarrow_F \text{insertAssertion } G \Psi'$ 
(proof)

lemma  $\text{insertAssertionStatEq:}$ 
fixes  $F :: 'b \text{ frame}$ 
and    $\Psi :: 'b$ 
and    $G :: 'b \text{ frame}$ 
and    $\Psi' :: 'b$ 

assumes  $\text{FeqG: } \text{insertAssertion } F \Psi \simeq_F \text{insertAssertion } G \Psi$ 
and    $\Psi \simeq \Psi'$ 

shows  $\text{insertAssertion } F \Psi' \simeq_F \text{insertAssertion } G \Psi'$ 
(proof)

lemma  $\text{weakTransitionStatEq:}$ 
fixes  $\Psi :: 'b$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $\alpha :: 'a \text{ action}$ 
and    $P' :: ('a, 'b, 'c) \text{ psi}$ 
and    $\Psi' :: 'b$ 

assumes  $\text{PTrans: } \Psi : Q \triangleright P \implies \alpha \prec P'$ 
and    $\Psi \simeq \Psi'$ 

shows  $\Psi' : Q \triangleright P \implies \alpha \prec P'$ 
(proof)

lemma  $\text{transitionWeakTransition:}$ 
fixes  $\Psi :: 'b$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $\alpha :: 'a \text{ action}$ 
and    $P' :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\Psi \triangleright P \mapsto \alpha \prec P'$ 
and    $\text{insertAssertion}(\text{extractFrame } Q) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } P)$ 
and    $\Psi$ 

shows  $\Psi : Q \triangleright P \implies \alpha \prec P'$ 
(proof)

```

```

lemma weakPar1Guarded:
  fixes  $\Psi :: 'b$ 
  and  $R :: ('a, 'b, 'c) \text{psi}$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $\alpha :: 'a \text{ action}$ 
  and  $P' :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 

  assumes  $PTrans: \Psi : R \triangleright P \Rightarrow \alpha \prec P'$ 
  and  $bn \alpha \sharp* Q$ 
  and  $guarded Q$ 

  shows  $\Psi : (R \parallel Q) \triangleright P \parallel Q \Rightarrow \alpha \prec P' \parallel Q$ 
   $\langle proof \rangle$ 

lemma weakBang:
  fixes  $\Psi :: 'b$ 
  and  $R :: ('a, 'b, 'c) \text{psi}$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $\alpha :: 'a \text{ action}$ 
  and  $P' :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 

  assumes  $PTrans: \Psi : R \triangleright P \parallel !P \Rightarrow \alpha \prec P'$ 
  and  $guarded P$ 

  shows  $\Psi : R \triangleright !P \Rightarrow \alpha \prec P'$ 
   $\langle proof \rangle$ 

lemma weakTransitionFrameImp:
  fixes  $\Psi :: 'b$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $M :: 'a$ 
  and  $xvec :: name list$ 
  and  $N :: 'a$ 
  and  $P' :: ('a, 'b, 'c) \text{psi}$ 
  and  $R :: ('a, 'b, 'c) \text{psi}$ 

  assumes  $PTrans: \Psi : Q \triangleright P \Rightarrow \alpha \prec P'$ 
  and  $insertAssertion(extractFrame R) \Psi \hookrightarrow_F insertAssertion(extractFrame Q) \Psi$ 

  shows  $\Psi : R \triangleright P \Rightarrow \alpha \prec P'$ 
   $\langle proof \rangle$ 

lemma guardedFrameStatEq:
  fixes  $P :: ('a, 'b, 'c) \text{psi}$ 

```

```

assumes guarded  $P$ 

shows extractFrame  $P \simeq_F \langle \varepsilon, \mathbf{1} \rangle$ 
⟨proof⟩

lemma weakGuardedTransition:
  fixes  $\Psi :: 'b$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\alpha :: 'a \text{ action}$ 
  and  $P' :: ('a, 'b, 'c) \text{ psi}$ 
  and  $R :: ('a, 'b, 'c) \text{ psi}$ 

  assumes  $PTrans: \Psi : Q \triangleright P \Longrightarrow^\alpha \prec P'$ 
  and guarded  $Q$ 

  shows  $\Psi : \mathbf{0} \triangleright P \Longrightarrow^\alpha \prec P'$ 
⟨proof⟩

lemma expandTauChainFrame:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $P' :: ('a, 'b, 'c) \text{ psi}$ 
  and  $A_P :: \text{name list}$ 
  and  $\Psi_P :: 'b$ 
  and  $C :: 'd::fs-name$ 

  assumes  $PChain: \Psi \triangleright P \Longrightarrow^\tau P'$ 
  and  $FrP: \text{extractFrame } P = \langle A_P, \Psi_P \rangle$ 
  and distinct  $A_P$ 
  and  $A_P \nparallel P$ 
  and  $A_P \nparallel C$ 

  obtains  $\Psi' A_P' \Psi_P'$  where  $\text{extractFrame } P' = \langle A_P', \Psi_P' \rangle$  and  $\Psi_P \otimes \Psi' \simeq \Psi_P'$ 
  and  $A_P' \nparallel P'$  and  $A_P' \nparallel C$  and distinct  $A_P'$ 
⟨proof⟩

lemma frameIntImpComposition:
  fixes  $\Psi :: 'b$ 
  and  $\Psi' :: 'b$ 
  and  $A_F :: \text{name list}$ 
  and  $\Psi_F :: 'b$ 

  assumes  $\Psi \simeq \Psi'$ 

  shows  $\langle A_F, \Psi \otimes \Psi_F \rangle \hookrightarrow_F \langle A_F, \Psi' \otimes \Psi_F \rangle$ 
⟨proof⟩

lemma tauChainInduct2[consumes 1, case-names TauBase TauStep]:

```

```

fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes  $PChain: \Psi \triangleright P \Rightarrow_{\tau}^{\hat{}} P'$ 
and  $cBase: \bigwedge P. Prop P P$ 
and  $cStep: \bigwedge P P' P''. [\Psi \triangleright P' \xrightarrow{\tau} \prec P''; \Psi \triangleright P \Rightarrow_{\tau}^{\hat{}} P'; Prop P P''] \Rightarrow Prop P P''$ 

shows  $Prop P P'$ 
⟨proof⟩

lemma  $\tauStepChainInduct2$ [consumes 1, case-names  $TauBase$   $TauStep$ ]:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes  $PChain: \Psi \triangleright P \Rightarrow_{\tau} P'$ 
and  $cBase: \bigwedge P P'. \Psi \triangleright P \xrightarrow{\tau} \prec P' \Rightarrow Prop P P'$ 
and  $cStep: \bigwedge P P' P''. [\Psi \triangleright P' \xrightarrow{\tau} \prec P''; \Psi \triangleright P \Rightarrow_{\tau} P'; Prop P P''] \Rightarrow Prop P P''$ 

shows  $Prop P P'$ 
⟨proof⟩

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  $PChain: \Psi_F \triangleright P \Rightarrow_{\tau}^{\hat{}} P'$ 
and  $FrP: extractFrame P = \langle A_P, \Psi_P \rangle$ 
and  $distinct A_P$ 
and  $FeeqG: \bigwedge \Psi. insertAssertion (\langle A_F, \Psi_F \otimes \Psi_P \rangle) \Psi \hookrightarrow_F insertAssertion (\langle A_G, \Psi_G \otimes \Psi_P \rangle) \Psi$ 
and  $A_F \#* \Psi_G$ 
and  $A_G \#* \Psi$ 
and  $A_G \#* \Psi_F$ 
and  $A_F \#* A_G$ 
and  $A_F \#* P$ 
and  $A_G \#* P$ 
and  $A_P \#* A_F$ 
and  $A_P \#* A_G$ 
and  $A_P \#* \Psi_F$ 

```

```

and       $A_P \#* \Psi_G$ 
and       $A_P \#* P$ 

shows  $\Psi_G \triangleright P \implies_{\tau}^{\hat{}} P'$ 
(proof)

coinductive quiet :: ('a, 'b, 'c) psi  $\Rightarrow$  bool
where  $\llbracket \forall \Psi. (\text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \langle \varepsilon, \Psi \rangle \wedge$ 
        $(\forall Rs. \Psi \triangleright P \mapsto Rs \longrightarrow (\exists P'. Rs = \tau \prec P' \wedge \text{quiet } P')) \rrbracket \implies \text{quiet}$ 
        $P$ 

lemma quietFrame:
fixes  $\Psi :: 'b$ 
and     $P :: ('a, 'b, 'c) \text{psi}$ 

assumes quiet  $P$ 

shows insertAssertion(extractFrame  $P$ )  $\Psi \simeq_F \langle \varepsilon, \Psi \rangle$ 
(proof)

lemma quietTransition:
fixes  $\Psi :: 'b$ 
and     $P :: ('a, 'b, 'c) \text{psi}$ 
and     $Rs :: ('a, 'b, 'c) \text{residual}$ 

assumes quiet  $P$ 
and     $\Psi \triangleright P \mapsto Rs$ 

obtains  $P'$  where  $Rs = \tau \prec P'$  and quiet  $P'$ 
(proof)

lemma quietEqvt:
fixes  $P :: ('a, 'b, 'c) \text{psi}$ 
and     $p :: \text{name} \text{prm}$ 

assumes quiet  $P$ 

shows quiet( $p \cdot P$ )
(proof)

lemma quietOutput:
fixes  $\Psi :: 'b$ 
and     $P :: ('a, 'b, 'c) \text{psi}$ 
and     $M :: 'a$ 
and     $xvec :: \text{name list}$ 
and     $N :: 'a$ 
and     $P' :: ('a, 'b, 'c) \text{psi}$ 

```

```

assumes  $\Psi \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P'$ 
and      quiet  $P$ 

shows False
⟨proof⟩

lemma quietInput:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $M :: 'a$ 
and    $N :: 'a$ 
and    $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \longmapsto M\langle N \rangle \prec P'$ 
and      quiet  $P$ 

shows False
⟨proof⟩

lemma quietTau:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \longmapsto \tau \prec P'$ 
and      insertAssertion (extractFrame  $P$ )  $\Psi \simeq_F \langle \varepsilon, \Psi \rangle$ 
and      quiet  $P$ 

shows quiet  $P'$ 
⟨proof⟩

lemma tauChainCases[consumes 1, case-names TauBase TauStep]:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \Rightarrow \hat{\tau} P'$ 
and       $P = P' \Rightarrow \text{Prop}$ 
and       $\Psi \triangleright P \Rightarrow \tau P' \Rightarrow \text{Prop}$ 

shows Prop
⟨proof⟩

end

theory Weak-Simulation
imports Simulation Tau-Chain

```

begin

context env **begin**

definition

$weakSimulation :: 'b \Rightarrow ('a, 'b, 'c) psi \Rightarrow$
 $('b \times ('a, 'b, 'c) psi \times ('a, 'b, 'c) psi) set \Rightarrow$
 $('a, 'b, 'c) psi \Rightarrow bool (\langle - \rangle \rightarrow [80, 80, 80] 80)$

where

$\Psi \triangleright P \rightsquigarrow_{Rel} Q \equiv (\forall \Psi' \alpha Q'. \Psi \triangleright Q \mapsto \alpha \prec Q' \rightarrow bn \alpha \sharp* \Psi \rightarrow bn \alpha \sharp* P \rightarrow \alpha \neq \tau \rightarrow$
 $(\exists P''. \Psi : Q \triangleright P \Rightarrow \alpha \prec P'' \wedge (\exists P'. \Psi \otimes \Psi' \triangleright P'' \Rightarrow \hat{\tau} P'$
 $\wedge (\Psi \otimes \Psi', P', Q') \in Rel)) \wedge$
 $(\forall Q'. \Psi \triangleright Q \mapsto \tau \prec Q' \rightarrow (\exists P'. \Psi \triangleright P \Rightarrow \hat{\tau} P' \wedge (\Psi, P', Q') \in Rel))$

abbreviation

$weakSimulationNilJudge (\langle - \rangle \rightarrow [80, 80, 80] 80)$ **where** $P \rightsquigarrow_{Rel} Q \equiv SBottom' \triangleright P \rightsquigarrow_{Rel} Q$

lemma $weakSimI[consumes 1, case-names cAct cTau]$:

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 $C :: 'd::fs-name$

assumes $Eqvt: eqvt Rel$

and $rAct: \bigwedge \Psi' \alpha Q'. [\Psi \triangleright Q \mapsto \alpha \prec Q'; bn \alpha \sharp* \Psi; bn \alpha \sharp* P; bn \alpha \sharp* Q;$
 $bn \alpha \sharp* subject \alpha; bn \alpha \sharp* C; distinct(bn \alpha); \alpha \neq \tau] \Rightarrow$

$\exists P''. \Psi : Q \triangleright P \Rightarrow \alpha \prec P'' \wedge (\exists P'. \Psi \otimes \Psi' \triangleright P'' \Rightarrow \hat{\tau}$
 $P' \wedge (\Psi \otimes \Psi', P', Q') \in Rel)$
and $rTau: \bigwedge Q'. \Psi \triangleright Q \mapsto \tau \prec Q' \Rightarrow \exists P'. \Psi \triangleright P \Rightarrow \hat{\tau} P' \wedge (\Psi, P', Q')$
 $\in Rel$

shows $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
 $\langle proof \rangle$

lemma $weakSimI2[case-names cAct cTau]$:

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 $C :: 'd::fs-name$

assumes $rOutput: \bigwedge \Psi' \alpha Q'. [\Psi \triangleright Q \mapsto \alpha \prec Q'; bn \alpha \sharp* \Psi; bn \alpha \sharp* P; \alpha \neq \tau] \Rightarrow$
 $\exists P''. \Psi : Q \triangleright P \Rightarrow \alpha \prec P'' \wedge (\exists P'. \Psi \otimes \Psi' \triangleright P'' \Rightarrow \hat{\tau} P' \wedge (\Psi, P', Q') \in Rel)$

$\implies \hat{\tau} P' \wedge (\Psi \otimes \Psi', P', Q') \in Rel$
and $rTau: \bigwedge Q'. \Psi \triangleright Q \xrightarrow{\tau} Q' \implies \exists P'. \Psi \triangleright P \implies \hat{\tau} P' \wedge (\Psi, P', Q') \in Rel$

shows $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
 $\langle proof \rangle$

lemma *weakSimIChainFresh[consumes 4, case-names cOutput cInput]*:

```

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  $yvec :: name list$ 
and  $C :: 'd::fs-name$ 

assumes  $Eqvt: eqvt Rel$ 
and  $yvec \#* \Psi$ 
and  $yvec \#* P$ 
and  $yvec \#* Q$ 
and  $rAct: \bigwedge \Psi' \alpha Q'. [\Psi \triangleright Q \xrightarrow{\alpha} Q'; bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* Q;$ 
 $\alpha \neq \tau;$ 
 $bn \alpha \#* subject \alpha; bn \alpha \#* C; yvec \#* \alpha; yvec \#* Q'; yvec$ 
 $\#* \Psi'] \implies$ 
 $\exists P''. \Psi : Q \triangleright P \xrightarrow{\alpha} Q'' \wedge (\exists P'. \Psi \otimes \Psi' \triangleright P'' \implies \hat{\tau}$ 
 $P' \wedge (\Psi \otimes \Psi', P', Q') \in Rel)$ 
and  $rTau: \bigwedge Q'. [\Psi \triangleright Q \xrightarrow{\tau} Q'; yvec \#* Q'] \implies \exists P'. \Psi \triangleright P \implies \hat{\tau} P'$ 
 $\wedge (\Psi, P', Q') \in Rel$ 

```

shows $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
 $\langle proof \rangle$

lemma *weakSimIFresh[consumes 4, case-names cAct cTau]*:

```

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  $x :: name$ 
and  $C :: 'd::fs-name$ 

assumes  $Eqvt: eqvt Rel$ 
and  $x \# \Psi$ 
and  $x \# P$ 
and  $x \# Q$ 
and  $\bigwedge \alpha Q' \Psi'. [\Psi \triangleright Q \xrightarrow{\alpha} Q'; bn \alpha \#* \Psi; bn \alpha \#* P; bn \alpha \#* Q; \alpha \neq \tau;$ 
 $bn \alpha \#* subject \alpha; bn \alpha \#* C; x \# \alpha; x \# Q'; x \# \Psi'] \implies$ 
 $\exists P''. \Psi : Q \triangleright P \xrightarrow{\alpha} P'' \wedge (\exists P'. \Psi \otimes \Psi' \triangleright P'' \implies \hat{\tau} P'$ 
 $\wedge (\Psi \otimes \Psi', P', Q') \in Rel)$ 
and  $\bigwedge Q'. [\Psi \triangleright Q \xrightarrow{\tau} Q'; x \# Q'] \implies \exists P'. \Psi \triangleright P \implies \hat{\tau} P' \wedge (\Psi, P',$ 
 $Q') \in Rel$ 

```

shows $\Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q$
 $\langle \text{proof} \rangle$

lemma *weakSimE*:

fixes $F :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$

assumes $\Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q$

shows $\bigwedge \Psi' \alpha \ Q'. [\Psi \triangleright Q \xrightarrow{\alpha} Q'; \text{bn } \alpha \not\models \Psi; \text{bn } \alpha \not\models P; \alpha \neq \tau] \implies \exists P''. \Psi : Q \triangleright P \xrightarrow{\alpha} P'' \wedge (\exists P'. \Psi \otimes \Psi' \triangleright P'' \implies \hat{\tau} P' \wedge (\Psi, P', Q') \in \text{Rel})$
and $\bigwedge Q'. \Psi \triangleright Q \xrightarrow{\tau} Q' \implies \exists P'. \Psi \triangleright P \implies \hat{\tau} P' \wedge (\Psi, P', Q') \in \text{Rel}$
 $\langle \text{proof} \rangle$

lemma *weakSimClosedAux*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $p :: \text{name prm}$

assumes *EqvtRel*: eqvt Rel
and $\text{PSimQ} : \Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q$

shows $(p \cdot \Psi) \triangleright (p \cdot P) \rightsquigarrow_{\text{Rel}} (p \cdot Q)$
 $\langle \text{proof} \rangle$

lemma *weakSimClosed*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $p :: \text{name prm}$

assumes *EqvtRel*: eqvt Rel

shows $\Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q \implies (p \cdot \Psi) \triangleright (p \cdot P) \rightsquigarrow_{\text{Rel}} (p \cdot Q)$
and $P \rightsquigarrow_{\text{Rel}} Q \implies (p \cdot P) \rightsquigarrow_{\text{Rel}} (p \cdot Q)$
 $\langle \text{proof} \rangle$

lemma *weakSimReflexive*:

fixes $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$

assumes $\{(\Psi, P, P) \mid \Psi P. \text{True}\} \subseteq \text{Rel}$

shows $\Psi \triangleright P \rightsquigarrow_{\text{Rel}} P$
 $\langle \text{proof} \rangle$

lemma *weakSimTauChain*:

fixes $\Psi :: 'b$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $Q' :: ('a, 'b, 'c) \text{ psi}$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes $\Psi \triangleright Q \Rightarrow \hat{\tau} Q'$
and $(\Psi, P, Q) \in \text{Rel}$
and $\text{Sim}: \bigwedge \Psi' R S. (\Psi', R, S) \in \text{Rel} \Rightarrow \Psi' \triangleright R \rightsquigarrow_{\text{Rel}} S$

obtains P' **where** $\Psi \triangleright P \Rightarrow \hat{\tau} P'$ **and** $(\Psi, P', Q') \in \text{Rel}$
 $\langle \text{proof} \rangle$

lemma *weakSimE2*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$

assumes $P\text{Rel}Q: (\Psi, P, Q) \in \text{Rel}$
and $\text{Sim}: \bigwedge \Psi' R S. (\Psi', R, S) \in \text{Rel} \Rightarrow \Psi' \triangleright R \rightsquigarrow_{\text{Rel}} S$
and $Q\text{Trans}: \Psi : R \triangleright Q \Rightarrow \alpha \prec Q'$
and $bn \alpha \sharp* \Psi$
and $bn \alpha \sharp* P$
and $\alpha \neq \tau$

obtains $P'' P'$ **where** $\Psi : R \triangleright P \Rightarrow \alpha \prec P''$ **and** $\Psi \otimes \Psi' \triangleright P'' \Rightarrow \hat{\tau} P'$ **and**
 $(\Psi \otimes \Psi', P', Q') \in \text{Rel}$
 $\langle \text{proof} \rangle$

lemma *weakSimTransitive*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel}' :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $T :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel}'' :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes $P\text{Rel}Q: (\Psi, P, Q) \in \text{Rel}$
and $Q\text{Sim}R: \Psi \triangleright Q \rightsquigarrow_{\text{Rel}'} R$
and $\text{Eqvt}: \text{eqvt Rel}''$
and $\text{Set}: \{(\Psi, P, R) \mid \Psi P R. \exists Q. (\Psi, P, Q) \in \text{Rel} \wedge (\Psi, Q, R) \in \text{Rel}'\} \subseteq$

Rel''

and $\text{Sim}: \bigwedge \Psi' R S. (\Psi', R, S) \in \text{Rel} \implies \Psi' \triangleright R \rightsquigarrow_{\text{Rel}} S$

shows $\Psi \triangleright P \rightsquigarrow_{\text{Rel}''} R$

(proof)

lemma *weakSimStatEq*:

fixes $\Psi :: 'b$

and $P :: ('a, 'b, 'c) \text{ psi}$

and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

and $Q :: ('a, 'b, 'c) \text{ psi}$

and $\Psi' :: 'b$

assumes *PSimQ*: $\Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q$

and $\text{eqvt Rel}'$

and $\Psi \simeq \Psi'$

and $C1: \bigwedge \Psi' R S \Psi''. [(\Psi', R, S) \in \text{Rel}; \Psi' \simeq \Psi''] \implies (\Psi'', R, S) \in \text{Rel}'$

shows $\Psi' \triangleright P \rightsquigarrow_{\text{Rel}'} Q$

(proof)

lemma *weakSimMonotonic*:

fixes $\Psi :: 'b$

and $P :: ('a, 'b, 'c) \text{ psi}$

and $A :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

and $Q :: ('a, 'b, 'c) \text{ psi}$

and $B :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes $\Psi \triangleright P \rightsquigarrow_A Q$

and $A \subseteq B$

shows $\Psi \triangleright P \rightsquigarrow_B Q$

(proof)

lemma *strongSimWeakSim*:

fixes $\Psi :: 'b$

and $P :: ('a, 'b, 'c) \text{ psi}$

and $Q :: ('a, 'b, 'c) \text{ psi}$

and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes *PRelQ*: $(\Psi, P, Q) \in \text{Rel}$

and $\text{StatImp}: \bigwedge \Psi' R S. (\Psi', R, S) \in \text{Rel} \implies \text{insertAssertion}(\text{extractFrame } S) \Psi' \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } R) \Psi'$

and $\text{Sim}: \bigwedge \Psi' R S. (\Psi', R, S) \in \text{Rel} \implies \Psi' \triangleright R \rightsquigarrow_{[\text{Rel}]} S$

and $\text{Ext}: \bigwedge \Psi' R S \Psi''. (\Psi', R, S) \in \text{Rel} \implies (\Psi' \otimes \Psi'', R, S) \in \text{Rel}$

shows $\Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q$

(proof)

```

lemma strongAppend:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 
  and  $R :: ('a, 'b, 'c) \text{psi}$ 
  and  $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{psi} \times ('a, 'b, 'c) \text{psi}) \text{set}$ 
  and  $\text{Rel}' :: ('b \times ('a, 'b, 'c) \text{psi} \times ('a, 'b, 'c) \text{psi}) \text{set}$ 
  and  $\text{Rel}'' :: ('b \times ('a, 'b, 'c) \text{psi} \times ('a, 'b, 'c) \text{psi}) \text{set}$ 

  assumes  $\text{PSimQ}: \Psi \triangleright P \rightsquigarrow_{\text{Rel}} Q$ 
  and  $\text{QSimR}: \Psi \triangleright Q \rightsquigarrow[\text{Rel}'] R$ 
  and  $\text{Eqvt}'' : \text{eqvt Rel}''$ 
  and  $\text{RimpQ}: \text{insertAssertion}(\text{extractFrame } R) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } Q) \Psi$ 
  and  $\text{Set}: \{(\Psi, P, R) \mid \Psi \triangleright P \text{ R. } \exists Q. (\Psi, P, Q) \in \text{Rel} \wedge (\Psi, Q, R) \in \text{Rel}'\} \subseteq \text{Rel}''$ 
  and  $C1: \bigwedge \Psi \triangleright P \text{ Q. } \Psi'. (\Psi, P, Q) \in \text{Rel}' \implies (\Psi \otimes \Psi', P, Q) \in \text{Rel}'$ 

  shows  $\Psi \triangleright P \rightsquigarrow_{\text{Rel}''} R$ 
   $\langle \text{proof} \rangle$ 

lemma quietSim:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 

  assumes quiet P
  and  $\text{eqvt Rel}$ 
  and  $cQuiet: \bigwedge P. \text{quiet } P \implies (\Psi, \mathbf{0}, P) \in \text{Rel}$ 

  shows  $\Psi \triangleright \mathbf{0} \rightsquigarrow_{\text{Rel}} P$ 
   $\langle \text{proof} \rangle$ 

end

end

theory Weak-Stat-Imp
  imports Tau-Chain
begin

context env begin

definition
   $\text{weakStatImp} :: 'b \Rightarrow ('a, 'b, 'c) \text{psi} \Rightarrow$ 
   $('b \times ('a, 'b, 'c) \text{psi} \times ('a, 'b, 'c) \text{psi}) \text{set} \Rightarrow$ 
   $('a, 'b, 'c) \text{psi} \Rightarrow \text{bool} ((\leftarrow \triangleright - \approx \leftarrow \rightarrow [80, 80, 80, 80] 80))$ 
where  $\Psi \triangleright P \approx_{\text{Rel}} Q \equiv \forall \Psi'. \exists Q' Q''. \Psi \triangleright Q \implies \hat{\tau} Q' \wedge \text{insertAssertion}(\text{extractFrame } P) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } Q') \Psi \wedge \Psi \otimes \Psi' \triangleright Q'$ 

```

```

 $\Rightarrow^{\hat{\tau}} Q'' \wedge (\Psi \otimes \Psi', P, Q'') \in Rel$ 

lemma weakStatImpMonotonic:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $A :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $B :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 

  assumes  $\Psi \triangleright P \lessapprox_{\langle A \rangle} Q$ 
  and  $A \subseteq B$ 

  shows  $\Psi \triangleright P \lessapprox_{\langle B \rangle} Q$ 
   $\langle proof \rangle$ 

lemma weakStatImpI[case-names cStatImp]:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Rel :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\Psi' :: 'b$ 

  assumes  $\bigwedge \Psi'. \exists Q' Q''. \Psi \triangleright Q \Rightarrow^{\hat{\tau}} Q' \wedge \text{insertAssertion}(\text{extractFrame } P) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } Q') \Psi \wedge \Psi \otimes \Psi' \triangleright Q' \Rightarrow^{\hat{\tau}} Q'' \wedge (\Psi \otimes \Psi', P, Q'') \in Rel$ 

  shows  $\Psi \triangleright P \lessapprox_{\langle Rel \rangle} Q$ 
   $\langle proof \rangle$ 

lemma weakStatImpE:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Rel :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\Psi' :: 'b$ 

  assumes  $\Psi \triangleright P \lessapprox_{\langle Rel \rangle} Q$ 

  obtains  $Q' Q''$  where  $\Psi \triangleright Q \Rightarrow^{\hat{\tau}} Q'$  and  $\text{insertAssertion}(\text{extractFrame } P) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } Q') \Psi$  and  $\Psi \otimes \Psi' \triangleright Q' \Rightarrow^{\hat{\tau}} Q''$  and  $(\Psi \otimes \Psi', P, Q'') \in Rel$ 

   $\langle proof \rangle$ 

lemma weakStatImpClosed:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Rel :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 

```

```

and    $p :: name\ prm$ 

assumes  $EqvtRel: eqvt\ Rel$ 
and     $PStatImpQ: \Psi \triangleright P \lessapprox_{\langle Rel \rangle} Q$ 

shows  $(p \cdot \Psi) \triangleright (p \cdot P) \lessapprox_{\langle Rel \rangle} (p \cdot Q)$ 
 $\langle proof \rangle$ 

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  $\Psi \triangleright P \lessapprox_{\langle Rel \rangle} P$ 
 $\langle proof \rangle$ 

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  $PStatImpQ: \Psi \triangleright P \lessapprox_{\langle Rel \rangle} Q$ 
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     $C1: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel' \implies \Psi' \triangleright S \lessapprox_{\langle Rel' \rangle} T$ 
and     $C2: \bigwedge \Psi' S T S'. [(\Psi', S, T) \in Rel'; \Psi' \triangleright S \implies \hat{\tau} S] \implies \exists T'. \Psi' \triangleright T \implies \hat{\tau} T' \wedge (\Psi', S', T') \in Rel'$ 

shows  $\Psi \triangleright P \lessapprox_{\langle Rel'' \rangle} R$ 
 $\langle proof \rangle$ 

lemma  $weakStatImpStatEq:$ 
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  $PSimQ: \Psi \triangleright P \lessapprox_{\langle Rel \rangle} Q$ 
and     $\Psi \simeq \Psi'$ 
and     $C1: \bigwedge \Psi' R S \Psi''. [(\Psi', R, S) \in Rel; \Psi' \simeq \Psi''] \implies (\Psi'', R, S) \in Rel'$ 

```

```

shows  $\Psi' \triangleright P \approx_{\text{Rel}'} Q$ 
⟨proof⟩

lemma statImpWeakStatImp:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $\text{Rel} :: ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) \text{ set}$ 

  assumes  $\text{PImpQ}: \text{insertAssertion}(\text{extractFrame } P) \Psi \hookrightarrow_F \text{insertAssertion}(\text{extractFrame } Q) \Psi$ 
  and  $C1: \bigwedge \Psi'. (\Psi \otimes \Psi', P, Q) \in \text{Rel}$ 

  shows  $\Psi \triangleright P \approx_{\text{Rel}} Q$ 
⟨proof⟩

end

end

theory Bisimulation
  imports Simulation
begin

context env begin

lemma monoCoinduct:  $\bigwedge x y xa xb xc P Q \Psi.$ 
   $x \leq y \implies$ 
   $(\Psi \triangleright Q \rightsquigarrow [ \{(xc, xb, xa). x xc xb xa\} ] P) \longrightarrow$ 
   $(\Psi \triangleright Q \rightsquigarrow [ \{(xb, xa, xc). y xb xa xc\} ] P)$ 
⟨proof⟩

coinductive-set bisim ::  $('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) \text{ set}$ 
where
  step:  $\llbracket (\text{insertAssertion}(\text{extractFrame } P)) \Psi \simeq_F (\text{insertAssertion}(\text{extractFrame } Q) \Psi);$ 
     $\Psi \triangleright P \rightsquigarrow [\text{bisim}] Q;$ 
     $\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in \text{bisim}; (\Psi, Q, P) \in \text{bisim} \rrbracket \implies (\Psi, P, Q) \in \text{bisim}$ 
monos monoCoinduct

abbreviation
  bisimJudge ( $\langle \cdot \triangleright \cdot \sim \cdot \rangle [70, 70, 70] 65$ ) where  $\Psi \triangleright P \sim Q \equiv (\Psi, P, Q) \in \text{bisim}$ 
abbreviation
  bisimNilJudge ( $\langle \cdot \sim \cdot \rangle [70, 70] 65$ ) where  $P \sim Q \equiv SBottom' \triangleright P \sim Q$ 

lemma bisimCoinductAux[consumes 1]:
  fixes  $F :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

```

and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \text{insertAssertion}(\text{extractFrame } Q) \Psi \wedge$
 $(\Psi \triangleright P \rightsquigarrow [(X \cup \text{bisim})] Q) \wedge$
 $(\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in X \vee (\Psi \otimes \Psi', P, Q) \in \text{bisim}) \wedge$
 $((\Psi, Q, P) \in X \vee (\Psi, Q, P) \in \text{bisim})$

shows $(\Psi, P, Q) \in \text{bisim}$
 $\langle \text{proof} \rangle$

lemma $\text{bisimCoinduct}[\text{consumes 1, case-names cStatEq cSim cExt cSym}]$:

fixes $F :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \text{insertAssertion}(\text{extractFrame } R) \Psi' \simeq_F \text{insertAssertion}(\text{extractFrame } S) \Psi'$
and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \Psi' \triangleright R \rightsquigarrow [(X \cup \text{bisim})] S$
and $\bigwedge \Psi' R S \Psi''. (\Psi', R, S) \in X \implies (\Psi' \otimes \Psi'', R, S) \in X \vee (\Psi' \otimes \Psi'', R, S) \in \text{bisim}$
and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies (\Psi', S, R) \in X \vee (\Psi', S, R) \in \text{bisim}$

shows $(\Psi, P, Q) \in \text{bisim}$
 $\langle \text{proof} \rangle$

lemma $\text{bisimWeakCoinductAux}[\text{consumes 1}]$:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \text{insertAssertion}(\text{extractFrame } Q) \Psi \wedge$
 $\Psi \triangleright P \rightsquigarrow [X] Q \wedge$
 $(\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in X) \wedge (\Psi, Q, P) \in X$

shows $(\Psi, P, Q) \in \text{bisim}$
 $\langle \text{proof} \rangle$

lemma $\text{bisimWeakCoinduct}[\text{consumes 1, case-names cStatEq cSim cExt cSym}]$:

fixes $F :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$

and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes $(\Psi, P, Q) \in X$
and $\wedge \Psi P Q. (\Psi, P, Q) \in X \implies \text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \text{insertAssertion}(\text{extractFrame } Q) \Psi$
and $\wedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow[X] Q$
and $\wedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X$
and $\wedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X$

shows $(\Psi, P, Q) \in \text{bisim}$
 $\langle \text{proof} \rangle$

lemma $\text{bisimE}:$
fixes $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $\Psi :: 'b$
and $\Psi' :: 'b$

assumes $(\Psi, P, Q) \in \text{bisim}$

shows $\text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \text{insertAssertion}(\text{extractFrame } Q) \Psi$
 $\langle \text{proof} \rangle$

and $\Psi \triangleright P \rightsquigarrow[\text{bisim}] Q$
and $(\Psi \otimes \Psi', P, Q) \in \text{bisim}$
and $(\Psi, Q, P) \in \text{bisim}$

$\langle \text{proof} \rangle$

lemma $\text{bisimI}:$
fixes $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $\Psi :: 'b$

assumes $\text{insertAssertion}(\text{extractFrame } P) \Psi \simeq_F \text{insertAssertion}(\text{extractFrame } Q) \Psi$
and $\Psi \triangleright P \rightsquigarrow[\text{bisim}] Q$
and $\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in \text{bisim}$
and $(\Psi, Q, P) \in \text{bisim}$

shows $(\Psi, P, Q) \in \text{bisim}$
 $\langle \text{proof} \rangle$

lemma $\text{bisimReflexive}:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$

shows $\Psi \triangleright P \sim P$
 $\langle \text{proof} \rangle$

```

lemma bisimClosed:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $p :: name prm$ 

  assumes  $PBisimQ: \Psi \triangleright P \sim Q$ 

  shows  $(p \cdot \Psi) \triangleright (p \cdot P) \sim (p \cdot Q)$ 
   $\langle proof \rangle$ 

lemma bisimEqvt[simp]:
  shows eqvt bisim
   $\langle proof \rangle$ 

lemma statEqBisim:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $\Psi' :: 'b$ 

  assumes  $\Psi \triangleright P \sim Q$ 
  and  $\Psi \simeq \Psi'$ 

  shows  $\Psi' \triangleright P \sim Q$ 
   $\langle proof \rangle$ 

lemma bisimTransitive:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $R :: ('a, 'b, 'c) \psi$ 

  assumes  $PQ: \Psi \triangleright P \sim Q$ 
  and  $QR: \Psi \triangleright Q \sim R$ 

  shows  $\Psi \triangleright P \sim R$ 
   $\langle proof \rangle$ 

lemma weakTransitiveCoinduct[case-names cStateEq cSim cExt cSym, case-conclusion
  bisim step, consumes 2]:
  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  $p: (\Psi, P, Q) \in X$ 
  and Eqvt: eqvt  $X$ 

```

and $rStatEq: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies insertAssertion(extractFrame P) \Psi \simeq_F insertAssertion(extractFrame Q) \Psi$
and $rSim: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow [(\{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P'\} \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\})] Q$
and $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X$
and $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X$

shows $\Psi \triangleright P \sim Q$
 $\langle proof \rangle$

lemma $weakTransitiveCoinduct'[\text{case-names } cStatEq \ cSim \ cExt \ cSym, \text{ case-conclusion bisim step, consumes 2}]:$

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $X :: ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) \text{ set}$

assumes $p: (\Psi, P, Q) \in X$
and $Eqvt: eqvt X$
and $rStatEq: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies insertAssertion(extractFrame P) \Psi \simeq_F insertAssertion(extractFrame Q) \Psi$
and $rSim: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow [(\{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P'\} \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\})] Q$
and $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X$
and $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in \{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\}$

shows $\Psi \triangleright P \sim Q$
 $\langle proof \rangle$

lemma $weakTransitiveCoinduct''[\text{case-names } cStatEq \ cSim \ cExt \ cSym, \text{ case-conclusion bisim step, consumes 2}]:$

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $X :: ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) \text{ set}$

assumes $p: (\Psi, P, Q) \in X$
and $Eqvt: eqvt X$
and $rStatEq: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies insertAssertion(extractFrame P) \Psi \simeq_F insertAssertion(extractFrame Q) \Psi$
and $rSim: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow [(\{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P'\} \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\})] Q$

$\Psi \triangleright Q' \sim Q\})] Q$

and $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in \{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\} \implies$
 $(\Psi \otimes \Psi', P, Q) \in \{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\}$

and $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in \{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\} \implies$
 $(\Psi, Q, P) \in \{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\}$

shows $\Psi \triangleright P \sim Q$
 $\langle proof \rangle$

lemma $transitiveCoinduct[case-names cStatEq cSim cExt cSym, case-conclusion bisim step, consumes 2]:$

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 $p: (\Psi, P, Q) \in X$
and $Eqvt: eqvt X$
and $rStatEq: \bigwedge \Psi' R S. (\Psi', R, S) \in X \implies insertAssertion(extractFrame R) \Psi' \simeq_F insertAssertion(extractFrame S) \Psi'$
and $rSim: \bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \Psi' \triangleright R \rightsquigarrow [(\{(\Psi', R, S) \mid \Psi' R R' S' \wedge S. \Psi' \triangleright R \sim R' \wedge ((\Psi', R', S') \in X \vee \Psi' \triangleright R' \sim S') \wedge \Psi' \triangleright S'\})] S$
and $rExt: \bigwedge \Psi' R S \Psi''. (\Psi', R, S) \in X \implies (\Psi' \otimes \Psi'', R, S) \in X \vee \Psi' \otimes \Psi'' \triangleright R \sim S$
and $rSym: \bigwedge \Psi' R S. (\Psi', R, S) \in X \implies (\Psi', S, R) \in X \vee \Psi' \triangleright S \sim R$

shows $\Psi \triangleright P \sim Q$
 $\langle proof \rangle$

lemma $transitiveCoinduct'[case-names cStatEq cSim cExt cSym, case-conclusion bisim step, consumes 2]:$

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 $p: (\Psi, P, Q) \in X$
and $Eqvt: eqvt X$
and $rStatEq: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies insertAssertion(extractFrame P) \Psi \simeq_F insertAssertion(extractFrame Q) \Psi$
and $rSim: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow [(\{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\})] Q$

```

 $\Psi \triangleright P \sim P' \wedge$ 
 $(\Psi, P', Q') \in (X \cup bisim) \wedge$ 
 $\Psi \triangleright Q' \sim Q\})] Q$ 
and  $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X \vee \Psi \otimes \Psi' \triangleright P$ 
 $\sim Q$ 
and  $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies$ 
 $(\Psi, Q, P) \in \{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge ((\Psi, P', Q') \in (X \cup bisim)) \wedge \Psi \triangleright Q' \sim Q\}$ 

shows  $\Psi \triangleright P \sim Q$ 
⟨proof⟩

lemma bisimSymmetric:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) psi$ 
and  $Q :: ('a, 'b, 'c) psi$ 

assumes  $\Psi \triangleright P \sim Q$ 

shows  $\Psi \triangleright Q \sim P$ 
⟨proof⟩

lemma eqvtTrans[intro]:
assumes eqvt X

shows eqvt  $\{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge ((\Psi, P', Q') \in X \vee \Psi \triangleright P' \sim Q') \wedge \Psi \triangleright Q' \sim Q\}$ 
⟨proof⟩

lemma eqvtWeakTrans[intro]:
assumes eqvt X

shows eqvt  $\{(\Psi, P, Q) \mid \Psi P P' Q' Q. \Psi \triangleright P \sim P' \wedge ((\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q)\}$ 
⟨proof⟩

end

end

theory Sim-Pres
imports Simulation
begin

context env begin

lemma inputPres:
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 M :: 'a
and xvec :: name list
and N :: 'a

assumes PRelQ:  $\bigwedge Tvec. \text{length } xvec = \text{length } Tvec \implies (\Psi, P[xvec:=Tvec], Q[xvec:=Tvec]) \in \text{Rel}$ 

shows  $\Psi \triangleright M(\lambda*xvec\ N).P \rightsquigarrow[\text{Rel}] M(\lambda*xvec\ N).Q$ 
⟨proof⟩

```

```

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: $(\Psi, P, Q) \in \text{Rel}$

shows $\Psi \triangleright M\langle N \rangle.P \rightsquigarrow[\text{Rel}] M\langle N \rangle.Q$
⟨proof⟩

```

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: $\bigwedge \varphi\ Q. (\varphi, Q) \text{ mem } CsQ \implies \exists P. (\varphi, P) \text{ mem } CsP \wedge \text{guarded}$
 $P \wedge (\Psi, P, Q) \in \text{Rel}$
and Sim: $\bigwedge \Psi' R S. (\Psi', R, S) \in \text{Rel} \implies \Psi' \triangleright R \rightsquigarrow[\text{Rel}] S$
and Rel ⊆ Rel'

shows $\Psi \triangleright \text{Cases } CsP \rightsquigarrow[\text{Rel}'] \text{Cases } CsQ$
⟨proof⟩

```

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 $PSimQ: \Psi \triangleright P \rightsquigarrow[Rel] Q$
and $eqvt Rel'$
and $x \notin \Psi$
and $Rel \subseteq Rel'$
and $C1: \bigwedge \Psi' R S y. [(\Psi', R, S) \in Rel; y \notin \Psi] \implies (\Psi', (\nu y)R, (\nu y)S) \in Rel'$

shows $\Psi \triangleright (\nu x)P \rightsquigarrow[Rel] (\nu x)Q$
 $\langle proof \rangle$

lemma $resChainPres:$

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 $xvec :: name list$

assumes $PSimQ: \Psi \triangleright P \rightsquigarrow[Rel] Q$
and $eqvt Rel$
and $xvec \#* \Psi$
and $C1: \bigwedge \Psi' R S y. [(\Psi', R, S) \in Rel; y \notin \Psi] \implies (\Psi', (\nu y)R, (\nu y)S) \in Rel$

shows $\Psi \triangleright (\nu * xvec)P \rightsquigarrow[Rel] (\nu * xvec)Q$
 $\langle proof \rangle$

lemma $parPres:$

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 $R :: ('a, 'b, 'c) psi$
and $Rel' :: ('b \times ('a, 'b, 'c) psi \times ('a, 'b, 'c) psi) set$

assumes $PRelQ: \bigwedge A_R \Psi_R. [extractFrame R = \langle A_R, \Psi_R \rangle; A_R \#* \Psi; A_R \#* P; A_R \#* Q] \implies (\Psi \otimes \Psi_R, P, Q) \in Rel$
and $Eqvt: eqvt Rel$
and $Eqvt': eqvt Rel'$

and $StatImp: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies insertAssertion (extractFrame T) \Psi' \hookrightarrow_F insertAssertion (extractFrame S) \Psi'$
and $Sim: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies \Psi' \triangleright S \rightsquigarrow[Rel] T$
and $Ext: \bigwedge \Psi' S T \Psi''. [(\Psi', S, T) \in Rel] \implies (\Psi' \otimes \Psi'', S, T) \in Rel$

and $C1: \bigwedge \Psi' S T A_U \Psi_U U. [(\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = \langle A_U, \Psi_U \rangle; A_U \#* \Psi'; A_U \#* S; A_U \#* T] \implies (\Psi', S \parallel U, T \parallel U) \in Rel'$
and $C2: \bigwedge \Psi' S T xvec. [(\Psi', S, T) \in Rel'; xvec \#* \Psi'] \implies (\Psi', (\nu * xvec)S, (\nu * xvec)T) \in Rel'$

and $C3: \bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel; \Psi' \simeq \Psi' \rrbracket \implies (\Psi'', S, T) \in Rel$

shows $\Psi \triangleright P \parallel R \rightsquigarrow [Rel'] Q \parallel R$
 $\langle proof \rangle$

unbundle no relcomp-syntax

lemma $bangPres$:

fixes $\Psi :: 'b$

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 $cSim: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies \Psi' \triangleright S \rightsquigarrow [Rel] T$

and $cExt: \bigwedge \Psi' S T \Psi''. (\Psi', S, T) \in Rel \implies (\Psi' \otimes \Psi'', S, T) \in Rel$

and $cSym: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies (\Psi', T, S) \in Rel$

and $StatEq: \bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel; \Psi' \simeq \Psi' \rrbracket \implies (\Psi'', S, T) \in Rel$

and $Closed: \bigwedge \Psi' S T p. (\Psi', S, T) \in Rel \implies ((p :: name prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel$

and $Assoc: \bigwedge \Psi' S T U. (\Psi', S \parallel (T \parallel U), (S \parallel T) \parallel U) \in Rel$

and $ParPres: \bigwedge \Psi' S T U. (\Psi', S, T) \in Rel \implies (\Psi', S \parallel U, T \parallel U) \in Rel$

and $FrameParPres: \bigwedge \Psi' \Psi_U S T U A_U. \llbracket (\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = \langle A_U, \Psi_U \rangle; A_U \#* \Psi'; A_U \#* S; A_U \#* T \rrbracket \implies (\Psi', U \parallel S, U \parallel T) \in Rel$

and $ResPres: \bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu * xvec) S, (\nu * xvec) T) \in Rel$

and $ScopeExt: \bigwedge xvec \Psi' S T. \llbracket xvec \#* \Psi'; xvec \#* T \rrbracket \implies (\Psi', (\nu * xvec)(S \parallel T), ((\nu * xvec) S) \parallel T) \in Rel$

and $Trans: \bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel; (\Psi', T, U) \in Rel \rrbracket \implies (\Psi', S, U) \in Rel$

and $Compose: \bigwedge \Psi' S T U O. \llbracket (\Psi', S, T) \in Rel; (\Psi', T, U) \in Rel'; (\Psi', U, O) \in Rel \rrbracket \implies (\Psi', S, O) \in Rel'$

and $C1: \bigwedge \Psi S T U. \llbracket (\Psi, S, T) \in Rel; guarded S; guarded T \rrbracket \implies (\Psi, U \parallel !S, U \parallel !T) \in Rel'$

and $Der: \bigwedge \Psi' S \alpha S' T. \llbracket \Psi' \triangleright !S \mapsto \alpha \prec S'; (\Psi', S, T) \in Rel; bn \alpha \#* \Psi'; bn \alpha \#* S; bn \alpha \#* T; guarded T; bn \alpha \#* subject \alpha \rrbracket \implies \exists T' U O. \Psi' \triangleright !T \mapsto \alpha \prec T' \wedge (\Psi', S', U \parallel !S) \in Rel \wedge (\Psi', T', O \parallel !T) \in Rel \wedge$

$(\Psi', U, O) \in Rel \wedge ((supp U) :: name set) \subseteq supp S' \wedge$

$((supp O) :: name set) \subseteq supp T'$

shows $\Psi \triangleright R \parallel !P \rightsquigarrow [Rel'] R \parallel !Q$
 $\langle proof \rangle$

```

unbundle relcomp-syntax
end

end

theory Bisim-Pres
  imports Bisimulation Sim-Pres
begin

context env begin

lemma bisimInputPres:
  fixes  $\Psi$  :: ' $b$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $Q$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $M$  :: ' $a$ 
  and  $xvec$  :: name list
  and  $N$  :: ' $a$ 

assumes  $\bigwedge Tvec. \text{length } xvec = \text{length } Tvec \implies \Psi \triangleright P[xvec:=Tvec] \sim Q[xvec:=Tvec]$ 

shows  $\Psi \triangleright M(\lambda*xvec N).P \sim M(\lambda*xvec N).Q$ 
(proof)

lemma bisimOutputPres:
  fixes  $\Psi$  :: ' $b$ 
  and  $P$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $Q$  :: (' $a$ , ' $b$ , ' $c$ )  $\psi$ 
  and  $M$  :: ' $a$ 
  and  $N$  :: ' $a$ 

assumes  $\Psi \triangleright P \sim Q$ 

shows  $\Psi \triangleright M\langle N \rangle.P \sim M\langle N \rangle.Q$ 
(proof)

lemma bisimCasePres:
  fixes  $\Psi$  :: ' $b$ 
  and  $CsP$  :: (' $c \times ('a, 'b, 'c) \psi$ ) list
  and  $CsQ$  :: (' $c \times ('a, 'b, 'c) \psi$ ) list

assumes  $\bigwedge \varphi P. (\varphi, P) \text{ mem } CsP \implies \exists Q. (\varphi, Q) \text{ mem } CsQ \wedge \text{guarded } Q \wedge \Psi \triangleright P \sim Q$ 
  and  $\bigwedge \varphi Q. (\varphi, Q) \text{ mem } CsQ \implies \exists P. (\varphi, P) \text{ mem } CsP \wedge \text{guarded } P \wedge \Psi \triangleright P \sim Q$ 

shows  $\Psi \triangleright \text{Cases } CsP \sim \text{Cases } CsQ$ 
(proof)

```

```

lemma bisimResPres:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 
  and  $x :: \text{name}$ 

  assumes  $\Psi \triangleright P \sim Q$ 
  and  $x \notin \Psi$ 

  shows  $\Psi \triangleright (\nu x)P \sim (\nu x)Q$ 
   $\langle \text{proof} \rangle$ 

lemma bisimResChainPres:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 
  and  $xvec :: \text{name list}$ 

  assumes  $\Psi \triangleright P \sim Q$ 
  and  $xvec \nsubseteq \Psi$ 

  shows  $\Psi \triangleright (\nu * xvec)P \sim (\nu * xvec)Q$ 
   $\langle \text{proof} \rangle$ 

lemma bisimParPresAux:
  fixes  $\Psi :: 'b$ 
  and  $\Psi_R :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 
  and  $R :: ('a, 'b, 'c) \text{psi}$ 
  and  $A_R :: \text{name list}$ 

  assumes  $\Psi \otimes \Psi_R \triangleright P \sim Q$ 
  and  $\text{FrR: extractFrame } R = \langle A_R, \Psi_R \rangle$ 
  and  $A_R \nsubseteq \Psi$ 
  and  $A_R \nsubseteq P$ 
  and  $A_R \nsubseteq Q$ 

  shows  $\Psi \triangleright P \parallel R \sim Q \parallel R$ 
   $\langle \text{proof} \rangle$ 

lemma bisimParPres:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{psi}$ 
  and  $R :: ('a, 'b, 'c) \text{psi}$ 

  assumes  $\Psi \triangleright P \sim Q$ 

```

```

shows  $\Psi \triangleright P \parallel R \sim Q \parallel R$ 
⟨proof⟩

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
⟨proof⟩

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'
⟨proof⟩

context env begin

lemma resComm:
fixes  $\Psi :: 'b$ 
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  $x \notin \Psi$ 
and  $y \notin \Psi$ 
and  $eqvt Rel$ 
and  $R1: \bigwedge \Psi' Q. (\Psi', Q, Q) \in Rel$ 
and  $R2: \bigwedge \Psi' a b Q. [a \notin \Psi'; b \notin \Psi] \implies (\Psi', (\lambda a)(\lambda b)(Q), (\lambda a)(\lambda b)(Q)) \in Rel$ 

shows  $\Psi \triangleright (\lambda x)(\lambda y)(P) \rightsquigarrow [Rel] (\lambda y)(\lambda x)(P)$ 
⟨proof⟩

lemma parAssocLeft:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 

```

```

and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $R :: ('a, 'b, 'c) \text{ psi}$ 
and    $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 

assumes  $\text{eqvt Rel}$ 
and     $C1: \bigwedge \Psi' S T U. (\Psi, (S \parallel T) \parallel U, S \parallel (T \parallel U)) \in \text{Rel}$ 
and     $C2: \bigwedge xvec \Psi' S T U. [xvec \#* \Psi'; xvec \#* S] \implies (\Psi', (\nu*xvec)((S \parallel T) \parallel U), S \parallel (\nu*xvec)(T \parallel U)) \in \text{Rel}$ 
and     $C3: \bigwedge xvec \Psi' S T U. [xvec \#* \Psi'; xvec \#* U] \implies (\Psi', ((\nu*xvec)(S \parallel T)) \parallel U, (\nu*xvec)(S \parallel (T \parallel U))) \in \text{Rel}$ 
and     $C4: \bigwedge \Psi' S T xvec. [(\Psi', S, T) \in \text{Rel}; xvec \#* \Psi'] \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in \text{Rel}$ 

shows  $\Psi \triangleright (P \parallel Q) \parallel R \rightsquigarrow[\text{Rel}] P \parallel (Q \parallel R)$ 
(proof)

lemma  $\text{parNilLeft}:$ 
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 

assumes  $\text{eqvt Rel}$ 
and     $C1: \bigwedge Q. (\Psi, Q \parallel \mathbf{0}, Q) \in \text{Rel}$ 

shows  $\Psi \triangleright (P \parallel \mathbf{0}) \rightsquigarrow[\text{Rel}] P$ 
(proof)

lemma  $\text{parNilRight}:$ 
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 

assumes  $\text{eqvt Rel}$ 
and     $C1: \bigwedge Q. (\Psi, Q, (Q \parallel \mathbf{0})) \in \text{Rel}$ 

shows  $\Psi \triangleright P \rightsquigarrow[\text{Rel}] (P \parallel \mathbf{0})$ 
(proof)

lemma  $\text{resNilLeft}:$ 
fixes  $x :: \text{name}$ 
and    $\Psi :: 'b$ 
and    $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$ 

shows  $\Psi \triangleright (\nu x)\mathbf{0} \rightsquigarrow[\text{Rel}] \mathbf{0}$ 
(proof)

lemma  $\text{resNilRight}:$ 
fixes  $x :: \text{name}$ 
and    $\Psi :: 'b$ 

```

and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{psi} \times ('a, 'b, 'c) \text{psi}) \text{ set}$

shows $\Psi \triangleright \mathbf{0} \rightsquigarrow [\text{Rel}] (\nu x) \mathbf{0}$
 $\langle \text{proof} \rangle$

lemma $\text{inputPushResLeft}:$

fixes $\Psi :: 'b$
and $x :: \text{name}$
and $M :: 'a$
and $xvec :: \text{name list}$
and $N :: 'a$
and $P :: ('a, 'b, 'c) \text{psi}$

assumes eqvt Rel
and $x \notin \Psi$
and $x \notin M$
and $x \notin xvec$
and $x \notin N$
and $C1: \bigwedge Q. (\Psi, Q, Q) \in \text{Rel}$

shows $\Psi \triangleright (\nu x)(M(\lambda*xvec N).P) \rightsquigarrow [\text{Rel}] M(\lambda*xvec N).(\nu x)P$
 $\langle \text{proof} \rangle$

lemma $\text{inputPushResRight}:$

fixes $\Psi :: 'b$
and $x :: \text{name}$
and $M :: 'a$
and $xvec :: \text{name list}$
and $N :: 'a$
and $P :: ('a, 'b, 'c) \text{psi}$

assumes eqvt Rel
and $x \notin \Psi$
and $x \notin M$
and $x \notin xvec$
and $x \notin N$
and $C1: \bigwedge Q. (\Psi, Q, Q) \in \text{Rel}$

shows $\Psi \triangleright M(\lambda*xvec N).(\nu x)P \rightsquigarrow [\text{Rel}] (\nu x)(M(\lambda*xvec N).P)$
 $\langle \text{proof} \rangle$

lemma $\text{outputPushResLeft}:$

fixes $\Psi :: 'b$
and $x :: \text{name}$
and $M :: 'a$
and $N :: 'a$
and $P :: ('a, 'b, 'c) \text{psi}$

assumes eqvt Rel

```

and       $x \notin \Psi$ 
and       $x \notin M$ 
and       $x \notin N$ 
and       $C1: \bigwedge Q. (\Psi, Q, Q) \in Rel$ 

shows  $\Psi \triangleright (\nu x)(M\langle N \rangle.P) \rightsquigarrow [Rel] M\langle N \rangle.(\nu x)P$ 
 $\langle proof \rangle$ 

lemma outputPushResRight:
  fixes  $\Psi :: 'b$ 
  and    $x :: name$ 
  and    $M :: 'a$ 
  and    $N :: 'a$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  assumes eqvt Rel
  and    $x \notin \Psi$ 
  and    $x \notin M$ 
  and    $x \notin N$ 
  and    $C1: \bigwedge Q. (\Psi, Q, Q) \in Rel$ 

shows  $\Psi \triangleright M\langle N \rangle.(\nu x)P \rightsquigarrow [Rel] (\nu x)(M\langle N \rangle.P)$ 
 $\langle proof \rangle$ 

lemma casePushResLeft:
  fixes  $\Psi :: 'b$ 
  and    $x :: name$ 
  and    $Cs :: ('c \times ('a, 'b, 'c) \psi) list$ 

  assumes eqvt Rel
  and    $x \notin \Psi$ 
  and    $x \notin map fst Cs$ 
  and    $C1: \bigwedge Q. (\Psi, Q, Q) \in Rel$ 

shows  $\Psi \triangleright (\nu x)(Cases\ Cs) \rightsquigarrow [Rel] Cases (map (\lambda(\varphi, P). (\varphi, (\nu x)P)) Cs)$ 
 $\langle proof \rangle$ 

lemma casePushResRight:
  fixes  $\Psi :: 'b$ 
  and    $x :: name$ 
  and    $Cs :: ('c \times ('a, 'b, 'c) \psi) list$ 

  assumes eqvt Rel
  and    $x \notin \Psi$ 
  and    $x \notin map fst Cs$ 
  and    $C1: \bigwedge Q. (\Psi, Q, Q) \in Rel$ 

shows  $\Psi \triangleright Cases (map (\lambda(\varphi, P). (\varphi, (\nu x)P)) Cs) \rightsquigarrow [Rel] (\nu x)(Cases\ Cs)$ 
 $\langle proof \rangle$ 

```

```

lemma resInputCases[consumes 5, case-names cRes]:
  fixes  $\Psi$  :: '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:  $\Psi \triangleright (\nu x)P \longmapsto M(N) \prec P'$ 
  and  $x \notin \Psi$ 
  and  $x \notin M$ 
  and  $x \notin N$ 
  and  $x \notin P'$ 
  and rScope:  $\bigwedge P'. [\Psi \triangleright P \longmapsto M(N) \prec P] \implies Prop((\nu x)P')$ 

  shows Prop  $P'$ 
  {proof}

lemma scopeExtLeft:
  fixes  $x$  :: name
  and  $P$  :: ('a, 'b, 'c) psi
  and  $\Psi$  :: 'b
  and  $Q$  :: ('a, 'b, 'c) psi
  and Rel :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

  assumes  $x \notin P$ 
  and  $x \notin \Psi$ 
  and eqvt Rel
  and C1:  $\bigwedge \Psi' R. (\Psi', R, R) \in Rel$ 
  and C2:  $\bigwedge y \Psi' R S zvec. [y \notin \Psi'; y \notin R; zvec \notin \Psi] \implies (\Psi', (\nu y)(\nu zvec)(R \parallel S)), (\nu zvec)(R \parallel (\nu y)S)) \in Rel$ 
  and C3:  $\bigwedge \Psi' zvec R y. [y \notin \Psi'; zvec \notin \Psi] \implies (\Psi', (\nu y)(\nu zvec)(R), (\nu zvec)((\nu y)R)) \in Rel$ 

  shows  $\Psi \triangleright (\nu x)(P \parallel Q) \rightsquigarrow [Rel] P \parallel (\nu x)Q$ 
  {proof}

lemma scopeExtRight:
  fixes  $x$  :: name
  and  $P$  :: ('a, 'b, 'c) psi
  and  $\Psi$  :: 'b
  and  $Q$  :: ('a, 'b, 'c) psi
  and Rel :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

  assumes  $x \notin P$ 
  and  $x \notin \Psi$ 
  and eqvt Rel

```

```

and       $C1: \bigwedge \Psi' R. (\Psi, R, R) \in Rel$ 
and       $C2: \bigwedge y \Psi' R S zvec. \llbracket y \notin \Psi'; y \notin R; zvec \models \Psi \rrbracket \implies (\Psi', (\nu * zvec)(R \parallel (\nu y)S), (\nu y)((\nu * zvec)(R \parallel S))) \in Rel$ 

shows  $\Psi \triangleright P \parallel (\nu x)Q \rightsquigarrow [Rel] (\nu x)(P \parallel Q)$ 
 $\langle proof \rangle$ 

lemma simParComm:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 
  and    $Rel :: ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) set$ 

  assumes eqvt Rel
  and       $C1: \bigwedge \Psi' R S. (\Psi', R \parallel S, S \parallel R) \in Rel$ 
  and       $C2: \bigwedge \Psi' R S xvec. \llbracket (\Psi', R, S) \in Rel; xvec \models \Psi \rrbracket \implies (\Psi', (\nu * xvec)R, (\nu * xvec)S) \in Rel$ 

shows  $\Psi \triangleright P \parallel Q \rightsquigarrow [Rel] Q \parallel P$ 
 $\langle proof \rangle$ 

lemma bangExtLeft:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  assumes guarded P
  and       $\bigwedge \Psi' Q. (\Psi', Q, Q) \in Rel$ 

shows  $\Psi \triangleright !P \rightsquigarrow [Rel] P \parallel !P$ 
 $\langle proof \rangle$ 

lemma bangExtRight:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  assumes  $C1: \bigwedge \Psi' Q. (\Psi', Q, Q) \in Rel$ 

shows  $\Psi \triangleright P \parallel !P \rightsquigarrow [Rel] !P$ 
 $\langle proof \rangle$ 

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:  $P \equiv_s Q \Rightarrow Q \equiv_s P$ 
  | Trans:  $[P \equiv_s Q; Q \equiv_s R] \Rightarrow P \equiv_s R$ 

  | 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:  $x \notin P \Rightarrow (\nu x)(P \parallel Q) \equiv_s P \parallel (\nu x)Q$ 
  | InputRes:  $[x \notin M; x \notin xvec; x \notin N] \Rightarrow (\nu x)(M(\lambda*xvec N).P) \equiv_s M(\lambda*xvec N).(\nu x)P$ 
  | OutputRes:  $[x \notin M; x \notin N] \Rightarrow (\nu x)(M(N).P) \equiv_s M(N).(\nu x)P$ 
  | CaseRes:  $x \notin (map fst Cs) \Rightarrow (\nu x)(Cases Cs) \equiv_s Cases(map (\lambda(\varphi, P). (\varphi, (\nu x)P)) Cs)$ 

  | BangUnfold: guarded P => !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  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \parallel Q \sim Q \parallel P$ 
   $\langle proof \rangle$ 

lemma bisimResComm:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $y :: name$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright (\nu x)(\nu y)(P) \sim (\nu y)(\nu x)(P)$ 
   $\langle proof \rangle$ 

lemma bisimResComm':
  fixes  $x :: name$ 

```

```

and    $\Psi :: 'b$ 
and    $xvec :: name\ list$ 
and    $P :: ('a, 'b, 'c) \psi$ 

assumes  $x \notin \Psi$ 
and    $xvec \not\models \Psi$ 

```

shows $\Psi \triangleright (\nu x)(\nu * xvec)P \sim (\nu * xvec)(\nu x)P$
 $\langle proof \rangle$

```

lemma bisimScopeExt:
  fixes  $x :: name$ 
  and    $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 

```

assumes $x \notin P$

shows $\Psi \triangleright (\nu x)(P \parallel Q) \sim P \parallel (\nu x)Q$
 $\langle proof \rangle$

```

lemma bisimScopeExtChain:
  fixes  $xvec :: name\ list$ 
  and    $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 

```

assumes $xvec \not\models \Psi$
and $xvec \not\models P$

shows $\Psi \triangleright (\nu * xvec)(P \parallel Q) \sim P \parallel (\nu * xvec)Q$
 $\langle proof \rangle$

```

lemma bisimParAssoc:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 
  and    $R :: ('a, 'b, 'c) \psi$ 

```

shows $\Psi \triangleright (P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$
 $\langle proof \rangle$

```

lemma bisimParNil:
  fixes  $P :: ('a, 'b, 'c) \psi$ 

```

shows $\Psi \triangleright P \parallel \mathbf{0} \sim P$
 $\langle proof \rangle$

lemma bisimResNil:

```

fixes x :: name
and   Ψ :: 'b

shows Ψ ⊢ (⟨νx⟩)0 ~ 0
⟨proof⟩

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⟩

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⟩

lemma bisimCasePushRes:
fixes x :: name
and   Ψ :: 'b
and   Cs :: ('c × ('a, 'b, 'c) psi) list

assumes x ∉ (map fst Cs)

shows Ψ ⊢ (⟨νx⟩)(Cases Cs) ~ Cases(map (λ(φ, P). (φ, (⟨νx⟩)P)) Cs)
⟨proof⟩

lemma bangExt:
fixes Ψ :: 'b
and   P :: ('a, 'b, 'c) psi

assumes guarded P

```

shows $\Psi \triangleright !P \sim P \parallel !P$
 $\langle proof \rangle$

lemma *bisimParPresSym*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \sim Q$

shows $\Psi \triangleright R \parallel P \sim R \parallel Q$
 $\langle proof \rangle$

lemma *bisimScopeExtSym*:

fixes $x :: name$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$

assumes $x \notin \Psi$
and $x \notin Q$

shows $\Psi \triangleright (\nu x)(P \parallel Q) \sim ((\nu x)P) \parallel Q$
 $\langle proof \rangle$

lemma *bisimScopeExtChainSym*:

fixes $xvec :: name list$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$

assumes $xvec \#* \Psi$
and $xvec \#* Q$

shows $\Psi \triangleright (\nu * xvec)(P \parallel Q) \sim ((\nu * xvec)P) \parallel Q$
 $\langle proof \rangle$

lemma *bisimParPresAuxSym*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \otimes \Psi_R \triangleright P \sim Q$
and $extractFrame R = \langle A_R, \Psi_R \rangle$
and $A_R \#* \Psi$
and $A_R \#* P$
and $A_R \#* Q$

shows $\Psi \triangleright R \parallel P \sim R \parallel Q$
 $\langle proof \rangle$

lemma *bangDerivative*:

```
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $\alpha :: 'a \text{ action}$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright !P \xrightarrow{\alpha} P'$ 
and  $\Psi \triangleright P \sim Q$ 
and  $bn \alpha \sharp* \Psi$ 
and  $bn \alpha \sharp* P$ 
and  $bn \alpha \sharp* Q$ 
and  $bn \alpha \sharp* \text{subject } \alpha$ 
and  $guarded Q$ 
```

obtains $Q' R T$ **where** $\Psi \triangleright !Q \xrightarrow{\alpha} Q'$ **and** $\Psi \triangleright P' \sim R \parallel !P$ **and** $\Psi \triangleright Q' \sim T \parallel !Q$ **and** $\Psi \triangleright R \sim T$
and $((\text{supp } R)::\text{name set}) \subseteq \text{supp } P'$ **and** $((\text{supp } T)::\text{name set}) \subseteq \text{supp } Q'$
 $\langle proof \rangle$

lemma *structCongBisim*:

```
fixes  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
```

assumes $P \equiv_s Q$

shows $P \sim Q$

$\langle proof \rangle$

lemma *bisimBangPres*:

```
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
```

assumes $\Psi \triangleright P \sim Q$
and $guarded P$
and $guarded Q$

shows $\Psi \triangleright !P \sim !Q$

$\langle proof \rangle$

end

end

theory *Weak-Bisimulation*

```

imports Weak-Simulation Weak-Stat-Imp Bisim-Struct-Cong
begin

context env begin

lemma monoCoinduct:  $\bigwedge x y xa xb xc P Q \Psi$ .
   $x \leq y \implies$ 
   $(\Psi \triangleright Q \rightsquigarrow \langle \{xc, xb, xa\} \rangle P) \longrightarrow$ 
   $(\Psi \triangleright Q \rightsquigarrow \langle \{xb, xa, xc\} \rangle P)$ 
⟨proof⟩

lemma monoCoinduct2:  $\bigwedge x y xa xb xc P Q \Psi$ .
   $x \leq y \implies$ 
   $(\Psi \triangleright Q \widetilde{\rightsquigarrow} \langle \{xc, xb, xa\} \rangle P) \longrightarrow$ 
   $(\Psi \triangleright Q \widetilde{\rightsquigarrow} \langle \{xb, xa, xc\} \rangle P)$ 
⟨proof⟩

coinductive-set weakBisim :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set
where
  step:  $\llbracket \Psi \triangleright P \widetilde{\rightsquigarrow} \langle \text{weakBisim} \rangle Q; \Psi \triangleright P \rightsquigarrow \langle \text{weakBisim} \rangle Q;$ 
         $\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in \text{weakBisim}; (\Psi, Q, P) \in \text{weakBisim} \rrbracket \implies (\Psi, P, Q) \in \text{weakBisim}$ 
monos monoCoinduct monoCoinduct2

abbreviation
  weakBisimJudge ( $\cdot \triangleright \cdot \approx \cdot$  [70, 70, 70] 65) where  $\Psi \triangleright P \approx Q \equiv (\Psi, P, Q) \in \text{weakBisim}$ 
abbreviation
  weakBisimNilJudge ( $\cdot \approx \cdot$  [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 × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

  assumes  $(\Psi, P, Q) \in X$ 
  and  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi \triangleright P \widetilde{\rightsquigarrow} \langle X \cup \text{weakBisim} \rangle Q) \wedge$ 
         $(\Psi \triangleright P \rightsquigarrow \langle X \cup \text{weakBisim} \rangle Q) \wedge$ 
         $(\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in X \vee (\Psi \otimes \Psi', P, Q) \in \text{weakBisim}) \wedge$ 
         $((\Psi, Q, P) \in X \vee (\Psi, Q, P) \in \text{weakBisim})$ 

  shows  $(\Psi, P, Q) \in \text{weakBisim}$ 
⟨proof⟩

lemma weakBisimCoinduct[consumes 1, case-names cStatImp cSim cExt cSym]:
  fixes F :: 'b
  and P :: ('a, 'b, 'c) psi

```

and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \Psi' \triangleright R \lesssim_{(X \cup \text{weakBisim})} S$
and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \Psi' \triangleright R \rightsquigarrow_{(X \cup \text{weakBisim})} S$
and $\bigwedge \Psi' R S \Psi''. (\Psi', R, S) \in X \implies (\Psi' \otimes \Psi'', R, S) \in X \vee \Psi' \otimes \Psi'' \triangleright R \approx S$
and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies (\Psi', S, R) \in X \vee \Psi' \triangleright S \approx R$

shows $\Psi \triangleright P \approx Q$
(proof)

lemma *weakBisimWeakCoinductAux[consumes 1]*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_{X} Q \wedge \Psi \triangleright P \rightsquigarrow_{X} Q \wedge$
 $(\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in X) \wedge (\Psi, Q, P) \in X$

shows $\Psi \triangleright P \approx Q$
(proof)

lemma *weakBisimWeakCoinduct[consumes 1, case-names cStatImp cSim cExt cSym]*:

fixes $F :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $X :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$

assumes $(\Psi, P, Q) \in X$
and $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_{X} Q$
and $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow_{X} Q$
and $\bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X$
and $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X$

shows $(\Psi, P, Q) \in \text{weakBisim}$
(proof)

lemma *weakBisimE*:

fixes $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $\Psi :: 'b$
and $\Psi' :: 'b$

assumes $\Psi \triangleright P \approx Q$

```

shows  $\Psi \triangleright P \lesssim_{\text{weakBisim}} Q$ 
and  $\Psi \triangleright P \rightsquigarrow_{\text{weakBisim}} Q$ 
and  $\Psi \otimes \Psi' \triangleright P \approx Q$ 
and  $\Psi \triangleright Q \approx P$ 
⟨proof⟩

```

```

lemma weakBisimI:
fixes  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
and  $\Psi :: 'b$ 

assumes  $\Psi \triangleright P \lesssim_{\text{weakBisim}} Q$ 
and  $\Psi \triangleright P \rightsquigarrow_{\text{weakBisim}} Q$ 
and  $\forall \Psi'. \Psi \otimes \Psi' \triangleright P \approx Q$ 
and  $\Psi \triangleright Q \approx P$ 

shows  $\Psi \triangleright P \approx Q$ 
⟨proof⟩

```

```

lemma weakBisimReflexive:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 

```

```

shows  $\Psi \triangleright P \approx P$ 
⟨proof⟩

```

```

lemma weakBisimClosed:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
and  $p :: \text{name} \text{ prm}$ 

```

```

assumes  $\Psi \triangleright P \approx Q$ 

shows  $(p \cdot \Psi) \triangleright (p \cdot P) \approx (p \cdot Q)$ 
⟨proof⟩

```

```

lemma weakBisimEqvt[simp]:
shows eqvt weakBisim
⟨proof⟩

```

```

lemma statEqWeakBisim:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
and  $\Psi' :: 'b$ 

```

```

assumes  $\Psi \triangleright P \approx Q$ 

```

and $\Psi \simeq \Psi'$

shows $\Psi' \triangleright P \approx Q$
 $\langle proof \rangle$

lemma *weakBisimTransitive*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $PQ: \Psi \triangleright P \approx Q$
and $QR: \Psi \triangleright Q \approx R$

shows $\Psi \triangleright P \approx R$
 $\langle proof \rangle$

lemma *strongBisimWeakBisim*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \sim Q$

shows $\Psi \triangleright P \approx Q$
 $\langle proof \rangle$

lemma *structCongWeakBisim*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$

assumes $P \equiv_s Q$

shows $P \approx Q$
 $\langle proof \rangle$

lemma *simTauChain*:

fixes $\Psi :: 'b$
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 $\Psi \triangleright Q \Rightarrow \hat{\tau} Q'$
and $Sim: \bigwedge \Psi P Q. (\Psi, P, Q) \in Rel \Rightarrow \Psi \triangleright P \rightsquigarrow [Rel] Q$

obtains P' **where** $\Psi \triangleright P \Rightarrow \hat{\tau} P'$ **and** $(\Psi, P', Q') \in Rel$
 $\langle proof \rangle$

```

lemma quietBisimNil:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  assumes quiet  $P$ 

  shows  $\Psi \triangleright P \approx \mathbf{0}$ 
   $\langle proof \rangle$ 

lemma weakTransitiveWeakCoinduct[case-names cStatImp cSim cExt cSym, case-conclusion
bisim step, consumes 2]:
  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  $p: (\Psi, P, Q) \in X$ 
  and Eqvt: eqvt  $X$ 
  and rStatImp:  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_{\langle X \rangle} Q$ 
  and rSim:  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow \langle \{(\Psi, P, Q) \mid \Psi P Q. \exists P'. \Psi \triangleright P \sim P' \wedge$ 
 $\quad (\Psi, P', Q') \in X \wedge$ 
 $\quad \Psi \triangleright Q' \sim Q \} \rangle Q$ 
  and rExt:  $\bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X$ 
  and rSym:  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X$ 

  shows  $\Psi \triangleright P \approx Q$ 
   $\langle proof \rangle$ 

lemma weakTransitiveCoinduct[case-names cStatImp cSim cExt cSym, case-conclusion
bisim step, consumes 2]:
  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  $p: (\Psi, P, Q) \in X$ 
  and Eqvt: eqvt  $X$ 
  and rStatImp:  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_{\langle (X \cup weakBisim) \rangle} Q$ 
  and rSim:  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow \langle \{(\Psi, P, Q) \mid \Psi P Q. \exists P'. \Psi \triangleright P \sim P' \wedge$ 
 $\quad (\Psi, P', Q') \in (X \cup$ 
 $\quad weakBisim) \wedge$ 
 $\quad \Psi \triangleright Q' \sim Q \} \rangle Q$ 
  and rExt:  $\bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X \cup weakBisim$ 
  and rSym:  $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X \cup weakBisim$ 

  shows  $\Psi \triangleright P \approx Q$ 

```

$\langle proof \rangle$

lemma *weakTransitiveCoinduct2*[*case-names cStatImp cSim cExt cSym, case-conclusion bisim step, consumes 2*]:

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 $p: (\Psi, P, Q) \in X$

and $Egvt: eqvt X$

and $rStatImp: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_{\sim} Q$

and $rSim: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow \{(\Psi, P, Q) \mid \Psi P Q. \exists P' Q'. \Psi \triangleright P \approx P' \wedge (\Psi, P', Q') \in X \wedge \Psi \triangleright Q' \sim Q\} \triangleright Q$

and $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X$

and $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X$

shows $\Psi \triangleright P \approx Q$

$\langle proof \rangle$

end

end

theory *Weak-Sim-Pres*

imports *Sim-Pres Weak-Simulation Weak-Stat-Imp*

begin

context *env* **begin**

lemma *weakInputPres*:

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 $M :: 'a$

and $xvec :: name list$

and $N :: 'a$

assumes $PRelQ: \bigwedge Tvec \Psi'. length xvec = length Tvec \implies (\Psi \otimes \Psi', P[xvec:=Tvec], Q[xvec:=Tvec]) \in Rel$

shows $\Psi \triangleright M(\lambda*xvec N).P \rightsquigarrow_{Rel} M(\lambda*xvec N).Q$

$\langle proof \rangle$

lemma *weakOutputPres*:

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   M   :: 'a
and   N   :: 'a

assumes PRelQ:  $\bigwedge \Psi'. (\Psi \otimes \Psi', P, Q) \in Rel$ 

shows  $\Psi \triangleright M\langle N \rangle.P \rightsquigarrow_{Rel} M\langle N \rangle.Q$ 
⟨proof⟩

lemma resTauCases[consumes 3, case-names cRes]:
  fixes  $\Psi$    :: 'b
  and   x   :: name
  and   P   :: ('a, 'b, 'c) psi
  and   P'  :: ('a, 'b, 'c) psi
  and   C   :: 'd::fs-name

  assumes Trans:  $\Psi \triangleright (\nu x)P \mapsto_{\tau} \prec P'$ 
  and   x  $\notin \Psi$ 
  and   x  $\notin P'$ 
  and   rScope:  $\bigwedge P'. [\Psi \triangleright P \mapsto_{\tau} \prec P'] \implies Prop((\nu x)P')$ 

  shows Prop P'
⟨proof⟩

lemma weakResPres:
  fixes  $\Psi$    :: '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   Rel' :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

  assumes PSimQ:  $\Psi \triangleright P \rightsquigarrow_{Rel} Q$ 
  and   eqvt Rel'
  and   x  $\notin \Psi$ 
  and   Rel  $\subseteq$  Rel'
  and   C1:  $\bigwedge \Psi' R S y. [(\Psi', R, S) \in Rel; y \notin \Psi] \implies (\Psi', (\nu y)R, (\nu y)S) \in Rel'$ 

  shows  $\Psi \triangleright (\nu x)P \rightsquigarrow_{Rel'} (\nu x)Q$ 
⟨proof⟩

lemma weakResChainPres:
  fixes  $\Psi$    :: '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   xvec :: name list

```

assumes $PSimQ: \Psi \triangleright P \rightsquigarrow_{Rel} Q$
and $eqvt Rel$
and $xvec \#* \Psi$
and $C1: \bigwedge \Psi' R S yvec. \llbracket (\Psi', R, S) \in Rel; yvec \#* \Psi' \rrbracket \implies (\Psi', (\nu * yvec) R, (\nu * yvec) S) \in Rel$

shows $\Psi \triangleright (\nu * xvec) P \rightsquigarrow_{Rel} (\nu * xvec) Q$
(proof)

lemma $parTauCases[consumes 1, case-names cPar1 cPar2 cComm1 cComm2]:$

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Q :: ('a, 'b, 'c) psi$
and $R :: ('a, 'b, 'c) psi$
and $C :: 'd::fs-name$

assumes $Trans: \Psi \triangleright P \parallel Q \xrightarrow{\tau} \prec R$
and $rPar1: \bigwedge P' A_Q \Psi_Q. \llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{\tau} \prec P'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q; A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* C \rrbracket \implies Prop (P' \parallel Q)$
and $rPar2: \bigwedge Q' A_P \Psi_P. \llbracket \Psi \otimes \Psi_P \triangleright Q \xrightarrow{\tau} \prec Q'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P; A_Q \#* \Psi; A_Q \#* P; A_Q \#* Q; A_Q \#* C \rrbracket \implies Prop (P \parallel Q')$
and $rComm1: \bigwedge \Psi_Q M N P' A_P \Psi_P K xvec Q' A_Q.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(N)} \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(\nu * xvec)(N)} \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; distinct xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q; xvec \#* \Psi_Q; xvec \#* C \rrbracket \implies Prop ((\nu * xvec)(P' \parallel Q'))$
and $rComm2: \bigwedge \Psi_Q M xvec N P' A_P \Psi_P K Q' A_Q.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \xrightarrow{M(\nu * xvec)(N)} \prec P'; extractFrame P = \langle A_P, \Psi_P \rangle; distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \xrightarrow{K(N)} \prec Q'; extractFrame Q = \langle A_Q, \Psi_Q \rangle; distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; distinct xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q; xvec \#* \Psi_Q; xvec \#* C \rrbracket \implies Prop ((\nu * xvec)(P' \parallel Q'))$

shows $Prop R$

$\langle proof \rangle$

```

lemma weakParPres:
  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  $R :: ('a, 'b, 'c) \psi$ 
  and  $Rel' :: ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) set$ 

  assumes  $PRelQ: \bigwedge A_R \Psi_R. \llbracket extractFrame R = \langle A_R, \Psi_R \rangle; A_R \#* \Psi; A_R \#* P; A_R \#* Q \rrbracket \implies (\Psi \otimes \Psi_R, P, Q) \in Rel$ 
  and  $Eqvt: eqvt Rel$ 
  and  $Eqvt': eqvt Rel'$ 

  and  $Sim: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies \Psi' \triangleright S \rightsquigarrow_{Rel} T$ 
  and  $Sym: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies (\Psi', T, S) \in Rel$ 
  and  $Ext: \bigwedge \Psi' S T \Psi'. \llbracket (\Psi', S, T) \in Rel \rrbracket \implies (\Psi' \otimes \Psi'', S, T) \in Rel$ 
  and  $StatImp: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies \Psi' \triangleright S \lesssim_{Rel} T$ 
  and  $C1: \bigwedge \Psi' S T A_U \Psi_U U. \llbracket (\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = \langle A_U, \Psi_U \rangle; A_U \#* \Psi'; A_U \#* S; A_U \#* T \rrbracket \implies (\Psi', S \parallel U, T \parallel U) \in Rel'$ 
  and  $C2: \bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel'; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in Rel'$ 
  and  $C3: \bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', S, T) \in Rel$ 

  shows  $\Psi \triangleright P \parallel R \rightsquigarrow_{Rel'} Q \parallel R$ 

```

$\langle proof \rangle$

```

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  $Rel' \subseteq Rel$ 

  and  $FrameParPres: \bigwedge \Psi' \Psi_U S T U A_U. \llbracket (\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = \langle A_U, \Psi_U \rangle; A_U \#* \Psi'; A_U \#* S; A_U \#* T \rrbracket \implies (\Psi', U \parallel S, U \parallel T) \in Rel$ 
  and  $C1: \bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel; guarded S; guarded T \rrbracket \implies (\Psi', U \parallel !S, U \parallel !T) \in Rel''$ 
  and  $ResPres: \bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in Rel$ 

```

```

and      ResPres':  $\bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel'; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in Rel'$ 

and      Closed:  $\bigwedge \Psi' S T p. (\Psi', S, T) \in Rel \implies ((p::name\ prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel$ 
and      Closed':  $\bigwedge \Psi' S T p. (\Psi', S, T) \in Rel' \implies ((p::name\ prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel'$ 
and      StatEq:  $\bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', S, T) \in Rel$ 
and      StatEq':  $\bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel'; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', S, T) \in Rel'$ 
and      Trans:  $\bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel; (\Psi', T, U) \in Rel \rrbracket \implies (\Psi', S, U) \in Rel$ 
and      Trans':  $\bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel'; (\Psi', T, U) \in Rel' \rrbracket \implies (\Psi', S, U) \in Rel'$ 

and      cSim:  $\bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies \Psi' \triangleright S \rightsquigarrow_{Rel} T$ 
and      cExt:  $\bigwedge \Psi' S T \Psi''. (\Psi', S, T) \in Rel \implies (\Psi' \otimes \Psi'', S, T) \in Rel$ 
and      cExt':  $\bigwedge \Psi' S T \Psi''. (\Psi', S, T) \in Rel' \implies (\Psi' \otimes \Psi'', S, T) \in Rel'$ 
and      cSym:  $\bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies (\Psi', T, S) \in Rel$ 
and      cSym':  $\bigwedge \Psi' S T. (\Psi', S, T) \in Rel' \implies (\Psi', T, S) \in Rel'$ 

and      ParPres:  $\bigwedge \Psi' S T U. (\Psi', S, T) \in Rel \implies (\Psi', S \parallel U, T \parallel U) \in Rel$ 
and      ParPres2:  $\bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies (\Psi', S \parallel S, T \parallel T) \in Rel$ 
and      ParPres':  $\bigwedge \Psi' S T U. (\Psi', S, T) \in Rel' \implies (\Psi', U \parallel S, U \parallel T) \in Rel'$ 

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:  $\bigwedge xvec \Psi' T S. \llbracket xvec \#* \Psi'; xvec \#* T \rrbracket \implies (\Psi', (\nu*xvec)(S \parallel T), ((\nu*xvec)S) \parallel T) \in Rel$ 
and      ScopeExt':  $\bigwedge xvec \Psi' T S. \llbracket xvec \#* \Psi'; xvec \#* T \rrbracket \implies (\Psi', (\nu*xvec)(S \parallel T), ((\nu*xvec)S) \parallel T) \in Rel'$ 

and      Compose:  $\bigwedge \Psi' S T U O. \llbracket (\Psi', S, T) \in Rel; (\Psi', T, U) \in Rel''; (\Psi', U, O) \in Rel' \rrbracket \implies (\Psi', S, O) \in Rel''$ 

and      rBangActE:  $\bigwedge \Psi' S \alpha S'. \llbracket \Psi' \triangleright !S \mapsto \alpha \prec S'; guarded S; bn \alpha \#* S; \alpha \neq \tau; bn \alpha \#* subject \alpha \rrbracket \implies \exists T. \Psi' \triangleright S \mapsto \alpha \prec T \wedge (\mathbf{1}, S', T \parallel !S) \in Rel'$ 
and      rBangTauE:  $\bigwedge \Psi' S S'. \llbracket \Psi' \triangleright !S \mapsto \tau \prec S'; guarded S \rrbracket \implies \exists T. \Psi' \triangleright S \parallel S \mapsto \tau \prec T \wedge (\mathbf{1}, S', T \parallel !S) \in Rel'$ 
and      rBangTauI:  $\bigwedge \Psi' S S'. \llbracket \Psi' \triangleright S \parallel S \implies \hat{\tau} S'; guarded S \rrbracket \implies \exists T. \Psi' \triangleright !S \implies \hat{\tau} T \wedge (\Psi', T, S' \parallel !S) \in Rel'$ 
shows  $\Psi \triangleright R \parallel !P \rightsquigarrow_{Rel''} R \parallel !Q$ 
{proof}
unbundle relcomp-syntax
end

end

```

```

theory Weak-Stat-Imp-Pres
imports Weak-Stat-Imp
begin

context env begin

lemma weakStatImpInputPres:
fixes  $\Psi$  :: ' $b$ 
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  $M$  :: ' $a$ 
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  $\Psi \triangleright M(\lambda*xvec N).P \lesssim_{\text{Rel}} M(\lambda*xvec N).Q$ 
⟨proof⟩

lemma weakStatImpOutputPres:
fixes  $\Psi$  :: ' $b$ 
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  $M$  :: ' $a$ 
and  $N$  :: ' $a$ 

assumes  $PRelQ: \bigwedge \Psi'. (\Psi \otimes \Psi', M\langle N \rangle.P, M\langle N \rangle.Q) \in Rel$ 

shows  $\Psi \triangleright M\langle N \rangle.P \lesssim_{\text{Rel}} M\langle N \rangle.Q$ 
⟨proof⟩

lemma weakStatImpResPres:
fixes  $\Psi$  :: ' $b$ 
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  $Rel'$  :: (' $b$  × (' $a$ , ' $b$ , ' $c$ )  $\psi$  × (' $a$ , ' $b$ , ' $c$ )  $\psi$ ) set

assumes  $PSimQ: \Psi \triangleright P \lesssim_{\text{Rel}} Q$ 
and  $eqvt Rel$ 
and  $x \notin \Psi$ 
and  $C1: \bigwedge \Psi' R S y. [( \Psi', R, S ) \in Rel; y \notin \Psi] \implies (\Psi', (\nu y)R, (\nu y)S) \in Rel'$ 

shows  $\Psi \triangleright (\nu x)P \lesssim_{\text{Rel}'} (\nu x)Q$ 
⟨proof⟩

```

```

lemma weakStatImpParPres:
  fixes  $\Psi$  :: '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  $PStatImpQ: \bigwedge A_R \Psi_R. [[extractFrame R = \langle A_R, \Psi_R \rangle; A_R \#* \Psi; A_R \#* P; A_R \#* Q]] \implies \Psi \otimes \Psi_R \triangleright P \lesssim_{\langle Rel \rangle} Q$ 
  and  $xvec \#* \Psi$ 
  and  $Eqvt: eqvt Rel$ 

  and  $C1: \bigwedge \Psi' S T A_U \Psi_U U. [(\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = \langle A_U, \Psi_U \rangle; A_U \#* \Psi'; A_U \#* S; A_U \#* T] \implies (\Psi', S \parallel U, T \parallel U) \in Rel'$ 
  and  $C2: \bigwedge \Psi' S T yvec. [(\Psi', S, T) \in Rel'; yvec \#* \Psi] \implies (\Psi', (\nu * yvec) S, (\nu * yvec) T) \in Rel'$ 
  and  $C3: \bigwedge \Psi' S T \Psi''. [(\Psi', S, T) \in Rel; \Psi' \simeq \Psi''] \implies (\Psi'', S, T) \in Rel$ 

  shows  $\Psi \triangleright (\nu * xvec)(P \parallel R) \lesssim_{\langle Rel' \rangle} (\nu * xvec)(Q \parallel R)$ 
   $\langle proof \rangle$ 

end

end

theory Weak-Bisim-Pres
  imports Weak-Bisimulation Weak-Sim-Pres Weak-Stat-Imp-Pres
begin

context env begin

lemma weakBisimInputPres:
  fixes  $\Psi$  :: 'b
  and  $P$  :: ('a, 'b, 'c) psi
  and  $Q$  :: ('a, 'b, 'c) psi
  and  $M$  :: 'a
  and  $xvec$  :: name list
  and  $N$  :: 'a

  assumes  $\bigwedge Tvec. length xvec = length Tvec \implies \Psi \triangleright P[xvec ::= Tvec] \approx Q[xvec ::= Tvec]$ 

  shows  $\Psi \triangleright M(\lambda * xvec N).P \approx M(\lambda * xvec N).Q$ 
   $\langle proof \rangle$ 

lemma weakBisimOutputPres:
  fixes  $\Psi$  :: 'b
  and  $P$  :: ('a, 'b, 'c) psi

```

```

and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $M :: 'a$ 
and    $xvec :: \text{name list}$ 
and    $N :: 'a$ 

```

assumes $\Psi \triangleright P \approx Q$

shows $\Psi \triangleright M\langle N \rangle.P \approx M\langle N \rangle.Q$
 $\langle \text{proof} \rangle$

lemma *weakBisimResPres*:

```

fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $x :: \text{name}$ 

```

assumes $\Psi \triangleright P \approx Q$
and $x \notin \Psi$

shows $\Psi \triangleright (\nu x)P \approx (\nu x)Q$
 $\langle \text{proof} \rangle$

lemma *weakBisimResChainPres*:

```

fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $xvec :: \text{name list}$ 

```

assumes $\Psi \triangleright P \approx Q$
and $xvec \nparallel \Psi$

shows $\Psi \triangleright (\nu * xvec)P \approx (\nu * xvec)Q$
 $\langle \text{proof} \rangle$

lemma *weakBisimParPresAux*:

```

fixes  $\Psi :: 'b$ 
and    $\Psi_R :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $R :: ('a, 'b, 'c) \text{ psi}$ 
and    $A_R :: \text{name list}$ 

```

assumes $\Psi \otimes \Psi_R \triangleright P \approx Q$
and $\text{FrR}: \text{extractFrame } R = \langle A_R, \Psi_R \rangle$
and $A_R \nparallel \Psi$
and $A_R \nparallel P$
and $A_R \nparallel Q$

shows $\Psi \triangleright P \parallel R \approx Q \parallel R$

```

⟨proof⟩

lemma weakBisimParPres:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $R :: ('a, 'b, 'c) \psi$ 

  assumes  $\Psi \triangleright P \approx Q$ 

  shows  $\Psi \triangleright P \parallel R \approx Q \parallel R$ 
⟨proof⟩

end

end

theory Weak-Bisim-Struct-Cong
  imports Weak-Bisim-Pres Bisim-Struct-Cong
begin

context env begin

lemma weakBisimParComm:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \parallel Q \approx Q \parallel P$ 
⟨proof⟩

lemma weakBisimResComm:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $y :: name$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  assumes  $x \notin \Psi$ 
  and  $y \notin \Psi$ 

  shows  $\Psi \triangleright (\nu x)(\nu y)P \approx (\nu y)(\nu x)P$ 
⟨proof⟩

lemma weakBisimResComm':
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $xvec :: name list$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

```

```

assumes  $x \notin \Psi$ 
and  $xvec \notin \Psi$ 

shows  $\Psi \triangleright (\nu x)(\nu*xvec)P \approx (\nu*xvec)(\nu x)P$ 
(proof)

lemma weakBisimScopeExt:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $x \notin \Psi$ 
  and  $x \notin P$ 

  shows  $\Psi \triangleright (\nu x)(P \parallel Q) \approx P \parallel (\nu x)Q$ 
(proof)

lemma weakBisimScopeExtChain:
  fixes  $xvec :: name list$ 
  and  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $xvec \notin \Psi$ 
  and  $xvec \notin P$ 

  shows  $\Psi \triangleright (\nu*xvec)(P \parallel Q) \approx P \parallel (\nu*xvec)Q$ 
(proof)

lemma weakBisimParAssoc:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $R :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright (P \parallel Q) \parallel R \approx P \parallel (Q \parallel R)$ 
(proof)

lemma weakBisimParNil:
  fixes  $P :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \parallel \mathbf{0} \approx P$ 
(proof)

lemma weakBisimResNil:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 

```

```
assumes  $x \notin \Psi$ 
```

```
shows  $\Psi \triangleright (\nu x) \mathbf{0} \approx \mathbf{0}$ 
⟨proof⟩
```

```
lemma weakBisimOutputPushRes:
```

```
fixes  $x :: name$ 
```

```
and  $\Psi :: 'b$ 
```

```
and  $M :: 'a$ 
```

```
and  $N :: 'a$ 
```

```
and  $P :: ('a, 'b, 'c) \psi$ 
```

```
assumes  $x \notin \Psi$ 
```

```
and  $x \notin M$ 
```

```
and  $x \notin N$ 
```

```
shows  $\Psi \triangleright (\nu x) (M \langle N \rangle . P) \approx M \langle N \rangle . (\nu x) P$ 
```

```
⟨proof⟩
```

```
lemma weakBisimInputPushRes:
```

```
fixes  $x :: name$ 
```

```
and  $\Psi :: 'b$ 
```

```
and  $M :: 'a$ 
```

```
and  $xvec :: name list$ 
```

```
and  $N :: 'a$ 
```

```
and  $P :: ('a, 'b, 'c) \psi$ 
```

```
assumes  $x \notin \Psi$ 
```

```
and  $x \notin M$ 
```

```
and  $x \notin xvec$ 
```

```
and  $x \notin N$ 
```

```
shows  $\Psi \triangleright (\nu x) (M (\lambda * xvec N) . P) \approx M (\lambda * xvec N) . (\nu x) P$ 
```

```
⟨proof⟩
```

```
lemma weakBisimCasePushRes:
```

```
fixes  $x :: name$ 
```

```
and  $\Psi :: 'b$ 
```

```
and  $Cs :: ('c \times ('a, 'b, 'c) \psi) list$ 
```

```
assumes  $x \notin \Psi$ 
```

```
and  $x \notin (map fst Cs)$ 
```

```
shows  $\Psi \triangleright (\nu x) (Cases Cs) \approx Cases (map (\lambda(\varphi, P). (\varphi, (\nu x) P)) Cs)$ 
```

```
⟨proof⟩
```

```
lemma weakBangExt:
```

```
fixes  $\Psi :: 'b$ 
```

```
and  $P :: ('a, 'b, 'c) \psi$ 
```

assumes *guarded P*

shows $\Psi \triangleright !P \approx P \parallel !P$
(proof)

lemma *weakBisimParSym*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \approx Q$

shows $\Psi \triangleright R \parallel P \approx R \parallel Q$
(proof)

lemma *weakBisimScopeExtSym*:

fixes $x :: name$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$

assumes $x \notin \Psi$
and $x \notin Q$

shows $\Psi \triangleright (\nu x)(P \parallel Q) \approx ((\nu x)P) \parallel Q$
(proof)

lemma *weakBisimScopeExtChainSym*:

fixes $xvec :: name list$
and $Q :: ('a, 'b, 'c) \psi$
and $P :: ('a, 'b, 'c) \psi$

assumes $xvec \notin \Psi$
and $xvec \notin Q$

shows $\Psi \triangleright (\nu * xvec)(P \parallel Q) \approx ((\nu * xvec)P) \parallel Q$
(proof)

lemma *weakBisimParPresAuxSym*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \otimes \Psi_R \triangleright P \approx Q$
and $extractFrame R = \langle A_R, \Psi_R \rangle$
and $A_R \notin \Psi$
and $A_R \notin P$

and $A_R \sharp* Q$

shows $\Psi \triangleright R \parallel P \approx R \parallel Q$
 $\langle proof \rangle$

lemma *weakBisimParPresSym*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \approx Q$

shows $\Psi \triangleright R \parallel P \approx R \parallel Q$
 $\langle proof \rangle$

lemma *guardedFrameStatEq*:

fixes $P :: ('a, 'b, 'c) \psi$

assumes *guarded P*

shows *extractFrame P* $\simeq_F \langle \varepsilon, 1 \rangle$
 $\langle proof \rangle$

lemma *guardedInsertAssertion*:

fixes $P :: ('a, 'b, 'c) \psi$
and $\Psi :: 'b$

assumes *guarded P*

shows *insertAssertion (extractFrame P)* $\Psi \simeq_F \langle \varepsilon, \Psi \rangle$
 $\langle proof \rangle$

lemma *insertDoubleAssertionStatEq'*:

fixes $F :: 'b frame$
and $\Psi :: 'b$
and $\Psi' :: 'b$

shows *insertAssertion(insertAssertion F Ψ)* $\Psi' \simeq_F (insertAssertion F) (\Psi' \otimes \Psi)$
 $\langle proof \rangle$

lemma *bangActE*:

assumes $\Psi \triangleright !P \longmapsto^\alpha \prec P'$
and $bn \alpha \sharp* subject \alpha$
and *guarded P*
and $\alpha \neq \tau$
and $bn \alpha \sharp* P$

obtains Q **where** $\Psi \triangleright P \xrightarrow{\alpha} Q$ **and** $P' \sim Q \parallel !P$
 $\langle proof \rangle$

lemma *bangTauE*:
assumes $\Psi \triangleright !P \xrightarrow{\tau} P'$
and *guarded P*

obtains Q **where** $\Psi \triangleright P \parallel P \xrightarrow{\tau} Q$ **and** $P' \sim Q \parallel !P$
 $\langle proof \rangle$

lemma *tauBangI*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $P' :: ('a, 'b, 'c) \psi$
assumes $\Psi \triangleright P \parallel P \xrightarrow{\tau} P'$
and *guarded P*

obtains Q **where** $\Psi \triangleright !P \xrightarrow{\tau} Q$ **and** $Q \sim P' \parallel !P$
 $\langle proof \rangle$

lemma *tauChainBangI*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $P' :: ('a, 'b, 'c) \psi$
assumes $\Psi \triangleright P \parallel P \xrightarrow{\tau} P'$
and *guarded P*

obtains Q **where** $\Psi \triangleright !P \xrightarrow{\tau} Q$ **and** $\Psi \triangleright Q \sim P' \parallel !P$
 $\langle proof \rangle$

lemma *weakBisimBangPresAux*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$
assumes $\Psi \triangleright P \approx Q$
and *guarded P*
and *guarded Q*

shows $\Psi \triangleright R \parallel !P \approx R \parallel !Q$
 $\langle proof \rangle$

lemma *weakBisimBangPres*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$

```

and    $Q :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \approx Q$ 
and    guarded P
and    guarded Q

shows   $\Psi \triangleright !P \approx !Q$ 
(proof)

end

end

theory Close-Subst
imports Agent
begin

context substPsi
begin

definition closeSubst ::  $('b::fs\text{-}name \times ('a::fs\text{-}name, 'b, 'c::fs\text{-}name) \psi \times ('a, 'b, 'c) \psi) set \Rightarrow ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) set$ 
where closeSubst Rel  $\equiv \{(\Psi, P, Q) \mid \Psi \triangleright P \approx Q. (\forall \sigma. wellFormedSubst \sigma \longrightarrow (\Psi, P[<\sigma>], Q[<\sigma>]) \in Rel\}$ 

lemma closeSubstI:
fixes  $\Psi :: 'b$ 
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  $(\Psi, P, Q) \in closeSubst Rel$ 
(proof)

lemma closeSubstE:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $Q :: ('a, 'b, 'c) \psi$ 
and    $\sigma :: (name\ list \times 'a\ list)\ list$ 

assumes  $(\Psi, P, Q) \in closeSubst Rel$ 
and    wellFormedSubst σ

shows  $(\Psi, P[<\sigma>], Q[<\sigma>]) \in Rel$ 
(proof)

lemma closeSubstClosed:
fixes  $\Psi :: 'b$ 

```

```

and    $P :: ('a, 'b, 'c) \psi$ 
and    $Q :: ('a, 'b, 'c) \psi$ 
and    $p :: \text{name} \text{ prm}$ 

assumes  $\text{eqvt Rel}$ 
and      $(\Psi, P, Q) \in \text{closeSubst Rel}$ 

shows  $(p \cdot \Psi, p \cdot P, p \cdot Q) \in \text{closeSubst Rel}$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{closeSubstEqvt}:$ 
assumes  $\text{eqvt Rel}$ 

shows  $\text{eqvt}(\text{closeSubst Rel})$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{closeSubstUnfold}:$ 
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $Q :: ('a, 'b, 'c) \psi$ 
and    $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$ 

assumes  $(\Psi, P, Q) \in \text{closeSubst Rel}$ 
and      $\text{wellFormedSubst } \sigma$ 

shows  $(\Psi, P[<\sigma>], Q[<\sigma>]) \in \text{closeSubst Rel}$ 
 $\langle \text{proof} \rangle$ 

end

end

theory Bisim-Subst
imports Bisim-Struct-Cong Close-Subst
begin

context env begin

abbreviation
  bisimSubstJudge ( $\langle - \triangleright - \sim_s \rightarrow [70, 70, 70] \rangle 65$ ) where  $\Psi \triangleright P \sim_s Q \equiv (\Psi, P, Q) \in \text{closeSubst bisim}$ 
abbreviation
  bisimSubstNilJudge ( $\langle - \sim_s \rightarrow [70, 70] \rangle 65$ ) 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  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \text{ psi}$ 
and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
and  $M :: 'a$ 
and  $N :: 'a$ 

assumes  $\Psi \triangleright P \sim_s Q$ 

shows  $\Psi \triangleright M\langle N \rangle.P \sim_s M\langle N \rangle.Q$ 
⟨proof⟩

lemma seqSubstInputChain[simp]:
fixes  $xvec :: \text{name list}$ 
and  $N :: 'a$ 
and  $P :: ('a, 'b, 'c) \text{ psi}$ 
and  $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$ 

assumes  $xvec \sharp* \sigma$ 

shows  $\text{seqSubs}'(\text{inputChain } xvec \ N \ P) \ \sigma = \text{inputChain } xvec \ (\text{substTerm.seqSubst } N \ \sigma) \ (\text{seqSubs } P \ \sigma)$ 
⟨proof⟩

lemma bisimSubstInputPres:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \text{ psi}$ 
and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
and  $M :: 'a$ 
and  $xvec :: \text{name list}$ 
and  $N :: 'a$ 

assumes  $\Psi \triangleright P \sim_s Q$ 
and  $xvec \sharp* \Psi$ 
and  $\text{distinct } xvec$ 

shows  $\Psi \triangleright M(\lambda*xvec \ N).P \sim_s M(\lambda*xvec \ N).Q$ 
⟨proof⟩

lemma bisimSubstCasePresAux:
fixes  $\Psi :: 'b$ 
and  $CsP :: ('c \times ('a, 'b, 'c) \text{ psi}) \text{ list}$ 
and  $CsQ :: ('c \times ('a, 'b, 'c) \text{ psi}) \text{ list}$ 

assumes  $C1: \bigwedge \varphi \ P. (\varphi, P) \ \text{mem} \ CsP \implies \exists Q. (\varphi, Q) \ \text{mem} \ CsQ \wedge \text{guarded } Q$ 
 $\wedge \Psi \triangleright P \sim_s Q$ 
and  $C2: \bigwedge \varphi \ Q. (\varphi, Q) \ \text{mem} \ CsQ \implies \exists P. (\varphi, P) \ \text{mem} \ CsP \wedge \text{guarded } P \wedge$ 
 $\Psi \triangleright P \sim_s Q$ 

```

shows $\Psi \triangleright \text{Cases } CsP \sim_s \text{Cases } CsQ$
 $\langle proof \rangle$

lemma *bisimSubstReflexive*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$

shows $\Psi \triangleright P \sim_s P$
 $\langle proof \rangle$

lemma *bisimSubstTransitive*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \sim_s Q$
and $\Psi \triangleright Q \sim_s R$

shows $\Psi \triangleright P \sim_s R$
 $\langle proof \rangle$

lemma *bisimSubstSymmetric*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \sim_s Q$

shows $\Psi \triangleright Q \sim_s P$
 $\langle proof \rangle$

lemma *bisimSubstParPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \sim_s Q$

shows $\Psi \triangleright P \parallel R \sim_s Q \parallel R$
 $\langle proof \rangle$

lemma *bisimSubstResPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $x :: \text{name}$

```

assumes  $\Psi \triangleright P \sim_s Q$ 
and  $x \notin \Psi$ 

shows  $\Psi \triangleright (\nu x)P \sim_s (\nu x)Q$ 
(proof)

lemma bisimSubstBangPres:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \sim_s Q$ 
and guarded  $P$ 
and guarded  $Q$ 

shows  $\Psi \triangleright !P \sim_s !Q$ 
(proof)

lemma substNil[simp]:
fixes  $xvec :: name list$ 
and  $Tvec :: 'a list$ 

assumes wellFormedSubst  $\sigma$ 
and distinct  $xvec$ 

shows  $(\mathbf{0}[\langle \sigma \rangle]) = \mathbf{0}$ 
(proof)

lemma bisimSubstParNil:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 

shows  $\Psi \triangleright P \parallel \mathbf{0} \sim_s P$ 
(proof)

lemma bisimSubstParComm:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

shows  $\Psi \triangleright P \parallel Q \sim_s Q \parallel P$ 
(proof)

lemma bisimSubstParAssoc:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
and  $R :: ('a, 'b, 'c) \psi$ 

```

shows $\Psi \triangleright (P \parallel Q) \parallel R \sim_s P \parallel (Q \parallel R)$
 $\langle proof \rangle$

lemma *bisimSubstResNil*:

fixes $\Psi :: 'b$
and $x :: name$

shows $\Psi \triangleright (\nu x) 0 \sim_s 0$
 $\langle proof \rangle$

lemma *seqSubst2*:

fixes $x :: name$
and $P :: ('a, 'b, 'c) \psi$

assumes *wellFormedSubst* σ

and $x \notin \sigma$
and $x \notin P$

shows $x \notin P[<\sigma>]$
 $\langle proof \rangle$

notation *substTerm.seqSubst* ($\langle -[->] \rangle [100, 100] 100$)

lemma *bisimSubstScopeExt*:

fixes $\Psi :: 'b$
and $x :: name$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$

assumes $x \notin P$

shows $\Psi \triangleright (\nu x)(P \parallel Q) \sim_s P \parallel (\nu x)Q$
 $\langle proof \rangle$

lemma *bisimSubstCasePushRes*:

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

assumes $x \notin map fst Cs$

shows $\Psi \triangleright (\nu x)(Cases Cs) \sim_s Cases map (\lambda(\varphi, P). (\varphi, (\nu x)P)) Cs$
 $\langle proof \rangle$

lemma *bisimSubstOutputPushRes*:

fixes $x :: name$
and $\Psi :: 'b$
and $M :: 'a$
and $N :: 'a$

```

and    $P :: ('a, 'b, 'c) \psi$ 

assumes  $x \notin M$ 
and    $x \notin N$ 

shows  $\Psi \triangleright (\nu x)(M\langle N \rangle.P) \sim_s M\langle N \rangle.(\nu x)P$ 
 $\langle proof \rangle$ 

lemma bisimSubstInputPushRes:
  fixes  $x :: name$ 
  and    $\Psi :: 'b$ 
  and    $M :: 'a$ 
  and    $xvec :: name list$ 
  and    $N :: 'a$ 

  assumes  $x \notin M$ 
  and    $x \notin xvec$ 
  and    $x \notin N$ 

  shows  $\Psi \triangleright (\nu x)(M(\lambda*xvec N).P) \sim_s M(\lambda*xvec N).(\nu x)P$ 
 $\langle proof \rangle$ 

lemma bisimSubstResComm:
  fixes  $x :: name$ 
  and    $y :: name$ 

  shows  $\Psi \triangleright (\nu x)((\nu y)P) \sim_s (\nu y)((\nu x)P)$ 
 $\langle proof \rangle$ 

lemma bisimSubstExtBang:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  assumes guarded P

  shows  $\Psi \triangleright !P \sim_s P \parallel !P$ 
 $\langle proof \rangle$ 

lemma structCongBisimSubst:
  fixes  $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $P \equiv_s Q$ 

  shows  $P \sim_s Q$ 
 $\langle proof \rangle$ 

end

```

```

end

theory Weak-Bisim-Subst
  imports Weak-Bisim-Struct-Cong Weak-Bisim-Pres Bisim-Subst
begin

context env begin

abbreviation
  weakBisimSubstJudge ( $\langle \cdot \triangleright \cdot \approx_s \cdot \rangle [70, 70, 70] 65$ ) where  $\Psi \triangleright P \approx_s Q \equiv (\Psi, P, Q) \in closeSubst weakBisim$ 
abbreviation
  weakBisimSubstNilJudge ( $\langle \cdot \approx_s \cdot \rangle [70, 70] 65$ ) 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  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $\Psi \triangleright P \sim_s Q$ 

  shows  $\Psi \triangleright P \approx_s Q$ 
   $\langle proof \rangle$ 

lemma weakBisimSubstOutputPres:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $M :: 'a$ 
  and  $N :: 'a$ 

  assumes  $\Psi \triangleright P \approx_s Q$ 

  shows  $\Psi \triangleright M\langle N \rangle.P \approx_s M\langle N \rangle.Q$ 
   $\langle proof \rangle$ 

lemma bisimSubstInputPres:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $M :: 'a$ 
  and  $xvec :: name list$ 
  and  $N :: 'a$ 

  assumes  $\Psi \triangleright P \approx_s Q$ 
  and  $xvec \sharp* \Psi$ 

```

```

and      distinct xvec

shows  $\Psi \triangleright M(\lambda*xvec\ N).P \approx_s M(\lambda*xvec\ N).Q$ 
⟨proof⟩

lemma weakBisimSubstReflexive:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 

shows  $\Psi \triangleright P \approx_s P$ 
⟨proof⟩

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  $\Psi \triangleright P \approx_s R$ 
⟨proof⟩

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  $\Psi \triangleright Q \approx_s P$ 
⟨proof⟩

lemma weakBisimSubstParPres:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $Q :: ('a, 'b, 'c) \psi$ 
and    $R :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \approx_s Q$ 

shows  $\Psi \triangleright P \parallel R \approx_s Q \parallel R$ 
⟨proof⟩

lemma weakBisimSubstResPres:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \psi$ 
and    $Q :: ('a, 'b, 'c) \psi$ 

```

```

and    $x :: name$ 

assumes  $\Psi \triangleright P \approx_s Q$ 
and    $x \notin \Psi$ 

shows  $\Psi \triangleright (\nu x)P \approx_s (\nu x)Q$ 
 $\langle proof \rangle$ 

```

```

lemma weakBisimSubstParNil:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \parallel \mathbf{0} \approx_s P$ 
 $\langle proof \rangle$ 

lemma weakBisimSubstParComm:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \parallel Q \approx_s Q \parallel P$ 
 $\langle proof \rangle$ 

lemma weakBisimSubstParAssoc:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 
  and    $R :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright (P \parallel Q) \parallel R \approx_s P \parallel (Q \parallel R)$ 
 $\langle proof \rangle$ 

lemma weakBisimSubstResNil:
  fixes  $\Psi :: 'b$ 
  and    $x :: name$ 

  shows  $\Psi \triangleright (\nu x)\mathbf{0} \sim_s \mathbf{0}$ 
 $\langle proof \rangle$ 

lemma weakBisimSubstScopeExt:
  fixes  $\Psi :: 'b$ 
  and    $x :: name$ 
  and    $P :: ('a, 'b, 'c) \psi$ 
  and    $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $x \notin P$ 

```

```

shows  $\Psi \triangleright (\nu x)(P \parallel Q) \approx_s P \parallel (\nu x)Q$ 
⟨proof⟩

lemma weakBisimSubstCasePushRes:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $Cs :: ('c \times ('a, 'b, 'c) psi) list$ 

  assumes  $x \notin map fst Cs$ 

  shows  $\Psi \triangleright (\nu x)(Cases\ Cs) \approx_s Cases\ map (\lambda(\varphi, P). (\varphi, (\nu x)P))\ Cs$ 
⟨proof⟩

lemma weakBisimSubstOutputPushRes:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $M :: 'a$ 
  and  $N :: 'a$ 
  and  $P :: ('a, 'b, 'c) psi$ 

  assumes  $x \notin \Psi$ 
  and  $x \notin M$ 
  and  $x \notin N$ 

  shows  $\Psi \triangleright (\nu x)(M\langle N \rangle.P) \approx_s M\langle N \rangle.(\nu x)P$ 
⟨proof⟩

lemma weakBisimSubstInputPushRes:
  fixes  $x :: name$ 
  and  $\Psi :: 'b$ 
  and  $M :: 'a$ 
  and  $xvec :: name list$ 
  and  $N :: 'a$ 

  assumes  $x \notin M$ 
  and  $x \notin xvec$ 
  and  $x \notin N$ 

  shows  $\Psi \triangleright (\nu x)(M(\lambda*xvec\ N).P) \approx_s M(\lambda*xvec\ N).(\nu x)P$ 
⟨proof⟩

lemma weakBisimSubstResComm:
  fixes  $x :: name$ 
  and  $y :: name$ 

  shows  $\Psi \triangleright (\nu x)(\nu y)P \approx_s (\nu y)(\nu x)P$ 
⟨proof⟩

lemma weakBisimSubstExtBang:

```

```

fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes guarded P

shows  $\Psi \triangleright !P \approx_s P \parallel !P$ 
(proof)

end

end

theory Weakening
imports Weak-Bisimulation
begin

locale weak = env +
assumes weaken:  $\Psi \hookrightarrow \Psi \otimes \Psi'$ 
begin

lemma entWeaken:
fixes  $\Psi :: 'b$ 
and  $\varphi :: 'c$ 

assumes  $\Psi \vdash \varphi$ 

shows  $\Psi \otimes \Psi' \vdash \varphi$ 
(proof)

lemma assertWeaken:
fixes  $\Psi :: 'b$ 
and  $\Psi' :: 'b$ 

shows  $\Psi \hookrightarrow \Psi \otimes \Psi'$ 
(proof)

lemma frameWeaken:
fixes  $F :: 'b \text{ frame}$ 
and  $G :: 'b \text{ frame}$ 

shows  $F \hookrightarrow_F F \otimes_F G$ 
(proof)

lemma unitAssertWeaken:
fixes  $\Psi :: 'b$ 

shows  $\mathbf{1} \hookrightarrow \Psi$ 
(proof)

```

```

lemma unitFrameWeaken:
  fixes F :: 'b frame

  shows ⟨ε, 1⟩ ↣F F
  ⟨proof⟩

lemma insertAssertionWeaken:
  fixes F :: 'b frame
  and Ψ :: 'b

  shows ⟨ε, Ψ⟩ ↣F insertAssertion F Ψ
  ⟨proof⟩

lemma frameImpStatEq:
  fixes AF :: name list
  and Ψ :: 'b
  and Ψ' :: 'b
  and φ :: 'c

  assumes ((AF, Ψ)) ⊢F φ
  and Ψ ≈ Ψ'

  shows ((AF, Ψ')) ⊢F φ
  ⟨proof⟩

lemma statImpTauDerivative:
  fixes Ψ :: 'b
  and P :: ('a, 'b, 'c) psi
  and P' :: ('a, 'b, 'c) psi

  assumes Ψ ▷ P ↤τ ↲ P'

  shows insertAssertion (extractFrame P) Ψ ↣F insertAssertion (extractFrame
P') Ψ
  ⟨proof⟩

lemma weakenTransition:
  fixes Ψ :: 'b
  and P :: ('a, 'b, 'c) psi
  and Rs :: ('a, 'b, 'c) residual
  and Ψ' :: 'b

  assumes Ψ ▷ P ↤ Rs

  shows Ψ ⊗ Ψ' ▷ P ↤ Rs
  ⟨proof⟩

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 ( $\langle\cdot\rangle \triangleright - \implies - \prec -$  [80, 80, 80, 80] 80)
where
   $\Psi \triangleright P \implies \alpha \prec P' \equiv (\exists P''' P''. \Psi \triangleright P \implies \hat{\tau} P''' \wedge \Psi \triangleright P''' \mapsto \alpha \prec P'' \wedge \Psi$ 
 $\triangleright P'' \implies \hat{\tau} P') \vee (P = P' \wedge \alpha = \tau)$ 

lemma weakenTransitionCases[consumes 1, case-names cBase cStep]:
  assumes  $\Psi \triangleright P \implies \alpha \prec P'$ 
  and Prop( $\tau$ ) P
  and  $\bigwedge P''' P''.$  [ $\Psi \triangleright P \implies \hat{\tau} P'''$ ;  $\Psi \triangleright P''' \mapsto \alpha \prec P''$ ;  $\Psi \triangleright P'' \implies \hat{\tau} P'$ ]  $\implies$ 
  Prop  $\alpha$  P'

  shows Prop  $\alpha$  P'
   $\langle proof \rangle$ 

lemma statImpTauChainDerivative:
  fixes Psi :: 'b
  and P :: ('a, 'b, 'c) psi
  and P' :: ('a, 'b, 'c) psi

  assumes  $\Psi \triangleright P \implies \hat{\tau} P'$ 

  shows insertAssertion(extractFrame P) Psi  $\hookrightarrow_F$  insertAssertion(extractFrame P') Psi
   $\langle proof \rangle$ 

lemma weakenTauChain:
  fixes Psi :: 'b
  and P :: ('a, 'b, 'c) psi
  and P' :: ('a, 'b, 'c) psi
  and Psi' :: 'b

  assumes  $\Psi \triangleright P \implies \hat{\tau} P'$ 
  shows  $\Psi \otimes \Psi' \triangleright P \implies \hat{\tau} P'$ 
   $\langle proof \rangle$ 

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 ( $\langle$ -  $\triangleright$  -  $\lesssim_w$   $\langle$ -  $\triangleright$  - [80, 80, 80, 80] 80)
  where  $\Psi \triangleright P \lesssim_w \langle Rel \rangle Q \equiv \exists Q'. \Psi \triangleright Q \implies \hat{\tau} Q' \wedge insertAssertion(extractFrame P) \Psi \hookrightarrow_F insertAssertion(extractFrame Q') \Psi \wedge (\Psi, P, Q') \in Rel$ 

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  $\Psi \triangleright P \lesssim_w \langle A \rangle Q$ 
  and  $A \subseteq B$ 

  shows  $\Psi \triangleright P \lesssim_w \langle B \rangle Q$ 
   $\langle proof \rangle$ 

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  $\Psi \triangleright Q \implies \hat{\tau} Q'$ 
  and  $insertAssertion(extractFrame P) \Psi \hookrightarrow_F insertAssertion(extractFrame Q')$ 
   $\Psi$ 
  and  $(\Psi, P, Q') \in Rel$ 

  shows  $\Psi \triangleright P \lesssim_w \langle Rel \rangle Q$ 
   $\langle proof \rangle$ 

lemma weakenStatImpE:
  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  $\Psi \triangleright P \lesssim_w \langle Rel \rangle Q$ 

```

obtains Q' **where** $\Psi \triangleright Q \Rightarrow \hat{\tau} Q' \text{ and } insertAssertion(extractFrame P) \Psi \rightarrow_F insertAssertion(extractFrame Q') \Psi \text{ and } (\Psi, P, Q') \in Rel$
 $\langle proof \rangle$

lemma *weakStatImp WeakenStatImp*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Rel :: ('b \times ('a, 'b, 'c) psi \times ('a, 'b, 'c) psi) set$
and $Q :: ('a, 'b, 'c) psi$
assumes $cSim: \Psi \triangleright P \lesssim_{Rel} Q$
and $cStatEq: \bigwedge \Psi' R S \Psi''. \llbracket (\Psi', R, S) \in Rel; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', R, S) \in Rel$
shows $\Psi \triangleright P \lesssim_w Rel Q$
 $\langle proof \rangle$

lemma *weakenStatImp WeakStatImp*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Rel :: ('b \times ('a, 'b, 'c) psi \times ('a, 'b, 'c) psi) set$
and $Q :: ('a, 'b, 'c) psi$
assumes $\Psi \triangleright P \lesssim_w Rel Q$
and $cExt: \bigwedge \Psi' R S \Psi''. (\Psi', R, S) \in Rel \implies (\Psi' \otimes \Psi'', R, S) \in Rel$
shows $\Psi \triangleright P \lesssim_{Rel} Q$
 $\langle proof \rangle$

end

end

theory *Weaken-Simulation*

imports *Weaken-Stat-Imp*
begin

context *weak*
begin

definition

$weakenSimulation :: 'b \Rightarrow ('a, 'b, 'c) psi \Rightarrow$
 $('b \times ('a, 'b, 'c) psi \times ('a, 'b, 'c) psi) set \Rightarrow$
 $('a, 'b, 'c) psi \Rightarrow bool (\langle \triangleright - \rightsquigarrow_w \rangle [80, 80, 80, 80] 80)$

where

$\Psi \triangleright P \rightsquigarrow_w Rel Q \equiv \forall \alpha Q'. \Psi \triangleright Q \mapsto \alpha \prec Q' \longrightarrow bn \alpha \sharp * \Psi \longrightarrow bn \alpha \sharp *$
 $P \longrightarrow (\exists P'. \Psi \triangleright P \Rightarrow \alpha \prec P' \wedge (\Psi, P', Q') \in Rel)$

```

lemma weakenSimI[case-names cAct]:
  fixes  $\Psi$  :: ' $b$ 
  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:  $\bigwedge \alpha Q' . [\Psi \triangleright Q \xrightarrow{\alpha} Q'; bn \alpha \sharp* \Psi; bn \alpha \sharp* P] \implies \exists P'. \Psi \triangleright P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow_w^{<Rel>} Q$ 
⟨proof⟩

lemma weakenSimWeakSim:
  fixes  $\Psi$  :: ' $b$ 
  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  $(\Psi, P, Q) \in Rel$ 
  and cStatImp:  $\bigwedge \Psi' R S . (\Psi, R, S) \in Rel \implies \Psi \triangleright R \lesssim_w^{<Rel>} S$ 
  and cSim:  $\bigwedge \Psi' R S . (\Psi, R, S) \in Rel \implies \Psi \triangleright R \rightsquigarrow_w^{<Rel>} S$ 
  and cExt:  $\bigwedge \Psi' R S \Psi' . (\Psi, R, S) \in Rel' \implies (\Psi \otimes \Psi', R, S) \in Rel'$ 
  and cSym:  $\bigwedge \Psi' R S . (\Psi, R, S) \in Rel \implies (\Psi, S, R) \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow^{<Rel'>} Q$ 
⟨proof⟩

lemma weakSimWeakenSim:
  fixes  $\Psi$  :: ' $b$ 
  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 cSim:  $\Psi \triangleright P \rightsquigarrow^{<Rel>} Q$ 
  and cStatEq:  $\bigwedge \Psi' R S \Psi'' . [(\Psi', R, S) \in Rel; \Psi' \simeq \Psi''] \implies (\Psi'', R, S) \in Rel$ 

shows  $\Psi \triangleright P \rightsquigarrow_w^{<Rel>} Q$ 
⟨proof⟩

lemma weakenSimE:
  fixes  $F$  :: ' $b$ 
  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  $\Psi \triangleright P \rightsquigarrow_w^{<Rel>} Q$ 

```

shows $\wedge \alpha \ Q'. [\Psi \triangleright Q \xrightarrow{\alpha} Q'; bn \alpha \sharp* \Psi; bn \alpha \sharp* P] \implies \exists P'. \Psi \triangleright P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in Rel$
 $\langle proof \rangle$

lemma *weakenSimMonotonic*:

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 $\Psi \triangleright P \rightsquigarrow_w A \ Q$
and $A \subseteq B$

shows $\Psi \triangleright P \rightsquigarrow_w B \ Q$

$\langle proof \rangle$

end

end

theory *Weaken-Bisimulation*

imports *Weaken-Simulation Weaken-Stat-Imp*

begin

context *weak*
begin

lemma *weakenMonoCoinduct*: $\wedge x y xa xb xc P Q \Psi.$

$x \leq y \implies$
 $(\Psi \triangleright Q \rightsquigarrow_w \{(xc, xb, xa), x xc xb xa\} > P) \longrightarrow$
 $(\Psi \triangleright Q \rightsquigarrow_w \{(xb, xa, xc), y xb xa xc\} > P)$

$\langle proof \rangle$

lemma *weakenMonoCoinduct2*: $\wedge x y xa xb xc P Q \Psi.$

$x \leq y \implies$
 $(\Psi \triangleright Q \lesssim_w \{(xc, xb, xa), x xc xb xa\} > P) \longrightarrow$
 $(\Psi \triangleright Q \lesssim_w \{(xb, xa, xc), y xb xa xc\} > P)$

$\langle proof \rangle$

coinductive-set *weakenBisim* :: $('b \times ('a, 'b, 'c) psi \times ('a, 'b, 'c) psi) set$
where

step: $[\Psi \triangleright P \lesssim_w \{weakenBisim\} Q; \Psi \triangleright P \rightsquigarrow_w \{weakenBisim\} Q;$
 $\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in weakenBisim; (\Psi, Q, P) \in weakenBisim] \implies (\Psi, P, Q) \in weakenBisim$

monos *weakenMonoCoinduct* *weakenMonoCoinduct2*

abbreviation

weakenBisimJudge ($\langle - \triangleright - \approx_w \rangle [70, 70, 70] 65$) **where** $\Psi \triangleright P \approx_w Q \equiv (\Psi,$

$P, Q) \in weakenBisim$

abbreviation

weakenBisimNilJudge ($\langle \cdot \approx_w \cdot \rangle [70, 70] 65$) **where** $P \approx_w Q \equiv \mathbf{1} \triangleright P \approx_w Q$

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 $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi \triangleright P \lesssim_w <(X \cup weakenBisim)> Q) \wedge$

$(\Psi \triangleright P \rightsquigarrow_w <(X \cup weakenBisim)> Q) \wedge$

$(\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in X \vee (\Psi \otimes \Psi', P, Q) \in$

weakenBisim) \wedge

$((\Psi, Q, P) \in X \vee (\Psi, Q, P) \in weakenBisim)$

shows $(\Psi, P, Q) \in weakenBisim$

{proof}

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 $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \Psi' \triangleright R \lesssim_w <(X \cup weakenBisim)> S$

and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies \Psi' \triangleright R \rightsquigarrow_w <(X \cup weakenBisim)> S$

and $\bigwedge \Psi' R S \Psi''. (\Psi', R, S) \in X \implies (\Psi' \otimes \Psi'', R, S) \in X \vee \Psi' \otimes \Psi'' \triangleright$

$R \approx_w S$

and $\bigwedge \Psi' R S. (\Psi', R, S) \in X \implies (\Psi', S, R) \in X \vee \Psi' \triangleright S \approx_w R$

shows $\Psi \triangleright P \approx_w Q$

{proof}

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 $\bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_w <X> Q \wedge$

$\Psi \triangleright P \rightsquigarrow_w <X> Q \wedge (\forall \Psi'. (\Psi \otimes \Psi', P, Q) \in X) \wedge$

$(\Psi, Q, P) \in X$

shows $\Psi \triangleright P \approx_w Q$

{proof}

```

lemma weakenBisimE:
  fixes P :: ('a, 'b, 'c) psi
  and Q :: ('a, 'b, 'c) psi
  and Ψ :: 'b
  and Ψ' :: 'b

  assumes Ψ ⊢ P ≈w Q

  shows Ψ ⊢ P ≈w<weakenBisim> Q
  and Ψ ⊢ P ~≈w<weakenBisim> Q
  and Ψ ⊗ Ψ' ⊢ P ≈w Q
  and Ψ ⊢ Q ≈w P
  ⟨proof⟩

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 × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

  assumes (Ψ, P, Q) ∈ X
  and ⋀Ψ P Q. (Ψ, P, Q) ∈ X ⇒ Ψ ⊢ P ≈w<X> Q
  and ⋀Ψ P Q. (Ψ, P, Q) ∈ X ⇒ Ψ ⊢ P ~≈w<X> Q
  and ⋀Ψ P Q Ψ'. (Ψ, P, Q) ∈ X ⇒ (Ψ ⊗ Ψ', P, Q) ∈ X
  and ⋀Ψ P Q. (Ψ, P, Q) ∈ X ⇒ (Ψ, Q, P) ∈ X

  shows (Ψ, P, Q) ∈ weakenBisim
  ⟨proof⟩

lemma weakenBisimEqWeakBisim[simp]: weakenBisim = weakBisim
  ⟨proof⟩

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 × ('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 ≈w<X> Q
  and rSim: ⋀Ψ P Q. (Ψ, P, Q) ∈ X ⇒ Ψ ⊢ P ~≈w<((Ψ, P, Q) | Ψ P Q. ∃P' Q'. Ψ ⊢ P ~ P' ∧
    (Ψ, P', Q') ∈ X ∧
    Ψ ⊢ Q' ~ Q})> Q
  and rExt: ⋀Ψ P Q Ψ'. (Ψ, P, Q) ∈ X ⇒ (Ψ ⊗ Ψ', P, Q) ∈ X

```

and $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X$

shows $\Psi \triangleright P \approx_w Q$
 $\langle proof \rangle$

lemma $weakenTransitiveCoinduct[\text{case-names } cStatImp\ cSim\ cExt\ cSym, \text{case-conclusion bisim step, consumes 2}]:$

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $X :: ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) \text{ set}$

assumes $p: (\Psi, P, Q) \in X$
and $Eqvt: eqvt X$
and $rStatImp: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \lesssim_w (X \cup weakenBisim) \triangleright Q$
and $rSim: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies \Psi \triangleright P \rightsquigarrow_w (\{(\Psi, P, Q) \mid \Psi \triangleright P \exists P'. \Psi \triangleright P \sim P' \wedge (\Psi, P', Q') \in (X \cup weakenBisim)\} \wedge \Psi \triangleright Q' \sim Q)$
and $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in X \implies (\Psi \otimes \Psi', P, Q) \in X \cup weakenBisim$
and $rSym: \bigwedge \Psi P Q. (\Psi, P, Q) \in X \implies (\Psi, Q, P) \in X \cup weakenBisim$

shows $\Psi \triangleright P \approx_w Q$
 $\langle proof \rangle$

end

end

theory *Weak-Cong-Simulation*
imports *Weak-Simulation Tau-Chain*

begin

context *env* **begin**

definition

$weakCongSimulation :: 'b \Rightarrow ('a, 'b, 'c) \psi \Rightarrow ('b \times ('a, 'b, 'c) \psi \times ('a, 'b, 'c) \psi) \text{ set} \Rightarrow ('a, 'b, 'c) \psi \Rightarrow \text{bool} ((\dashv \triangleright - \rightsquigarrow \dashrightarrow [80, 80, 80, 80]) 80)$

where

$\Psi \triangleright P \rightsquigarrow \langle Rel \rangle Q \equiv \forall Q'. \Psi \triangleright Q \mapsto_{\tau} \prec Q' \longrightarrow (\exists P'. \Psi \triangleright P \Longrightarrow_{\tau} P' \wedge (\Psi, P', Q') \in Rel)$

abbreviation

$weakCongSimulationNilJudge ((\dashv \rightsquigarrow \dashrightarrow [80, 80, 80] 80) \text{ where } P \rightsquigarrow \langle Rel \rangle Q \equiv SBottom' \triangleright P \rightsquigarrow \langle Rel \rangle Q)$

lemma *weakCongSimI*[case-names *cTau*]:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $C :: 'd::fs-name$

assumes $rTau: \bigwedge Q'. \Psi \triangleright Q \xrightarrow{\tau} Q' \implies \exists P'. \Psi \triangleright P \implies_{\tau} P' \wedge (\Psi, P', Q') \in \text{Rel}$

shows $\Psi \triangleright P \rightsquigarrow \langle \text{Rel} \rangle Q$
(proof)

lemma *weakCongSimE*:
fixes $F :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$

assumes $\Psi \triangleright P \rightsquigarrow \langle \text{Rel} \rangle Q$
and $\Psi \triangleright Q \xrightarrow{\tau} Q'$

obtains P' **where** $\Psi \triangleright P \implies_{\tau} P'$ **and** $(\Psi, P', Q') \in \text{Rel}$
(proof)

lemma *weakCongSimClosedAux*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $p :: \text{name prm}$

assumes *EqvtRel*: *eqvt Rel*
and *PSimQ*: $\Psi \triangleright P \rightsquigarrow \langle \text{Rel} \rangle Q$

shows $(p \cdot \Psi) \triangleright (p \cdot P) \rightsquigarrow \langle \text{Rel} \rangle (p \cdot Q)$
(proof)

lemma *weakCongSimClosed*:
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $\text{Rel} :: ('b \times ('a, 'b, 'c) \text{ psi} \times ('a, 'b, 'c) \text{ psi}) \text{ set}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $p :: \text{name prm}$

assumes *EqvtRel*: *eqvt Rel*

shows $\Psi \triangleright P \rightsquigarrow \langle \text{Rel} \rangle Q \implies (p \cdot \Psi) \triangleright (p \cdot P) \rightsquigarrow \langle \text{Rel} \rangle (p \cdot Q)$

and $P \rightsquigarrow \langle Rel \rangle Q \implies (p \cdot P) \rightsquigarrow \langle Rel \rangle (p \cdot Q)$
 $\langle proof \rangle$

lemma *weakCongSimReflexive*:
fixes $Rel :: (\langle b \rangle \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi) set$
and $\Psi :: \langle b \rangle$
and $P :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$

assumes $\{(\Psi, P, P) \mid \Psi P. True\} \subseteq Rel$

shows $\Psi \triangleright P \rightsquigarrow \langle Rel \rangle P$
 $\langle proof \rangle$

lemma *weakStepSimTauChain*:

fixes $\Psi :: \langle b \rangle$
and $Q :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$
and $Q' :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$
and $P :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$
and $Rel :: (\langle b \rangle \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi) set$

assumes $\Psi \triangleright Q \implies_{\tau} Q'$
and $\Psi \triangleright P \rightsquigarrow \langle Rel \rangle Q$
and $Sim: \bigwedge \Psi P Q. (\Psi, P, Q) \in Rel \implies \Psi \triangleright P \rightsquigarrow \langle Rel \rangle Q$

obtains P' **where** $\Psi \triangleright P \implies_{\tau} P'$ **and** $(\Psi, P', Q') \in Rel$
 $\langle proof \rangle$

lemma *weakCongSimTransitive*:

fixes $\Psi :: \langle b \rangle$
and $P :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$
and $Rel :: (\langle b \rangle \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi) set$
and $Q :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$
and $Rel' :: (\langle b \rangle \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi) set$
and $T :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$
and $Rel'' :: (\langle b \rangle \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi \times (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi) set$

assumes $PRelQ: (\Psi, P, Q) \in Rel$
and $PSimQ: \Psi \triangleright P \rightsquigarrow \langle Rel \rangle Q$
and $QSimR: \Psi \triangleright Q \rightsquigarrow \langle Rel' \rangle R$
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 $Sim: \bigwedge \Psi P Q. (\Psi, P, Q) \in Rel \implies \Psi \triangleright P \rightsquigarrow \langle Rel \rangle Q$

shows $\Psi \triangleright P \rightsquigarrow \langle Rel'' \rangle R$
 $\langle proof \rangle$

lemma *weakCongSimStatEq*:

fixes $\Psi :: \langle b \rangle$
and $P :: (\langle a \rangle, \langle b \rangle, \langle c \rangle) \psi$

```

and   Rel :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set
and   Q   :: ('a, 'b, 'c) psi
and   Ψ' :: 'b

assumes PSimQ: Ψ ⊢ P ~«Rel» Q
and   Ψ ≈ Ψ'
and   C1: ∩Ψ P Q Ψ'. [(Ψ, P, Q) ∈ Rel; Ψ ≈ Ψ'] ==> (Ψ', P, Q) ∈ Rel'

shows Ψ' ⊢ P ~«Rel'» Q
⟨proof⟩

lemma weakCongSimMonotonic:
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 Ψ ⊢ P ~«A» Q
and   A ⊆ B

shows Ψ ⊢ P ~«B» Q
⟨proof⟩

lemma strongSimWeakCongSim:
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 Ψ ⊢ P ~*[Rel] Q
and   Rel ⊆ Rel'

shows Ψ ⊢ P ~«Rel'» Q
⟨proof⟩

end

end

theory Weak-Psi-Congruence
imports Weak-Cong-Simulation Weak-Bisimulation
begin

context env begin

definition weakPsiCongruence :: 'b ⇒ ('a, 'b, 'c) psi ⇒ ('a, 'b, 'c) psi ⇒ bool (⊣
⊣ - ≡ - [70, 70, 70] 65)
where

```

$\Psi \triangleright P \doteq Q \equiv \Psi \triangleright P \approx Q \wedge \Psi \triangleright P \rightsquigarrow \text{weakBisim} \quad Q \wedge \Psi \triangleright Q \rightsquigarrow \text{weakBisim}$

abbreviation

weakPsiCongNilJudge ($\langle - \doteq - \rangle [70, 70] 65$) **where** $P \doteq Q \equiv \mathbf{1} \triangleright P \doteq Q$

lemma *weakPsiCongSym*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$

assumes $\Psi \triangleright P \doteq Q$

shows $\Psi \triangleright Q \doteq P$

(proof)

lemma *weakPsiCongE*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $\Psi' :: 'b$

assumes $\Psi \triangleright P \doteq Q$

shows $\Psi \triangleright P \approx Q$

and $\Psi \triangleright P \rightsquigarrow \text{weakBisim} \quad Q$
and $\Psi \triangleright Q \rightsquigarrow \text{weakBisim} \quad P$

(proof)

lemma *weakPsiCongI*[*case-names cWeakBisim cSimLeft cSimRight*]:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $\Psi' :: 'b$

assumes $\Psi \triangleright P \approx Q$

and $\Psi \triangleright P \rightsquigarrow \text{weakBisim} \quad Q$
and $\Psi \triangleright Q \rightsquigarrow \text{weakBisim} \quad P$

shows $\Psi \triangleright P \doteq Q$

(proof)

lemma *weakPsiCongSymI*[*consumes 1, case-names cSym cWeakBisim cSim*]:

fixes $\Psi :: 'b$
and $P :: 'd::fs\text{-name}$
and $Q :: 'd$
and $\Psi' :: 'b$

```

assumes Prop P Q
and    $\bigwedge P Q. \text{Prop } P Q \implies \text{Prop } Q P$ 

and    $\bigwedge P Q. \text{Prop } P Q \implies \Psi \triangleright (C P) \approx (C Q)$ 

and    $\bigwedge P Q. \text{Prop } P Q \implies \Psi \triangleright (C P) \rightsquigarrow \langle\!\langle \text{weakBisim} \rangle\!\rangle (C Q)$ 

shows  $\Psi \triangleright (C P) \doteq (C Q)$ 
⟨proof⟩

lemma weakPsiCongSym2[consumes 1, case-names cWeakBisim cSim]:
fixes  $\Psi :: 'b$ 
and    $\Psi' :: 'b$ 

assumes  $\Psi \triangleright P \doteq Q$ 

and    $\bigwedge P Q. \Psi \triangleright P \doteq Q \implies \Psi \triangleright (C P) \approx (C Q)$ 

and    $\bigwedge P Q. \Psi \triangleright P \doteq Q \implies \Psi \triangleright (C P) \rightsquigarrow \langle\!\langle \text{weakBisim} \rangle\!\rangle (C Q)$ 

shows  $\Psi \triangleright (C P) \doteq (C Q)$ 
⟨proof⟩

lemma statEqWeakCong:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{psi}$ 
and    $Q :: ('a, 'b, 'c) \text{psi}$ 
and    $\Psi' :: 'b$ 

assumes  $\Psi \triangleright P \doteq Q$ 
and    $\Psi \simeq \Psi'$ 

shows  $\Psi' \triangleright P \doteq Q$ 
⟨proof⟩

lemma weakPsiCongReflexive:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{psi}$ 

shows  $\Psi \triangleright P \doteq P$ 
⟨proof⟩

lemma weakPsiCongClosed:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{psi}$ 
and    $Q :: ('a, 'b, 'c) \text{psi}$ 
and    $p :: \text{name prm}$ 

```

```

assumes  $\Psi \triangleright P \doteq Q$ 

shows  $(p \cdot \Psi) \triangleright (p \cdot P) \doteq (p \cdot Q)$ 
⟨proof⟩

lemma weakPsiCongTransitive:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
and  $R :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \doteq Q$ 
and  $\Psi \triangleright Q \doteq R$ 

shows  $\Psi \triangleright P \doteq R$ 
⟨proof⟩

lemma strongBisimWeakPsiCong:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \sim Q$ 

shows  $\Psi \triangleright P \doteq Q$ 
⟨proof⟩

lemma structCongWeakPsiCong:
fixes  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

assumes  $P \equiv_s Q$ 

shows  $P \doteq Q$ 
⟨proof⟩

end

theory Weak-Cong-Sim-Pres
imports Weak-Sim-Pres Weak-Cong-Simulation
begin

context env begin

lemma caseWeakSimPres:
fixes  $\Psi :: 'b$ 
and  $CsP :: ('c \times ('a, 'b, 'c) \psi) list$ 

```

```

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:  $\bigwedge \varphi Q. (\varphi, Q) \text{ mem } CsQ \implies \exists P. (\varphi, P) \text{ mem } CsP \wedge \text{guarded}$   

 $P \wedge Eq \Psi P Q$ 
and   Sim:  $\bigwedge \Psi' P Q. (\Psi', P, Q) \in Rel \implies \Psi' \triangleright P \rightsquigarrow \langle Rel \rangle Q$ 
and   EqRel:  $\bigwedge \Psi' P Q. Eq \Psi' P Q \implies (\Psi', P, Q) \in Rel$ 
and   EqSim:  $\bigwedge \Psi' P Q. Eq \Psi' P Q \implies \Psi' \triangleright P \rightsquigarrow \langle Rel \rangle Q$ 

shows  $\Psi \triangleright Cases\ CsP \rightsquigarrow \langle Rel \rangle Cases\ CsQ$   

(proof)

lemma weakCongSimCasePres:
fixes  $\Psi :: 'b$ 
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:  $\bigwedge \varphi Q. (\varphi, Q) \text{ mem } CsQ \implies \exists P. (\varphi, P) \text{ mem } CsP \wedge \text{guarded}$   

 $P \wedge Eq \Psi P Q$ 
and   EqSim:  $\bigwedge \Psi' P Q. Eq \Psi' P Q \implies \Psi' \triangleright P \rightsquigarrow \langle Rel \rangle Q$ 

shows  $\Psi \triangleright Cases\ CsP \rightsquigarrow \langle Rel \rangle Cases\ CsQ$   

(proof)

lemma weakCongSimResPres:
fixes  $\Psi :: 'b$ 
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   Rel' :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set

assumes PSimQ:  $\Psi \triangleright P \rightsquigarrow \langle Rel \rangle Q$ 
and   eqvt Rel'
and   x ∉ Ψ
and   Rel ⊆ Rel'
and   C1:  $\bigwedge \Psi' R S x. [( \Psi', R, S ) \in Rel; x \notin \Psi] \implies (\Psi', (\nu x)R, (\nu x)S) \in Rel'$ 

shows  $\Psi \triangleright (\nu x)P \rightsquigarrow \langle Rel' \rangle (\nu x)Q$   

(proof)

lemma weakCongSimResChainPres:
fixes  $\Psi :: 'b$ 

```

```

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 xvec :: name list

assumes PSimQ: Ψ ⊢ P ~~~«Rel» Q
and eqvt Rel
and xvec #* Ψ
and C1:  $\bigwedge \Psi' R S xvec. \llbracket (\Psi', R, S) \in Rel; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)R, (\nu*xvec)S) \in Rel$ 

shows Ψ ⊢ (\nu*xvec)P ~~~«Rel» (\nu*xvec)Q
⟨proof⟩

lemma weakCongSimParPres:
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 PSimQ:  $\bigwedge \Psi'. \Psi' \triangleright P \rightsquigarrow «Rel» Q$ 
and PSimQ':  $\bigwedge \Psi'. \Psi' \triangleright P \rightsquigarrow <Rel> Q$ 
and StatImp:  $\bigwedge \Psi'. \Psi' \triangleright Q \lesssim <Rel> P$ 

and eqvt Rel
and eqvt Rel'

and Sym:  $\bigwedge \Psi' S T. \llbracket (\Psi', S, T) \in Rel \rrbracket \implies (\Psi', T, S) \in Rel$ 
and Ext:  $\bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel \rrbracket \implies (\Psi' \otimes \Psi'', S, T) \in Rel$ 

and C1:  $\bigwedge \Psi' S T A_U \Psi_U U. \llbracket (\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = (A_U, \Psi_U); A_U \#* \Psi'; A_U \#* S; A_U \#* T \rrbracket \implies (\Psi', S \parallel U, T \parallel U) \in Rel'$ 
and C2:  $\bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel'; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in Rel'$ 
and C3:  $\bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', S, T) \in Rel$ 

shows Ψ ⊢ P || R ~~~«Rel'» Q || R
⟨proof⟩
unbundle no relcomp-syntax

```

```

lemma weakCongSimBangPres:
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
and Rel' :: ('b × ('a, 'b, 'c) psi × ('a, 'b, 'c) psi) set
and Rel'' :: ('b × ('a, 'b, 'c) psi × ('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 $FrameParPres: \bigwedge \Psi' \Psi_U S T U A_U. \llbracket (\Psi' \otimes \Psi_U, S, T) \in Rel; extractFrame U = \langle A_U, \Psi_U \rangle; A_U \#* \Psi'; A_U \#* S; A_U \#* T \rrbracket \implies (\Psi', U \parallel S, U \parallel T) \in Rel$
and $C1: \bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel; guarded S; guarded T \rrbracket \implies (\Psi', U \parallel !S, U \parallel !T) \in Rel''$
and $Closed: \bigwedge \Psi' S T p. (\Psi', S, T) \in Rel \implies ((p::name prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel$
and $Closed': \bigwedge \Psi' S T p. (\Psi', S, T) \in Rel' \implies ((p::name prm) \cdot \Psi', p \cdot S, p \cdot T) \in Rel'$
and $StatEq: \bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', S, T) \in Rel$
and $StatEq': \bigwedge \Psi' S T \Psi''. \llbracket (\Psi', S, T) \in Rel'; \Psi' \simeq \Psi'' \rrbracket \implies (\Psi'', S, T) \in Rel'$
and $Trans: \bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel; (\Psi', T, U) \in Rel \rrbracket \implies (\Psi', S, U) \in Rel$
and $Trans': \bigwedge \Psi' S T U. \llbracket (\Psi', S, T) \in Rel'; (\Psi', T, U) \in Rel \rrbracket \implies (\Psi', S, U) \in Rel'$
and $EqSim: \bigwedge \Psi' S T. Eq S T \implies \Psi' \triangleright S \rightsquigarrow \llbracket Rel \rrbracket T$
and $cSim: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies \Psi' \triangleright S \rightsquigarrow \llbracket Rel \rrbracket T$
and $cSym: \bigwedge \Psi' S T. (\Psi', S, T) \in Rel \implies (\Psi', T, S) \in Rel$
and $cSym': \bigwedge \Psi' S T. (\Psi', S, T) \in Rel' \implies (\Psi', T, S) \in Rel'$
and $cExt: \bigwedge \Psi' S T \Psi''. (\Psi', S, T) \in Rel \implies (\Psi' \otimes \Psi'', S, T) \in Rel$
and $cExt': \bigwedge \Psi' S T \Psi''. (\Psi', S, T) \in Rel' \implies (\Psi' \otimes \Psi'', S, T) \in Rel'$
and $ParPres: \bigwedge \Psi' S T U. (\Psi', S, T) \in Rel \implies (\Psi', S \parallel U, T \parallel U) \in Rel$
and $ParPres': \bigwedge \Psi' S T U. (\Psi', S, T) \in Rel' \implies (\Psi', U \parallel S, U \parallel T) \in Rel'$
and $ParPres2: \bigwedge \Psi' S T. Eq S T \implies Eq(S \parallel S)(T \parallel T)$
and $ResPres: \bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in Rel$
and $ResPres': \bigwedge \Psi' S T xvec. \llbracket (\Psi', S, T) \in Rel'; xvec \#* \Psi \rrbracket \implies (\Psi', (\nu*xvec)S, (\nu*xvec)T) \in Rel'$
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 $ScopeExt: \bigwedge xvec \Psi' T S. \llbracket xvec \#* \Psi'; xvec \#* T \rrbracket \implies (\Psi', (\nu*xvec)(S \parallel T), ((\nu*xvec)S) \parallel T) \in Rel$
and $ScopeExt': \bigwedge xvec \Psi' T S. \llbracket xvec \#* \Psi'; xvec \#* T \rrbracket \implies (\Psi', (\nu*xvec)(S \parallel T), ((\nu*xvec)S) \parallel T) \in Rel'$
and $Compose: \bigwedge \Psi' S T U O. \llbracket (\Psi', S, T) \in Rel; (\Psi', T, U) \in Rel''; (\Psi', U, O) \in Rel \rrbracket \implies (\Psi', S, O) \in Rel''$
and $rBangActE: \bigwedge \Psi' S \alpha S'. \llbracket \Psi' \triangleright !S \mapsto \alpha \prec S'; guarded S; bn \alpha \#* S; \alpha \neq \tau; bn \alpha \#* subject \alpha \rrbracket \implies \exists T. \Psi' \triangleright S \mapsto \alpha \prec T \wedge (\mathbf{1}, S', T \parallel !S) \in Rel'$
and $rBangTauE: \bigwedge \Psi' S S'. \llbracket \Psi' \triangleright !S \mapsto \tau \prec S'; guarded S \rrbracket \implies \exists T. \Psi' \triangleright S \parallel S \mapsto \tau \prec T \wedge (\mathbf{1}, S', T \parallel !S) \in Rel'$
and $rBangTauI: \bigwedge \Psi' S S'. \llbracket \Psi' \triangleright S \parallel S \implies_{\tau} S'; guarded S \rrbracket \implies \exists T. \Psi' \triangleright$

```

!S ==>τ T ∧ (Ψ', T, S' ∥ !S) ∈ Rel'
  shows Ψ ▷ R ∥ !P ~ ``Rel''» R ∥ !Q
⟨proof⟩
  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 ∀ 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⟩

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 Ψ ▷ P ≈ Q

shows Ψ ▷ M⟨N⟩.P ≈ M⟨N⟩.Q
⟨proof⟩

lemma weakBisimCasePres:
  fixes Ψ :: 'b
  and CsP :: ('c × ('a, 'b, 'c) psi) list
  and CsQ :: ('c × ('a, 'b, 'c) psi) list

assumes A: ∀φ P. (φ, P) mem CsP ==> ∃ Q. (φ, Q) mem CsQ ∧ guarded Q ∧
(∀Ψ. Ψ ▷ P ≈ Q)
  and B: ∀φ Q. (φ, Q) mem CsQ ==> ∃ P. (φ, P) mem CsP ∧ guarded P ∧

```

$(\forall \Psi. \Psi \triangleright P \doteq Q)$

shows $\Psi \triangleright \text{Cases } CsP \approx \text{Cases } CsQ$
 $\langle \text{proof} \rangle$

lemma *weakPsiCongCasePres*:

fixes $\Psi :: 'b$
and $CsP :: ('c \times ('a, 'b, 'c) \psi) \text{ list}$
and $CsQ :: ('c \times ('a, 'b, 'c) \psi) \text{ list}$

assumes $A: \bigwedge \varphi P. (\varphi, P) \text{ mem } CsP \implies \exists Q. (\varphi, Q) \text{ mem } CsQ \wedge \text{guarded } Q \wedge (\forall \Psi. \Psi \triangleright P \doteq Q)$
and $B: \bigwedge \varphi Q. (\varphi, Q) \text{ mem } CsQ \implies \exists P. (\varphi, P) \text{ mem } CsP \wedge \text{guarded } P \wedge (\forall \Psi. \Psi \triangleright P \doteq Q)$

shows $\Psi \triangleright \text{Cases } CsP \doteq \text{Cases } CsQ$
 $\langle \text{proof} \rangle$

lemma *weakPsiCongResPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $x :: \text{name}$

assumes $\Psi \triangleright P \doteq Q$
and $x \notin \Psi$

shows $\Psi \triangleright (\nu x)P \doteq (\nu x)Q$
 $\langle \text{proof} \rangle$

lemma *weakPsiCongResChainPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $xvec :: \text{name list}$

assumes $\Psi \triangleright P \doteq Q$
and $xvec \notin \Psi$

shows $\Psi \triangleright (\nu * xvec)P \doteq (\nu * xvec)Q$
 $\langle \text{proof} \rangle$

lemma *weakPsiCongParPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $R :: ('a, 'b, 'c) \psi$

assumes $\forall \Psi. \Psi \triangleright P \doteq Q$

```

shows  $\Psi \triangleright P \parallel R \doteq Q \parallel R$ 
⟨proof⟩

end

end

theory Weak-Cong-Struct-Cong
imports Weak-Cong-Pres Weak-Bisim-Struct-Cong
begin

context env begin

lemma weakPsiCongParComm:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

shows  $\Psi \triangleright P \parallel Q \doteq Q \parallel P$ 
⟨proof⟩

lemma weakPsiCongResComm:
fixes  $x :: name$ 
and  $\Psi :: 'b$ 
and  $y :: name$ 
and  $P :: ('a, 'b, 'c) \psi$ 

assumes  $x \notin \Psi$ 
and  $y \notin \Psi$ 

shows  $\Psi \triangleright (\nu x)(\nu y)P \doteq (\nu y)(\nu x)P$ 
⟨proof⟩

lemma weakPsiCongResComm':
fixes  $x :: name$ 
and  $\Psi :: 'b$ 
and  $xvec :: name list$ 
and  $P :: ('a, 'b, 'c) \psi$ 

assumes  $x \notin \Psi$ 
and  $xvec \notin \Psi$ 

shows  $\Psi \triangleright (\nu x)(\nu *xvec)P \doteq (\nu *xvec)(\nu x)P$ 
⟨proof⟩

lemma weakPsiCongScopeExt:
fixes  $x :: name$ 
and  $\Psi :: 'b$ 

```

```

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

assumes x # Ψ
and x # P

shows Ψ ⊢ (λx)(P || Q) ≡ P || (λx)Q
⟨proof⟩

lemma weakPsiCongScopeExtChain:
  fixes xvec :: name list
  and Ψ :: 'b
  and P :: ('a, 'b, 'c) psi
  and Q :: ('a, 'b, 'c) psi

  assumes xvec #* Ψ
  and xvec #* P

  shows Ψ ⊢ (λ*xvec)(P || Q) ≡ P || ((λ*xvec)Q)
⟨proof⟩

lemma weakPsiCongParAssoc:
  fixes Ψ :: 'b
  and P :: ('a, 'b, 'c) psi
  and Q :: ('a, 'b, 'c) psi
  and R :: ('a, 'b, 'c) psi

  shows Ψ ⊢ (P || Q) || R ≡ P || (Q || R)
⟨proof⟩

lemma weakPsiCongParNil:
  fixes P :: ('a, 'b, 'c) psi

  shows Ψ ⊢ P || 0 ≡ P
⟨proof⟩

lemma weakPsiCongResNil:
  fixes x :: name
  and Ψ :: 'b

  assumes x # Ψ

  shows Ψ ⊢ (λx)0 ≡ 0
⟨proof⟩

lemma weakPsiCongOutputPushRes:
  fixes x :: name
  and Ψ :: 'b
  and M :: 'a

```

```

and    $N :: 'a$ 
and    $P :: ('a, 'b, 'c) \psi$ 

assumes  $x \notin \Psi$ 
and      $x \notin M$ 
and      $x \notin N$ 

shows  $\Psi \triangleright (\nu x)(M\langle N \rangle.P) \doteq M\langle N \rangle.(\nu x)P$ 
 $\langle proof \rangle$ 

```

```

lemma weakPsiCongInputPushRes:
  fixes  $x :: name$ 
  and    $\Psi :: 'b$ 
  and    $M :: 'a$ 
  and    $xvec :: name list$ 
  and    $N :: 'a$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  assumes  $x \notin \Psi$ 
  and      $x \notin M$ 
  and      $x \notin xvec$ 
  and      $x \notin N$ 

  shows  $\Psi \triangleright (\nu x)(M(\lambda*xvec N).P) \doteq M(\lambda*xvec N).(\nu x)P$ 
 $\langle proof \rangle$ 

```

```

lemma weakPsiCongCasePushRes:
  fixes  $x :: name$ 
  and    $\Psi :: 'b$ 
  and    $Cs :: ('c \times ('a, 'b, 'c) \psi) list$ 

  assumes  $x \notin \Psi$ 
  and      $x \notin (map fst Cs)$ 

  shows  $\Psi \triangleright (\nu x)(Cases Cs) \doteq Cases(map (\lambda(\varphi, P). (\varphi, (\nu x)P)) Cs)$ 
 $\langle proof \rangle$ 

```

```

lemma weakBangExt:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

  assumes guarded  $P$ 

```

```

  shows  $\Psi \triangleright !P \doteq P \parallel !P$ 
 $\langle proof \rangle$ 

```

```

lemma weakPsiCongParSym:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \psi$ 

```

```

and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $R :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\forall \Psi. \Psi \triangleright P \doteq Q$ 

shows  $\Psi \triangleright R \parallel P \doteq R \parallel Q$ 
⟨proof⟩

lemma weakPsiCongScopeExtSym:
fixes  $x :: \text{name}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $x \notin \Psi$ 
and    $x \notin Q$ 

shows  $\Psi \triangleright (\nu x)(P \parallel Q) \doteq ((\nu x)P) \parallel Q$ 
⟨proof⟩

lemma weakPsiCongScopeExtChainSym:
fixes  $xvec :: \text{name list}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $xvec \notin \Psi$ 
and    $xvec \notin Q$ 

shows  $\Psi \triangleright (\nu * xvec)(P \parallel Q) \doteq ((\nu * xvec)P) \parallel Q$ 
⟨proof⟩

lemma weakPsiCongParPresSym:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $R :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\bigwedge \Psi. \Psi \triangleright P \doteq Q$ 

shows  $\Psi \triangleright R \parallel P \doteq R \parallel Q$ 
⟨proof⟩

lemma tauCongChainBangI:
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $P' :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\Psi \triangleright P \parallel P \Rightarrow_\tau P'$ 
and    $\text{guarded } P$ 

```

obtains Q **where** $\Psi \triangleright !P \implies_{\tau} Q$ **and** $\Psi \triangleright Q \sim P' \parallel !P$
 $\langle proof \rangle$

lemma *weakPsiCongBangPres*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$

assumes $P = Q : \forall \Psi. \Psi \triangleright P \doteq Q$
and $\text{guarded } P$
and $\text{guarded } Q$

shows $\Psi \triangleright !P \doteq !Q$
 $\langle proof \rangle$

end

end

theory *Weak-Congruence*

imports *Weak-Cong-Struct-Cong Bisim-Subst*
begin

context *env* **begin**

definition *weakCongruence* :: $('a, 'b, 'c) \text{ psi} \Rightarrow ('a, 'b, 'c) \text{ psi} \Rightarrow \text{bool} (\dashv \doteq_c \dashv [70, 70] 65)$

where

$P \doteq_c Q \equiv \forall \Psi \sigma. \text{wellFormedSubst } \sigma \longrightarrow \Psi \triangleright P[<\sigma>] \doteq Q[<\sigma>]$

lemma *weakCongE*:

fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$
and $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$

assumes $P \doteq_c Q$
 $\text{wellFormedSubst } \sigma$

shows $\Psi \triangleright P[<\sigma>] \doteq Q[<\sigma>]$

$\langle proof \rangle$

lemma *weakCongI*[case-names *cWeakPsiCong*]:

fixes $P :: ('a, 'b, 'c) \text{ psi}$
and $Q :: ('a, 'b, 'c) \text{ psi}$

assumes $\bigwedge \Psi \sigma. \text{wellFormedSubst } \sigma \implies \Psi \triangleright P[<\sigma>] \doteq Q[<\sigma>]$

shows $P \doteq_c Q$

$\langle proof \rangle$

```
lemma weakCongClosed:  
  fixes  $\Psi :: 'b$   
  and  $P :: ('a, 'b, 'c) \psi$   
  and  $Q :: ('a, 'b, 'c) \psi$   
  and  $p :: name prm$ 
```

```
assumes  $P \doteq_c Q$ 
```

```
shows  $(p \cdot P) \doteq_c (p \cdot Q)$   
 $\langle proof \rangle$ 
```

```
lemma weakCongReflexive:  
  fixes  $\Psi :: 'b$   
  and  $P :: ('a, 'b, 'c) \psi$ 
```

```
shows  $P \doteq_c P$   
 $\langle proof \rangle$ 
```

```
lemma weakCongSym:  
  fixes  $\Psi :: 'b$   
  and  $P :: ('a, 'b, 'c) \psi$   
  and  $Q :: ('a, 'b, 'c) \psi$ 
```

```
assumes  $P \doteq_c Q$ 
```

```
shows  $Q \doteq_c P$   
 $\langle proof \rangle$ 
```

```
lemma weakCongTransitive:  
  fixes  $\Psi :: 'b$   
  and  $P :: ('a, 'b, 'c) \psi$   
  and  $Q :: ('a, 'b, 'c) \psi$   
  and  $R :: ('a, 'b, 'c) \psi$ 
```

```
assumes  $\Psi \triangleright P \doteq Q$   
and  $\Psi \triangleright Q \doteq R$ 
```

```
shows  $\Psi \triangleright P \doteq R$   
 $\langle proof \rangle$ 
```

```
lemma weakCongWeakBisim:  
  fixes  $\Psi :: 'b$   
  and  $P :: ('a, 'b, 'c) \psi$   
  and  $Q :: ('a, 'b, 'c) \psi$ 
```

```
assumes  $P \doteq_c Q$ 
```

shows $\Psi \triangleright P \approx Q$
 $\langle proof \rangle$

lemma *weakCongWeakPsiCong*:
 fixes $\Psi :: 'b$
 and $P :: ('a, 'b, 'c) \psi$
 and $Q :: ('a, 'b, 'c) \psi$

assumes $P \doteq_c Q$

shows $\Psi \triangleright P \doteq Q$
 $\langle proof \rangle$

lemma *strongBisimWeakCong*:
 fixes $\Psi :: 'b$
 and $P :: ('a, 'b, 'c) \psi$
 and $Q :: ('a, 'b, 'c) \psi$

assumes $P \sim_s Q$

shows $P \doteq_c Q$
 $\langle proof \rangle$

lemma *structCongWeakCong*:
 fixes $\Psi :: 'b$
 and $P :: ('a, 'b, 'c) \psi$
 and $Q :: ('a, 'b, 'c) \psi$

assumes $P \equiv_s Q$

shows $P \doteq_c Q$
 $\langle proof \rangle$

lemma *weakCongUnfold*:
 fixes $\Psi :: 'b$
 and $P :: ('a, 'b, 'c) \psi$
 and $Q :: ('a, 'b, 'c) \psi$
 and $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$

assumes $P \doteq_c Q$
and $\text{wellFormedSubst } \sigma$

shows $P[<\sigma>] \doteq_c Q[<\sigma>]$
 $\langle proof \rangle$

lemma *weakCongOutputPres*:
 fixes $\Psi :: 'b$
 and $P :: ('a, 'b, 'c) \psi$
 and $Q :: ('a, 'b, 'c) \psi$

```

and M :: 'a
and N :: 'a

assumes P ≡c Q

shows M⟨N⟩.P ≡c M⟨N⟩.Q
⟨proof⟩

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 ≡c Q
  and distinct xvec

  shows M(λ*xvec N).P ≡c M(λ*xvec N).Q
⟨proof⟩

lemma weakCongCasePresAux:
  fixes Ψ :: 'b
  and CsP :: ('c × ('a, 'b, 'c) psi) list
  and CsQ :: ('c × ('a, 'b, 'c) psi) list

  assumes C1: ∀φ P. (φ, P) mem CsP ⇒ ∃Q. (φ, Q) mem CsQ ∧ guarded Q
  ∧ P ≡c Q
  and C2: ∀φ Q. (φ, Q) mem CsQ ⇒ ∃P. (φ, P) mem CsP ∧ guarded P ∧
  P ≡c Q

  shows Cases CsP ≡c Cases CsQ
⟨proof⟩

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 ≡c Q

  shows P || R ≡c Q || R
⟨proof⟩

lemma weakCongResPres:
  fixes P :: ('a, 'b, 'c) psi
  and Q :: ('a, 'b, 'c) psi

```

```

and    $x :: name$ 

assumes  $P \doteq_c Q$ 

shows  $(\nu x)P \doteq_c (\nu x)Q$ 
 $\langle proof \rangle$ 

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  $!P \doteq_c !Q$ 
 $\langle proof \rangle$ 

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:  $\llbracket length\; xvec = length\; Tvec; distinct\; xvec; xvec \#* (M :: 'a) \rrbracket \implies$ 
   $M[xvec ::= Tvec] = M$ 
begin

lemma ntChanEq[simp]:
  fixes  $\Psi :: 'b$ 
  and    $x :: name$ 

  shows  $\Psi \vdash (nameTerm\; x) \leftrightarrow (nameTerm\; x)$ 
 $\langle proof \rangle$ 

lemma nameTermFresh[simp]:
  fixes  $x :: name$ 
  and    $y :: name$ 

  shows  $x \# (nameTerm\; y) = (x \neq y)$ 
 $\langle proof \rangle$ 

```

```

lemma nameTermFreshChain[simp]:
  fixes xvec :: name list
  and   x   :: name

  shows xvec #* (nameTerm x) = x # xvec
  <proof>

definition tauPrefix :: ('a, 'b, 'c) psi  $\Rightarrow$  ('a, 'b, 'c) psi ( $\langle\tau.\rightarrow [85] 85$ )
  where tauPrefix P  $\equiv$  THE P'.  $\exists x::name.$  x  $\notin P \wedge P' = (\nu x)((nameTerm x)(\lambda*([])(nameTerm x)).\mathbf{0}) \parallel ((nameTerm x)\langle(nameTerm x)\rangle.P)$ 

lemma tauActionUnfold:
  fixes P :: ('a, 'b, 'c) psi
  and   C :: 'd::fs-name

  obtains x::name where x  $\notin P$  and x  $\notin C$  and  $\tau.(P) = (\nu x)((nameTerm x)(\lambda*([])(nameTerm x)).\mathbf{0}) \parallel ((nameTerm x)\langle(nameTerm x)\rangle.P)$ 
  <proof>

lemma tauActionI:
  fixes P :: ('a, 'b, 'c) psi

  shows  $\exists P'. \Psi \triangleright \tau.(P) \longmapsto \tau \prec P' \wedge \Psi \triangleright P \sim P'$ 
  <proof>

lemma outputEmpty[dest]:
  assumes  $\Psi \triangleright M\langle N \rangle.P \longmapsto K(\nu*xvec)\langle N \rangle \prec P'$ 

  shows xvec = []
  <proof>

lemma tauActionE:
  fixes P :: ('a, 'b, 'c) psi

  assumes  $\Psi \triangleright \tau.(P) \longmapsto \tau \prec P'$ 

  shows  $\Psi \triangleright P \sim P' \text{ and } supp P' = ((supp P)::name set)$ 
  <proof>

lemma tauActionEqvt[eqvt]:
  fixes P :: ('a, 'b, 'c) psi
  and   p :: name prm

  shows  $(p \cdot \tau.(P)) = \tau.(p \cdot P)$ 
  <proof>

lemma resCases'[consumes 7, case-names cOpen cRes]:
  fixes  $\Psi \quad :: 'b$ 

```

and $x :: name$
and $P :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a action$
and $P' :: ('a, 'b, 'c) \psi$
and $C :: 'd::fs-name$

assumes $Trans: \Psi \triangleright (\nu x)P \mapsto x\alpha \prec xP'$
and $x \notin \Psi$
and $x \notin x\alpha$
and $x \notin xP'$
and $bn\ x\alpha \notin \Psi$
and $bn\ x\alpha \notin P$
and $bn\ x\alpha \notin subject\ x\alpha$
and $rOpen: \bigwedge M\ xvec\ yvec\ y\ N\ P'. [\Psi \triangleright P \mapsto M(\nu*(xvec@yvec)) \langle ((x, y) \cdot N) \rangle \prec ((x, y)] \cdot P'); y \in supp\ N;$
 $x \notin N; x \notin P'; x \neq y; y \notin xvec; y \notin yvec; y \notin M;$
distinct $xvec; distinct\ yvec;$
 $xvec \notin \Psi; y \notin \Psi; yvec \notin \Psi; xvec \notin P; y \notin P;$
 $yvec \notin P; xvec \notin M; y \notin M;$
 $yvec \notin M; xvec \notin yvec; x\alpha = M(\nu*(xvec@y#yvec)) \langle N \rangle;$
 $xP' = P] \implies Prop$
and $rScope: \bigwedge P'. [\Psi \triangleright P \mapsto x\alpha \prec P'; xP' = (\nu x)P] \implies Prop$

shows $Prop$
[proof]

lemma $parCases' [consumes 5, case-names cPar1 cPar2 cComm1 cComm2]:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) \psi$
and $Q :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a action$
and $T :: ('a, 'b, 'c) \psi$
and $C :: 'd::fs-name$

assumes $Trans: \Psi \triangleright P \parallel Q \mapsto x\alpha \prec xT$
and $bn\ x\alpha \notin \Psi$
and $bn\ x\alpha \notin P$
and $bn\ x\alpha \notin Q$
and $bn\ x\alpha \notin subject\ x\alpha$
and $rPar1: \bigwedge P' A_Q \Psi_Q. [\Psi \otimes \Psi_Q \triangleright P \mapsto x\alpha \prec P'; extractFrame\ Q = A_Q, \Psi_Q]; distinct\ A_Q;$
 $A_Q \notin \Psi; A_Q \notin P; A_Q \notin Q; A_Q \notin x\alpha; A_Q \notin P'; A_Q \notin C; xT = P' \parallel Q] \implies Prop$
and $rPar2: \bigwedge Q' A_P \Psi_P. [\Psi \otimes \Psi_P \triangleright Q \mapsto x\alpha \prec Q'; extractFrame\ P = A_P, \Psi_P]; distinct\ A_P;$
 $A_P \notin \Psi; A_P \notin P; A_P \notin Q; A_P \notin x\alpha; A_P \notin Q'; A_P \notin C; xT = P \parallel Q] \implies Prop$
and $rComm1: \bigwedge \Psi_O M\ N\ P'\ A_P \Psi_P K\ xvec\ Q'\ A_Q.$

$\llbracket \Psi \otimes \Psi_Q \triangleright P \longmapsto M(N) \prec P' ; extractFrame P = \langle A_P, \Psi_P \rangle ; distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \longmapsto K(\nu*xvec)\langle N \rangle \prec Q' ; extractFrame Q = \langle A_Q, \Psi_Q \rangle;$
 $distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; distinct xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q; xvec \#* \Psi_Q; xvec \#* C; x\alpha=\tau; xT = (\nu*xvec)(P' \parallel Q') \implies Prop$
and $rComm2: \bigwedge \Psi_Q M xvec N P' A_P \Psi_P K Q' A_Q.$
 $\llbracket \Psi \otimes \Psi_Q \triangleright P \longmapsto M(\nu*xvec)\langle N \rangle \prec P' ; extractFrame P = \langle A_P, \Psi_P \rangle;$
 $distinct A_P;$
 $\Psi \otimes \Psi_P \triangleright Q \longmapsto K(N) \prec Q' ; extractFrame Q = \langle A_Q, \Psi_Q \rangle ; distinct A_Q;$
 $\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K; distinct xvec;$
 $A_P \#* \Psi; A_P \#* \Psi_Q; A_P \#* P; A_P \#* M; A_P \#* N; A_P \#* P'; A_P \#* Q; A_P \#* xvec; A_P \#* Q'; A_P \#* A_Q; A_P \#* C;$
 $A_Q \#* \Psi; A_Q \#* \Psi_P; A_Q \#* P; A_Q \#* K; A_Q \#* N; A_Q \#* P'; A_Q \#* Q; A_Q \#* xvec; A_Q \#* Q'; A_Q \#* C;$
 $xvec \#* \Psi; xvec \#* \Psi_P; xvec \#* P; xvec \#* M; xvec \#* K; xvec \#* Q; xvec \#* \Psi_Q; xvec \#* C; x\alpha=\tau; xT = (\nu*xvec)(P' \parallel Q') \implies Prop$

shows $Prop$
 $\langle proof \rangle$

lemma $inputCases'[\text{consumes 1, case-names } cInput]:$

fixes $\Psi :: 'b$
and $M :: 'a$
and $xvec :: name list$
and $N :: 'a$
and $P :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a \text{ action}$
and $P' :: ('a, 'b, 'c) \psi$

assumes $Trans: \Psi \triangleright M(\lambda*xvec N).P \longmapsto \alpha \prec P'$

and $rInput: \bigwedge K Tvec. [\Psi \vdash M \leftrightarrow K; set xvec \subseteq supp N; length xvec = length Tvec; distinct xvec; \alpha = K(N[xvec:=Tvec])]; P' = P[xvec:=Tvec] \implies Prop(K(N[xvec:=Tvec])) (P[xvec:=Tvec])$

shows $Prop \alpha P'$
 $\langle proof \rangle$

lemma $outputCases'[\text{consumes 1, case-names } cOutput]:$

fixes $\Psi :: 'b$
and $M :: 'a$
and $N :: 'a$
and $P :: ('a, 'b, 'c) \psi$
and $\alpha :: 'a \text{ action}$
and $P' :: ('a, 'b, 'c) \psi$

```

assumes  $\Psi \triangleright M\langle N \rangle.P \xrightarrow{\alpha} P'$ 
and  $\bigwedge K. [\![\Psi \vdash M \leftrightarrow K; subject \alpha=Some K]\!] \implies Prop(K\langle N \rangle) P$ 

shows  $Prop \alpha P'$ 
(proof)

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  $\Psi \triangleright \tau.(P) \xrightarrow{M(\nu*xvec)} \langle N \rangle \prec P'$ 

shows False
(proof)

lemma tauInput[dest]:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $M :: 'a$ 
and  $N :: 'a$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright \tau.(P) \xrightarrow{M(N)} \prec P'$ 

shows False
(proof)

lemma tauPrefixFrame:
fixes  $P :: ('a, 'b, 'c) \psi$ 

shows  $extractFrame(\tau.(P)) \simeq_F \langle \varepsilon, 1 \rangle$ 
(proof)

lemma insertTauAssertion:
fixes  $P :: ('a, 'b, 'c) \psi$ 
and  $\Psi :: 'b$ 

shows  $insertAssertion(extractFrame(\tau.(P))) \Psi \simeq_F \langle \varepsilon, \Psi \rangle$ 
(proof)

lemma seqSubst4:
assumes  $y \notin \sigma$ 
and  $wellFormedSubst \sigma$ 

```

```

shows substTerm.seqSubst (nameTerm y) σ = nameTerm y
⟨proof⟩

lemma tauSeqSubst[simp]:
  fixes P :: ('a, 'b, 'c) psi
  and σ :: (name list × 'a list) list

  assumes wellFormedSubst σ

  shows (τ.(P))<σ> = τ.(P[<σ>])
⟨proof⟩

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 (τ.(P))[xvec:=Tvec] = τ.(P[xvec:=Tvec])
⟨proof⟩

lemma tauFresh[simp]:
  fixes P :: ('a, 'b, 'c) psi
  and x :: name

  shows x # τ.(P) = x # P
⟨proof⟩

lemma tauFreshChain[simp]:
  fixes P :: ('a, 'b, 'c) psi
  and xvec :: name list

  shows xvec #* (τ.(P)) = (xvec #* P)
⟨proof⟩

lemma guardedTau:
  fixes P :: ('a, 'b, 'c) psi

  shows guarded(τ.(P))
⟨proof⟩

lemma tauChainBisim:
  fixes Ψ :: 'b
  and P :: ('a, 'b, 'c) psi
  and P' :: ('a, 'b, 'c) psi
  and P'' :: ('a, 'b, 'c) psi

```

```

assumes  $\Psi \triangleright P \Rightarrow_{\tau}^{\hat{}} P'$ 
and  $\Psi \triangleright P \sim P''$ 

obtains  $P'''$  where  $\Psi \triangleright P'' \Rightarrow_{\tau}^{\hat{}} P'''$  and  $\Psi \triangleright P' \sim P'''$ 
⟨proof⟩

lemma tauChainStepCons:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes PChain:  $\Psi \triangleright P \Rightarrow_{\tau}^{\hat{}} P'$ 

obtains  $P''$  where  $\Psi \triangleright \tau.(P) \Rightarrow_{\tau}^{\hat{}} P''$  and  $\Psi \triangleright P' \sim P''$ 
⟨proof⟩

lemma tauChainCons:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes  $\Psi \triangleright P \Rightarrow_{\tau}^{\hat{}} P'$ 

obtains  $P''$  where  $\Psi \triangleright \tau.(P) \Rightarrow_{\tau}^{\hat{}} P''$  and  $\Psi \triangleright P' \sim P''$ 
⟨proof⟩

lemma weakTransitionTau:
fixes  $\Psi :: 'b$ 
and  $Q :: ('a, 'b, 'c) \psi$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $\alpha :: 'a \text{ action}$ 
and  $P' :: ('a, 'b, 'c) \psi$ 

assumes PTrans:  $\Psi : Q \triangleright P \Rightarrow_{\alpha}^{\hat{}} P'$ 
and  $\text{bn } \alpha \sharp* \Psi$ 
and  $\text{bn } \alpha \sharp* P$ 

obtains  $P''$  where  $\Psi : Q \triangleright \tau.(P) \Rightarrow_{\alpha}^{\hat{}} P''$  and  $\Psi \triangleright P' \sim P''$ 
⟨proof⟩

end

end

theory Sum
imports Semantics Close-Subst
begin

context env

```

```

begin

abbreviation sumAssertJudge ( $\leftarrow \oplus_{} \rightarrow [150, 50, 50] 150$ )
  where  $(P :: ('a, 'b, 'c) \psi) \oplus_{\varphi} Q \equiv \text{Cases } [(\varphi, P), (\varphi, Q)]$ 

lemma SumAssert1:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $\Psi \triangleright P \mapsto R_s$ 
  and  $\Psi \vdash \varphi$ 
  and  $\text{guarded } P$ 

  shows  $\Psi \triangleright P \oplus_{\varphi} Q \mapsto R_s$ 
  ⟨proof⟩

lemma SumAssert2:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes  $\Psi \triangleright Q \mapsto R_s$ 
  and  $\Psi \vdash \varphi$ 
  and  $\text{guarded } Q$ 

  shows  $\Psi \triangleright P \oplus_{\varphi} Q \mapsto R_s$ 
  ⟨proof⟩

lemma sumAssertCases[consumes 2, case-names cSum1 cSum2]:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $\varphi :: 'c$ 

  assumes  $\Psi \triangleright P \oplus_{\varphi} Q \mapsto R_s$ 
  and  $\Psi \vdash \varphi$ 
  and  $rSum1: [\Psi \triangleright P \mapsto R_s; \text{guarded } P] \implies \text{Prop}$ 
  and  $rSum2: [\Psi \triangleright Q \mapsto R_s; \text{guarded } Q] \implies \text{Prop}$ 

  shows  $\text{Prop}$ 
  ⟨proof⟩

lemma sumElim[dest]:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $\varphi :: 'c$ 

```

```

assumes  $\Psi \triangleright P \oplus_{\varphi} Q \longmapsto R_s$ 
and  $\neg(\Psi \vdash \varphi)$ 

shows False
⟨proof⟩

end

locale sum = env +
  fixes T :: 'c

assumes Top:  $\Psi \vdash T$ 
and TopEqvt[eqvt]:  $((p::name\ prm) \cdot T) = T$ 
and TopSubst[simp]: substCond T xvec Tvec = T
begin

abbreviation topJudge (⊤ 150) where  $\top \equiv T$ 
abbreviation sumJudge (infixr  $\oplus$  80) where  $P \oplus Q \equiv P \oplus_{\top} Q$ 

lemma topSeqSubst[simp]:
  shows (substCond.seqSubst T σ) = T
⟨proof⟩

lemma suppTop:
  shows ((supp(⊤))::name set) = {}
⟨proof⟩

lemma freshTop[simp]:
  fixes x :: name
  and xvec :: name list
  and Xs :: name set

  shows x # ⊤ and xvec #* ⊤ and Xs #* ⊤
⟨proof⟩

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  $(P \oplus Q)[xvec ::= Tvec] = (P[xvec ::= Tvec] \oplus Q[xvec ::= Tvec])$ 
⟨proof⟩

lemma sumSeqSubst[simp]:
  fixes P :: ('a, 'b, 'c) psi

```

```

and    $Q :: ('a, 'b, 'c) \text{ psi}$ 
and    $\sigma :: (\text{name list} \times 'a \text{ list}) \text{ list}$ 

assumes wellFormedSubst  $\sigma$ 

shows  $(P \oplus Q)[<\sigma>] = ((P[<\sigma>]) \oplus (Q[<\sigma>]))$ 
⟨proof⟩

lemma Sum1:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \text{ psi}$ 
  and    $Q :: ('a, 'b, 'c) \text{ psi}$ 

  assumes  $\Psi \triangleright P \longmapsto R_s$ 
  and     guarded  $P$ 

  shows  $\Psi \triangleright P \oplus Q \longmapsto R_s$ 
⟨proof⟩

lemma Sum2:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \text{ psi}$ 
  and    $Q :: ('a, 'b, 'c) \text{ psi}$ 

  assumes  $\Psi \triangleright Q \longmapsto R_s$ 
  and     guarded  $Q$ 

  shows  $\Psi \triangleright P \oplus Q \longmapsto R_s$ 
⟨proof⟩

lemma sumCases[consumes 1, case-names cSum1 cSum2]:
  fixes  $\Psi :: 'b$ 
  and    $P :: ('a, 'b, 'c) \text{ psi}$ 
  and    $Q :: ('a, 'b, 'c) \text{ psi}$ 

  assumes  $\Psi \triangleright P \oplus Q \longmapsto R_s$ 
  and      $rSum1: [\Psi \triangleright P \longmapsto R_s; \text{guarded } P] \implies \text{Prop}$ 
  and      $rSum2: [\Psi \triangleright Q \longmapsto R_s; \text{guarded } Q] \implies \text{Prop}$ 

  shows Prop
⟨proof⟩

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  $\Rightarrow$  ('a, 'b, 'c) psi  $\Rightarrow$  ('a, 'b, 'c) psi ( $\leftrightarrow$ )
[100, 100] 100)
where
bindPrefix (pInput M xvec N) P = M( $\lambda*xvec\ N$ ).P
| bindPrefix (pOutput M N) P = M(N).P
| bindPrefix (pTau) P =  $\tau.(P)$ 
⟨proof⟩

lemma bindPrefixEqvt[eqvt]:
fixes p :: name prm
and α :: 'a prefix
and P :: ('a, 'b, 'c) psi

shows (p  $\cdot$  (α  $\cdot$  P)) = (p  $\cdot$  α)  $\cdot$  (p  $\cdot$  P)
⟨proof⟩

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  $\triangleright$  α  $\cdot$  P  $\longmapsto$  β  $\prec$  P'
and rInput:  $\bigwedge M xvec\ N\ K\ Tvec.\ [\Psi \vdash M \leftrightarrow K; set\ xvec \subseteq supp\ N; length\ xvec = length\ Tvec; distinct\ xvec]$   $\Longrightarrow$ 
Prop (pInput M xvec N) (K(N[xvec:=Tvec]))
(P[xvec:=Tvec])
and rOutput:  $\bigwedge M\ N\ K.\ \Psi \vdash M \leftrightarrow K \Longrightarrow$  Prop (pOutput M N) (K(N))
and rTau: Ψ  $\triangleright$  P  $\sim$  P'  $\Longrightarrow$  Prop (pTau) (τ) P'

shows Prop α β P'
⟨proof⟩

lemma prefixTauCases[consumes 1, case-names cTau]:
fixes α :: 'a prefix
and P :: ('a, 'b, 'c) psi
and P' :: ('a, 'b, 'c) psi

assumes Ψ  $\triangleright$  α  $\cdot$  P  $\longmapsto$  τ  $\prec$  P'

```

and $rTau: \Psi \triangleright P \sim P' \implies Prop(pTau) P'$

shows $Prop \alpha P'$
 $\langle proof \rangle$

lemma $hennessySim1:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Q :: ('a, 'b, 'c) psi$

assumes $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
and $C1: \bigwedge \Psi P Q R. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in Rel] \implies (\Psi, P, R) \in Rel$

shows $\Psi \triangleright \tau.(P) \rightsquigarrow_{Rel} Q$
 $\langle proof \rangle$

lemma $hennessySim2:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Q :: ('a, 'b, 'c) psi$

assumes $PTrans: \Psi \triangleright P \xrightarrow{\tau} \prec P'$
and $P'RelQ: (\Psi, P', Q) \in Rel$
and $C1: \bigwedge \Psi P Q R. [(\Psi, P, Q) \in Rel; \Psi \triangleright Q \sim R] \implies (\Psi, P, R) \in Rel$

shows $\Psi \triangleright P \rightsquigarrow_{Rel} \tau.(Q)$
 $\langle proof \rangle$

lemma $hennessySim3:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Q :: ('a, 'b, 'c) psi$

assumes $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
and $C1: \bigwedge Q'. \Psi \triangleright Q \xrightarrow{\tau} \prec Q' \implies (\Psi, P, Q') \notin Rel$

shows $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
 $\langle proof \rangle$

lemma $tauLaw1SimLeft:$
fixes $\Psi :: 'b$
and $P :: ('a, 'b, 'c) psi$
and $Q :: ('a, 'b, 'c) psi$

assumes $\Psi \triangleright P \rightsquigarrow_{Rel} Q$
and $eqvt Rel$
and $C1: \bigwedge \Psi P Q R. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in Rel] \implies (\Psi, P, R) \in Rel$

shows $\Psi \triangleright \tau.(P) \rightsquigarrow_{Rel} Q$

$\langle proof \rangle$

```

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  $C1: \bigwedge \Psi P Q R. [(\Psi, P, Q) \in Rel; \Psi \triangleright Q \sim R] \implies (\Psi, P, R) \in Rel$ 

  shows  $\Psi \triangleright P \rightsquigarrow_{Rel} \tau.(Q)$ 

```

$\langle proof \rangle$

```

lemma tauLaw3SimLeft:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $\alpha :: 'a$  prefix

  assumes eqvt Rel
  and  $(\Psi, P, Q) \in Rel$ 
  and  $Subst: \bigwedge xvec Tvec. length xvec = length Tvec \implies (\Psi, P[xvec:=Tvec],$ 
 $Q[xvec:=Tvec]) \in Rel$ 
  and  $C1: \bigwedge \Psi P Q R S. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in Rel; \Psi \triangleright R \sim S] \implies (\Psi,$ 
 $P, S) \in Rel$ 
  and  $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in Rel \implies (\Psi \otimes \Psi', P, Q) \in Rel$ 

  shows  $\Psi \triangleright \alpha \cdot (\tau.(P)) \rightsquigarrow_{Rel} \alpha \cdot Q$ 

```

$\langle proof \rangle$

```

lemma tauLaw3SimRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes eqvt Rel
  and  $Subst: \bigwedge \Psi xvec Tvec. length xvec = length Tvec \implies (\Psi, P[xvec:=Tvec],$ 
 $\tau.(Q[xvec:=Tvec])) \in Rel$ 
  and  $C1: \bigwedge \Psi P Q R S. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in Rel; \Psi \triangleright R \sim S] \implies (\Psi,$ 
 $P, S) \in Rel$ 
  and  $\bigwedge \Psi. (\Psi, P, \tau.(Q)) \in Rel$ 

  shows  $\Psi \triangleright \alpha \cdot P \rightsquigarrow_{Rel} \alpha \cdot (\tau.(Q))$ 

```

$\langle proof \rangle$

```

lemma tauLaw3CongSimLeft:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

```

```

and    $Q :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $(\Psi, P, Q) \in \text{Rel}$ 
and    $C1: \bigwedge \Psi P Q R S. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in \text{Rel}; \Psi \triangleright R \sim S] \implies (\Psi, P, S) \in \text{Rel}$ 
and    $rExt: \bigwedge \Psi P Q \Psi'. (\Psi, P, Q) \in \text{Rel} \implies (\Psi \otimes \Psi', P, Q) \in \text{Rel}$ 

shows  $\Psi \triangleright \alpha \cdot (\tau.(P)) \rightsquigarrow \llbracket \text{Rel} \rrbracket \alpha \cdot Q$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{tauLaw3CongSimRight}:$ 
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 
and    $Q :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $(\Psi, P, Q) \in \text{Rel}$ 
and    $C1: \bigwedge \Psi P Q R S. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in \text{Rel}; \Psi \triangleright R \sim S] \implies (\Psi, P, S) \in \text{Rel}$ 
and    $\bigwedge \Psi. (\Psi, P, \tau.(Q)) \in \text{Rel}$ 

shows  $\Psi \triangleright \alpha \cdot P \rightsquigarrow \llbracket \text{Rel} \rrbracket \alpha \cdot (\tau.(Q))$ 
 $\langle \text{proof} \rangle$ 

end

locale  $\text{tauSum} = \text{tau} + \text{sum}$ 
begin

lemma  $\text{tauLaw2SimLeft}:$ 
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $Id: \bigwedge \Psi P. (\Psi, P, P) \in \text{Rel}$ 
and    $C1: \bigwedge \Psi P Q R S. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in \text{Rel}; \Psi \triangleright R \sim S] \implies (\Psi, P, S) \in \text{Rel}$ 

shows  $\Psi \triangleright P \oplus \tau.(P) \rightsquigarrow \llbracket \text{Rel} \rrbracket \tau.(P)$ 
 $\langle \text{proof} \rangle$ 

lemma  $\text{tauLaw2CongSimLeft}:$ 
fixes  $\Psi :: 'b$ 
and    $P :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $Id: \bigwedge \Psi P. (\Psi, P, P) \in \text{Rel}$ 
and    $C1: \bigwedge \Psi P Q R S. [\Psi \triangleright P \sim Q; (\Psi, Q, R) \in \text{Rel}; \Psi \triangleright R \sim S] \implies (\Psi, P, S) \in \text{Rel}$ 

shows  $\Psi \triangleright P \oplus \tau.(P) \rightsquigarrow \llbracket \text{Rel} \rrbracket \tau.(P)$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma tauLaw2SimRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  assumes C1:  $\bigwedge \Psi P Q. \Psi \triangleright P \sim Q \implies (\Psi, P, Q) \in Rel$ 

  shows  $\Psi \triangleright \tau.(P) \rightsquigarrow_{\langle Rel \rangle} P \oplus \tau.(P)$ 
   $\langle proof \rangle$ 

lemma tauLaw2CongSimRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  assumes C1:  $\bigwedge \Psi P Q. \Psi \triangleright P \sim Q \implies (\Psi, P, Q) \in Rel$ 

  shows  $\Psi \triangleright \tau.(P) \rightsquigarrow_{\langle Rel \rangle} P \oplus \tau.(P)$ 
   $\langle proof \rangle$ 

lemma tauLaw4SimLeft:
  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:  $\bigwedge \Psi P Q. \Psi \triangleright P \sim Q \implies (\Psi, P, Q) \in Rel$ 

  shows  $\Psi \triangleright \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q) \rightsquigarrow_{\langle Rel \rangle} \alpha \cdot (\tau.(P) \oplus Q)$ 
   $\langle proof \rangle$ 

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:  $\bigwedge \Psi P Q. \Psi \triangleright P \sim Q \implies (\Psi, P, Q) \in Rel$ 

  shows  $\Psi \triangleright \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q) \rightsquigarrow_{\langle Rel \rangle} \alpha \cdot (\tau.(P) \oplus Q)$ 
   $\langle proof \rangle$ 

lemma tauLaw4SimRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

```

```

and  $\alpha :: 'a \text{ prefix}$ 

assumes  $C1: \bigwedge \Psi P Q. \Psi \triangleright P \sim Q \implies (\Psi, P, Q) \in Rel$ 
and  $\bigwedge \Psi P. (\Psi, P, P) \in Rel$ 

shows  $\Psi \triangleright \alpha \cdot (\tau.(P) \oplus Q) \rightsquigarrow_{Rel} \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q)$ 
 $\langle proof \rangle$ 

lemma tauLaw4CongSimRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\alpha :: 'a \text{ prefix}$ 

assumes  $C1: \bigwedge \Psi P Q. \Psi \triangleright P \sim Q \implies (\Psi, P, Q) \in Rel$ 

shows  $\Psi \triangleright \alpha \cdot (\tau.(P) \oplus Q) \rightsquigarrow_{Rel} \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q)$ 
 $\langle proof \rangle$ 

end

end

theory Tau-Stat-Imp
  imports Tau-Sim Weaken-Stat-Imp
begin

locale weakTauLaws = weak + tau
begin

lemma tauLaw1StatImpLeft:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\Psi \triangleright P \lesssim_w Rel Q$ 
and  $(\Psi, \tau.(P), Q) \in Rel$ 

shows  $\Psi \triangleright \tau.(P) \lesssim_w Rel Q$ 
 $\langle proof \rangle$ 

lemma tauLaw1StatImpRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 

assumes  $\Psi \triangleright P \lesssim Rel Q$ 
and  $C1: \bigwedge \Psi P Q R. [(\Psi, P, Q) \in Rel; \Psi \triangleright Q \sim R] \implies (\Psi, P, R) \in Rel'$ 

```

```

shows  $\Psi \triangleright P \lessapprox_{\text{Rel}'} \tau.(Q)$ 
⟨proof⟩

end

context tau begin

lemma tauLaw3StatImpLeft:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 
  and  $\alpha :: 'a \text{ prefix}$ 

  assumes C1:  $\bigwedge \Psi'. (\Psi \otimes \Psi', \alpha \cdot (\tau.(P)), \alpha \cdot Q) \in \text{Rel}$ 

  shows  $\Psi \triangleright \alpha \cdot (\tau.(P)) \lessapprox_{\text{Rel}} \alpha \cdot Q$ 
⟨proof⟩

lemma tauLaw3StatImpRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 
  and  $Q :: ('a, 'b, 'c) \psi$ 

  assumes C1:  $\bigwedge \Psi'. (\Psi \otimes \Psi', \alpha \cdot P, \alpha \cdot (\tau.(Q))) \in \text{Rel}$ 

  shows  $\Psi \triangleright \alpha \cdot P \lessapprox_{\text{Rel}} \alpha \cdot (\tau.(Q))$ 
⟨proof⟩

end

context tauSum begin

lemma tauLaw2StatImpLeft:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  assumes C1:  $\bigwedge \Psi'. (\Psi \otimes \Psi', P \oplus \tau.(P), \tau.(P)) \in \text{Rel}$ 

  shows  $\Psi \triangleright P \oplus \tau.(P) \lessapprox_{\text{Rel}} \tau.(P)$ 
⟨proof⟩

lemma tauLaw2StatImpRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  assumes C1:  $\bigwedge \Psi'. (\Psi \otimes \Psi', \tau.(P), P \oplus \tau.(P)) \in \text{Rel}$ 

  shows  $\Psi \triangleright \tau.(P) \lessapprox_{\text{Rel}} P \oplus \tau.(P)$ 
⟨proof⟩

```

```

lemma tauLaw4StatImpLeft:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $M :: 'a$ 
  and  $N :: 'a$ 

  assumes C1:  $\bigwedge \Psi'. (\Psi \otimes \Psi', \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q), \alpha \cdot (\tau.(P) \oplus Q)) \in \text{Rel}$ 

  shows  $\Psi \triangleright \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q) \lesssim_{\text{Rel}} \alpha \cdot (\tau.(P) \oplus Q)$ 
   $\langle \text{proof} \rangle$ 

lemma tauLaw4StatImpRight:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 
  and  $Q :: ('a, 'b, 'c) \text{ psi}$ 
  and  $\alpha :: 'a \text{ prefix}$ 

  assumes C1:  $\bigwedge \Psi'. (\Psi \otimes \Psi', \alpha \cdot (\tau.(P) \oplus Q), \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q)) \in \text{Rel}$ 

  shows  $\Psi \triangleright \alpha \cdot (\tau.(P) \oplus Q) \lesssim_{\text{Rel}} \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q)$ 
   $\langle \text{proof} \rangle$ 

end

end

theory Tau-Laws-Weak
  imports Weaken-Bisimulation Weak-Congruence Tau-Sim Tau-Stat-Imp
begin

context weakTauLaws begin

lemma tauLaw1:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 

  shows  $\Psi \triangleright \tau.(P) \approx P$ 
   $\langle \text{proof} \rangle$ 

lemma tauLaw3:
  fixes  $\Psi :: 'b$ 
  and  $\alpha :: 'a \text{ prefix}$ 
  and  $P :: ('a, 'b, 'c) \text{ psi}$ 

  shows  $\Psi \triangleright \alpha \cdot (\tau.(P)) \approx \alpha \cdot P$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma tauLaw3PsiCong:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright \alpha \cdot (\tau.(P)) \doteq \alpha \cdot P$ 
   $\langle proof \rangle$ 

lemma tauLaw3Cong:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  shows  $\alpha \cdot (\tau.(P)) \doteq_c \alpha \cdot P$ 
   $\langle proof \rangle$ 

end

end

theory Tau-Laws-No-Weak
  imports Tau-Sim Tau-Stat-Imp
begin

context tauSum
begin

lemma tauLaw2:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \oplus \tau.(P) \approx \tau.(P)$ 
   $\langle proof \rangle$ 

lemma tauLawPsiCong2:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  shows  $\Psi \triangleright P \oplus \tau.(P) \doteq \tau.(P)$ 
   $\langle proof \rangle$ 

lemma tauLawCong2:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

  shows  $P \oplus \tau.(P) \doteq_c \tau.(P)$ 
   $\langle proof \rangle$ 

lemma tauLaw4:
  fixes  $\Psi :: 'b$ 
  and  $P :: ('a, 'b, 'c) \psi$ 

```

```

and  $\alpha :: 'a\ prefix$ 
shows  $\Psi \triangleright \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q) \approx \alpha \cdot (\tau.(P) \oplus Q)$ 
 $\langle proof \rangle$ 

lemma tauLaw4PsiCong:
fixes  $\Psi :: 'b$ 
and  $P :: ('a, 'b, 'c) \psi$ 
and  $\alpha :: 'a\ prefix$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

shows  $\Psi \triangleright \alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q) \doteq \alpha \cdot (\tau.(P) \oplus Q)$ 
 $\langle proof \rangle$ 

lemma tauLaw4Cong:
fixes  $P :: ('a, 'b, 'c) \psi$ 
and  $\alpha :: 'a\ prefix$ 
and  $Q :: ('a, 'b, 'c) \psi$ 

shows  $\alpha \cdot P \oplus \alpha \cdot (\tau.(P) \oplus Q) \doteq_c \alpha \cdot (\tau.(P) \oplus Q)$ 
 $\langle proof \rangle$ 

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.