

HOL-CSP_OpSem – Operational Semantics
formally proven in HOL-CSP

Benoît Ballenghien and Burkhart Wolff

April 18, 2024

Abstract

Recently, a modern version of Roscoe and Brookes [2] Failure-Divergence Semantics for CSP has been formalized in Isabelle [6] and extended [1]. The resulting framework is purely denotational and, given the possibility to define arbitrary events in a HOL-type, more expressive than the original.

However, there is a need for an operational semantics for CSP. From the latter, model-checkers, symbolic execution engines for test-case generators, and animators and simulators can be constructed. In the literature, a few versions of operational semantics for CSP have been proposed, where it is assumed, of course, that denotational and operational constructs coincide, but this is not obvious at first glance. Recently, a modern version of Roscoe and Brookes [2] Failure-Divergence Semantics for CSP has been formalized in Isabelle [6] and extended [1]. The resulting framework is purely denotational and, given the possibility to define arbitrary events in a HOL-type, more expressive than the original.

However, there is a need for an operational semantics for CSP. From the latter, model-checkers, symbolic execution engines for test-case generators, and animators and simulators can be constructed. In the literature, a few versions of operational semantics for CSP have been proposed, where it is assumed, of course, that denotational and operational constructs coincide, but this is not obvious at first glance.

The present work addresses this issue by providing the first (to our knowledge) formal theory of operational behavior derived from HOL-CSP via a bridge definition between the denotational and the operational semantics. In fact, several possibilities are discussed.

As a bonus, we have defined three new operators: Sliding, Throw and Interrupt which are of particular pragmatic interest in operational semantics. Moreover, we have proven new “laws” for HOL-CSP improving the latter.

The present work addresses this issue by providing the (to our knowledge) first formal theory of operational behaviour derived from HOL-CSP via a bridge definition between the denotational and the operational semantics. In fact, several possibilities are discussed.

As a bonus, we have defined three new operators: Sliding, Throw and Interrupt which are of particular pragmatic interest in operational semantics. Moreover, we have proven new “laws” for HOL-CSP improving the latter.

Contents

1	Introduction	9
1.1	Motivations	9
1.2	The Global Architecture of HOL-CSP_OpSem	11
2	New Operators	13
2.1	The Sliding Operator (also called Timeout)	13
2.1.1	Definition	13
2.1.2	Projections	13
2.1.3	Monotony	14
2.1.4	Properties	14
2.1.5	Continuity	15
2.2	The Throw Operator	15
2.2.1	Definition	16
2.2.2	Projections	19
2.2.3	Monotony	19
2.2.4	Properties	23
2.2.5	Key Property	27
2.2.6	Continuity	31
2.3	The Interrupt Operator	36
2.3.1	Definition	36
2.3.2	Projections	41
2.3.3	Monotony	42
2.3.4	Properties	43
2.3.5	Key Property	51
2.3.6	Continuity	54
3	The Ready Set Notion	63
3.1	Definition	63
3.2	Anti-Mono Rules	64
3.3	Behaviour of <i>ready-set</i> with <i>STOP</i> , <i>SKIP</i> and \perp	65
3.4	Behaviour of <i>ready-set</i> with Operators of HOL-CSP	65
3.5	Behaviour of <i>ready-set</i> with Operators of HOL-CSPM	74
3.6	Behaviour of <i>ready-set</i> with Operators of HOL-CSP_OpSem	75

3.7	Behaviour of <i>ready-set</i> with Reference Processes	76
4	Construction of the After Operator	77
4.1	Definition	77
4.2	Projections	78
4.3	Monotony	79
4.4	Behaviour of <i>After</i> with <i>STOP</i> , <i>SKIP</i> and \perp	81
4.5	Behaviour of <i>After</i> with Operators of HOL-CSP	81
4.5.1	Loss of Determinism	81
4.5.2	<i>After</i> Sequential Composition	83
4.5.3	<i>After</i> Synchronization	90
4.5.4	<i>After</i> Hiding Operator	101
4.5.5	<i>After</i> Renaming	104
4.6	Behaviour of <i>After</i> with Operators of HOL-CSPM	107
4.7	Behaviour of <i>After</i> with Operators of HOL-CSP_OpSem	108
4.7.1	<i>After</i> Sliding	108
4.7.2	<i>After</i> Throwing	108
4.7.3	<i>After</i> Interrupting	112
4.8	Behaviour of <i>After</i> with Reference Processes	114
5	Extension of the After Operator	117
5.1	The AfterExt Operator	117
5.1.1	Definition	117
5.1.2	Projections	117
5.1.3	Monotony	118
5.1.4	Behaviour of <i>AfterExt</i> with <i>STOP</i> , <i>SKIP</i> and \perp	119
5.1.5	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP	120
5.1.6	Behaviour of <i>AfterExt</i> with Operators of HOL-CSPM	122
5.1.7	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP_OpSem	123
5.1.8	Behaviour of <i>AfterExt</i> with Reference Processes	123
5.2	The AfterTrace Operator	125
5.2.1	Definition	125
5.2.2	Projections	125
5.2.3	Monotony	126
5.2.4	Another Definition of <i>events-of</i>	126
5.2.5	Characterizations for Deadlock Freeness	129
6	Motivations for our Definitions	131
7	Generic Operational Semantics as a Locale	135
7.1	Definition	135
7.2	Consequences of $P \rightsquigarrow^*_s Q$ on \mathcal{F} , \mathcal{T} and \mathcal{D}	137
7.3	Characterizations for $P \rightsquigarrow^*_s Q$	138
7.4	Finally: $P \rightsquigarrow^*_s Q$ is $P \text{ afterTrace } s \rightsquigarrow_{\mathcal{T}} Q$	140

7.5	General Rules of Operational Semantics	142
8	Failure Divergence Operational Semantics	147
8.1	Operational Semantics Laws	147
8.2	Reality Checks	151
8.3	Other Results	152
8.4	Summary: Operational Rules	152
9	Trace Divergence Operational Semantics	157
9.1	Operational Semantics Laws	157
9.2	Reality Checks	161
9.3	Other Results	161
9.4	Summary: Operational Rules	162
10	Extension of the After Operator, bis	165
10.1	The AfterExt Operator, bis	165
10.1.1	Definition	165
10.1.2	Projections	165
10.1.3	Monotony	166
10.1.4	Behaviour of <i>AfterExt</i> with <i>STOP</i> , <i>SKIP</i> and \perp . . .	167
10.1.5	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP . .	167
10.1.6	Behaviour of <i>AfterExt</i> with Operators of HOL-CSPM . .	170
10.1.7	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP_OpSem	170
10.1.8	Behaviour of <i>AfterExt</i> with Reference Processes	171
10.2	The AfterTrace Operator, bis	172
10.2.1	Definition	172
10.2.2	Projections	173
10.2.3	Monotony	174
10.2.4	Another Definition of <i>events-of</i>	174
10.2.5	Characterizations for Deadlock Freeness	176
11	Generic Operational Semantics as a Locale, bis	179
11.1	Definition	179
11.2	Consequences of $P \rightsquigarrow^*_s Q$ on \mathcal{F} , \mathcal{T} and \mathcal{D}	181
11.3	Characterizations for $P \rightsquigarrow^*_s Q$	182
11.4	Finally: $P \rightsquigarrow^*_s Q$ is P <i>afterTrace</i> $s \rightsquigarrow_{\mathcal{T}} Q$	184
11.5	General Rules of Operational Semantics	186
12	Failure Divergence Operational Semantics, bis	191
12.1	Operational Semantics Laws	191
12.2	Reality Checks	195
12.3	Other Results	196
12.4	Summary: Operational Rules	196

13 Trace Divergence Operational Semantics, bis	201
13.1 Operational Semantics Laws	201
13.2 Reality Checks	205
13.3 Other Results	206
13.4 Summary: Operational Rules	206
14 Failure Operational Semantics, bis	211
14.1 Operational Semantics Laws	212
14.2 Reality Checks	214
14.3 Other Results	215
14.4 Nicely written operational rules	215
15 Trace Operational Semantics, bis	219
15.1 Operational Semantics Laws	219
15.2 Reality Checks	222
15.3 Other Results	223
15.4 Summary: Operational Rules	223
16 Bonus: powerful new Laws	227
16.1 Powerful Results about <i>Renaming</i>	227
16.1.1 Some Generalizations	227
16.1.2 <i>Renaming</i> and <i>Hiding</i>	230
16.1.3 <i>Renaming</i> and <i>Sync</i>	236
16.2 <i>Hiding</i> and <i>Mprefix</i>	243
16.2.1 Two intermediate Results	243
16.2.2 <i>Hiding</i> and <i>Mprefix</i> for disjoint Sets	245
16.2.3 <i>Hiding</i> and <i>Mprefix</i> for non-disjoint Sets	249
16.3 (\triangleright) behaviour	258
17 Conclusion	265

Chapter 1

Introduction

1.1 Motivations

HOL-CSP [6] is a formalization in Isabelle/HOL of the work of Hoare and Roscoe on the denotational semantics of the Failure/Divergence Model of CSP. It follows essentially the presentation of CSP in Roscoe's Book "Theory and Practice of Concurrency" [3] and the semantic details in a joint paper of Roscoe and Brooks "An improved failures model for communicating processes" [2].

Basically, the session HOL-CSP introduces the type $'\alpha$ *process*, several classic CSP operators and number of "laws" (i.e. derived equations) that govern their interactions. HOL-CSP has been extended by a theory of architectural operators HOL-CSPM inspired by the CSP_M language of the model-checker FDR. While in FDR these operators are basically macros over finite lists and sets, the HOL-CSPM theory treats them in their own right for the most general cases.

The present work addresses the problem of operational semantics for CSP which are the foundations for finite model-checking and process simulation techniques. In the literature, there are a few versions of operational semantics for CSP, which lend themselves to the constructions of labelled transition systems (LTS). Of course, denotational and operational constructs are expected to coincide, but this is not obvious at first glance. As a key contribution, we will define the operational derivation operators $P \rightsquigarrow_\tau Q$ (" P evolves internally to Q ") and $P \rightsquigarrow_e Q$ (" P evolves to Q by emitting e ") in terms of the denotational semantics and derive the expected laws for operational semantics from these.

Additionally, we developed the theory of the interrupt operators *Sliding*, *Throw* and *Interrupt* [4] which have been traditionally introduced in the context of operational semantics. This part of the present theory reintroduces denotational semantics for these operators and constructs on this basis

the operational laws for them.

The overall objective of this work is to provide a formal, machine checked foundation for the laws provided by Roscoe in [3, 5]. In several places, our formalization efforts led to slight modifications of the original definitions in order to achieve the goal of a combined integrated theory. In some cases – in particular in connection with the *Interrupt* operator definition – some corrections have been necessary since the fundamental invariants were not respected.

1.2 The Global Architecture of HOL-CSP_OpSem

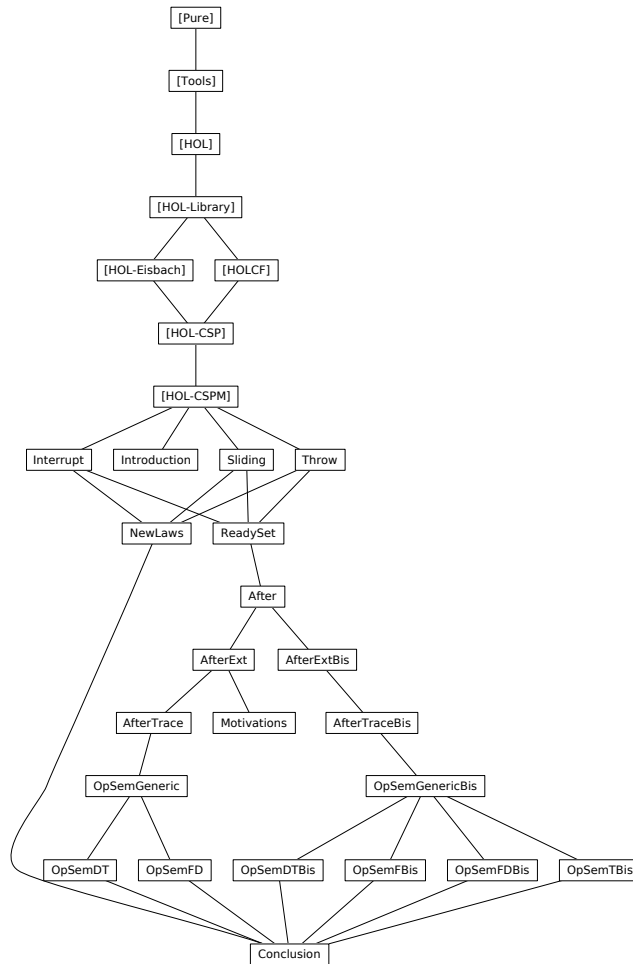


Figure 1.1: The overall architecture

The global architecture of HOL-CSP_OpSem is shown in [Figure 1.1](#).

The package resides on:

- HOL-CSP 2.0 from the Isabelle Archive of Formal Proofs
- HOL-CSPM from the Isabelle Archive of Formal Proofs.

Chapter 2

New Operators

Three operators of CSP has not been defined yet in HOL-CSP (and not in HOL-CSPM either): Sliding, Interrupt and Throw. Since they are mentioned by Roscoe [4] (and since he provides operational laws for them too in [5]), it would be a shame not to include them in our work.

We will therefore define them now before moving on to the construction of our correspondence between semantics.

2.1 The Sliding Operator (also called Timeout)

```
theory Sliding
  imports HOL-CSPM.CSPM
begin
```

2.1.1 Definition

definition *Sliding* :: $\langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow 'a \text{ process} \rangle$ (**infixl** $\langle \triangleright \rangle$ 78)
 where $\langle P \triangleright Q \equiv (P \sqcap Q) \sqcap Q \rangle$

— See if we want to define a MultiSliding operator like MultiSeq.

2.1.2 Projections

lemma *F-Sliding*:

$$\langle \mathcal{F} (P \triangleright Q) = \mathcal{F} Q \cup \{(s, X). s \neq [] \wedge (s, X) \in \mathcal{F} P \vee s = [] \wedge (s \in \mathcal{D} P \vee tick \notin X \wedge [tick] \in \mathcal{T} P)\} \rangle$$

by (auto simp add: Sliding-def F-Ndet F-Det NF-ND is-processT6-S2)

corollary $\langle \mathcal{F} (Mprefix A P \triangleright Q) = \mathcal{F} Q \cup \{(s, X) \in \mathcal{F} (Mprefix A P). s \neq []\} \rangle$
 by (auto simp add: F-Sliding)

lemma *D-Sliding*: $\langle \mathcal{D} (P \triangleright Q) = \mathcal{D} P \cup \mathcal{D} Q \rangle$
 by (simp add: Sliding-def D-Ndet D-Det)

lemma *T-Sliding*: $\langle \mathcal{T} (P \triangleright Q) = \mathcal{T} P \cup \mathcal{T} Q \rangle$
by (*simp add: Sliding-def T-Ndet T-Det*)

2.1.3 Monotony

lemma *mono-right-Sliding-F*: $\langle Q \sqsubseteq_F Q' \implies P \triangleright Q \sqsubseteq_F P \triangleright Q' \rangle$
and *mono-Sliding-D*: $\langle P \sqsubseteq_D P' \implies Q \sqsubseteq_D Q' \implies P \triangleright Q \sqsubseteq_D P' \triangleright Q' \rangle$
and *mono-Sliding-T*: $\langle P \sqsubseteq_T P' \implies Q \sqsubseteq_T Q' \implies P \triangleright Q \sqsubseteq_T P' \triangleright Q' \rangle$
and *mono-Sliding-FD*: $\langle P \sqsubseteq_{FD} P' \implies Q \sqsubseteq_{FD} Q' \implies P \triangleright Q \sqsubseteq_{FD} P' \triangleright Q' \rangle$
and *mono-Sliding-DT*: $\langle P \sqsubseteq_{DT} P' \implies Q \sqsubseteq_{DT} Q' \implies P \triangleright Q \sqsubseteq_{DT} P' \triangleright Q' \rangle$
by (*simp add: failure-refine-def F-Sliding subset-iff*)
(simp-all add: Sliding-def)

2.1.4 Properties

lemma *Sliding-id*: $\langle P \triangleright P = P \rangle$
by (*simp add: Det-id Ndet-id Sliding-def*)

lemma *STOP-Sliding*: $\langle STOP \triangleright P = P \rangle$
unfolding *Sliding-def* **by** (*simp add: Det-commute Det-STOP Ndet-id*)

Of course, $P \triangleright STOP \neq STOP$ and $P \triangleright STOP \neq P$ in general.

lemma $\langle \exists P. P \triangleright STOP \neq STOP \wedge P \triangleright STOP \neq P \rangle$
proof (*intro exI*)
show $\langle SKIP \triangleright STOP \neq STOP \wedge SKIP \triangleright STOP \neq SKIP \rangle$
by (*metis Det-STOP Ndet-commute SKIP-F-iff SKIP-Neq-STOP STOP-F-iff Sliding-def mono-Ndet-F-left*)

qed

But we still have this result.

lemma *Sliding-is-STOP-iff*: $\langle P \triangleright Q = STOP \iff P = STOP \wedge Q = STOP \rangle$
by (*auto simp add: STOP-iff-T T-Sliding intro: Nil-elem-T*)

lemma *Sliding-STOP-Det*: $\langle (P \triangleright STOP) \sqcap Q = P \triangleright Q \rangle$
by (*simp add: Det-STOP Det-commute Det-distrib Sliding-def*)

lemma *BOT-Sliding*: $\langle \perp \triangleright P = \perp \rangle$
and *Sliding-BOT*: $\langle P \triangleright \perp = \perp \rangle$
unfolding *Sliding-def* **by** (*simp-all add: Det-commute Det-BOT Ndet-commute Ndet-BOT*)

lemma *Sliding-is-BOT-iff*: $\langle P \triangleright Q = \perp \iff P = \perp \vee Q = \perp \rangle$
by (*simp add: Det-is-BOT-iff Ndet-is-BOT-iff Sliding-def*)

lemma *Sliding-assoc*: $\langle P1 \triangleright P2 \triangleright P3 = P1 \triangleright (P2 \triangleright P3) \rangle$
by (*metis Det-assoc Det-commute Det-distrib Ndet-assoc*)

Ndet-commute Ndet-distrib Ndet-id Sliding-def)

lemma *SKIP-Sliding*: $\langle \text{SKIP} \triangleright P = P \sqcap \text{SKIP} \rangle$
by (*auto simp add: Sliding-def Process-eq-spec F-Ndet*
F-Det T-SKIP D-Ndet D-Det F-SKIP D-SKIP NF-ND)

lemma *Sliding-SKIP*: $\langle P \triangleright \text{SKIP} = P \sqcap \text{SKIP} \rangle$
by (*auto simp add: Sliding-def Process-eq-spec F-Ndet*
F-Det T-SKIP D-Ndet D-Det F-SKIP D-SKIP)

lemma *Sliding-Det*: $\langle (P \triangleright P') \sqcap Q = P \triangleright P' \sqcap Q \rangle$
by (*metis Det-assoc Det-commute Det-distrib Sliding-def*)

lemma *Sliding-Ndet*: $\langle (P \sqcap P') \triangleright Q = (P \triangleright Q) \sqcap (P' \triangleright Q) \rangle$
 $\langle P \triangleright (Q \sqcap Q') = (P \triangleright Q) \sqcap (P \triangleright Q') \rangle$
by (*auto simp add: Process-eq-spec F-Ndet D-Ndet T-Ndet F-Sliding D-Sliding*)

lemma *Renaming-Sliding*:
 $\langle \text{Renaming } (P \triangleright Q) f = \text{Renaming } P f \triangleright \text{Renaming } Q f \rangle$
by (*simp add: Renaming-Det Renaming-Ndet Sliding-def*)

lemma *events-Sliding*: $\langle \text{events-of } (P \triangleright Q) = \text{events-of } P \cup \text{events-of } Q \rangle$
unfolding *Sliding-def* **by** (*simp add: events-of-def T-Det T-Ndet*)

2.1.5 Continuity

From the definition, continuity is obvious.

lemma *Sliding-cont[simp]* : $\langle \text{cont } f \implies \text{cont } g \implies \text{cont } (\lambda x. f x \triangleright g x) \rangle$
by (*simp add: Sliding-def*)

end

2.2 The Throw Operator

theory *Throw*
imports *HOL-CSPM.CSPM*
begin

2.2.1 Definition

The Throw operator allows error handling. Whenever an error (or more generally any event $ev \in ev \text{ ' } A$) occurs in P , P is shut down and $Q \text{ } e$ is started.

This operator can somehow be seen as a generalization of sequential composition ($;$): P terminates on any event in $ev \text{ ' } A$ rather than $tick$ (however it do not hide these events like $;$) do for $tick$, but we can use an additional $\lambda P. P \setminus A$).

This is a relatively new addition to CSP (see [4, p.140]).

lift-definition $Throw :: \langle [\alpha \text{ process}, \alpha \text{ set}, \alpha \Rightarrow \alpha \text{ process}] \Rightarrow \alpha \text{ process} \rangle$

is $\langle \lambda P A Q.$

$$\begin{aligned} & \{ (t1, X) \in \mathcal{F} P. \text{set } t1 \cap ev \text{ ' } A = \{\} \} \cup \\ & \{ (t1 @ t2, X) \mid t1 \ t2 \ X. t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \\ & \quad \text{set } t1 \cap ev \text{ ' } A = \{\} \wedge \text{front-tickFree } t2 \} \cup \\ & \{ (t1 @ ev \ a \ # \ t2, X) \mid t1 \ a \ t2 \ X. t1 @ [ev \ a] \in \mathcal{T} P \wedge \text{set } t1 \cap ev \text{ ' } A = \{\} \wedge \\ & \quad a \in A \wedge (t2, X) \in \mathcal{F} (Q \ a) \}, \\ & \{ t1 @ t2 \mid t1 \ t2. t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \\ & \quad \text{set } t1 \cap ev \text{ ' } A = \{\} \wedge \text{front-tickFree } t2 \} \cup \\ & \{ t1 @ ev \ a \ # \ t2 \mid t1 \ a \ t2. t1 @ [ev \ a] \in \mathcal{T} P \wedge \text{set } t1 \cap ev \text{ ' } A = \{\} \wedge \\ & \quad a \in A \wedge t2 \in \mathcal{D} (Q \ a) \} \rangle \end{aligned}$$

proof –

show $\langle ?thesis \ P \ A \ Q \rangle$ (**is** $\langle is\text{-process} \ (?f, ?d) \rangle$) **for** $P \ A \ Q$

unfolding $is\text{-process}\text{-def}$ $FAILURES\text{-def}$ $DIVERGENCES\text{-def}$ $fst\text{-conv}$ $snd\text{-conv}$

proof ($intro \ conjI$ $allI$ $impI$; ($elim \ conjE$)?)

show $\langle [], \{\} \rangle \in ?f$ **by** ($simp \ add: is\text{-process}T1$)

next

show $\langle s, X \rangle \in ?f \implies \text{front-tickFree } s$ **for** $s \ X$

apply ($simp, elim \ disjE \ exE$)

subgoal by ($metis is\text{-process}T$)

subgoal by ($solves \ \langle simp \ add: \text{front-tickFree-append} \rangle$)

by ($metis \ F\text{-T} \ \text{append-Cons} \ \text{append-Nil} \ \text{append-T-imp-tickFree} \ \text{butlast.simps}(2)$ $event.simps(3)$)

$\text{front-tickFree-append}$ $is\text{-process}T2\text{-TR}$ $last\text{-ConsL}$ $not\text{-Cons-self}$ $tickFree\text{-butlast}$)

next

show $\langle s @ t, \{\} \rangle \in ?f \implies \langle s, \{\} \rangle \in ?f$ **for** $s \ t$

proof ($induct \ t \ \text{rule: rev-induct}$)

case Nil

thus $\langle s, \{\} \rangle \in ?f$ **by** $simp$

next

case ($snoc \ b \ t$)

consider $\langle s @ t @ [b], \{\} \rangle \in \mathcal{F} P$ $\langle (\text{set } s \cup \text{set } t) \cap ev \text{ ' } A = \{\} \rangle$

$\mid \langle \exists t1 \ t2. s @ t @ [b] = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge$

$\text{set } t1 \cap ev \text{ ' } A = \{\} \wedge \text{front-tickFree } t2 \vee$

$(\exists a. s @ t @ [b] = t1 @ ev \ a \ # \ t2 \wedge t1 @ [ev \ a] \in \mathcal{T} P \wedge$

$\text{set } t1 \cap ev \text{ ' } A = \{\} \wedge a \in A \wedge (t2, \{\}) \in \mathcal{F} (Q \ a) \rangle$

using $snoc.premis$ **by** $simp \ blast$


```

thus  $\langle (s, \{\}) \in ?f \rangle$ 
proof cases
  show  $\langle (s @ t @ [b], \{\}) \in \mathcal{F} P \implies (set\ s \cup set\ t) \cap ev\ 'A = \{\} \implies (s,$ 
 $\{\}) \in ?f \rangle$ 
    by (drule is-processT3[rule-format]) (simp add: Int-Un-distrib2)
  next
    assume  $\langle \exists t1\ t2. s @ t @ [b] = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge tickFree\ t1 \wedge$ 
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \vee$ 
 $(\exists a. s @ t @ [b] = t1 @ ev\ a \# t2 \wedge t1 @ [ev\ a] \in \mathcal{T} P \wedge$ 
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge (t2, \{\}) \in \mathcal{F} (Q\ a)) \rangle$ 
    (is  $\langle \exists t1\ t2. ?disj\ t1\ t2 \rangle$ )
    then obtain  $t1\ t2$  where  $\langle ?disj\ t1\ t2 \rangle$  by blast
    show  $\langle (s, \{\}) \in ?f \rangle$ 
    apply (rule snoc.hyps)
    using  $\langle ?disj\ t1\ t2 \rangle$  apply (elim disjE exE)
    apply (all  $\langle cases\ t2\ rule: rev-cases \rangle$ , simp-all)
    subgoal by (metis Int-Un-distrib2 T-F D-T Un-empty append.assoc
is-processT3-SR set-append)
    subgoal by (metis front-tickFree-dw-closed)
    subgoal by (meson T-F process-charn)
    by (metis process-charn)
  qed
qed
next
  show  $\langle (s, Y) \in ?f \implies X \subseteq Y \implies (s, X) \in ?f \rangle$  for  $s\ X\ Y$ 
    by simp (metis is-processT4)
next
  fix  $s\ X\ Y$ 
  assume assms :  $\langle (s, X) \in ?f \rangle \langle \forall c. c \in Y \longrightarrow (s @ [c], \{\}) \notin ?f \rangle$ 
  consider  $\langle (s, X) \in \mathcal{F} P \rangle \langle set\ s \cap ev\ 'A = \{\} \rangle$ 
    |  $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge tickFree\ t1 \wedge$ 
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \rangle$ 
    |  $\langle \exists t1\ a\ t2. s = t1 @ ev\ a \# t2 \wedge t1 @ [ev\ a] \in \mathcal{T} P \wedge$ 
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q\ a) \rangle$ 
  using assms(1) by blast
thus  $\langle (s, X \cup Y) \in ?f \rangle$ 
proof cases
  assume * :  $\langle (s, X) \in \mathcal{F} P \rangle \langle set\ s \cap ev\ 'A = \{\} \rangle$ 
  have  $\langle (s @ [c], \{\}) \notin \mathcal{F} P \rangle$  if  $\langle c \in Y \rangle$  for  $c$ 
  proof (cases  $\langle c \in ev\ 'A \rangle$ )
    from *(2) assms(2)[rule-format, OF that]
    show  $\langle c \in ev\ 'A \implies (s @ [c], \{\}) \notin \mathcal{F} P \rangle$ 
    by auto (metis F-T is-processT1)
  next
    from *(2) assms(2)[rule-format, OF that]
    show  $\langle c \notin ev\ 'A \implies (s @ [c], \{\}) \notin \mathcal{F} P \rangle$  by simp
  qed
  with *(1) is-processT5 have  $\langle (s, X \cup Y) \in \mathcal{F} P \rangle$  by blast
  with *(2) show  $\langle (s, X \cup Y) \in ?f \rangle$  by blast

```

next
assume $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{D}\ P \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \rangle$
hence $\langle s \in ?d \rangle$ **by** *blast*
thus $\langle (s, X \cup Y) \in ?f \rangle$ **by** *simp (metis NF-ND)*
next
assume $\langle \exists t1\ a\ t2. s = t1 @ ev\ a \# t2 \wedge t1 @ [ev\ a] \in \mathcal{T}\ P \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F}\ (Q\ a) \rangle$
then obtain $t1\ a\ t2$
where $*$: $\langle s = t1 @ ev\ a \# t2 \rangle \langle t1 @ [ev\ a] \in \mathcal{T}\ P \rangle$
 $\langle set\ t1 \cap ev\ 'A = \{\} \rangle \langle a \in A \rangle \langle (t2, X) \in \mathcal{F}\ (Q\ a) \rangle$ **by** *blast*
have $\langle (t2 @ [c], \{\}) \notin \mathcal{F}\ (Q\ a) \rangle$ **if** $\langle c \in Y \rangle$ **for** c
using *assms(2)[rule-format, OF that, simplified, THEN conjunct2,*
THEN conjunct2, rule-format, of a t1 <t2 @ [c]>
by (*simp add: *(1, 2, 3, 4)*)
with **(5) is-processT5* **have** $**$: $\langle (t2, X \cup Y) \in \mathcal{F}\ (Q\ a) \rangle$ **by** *blast*
show $\langle (s, X \cup Y) \in ?f \rangle$
using **(1, 2, 3, 4) ** by blast*
qed
next
have $*$: $\langle \bigwedge s\ t1\ a\ t2. s @ [tick] = t1 @ ev\ a \# t2 \implies \exists t2'. t2 = t2' @ [tick] \rangle$
by (*simp add: snoc-eq-iff-butlast split: if-split-asm*)
(metis append-butlast-last-id)
show $\langle (s @ [tick], \{\}) \in ?f \implies (s, X - \{tick\}) \in ?f \rangle$ **for** $s\ X$
apply (*simp, elim disjE exE conjE*)
subgoal by (*solves <simp add: is-processT6>*)
subgoal by (*metis butlast-append butlast-snoc front-tickFree-butlast*
non-tickFree-tick tickFree-Nil tickFree-append
tickFree-implies-front-tickFree)
by (*frule *, elim exE, simp, metis is-processT6*)
next
show $\langle \llbracket s \in ?d; tickFree\ s; front-tickFree\ t \rrbracket \implies s @ t \in ?d \rangle$ **for** $s\ t$
by *auto (metis front-tickFree-append, metis is-processT7)*
next
show $\langle s \in ?d \implies (s, X) \in ?f \rangle$ **for** $s\ X$
by *simp (metis NF-ND)*
next
show $\langle s @ [tick] \in ?d \implies s \in ?d \rangle$ **for** s
apply (*simp, elim disjE*)
by (*metis butlast-append butlast-snoc front-tickFree-butlast non-tickFree-tick*
tickFree-Nil tickFree-append tickFree-implies-front-tickFree)
(metis D-T T-nonTickFree-imp-decomp append-single-T-imp-tickFree
butlast.simps(2) butlast-append butlast-snoc event.distinct(1)
process-charn tickFree-Cons tickFree-append)
qed
qed

We add some syntactic sugar.

syntax $\text{-Throw} :: \langle [\alpha \text{ process}, \text{pttrn}, \alpha \text{ set}, \alpha \Rightarrow \alpha \text{ process}] \Rightarrow \alpha \text{ process} \rangle$
 $\langle \langle (-) \Theta (-) \cdot (-) \rangle [73, 0, 0, 73] 72 \rangle$

translations $P \Theta a \in A. Q \Rightarrow \text{CONST Throw } P A (\lambda a. Q)$

abbreviation $\text{Throw-without-free-var} ::$

$\langle [\alpha \text{ process}, \alpha \text{ set}, \alpha \text{ process}] \Rightarrow \alpha \text{ process} \rangle \langle \langle (-) \Theta (-) \cdot (-) \rangle [73, 0, 73] 72 \rangle$
where $\langle P \Theta A Q \equiv P \Theta a \in A. Q \rangle$

Now we can write $P \Theta a \in A. Q a$, and when we do not want Q to be parameterized we can just write $P \Theta A Q$.

lemma $\langle P \Theta a \in A. Q = P \Theta A Q \rangle$ **by** (*fact refl*)

2.2.2 Projections

lemma $F\text{-Throw}$:

$\langle \mathcal{F} (P \Theta a \in A. Q a) =$
 $\{(t1, X) \in \mathcal{F} P. \text{set } t1 \cap \text{ev } 'A = \{\}\} \cup$
 $\{(t1 @ t2, X) \mid t1 \ t2 \ X.$
 $t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{(t1 @ \text{ev } a \ \# \ t2, X) \mid t1 \ a \ t2 \ X.$
 $t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a)\} \rangle$
by (*simp add: Failures-def FAILURES-def Throw.rep-eq*)

lemma $D\text{-Throw}$:

$\langle \mathcal{D} (P \Theta a \in A. Q a) =$
 $\{t1 @ t2 \mid t1 \ t2.$
 $t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{t1 @ \text{ev } a \ \# \ t2 \mid t1 \ a \ t2.$
 $t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D} (Q a)\} \rangle$
by (*simp add: Divergences-def DIVERGENCES-def Throw.rep-eq*)

lemma $T\text{-Throw}$:

$\langle \mathcal{T} (P \Theta a \in A. Q a) =$
 $\{t1 \in \mathcal{T} P. \text{set } t1 \cap \text{ev } 'A = \{\}\} \cup$
 $\{t1 @ t2 \mid t1 \ t2.$
 $t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{t1 @ \text{ev } a \ \# \ t2 \mid t1 \ a \ t2.$
 $t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{T} (Q a)\} \rangle$
by (*auto simp add: Traces.rep-eq TRACES-def Failures.rep-eq[symmetric] F-Throw*)
blast+

2.2.3 Monotony

lemma $\text{min-elems-Un-subset}$:

$\langle \text{min-elems } (A \cup B) \subseteq \text{min-elems } A \cup (\text{min-elems } B - A) \rangle$

by (auto simp add: min-elems-def subset-iff)

lemma *mono-Throw*[simp] : $\langle P \Theta a \in A. Q a \sqsubseteq P' \Theta a \in A. Q' a \rangle$
if $\langle P \sqsubseteq P' \rangle$ **and** $\langle \forall a \in A. Q a \sqsubseteq Q' a \rangle$
proof (unfold le-approx-def Ra-def, safe)
from le-approx1[OF that(1)] le-approx-lemma-T[OF that(1)]
le-approx1[OF that(2)][rule-format]
show $\langle s \in \mathcal{D} (P' \Theta a \in A. Q' a) \implies s \in \mathcal{D} (P \Theta a \in A. Q a) \rangle$ **for** s
by (simp add: D-Throw subset-iff) metis
next
fix $s X$
assume *assms* : $\langle s \notin \mathcal{D} (P \Theta a \in A. Q a) \rangle \langle (s, X) \in \mathcal{F} (P \Theta a \in A. Q a) \rangle$
from *assms*(2) **consider** $\langle (s, X) \in \mathcal{F} P \rangle \langle \text{set } s \cap \text{ev } 'A = \{\} \rangle$
| $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge$
set $t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2 \rangle$
| $\langle \exists t1 a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} P \wedge$
set $t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a) \rangle$
by (simp add: F-Throw) blast
thus $\langle (s, X) \in \mathcal{F} (P' \Theta a \in A. Q' a) \rangle$
proof *cases*
assume * : $\langle (s, X) \in \mathcal{F} P \rangle \langle \text{set } s \cap \text{ev } 'A = \{\} \rangle$
from *assms*(1)[simplified D-Throw, simplified, THEN conjunct1, rule-format,
of s]
assms(1)[simplified D-Throw, simplified, THEN conjunct1, rule-format, of
 $\langle \text{butlast } s \rangle$]
have ** : $\langle s \notin \mathcal{D} P \rangle$
using *(2) **apply** (cases $\langle \text{tickFree } s \rangle$, auto)
by (metis append-butlast-last-id disjoint-iff front-tickFree-butlast
front-tickFree-single in-set-butlastD process-charn tickFree-butlast)
show $\langle (s, X) \in \mathcal{F} P \implies \text{set } s \cap \text{ev } 'A = \{\} \implies (s, X) \in \mathcal{F} (\text{Throw } P' A Q') \rangle$
by (simp add: F-Throw le-approx2[OF that(1) **])
next
assume $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge$
set $t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2 \rangle$
with *assms*(1) **show** $\langle (s, X) \in \mathcal{F} (\text{Throw } P' A Q') \rangle$
by (simp add: F-Throw D-Throw)
next
assume $\langle \exists t1 a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} P \wedge$
set $t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a) \rangle$
then obtain $t1 a t2$
where * : $\langle s = t1 @ \text{ev } a \# t2 \rangle \langle t1 @ [\text{ev } a] \in \mathcal{T} P \rangle$
 $\langle \text{set } t1 \cap \text{ev } 'A = \{\} \rangle \langle a \in A \rangle \langle (t2, X) \in \mathcal{F} (Q a) \rangle$ **by** blast
from *(2) append-single-T-imp-tickFree **have** ** : $\langle \text{tickFree } t1 \rangle$ **by** blast
have *** : $\langle (t2, X) \in \mathcal{F} (Q' a) \rangle$
by (fact *assms*(1)[simplified D-Throw, simplified, THEN conjunct2, rule-format,
OF *(4, 3, 2, 1), THEN le-approx2[OF that(2)][rule-format,
OF *(4)]]],
of X , simplified *(5), simplified]
have **** : $\langle t1 \notin \mathcal{D} P \rangle$

```

apply (rule notI)
  apply (drule assms(1)[simplified D-Throw, simplified, THEN conjunct1,
rule-format,
      OF *(3) **, of ⟨ev a # t2⟩, simplified *(1), simplified])
  by (metis *(1) assms(2) front-tickFree-mono process-chaen)
show ⟨(s, X) ∈  $\mathcal{F}$  (Throw P' A Q')⟩
  apply (simp add: F-Throw D-Throw *(1))
  by (metis *(2, 3, 4) *** **** T-F-spec le-approx2 min-elems6 that(1))
qed
next
from le-approx1[OF that(1)] le-approx2[OF that(1)] le-approx2T[OF that(1)]
  le-approx2[OF that(2)[rule-format]]
show ⟨s ∉  $\mathcal{D}$  (P  $\Theta$  a ∈ A. Q a) ⟹
  (s, X) ∈  $\mathcal{F}$  (P'  $\Theta$  a ∈ A. Q' a) ⟹ (s, X) ∈  $\mathcal{F}$  (P  $\Theta$  a ∈ A. Q a)⟩ for s X
  apply (simp add: F-Throw D-Throw subset-eq, safe, simp-all)
  by (meson NF-ND) (metis D-T)+
next
define S-left
  where ⟨S-left ≡ {t1 @ t2 | t1 t2. t1 ∈  $\mathcal{D}$  P ∧ tickFree t1 ∧
  set t1 ∩ ev ' A = {} ∧ front-tickFree t2}⟩
define S-right
  where ⟨S-right ≡ {t1 @ ev a # t2 | t1 a t2. t1 @ [ev a] ∈  $\mathcal{T}$  P ∧
  set t1 ∩ ev ' A = {} ∧ a ∈ A ∧ t2 ∈  $\mathcal{D}$  (Q a)}⟩

have * : ⟨min-elems ( $\mathcal{D}$  (P  $\Theta$  a ∈ A. Q a)) ⊆ min-elems S-left ∪ (min-elems
S-right - S-left)⟩
  unfolding S-left-def S-right-def
  by (simp add: D-Throw min-elems-Un-subset)
have ** : ⟨min-elems S-left = {t1 ∈ min-elems ( $\mathcal{D}$  P). set t1 ∩ ev ' A = {} }⟩
  unfolding S-left-def min-elems-def le-list-def less-list-def
  apply (simp, safe)
  subgoal by (solves ⟨meson is-processT7-S⟩)
  subgoal by (metis Int-Un-distrib2 Un-empty append.right-neutral front-tickFree-Nil
front-tickFree-append front-tickFree-mono set-append)
  subgoal by (metis IntI append-Nil2 front-tickFree-Nil imageI)
  subgoal by (metis list.distinct(1) nonTickFree-n-frontTickFree process-chaen
self-append-conv)
  by (metis Nil-is-append-conv append-eq-appendI self-append-conv)

{ fix t1 a t2
  assume assms : ⟨t1 @ [ev a] ∈  $\mathcal{T}$  P⟩ ⟨set t1 ∩ ev ' A = {}⟩ ⟨a ∈ A⟩
  ⟨t2 ∈ ( $\mathcal{D}$  (Q a))⟩ ⟨t1 @ ev a # t2 ∈ min-elems S-right⟩ ⟨t1 @ ev a
# t2 ∉ S-left⟩
  have ⟨t2 ∈ min-elems ( $\mathcal{D}$  (Q a))⟩
  ⟨t1 @ [ev a] ∈  $\mathcal{D}$  P ⟹ t1 @ [ev a] ∈ min-elems ( $\mathcal{D}$  P)⟩
  proof (all ⟨rule ccontr⟩)
  assume ⟨t2 ∉ min-elems ( $\mathcal{D}$  (Q a))⟩
  with assms(4) obtain t2' where ⟨t2' < t2⟩ ⟨t2' ∈  $\mathcal{D}$  (Q a)⟩
  unfolding min-elems-def by blast

```

hence $\langle t1 @ ev a \# t2' \in S\text{-right} \rangle \langle t1 @ ev a \# t2' < t1 @ ev a \# t2 \rangle$
unfolding $S\text{-right-def}$ **using** $assms(1, 2, 3)$
by $(auto simp add: less-append less-cons)$
with $assms(5)$ $min\text{-elems-no nless-le}$ **show** $False$ **by** $blast$
next
assume $\langle t1 @ [ev a] \in \mathcal{D} P \rangle \langle t1 @ [ev a] \notin min\text{-elems} (\mathcal{D} P) \rangle$
hence $\langle t1 \in \mathcal{D} P \rangle$ **using** $min\text{-elems1}$ **by** $blast$
with $\langle t1 @ [ev a] \in \mathcal{D} P \rangle$ **have** $\langle t1 @ ev a \# t2 \in S\text{-left} \rangle$
by $(simp add: S\text{-left-def})$
 $(metis D\text{-imp-front-tickFree append-Cons append-Nil assms(2, 4)$
 $event.simps(3) front\text{-tickFree-append tickFree-Nil}$
 $front\text{-tickFree-implies-tickFree tickFree-Cons})$
with $assms(6)$ **show** $False$ **by** $simp$
qed
} note $*** = this$
have $**** : \langle min\text{-elems} S\text{-right} - S\text{-left} \subseteq$
 $\{t1 @ ev a \# t2 \mid t1 a t2. t1 @ [ev a] \in \mathcal{T} P - \mathcal{D} P \wedge$
 $set t1 \cap ev 'A = \{\} \wedge a \in A \wedge t2 \in min\text{-elems} (\mathcal{D} (Q a))\} \cup$
 $\{t1 @ ev a \# t2 \mid t1 a t2. t1 @ [ev a] \in min\text{-elems} (\mathcal{D} P) \wedge$
 $set t1 \cap ev 'A = \{\} \wedge a \in A \wedge t2 \in min\text{-elems} (\mathcal{D} (Q a))\}$
apply $(intro subsetI, simp, elim conjE)$
apply $(frule set\text{-mp}[OF min\text{-elems-le-self], subst (asm) (2) S\text{-right-def})$
using $***$ **by** $fastforce$

fix s
assume $assm: \langle s \in min\text{-elems} (\mathcal{D} (P \Theta a \in A. Q a)) \rangle$
from $set\text{-mp}[OF *, OF this]$
consider $\langle s \in min\text{-elems} (\mathcal{D} P) \rangle \langle set s \cap ev 'A = \{\} \rangle$
 $\mid \langle \exists t1 a t2.$
 $s = t1 @ ev a \# t2 \wedge set t1 \cap ev 'A = \{\} \wedge a \in A \wedge t2 \in min\text{-elems} (\mathcal{D}$
 $(Q a)) \wedge$
 $(t1 @ [ev a] \in min\text{-elems} (\mathcal{D} P) \vee t1 @ [ev a] \in \mathcal{T} P \wedge t1 @ [ev a] \notin \mathcal{D} P) \rangle$
using $****$ **by** $(simp add: **) blast$
thus $\langle s \in \mathcal{T} (P' \Theta a \in A. Q' a) \rangle$
proof cases
show $\langle s \in min\text{-elems} (\mathcal{D} P) \implies set s \cap ev 'A = \{\} \implies s \in \mathcal{T} (Throw P' A$
 $Q') \rangle$
by $(drule set\text{-mp}[OF le\text{-approx3}[OF that(1)]], simp add: T\text{-Throw})$
next
assume $\langle \exists t1 a t2.$
 $s = t1 @ ev a \# t2 \wedge set t1 \cap ev 'A = \{\} \wedge a \in A \wedge t2 \in min\text{-elems}$
 $(\mathcal{D} (Q a)) \wedge$
 $(t1 @ [ev a] \in min\text{-elems} (\mathcal{D} P) \vee t1 @ [ev a] \in \mathcal{T} P \wedge t1 @ [ev a] \notin \mathcal{D}$
 $P) \rangle$
then obtain $t1 a t2$
where $***** : \langle s = t1 @ ev a \# t2 \rangle \langle set t1 \cap ev 'A = \{\} \rangle$
 $\langle a \in A \rangle \langle t2 \in min\text{-elems} (\mathcal{D} (Q a)) \rangle$
 $\langle t1 @ [ev a] \in min\text{-elems} (\mathcal{D} P) \vee$
 $t1 @ [ev a] \in \mathcal{T} P \wedge t1 @ [ev a] \notin \mathcal{D} P \rangle$ **by** $blast$

have $\langle t1 \ @ \ [ev \ a] \in \mathcal{T} \ P' \wedge t2 \in \mathcal{T} \ (Q' \ a) \rangle$
by (*meson ******(3, 4, 5) *le-approx2T le-approx3 subsetD that*)
with ***** **show** $\langle s \in \mathcal{T} \ (Throw \ P' \ A \ Q') \rangle$
by (*simp add: T-Throw*) *blast*
qed
qed

lemma *mono-right-Throw-F* :
 $\langle \forall a \in A. Q \ a \sqsubseteq_F Q' \ a \implies P \ \Theta \ a \in A. Q \ a \sqsubseteq_F P \ \Theta \ a \in A. Q' \ a \rangle$
unfolding *failure-refine-def*
by (*simp add: F-Throw subset-iff disjoint-iff*) *blast*

lemma *mono-right-Throw-T* :
 $\langle \forall a \in A. Q \ a \sqsubseteq_T Q' \ a \implies P \ \Theta \ a \in A. Q \ a \sqsubseteq_T P \ \Theta \ a \in A. Q' \ a \rangle$
unfolding *trace-refine-def*
by (*simp add: T-Throw subset-iff disjoint-iff*) *blast*

lemma *mono-right-Throw-D*:
 $\langle \forall a \in A. Q \ a \sqsubseteq_D Q' \ a \implies P \ \Theta \ a \in A. Q \ a \sqsubseteq_D P \ \Theta \ a \in A. Q' \ a \rangle$
unfolding *divergence-refine-def*
by (*simp add: D-Throw subset-iff disjoint-iff*) *blast*

lemma *mono-Throw-FD* : $\langle P \sqsubseteq_{FD} P' \implies \forall a \in A. Q \ a \sqsubseteq_{FD} Q' \ a \implies$
 $P \ \Theta \ a \in A. Q \ a \sqsubseteq_{FD} P' \ \Theta \ a \in A. Q' \ a \rangle$
apply (*rule trans-FD[of - $\langle P' \ \Theta \ a \in A. Q \ a \rangle$]*)
subgoal by (*simp add: failure-divergence-refine-def*
le-ref-def F-Throw D-Throw subset-iff; safe;
metis [[metis-verbose = false]] T-F no-Trace-implies-no-Failure)
by (*meson leFD-imp-leD leFD-imp-leF leF-leD-imp-leFD*
mono-right-Throw-D mono-right-Throw-F)

lemma *mono-Throw-DT* : $\langle P \sqsubseteq_{DT} P' \implies \forall a \in A. Q \ a \sqsubseteq_{DT} Q' \ a \implies$
 $P \ \Theta \ a \in A. Q \ a \sqsubseteq_{DT} P' \ \Theta \ a \in A. Q' \ a \rangle$
apply (*rule trans-DT[of - $\langle P' \ \Theta \ a \in A. Q \ a \rangle$]*)
subgoal by (*simp add: trace-divergence-refine-def trace-refine-def*
divergence-refine-def D-Throw T-Throw subset-iff, blast)
by (*simp add: mono-right-Throw-D mono-right-Throw-T trace-divergence-refine-def*)

2.2.4 Properties

lemma *Throw-STOP*: $\langle STOP \ \Theta \ a \in A. Q \ a = STOP \rangle$
by (*auto simp add: STOP-iff-T T-Throw T-STOP D-STOP*)

lemma *Throw-SKIP*: $\langle SKIP \ \Theta \ a \in A. Q \ a = SKIP \rangle$
by (*auto simp add: Process-eq-spec F-Throw F-SKIP D-Throw D-SKIP T-SKIP*)

lemma *Throw-BOT*: $\langle \perp \ \Theta \ a \in A. Q \ a = \perp \rangle$
by (*simp add: BOT-iff-D D-Throw D-UU*)

lemma *Throw-is-BOT-iff*: $\langle P \Theta a \in A. Q a = \perp \longleftrightarrow P = \perp \rangle$
by (*simp add: BOT-iff-D D-Throw*)

lemma *Throw-empty-set*: $\langle P \Theta a \in \{\}, Q a = P \rangle$
by (*auto simp add: Process-eq-spec F-Throw D-Throw*
D-expand[symmetric] is-processT7-S is-processT8-S)

lemma *Throw-Ndet*:
 $\langle P \sqcap P' \Theta a \in A. Q a = (P \Theta a \in A. Q a) \sqcap (P' \Theta a \in A. Q a) \rangle$
 $\langle P \Theta a \in A. Q a \sqcap Q' a = (P \Theta a \in A. Q a) \sqcap (P' \Theta a \in A. Q' a) \rangle$
by (*simp add: Process-eq-spec F-Throw F-Ndet D-Throw D-Ndet T-Ndet,*
safe, simp-all; blast) $\+$

lemma *Throw-Det*:
 $\langle P \sqcap P' \Theta a \in A. Q a = (P \Theta a \in A. Q a) \sqcap (P' \Theta a \in A. Q a) \rangle$
 $\langle P \Theta a \in A. Q a \sqcap Q' a = (P \Theta a \in A. Q a) \sqcap (P' \Theta a \in A. Q' a) \rangle$
proof –
show $\langle \text{Throw } (P \sqcap P') A Q = \text{Throw } P A Q \sqcap \text{Throw } P' A Q \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
proof (*subst Process-eq-spec-optimized, safe*)
show $\langle s \in \mathcal{D} ?lhs \implies s \in \mathcal{D} ?rhs \rangle$ **for** s
by (*auto simp add: D-Det T-Det D-Throw*)
next
show $\langle s \in \mathcal{D} ?rhs \implies s \in \mathcal{D} ?lhs \rangle$ **for** s
by (*simp add: D-Det T-Det D-Throw*) *blast*
next
fix $s X$
assume *same-div*: $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
hence $\langle (s, X) \in \mathcal{F} (P \sqcap P') \wedge \text{set } s \cap \text{ev } 'A = \{\} \vee s \in \mathcal{D} ?lhs \vee$
 $(\exists t1 a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} (P \sqcap P') \wedge$
 $\text{set } t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a)) \rangle$
by (*simp add: F-Throw D-Throw*) *blast*
thus $\langle (s, X) \in \mathcal{F} ?rhs \rangle$
apply (*elim disjE exE*)
subgoal by (*cases s, auto simp add: F-Det F-Throw T-Throw D-Throw*) $[1]$
subgoal by (*use D-F same-div in blast*)
by (*auto simp add: F-Det T-Det F-Throw T-Throw D-Throw*)
next
show $\langle (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} ?lhs \rangle$ **for** $s X$
apply (*simp add: F-Det, elim disjE conjE*)
by (*auto simp add: F-Det D-Det T-Det F-Throw D-Throw*
T-Throw D-T is-processT7-S subset-iff)
(simp-all add: Cons-eq-append-conv)
qed
next

show $\langle P \Theta a \in A. Q a \sqcap Q' a = \text{Throw } P A Q \sqcap \text{Throw } P A Q' \rangle$ (is $\langle ?lhs Q Q' = ?rhs \rangle$)
proof (subst Process-eq-spec-optimized, safe)
show $\langle s \in \mathcal{D} (?lhs Q Q') \implies s \in \mathcal{D} ?rhs \rangle$ **for** s
by (auto simp add: D-Det D-Throw D-Ndet)
next
show $\langle s \in \mathcal{D} ?rhs \implies s \in \mathcal{D} (?lhs Q Q') \rangle$ **for** s
by (simp add: D-Det D-Throw D-Ndet) blast
next
fix $s X$
assume same-div: $\langle \mathcal{D} (?lhs Q Q') = \mathcal{D} ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} (?lhs Q Q') \rangle$
hence $\langle (s, X) \in \mathcal{F} P \wedge \text{set } s \cap \text{ev } A = \{\} \vee s \in \mathcal{D} (?lhs Q Q') \vee$
 $\langle \exists t1 a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} P \wedge$
 $\text{set } t1 \cap \text{ev } A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a \sqcap Q' a) \rangle$
by (simp add: F-Throw D-Throw) blast
thus $\langle (s, X) \in \mathcal{F} ?rhs \rangle$
apply (elim disjE exE)
subgoal by (solves $\langle \text{simp add: F-Det F-Throw} \rangle$)
subgoal by (use D-F same-div in blast)
by (auto simp add: F-Det F-Ndet F-Throw)
next
show $\langle (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} (?lhs Q Q') \rangle$ **for** $s X$
proof (cases s)
show $\langle s = [] \implies (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} (?lhs Q Q') \rangle$
by (auto simp add: F-Det F-Throw D-Throw T-Throw
is-processT6-S2 Cons-eq-append-conv)
next
fix $a s'$
have $*$: $\langle (a \# s', X) \in \mathcal{F} (\text{Throw } P A Q) \implies$
 $(a \# s', X) \in \mathcal{F} (?lhs Q Q') \rangle$ **for** $Q Q'$
by (auto simp add: F-Throw F-Det F-Ndet)
show $\langle s = a \# s' \implies (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} (?lhs Q Q') \rangle$
apply (simp add: F-Det, elim disjE)
subgoal by (erule *)
by (subst Ndet-commute) (erule *)
qed
qed
qed

lemma *Throw-GlobalNdet*:

$\langle (\sqcap a \in A. P a) \Theta b \in B. Q b = \sqcap a \in A. (P a \Theta b \in B. Q b) \rangle$
 $\langle P' \Theta a \in A. (\sqcap b \in B. Q' a b) =$
 $(\text{if } B = \{\} \text{ then } P' \Theta A \text{ STOP else } \sqcap b \in B. (P' \Theta a \in A. Q' a b)) \rangle$
by (simp add: Process-eq-spec F-Throw D-Throw
F-GlobalNdet D-GlobalNdet T-GlobalNdet, safe, simp-all; blast)
(simp add: Process-eq-spec F-Throw D-Throw
F-GlobalNdet D-GlobalNdet T-GlobalNdet F-STOP D-STOP; blast)

lemma *Throw-disjoint-events*: $\langle A \cap \text{events-of } P = \{\} \implies P \Theta a \in A. Q a = P \rangle$
proof (*subst Process-eq-spec-optimized, safe*)
show $\langle A \cap \text{events-of } P = \{\} \implies s \in \mathcal{D} (Throw P A Q) \implies s \in \mathcal{D} P \rangle$ **for** s
by (*simp add: D-Throw disjoint-iff events-of-def*)
(*meson in-set-conv-decomp is-processT7-S*)
next
show $\langle A \cap \text{events-of } P = \{\} \implies s \in \mathcal{D} P \implies s \in \mathcal{D} (Throw P A Q) \rangle$ **for** s
by (*simp add: D-Throw disjoint-iff events-of-def image-iff*)
(*metis (no-types, lifting) D-T append-Nil2 butlast-snoc front-tickFree-mono front-tickFree-butlast process-chnon nonTickFree-n-frontTickFree*)
next
show $\langle A \cap \text{events-of } P = \{\} \implies (s, X) \in \mathcal{F} (Throw P A Q) \implies (s, X) \in \mathcal{F} P \rangle$ **for** $s X$
by (*simp add: F-Throw disjoint-iff events-of-def*)
(*meson in-set-conv-decomp process-chnon*)
next
show $\langle A \cap \text{events-of } P = \{\} \implies (s, X) \in \mathcal{F} P \implies (s, X) \in \mathcal{F} (Throw P A Q) \rangle$ **for** $s X$
by (*simp add: F-Throw disjoint-iff events-of-def image-iff*) (*metis F-T*)
qed

lemma *events-Throw*:
 $\langle \text{events-of } (P \Theta a \in A. Q a) \subseteq \text{events-of } P \cup (\bigcup a \in (A \cap \text{events-of } P). \text{events-of } (Q a)) \rangle$
proof (*intro subsetI*)
fix e
assume $\langle e \in \text{events-of } (P \Theta a \in A. Q a) \rangle$
then obtain s **where** $*$: $\langle \text{ev } e \in \text{set } s \rangle \langle s \in \mathcal{T} (P \Theta a \in A. Q a) \rangle$
by (*simp add: events-of-def*) *blast*
from $*(2)$ **consider** $\langle s \in \mathcal{T} P \rangle \langle \text{set } s \cap \text{ev } 'A = \{\} \rangle$
 $| \langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2 \rangle$
 $| \langle \exists t1 a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{T} (Q a) \rangle$
by (*simp add: T-Throw*) *blast*
thus $\langle e \in \text{events-of } P \cup (\bigcup a \in (A \cap \text{events-of } P). \text{events-of } (Q a)) \rangle$
proof cases
from $*(1)$ **show** $\langle s \in \mathcal{T} P \implies \text{set } s \cap \text{ev } 'A = \{\} \implies e \in \text{events-of } P \cup (\bigcup a \in A \cap \text{events-of } P. \text{events-of } (Q a)) \rangle$
by (*simp add: events-of-def*) *blast*
next
show $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } 'A = \{\} \wedge \text{front-tickFree } t2 \implies e \in \text{events-of } P \cup (\bigcup a \in (A \cap \text{events-of } P). \text{events-of } (Q a)) \rangle$
by (*metis UNIV-I UnI1 empty-iff events-div*)
next
assume $\langle \exists t1 a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} P \wedge$

$set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{T}\ (Q\ a)\rangle$
then obtain $t1\ a\ t2$
where $** : \langle s = t1 \ @\ ev\ a \ \# \ t2 \rangle \langle t1 \ @\ [ev\ a] \in \mathcal{T}\ P \rangle$
 $\langle set\ t1 \cap ev\ 'A = \{\} \rangle \langle a \in A \rangle \langle t2 \in \mathcal{T}\ (Q\ a) \rangle$ **by** *blast*
from $*(1)\ ** (1)$ **have** $\langle ev\ e \in set\ (t1 \ @\ [ev\ a]) \vee ev\ e \in set\ t2 \rangle$ **by** *simp*
thus $\langle e \in events-of\ P \cup (\bigcup a \in (A \cap events-of\ P).\ events-of\ (Q\ a)) \rangle$
proof (*elim disjE*)
show $\langle ev\ e \in set\ (t1 \ @\ [ev\ a]) \implies$
 $e \in events-of\ P \cup (\bigcup a \in A \cap events-of\ P.\ events-of\ (Q\ a)) \rangle$
unfolding *events-of-def* **using** $**(2)$ **by** *blast*
next
show $\langle ev\ e \in set\ t2 \implies e \in events-of\ P \cup (\bigcup a \in A \cap events-of\ P.\ events-of\ (Q\ a)) \rangle$
unfolding *events-of-def* **using** $**(2, 4, 5)$ *mem-Collect-eq* **by** *fastforce*
qed
qed
qed

2.2.5 Key Property

lemma *Throw-Mprefix*:

$\langle (\Box a \in A \rightarrow P\ a) \Theta\ b \in B.\ Q\ b =$
 $\Box a \in A \rightarrow (if\ a \in B\ then\ Q\ a\ else\ P\ a \Theta\ b \in B.\ Q\ b) \rangle$
(is $\langle ?lhs = ?rhs \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

fix s

assume $\langle s \in \mathcal{D}\ ?lhs \rangle$

then consider $\langle \exists t1\ t2.\ s = t1 \ @\ t2 \wedge t1 \in \mathcal{D}\ (Mprefix\ A\ P) \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge front-tickFree\ t2 \rangle$

$| \langle \exists t1\ b\ t2.\ s = t1 \ @\ ev\ b \ \# \ t2 \wedge t1 \ @\ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D}\ (Q\ b) \rangle$

by (*simp add: D-Throw*) *blast*

thus $\langle s \in \mathcal{D}\ ?rhs \rangle$

proof *cases*

assume $\langle \exists t1\ t2.\ s = t1 \ @\ t2 \wedge t1 \in \mathcal{D}\ (Mprefix\ A\ P) \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge front-tickFree\ t2 \rangle$

then obtain $t1\ t2$

where $* : \langle s = t1 \ @\ t2 \rangle \langle t1 \in \mathcal{D}\ (Mprefix\ A\ P) \rangle \langle tickFree\ t1 \rangle$
 $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle front-tickFree\ t2 \rangle$ **by** *blast*

from $*(2)$ **obtain** $a\ t1'$ **where** $** : \langle t1 = ev\ a \ \# \ t1' \rangle \langle a \in A \rangle \langle t1' \in \mathcal{D}\ (P\ a) \rangle$

by (*simp add: D-Mprefix*) (*metis event.inject image-iff list.collapse*)

from $*(4)\ ** (1)$ **have** $*** : \langle a \notin B \rangle$ **by** (*simp add: image-iff*)

have $\langle t1' \ @\ t2 \in \mathcal{D}\ (Throw\ (P\ a)\ B\ Q) \rangle$

using $*(3, 4, 5)\ ** (1, 3)$ **by** (*auto simp add: D-Throw*)

with $***$ **show** $\langle s \in \mathcal{D}\ ?rhs \rangle$

by (*simp add: D-Mprefix*) $*(1)\ ** (1, 2)$

next

assume $\langle \exists t1\ b\ t2.\ s = t1 \ @\ ev\ b \ \# \ t2 \wedge t1 \ @\ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D}\ (Q\ b) \rangle$

```

then obtain  $t1\ b\ t2$ 
  where  $*$  :  $\langle s = t1 @ ev\ b \# t2 \rangle \langle t1 @ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \rangle$ 
            $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle b \in B \rangle \langle t2 \in \mathcal{D}\ (Q\ b) \rangle$  by blast
show  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
proof (cases  $\langle t1 \rangle$ )
  from  $*(2)$  show  $\langle t1 = [] \implies s \in \mathcal{D}\ ?rhs \rangle$ 
    by (simp add: D-Mprefix T-Mprefix  $*(1, 4, 5)$ )
next
  fix  $a\ t1'$ 
  assume  $\langle t1 = a \# t1' \rangle$ 
  then obtain  $a'$  where  $\langle t1 = ev\ a' \# t1' \rangle$ 
    by (metis  $*(2)$  append-single-T-imp-tickFree event.exhaust tickFree-Cons)
  with  $*(2, 3, 4, 5)$  show  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
    by (auto simp add:  $*(1)$  D-Mprefix T-Mprefix D-Throw)
qed
qed
next
  fix  $s$ 
  assume  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
  then obtain  $a\ s'$  where  $*$  :  $\langle a \in A \rangle \langle s = ev\ a \# s' \rangle$ 
            $\langle s' \in \mathcal{D}\ (if\ a \in B\ then\ Q\ a\ else\ Throw\ (P\ a)\ B\ Q) \rangle$ 
  by (simp add: D-Mprefix) (metis event.inject image-iff list.collapse)
show  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 
proof (cases  $\langle a \in B \rangle$ )
  assume  $\langle a \in B \rangle$ 
  hence  $**$  :  $\langle [] @ [ev\ a] \in \mathcal{T}\ (Mprefix\ A\ P) \wedge set\ [] \cap ev\ 'B = \{\} \wedge s' \in \mathcal{D}\ (Q\ a) \rangle$ 
  using  $*(3)$  by (simp add: T-Mprefix Nil-elem-T  $*(1)$ )
  show  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 
    by (simp add: D-Throw) (metis  $*(2)$   $** \langle a \in B \rangle$  append-Nil)
next
  assume  $\langle a \notin B \rangle$ 
  with  $*(2, 3)$ 
  consider  $\langle \exists t1\ t2. s = ev\ a \# t1 @ t2 \wedge t1 \in \mathcal{D}\ (P\ a) \wedge tickFree\ t1 \wedge$ 
            $set\ t1 \cap ev\ 'B = \{\} \wedge front-tickFree\ t2 \rangle$ 
  |  $\langle \exists t1\ b\ t2. s = ev\ a \# t1 @ ev\ b \# t2 \wedge t1 @ [ev\ b] \in \mathcal{T}\ (P\ a) \wedge$ 
            $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D}\ (Q\ b) \rangle$ 
  by (simp add: D-Throw) blast
  thus  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 
proof cases
  assume  $\langle \exists t1\ t2. s = ev\ a \# t1 @ t2 \wedge t1 \in \mathcal{D}\ (P\ a) \wedge tickFree\ t1 \wedge$ 
            $set\ t1 \cap ev\ 'B = \{\} \wedge front-tickFree\ t2 \rangle$ 
  then obtain  $t1\ t2$ 
    where  $**$  :  $\langle s = ev\ a \# t1 @ t2 \rangle \langle t1 \in \mathcal{D}\ (P\ a) \rangle \langle tickFree\ t1 \rangle$ 
            $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle front-tickFree\ t2 \rangle$  by blast
  have  $***$  :  $\langle ev\ a \# t1 \in \mathcal{D}\ (Mprefix\ A\ P) \wedge tickFree\ (ev\ a \# t1) \wedge$ 
            $set\ (ev\ a \# t1) \cap ev\ 'B = \{\} \rangle$ 
  by (simp add: D-Mprefix image-iff  $*(1)$   $** (2, 3, 4)$   $\langle a \notin B \rangle$ )
  show  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 

```

by (*simp add: D-Throw*) (*metis ***(1, 5) **** append-Cons*)
 next
 assume $\langle \exists t1\ b\ t2. s = ev\ a\ \# t1\ @\ ev\ b\ \# t2 \wedge t1\ @\ [ev\ b] \in \mathcal{T}\ (P\ a) \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D}\ (Q\ b) \rangle$
 then obtain $t1\ b\ t2$
 where $** : \langle s = ev\ a\ \# t1\ @\ ev\ b\ \# t2 \rangle \langle t1\ @\ [ev\ b] \in \mathcal{T}\ (P\ a) \rangle$
 $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle b \in B \rangle \langle t2 \in \mathcal{D}\ (Q\ b) \rangle$ by *blast*
 have $*** : \langle (ev\ a\ \# t1)\ @\ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \wedge set\ (ev\ a\ \# t1) \cap ev$
 $'B = \{\} \rangle$
 by (*simp add: T-Mprefix image-iff* *(1) ****(2, 3) $\langle a \notin B \rangle$)
 show $\langle s \in \mathcal{D}\ ?lhs \rangle$
 by (*simp add: D-Throw*) (*metis ***(1, 4, 5) **** append-Cons*)
 qed
 qed
 next
 fix $s\ X$
 assume *same-div* : $\langle \mathcal{D}\ ?lhs = \mathcal{D}\ ?rhs \rangle$
 assume $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
 then consider $\langle (s, X) \in \mathcal{F}\ (Mprefix\ A\ P) \rangle \langle set\ s \cap ev\ 'B = \{\} \rangle$
 | $\langle s \in \mathcal{D}\ ?lhs \rangle$
 | $\langle \exists t1\ b\ t2. s = t1\ @\ ev\ b\ \# t2 \wedge t1\ @\ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge (t2, X) \in \mathcal{F}\ (Q\ b) \rangle$
 by (*simp add: F-Throw D-Throw*) *blast*
 thus $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
 proof *cases*
 show $\langle (s, X) \in \mathcal{F}\ (Mprefix\ A\ P) \implies set\ s \cap ev\ 'B = \{\} \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
 by (*simp add: F-Mprefix F-Throw*)
 (*metis disjoint-iff hd-in-set imageI list.set-sel*(2))
 next
 show $\langle s \in \mathcal{D}\ ?lhs \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
 using *same-div D-F* by *blast*
 next
 assume $\langle \exists t1\ b\ t2. s = t1\ @\ ev\ b\ \# t2 \wedge t1\ @\ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge (t2, X) \in \mathcal{F}\ (Q\ b) \rangle$
 then obtain $t1\ b\ t2$
 where $* : \langle s = t1\ @\ ev\ b\ \# t2 \rangle \langle t1\ @\ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \rangle$
 $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle b \in B \rangle \langle (t2, X) \in \mathcal{F}\ (Q\ b) \rangle$ by *blast*
 show $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
 proof (*cases t1*)
 from *(2) show $\langle t1 = [] \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
 by (*auto simp add: F-Mprefix T-Mprefix F-Throw* *(1, 4, 5))
 next
 fix $a\ t1'$
 assume $\langle t1 = a\ \# t1' \rangle$
 then obtain a' where $\langle t1 = ev\ a' \# t1' \rangle$
 by (*metis* *(2) *append-single-T-imp-tickFree event.exhaust tickFree-Cons*)
 with *(2, 3, 5) show $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
 by (*auto simp add: F-Mprefix T-Mprefix F-Throw* *(1, 4))
 qed

```

qed
next
show  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$  for  $s \ X$ 
proof (cases s)
  show  $\langle s = [] \implies (s, X) \in \mathcal{F} \text{ ?rhs} \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Mprefix F-Throw)
next
fix a s'
assume assms :  $\langle s = a \# s' \rangle \langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
from assms(2) obtain a'
  where * :  $\langle a' \in A \rangle \langle s = \text{ev } a' \# s' \rangle$ 
   $\langle (s', X) \in \mathcal{F} \text{ (if } a' \in B \text{ then } Q \ a' \text{ else Throw (P } a') \ B \ Q)} \rangle$ 
  by (simp add: assms(1) F-Mprefix) blast
show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
proof (cases  $\langle a' \in B \rangle$ )
  assume  $\langle a' \in B \rangle$ 
  hence ** :  $\langle [] \ @ \ [\text{ev } a'] \in \mathcal{T} \text{ (Mprefix } A \ P) \ \wedge$ 
     $\text{set } [] \cap \text{ev } ' B = \{\} \ \wedge \ (s', X) \in \mathcal{F} \text{ (Q } a') \rangle$ 
    using *(3) by (simp add: T-Mprefix Nil-elem-T *(1))
  show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Throw) (metis *(2) **  $\langle a' \in B \rangle$  append-Nil)
next
assume  $\langle a' \notin B \rangle$ 
then consider  $\langle (s', X) \in \mathcal{F} \text{ (P } a') \rangle \langle \text{set } s' \cap \text{ev } ' B = \{\} \rangle$ 
  |  $\langle \exists t1 \ t2. s' = t1 \ @ \ t2 \ \wedge \ t1 \in \mathcal{D} \text{ (P } a') \ \wedge \ \text{tickFree } t1 \ \wedge$ 
     $\text{set } t1 \cap \text{ev } ' B = \{\} \ \wedge \ \text{front-tickFree } t2 \rangle$ 
  |  $\langle \exists t1 \ b \ t2. s' = t1 \ @ \ \text{ev } b \ \# \ t2 \ \wedge \ t1 \ @ \ [\text{ev } b] \in \mathcal{T} \text{ (P } a') \ \wedge$ 
     $\text{set } t1 \cap \text{ev } ' B = \{\} \ \wedge \ b \in B \ \wedge \ (t2, X) \in \mathcal{F} \text{ (Q } b) \rangle$ 
  using *(3) by (simp add: F-Throw D-Throw) blast
thus  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
proof cases
  show  $\langle (s', X) \in \mathcal{F} \text{ (P } a') \implies \text{set } s' \cap \text{ev } ' B = \{\} \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Mprefix F-Throw *(1, 2)  $\langle a' \notin B \rangle$  image-iff)
next
assume  $\langle \exists t1 \ t2. s' = t1 \ @ \ t2 \ \wedge \ t1 \in \mathcal{D} \text{ (P } a') \ \wedge \ \text{tickFree } t1 \ \wedge$ 
   $\text{set } t1 \cap \text{ev } ' B = \{\} \ \wedge \ \text{front-tickFree } t2 \rangle$ 
then obtain t1 t2
  where ** :  $\langle s' = t1 \ @ \ t2 \rangle \langle t1 \in \mathcal{D} \text{ (P } a') \rangle \langle \text{tickFree } t1 \rangle$ 
   $\langle \text{set } t1 \cap \text{ev } ' B = \{\} \rangle \langle \text{front-tickFree } t2 \rangle$  by blast
have *** :  $\langle s = (\text{ev } a' \# t1) \ @ \ t2 \ \wedge \ \text{ev } a' \# t1 \in \mathcal{D} \text{ (Mprefix } A \ P) \ \wedge$ 
   $\text{tickFree } (\text{ev } a' \# t1) \ \wedge \ \text{set } (\text{ev } a' \# t1) \cap \text{ev } ' B = \{\} \rangle$ 
  by (simp add: D-Mprefix  $\langle a' \notin B \rangle$  image-iff *(1, 2) ***(1, 2, 3, 4))
show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
  by (simp add: F-Throw F-Mprefix) (metis ***(5) ***)
next
assume  $\langle \exists t1 \ b \ t2. s' = t1 \ @ \ \text{ev } b \ \# \ t2 \ \wedge \ t1 \ @ \ [\text{ev } b] \in \mathcal{T} \text{ (P } a') \ \wedge$ 
   $\text{set } t1 \cap \text{ev } ' B = \{\} \ \wedge \ b \in B \ \wedge \ (t2, X) \in \mathcal{F} \text{ (Q } b) \rangle$ 
then obtain t1 b t2
  where ** :  $\langle s' = t1 \ @ \ \text{ev } b \ \# \ t2 \rangle \langle t1 \ @ \ [\text{ev } b] \in \mathcal{T} \text{ (P } a') \rangle$ 

```

$\langle \text{set } t1 \cap \text{ev } ' B = \{\} \rangle \langle b \in B \rangle \langle (t2, X) \in \mathcal{F} (Q b) \rangle$ **by** *blast*
have ***** : $\langle s = (\text{ev } a' \# t1) @ \text{ev } b \# t2 \wedge \text{set } (\text{ev } a' \# t1) \cap \text{ev } ' B = \{\} \rangle$
 \wedge
 $(\text{ev } a' \# t1) @ [\text{ev } b] \in \mathcal{T} (\text{Mprefix } A P)$
by (*simp add: T-Mprefix* $\langle a' \notin B \rangle$ *image-iff* $*(1, 2)$ $** (1, 2, 3)$)
show $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Throw F-Mprefix*) (*metis* $** (4, 5)$ *****)
qed
qed
qed
qed

corollary *Throw-prefix*: $\langle (a \rightarrow P) \Theta b \in B. Q b =$
 $(a \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } (P \Theta b \in B. Q b))) \rangle$
unfolding *write0-def* **by** (*auto simp add: Throw-Mprefix intro: mono-Mprefix-eq*)

corollary *Throw-Mndetprefix*:
 $\langle (\sqcap a \in A \rightarrow P a) \Theta b \in B. Q b =$
 $\sqcap a \in A \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } P a \Theta b \in B. Q b) \rangle$
apply (*subst Mndetprefix-GlobalNdet*)
apply (*simp add: Throw-GlobalNdet(1) Throw-prefix*)
apply (*subst Mndetprefix-GlobalNdet[symmetric]*)
by *simp*

— We may prove some results about deadlock freeness as corollaries

2.2.6 Continuity

lemma *chain-left-Throw*: $\langle \text{chain } Y \implies \text{chain } (\lambda i. Y i \Theta a \in A. Q a) \rangle$
by (*simp add: chain-def*)

lemma *chain-right-Throw*: $\langle \text{chain } Y \implies \text{chain } (\lambda i. P \Theta a \in A. Y i a) \rangle$
by (*simp add: chain-def fun-belowD*)

lemma *cont-left-prem-Throw* :
 $\langle (\sqcup i. Y i) \Theta a \in A. Q a = (\sqcup i. Y i \Theta a \in A. Q a) \rangle$
 $(\text{is } \langle ?lhs = ?rhs \rangle)$ **if** *chain* : $\langle \text{chain } Y \rangle$
proof (*subst Process-eq-spec, safe*)
show $\langle s \in \mathcal{D} ?lhs \implies s \in \mathcal{D} ?rhs \rangle$ **for** *s*
by (*auto simp add: limproc-is-thelub chain*
chain-left-Throw D-Throw T-LUB D-LUB)

next
fix *s*
define *S*
where $\langle S i \equiv \{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} (Y i) \wedge \text{tickFree } t1 \wedge$
 $\text{set } t1 \cap \text{ev } ' A = \{\} \wedge \text{front-tickFree } t2\} \cup$

$t1 \wedge \{t1. \exists a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} (Y i) \wedge tickFree$
 $t1 \wedge \text{set } t1 \cap ev ' A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D} (Q a)\}$ **for** i
assume $\langle s \in \mathcal{D} ?rhs \rangle$
hence $ftF: \langle front-tickFree s \rangle$ **using** $D\text{-imp-front-tickFree}$ **by** $blast$
from $\langle s \in \mathcal{D} ?rhs \rangle$ **have** $\langle s \in \mathcal{D} (Y i \ominus a \in A. Q a) \rangle$ **for** i
by $(simp \text{ add: } limproc\text{-is-thelub } D\text{-LUB } chain\text{-left-Throw } chain)$
hence $\langle \forall i. S i \neq \{\} \rangle$
by $(simp \text{ add: } S\text{-def } D\text{-Throw})$
 $(metis ftF front-tickFree\text{-mono } list.\text{distinct}(1))$
moreover **have** $\langle finite (S 0) \rangle$
unfolding $S\text{-def}$
apply $(rule finite\text{-subset}[of - \langle t1. \exists t2. s = t1 @ t2 \rangle], blast)$
by $(metis prefixes\text{-fin})$
moreover **have** $\langle \forall i. S (Suc i) \subseteq S i \rangle$
unfolding $S\text{-def}$ **apply** $(intro allI Un\text{-mono } subsetI; simp)$
by $(metis in\text{-mono } le\text{-approx1 } po\text{-class.chainE } chain)$
 $(metis le\text{-approx-lemma-T } po\text{-class.chain-def } subset\text{-eq } chain)$
ultimately **have** $\langle (\bigcap i. S i) \neq \{\} \rangle$
by $(rule Inter\text{-nonempty-finite-chained-sets})$
then **obtain** $t1$ **where** $*$: $\langle \forall i. t1 \in S i \rangle$
by $(meson INT\text{-iff } ex\text{-in-conv } iso\text{-tuple-UNIV-I})$
show $\langle s \in \mathcal{D} ?lhs \rangle$
proof $(cases \langle \exists j a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} (Y j) \wedge a \in A \wedge$
 $t2 \in \mathcal{D} (Q a) \rangle)$
case $True$
then **obtain** $j a t2$ **where** $**$: $\langle s = t1 @ ev a \# t2 \rangle \langle t1 @ [ev a] \in \mathcal{T} (Y j) \rangle$
 $\langle a \in A \rangle \langle t2 \in \mathcal{D} (Q a) \rangle$ **by** $blast$
from $** (1)$ **have** $\langle \forall i. t1 @ [ev a] \in \mathcal{T} (Y i) \rangle$
by $(simp \text{ add: } S\text{-def}) (meson D\text{-T } front\text{-tickFree-single } is\text{-processT7-S})$
with $** (1, 3, 4)$ **show** $\langle s \in \mathcal{D} ?lhs \rangle$
by $(simp \text{ add: } S\text{-def } D\text{-Throw } limproc\text{-is-thelub } chain T\text{-LUB}) blast$
next
case $False$
with $*$ **have** $\langle \forall i. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} (Y i) \wedge front\text{-tickFree } t2 \rangle$
by $(simp \text{ add: } S\text{-def}) blast$
hence $\langle \exists t2. s = t1 @ t2 \wedge (\forall i. t1 \in \mathcal{D} (Y i)) \wedge front\text{-tickFree } t2 \rangle$ **by** $blast$
with $*$ **show** $\langle s \in \mathcal{D} ?lhs \rangle$
by $(simp \text{ add: } S\text{-def } D\text{-Throw } limproc\text{-is-thelub } chain D\text{-LUB}) blast$
qed
next
show $\langle (s, X) \in \mathcal{F} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$ **for** $s X$
by $(auto simp \text{ add: } limproc\text{-is-thelub } chain\text{-left-Throw}$
 $F\text{-Throw } F\text{-LUB } T\text{-LUB } D\text{-LUB})$
next
fix $s X$
define S
where $\langle S i \equiv \{t1. s = t1 \wedge (t1, X) \in \mathcal{F} (Y i) \wedge \text{set } t1 \cap ev ' A = \{\}\} \cup$
 $\{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} (Y i) \wedge tickFree t1 \wedge$

$$\text{set } t1 \cap \text{ev } \langle A = \{\} \wedge \text{front-tickFree } t2 \rangle \cup$$

$$\{t1. \exists a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} (Y i) \wedge$$

$$\text{set } t1 \cap \text{ev } \langle A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a)\rangle\} \text{ for } i$$

assume $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
hence $\text{ftF}: \langle \text{front-tickFree } s \rangle$ **using** is-processT2 **by** blast
from $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ **have** $\langle (s, X) \in \mathcal{F} (Y i \ominus a \in A. Q a) \rangle$ **for** i
by $(\text{simp add: limproc-is-thelub F-LUB chain-left-Throw chain})$

hence $\langle \forall i. S i \neq \{\} \rangle$ **by** $(\text{simp add: S-def F-Throw, safe; simp, blast})$
moreover **have** $\langle \text{finite } (S 0) \rangle$
unfolding $S\text{-def}$
apply $(\text{intro finite-UnI})$
apply $(\text{all } \langle \text{rule finite-subset[of - } \langle \{t1. \exists t2. s = t1 @ t2\} \rangle, \text{blast} \rangle)$
by $(\text{metis prefixes-fin})+$
moreover **have** $\langle \forall i. S (Suc i) \subseteq S i \rangle$
unfolding $S\text{-def}$ **apply** $(\text{intro allI Un-mono subsetI; simp})$
subgoal **by** $(\text{meson is-processT8 po-class.chainE proc-ord2a chain})$
subgoal **by** $(\text{metis in-mono le-approx1 po-class.chainE chain})$
by $(\text{metis le-approx-lemma-T po-class.chain-def subset-eq chain})$
ultimately **have** $\langle (\bigcap i. S i) \neq \{\} \rangle$
by $(\text{rule Inter-nonempty-finite-chained-sets})$
then obtain $t1$ **where** $*$: $\langle \forall i. t1 \in S i \rangle$
by $(\text{meson INT-iff ex-in-conv iso-tuple-UNIV-I})$
show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof $(\text{cases } \langle \exists j a t2. s = t1 @ \text{ev } a \# t2 \wedge t1 @ [\text{ev } a] \in \mathcal{T} (Y j) \wedge$
 $a \in A \wedge (t2, X) \in \mathcal{F} (Q a) \rangle)$

case $\text{True1} : \text{True}$
then obtain $j a t2$ **where** $**$: $\langle s = t1 @ \text{ev } a \# t2 \rangle \langle t1 @ [\text{ev } a] \in \mathcal{T} (Y j) \rangle$
 $\langle a \in A \rangle \langle (t2, X) \in \mathcal{F} (Q a) \rangle$ **by** blast
from $** (1)$ **have** $\langle \forall i. t1 @ [\text{ev } a] \in \mathcal{T} (Y i) \rangle$
by (simp add: S-def) $(\text{meson D-T front-tickFree-single is-processT7})$
with $** (1, 3, 4)$ **show** $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by $(\text{simp add: S-def F-Throw limproc-is-thelub chain T-LUB})$ blast

next
case $\text{False1} : \text{False}$
show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof $(\text{cases } \langle \forall i. t1 \in \mathcal{D} (Y i) \rangle)$
case $\text{True2} : \text{True}$
with $*$ **show** $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by $(\text{simp add: S-def F-Throw limproc-is-thelub chain})$
 $(\text{metis D-LUB-2 append-Nil2 front-tickFree-mono ftF process-charn chain})$

next
case $\text{False2} : \text{False}$
then obtain j **where** $\langle t1 \notin \mathcal{D} (Y j) \rangle$ **by** blast
with $\text{False1 } *$ **have** $**$: $\langle s = t1 \wedge (t1, X) \in \mathcal{F} (Y j) \wedge \text{set } t1 \cap \text{ev } \langle A = \{\} \rangle$
by (simp add: S-def) blast
with $*$ $D\text{-F}$ **have** $\langle \forall i. (t1, X) \in \mathcal{F} (Y i) \rangle$ **by** $(\text{auto simp add: S-def})$
with $**$ **show** $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by $(\text{simp add: F-Throw limproc-is-thelub F-LUB chain})$

qed
 qed
 qed

lemma *cont-right-prem-Throw* :

$\langle P \Theta a \in A. (\bigsqcup i. Y i a) = (\bigsqcup i. P \Theta a \in A. Y i a) \rangle$

(is $\langle ?lhs = ?rhs \rangle$) **if** *chain* : $\langle chain Y \rangle$

proof (*subst Process-eq-spec, safe*)

show $\langle s \in \mathcal{D} ?lhs \implies s \in \mathcal{D} ?rhs \rangle$ **for** *s*

by (*simp add: limproc-is-thelub chain chain-right-Throw*
ch2ch-fun[OF chain] D-Throw D-LUB) *blast*

next

fix *s*

assume $\langle s \in \mathcal{D} ?rhs \rangle$

define *S*

where $\langle S i \equiv \{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge tickFree t1 \wedge$
 $set t1 \cap ev ' A = \{\} \wedge front-tickFree t2\} \cup$
 $\{t1. \exists a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} P \wedge$
 $set t1 \cap ev ' A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D} (Y i a)\} \rangle$ **for** *i*

assume $\langle s \in \mathcal{D} ?rhs \rangle$

hence $\langle s \in \mathcal{D} (P \Theta a \in A. Y i a) \rangle$ **for** *i*

by (*simp add: limproc-is-thelub D-LUB chain-right-Throw chain*)

hence $\langle \forall i. S i \neq \{\} \rangle$ **by** (*simp add: S-def D-Throw metis*)

moreover have $\langle finite (S 0) \rangle$

unfolding *S-def*

apply (*rule finite-subset[of - $\langle \{t1. \exists t2. s = t1 @ t2\} \rangle$], *blast*)*

by (*metis prefixes-fin*)

moreover have $\langle \forall i. S (Suc i) \subseteq S i \rangle$

unfolding *S-def* **apply** (*intro allI Un-mono subsetI; simp*)

by (*metis fun-belowD le-approx1 po-class.chainE subset-iff chain*)

ultimately have $\langle (\bigcap i. S i) \neq \{\} \rangle$

by (*rule Inter-nonempty-finite-chained-sets*)

then obtain *t1* **where** $\langle \forall i. t1 \in S i \rangle$

by (*meson INT-iff ex-in-conv iso-tuple-UNIV-I*)

then consider $\langle t1 \in \mathcal{D} P \rangle$ $\langle tickFree t1 \rangle$

$\langle set t1 \cap ev ' A = \{\} \rangle$ $\langle \exists t2. s = t1 @ t2 \wedge front-tickFree t2 \rangle$

| $\langle set t1 \cap ev ' A = \{\} \rangle$

$\langle \forall i. \exists a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} P \wedge a \in A \wedge t2 \in \mathcal{D} (Y i$

a) \rangle

by (*simp add: S-def*) *blast*

thus $\langle s \in \mathcal{D} ?lhs \rangle$

proof cases

show $\langle t1 \in \mathcal{D} P \implies tickFree t1 \implies set t1 \cap ev ' A = \{\} \implies$

$\exists t2. s = t1 @ t2 \wedge front-tickFree t2 \implies s \in \mathcal{D} ?lhs \rangle$

by (*simp add: D-Throw*) *blast*

next

assume *assms*: $\langle set t1 \cap ev ' A = \{\} \rangle$

$\langle \forall i. \exists a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} P \wedge$

$a \in A \wedge t2 \in \mathcal{D} (Y i a)$

from $assms(2)$ **obtain** $a t2$

where $*$: $\langle s = t1 @ ev a \# t2 \rangle \langle t1 @ [ev a] \in \mathcal{T} P \rangle \langle a \in A \rangle$ **by** *blast*

with $assms(2)$ **have** $\langle \forall i. t2 \in \mathcal{D} (Y i a) \rangle$ **by** *blast*

with $assms(1) *(1, 2, 3)$ **show** $\langle s \in \mathcal{D} ?lhs \rangle$

by (*simp add: D-Throw limproc-is-thelub chain ch2ch-fun D-LUB*) *blast*

qed

next

show $\langle (s, X) \in \mathcal{F} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$ **for** $s X$

by (*simp add: limproc-is-thelub chain chain-right-Throw ch2ch-fun[OF chain] F-Throw F-LUB T-LUB D-LUB*) *blast*

next

fix $s X$

define S

where $\langle S i \equiv \{t1. s = t1 \wedge (t1, X) \in \mathcal{F} P \wedge set t1 \cap ev 'A = \{\}\} \cup$
 $\{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge tickFree t1 \wedge$
 $set t1 \cap ev 'A = \{\} \wedge front-tickFree t2\} \cup$
 $\{t1. \exists a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} P \wedge$
 $set t1 \cap ev 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Y i a)\} \rangle$ **for** i

assume $\langle (s, X) \in \mathcal{F} ?rhs \rangle$

hence $\langle (s, X) \in \mathcal{F} (P \Theta a \in A. Y i a) \rangle$ **for** i

by (*simp add: limproc-is-thelub F-LUB chain-right-Throw chain*)

hence $\langle \forall i. S i \neq \{\} \rangle$ **by** (*simp add: S-def F-Throw, safe; simp, blast*)

moreover **have** $\langle finite (S 0) \rangle$

unfolding *S-def*

apply (*intro finite-UnI*)

apply (*all rule finite-subset[of - <\{t1. \exists t2. s = t1 @ t2\}], blast*)

by (*metis prefixes-fin*) $+$

moreover **have** $\langle \forall i. S (Suc i) \subseteq S i \rangle$

unfolding *S-def* **apply** (*intro allI Un-mono subsetI; simp*)

by (*metis NF-ND fun-below-iff po-class.chain-def proc-ord2a chain*)

ultimately **have** $\langle (\bigcap i. S i) \neq \{\} \rangle$

by (*rule Inter-nonempty-finite-chained-sets*)

then **obtain** $t1$ **where** $\langle \forall i. t1 \in S i \rangle$

by (*meson INT-iff ex-in-conv iso-tuple-UNIV-I*)

then **consider** $\langle s = t1 \wedge (t1, X) \in \mathcal{F} P \rangle \langle set t1 \cap ev 'A = \{\} \rangle$

| $\langle \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge tickFree t1 \wedge$
 $set t1 \cap ev 'A = \{\} \wedge front-tickFree t2 \rangle$

| $\langle set t1 \cap ev 'A = \{\} \rangle$

$\langle \forall i. \exists a t2. s = t1 @ ev a \# t2 \wedge t1 @ [ev a] \in \mathcal{T} P \wedge a \in A \wedge (t2, X) \in$
 $\mathcal{F} (Y i a) \rangle$

by (*simp add: S-def*) *blast*

thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$

proof *cases*

show $\langle s = t1 \wedge (t1, X) \in \mathcal{F} P \implies set t1 \cap ev 'A = \{\} \implies (s, X) \in \mathcal{F} ?lhs \rangle$

by (*simp add: F-Throw*)

next

show $\langle \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} P \wedge tickFree t1 \wedge$
 $set t1 \cap ev 'A = \{\} \wedge front-tickFree t2 \implies (s, X) \in \mathcal{F} ?lhs \rangle$

```

    by (simp add: F-Throw) blast
next
assume assms: ⟨set t1 ∩ ev ' A = {}⟩
            ⟨∀ i. ∃ a t2. s = t1 @ ev a # t2 ∧ t1 @ [ev a] ∈ T P ∧
              a ∈ A ∧ (t2, X) ∈ F (Y i a)⟩
from this(2) obtain a t2
  where * : ⟨s = t1 @ ev a # t2⟩ ⟨t1 @ [ev a] ∈ T P⟩ ⟨a ∈ A⟩ by blast
with assms(2) have ⟨∀ i. (t2, X) ∈ F (Y i a)⟩ by blast
with *(1, 2, 3) assms(1) show ⟨(s, X) ∈ F ?lhs⟩
  by (simp add: F-Throw limproc-is-thelub ch2ch-fun chain F-LUB) blast
qed
qed

```

```

lemma Throw-cont[simp] :
  assumes cont-f : ⟨cont f⟩ and cont-g : ⟨∀ a. cont (g a)⟩
  shows ⟨cont (λx. f x ⊖ a ∈ A. g a x)⟩
proof -
  have * : ⟨cont (λy. y ⊖ a ∈ A. g a x)⟩ for x
  by (rule contI2, rule monofunI, solves simp, simp add: cont-left-prem-Throw)
  have ⟨cont (Throw y A)⟩ for y
  by (simp add: contI2 cont-right-prem-Throw fun-belowD lub-fun monofunI)
  hence ** : ⟨cont (λx. y ⊖ a ∈ A. g a x)⟩ for y
  by (rule cont-compose) (simp add: cont-g)
  show ?thesis by (fact cont-apply[OF cont-f * **])
qed
end

```

2.3 The Interrupt Operator

```

theory Interrupt
  imports HOL-CSPM.CSPM
begin

```

2.3.1 Definition

We want to add the binary operator of interruption of P by Q : it behaves like P except that at any time Q can take over.

The definition provided by Roscoe [4, p.239] does not respect the invariant *is-process*: it seems like *tick* is not handled.

We propose here our corrected version.

```

lift-definition Interrupt :: ⟨['α process, 'α process] ⇒ 'α process⟩ (infixl ⟨Δ⟩ 75)
is ⟨λP Q.
  ({(t1 @ [tick], X) | t1 X. t1 @ [tick] ∈ T P} ∪
  {(t1, X - {tick}) | t1 X. t1 @ [tick] ∈ T P}) ∪

```

$\{(t1, X) \in \mathcal{F} P. \text{tickFree } t1 \wedge ([], X) \in \mathcal{F} Q\} \cup$
 $\{(t1 @ t2, X) \mid t1 t2 X. t1 \in \mathcal{T} P \wedge \text{tickFree } t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq []\} \cup$
 $\{(t1, X - \{\text{tick}\}) \mid t1 X. t1 \in \mathcal{T} P \wedge \text{tickFree } t1 \wedge [\text{tick}] \in \mathcal{T} Q\} \cup$
 $\{(t1, X). t1 \in \mathcal{D} P\} \cup$
 $\{(t1 @ t2, X) \mid t1 t2 X. t1 \in \mathcal{T} P \wedge \text{tickFree } t1 \wedge t2 \in \mathcal{D} Q\},$
 $\mathcal{D} P \cup \{t1 @ t2 \mid t1 t2. t1 \in \mathcal{T} P \wedge \text{tickFree } t1 \wedge t2 \in \mathcal{D} Q\}$

proof –

show $\langle ?thesis P Q \rangle$

(is $\langle is\text{-process } (?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7, ?d1 \cup ?d2) \rangle$ **for** $P Q$

unfolding $is\text{-process-def FAILURES-def DIVERGENCES-def fst\text{-conv snd\text{-conv}}$

proof ($intro conjI allI impI$)

have $\langle ([], \{\}) \in ?f3 \rangle$ **by** ($simp add: is\text{-process}T1$)

thus $\langle ([], \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ **by** $fast$

next

show $\langle (s, X) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \implies \text{front}\text{-tickFree } s \rangle$

for $s X$

by ($simp add: is\text{-process}T2 D\text{-imp}\text{-front}\text{-tickFree front}\text{-tickFree}\text{-append}$)

($meson \text{front}\text{-tickFree}\text{-append front}\text{-tickFree}\text{-dw}\text{-closed is}\text{-process}T2\text{-TR process}\text{-charn}$)

next

fix $s t$

show $\langle (s @ t, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \implies$
 $(s, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$

proof ($induct t \text{ rule: rev}\text{-induct}$)

show $\langle (s @ [], \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \implies$
 $(s, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ **by** $simp$

next

fix $a t$

assume $assm : \langle (s @ t @ [a], \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$

and $hyp : \langle (s @ t, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \implies$
 $(s, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$

from $assm$ **have** $\langle (s @ t @ [a], \{\}) \in ?f1 \vee (s @ t @ [a], \{\}) \in ?f2 \vee$
 $(s @ t @ [a], \{\}) \in ?f3 \vee (s @ t @ [a], \{\}) \in ?f4 \vee (s @ t @ [a], \{\}) \in ?f5$

\vee

$(s @ t @ [a], \{\}) \in ?f6 \vee (s @ t @ [a], \{\}) \in ?f7 \rangle$ **by** $fast$

thus $\langle (s, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$

proof ($elim \text{disjE}$)

assume $\langle (s @ t @ [a], \{\}) \in ?f1 \rangle$

hence $\langle (s, \{\}) \in ?f3 \rangle$

by $simp (meson NF\text{-NT append}\text{-T}\text{-imp}\text{-tickFree is}\text{-process}T \text{snoc}\text{-eq}\text{-iff}\text{-butlast})$

thus $\langle (s, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ **by** $blast$

next

assume $\langle (s @ t @ [a], \{\}) \in ?f2 \rangle$

hence $\langle (s, \{\}) \in ?f3 \rangle$

by $simp (metis NF\text{-NT Nil}\text{-is}\text{-append}\text{-conv append}\text{-T}\text{-imp}\text{-tickFree is}\text{-process}T \text{list}\text{-discI})$

thus $\langle (s, \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ **by** $blast$

```

next
  assume  $\langle s @ t @ [a], \{\} \rangle \in ?f3$ 
  with is-processT3 have  $\langle s, \{\} \rangle \in ?f3$  by simp blast
  thus  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$  by blast
next
  assume  $\langle s @ t @ [a], \{\} \rangle \in ?f4$ 
  then obtain  $t1\ t2$  where  $* : \langle s @ t = t1 @ t2 \rangle \langle t1 \in \mathcal{T}\ P \rangle$ 
     $\langle tickFree\ t1 \rangle \langle t2 @ [a], \{\} \rangle \in \mathcal{F}\ Q$ 
    by simp (metis butlast-append last-appendR snoc-eq-iff-butlast)
  show  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$ 
  proof (cases  $\langle t2 = [] \rangle$ )
    assume  $\langle t2 = [] \rangle$ 
    with  $*(1, 2, 3)$  have  $\langle s, \{\} \rangle \in ?f3$ 
    by simp (metis T-F process-charn tickFree-append)
    thus  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$  by blast
  next
    assume  $\langle t2 \neq [] \rangle$ 
    with  $*\ is-processT3$  have  $\langle s @ t, \{\} \rangle \in ?f4$  by simp blast
    thus  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$  by (intro hyp)
blast
qed
next
  assume  $\langle s @ t @ [a], \{\} \rangle \in ?f5$ 
  hence  $\langle s, \{\} \rangle \in ?f3$  by simp (metis T-F process-charn)
  thus  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$  by blast
next
  assume  $\langle s @ t @ [a], \{\} \rangle \in ?f6$ 
  hence  $\langle s, \{\} \rangle \in ?f3$ 
  by simp (meson front-tickFree-mono is-processT snoc-eq-iff-butlast)
  thus  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$  by blast
next
  assume  $\langle s @ t @ [a], \{\} \rangle \in ?f7$ 
  then obtain  $t1\ t2$  where  $* : \langle s @ t @ [a] = t1 @ t2 \rangle \langle t1 \in \mathcal{T}\ P \rangle$ 
     $\langle tickFree\ t1 \rangle \langle t2 \in \mathcal{D}\ Q \rangle$  by blast
  hence  $\langle s @ t, \{\} \rangle \in (\text{if length } t2 \leq 1 \text{ then } ?f3 \text{ else } ?f4)$ 
  apply (cases  $t2$  rule: rev-cases; simp)
  by (metis T-F append-assoc process-charn tickFree-append)
    (metis D-T T-F is-processT3-ST)
  thus  $\langle s, \{\} \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$ 
  by (intro hyp) (meson UnI1 UnI2)
qed
next
  fix  $s\ X\ Y$ 
  assume assm :  $\langle s, Y \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \wedge X \subseteq Y$ 
  hence  $\langle s, Y \rangle \in ?f1 \vee \langle s, Y \rangle \in ?f2 \vee \langle s, Y \rangle \in ?f3 \vee \langle s, Y \rangle \in ?f4 \vee$ 
     $\langle s, Y \rangle \in ?f5 \vee \langle s, Y \rangle \in ?f6 \vee \langle s, Y \rangle \in ?f7$  by fast
  thus  $\langle s, X \rangle \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7$ 
  proof (elim disjE)

```

```

    assume ⟨(s, Y) ∈ ?f1⟩
    hence ⟨(s, X) ∈ ?f1⟩ by simp
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
    assume ⟨(s, Y) ∈ ?f2⟩
    with assem[THEN conjunct2] have ⟨(s, X) ∈ ?f2⟩ by simp blast
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
    assume ⟨(s, Y) ∈ ?f3⟩
    with assem[THEN conjunct2] is-processT4 have ⟨(s, X) ∈ ?f3⟩ by simp blast
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
    assume ⟨(s, Y) ∈ ?f4⟩
    with assem[THEN conjunct2] is-processT4 have ⟨(s, X) ∈ ?f4⟩ by simp blast
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
    assume ⟨(s, Y) ∈ ?f5⟩
    with assem[THEN conjunct2] have ⟨(s, X) ∈ ?f5⟩ by simp blast
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
    assume ⟨(s, Y) ∈ ?f6⟩
    hence ⟨(s, X) ∈ ?f6⟩ by simp
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
    assume ⟨(s, Y) ∈ ?f7⟩
    hence ⟨(s, X) ∈ ?f7⟩ by simp
    thus ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
qed
next
fix s X Y
assume assem : ⟨(s, X) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7 ∧
  (∀ c. c ∈ Y → (s @ [c], {}) ∉ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6
  ∪ ?f7)⟩
have ⟨(s, X) ∈ ?f1 ∨ (s, X) ∈ ?f2 ∨ (s, X) ∈ ?f3 ∨ (s, X) ∈ ?f4 ∨
  (s, X) ∈ ?f5 ∨ (s, X) ∈ ?f6 ∨ (s, X) ∈ ?f7⟩ using assem[THEN conjunct1]
by fast
thus ⟨(s, X ∪ Y) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩
proof (elim disjE)
  assume ⟨(s, X) ∈ ?f1⟩
  hence ⟨(s, X ∪ Y) ∈ ?f1⟩ by simp
  thus ⟨(s, X ∪ Y) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
  assume ⟨(s, X) ∈ ?f2⟩
  with assem[THEN conjunct2] have ⟨(s, X ∪ Y) ∈ ?f2⟩ by simp blast
  thus ⟨(s, X ∪ Y) ∈ ?f1 ∪ ?f2 ∪ ?f3 ∪ ?f4 ∪ ?f5 ∪ ?f6 ∪ ?f7⟩ by blast
next
  assume ⟨(s, X) ∈ ?f3⟩
  with assem[THEN conjunct2] have ⟨(s, X ∪ Y) ∈ ?f3⟩
  by simp (metis F-T T-F append-Nil is-processT5-S7' list.distinct(1))

```

```

thus  $\langle (s, X \cup Y) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
next
  assume  $\langle (s, X) \in ?f4 \rangle$ 
  with assm[THEN conjunct2] have  $\langle (s, X \cup Y) \in ?f4 \rangle$ 
  by simp (metis append.assoc append-is-Nil-conv is-processT5-S1)
  thus  $\langle (s, X \cup Y) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
next
  assume  $\langle (s, X) \in ?f5 \rangle$ 
  with assm[THEN conjunct2] have  $\langle (s, X \cup Y) \in ?f5 \rangle$ 
  by simp (metis Diff-empty Diff-insert0 T-F Un-Diff not-Cons-self)
  thus  $\langle (s, X \cup Y) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
next
  assume  $\langle (s, X) \in ?f6 \rangle$ 
  hence  $\langle (s, X \cup Y) \in ?f6 \rangle$  by simp
  thus  $\langle (s, X \cup Y) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
next
  assume  $\langle (s, X) \in ?f7 \rangle$ 
  hence  $\langle (s, X \cup Y) \in ?f7 \rangle$  by simp
  thus  $\langle (s, X \cup Y) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
qed
next
fix  $s X$ 
have  $*$  :  $\langle (s @ [tick], \{\}) \notin ?f2 \cup ?f3 \cup ?f5 \rangle$ 
  by simp (metis Cons-eq-appendI append-self-conv2 front-tickFree-mono
    is-processT2-TR list.distinct(1) non-tickFree-tick)
assume  $\langle (s @ [tick], \{\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ 
with  $*$  have  $\langle (s @ [tick], \{\}) \in ?f1 \vee (s @ [tick], \{\}) \in ?f4 \vee$ 
   $(s @ [tick], \{\}) \in ?f6 \vee (s @ [tick], \{\}) \in ?f7 \rangle$  by fast
thus  $\langle (s, X - \{tick\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ 
proof (elim disjE)
  assume  $\langle (s @ [tick], \{\}) \in ?f1 \rangle$ 
  hence  $\langle (s, X - \{tick\}) \in ?f2 \rangle$  by blast
  thus  $\langle (s, X - \{tick\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
next
  assume  $\langle (s @ [tick], \{\}) \in ?f4 \rangle$ 
  then obtain  $t1 t2$ 
  where  $**$  :  $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T} P \rangle \langle tickFree t1 \rangle \langle (t2 @ [tick], \{\}) \in \mathcal{F}$ 
 $Q \rangle$ 
  by simp (metis butlast-append last-appendR snoc-eq-iff-butlast)
  hence  $\langle (s, X - \{tick\}) \in (if t2 = [] then ?f5 else ?f4) \rangle$ 
  by simp (metis F-T process-charn self-append-conv2)
  thus  $\langle (s, X - \{tick\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by (meson
UnCI)
next
  assume  $\langle (s @ [tick], \{\}) \in ?f6 \rangle$ 
  with is-processT9 have  $\langle s \in \mathcal{D} P \rangle$  by simp blast
  thus  $\langle (s, X - \{tick\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
next
  assume  $\langle (s @ [tick], \{\}) \in ?f7 \rangle$ 

```



```

then obtain  $t1\ t2$  where  $** : \langle s @ [tick] = t1 @ t2 \rangle \langle t1 \in \mathcal{T}\ P \rangle$ 
       $\langle tickFree\ t1 \rangle \langle t2 \in \mathcal{D}\ Q \rangle$  by blast
from  $** (1, 3, 4)$  obtain  $t2'$  where  $\langle t2 = t2' @ [tick] \rangle \langle t2' @ [tick] \in \mathcal{D}\ Q \rangle$ 
  by (cases  $t2$  rule: rev-cases) auto
with  $** (1)$  is-processT9 have  $\langle s = t1 @ t2' \wedge t2' \in \mathcal{D}\ Q \rangle$  by simp blast
with  $** (2, 3)$  have  $\langle (s, X - \{tick\}) \in ?f7 \rangle$  by simp blast
thus  $\langle (s, X - \{tick\}) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$  by blast
qed
next
show  $\langle s \in ?d1 \cup ?d2 \wedge tickFree\ s \wedge front-tickFree\ t \implies s @ t \in ?d1 \cup ?d2 \rangle$ 
for  $s\ t$ 
  apply (simp, elim conjE disjE exE)
  by (solves  $\langle simp\ add: is-processT7 \rangle$ )
    (meson append.assoc is-processT7 tickFree-append)
next
show  $\langle s \in ?d1 \cup ?d2 \implies (s, X) \in ?f1 \cup ?f2 \cup ?f3 \cup ?f4 \cup ?f5 \cup ?f6 \cup ?f7 \rangle$ 
for  $s\ X$ 
  by blast
next
fix  $s$ 
assume  $\langle s @ [tick] \in ?d1 \cup ?d2 \rangle$ 
then consider  $\langle s @ [tick] \in ?d1 \rangle \mid \langle s @ [tick] \in ?d2 \rangle$  by blast
thus  $\langle s \in ?d1 \cup ?d2 \rangle$ 
proof cases
  assume  $\langle s @ [tick] \in ?d1 \rangle$ 
  with is-processT have  $\langle s \in ?d1 \rangle$  by blast
  thus  $\langle s \in ?d1 \cup ?d2 \rangle$  by blast
next
  assume  $\langle s @ [tick] \in ?d2 \rangle$ 
  then obtain  $t1\ t2$  where  $** : \langle s @ [tick] = t1 @ t2 \rangle \langle t1 \in \mathcal{T}\ P \rangle$ 
       $\langle tickFree\ t1 \rangle \langle t2 \in \mathcal{D}\ Q \rangle$  by blast
  from  $** (1, 3, 4)$  obtain  $t2'$  where  $\langle t2 = t2' @ [tick] \rangle \langle t2' @ [tick] \in \mathcal{D}\ Q \rangle$ 
    by (cases  $t2$  rule: rev-cases) auto
  with  $** (1)$  is-processT9 have  $\langle s = t1 @ t2' \wedge t2' \in \mathcal{D}\ Q \rangle$  by simp blast
  with  $** (2, 3)$  have  $\langle s \in ?d2 \rangle$  by simp blast
  thus  $\langle s \in ?d1 \cup ?d2 \rangle$  by blast
qed
qed
qed

```

2.3.2 Projections

lemma *F-Interrupt* :

$$\begin{aligned}
\langle \mathcal{F} (P \triangle Q) = & \\
& \{(t1 @ [tick], X) \mid t1\ X. t1 @ [tick] \in \mathcal{T}\ P\} \cup \\
& \{(t1, X - \{tick\}) \mid t1\ X. t1 @ [tick] \in \mathcal{T}\ P\} \cup \\
& \{(t1, X) \in \mathcal{F}\ P. tickFree\ t1 \wedge ([], X) \in \mathcal{F}\ Q\} \cup \\
& \{(t1 @ t2, X) \mid t1\ t2\ X. t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge (t2, X) \in \mathcal{F}\ Q \wedge t2 \neq []\} \cup \\
& \{(t1, X - \{tick\}) \mid t1\ X. t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge [tick] \in \mathcal{T}\ Q\} \cup
\end{aligned}$$

$\{(t1, X). t1 \in \mathcal{D} P\} \cup$
 $\{(t1 @ t2, X) \mid t1 t2 X. t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{D} Q\}$
by (*simp add: Failures-def FAILURES-def Interrupt.rep-eq*)

lemma *D-Interrupt* :
 $\langle \mathcal{D} (P \Delta Q) = \mathcal{D} P \cup \{t1 @ t2 \mid t1 t2. t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{D} Q\} \rangle$
by (*simp add: Divergences-def DIVERGENCES-def Interrupt.rep-eq*)

lemma *T-Interrupt* :
 $\langle \mathcal{T} (P \Delta Q) = \mathcal{T} P \cup \{t1 @ t2 \mid t1 t2. t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{T} Q\} \rangle$
apply (*simp add: Traces.rep-eq TRACES-def Failures.rep-eq[symmetric] F-Interrupt*)
apply (*safe, simp-all add: is-processT8*)
subgoal by (*metis is-processT3-SR*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*metis is-processT8-S*)
subgoal by (*metis is-processT4-empty nonTickFree-n-frontTickFree process-charn*)
by (*metis append.right-neutral is-processT4-empty tickFree-Nil*)

2.3.3 Monotony

lemma *mono-Interrupt[simp]*: $\langle P \Delta Q \sqsubseteq P' \Delta Q' \rangle$ **if** $\langle P \sqsubseteq P' \rangle$ **and** $\langle Q \sqsubseteq Q' \rangle$

proof (*unfold le-approx-def, intro conjI allI impI subsetI*)

show $\langle s \in \mathcal{D} (P' \Delta Q') \implies s \in \mathcal{D} (P \Delta Q) \rangle$ **for** s
using *that[THEN le-approx1] D-T that(1)[THEN le-approx2T]*
by (*simp add: D-Interrupt*) *blast*

next

show $\langle s \notin \mathcal{D} (P \Delta Q) \implies \mathcal{R}_a (P \Delta Q) s = \mathcal{R}_a (P' \Delta Q') s \rangle$ **for** s

apply (*simp add: D-Interrupt Ra-def F-Interrupt,*
intro subset-antisym subsetI; simp, elim disjE)

subgoal by (*metis le-approx2T that(1)*)

subgoal by (*metis is-processT9 le-approx2T that(1)*)

subgoal by (*metis F-T append.right-neutral le-approx2 that*)

subgoal by (*metis is-processT2 is-processT7 le-approx2T proc-ord2a that*)

subgoal by (*metis (no-types, lifting) append-Nil2 le-approx2T min-elems6*
no-Trace-implies-no-Failure self-append-conv2 that)

subgoal by *metis*

subgoal by (*metis le-approx2T that(1)*)

subgoal by (*metis le-approx-lemma-T subset-eq that(1)*)

subgoal by (*metis is-processT8-S le-approx2 that*)

subgoal by (*metis is-processT2 is-processT7-S le-approx2 le-approx2T that*)

subgoal by (*metis D-T le-approx2T that*)

subgoal by (*metis in-mono le-approx1 that(1)*)

by (*metis le-approx1 le-approx2T process-charn subsetD that*)

next

from *that[THEN le-approx3]*

show $\langle s \in \min\text{-elems} (\mathcal{D} (P \Delta Q)) \implies s \in \mathcal{T} (P' \Delta Q') \rangle$ **for** s

by (*auto simp add: min-elems-def D-Interrupt T-Interrupt subset-iff*)
 (*metis le-approx2T le-list-def less-append order-le-imp-less-or-eq that(1)*)
qed

lemma *mono-Interrupt-T*: $\langle P \sqsubseteq_T P' \implies Q \sqsubseteq_T Q' \implies P \triangle Q \sqsubseteq_T P' \triangle Q' \rangle$
unfolding *trace-refine-def* **by** (*auto simp add: T-Interrupt*)

lemma *mono-right-Interrupt-D*: $\langle Q \sqsubseteq_D Q' \implies P \triangle Q \sqsubseteq_D P \triangle Q' \rangle$
unfolding *divergence-refine-def* **by** (*auto simp add: D-Interrupt*)

— We have no monotony, even partial, with (\sqsubseteq_F) .

lemma *mono-Interrupt-FD*:
 $\langle P \sqsubseteq_{FD} P' \implies Q \sqsubseteq_{FD} Q' \implies P \triangle Q \sqsubseteq_{FD} P' \triangle Q' \rangle$
unfolding *failure-divergence-refine-def le-ref-def*
by (*simp add: D-Interrupt F-Interrupt, safe;*
metis [[metis-verbose = false]] F-subset-imp-T-subset subsetD)

lemma *mono-Interrupt-DT*:
 $\langle P \sqsubseteq_{DT} P' \implies Q \sqsubseteq_{DT} Q' \implies P \triangle Q \sqsubseteq_{DT} P' \triangle Q' \rangle$
unfolding *trace-divergence-refine-def trace-refine-def divergence-refine-def*
by (*auto simp add: T-Interrupt D-Interrupt subset-iff*)

2.3.4 Properties

lemma *Interrupt-STOP-neutral* : $\langle P \triangle STOP = P \rangle \langle STOP \triangle P = P \rangle$
apply (*all* \langle subst Process-eq-spec \rangle , *safe*)
apply (*simp-all add: Process-eq-spec F-Interrupt D-Interrupt F-STOP*
T-STOP
D-STOP T-F is-processT6 is-processT8-S tick-T-F)
subgoal by (*meson process-charn tick-T-F*)
subgoal by (*metis F-T T-nonTickFree-imp-decomp*)
subgoal by (*meson DiffE insertI1 is-processT6-S2 is-processT8-S*)
by *blast*

lemma *Interrupt-BOT-absorb* : $\langle P \triangle \perp = \perp \rangle \langle \perp \triangle P = \perp \rangle$
by (*simp-all add: BOT-iff-D D-Interrupt D-UU Nil-elem-T*)

lemma *Interrupt-is-BOT-iff* : $\langle P \triangle Q = \perp \iff P = \perp \vee Q = \perp \rangle$
by (*simp add: BOT-iff-D D-Interrupt Nil-elem-T*)

lemma *events-Interrupt*: \langle events-of $(P \triangle Q) =$ events-of $P \cup$ events-of $Q \rangle$
apply (*intro subset-antisym subsetI; simp add: events-of-def T-Interrupt*)
by (*metis UnE set-append*) (*metis Nil-elem-T append-Nil tickFree-Nil*)

lemma *Interrupt-Ndet-distrib* : $\langle P \triangle Q1 \sqcap Q2 = (P \triangle Q1) \sqcap (P \triangle Q2) \rangle$

```

(is ⟨?lhs = ?rhs⟩)
proof (subst Process-eq-spec-optimized, safe)
  show ⟨s ∈ D ?lhs ⟹ s ∈ D ?rhs⟩ for s
    by (auto simp add: D-Interrupt D-Ndet)
next
  show ⟨s ∈ D ?rhs ⟹ s ∈ D ?lhs⟩ for s
    by (auto simp add: D-Interrupt D-Ndet)
next
  fix s X
  assume same-div : ⟨D ?lhs = D ?rhs⟩
  assume ⟨(s, X) ∈ F ?lhs⟩
  then consider ⟨s ∈ D (P Δ Q1 □ Q2)⟩
    | ⟨∃ t1. s = t1 @ [tick] ∧ t1 @ [tick] ∈ T P⟩
    | ⟨s @ [tick] ∈ T P ∧ tick ∉ X⟩
    | ⟨(s, X) ∈ F P ∧ tickFree s ∧ ([], X) ∈ F (Q1 □ Q2)⟩
    | ⟨∃ t1 t2. s = t1 @ t2 ∧ t1 ∈ T P ∧ tickFree t1 ∧ (t2, X) ∈ F (Q1 □ Q2) ∧
t2 ≠ []⟩
    | ⟨s ∈ T P ∧ tickFree s ∧ [tick] ∈ T (Q1 □ Q2) ∧ tick ∉ X⟩
  by (simp add: F-Interrupt D-Interrupt) blast
  thus ⟨(s, X) ∈ F ?rhs⟩
proof cases
  from same-div D-F show ⟨s ∈ D ?lhs ⟹ (s, X) ∈ F ?rhs⟩ by blast
next
  show ⟨∃ t1. s = t1 @ [tick] ∧ t1 @ [tick] ∈ T P ⟹ (s, X) ∈ F ?rhs⟩
    by (simp add: F-Ndet F-Interrupt)
next
  show ⟨s @ [tick] ∈ T P ∧ tick ∉ X ⟹ (s, X) ∈ F ?rhs⟩
    by (simp add: F-Ndet F-Interrupt) (metis Diff-insert-absorb)
next
  show ⟨(s, X) ∈ F P ∧ tickFree s ∧ ([], X) ∈ F (Q1 □ Q2) ⟹ (s, X) ∈ F
?rhs⟩
    by (simp add: F-Ndet F-Interrupt) blast
next
  show ⟨∃ t1 t2. s = t1 @ t2 ∧ t1 ∈ T P ∧ tickFree t1 ∧
(t2, X) ∈ F (Q1 □ Q2) ∧ t2 ≠ [] ⟹ (s, X) ∈ F ?rhs⟩
    by (simp add: F-Ndet F-Interrupt) metis
next
  show ⟨s ∈ T P ∧ tickFree s ∧ [tick] ∈ T (Q1 □ Q2) ∧ tick ∉ X ⟹ (s, X) ∈
F ?rhs⟩
    by (simp add: F-Interrupt F-Ndet T-Ndet) (metis Diff-insert-absorb)
qed
next
  have ⟨(s, X) ∈ F (P Δ Q1) ⟹ (s, X) ∈ F (P Δ Q1 □ Q2)⟩ for s X Q1 Q2
    by (simp add: F-Interrupt F-Ndet D-Ndet T-Ndet) blast
  thus ⟨(s, X) ∈ F ?rhs ⟹ (s, X) ∈ F ?lhs⟩ for s X
    by (simp add: F-Ndet) (metis Ndet-commute)
qed

```

lemma *Ndet-Interrupt-distrib* : $\langle P1 \sqcap P2 \triangle Q = (P1 \triangle Q) \sqcap (P2 \triangle Q) \rangle$
 (is $\langle ?lhs = ?rhs \rangle$)
proof (*subst Process-eq-spec-optimized, safe*)
show $\langle s \in \mathcal{D} ?lhs \implies s \in \mathcal{D} ?rhs \rangle$ **for** s
by (*auto simp add: D-Ndet T-Ndet D-Interrupt*)
next
show $\langle s \in \mathcal{D} ?rhs \implies s \in \mathcal{D} ?lhs \rangle$ **for** s
by (*auto simp add: D-Ndet T-Ndet D-Interrupt*)
next
fix $s X$
assume *same-div* : $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
then consider $\langle s \in \mathcal{D} ?lhs \rangle$
 | $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (P1 \sqcap P2) \rangle$
 | $\langle s @ [tick] \in \mathcal{T} (P1 \sqcap P2) \wedge tick \notin X \rangle$
 | $\langle (s, X) \in \mathcal{F} (P1 \sqcap P2) \wedge tickFree s \wedge ([], X) \in \mathcal{F} Q \rangle$
 | $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (P1 \sqcap P2) \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \rangle$
 | $\langle s \in \mathcal{T} (P1 \sqcap P2) \wedge tickFree s \wedge [tick] \in \mathcal{T} Q \wedge tick \notin X \rangle$
by (*simp add: F-Interrupt D-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F} ?rhs \rangle$
proof cases
from same-div D-F **show** $\langle s \in \mathcal{D} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$ **by** *blast*
next
show $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (P1 \sqcap P2) \implies (s, X) \in \mathcal{F} ?rhs \rangle$
by (*simp add: T-Ndet F-Ndet F-Interrupt*) *metis*
next
show $\langle s @ [tick] \in \mathcal{T} (P1 \sqcap P2) \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?rhs \rangle$
by (*simp add: T-Ndet F-Ndet F-Interrupt*) (*metis Diff-insert-absorb*)
next
show $\langle (s, X) \in \mathcal{F} (P1 \sqcap P2) \wedge tickFree s \wedge ([], X) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} ?rhs \rangle$
by (*simp add: F-Ndet F-Interrupt*) *blast*
next
show $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (P1 \sqcap P2) \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \implies (s, X) \in \mathcal{F} ?rhs \rangle$
by (*simp add: T-Ndet F-Ndet F-Interrupt*) *metis*
next
show $\langle s \in \mathcal{T} (P1 \sqcap P2) \wedge tickFree s \wedge [tick] \in \mathcal{T} Q \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?rhs \rangle$
by (*simp add: T-Ndet F-Ndet F-Interrupt*) (*metis Diff-insert-absorb*)
qed
next
have $\langle (s, X) \in \mathcal{F} (P1 \triangle Q) \implies (s, X) \in \mathcal{F} (P1 \sqcap P2 \triangle Q) \rangle$ **for** $s X P1 P2$
by (*simp add: F-Interrupt F-Ndet D-Ndet T-Ndet*) *blast*
thus $\langle (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} ?lhs \rangle$ **for** $s X$
by (*simp add: F-Ndet*) (*metis Ndet-commute*)
qed

lemma *Interrupt-assoc*: $\langle P \triangle (Q \triangle R) = P \triangle Q \triangle R \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
proof –
have $\langle ?lhs = ?rhs \rangle$ **if** *non-BOT* : $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle R \neq \perp \rangle$
proof (*subst Process-eq-spec-optimized, safe*)
fix s
assume $\langle s \in \mathcal{D} ?lhs \rangle$
then consider $\langle s \in \mathcal{D} P \rangle$
| $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D} (Q \triangle R) \rangle$
by (*simp add: D-Interrupt*) *blast*
thus $\langle s \in \mathcal{D} ?rhs \rangle$
proof cases
show $\langle s \in \mathcal{D} P \implies s \in \mathcal{D} ?rhs \rangle$ **by** (*simp add: D-Interrupt*)
next
assume $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D} (Q \triangle R) \rangle$
then obtain $t1\ t2$ **where** $*$: $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T} P \rangle$
 $\langle tickFree\ t1 \rangle \langle t2 \in \mathcal{D} (Q \triangle R) \rangle$ **by** *blast*
from $*(4)$ **consider** $\langle t2 \in \mathcal{D} Q \rangle$
| $\langle \exists u1\ u2. t2 = u1 @ u2 \wedge u1 \in \mathcal{T} Q \wedge tickFree\ u1 \wedge u2 \in \mathcal{D} R \rangle$
by (*simp add: D-Interrupt*) *blast*
thus $\langle s \in \mathcal{D} ?rhs \rangle$
proof cases
from $*(1, 2, 3)$ **show** $\langle t2 \in \mathcal{D} Q \implies s \in \mathcal{D} ?rhs \rangle$ **by** (*simp add: D-Interrupt*)
blast
next
show $\langle \exists u1\ u2. t2 = u1 @ u2 \wedge u1 \in \mathcal{T} Q \wedge tickFree\ u1 \wedge u2 \in \mathcal{D} R \implies s \in \mathcal{D} ?rhs \rangle$
by (*simp add: *(1) D-Interrupt T-Interrupt*)
(*metis *(2, 3) append-assoc tickFree-append*)
qed
qed
next
fix s
assume $\langle s \in \mathcal{D} ?rhs \rangle$
then consider $\langle s \in \mathcal{D} (P \triangle Q) \rangle$
| $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (P \triangle Q) \wedge tickFree\ t1 \wedge t2 \in \mathcal{D} R \rangle$
by (*simp add: D-Interrupt*) *blast*
thus $\langle s \in \mathcal{D} ?lhs \rangle$
proof cases
show $\langle s \in \mathcal{D} (P \triangle Q) \implies s \in \mathcal{D} ?lhs \rangle$ **by** (*simp add: D-Interrupt*) *blast*
next
assume $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (P \triangle Q) \wedge tickFree\ t1 \wedge t2 \in \mathcal{D} R \rangle$
then obtain $t1\ t2$ **where** $*$: $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T} (P \triangle Q) \rangle$
 $\langle tickFree\ t1 \rangle \langle t2 \in \mathcal{D} R \rangle$ **by** *blast*
from $*(2)$ **consider** $\langle t1 \in \mathcal{T} P \rangle$
| $\langle \exists u1\ u2. t1 = u1 @ u2 \wedge u1 \in \mathcal{T} P \wedge tickFree\ u1 \wedge u2 \in \mathcal{T} Q \rangle$
by (*simp add: T-Interrupt*) *blast*
thus $\langle s \in \mathcal{D} ?lhs \rangle$
proof cases

```

    show  $\langle t1 \in \mathcal{T} P \implies s \in \mathcal{D} ?lhs \rangle$ 
      by (simp add: D-Interrupt *(1))
         (metis *(3, 4) Nil-lem-T append-Nil tickFree-Nil)
  next
    show  $\langle \exists u1 u2. t1 = u1 @ u2 \wedge u1 \in \mathcal{T} P \wedge tickFree u1 \wedge u2 \in \mathcal{T} Q \implies$ 
 $s \in \mathcal{D} ?lhs \rangle$ 
      by (simp add: D-Interrupt *(1))
         (metis *(3, 4) append.assoc tickFree-append)
  qed
next
fix s X
assume same-div:  $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$ 
assume  $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ 
then consider  $\langle s \in \mathcal{D} ?lhs \rangle$ 
  |  $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} P \rangle$ 
  |  $\langle s @ [tick] \in \mathcal{T} P \wedge tick \notin X \rangle$ 
  |  $\langle (s, X) \in \mathcal{F} P \wedge tickFree s \wedge ([], X) \in \mathcal{F} (Q \Delta R) \rangle$ 
  |  $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} (Q \Delta R) \wedge$ 
 $t2 \neq [] \rangle$ 
  |  $\langle s \in \mathcal{T} P \wedge tickFree s \wedge [tick] \in \mathcal{T} (Q \Delta R) \wedge tick \notin X \rangle$ 
  by (subst (asm) F-Interrupt, simp add: D-Interrupt) blast
thus  $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ 
proof cases
  from same-div D-F show  $\langle s \in \mathcal{D} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$  by blast
next
  show  $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} P \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
    by (auto simp add: F-Interrupt T-Interrupt)
next
  show  $\langle s @ [tick] \in \mathcal{T} P \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
    by (simp add: F-Interrupt T-Interrupt) (metis Diff-insert-absorb)
next
  assume assm :  $\langle (s, X) \in \mathcal{F} P \wedge tickFree s \wedge ([], X) \in \mathcal{F} (Q \Delta R) \rangle$ 
  with non-BOT(2, 3) consider  $\langle [tick] \in \mathcal{T} Q \wedge tick \notin X \rangle$ 
  |  $\langle ([], X) \in \mathcal{F} Q \wedge ([], X) \in \mathcal{F} R \rangle$ 
  |  $\langle [] \in \mathcal{T} Q \wedge [tick] \in \mathcal{T} R \wedge tick \notin X \rangle$ 
  by (simp add: F-Interrupt Nil-lem-T BOT-iff-D) blast
  thus  $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ 
proof cases
  show  $\langle [tick] \in \mathcal{T} Q \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
    by (simp add: F-Interrupt T-Interrupt) (metis Diff-insert-absorb F-T assm)
next
  show  $\langle ([], X) \in \mathcal{F} Q \wedge ([], X) \in \mathcal{F} R \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
    by (simp add: F-Interrupt assm)
next
  show  $\langle [] \in \mathcal{T} Q \wedge [tick] \in \mathcal{T} R \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
    by (simp add: F-Interrupt T-Interrupt) (metis Diff-insert-absorb F-T assm)
qed
next

```

assume $\langle \exists t1\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge (t2, X) \in \mathcal{F}\ (Q \triangle R) \wedge t2 \neq [] \rangle$
then obtain $t1\ t2$ **where** $*$: $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T}\ P \rangle \langle tickFree\ t1 \rangle \langle (t2, X) \in \mathcal{F}\ (Q \triangle R) \rangle \langle t2 \neq [] \rangle$ **by** *blast*
from $*(4)$ **consider** $\langle t2 \in \mathcal{D}\ (Q \triangle R) \rangle$
| $\langle \exists u1. t2 = u1 @ [tick] \wedge u1 @ [tick] \in \mathcal{T}\ Q \rangle$
| $\langle t2 @ [tick] \in \mathcal{T}\ Q \wedge tick \notin X \rangle$
| $\langle (t2, X) \in \mathcal{F}\ Q \wedge tickFree\ t2 \wedge ([], X) \in \mathcal{F}\ R \rangle$
| $\langle \exists u1\ u2. t2 = u1 @ u2 \wedge u1 \in \mathcal{T}\ Q \wedge tickFree\ u1 \wedge (u2, X) \in \mathcal{F}\ R \wedge u2 \neq [] \rangle$
| $\langle t2 \in \mathcal{T}\ Q \wedge tickFree\ t2 \wedge [tick] \in \mathcal{T}\ R \wedge tick \notin X \rangle$
by (*simp add: F-Interrupt D-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
proof cases
assume $\langle t2 \in \mathcal{D}\ (Q \triangle R) \rangle$
with $*(1, 2, 3)$ **have** $\langle s \in \mathcal{D}\ ?lhs \rangle$ **by** (*simp add: D-Interrupt*) *blast*
with *same-div D-F* **show** $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$ **by** *blast*
next
from $*(1, 2, 3)$ **show** $\langle \exists u1. t2 = u1 @ [tick] \wedge u1 @ [tick] \in \mathcal{T}\ Q \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
by (*simp add: F-Interrupt T-Interrupt*) (*metis append-assoc*)
next
from $*(1, 2, 3)$ **show** $\langle t2 @ [tick] \in \mathcal{T}\ Q \wedge tick \notin X \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
by (*simp add: F-Interrupt T-Interrupt*) (*metis Diff-insert-absorb*)
next
from $*(1)$ **show** $\langle (t2, X) \in \mathcal{F}\ Q \wedge tickFree\ t2 \wedge ([], X) \in \mathcal{F}\ R \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
by (*simp add: F-Interrupt T-Interrupt*) (*metis *(2, 3, 5)*)
next
from $*(1, 2, 3)$ **show** $\langle \exists u1\ u2. t2 = u1 @ u2 \wedge u1 \in \mathcal{T}\ Q \wedge tickFree\ u1 \wedge (u2, X) \in \mathcal{F}\ R \wedge u2 \neq [] \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
by (*simp add: F-Interrupt T-Interrupt*)
(*metis (mono-tags, lifting) append-assoc tickFree-append*)
next
from $*(1, 2, 3)$ **show** $\langle t2 \in \mathcal{T}\ Q \wedge tickFree\ t2 \wedge [tick] \in \mathcal{T}\ R \wedge tick \notin X \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
by (*simp add: F-Interrupt T-Interrupt*) (*metis Diff-insert-absorb*)
qed
next
show $\langle s \in \mathcal{T}\ P \wedge tickFree\ s \wedge [tick] \in \mathcal{T}\ (Q \triangle R) \wedge tick \notin X \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$
by (*simp add: F-Interrupt T-Interrupt*)
(*metis Diff-insert-absorb append-Nil hd-append2 hd-in-set list.sel(1) tickFree-def*)
qed
next
fix $s\ X$
assume *same-div* : $\langle \mathcal{D}\ ?lhs = \mathcal{D}\ ?rhs \rangle$

assume $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
then consider $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$
| $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (P \Delta Q) \rangle$
| $\langle s @ [tick] \in \mathcal{T} (P \Delta Q) \wedge tick \notin X \rangle$
| $\langle (s, X) \in \mathcal{F} (P \Delta Q) \wedge tickFree s \wedge ([], X) \in \mathcal{F} R \rangle$
| $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (P \Delta Q) \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} R \wedge t2 \neq [] \rangle$
| $\langle s \in \mathcal{T} (P \Delta Q) \wedge tickFree s \wedge [tick] \in \mathcal{T} R \wedge tick \notin X \rangle$
by (*subst asm F-Interrupt, simp add: D-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof cases
from *same-div D-F show* $\langle s \in \mathcal{D} \text{ ?rhs} \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ **by** *blast*
next
show $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (P \Delta Q) \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by (*simp add: F-Interrupt T-Interrupt*)
(*metis last-append self-append-conv snoc-eq-iff-butlast*)
next
assume *assm* : $\langle s @ [tick] \in \mathcal{T} (P \Delta Q) \wedge tick \notin X \rangle$
from *assm[THEN conjunct1]* **consider** $\langle s @ [tick] \in \mathcal{T} P \rangle$
| $\langle \exists t1 t2. s @ [tick] = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{T} Q \rangle$
by (*simp add: T-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof cases
show $\langle s @ [tick] \in \mathcal{T} P \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by (*simp add: F-Interrupt*) (*metis Diff-insert-absorb assm[THEN conjunct2]*)
next
show $\langle \exists t1 t2. s @ [tick] = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{T} Q \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by (*simp add: F-Interrupt T-Interrupt*)
(*metis Diff-insert-absorb T-nonTickFree-imp-decomp append.right-neutral append-Nil assm[THEN conjunct2] butlast.simps(2) butlast-append non-tickFree-tick tickFree-append*)
qed
next
assume *assm* : $\langle (s, X) \in \mathcal{F} (P \Delta Q) \wedge tickFree s \wedge ([], X) \in \mathcal{F} R \rangle$
from *assm[THEN conjunct1]* **consider** $\langle s \in \mathcal{D} (P \Delta Q) \rangle$
| $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} P \rangle$
| $\langle s @ [tick] \in \mathcal{T} P \wedge tick \notin X \rangle$
| $\langle (s, X) \in \mathcal{F} P \wedge tickFree s \wedge ([], X) \in \mathcal{F} Q \rangle$
| $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \rangle$
| $\langle s \in \mathcal{T} P \wedge tickFree s \wedge [tick] \in \mathcal{T} Q \wedge tick \notin X \rangle$
by (*simp add: F-Interrupt D-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof cases
assume $\langle s \in \mathcal{D} (P \Delta Q) \rangle$
hence $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$ **by** (*simp add: D-Interrupt*)
with *same-div D-F show* $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ **by** *blast*

```

next
  show  $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} P \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
  by (simp add: F-Interrupt)
next
  show  $\langle s @ [tick] \in \mathcal{T} P \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
  by (simp add: F-Interrupt) (metis Diff-insert-absorb)
next
  show  $\langle (s, X) \in \mathcal{F} P \wedge tickFree s \wedge ([], X) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
  by (simp add: F-Interrupt assem[THEN conjunct2])
next
  show  $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
  by (simp add: F-Interrupt) (metis assem[THEN conjunct2] tickFree-append)
next
  show  $\langle s \in \mathcal{T} P \wedge tickFree s \wedge [tick] \in \mathcal{T} Q \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
  by (simp add: F-Interrupt T-Interrupt) (metis Diff-insert-absorb)
qed
next
  assume  $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (P \Delta Q) \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} R \wedge t2 \neq [] \rangle$ 
  then obtain  $t1 t2$  where  $*$  :  $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T} (P \Delta Q) \rangle \langle tickFree t1 \rangle \langle (t2, X) \in \mathcal{F} R \rangle \langle t2 \neq [] \rangle$  by blast
  from  $*(2)$  consider  $\langle t1 \in \mathcal{T} P \rangle$ 
  |  $\langle \exists u1 u2. t1 = u1 @ u2 \wedge u1 \in \mathcal{T} P \wedge tickFree u1 \wedge u2 \in \mathcal{T} Q \rangle$ 
  by (simp add: T-Interrupt) blast
  thus  $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ 
  proof cases
    from  $*(1, 3, 4, 5)$  show  $\langle t1 \in \mathcal{T} P \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
    by (simp add: F-Interrupt T-Interrupt)
    (metis Nil-elem-T append-Nil tickFree-Nil)
  next
    from  $*(1, 3, 4, 5)$  show  $\langle \exists u1 u2. t1 = u1 @ u2 \wedge u1 \in \mathcal{T} P \wedge tickFree u1 \wedge u2 \in \mathcal{T} Q \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
    by (elim exE, simp add: F-Interrupt) (metis append-is-Nil-conv)
  qed
  next
  show  $\langle s \in \mathcal{T} (P \Delta Q) \wedge tickFree s \wedge [tick] \in \mathcal{T} R \wedge tick \notin X \implies (s, X) \in \mathcal{F} ?lhs \rangle$ 
  by (simp add: F-Interrupt T-Interrupt)
  (metis Diff-insert-absorb Nil-elem-T append.right-neutral append-Nil tickFree-append)
  qed
  qed
  thus  $\langle ?lhs = ?rhs \rangle$ 
  apply (cases  $\langle P = \perp \rangle$ , solves  $\langle simp add: Interrupt-BOT-absorb \rangle$ )
  apply (cases  $\langle Q = \perp \rangle$ , solves  $\langle simp add: Interrupt-BOT-absorb \rangle$ )
  apply (cases  $\langle R = \perp \rangle$ , solves  $\langle simp add: Interrupt-BOT-absorb \rangle$ )

```

by *blast*
qed

2.3.5 Key Property

lemma *Interrupt-Mprefix*:

$\langle (\Box a \in A \rightarrow P a) \Delta Q = Q \Box (\Box a \in A \rightarrow P a \Delta Q) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)
proof (*subst Process-eq-spec-optimized, safe*)
fix s
assume $\langle s \in \mathcal{D} ?lhs \rangle$
then consider $\langle s \in \mathcal{D} (\Box a \in A \rightarrow P a) \rangle$
 $| \langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (\Box a \in A \rightarrow P a) \wedge tickFree t1 \wedge t2 \in \mathcal{D} Q \rangle$
by (*simp add: D-Interrupt*) *blast*
thus $\langle s \in \mathcal{D} ?rhs \rangle$
proof cases
show $\langle s \in \mathcal{D} (\Box a \in A \rightarrow P a) \implies s \in \mathcal{D} ?rhs \rangle$
by (*simp add: D-Det D-Mprefix D-Interrupt*) *blast*
next
assume $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (\Box a \in A \rightarrow P a) \wedge tickFree t1 \wedge t2 \in \mathcal{D} Q \rangle$
then obtain $t1 t2$ **where** $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T} (\Box a \in A \rightarrow P a) \rangle$
 $\langle tickFree t1 \rangle \langle t2 \in \mathcal{D} Q \rangle$ **by** *blast*
thus $\langle s \in \mathcal{D} ?rhs \rangle$
by (*simp add: D-Det D-Mprefix T-Mprefix D-Interrupt*)
(metis (no-types, opaque-lifting) hd-append2 imageI self-append-conv2 tickFree-tl tl-append2)
qed
next
fix s
assume $\langle s \in \mathcal{D} ?rhs \rangle$
then consider $\langle s \in \mathcal{D} Q \rangle | \langle \exists a s'. s = ev a \# s' \wedge a \in A \wedge s' \in \mathcal{D} (P a \Delta Q) \rangle$
by (*simp add: D-Det D-Mprefix image-iff*) (*metis event.inject list.exhaust-sel*)
thus $\langle s \in \mathcal{D} ?lhs \rangle$
proof cases
show $\langle s \in \mathcal{D} Q \implies s \in \mathcal{D} ?lhs \rangle$
apply (*simp add: D-Interrupt*)
using *Nil-elim-T tickFree-Nil* **by** *blast*
next
assume $\langle \exists a s'. s = ev a \# s' \wedge a \in A \wedge s' \in \mathcal{D} (P a \Delta Q) \rangle$
then obtain $a s'$
where $*$: $\langle s = ev a \# s' \rangle \langle a \in A \rangle$
 $\langle s' \in \mathcal{D} (P a) \vee (\exists t1 t2. s' = t1 @ t2 \wedge t1 \in \mathcal{T} (P a) \wedge tickFree t1 \wedge t2 \in \mathcal{D} Q) \rangle$
by (*simp add: D-Interrupt*) *blast*
from $*(3)$ **show** $\langle s \in \mathcal{D} ?lhs \rangle$
apply (*elim disjE exE*)
subgoal by (*solves simp add: *(1, 2) D-Interrupt D-Mprefix image-iff*)
by (*simp add: *(1) D-Interrupt T-Mprefix*)
*(metis *(2) Cons-eq-appendI event.distinct(1) list.sel(1, 3) tickFree-Cons)*

qed
next
fix $s X$
assume $\text{same-div} : \langle \mathcal{D} \text{ ?lhs} = \mathcal{D} \text{ ?rhs} \rangle$
assume $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
then consider $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$
| $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (Mprefix A P) \rangle$
| $\langle s @ [tick] \in \mathcal{T} (Mprefix A P) \wedge tick \notin X \rangle$
| $\langle (s, X) \in \mathcal{F} (Mprefix A P) \wedge tickFree s \wedge ([], X) \in \mathcal{F} Q \rangle$
| $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (Mprefix A P) \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \rangle$
| $\langle s \in \mathcal{T} (Mprefix A P) \wedge tickFree s \wedge [tick] \in \mathcal{T} Q \wedge tick \notin X \rangle$
by (*simp add: F-Interrupt D-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
proof cases
from $D\text{-}F$ *same-div* **show** $\langle s \in \mathcal{D} \text{ ?lhs} \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ **by** *blast*
next
show $\langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (Mprefix A P) \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
by (*elim exE, simp add: F-Det T-Mprefix F-Mprefix F-Interrupt image-iff*)
(*metis event.distinct(1) hd-append list.sel(1) tl-append2*)
next
show $\langle s @ [tick] \in \mathcal{T} (Mprefix A P) \wedge tick \notin X \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
by (*simp add: F-Det T-Mprefix F-Mprefix F-Interrupt image-iff*)
(*metis Diff-insert-absorb event.simps(3) hd-append list.sel(1) tl-append-if*)
next
show $\langle (s, X) \in \mathcal{F} (Mprefix A P) \wedge tickFree s \wedge ([], X) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
by (*simp add: F-Det F-Mprefix F-Interrupt image-iff*) (*metis tickFree-tl*)
next
show $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (Mprefix A P) \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
by (*elim exE, simp add: F-Det T-Mprefix F-Mprefix F-Interrupt image-iff*)
(*metis hd-append2 self-append-conv2 tickFree-tl tl-append2*)
next
show $\langle s \in \mathcal{T} (Mprefix A P) \wedge tickFree s \wedge [tick] \in \mathcal{T} Q \wedge tick \notin X \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
by (*simp add: F-Det T-Mprefix F-Mprefix F-Interrupt image-iff*)
(*metis Diff-insert-absorb tickFree-tl*)
qed
next
fix $s X$
assume $\text{same-div} : \langle \mathcal{D} \text{ ?lhs} = \mathcal{D} \text{ ?rhs} \rangle$
assume $\text{assm} : \langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof (*cases* $\langle s = [] \rangle$)
from assm **show** $\langle s = [] \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
by (*simp add: F-Det F-Mprefix F-Interrupt Nil-elem-T*) *blast*
next

assume $\langle s \neq [] \rangle$
with *assm* **consider** $\langle (s, X) \in \mathcal{F} Q \rangle$
 $| \langle \exists a s'. s = \text{ev } a \# s' \wedge a \in A \wedge (s', X) \in \mathcal{F} (P a \Delta Q) \rangle$
by (*simp add: F-Det F-Mprefix image-iff*) (*metis event.inject list.exhaust-sel*)
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof cases
show $\langle (s, X) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Interrupt*)
(*metis Nil-elem-T* $\langle s \neq [] \rangle$ *append-Nil tickFree-Nil*)
next
assume $\langle \exists a s'. s = \text{ev } a \# s' \wedge a \in A \wedge (s', X) \in \mathcal{F} (P a \Delta Q) \rangle$
then obtain $a s'$
where $*$: $\langle s = \text{ev } a \# s' \rangle \langle a \in A \rangle \langle (s', X) \in \mathcal{F} (P a \Delta Q) \rangle$ **by** *blast*
from $*(3)$ **consider** $\langle s' \in \mathcal{D} (P a \Delta Q) \rangle$
 $| \langle \exists t1. s' = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (P a) \rangle$
 $| \langle s' @ [tick] \in \mathcal{T} (P a) \wedge \text{tick} \notin X \rangle$
 $| \langle (s', X) \in \mathcal{F} (P a) \wedge \text{tickFree } s' \wedge ([], X) \in \mathcal{F} Q \rangle$
 $| \langle \exists t1 t2. s' = t1 @ t2 \wedge t1 \in \mathcal{T} (P a) \wedge \text{tickFree } t1 \wedge (t2, X) \in \mathcal{F} Q \wedge$
 $t2 \neq [] \rangle$
 $| \langle s' \in \mathcal{T} (P a) \wedge \text{tickFree } s' \wedge [tick] \in \mathcal{T} Q \wedge \text{tick} \notin X \rangle$
by (*simp add: F-Interrupt D-Interrupt*) *blast*
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof cases
assume $\langle s' \in \mathcal{D} (P a \Delta Q) \rangle$
hence $\langle s \in \mathcal{D} ?lhs \rangle$
by (*simp add: D-Interrupt D-Mprefix T-Mprefix *(1, 2) image-iff*)
(*metis *(2) append-Cons event.distinct(1) list.sel(1, 3) tickFree-Cons*)
with *D-F same-div* **show** $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ **by** *blast*
next
show $\langle \exists t1. s' = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} (P a) \implies (s, X) \in \mathcal{F} ?lhs \rangle$
by (*elim exE, simp add: *(1, 2) F-Interrupt T-Mprefix*)
next
show $\langle s' @ [tick] \in \mathcal{T} (P a) \wedge \text{tick} \notin X \implies (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: *(1, 2) F-Interrupt T-Mprefix*) *blast*
next
show $\langle (s', X) \in \mathcal{F} (P a) \wedge \text{tickFree } s' \wedge ([], X) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: *(1, 2) F-Interrupt F-Mprefix*)
next
show $\langle \exists t1 t2. s' = t1 @ t2 \wedge t1 \in \mathcal{T} (P a) \wedge \text{tickFree } t1 \wedge$
 $(t2, X) \in \mathcal{F} Q \wedge t2 \neq [] \implies (s, X) \in \mathcal{F} ?lhs \rangle$
by (*elim exE, simp add: F-Interrupt T-Mprefix *(1)*)
(*metis *(2) Cons-eq-appendI event.distinct(1) list.sel(1, 3) tickFree-Cons*)
next
show $\langle s' \in \mathcal{T} (P a) \wedge \text{tickFree } s' \wedge [tick] \in \mathcal{T} Q \wedge \text{tick} \notin X \implies (s, X) \in$
 $\mathcal{F} ?lhs \rangle$
by (*simp add: F-Interrupt T-Mprefix *(1, 2) image-iff*) *blast*
qed
qed
qed

qed

corollary $\langle (\Box a \in A \rightarrow P a) \Delta (\Box b \in B \rightarrow Q b) =$
 $\Box x \in A \cup B \rightarrow (\quad \text{if } x \in A \cap B \text{ then } (P x \Delta (\Box b \in B \rightarrow Q b)) \sqcap Q x$
 $\quad \text{else if } x \in A \text{ then } P x \Delta (\Box b \in B \rightarrow Q b)$
 $\quad \quad \quad \text{else } Q x \rangle$
apply (*subst Interrupt-Mprefix, subst Mprefix-Det-distr*)
by (*metis Det-commute Mprefix-Det-distr*)

2.3.6 Continuity

lemma *chain-left-Interrupt*: $\langle \text{chain } Y \Longrightarrow \text{chain } (\lambda i. Y i \Delta Q) \rangle$
by (*simp add: chain-def*)

lemma *chain-right-Interrupt*: $\langle \text{chain } Y \Longrightarrow \text{chain } (\lambda i. P \Delta Y i) \rangle$
by (*simp add: chain-def*)

lemma *cont-left-prem-Interrupt* : $\langle (\bigsqcup i. Y i) \Delta Q = (\bigsqcup i. Y i \Delta Q) \rangle$
(is $\langle ?lhs = ?rhs \rangle$ **if** *chain* : $\langle \text{chain } Y \rangle$

proof (*subst Process-eq-spec, safe*)

show $\langle s \in \mathcal{D} ?lhs \Longrightarrow s \in \mathcal{D} ?rhs \rangle$ **for** *s*

by (*simp add: limproc-is-thelub chain chain-left-Interrupt*
D-Interrupt T-LUB D-LUB) *blast*

next

fix *s*

define *S*

where $\langle S i \equiv \{t1. s = t1 \wedge t1 \in \mathcal{D} (Y i)\} \cup$
 $\{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} (Y i) \wedge \text{tickFree } t1 \wedge t2 \in \mathcal{D} Q\} \rangle$

for *i*

assume $\langle s \in \mathcal{D} ?rhs \rangle$

hence $\langle s \in \mathcal{D} (Y i \Delta Q) \rangle$ **for** *i*

by (*simp add: limproc-is-thelub D-LUB chain-left-Interrupt chain*)

hence $\langle \forall i. S i \neq \{\} \rangle$ **by** (*simp add: S-def D-Interrupt*) *blast*

moreover have $\langle \text{finite } (S 0) \rangle$

unfolding *S-def*

apply (*rule finite-subset[of - $\langle \{t1. \exists t2. s = t1 @ t2\} \rangle]$, *blast*)*

by (*metis prefixes-fin*)

moreover have $\langle \forall i. S (Suc i) \subseteq S i \rangle$

unfolding *S-def* **apply** (*intro allI Un-mono subsetI; simp*)

by (*metis in-mono le-approx1 po-class.chainE chain*)

(*metis le-approx-lemma-T po-class.chain-def subset-eq chain*)

ultimately have $\langle (\bigcap i. S i) \neq \{\} \rangle$

by (*rule Inter-nonempty-finite-chained-sets*)

then obtain *t1* **where** $\ast : \langle \forall i. t1 \in S i \rangle$

by (*meson INT-iff ex-in-conv iso-tuple-UNIV-I*)

show $\langle s \in \mathcal{D} ?lhs \rangle$

proof (*cases $\langle \forall i. s \in \mathcal{D} (Y i) \rangle$*)

```

    case True
  thus ⟨s ∈ D ?lhs⟩ by (simp add: D-Interrupt limproc-is-thelub D-LUB chain)
next
  case False
  with * obtain j t2 where **: ⟨s = t1 @ t2⟩ ⟨t1 ∈ T (Y j)⟩ ⟨tickFree t1⟩ ⟨t2
∈ D Q⟩
    by (simp add: S-def) blast
  from * D-T have ⟨∀ i. t1 ∈ T (Y i)⟩ by (simp add: S-def) blast
  with *(1, 3, 4) show ⟨s ∈ D ?lhs⟩
    by (simp add: D-Interrupt limproc-is-thelub T-LUB chain) blast
qed
next
show ⟨(s, X) ∈ F ?lhs ⟹ (s, X) ∈ F ?rhs⟩ for s X
  by (simp add: limproc-is-thelub chain chain-left-Interrupt
      F-Interrupt F-LUB T-LUB D-LUB) blast
next
fix s X
define S
  where ⟨S i ≡ {t1. s = t1 @ [tick] ∧ t1 @ [tick] ∈ T (Y i)} ∪
    {t1. s = t1 ∧ t1 @ [tick] ∈ T (Y i) ∧ tick ∉ X} ∪
    {t1. s = t1 ∧ (t1, X) ∈ F (Y i) ∧ tickFree t1 ∧ ([], X) ∈ F Q} ∪
    {t1. ∃ t2. s = t1 @ t2 ∧ t1 ∈ T (Y i) ∧ tickFree t1 ∧ (t2, X) ∈ F
Q ∧ t2 ≠ []} ∪
    {t1. s = t1 ∧ t1 ∈ T (Y i) ∧ tickFree t1 ∧ [tick] ∈ T Q ∧ tick ∉
X} ∪
    {t1. s = t1 ∧ t1 ∈ D (Y i)} ∪
    {t1. ∃ t2. s = t1 @ t2 ∧ t1 ∈ T (Y i) ∧ tickFree t1 ∧ t2 ∈ D Q}⟩
for i
  assume ⟨(s, X) ∈ F ?rhs⟩
  hence ⟨(s, X) ∈ F (Y i Δ Q)⟩ for i
    by (simp add: limproc-is-thelub F-LUB chain-left-Interrupt chain)
  hence ⟨∀ i. S i ≠ {}⟩ by (simp add: S-def F-Interrupt, safe; simp, blast)
  moreover have ⟨finite (S 0)⟩
    unfolding S-def
    apply (intro finite-UnI)
    apply (all ⟨rule finite-subset[of - ⟨{t1. ∃ t2. s = t1 @ t2}⟩], blast⟩)
    by (metis prefixes-fin)+
  moreover have ⟨∀ i. S (Suc i) ⊆ S i⟩
    unfolding S-def apply (intro allI Un-mono subsetI; simp)
    subgoal using D-T le-approx2T po-class.chain-def chain by blast
    subgoal using D-T le-approx2T po-class.chain-def chain by blast
    subgoal using is-processT8-S le-approx2 po-class.chainE chain by blast
    subgoal by (metis NT-ND le-approx2T po-class.chainE chain)
    subgoal using D-T le-approx2T po-class.chain-def chain by blast
    subgoal by (meson in-mono le-approx1 po-class.chainE chain)
    by (metis NT-ND le-approx2T po-class.chainE chain)
  ultimately have ⟨(∩ i. S i) ≠ {}⟩
    by (rule Inter-nonempty-finite-chained-sets)
  then obtain t1 where *: ⟨∀ i. t1 ∈ S i⟩

```

by (*meson INT-iff ex-in-conv iso-tuple-UNIV-I*)
show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof (*cases* $\langle \exists j \ t2. \ s = t1 \ @ \ t2 \wedge t1 \in \mathcal{T} (Y \ j) \wedge tickFree \ t1 \wedge t2 \in \mathcal{D} \ Q \rangle$)
 case *True1 : True*
 then obtain $j \ t2$ **where** $** : \langle s = t1 \ @ \ t2 \rangle \langle t1 \in \mathcal{T} (Y \ j) \rangle$
 $\langle tickFree \ t1 \rangle \langle t2 \in \mathcal{D} \ Q \rangle$ **by** *blast*
 from $* \ F\text{-}T \ NT\text{-}ND \ is\text{-}processT3\text{-}ST$ **have** $\langle \forall i. \ t1 \in \mathcal{T} (Y \ i) \rangle$
 by (*simp add: S-def*) *blast*
 with $**(1, 3, 4)$ **show** $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 by (*simp add: F-Interrupt limproc-is-thelub chain T-LUB*) *blast*
 next
 case *False1 : False*
 show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 proof (*cases* $\langle \exists j \ t2. \ s = t1 \ @ \ t2 \wedge t1 \in \mathcal{T} (Y \ j) \wedge tickFree \ t1 \wedge$
 $(t2, X) \in \mathcal{F} \ Q \wedge t2 \neq [] \rangle$)
 case *True2 : True*
 then obtain $j \ t2$ **where** $** : \langle s = t1 \ @ \ t2 \rangle \langle t1 \in \mathcal{T} (Y \ j) \rangle \langle tickFree \ t1 \rangle$
 $\langle (t2, X) \in \mathcal{F} \ Q \rangle \langle t2 \neq [] \rangle$ **by** *blast*
 from $* \ F\text{-}T \ NT\text{-}ND \ is\text{-}processT3\text{-}ST$ **have** $\langle \forall i. \ t1 \in \mathcal{T} (Y \ i) \rangle$
 by (*simp add: S-def*) *blast*
 with $**(1, 3, 4, 5)$ *False1* **show** $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 by (*simp add: F-Interrupt limproc-is-thelub chain T-LUB*) *blast*
 next
 case *False2: False*
 show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 proof (*cases* $\langle \exists j. \ s = t1 \wedge t1 \in \mathcal{T} (Y \ j) \wedge tickFree \ t1 \wedge [tick] \in \mathcal{T} \ Q \wedge tick$
 $\notin X \rangle$)
 case *True3 : True*
 then obtain j **where** $** : \langle s = t1 \rangle \langle t1 \in \mathcal{T} (Y \ j) \rangle \langle tickFree \ t1 \rangle$
 $\langle [tick] \in \mathcal{T} \ Q \rangle \langle tick \notin X \rangle$ **by** *blast*
 from $* \ F\text{-}T \ NT\text{-}ND \ is\text{-}processT3\text{-}ST$ **have** $\langle \forall i. \ t1 \in \mathcal{T} (Y \ i) \rangle$
 by (*simp add: S-def*) *blast*
 with $**(1, 3, 4, 5)$ *False1* **show** $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 by (*simp add: F-Interrupt limproc-is-thelub chain T-LUB*) *blast*
 next
 case *False3: False*
 show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 proof (*cases* $\langle \exists j. \ s = t1 \wedge (t1, X) \in \mathcal{F} (Y \ j) \wedge tickFree \ t1 \wedge ([], X) \in \mathcal{F}$
 $Q \rangle$)
 case *True4 : True*
 then obtain j **where** $** : \langle s = t1 \rangle \langle (t1, X) \in \mathcal{F} (Y \ j) \rangle \langle tickFree \ t1 \rangle \langle ([],$
 $X) \in \mathcal{F} \ Q \rangle$ **by** *blast*
 have $\langle (t1, X) \in \mathcal{F} (Y \ i) \rangle$ **for** i
 using $*[rule\text{-}format, \ of \ i]$ **apply** (*simp add: S-def* $**(1)$, *elim disjE*)
 subgoal by (*solves* *simp add: T-F-spec is-processT6-S1*)
 subgoal by (*solves* *simp add: T-F-spec is-processT6-S1*)
 subgoal using $**(1)$ *False3* **by** *blast*
 subgoal by (*solves* *simp add: is-processT8*)
 using $**(1)$ *False1* **by** *blast*


```

with  $** (1, 3, 4)$  show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
  by (simp add: F-Interrupt limproc-is-thelub chain F-LUB)
next
case False4: False
show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
proof (cases  $\langle \exists j. s = t1 \ @ \ [tick] \wedge t1 \ @ \ [tick] \in \mathcal{T} (Y j) \rangle$ )
  case True5 : True
  then obtain j where  $** : \langle s = t1 \ @ \ [tick] \rangle \langle t1 \ @ \ [tick] \in \mathcal{T} (Y j) \rangle$  by
blast
  from * False1 False2 have  $\langle \forall i. t1 \ @ \ [tick] \in \mathcal{T} (Y i) \rangle$ 
    by (auto simp add: S-def  $** (1)$ )
  with  $** (1)$  show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Interrupt limproc-is-thelub chain T-LUB)
next
case False5 : False
show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
proof (cases  $\langle \exists j. s = t1 \wedge t1 \ @ \ [tick] \in \mathcal{T} (Y j) \wedge tick \notin X \rangle$ )
  case True6 : True
  then obtain j where  $** : \langle s = t1 \rangle \langle t1 \ @ \ [tick] \in \mathcal{T} (Y j) \rangle \langle tick \notin X \rangle$ 
by blast
  have  $\langle t1 \ @ \ [tick] \in \mathcal{T} (Y i) \rangle$  for i
    using  $*[rule-format, of i]$  apply (simp add: S-def  $** (1)$ , elim disjE)
  subgoal by (solves simp)
  subgoal using  $** (1)$  False4 by blast
  subgoal using  $** (1)$  False3 by blast
  subgoal using  $** (2)$  D-T front-tickFree-mono is-processT2-TR
is-processT7 by blast
  subgoal using  $** (1)$  False1 by blast
  done
  with  $** (1, 3)$  show  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Interrupt limproc-is-thelub chain T-LUB)
    (metis Diff-insert-absorb)
next
case False6: False
have  $\langle s = t1 \wedge t1 \in \mathcal{D} (Y i) \rangle$  for i
  using  $*[rule-format, of i]$  apply (simp add: S-def, elim disjE)
  subgoal using False5 by blast
  subgoal using False6 by blast
  subgoal using False4 by blast
  subgoal using False2 by blast
  subgoal using False3 by blast
  subgoal by (solves simp)
  subgoal using False1 by blast
  done
  thus  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Interrupt limproc-is-thelub chain D-LUB)
qed
qed
qed

```

qed
 qed
 qed
 qed

lemma *cont-right-prem-Interrupt* : $\langle P \Delta (\bigsqcup i. Y i) = (\bigsqcup i. P \Delta Y i) \rangle$
 (is $\langle ?lhs = ?rhs \rangle$) **if** *chain* : $\langle chain Y \rangle$

proof (*subst Process-eq-spec, safe*)

show $\langle s \in \mathcal{D} ?lhs \implies s \in \mathcal{D} ?rhs \rangle$ **for** *s*

by (*simp add: limproc-is-thelub chain chain-right-Interrupt D-Interrupt D-LUB*) *blast*

next

fix *s*

define *S* **where** $\langle S i \equiv \{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{D} (Y i)\} \rangle$ **for** *i*

assume *assm* : $\langle s \in \mathcal{D} ?rhs \rangle$

show $\langle s \in \mathcal{D} ?lhs \rangle$

proof (*cases* $\langle s \in \mathcal{D} P \rangle$)

show $\langle s \in \mathcal{D} P \implies s \in \mathcal{D} ?lhs \rangle$ **by** (*simp add: D-Interrupt*)

next

assume $\langle s \notin \mathcal{D} P \rangle$

with *assm* **have** $\langle \forall i. S i \neq \{\} \rangle$

by (*simp add: limproc-is-thelub chain chain-right-Interrupt S-def D-Interrupt D-LUB*) *blast*

moreover **have** $\langle finite (S 0) \rangle$

unfolding *S-def*

apply (*rule finite-subset[of - $\langle \{t1. \exists t2. s = t1 @ t2\} \rangle]$, *blast*)*

by (*metis prefixes-fin*)

moreover **have** $\langle \forall i. S (Suc i) \subseteq S i \rangle$

unfolding *S-def* **apply** (*intro allI Un-mono subsetI; simp*)

by (*metis in-mono le-approx1 po-class.chainE chain*)

ultimately **have** $\langle (\bigcap i. S i) \neq \{\} \rangle$

by (*rule Inter-nonempty-finite-chained-sets*)

then **obtain** *t1* **where** $\ast : \langle \forall i. t1 \in S i \rangle$

by (*meson INT-iff ex-in-conv iso-tuple-UNIV-I*)

thus $\langle s \in \mathcal{D} ?lhs \rangle$

by (*simp add: D-Interrupt limproc-is-thelub chain D-LUB S-def*) *blast*

qed

next

show $\langle (s, X) \in \mathcal{F} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$ **for** *s X*

by (*simp add: limproc-is-thelub chain chain-right-Interrupt F-Interrupt F-LUB T-LUB D-LUB*) *blast*

next

fix *s X*

define *S*

where $\langle S i \equiv \{t1. s = t1 \wedge (t1, X) \in \mathcal{F} P \wedge tickFree t1 \wedge (\bigsqcup, X) \in \mathcal{F} (Y i)\} \rangle$

\cup

$\langle t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge (t2, X) \in \mathcal{F} (Y$

$i) \wedge t2 \neq [] \} \cup$
 $\{t1. s = t1 \wedge t1 \in \mathcal{T} P \wedge tickFree\ t1 \wedge [tick] \in \mathcal{T} (Y\ i) \wedge tick \notin X\} \cup$
 $\{t1. \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D} (Y\ i)\}$

for i
assume $assm : \langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
show $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
proof ($cases\ \langle \exists t1. s = t1 @ [tick] \wedge t1 @ [tick] \in \mathcal{T} P \vee$
 $s = t1 \wedge t1 @ [tick] \in \mathcal{T} P \wedge tick \notin X \vee$
 $s = t1 \wedge t1 \in \mathcal{D} P \rangle$)

case $True$
thus $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
by ($simp\ add: F-Interrupt$) ($metis\ Diff-insert-absorb$)

next
case $False$
with $assm\ have\ \langle \forall i. S\ i \neq \{\} \rangle$
by ($simp\ add: limproc-is-thelub\ chain\ chain-right-Interrupt$
 $S-def\ F-Interrupt\ F-LUB$) $blast$

moreover $have\ \langle finite\ (S\ 0) \rangle$
unfolding $S-def$
apply ($rule\ finite-subset[of\ -\ \langle \{t1. \exists t2. s = t1 @ t2\} \rangle]$, $blast$)
by ($metis\ prefixes-fin$)

moreover $have\ \langle \forall i. S\ (Suc\ i) \subseteq S\ i \rangle$
unfolding $S-def$ **apply** ($intro\ allI\ Un-mono\ subsetI$; $simp$)
subgoal **by** ($meson\ le-approx2\ po-class.chainE\ process-charn\ chain$)
subgoal **by** ($metis\ le-approx2\ po-class.chainE\ process-charn\ chain$)
subgoal **by** ($metis\ insert-absorb\ insert-subset\ le-approx-lemma-T\ po-class.chainE$
 $chain$)

by ($metis\ in-mono\ le-approx1\ po-class.chainE\ chain$)
ultimately $have\ \langle (\bigcap i. S\ i) \neq \{\} \rangle$
by ($rule\ Inter-nonempty-finite-chained-sets$)

then **obtain** $t1$ **where** $*$: $\langle \forall i. t1 \in S\ i \rangle$
by ($meson\ INT-iff\ ex-in-conv\ iso-tuple-UNIV-I$)

show $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
proof ($cases\ \langle \exists j\ t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree\ t1 \wedge$
 $(t2, X) \in \mathcal{F}\ (Y\ j) \wedge t2 \neq [] \rangle$)

case $True1 : True$
then **obtain** $j\ t2$
where $** : \langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{T} P \rangle \langle tickFree\ t1 \rangle$
 $\langle (t2, X) \in \mathcal{F}\ (Y\ j) \rangle \langle t2 \neq [] \rangle$ **by** $blast$

from $*$ $** (1, 5)$ $D-F$ **have** $\langle \forall i. (t2, X) \in \mathcal{F}\ (Y\ i) \rangle$ **by** ($simp\ add: S-def$)

$blast$

with $** (1, 2, 3, 5)$ **show** $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
by ($simp\ add: F-Interrupt\ limproc-is-thelub\ chain\ F-LUB$) $blast$

next
case $False1: False$
show $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
proof ($cases\ \langle \exists j. s = t1 \wedge (t1, X) \in \mathcal{F}\ P \wedge tickFree\ t1 \wedge ([], X) \in \mathcal{F}\ (Y$

$j\rangle\rangle$
case $True2 : True$
then obtain j **where** $** : \langle s = t1 \rangle \langle (t1, X) \in \mathcal{F} P \rangle$
 $\langle tickFree t1 \rangle \langle ([], X) \in \mathcal{F} (Y j) \rangle$ **by** $blast$
from $* ** (1)$ $D-F$ *is-processT6-S2* **have** $\langle \forall i. ([], X) \in \mathcal{F} (Y i) \rangle$
by (*simp add: S-def*) $blast$
with $** (1, 2, 3)$ **show** $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Interrupt limproc-is-thelub chain F-LUB*)
next
case $False2 : False$
show $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof (*cases* $\langle \exists j. s = t1 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge$
 $[tick] \in \mathcal{T} (Y j) \wedge tick \notin X \rangle$)
case $True3 : True$
then obtain j **where** $** : \langle s = t1 \rangle \langle t1 \in \mathcal{T} P \rangle \langle tickFree t1 \rangle$
 $\langle [tick] \in \mathcal{T} (Y j) \rangle \langle tick \notin X \rangle$ **by** $blast$
from $* ** (1)$ $False2$ **have** $\langle \forall i. [tick] \in \mathcal{T} (Y i) \rangle$
by (*simp add: S-def*)
(*metis BOT-iff-D D-UU NT-ND front-tickFree-single mem-Collect-eq*)
with $** (1, 2, 3, 5)$ **show** $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Interrupt limproc-is-thelub chain T-LUB*) $blast$
next
case $False3 : False$
have $\langle \exists t2. s = t1 @ t2 \wedge t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{D} (Y i) \rangle$ **for** i
using $*[rule-format, of i]$ **apply** (*simp add: S-def, elim disjE*)
using $False1 False2 False3$ **by** $blast+$

thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Interrupt limproc-is-thelub chain D-LUB*)
(*metis same-append-eq*)

qed
qed
qed
qed
qed

lemma *Interrupt-cont[simp]* :
assumes $cont-f : \langle cont f \rangle$ **and** $cont-g : \langle cont g \rangle$
shows $\langle cont (\lambda x. f x \Delta g x) \rangle$
proof –
have $* : \langle cont (\lambda y. y \Delta g x) \rangle$ **for** x
by (*rule contI2, rule monofunI, solves simp, simp add: cont-left-prem-Interrupt*)
have $\langle \bigwedge y. cont (Interrupt y) \rangle$
by (*simp add: contI2 cont-right-prem-Interrupt fun-belowD lub-fun monofunI*)
hence $** : \langle cont (\lambda x. y \Delta g x) \rangle$ **for** y
by (*rule cont-compose*) (*simp add: cont-g*)
show $?thesis$ **by** (*fact cont-apply[OF cont-f * **]*)
qed

end

Chapter 3

The Ready Set Notion

```
theory ReadySet
  imports Sliding Throw Interrupt
begin
```

This will be discussed more precisely later, but we want to define a new operator which would in some way be the reciprocal of the prefix operator $e \rightarrow P$.

A first observation is that by prefixing P with e , we force its traces to begin with $ev\ e$.

Therefore we must define a notion that captures this idea.

We start by giving a notation to *tick* to be closer to Roscoe's book [3] (and also closer to classic CSP literature).

```
notation tick ( $\langle\checkmark\rangle$ )
```

3.1 Definition

The ready set notion captures the set of events that can be used to begin a given process.

```
definition ready-set ::  $\langle\alpha\ process \Rightarrow \alpha\ event\ set\rangle$ 
  where  $\langle ready\text{-}set\ P \equiv \{a. [a] \in \mathcal{T}\ P\}\rangle$ 
```

```
lemma ready-set-def-bis:  $\langle ready\text{-}set\ P = \{e. \exists s. e \# s \in \mathcal{T}\ P\}\rangle$ 
and Cons-in-T-imp-elem-ready-set:  $\langle e \# s \in \mathcal{T}\ P \Longrightarrow e \in ready\text{-}set\ P\rangle$ 
unfolding ready-set-def using is-processT3-ST by force+
```

We say here that the *ready-set* of a process P is the set of events e such that there is a trace of P starting with e .

One could also think about defining *ready-set* P as the set of events that P can not refuse at first: $\{e. \{e\} \notin \mathcal{R}\ P\}$. These two definitions are not equivalent (and the second one is more restrictive than the first one). Moreover,

the second does not behave well with the non-deterministic choice (\sqcap) (see the **notepad** below).

Therefore, we will keep the first one.

We also have a strong argument of authority: this is the definition given by Roscoe (where it is called *initials*) [4, p.40].

```

notepad
begin
  fix  $e :: \langle 'a \text{ event} \rangle$  — just fixing  $'a$  type

  define bad-ready-set
    where  $\langle \text{bad-ready-set } (P :: 'a \text{ process}) \rangle \equiv \{e. \{e\} \notin \mathcal{R} P\}$  for  $P$ 

  have bad-ready-set-subset-ready-set:
     $\langle \text{bad-ready-set } P \subseteq \text{ready-set } P \rangle$  for  $P$ 
    unfolding bad-ready-set-def ready-set-def Refusals-iff
    using F-T is-processT5-S6 by blast

  have bad-behaviour-with-Ndet:
     $\langle \exists P Q. \text{bad-ready-set } (P \sqcap Q) \neq \text{bad-ready-set } P \cup \text{bad-ready-set } Q \rangle$ 
  proof (intro exI)
    show  $\langle \text{bad-ready-set } (\text{SKIP} \sqcap \perp) \neq \text{bad-ready-set } \text{SKIP} \cup \text{bad-ready-set } \perp \rangle$ 
    by (simp add: Ndet-BOT)
      (simp add: bad-ready-set-def F-Ndet F-SKIP F-UU Refusals-iff)
  qed
end

```

3.2 Anti-Mono Rules

lemma *anti-mono-ready-set-T*: $\langle P \sqsubseteq_T Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
by (*simp add: Collect-mono-iff trace-refine-def ready-set-def subsetD*)

lemma *anti-mono-ready-set-F*: $\langle P \sqsubseteq_F Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
unfolding *failure-refine-def*
by (*drule F-subset-imp-T-subset*) (*fact anti-mono-ready-set-T[unfolded trace-refine-def]*)

Of course, this anti-monotony does not hold for (\sqsubseteq_D).

lemma *anti-mono-ready-set-FD*: $\langle P \sqsubseteq_{FD} Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
by (*simp add: anti-mono-ready-set-F leFD-imp-leF*)

lemma *anti-mono-ready-set*: $\langle P \sqsubseteq Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
by (*simp add: anti-mono-ready-set-T le-approx-lemma-T trace-refine-def*)

lemma *anti-mono-ready-set-DT*: $\langle P \sqsubseteq_{DT} Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
by (*simp add: anti-mono-ready-set-T leDT-imp-leT*)

3.3 Behaviour of *ready-set* with *STOP*, *SKIP* and \perp

lemma *ready-set-STOP*: $\langle \text{ready-set } STOP = \{\} \rangle$
by (*simp add: ready-set-def T-STOP*)

We already had $(?P = STOP) = (\mathcal{T} ?P = \{\})$. As an immediate consequence we obtain a characterization of being *STOP* involving *ready-set*.

lemma *ready-set-empty-iff-STOP*: $\langle \text{ready-set } P = \{\} \longleftrightarrow P = STOP \rangle$

proof (*intro iffI*)

{ **assume** $\langle \mathcal{T} P \neq \{\} \rangle$

then obtain $a \ s$ **where** $\langle a \ \# \ s \in \mathcal{T} P \rangle$

by (*metis Nil-elim-T is-singleton-the-elim is-singletonI'*
list.exhaust-sel singletonI empty-iff)

hence $\langle \exists a. [a] \in \mathcal{T} P \rangle$ **by** (*metis append-Cons append-Nil is-processT3-ST*)

thus $\langle \text{ready-set } P = \{\} \implies P = STOP \rangle$

by (*simp add: STOP-iff-T ready-set-def*) *presburger*

qed (*simp add: ready-set-STOP*)

lemma *ready-set-SKIP*: $\langle \text{ready-set } SKIP = \{\checkmark\} \rangle$

by (*simp add: ready-set-def T-SKIP*)

lemma *ready-set-BOT*: $\langle \text{ready-set } \perp = UNIV \rangle$

by (*simp add: ready-set-def T-UU*)

These two, on the other hand, are not characterizations.

lemma $\langle \exists P. \text{ready-set } P = \{\checkmark\} \wedge P \neq SKIP \rangle$

proof (*intro exI*)

show $\langle \text{ready-set } (STOP \sqcap SKIP) = \{\checkmark\} \wedge STOP \sqcap SKIP \neq SKIP \rangle$

by (*simp add: ready-set-def T-Ndet T-STOP T-SKIP*)

(*metis SKIP-F-iff SKIP-Neq-STOP idem-F mono-Ndet-F-left*)

qed

lemma $\langle \exists P. \text{ready-set } P = UNIV \wedge P \neq \perp \rangle$

proof (*intro exI*)

show $\langle \text{ready-set } ((\Box a \in UNIV \rightarrow \perp) \sqcap SKIP) = UNIV \wedge ((\Box a \in UNIV \rightarrow \perp) \sqcap SKIP) \neq \perp \rangle$

by (*auto simp add: ready-set-def T-Ndet T-Mprefix Nil-elim-T T-SKIP*

Ndet-is-BOT-iff SKIP-neq-BOT Mprefix-neq-BOT

intro: event.exhaust)

qed

3.4 Behaviour of *ready-set* with Operators of HOL-CSP

lemma *ready-set-Mprefix*: $\langle \text{ready-set } (\Box a \in A \rightarrow P \ a) = \text{ev } \langle A \rangle$

and *ready-set-Mndetprefix*: $\langle \text{ready-set } (\Box a \in A \rightarrow P \ a) = \text{ev } \langle A \rangle$

and *ready-set-prefix*: $\langle \text{ready-set } (a \rightarrow Q) = \{\text{ev } a\} \rangle$

by (*auto simp: ready-set-def T-Mndetprefix write0-def T-Mprefix Nil-elim-T*)

As discussed earlier, *ready-set* behaves well with (\sqcap) and (\sqcap) .

lemma *ready-set-Det*: $\langle \text{ready-set } (P \sqcap Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$
and *ready-set-Ndet*: $\langle \text{ready-set } (P \sqcap Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$
unfolding *ready-set-def* **by** (*auto simp add: T-Det T-Ndet*)

lemma *ready-set-Seq*:

$\langle \text{ready-set } (P ; Q) = (\text{if } P = \perp \text{ then UNIV else ready-set } P - \{\checkmark\} \cup$
 $(\text{if } \checkmark \in \text{ready-set } P \text{ then ready-set } Q \text{ else } \{\})) \rangle$

proof –

have $\langle \text{ready-set } (P ; Q) = \text{ready-set } P \rangle$ **if** $\langle \checkmark \notin \text{ready-set } P \rangle$

proof (*intro subset-antisym subsetI*)

fix e

assume $\langle e \in \text{ready-set } (P ; Q) \rangle$

then obtain s **where** $\langle e \# s \in \mathcal{T} (P ; Q) \rangle$ **by** (*simp add: ready-set-def*)

then consider $\langle e \# s \in \mathcal{T} P \rangle$

| $\langle \exists t1 t2. e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{T} Q \rangle$

by (*simp add: T-Seq*) (*metis F-T NF-ND*)

thus $\langle e \in \text{ready-set } P \rangle$

proof cases

show $\langle e \# s \in \mathcal{T} P \implies e \in \text{ready-set } P \rangle$

by (*simp add: Cons-in-T-imp-elem-ready-set*)

next

assume $\langle \exists t1 t2. e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{T} Q \rangle$

then obtain $t1 t2$ **where** $*$: $\langle e \# s = t1 @ t2 \rangle \langle t1 @ [\checkmark] \in \mathcal{T} P \rangle$ **by** *blast*

with that have $\langle t1 \neq [] \wedge \text{hd } t1 = e \rangle$

by (*metis Cons-in-T-imp-elem-ready-set append-Nil hd-append2 list.sel(1)*)

thus $\langle e \in \text{ready-set } P \rangle$

by (*metis Cons-in-T-imp-elem-ready-set *(2) is-processT3-ST list.exhaust-sel*)

qed

next

show $\langle e \in \text{ready-set } P \implies e \in \text{ready-set } (P ; Q) \rangle$ **for** e

by (*simp add: ready-set-def T-Seq*)

(*metis Cons-in-T-imp-elem-ready-set Nil-elem-T that*
append.right-neutral is-processT5-S7 singletonD)

qed

also have $\langle \text{ready-set } (P ; Q) = \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$

if $\langle P \neq \perp \rangle$ **and** $\langle \checkmark \in \text{ready-set } P \rangle$

proof (*intro subset-antisym subsetI*)

fix e

assume $\langle e \in \text{ready-set } (P ; Q) \rangle$

then obtain s **where** $\langle e \# s \in \mathcal{T} (P ; Q) \rangle$ **by** (*simp add: ready-set-def*)

with $\langle P \neq \perp \rangle$ **consider** $\langle e \# s \in \mathcal{T} P \rangle \langle e \neq \checkmark \rangle$

| $\langle \exists t1 t2. e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{T} Q \rangle$

by (*simp add: T-Seq BOT-iff-D*)

(*metis F-T append-T-imp-tickFree append-butlast-last-id butlast.simps(2)*
last-ConsL list.distinct(1) process-charn tickFree-Cons)

thus $\langle e \in \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$

proof cases

```

show  $\langle e \# s \in \mathcal{T} P \implies e \neq \checkmark \implies e \in \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$ 
  by (simp add: Cons-in-T-imp-elem-ready-set)
next
  assume  $\langle \exists t1\ t2. e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{T} Q \rangle$ 
  then obtain  $t1\ t2$  where  $*$  :  $\langle e \# s = t1 @ t2 \rangle \langle t1 @ [\checkmark] \in \mathcal{T} P \rangle \langle t2 \in \mathcal{T} Q \rangle$ 
by blast
  show  $\langle e \in \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$ 
  proof (cases t1)
    from  $*(1, 3)$  show  $\langle t1 = [] \implies e \in \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$ 
    using Cons-in-T-imp-elem-ready-set by auto
  next
    fix  $e'\ t1'$ 
    assume  $\langle t1 = e' \# t1' \rangle$ 
    with  $*(1, 2)$  have  $\langle t1 = e \# t1' \wedge e \neq \checkmark \rangle$ 
    by (metis append-T-imp-tickFree hd-append list.sel(1) neq-Nil-conv tick-Free-Cons)
    with  $*(2)$  show  $\langle e \in \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$ 
    using Cons-in-T-imp-elem-ready-set by auto
  qed
qed
next
  fix  $e$ 
  assume  $\langle e \in \text{ready-set } P - \{\checkmark\} \cup \text{ready-set } Q \rangle$ 
  then obtain  $s$  where  $\langle e \neq \checkmark \wedge e \# s \in \mathcal{T} P \vee e \# s \in \mathcal{T} Q \rangle$ 
  unfolding ready-set-def by blast
  thus  $\langle e \in \text{ready-set } (P ; Q) \rangle$ 
  proof (elim disjE conjE)
    show  $\langle e \neq \checkmark \implies e \# s \in \mathcal{T} P \implies e \in \text{ready-set } (P ; Q) \rangle$ 
    by (simp add: ready-set-def T-Seq)
      (metis Nil-elem-T append-Cons append-Nil is-processT3-ST is-processT5-S7 singletonD)
  next
    have  $\langle [\checkmark] \in \mathcal{T} P \rangle$  using ready-set-def that(2) by auto
    thus  $\langle e \# s \in \mathcal{T} Q \implies e \in \text{ready-set } (P ; Q) \rangle$ 
    by (simp add: ready-set-def T-Seq)
      (metis append-Cons append-Nil is-processT3-ST)
  qed
qed
ultimately show ?thesis by (simp add: BOT-Seq ready-set-BOT)
qed

```

lemma *ready-set-Sync*:

```

 $\langle \text{ready-set } (P \llbracket S \rrbracket Q) =$ 
  ( if  $P = \perp \vee Q = \perp$  then UNIV
  else  $(\text{ready-set } P - \text{insert } \checkmark (ev \text{ ' } S)) \cup$ 
     $(\text{ready-set } Q - \text{insert } \checkmark (ev \text{ ' } S)) \cup$ 
     $\text{ready-set } P \cap \text{ready-set } Q \cap \text{insert } \checkmark (ev \text{ ' } S) \rangle$ 

```

```

(is ⟨ready-set (P [S] Q) = (if P = ⊥ ∨ Q = ⊥ then UNIV else ?rhs)⟩)
proof –
  have ⟨ready-set (P [S] Q) = ?rhs⟩ if non-BOT : ⟨P ≠ ⊥⟩ ⟨Q ≠ ⊥⟩
  proof (intro subset-antisym subsetI)
    show ⟨e ∈ ?rhs ⟹ e ∈ ready-set (P [S] Q)⟩ for e
      by (use Nil-elem-T in ⟨fastforce simp add: ready-set-def T-Sync⟩)
  next
    fix e
    assume ⟨e ∈ ready-set (P [S] Q)⟩
    then consider ⟨∃ t u. t ∈ T P ∧ u ∈ T Q ∧ [e] setinterleaves ((t, u), insert
    ✓ (ev ‘ S))⟩
      | ⟨∃ t u r v. front-tickFree v ∧ (tickFree r ∨ v = []) ∧ [e] = r @ v ∧
        r setinterleaves ((t, u), insert ✓ (ev ‘ S)) ∧
        (t ∈ D P ∧ u ∈ T Q ∨ t ∈ D Q ∧ u ∈ T P)⟩
      by (simp add: ready-set-def T-Sync) blast
    thus ⟨e ∈ ?rhs⟩
  proof cases
    assume ⟨∃ t u. t ∈ T P ∧ u ∈ T Q ∧ [e] setinterleaves ((t, u), insert ✓ (ev
    ‘ S))⟩
    then obtain t u where assms : ⟨t ∈ T P⟩ ⟨u ∈ T Q⟩
      ⟨[e] setinterleaves ((t, u), insert ✓ (ev ‘ S))⟩ by blast
    thus ⟨e ∈ ?rhs⟩
    apply (cases t; cases u; simp add: ready-set-def split: if-split-asm)
    using empty-setinterleaving Sync.sym by blast+
  next
    assume ⟨∃ t u r v. front-tickFree v ∧ (tickFree r ∨ v = []) ∧ [e] = r @ v ∧
      r setinterleaves ((t, u), insert ✓ (ev ‘ S)) ∧
      (t ∈ D P ∧ u ∈ T Q ∨ t ∈ D Q ∧ u ∈ T P)⟩
    then obtain t u r v
      where * : ⟨front-tickFree v⟩ ⟨tickFree r ∨ v = []⟩ ⟨[e] = r @ v⟩
        ⟨r setinterleaves ((t, u), insert ✓ (ev ‘ S))⟩
        ⟨t ∈ D P ∧ u ∈ T Q ∨ t ∈ D Q ∧ u ∈ T P⟩ by blast
    have ⟨r ≠ []⟩ using *(4, 5) BOT-iff-D empty-setinterleaving that by blast
    hence ⟨r = [e] ∧ v = []⟩
      by (metis (no-types, lifting) *(3) Nil-is-append-conv append-eq-Cons-conv)
    also obtain e' t' where ⟨t = e' # t'⟩
      using *(5) BOT-iff-D non-BOT by (cases t; blast)
    ultimately show ⟨e ∈ ?rhs⟩
      using *(4, 5) apply (simp add: ready-set-def subset-iff T-Sync)
      apply (cases u; simp split: if-split-asm)
      by (metis (no-types, opaque-lifting) [[metis-verbose = false]]
        D-T Sync.sym empty-setinterleaving)+
  qed
qed
thus ⟨ready-set (P [S] Q) = (if P = ⊥ ∨ Q = ⊥ then UNIV else ?rhs)⟩
  by (simp add: Sync-BOT Sync-commute[of ⊥, simplified Sync-BOT] ready-set-BOT)
qed

```

lemma *ready-set-Renaming*:
 $\langle \text{ready-set } (\text{Renaming } P f) = (\text{if } P = \perp \text{ then UNIV else EvExt } f \text{ ' (ready-set } P)) \rangle$
proof –
{ **fix** $y \ t1 \ t2$
assume $\text{assms} : \langle [] \notin \mathcal{D} P \rangle \langle y = \text{map } (\text{EvExt } f) \ t1 \ @ \ t2 \rangle \langle t1 \in \mathcal{D} P \rangle$
from assms **have** $\langle t2 = [] \rangle$
by (*metis Nil-is-append-conv list.map-disc-iff list.sel(3) tl-append2*)
with $\text{assms}(2) \ D\text{-T}[OF \ \text{assms}(3)]$ **have** $\langle \exists x. [x] \in \mathcal{T} P \wedge y = \text{EvExt } f \ x \rangle$ **by**
auto
}
thus *?thesis* **by** (*auto simp add: ready-set-def T-Renaming image-iff BOT-iff-D*)
qed

Because for the expression of its traces (and more specifically of its divergences), dealing with *Hiding* is much more difficult.

We start with two characterizations:

- one to understand $P \setminus S = \perp$
- the other to understand $[e] \in \mathcal{D} (P \setminus S)$.

lemma *Hiding-is-BOT-iff* :
 $\langle P \setminus S = \perp \iff (\exists t. \text{set } t \subseteq \text{ev ' } S \wedge$
 $(t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f \ P \ S \wedge t \in \text{range } f))) \rangle$
(is $\langle P \setminus S = \perp \iff ?rhs \rangle$)
proof (*subst BOT-iff-D, intro iffI*)
show $\langle [] \in \mathcal{D} (P \setminus S) \implies ?rhs \rangle$
by (*simp add: D-Hiding*)
(*metis (no-types, lifting) filter-empty-conv subsetI*)
next
assume $\langle ?rhs \rangle$
then obtain t **where** $*$: $\langle \text{set } t \subseteq \text{ev ' } S \rangle$
 $\langle t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f \ P \ S \wedge t \in \text{range } f) \rangle$ **by**
blast
hence $\langle \text{tickFree } t \wedge [] = \text{trace-hide } t \ (\text{ev ' } S) \rangle$
unfolding *tickFree-def* **by** (*auto simp add: D-Hiding subset-iff*)
with $*(2)$ **show** $\langle [] \in \mathcal{D} (P \setminus S) \rangle$ **by** (*simp add: D-Hiding metis*)
qed

lemma *event-in-D-Hiding-iff* :
 $\langle [e] \in \mathcal{D} (P \setminus S) \iff$
 $P \setminus S = \perp \vee (\exists x \ t. e = \text{ev } x \wedge x \notin S \wedge [ev \ x] = \text{trace-hide } t \ (\text{ev ' } S) \wedge$
 $(t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f \ P \ S \wedge t \in \text{range } f))) \rangle$
(is $\langle [e] \in \mathcal{D} (P \setminus S) \iff P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$)
proof (*intro iffI*)
assume $\text{assm} : \langle [e] \in \mathcal{D} (P \setminus S) \rangle$
show $\langle P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$
proof (*cases e*)

```

assume  $\langle e = \checkmark \rangle$ 
with assm have  $\langle P \setminus S = \perp \rangle$ 
  using BOT-iff-D front-tickFree-Nil is-processT9-tick by blast
thus  $\langle P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$  by blast
next
fix x
assume  $\langle e = ev\ x \rangle$ 
with assm obtain t u
  where  $*$  :  $\langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle$ 
     $\langle [ev\ x] = \text{trace-hide } t (ev\ 'S) @\ u \rangle$ 
     $\langle t \in \mathcal{D}\ P \vee (\exists f. \text{isInfHiddenRun } f\ P\ S \wedge t \in \text{range } f) \rangle$ 
  by (simp add: D-Hiding) blast
from  $*(3)$  consider  $\langle \text{set } t \subseteq ev\ 'S \mid \langle x \notin S \rangle \langle ev\ x \in \text{set } t \rangle$ 
  by (metis (no-types, lifting) Cons-eq-append-conv empty-filter-conv
    filter-eq-Cons-iff filter-is-subset image-eqI list.set-intros(1) subset-code(1))
thus  $\langle P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$ 
proof cases
  assume  $\langle \text{set } t \subseteq ev\ 'S \rangle$ 
  hence  $\langle P \setminus S = \perp \rangle$  by (meson *(4) Hiding-is-BOT-iff)
  thus  $\langle P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$  by blast
next
  assume  $\langle x \notin S \rangle \langle ev\ x \in \text{set } t \rangle$ 
  with  $*(3)$  have  $\langle [ev\ x] = \text{trace-hide } t (ev\ 'S) \rangle$ 
  by (induct t) (auto split: if-split-asm)
  with  $*(4)$   $\langle e = ev\ x \rangle \langle x \notin S \rangle$  have  $\langle ?ugly\text{-assertion} \rangle$  by blast
  thus  $\langle P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$  by blast
qed
qed
next
show  $\langle P \setminus S = \perp \vee ?ugly\text{-assertion} \implies [e] \in \mathcal{D}\ (P \setminus S) \rangle$ 
proof (elim disjE)
  show  $\langle P \setminus S = \perp \implies [e] \in \mathcal{D}\ (P \setminus S) \rangle$  by (simp add: D-UU)
next
  show  $\langle ?ugly\text{-assertion} \implies [e] \in \mathcal{D}\ (P \setminus S) \rangle$ 
  by (elim exE, simp add: D-Hiding)
    (metis Hiding-tickFree append-Nil2 event.simps(3)
    front-tickFree-Nil tickFree-Cons tickFree-Nil)
qed
qed

```

Now we can express *ready-set* $(P \setminus S)$. This result contains the term $P \setminus S = \perp$ that can be unfolded with *Hiding-is-BOT-iff* and the term $[ev\ x] \in \mathcal{D}\ (P \setminus S)$ that can be unfolded with *event-in-D-Hiding-iff*.

lemma *ready-set-Hiding*:

```

 $\langle \text{ready-set } (P \setminus S) =$ 
  (if  $P \setminus S = \perp$  then UNIV else
    {e. case e of  $\checkmark \implies \exists t. \text{set } t \subseteq ev\ 'S \wedge t @\ [\checkmark] \in \mathcal{T}\ P$ 
      |  $ev\ x \implies x \notin S \wedge ([ev\ x] \in \mathcal{D}\ (P \setminus S) \vee$ 
         $(\exists t. [ev\ x] = \text{trace-hide } t (ev\ 'S) \wedge (t, ev\ 'S) \in \mathcal{F}\ P))\}$ 

```

```

(is ⟨ready-set (P \ S) = (if P \ S = ⊥ then UNIV else ?set)⟩)
proof (split if-split, intro conjI impI)
  show ⟨P \ S = ⊥ ⟹ ready-set (P \ S) = UNIV⟩ by (simp add: ready-set-BOT)
next
  assume non-BOT : ⟨P \ S ≠ ⊥⟩
  show ⟨ready-set (P \ S) = ?set⟩
  proof (intro subset-antisym subsetI)
    fix e
    assume ready : ⟨e ∈ ready-set (P \ S)⟩
    — This implies e ∉ ev ' S with our other assumptions.
    { fix x
      assume assms : ⟨x ∈ S⟩ ⟨ev x ∈ ready-set (P \ S)⟩
      then consider ⟨∃ t. [ev x] = trace-hide t (ev ' S) ∧ (t, ev ' S) ∈ F P⟩
        | ⟨∃ t u. front-tickFree u ∧ tickFree t ∧ [ev x] = trace-hide t (ev ' S) @ u ∧
          (t ∈ D P ∨ (∃ f. isInfHiddenRun f P S ∧ t ∈ range f))⟩
        by (simp add: ready-set-def T-Hiding) blast
      hence ⟨P \ S = ⊥⟩
    proof cases
      assume ⟨∃ t. [ev x] = trace-hide t (ev ' S) ∧ (t, ev ' S) ∈ F P⟩
      hence False by (metis Cons-eq-filterD image-eqI assms(1))
      thus ⟨P \ S = ⊥⟩ by blast
    next
      assume ⟨∃ t u. front-tickFree u ∧ tickFree t ∧ [ev x] = trace-hide t (ev ' S)
@ u ∧
          (t ∈ D P ∨ (∃ f. isInfHiddenRun f P S ∧ t ∈ range f))⟩
      then obtain t u
        where * : ⟨front-tickFree u⟩ ⟨tickFree t⟩ ⟨[ev x] = trace-hide t (ev ' S) @
u⟩
          ⟨t ∈ D P ∨ (∃ f. isInfHiddenRun f P S ∧ t ∈ range f)⟩ by blast
      from *(3) have ** : ⟨set t ⊆ ev ' S⟩
        by (induct t) (simp-all add: assms(1) split: if-split-asm)
      from *(4) ** Hiding-is-BOT-iff show ⟨P \ S = ⊥⟩ by blast
    qed
  }
  with ready have * : ⟨e ∉ ev ' S⟩ using non-BOT by blast

from ready consider ⟨[e] ∈ D (P \ S)⟩
  | ⟨∃ t. [e] = trace-hide t (ev ' S) ∧ (t, ev ' S) ∈ F P⟩
  unfolding ready-set-def by (simp add: T-Hiding D-Hiding) blast
thus ⟨e ∈ ?set⟩
proof cases
  assume assm : ⟨[e] ∈ D (P \ S)⟩
  then obtain x where ⟨e = ev x⟩
    — because [✓] ∈ D (P \ S) ⟹ P \ S = ⊥
    by (metis BOT-iff-D append-Nil event.exhaust non-BOT process-charn)
  with assm * show ⟨e ∈ ?set⟩ by (simp add: image-iff)
next
  assume ⟨∃ t. [e] = trace-hide t (ev ' S) ∧ (t, ev ' S) ∈ F P⟩
  then obtain t where ** : ⟨[e] = trace-hide t (ev ' S)⟩

```

```

      ⟨t, ev ‘ S⟩ ∈  $\mathcal{F} P$ ⟩ by blast
  thus ⟨e ∈ ?set⟩
  proof (cases e)
    have ⟨e = ✓ ⇒ set (butlast t) ⊆ ev ‘ S ∧ butlast t @ [✓] ∈  $\mathcal{T} P$ ⟩
      using ** apply (cases t rule: rev-cases; simp add: split: if-split-asm)
      by (metis F-T filter-empty-conv subset-code(1))
      (metis Hiding-tickFree front-tickFree-implies-tickFree process-charn
  tickFree-Cons)
    thus ⟨e = ✓ ⇒ e ∈ ?set⟩ by auto
  next
  fix x
  assume ⟨e = ev x⟩
  with * have ⟨x ∉ S⟩ by blast
  with ⟨e = ev x⟩ *(1) *(2) show ⟨e ∈ ?set⟩ by auto
  qed
  qed
  next
  fix e
  assume ⟨e ∈ ?set⟩
  then consider ⟨e = ✓⟩ ⟨∃ t. set t ⊆ ev ‘ S ∧ t @ [✓] ∈  $\mathcal{T} P$ ⟩
    | ⟨∃ x. e = ev x ∧ x ∉ S ∧
      ([ev x] ∈  $\mathcal{D} (P \setminus S)$  ∨
      (∃ t. [ev x] = trace-hide t (ev ‘ S) ∧ (t, ev ‘ S) ∈  $\mathcal{F} P$ ))⟩ by (cases e;
  simp)
  thus ⟨e ∈ ready-set (P \ S)⟩
  proof cases
    assume assms : ⟨e = ✓⟩ ⟨∃ t. set t ⊆ ev ‘ S ∧ t @ [✓] ∈  $\mathcal{T} P$ ⟩
    from assms(2) obtain t
      where * : ⟨set t ⊆ ev ‘ S⟩ ⟨t @ [✓] ∈  $\mathcal{T} P$ ⟩ by blast
    have ** : ⟨[e] = trace-hide (t @ [✓]) (ev ‘ S) ∧ (t @ [✓], ev ‘ S) ∈  $\mathcal{F} P$ ⟩
      using *(1) by (simp add: assms(1) image-iff tick-T-F[OF *(2)] subset-iff)
    show ⟨e ∈ ready-set (P \ S)⟩
      unfolding ready-set-def by (simp add: T-Hiding) (use ** in blast)
  next
  show ⟨∃ x. e = ev x ∧ x ∉ S ∧
    ([ev x] ∈  $\mathcal{D} (P \setminus S)$  ∨
    (∃ t. [ev x] = trace-hide t (ev ‘ S) ∧ (t, ev ‘ S) ∈  $\mathcal{F} P$ )) ⇒
    e ∈ ready-set (P \ S)⟩
    apply (elim exE conjE disjE)
    using Cons-in-T-imp-elem-ready-set D-T apply blast
    unfolding ready-set-def by (simp add: T-Hiding) blast
  qed
  qed
  qed

```

In the end the result would look something like this:

$$\text{ready-set } (P \setminus S) = (\text{if } \exists t. \text{set } t \subseteq \text{ev ‘ } S \wedge (t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f P S \wedge t \in \text{range } f)) \text{ then UNIV else } \{e. \text{case } e \text{ of ev } x \Rightarrow x \notin S \wedge ((\exists t. \text{set } t \subseteq \text{ev ‘ } S \wedge (t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f P S \wedge t \in \text{range } f))) \vee$$

$(\exists xa t. ev x = ev xa \wedge xa \notin S \wedge [ev xa] = trace\text{-}hide\ t (ev \text{' } S) \wedge (t \in \mathcal{D} P \vee (\exists f. isInfHiddenRun\ f\ P\ S \wedge t \in range\ f))) \vee (\exists t. [ev x] = trace\text{-}hide\ t (ev \text{' } S) \wedge (t, ev \text{' } S) \in \mathcal{F} P) \mid \checkmark \Rightarrow \exists t. set\ t \subseteq ev \text{' } S \wedge t @ [\checkmark] \in \mathcal{T} P\}$

Obviously, it is not very easy to use. We will therefore rely more on the corollaries below.

corollary *ready-tick-Hiding-iff* :

$\langle \checkmark \in ready\text{-}set (P \setminus B) \longleftrightarrow P \setminus B = \perp \vee (\exists t. set\ t \subseteq ev \text{' } B \wedge t @ [\checkmark] \in \mathcal{T} P) \rangle$
by (*simp add: ready-set-Hiding*)

corollary *ready-tick-imp-ready-tick-Hiding*:

$\langle \checkmark \in ready\text{-}set\ P \Longrightarrow \checkmark \in ready\text{-}set (P \setminus B) \rangle$
by (*subst ready-set-Hiding, simp add: ready-set-def*)
(metis append-Nil empty-iff empty-set subset-iff)

corollary *ready-inside-Hiding-iff* :

$\langle e \in S \Longrightarrow ev\ e \in ready\text{-}set (P \setminus S) \longleftrightarrow P \setminus S = \perp \rangle$
by (*simp add: ready-set-Hiding*)

corollary *ready-notin-Hiding-iff* :

$\langle e \notin S \Longrightarrow ev\ e \in ready\text{-}set (P \setminus S) \longleftrightarrow P \setminus S = \perp \vee (\exists t. [ev\ e] = trace\text{-}hide\ t (ev \text{' } S) \wedge (t \in \mathcal{D} P \vee (\exists f. isInfHiddenRun\ f\ P\ S \wedge t \in range\ f) \vee (t, ev \text{' } S) \in \mathcal{F} P)) \rangle$
by (*auto simp add: ready-set-Hiding event-in-D-Hiding-iff split: if-split-asm*)

corollary *ready-notin-imp-ready-Hiding*:

$\langle ev\ e \in ready\text{-}set (P \setminus S) \rangle$ **if** *ready* : $\langle ev\ e \in ready\text{-}set\ P \rangle$ **and** *notin* : $\langle e \notin S \rangle$

proof –

from *inf-hidden[of S <[ev e]> P]*

consider $\langle \exists f. isInfHiddenRun\ f\ P\ S \wedge [ev\ e] \in range\ f \rangle$

$\mid \langle \exists t. [ev\ e] = trace\text{-}hide\ t (ev \text{' } S) \wedge (t, ev \text{' } S) \in \mathcal{F} P \rangle$

by (*simp add: ready-set-Hiding image-iff[of <ev e> notin]*)

(metis mem-Collect-eq ready ready-set-def)

thus $\langle ev\ e \in ready\text{-}set (P \setminus S) \rangle$

proof *cases*

show $\langle \exists f. isInfHiddenRun\ f\ P\ S \wedge [ev\ e] \in range\ f \Longrightarrow ev\ e \in ready\text{-}set (P \setminus S) \rangle$

apply (*rule Cons-in-T-imp-elem-ready-set[of <ev e> <[]>], rule D-T*)

apply (*simp add: event-in-D-Hiding-iff image-iff[of <ev e> notin]*)

by (*metis (no-types, lifting) event.inject filter.simps image-iff notin*)

next

assume $\langle \exists t. [ev\ e] = trace\text{-}hide\ t (ev \text{' } S) \wedge (t, ev \text{' } S) \in \mathcal{F} P \rangle$

thus $\langle ev\ e \in ready\text{-}set (P \setminus S) \rangle$ **by** (*simp add: ready-set-Hiding notin*)

qed

qed

3.5 Behaviour of *ready-set* with Operators of HOL-CSPM

lemma *ready-set-MultiDet*:

$\langle \text{finite } A \implies \text{ready-set } (\text{MultiDet } A \ P) = (\bigcup a \in A. \text{ready-set } (P \ a)) \rangle$
by (*induct A rule: finite-induct*)
(simp-all add: ready-set-STOP ready-set-Det)

lemma *ready-set-MultiNdet*:

$\langle \text{finite } A \implies \text{ready-set } (\text{MultiNdet } A \ P) = (\bigcup a \in A. \text{ready-set } (P \ a)) \rangle$
apply (*cases $\langle A = \{\} \rangle$, simp add: ready-set-STOP*)
by (*rotate-tac, induct A rule: finite-set-induct-nonempty*)
(simp-all add: ready-set-Ndet)

lemma *ready-set-GlobalNdet*:

$\langle \text{ready-set } (\text{GlobalNdet } A \ P) = (\bigcup a \in A. \text{ready-set } (P \ a)) \rangle$
by (*auto simp add: ready-set-def T-GlobalNdet*)

lemma *ready-set-MultiSeq*:

$\langle \text{ready-set } (\text{MultiSeq } L \ P) =$
(if $L = []$ then $\{\checkmark\}$
else if $P \ (\text{hd } L) = \perp$ then UNIV
else if $\checkmark \in \text{ready-set } (P \ (\text{hd } L))$
then $\text{ready-set } (P \ (\text{hd } L)) - \{\checkmark\} \cup \text{ready-set } (\text{MultiSeq } (\text{tl } L) \ P)$
else $\text{ready-set } (P \ (\text{hd } L)) \rangle$
by (*induct L*) (*simp-all add: ready-set-SKIP ready-set-Seq*)

lemma *ready-set-MultiSync*:

$\langle \text{ready-set } (\llbracket S \rrbracket \ m \in\# \ M. \ P \ m) =$
(if $M = \{\#\}$ then $\{\}$
else if $\exists m \in\# \ M. \ P \ m = \perp$ then UNIV
else if $\exists m. \ M = \{\#m\# \}$ then $\text{ready-set } (P \ (\text{THE } m. \ M = \{\#m\# \}))$
else $\{e. \exists m \in\# \ M. e \in \text{ready-set } (P \ m) - \text{insert } \checkmark \ (ev \ ' \ S)\} \cup$
 $\{e \in \text{insert } \checkmark \ (ev \ ' \ S). \forall m \in\# \ M. e \in \text{ready-set } (P \ m)\} \rangle$

proof –

have $*$: $\langle \text{ready-set } (\llbracket S \rrbracket \ m \in\# \ M + \{\#a, a'\#\}. \ P \ m) =$
 $\{e. \exists m \in\# \ M + \{\#a, a'\#\}. e \in \text{ready-set } (P \ m) - \text{insert } \checkmark \ (ev \ ' \ S)\} \cup$
 $\{e \in \text{insert } \checkmark \ (ev \ ' \ S). \forall m \in\# \ M + \{\#a, a'\#\}. e \in \text{ready-set } (P \ m)\} \rangle$

if *non-BOT* : $\langle \forall m \in\# \ M + \{\#a, a'\#\}. \ P \ m \neq \perp \rangle$ **for** $a \ a' \ M$

proof (*induct M rule: msubset-induct'[OF subset-mset.refl]*)

case 1

then show *?case* **by** (*auto simp add: non-BOT ready-set-Sync*)

next

case ($2 \ a'' \ M'$)

```

have * : ⟨MultiSync S (add-mset a'' M' + {#a, a'#}) P =
  P a'' [[S]] (MultiSync S (M' + {#a, a'#}) P)⟩
by (simp add: add-mset-commute)
have ** : ⟨¬ (P a'' = ⊥ ∨ MultiSync S (M' + {#a, a'#}) P = ⊥)⟩
using 2.hyps(1, 2) in-diffD non-BOT
by (auto simp add: MultiSync-is-BOT-iff Sync-is-BOT-iff, fastforce, meson
mset-subset-eqD)
show ?case
by (auto simp only: * ready-set-Sync ** 2.hyps(3), auto)
qed

show ?thesis
proof (cases ⟨∃ m ∈ # M. P m = ⊥⟩)
show ⟨∃ m ∈ # M. P m = ⊥ ⟹ ?thesis⟩
by (simp add: ready-set-STOP) (metis MultiSync-BOT-absorb ready-set-BOT)
next
show ⟨¬ (∃ m ∈ # M. P m = ⊥) ⟹ ?thesis⟩
proof (cases ⟨∃ a a' M'. M = M' + {#a, a'#}⟩)
assume assms : ⟨¬ (∃ m ∈ # M. P m = ⊥)⟩ ⟨∃ a a' M'. M = M' + {#a, a'#}⟩
from assms(2) obtain a a' M' where ⟨M = M' + {#a, a'#}⟩ by blast
with * assms(1) show ?thesis by simp
next
assume ⟨∄ a a' M'. M = M' + {#a, a'#}⟩
hence ⟨M = {#} ∨ (∃ m. M = {#m#})⟩
by (metis add.right-neutral multiset-cases union-mset-add-mset-right)
thus ?thesis by (auto simp add: ready-set-STOP ready-set-BOT)
qed
qed
qed

```

3.6 Behaviour of *ready-set* with Operators of HOL-CSP_0pSem

lemma *ready-set-Sliding*:

```

⟨ready-set (P ▷ Q) = ready-set P ∪ ready-set Q⟩
unfolding ready-set-def by (auto simp add: T-Sliding)

```

lemma *ready-set-Throw*: ⟨ $ready-set (P \Theta a \in A. Q a) = ready-set P$ ⟩

```

apply (intro subset-antisym subsetI;
  simp add: ready-set-def T-Throw image-iff)
apply (elim disjE)
apply (solves ⟨simp⟩)
apply (metis D-T process-charn)
apply (metis Nil-is-append-conv list.sel(3) neq-Nil-conv self-append-conv2 tl-append2)
by (metis Nil-elem-T append-Nil empty-set inf-bot-left)

```

corollary *Throw-is-STOP-iff*: ⟨ $P \Theta a \in A. Q a = STOP \iff P = STOP$ ⟩

```

by (simp add: ready-set-empty-iff-STOP[symmetric] ready-set-Throw)

```

lemma *ready-set-Interrupt*: $\langle \text{ready-set } (P \triangle Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$
apply (*intro subset-antisym subsetI*;
simp add: ready-set-def T-Interrupt)
by (*metis Nil-is-append-conv append.left-neutral*
append.right-neutral butlast.simps(2) butlast-append)
(use Nil-elem-T tickFree-Nil in blast)

corollary *Interrupt-is-STOP-iff*: $\langle P \triangle Q = \text{STOP} \iff P = \text{STOP} \wedge Q = \text{STOP} \rangle$
by (*simp add: ready-set-empty-iff-STOP[symmetric] ready-set-Interrupt*)

3.7 Behaviour of *ready-set* with Reference Processes

lemma *ready-set-DF*: $\langle \text{ready-set } (DF A) = \text{ev } 'A \rangle$
by (*subst DF-unfold*) (*simp add: ready-set-Mndetprefix*)

lemma *ready-set-DF_SKIP*: $\langle \text{ready-set } (DF_{SKIP} A) = \text{insert } \checkmark (ev 'A) \rangle$
by (*subst DF_SKIP-unfold*)
(simp add: ready-set-Mndetprefix ready-set-Ndet ready-set-SKIP)

lemma *ready-set-RUN*: $\langle \text{ready-set } (RUN A) = \text{ev } 'A \rangle$
by (*subst RUN-unfold*) (*simp add: ready-set-Mprefix*)

lemma *ready-set-CHAOS*: $\langle \text{ready-set } (CHAOS A) = \text{ev } 'A \rangle$
by (*subst CHAOS-unfold*)
(simp add: ready-set-Mprefix ready-set-Ndet ready-set-STOP)

lemma *ready-set-CHAOS_SKIP*: $\langle \text{ready-set } (CHAOS_{SKIP} A) = \text{insert } \checkmark (ev 'A) \rangle$
by (*subst CHAOS_SKIP-unfold*)
(simp add: ready-set-Mprefix ready-set-Ndet
ready-set-STOP ready-set-SKIP)

end

Chapter 4

Construction of the After Operator

```
theory After
  imports ReadySet
begin
```

Now that we have defined *ready-set* P , we can talk about what happens to P after an event belonging to this set.

4.1 Definition

We want to define a new operator on a process P which would in some way be the reciprocal of the prefix operator $e \rightarrow P$.

The intuitive way of doing so is to only keep the tails of the traces beginning by $ev\ e$ (and similar for failures and divergences). However we have an issue if $ev\ e \notin \text{ready-set } P$ i.e. if no trace of P begins with $ev\ e$: the result would no longer verify the invariant *is-process* because its trace set would be empty. We must therefore distinguish this case and we agree to then obtain *STOP*. This convention is not really decisive since we will only use this operator when $ev\ e \in \text{ready-set } P$ to define operational semantics.

```
lift-definition After ::  $\langle [\alpha\ \text{process}, \alpha] \Rightarrow \alpha\ \text{process} \rangle$  (infixl  $\langle \text{after} \rangle$  77)
is  $\langle \lambda P\ e.$   if  $ev\ e \in \text{ready-set } P$ 
                then  $\langle \{(tl\ s, X) \mid s\ X. (s, X) \in \mathcal{F}\ P \wedge s \neq [] \wedge hd\ s = ev\ e\},$ 
                     $\{ tl\ s \mid s . s \in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = ev\ e \} \rangle$ 
                else  $\langle \{(s, X). s = []\}, \{ \} \rangle$ 
```

proof –

```
show  $\langle ?thesis\ P\ e \rangle$  (is  $\langle \text{is-process} \ (if\ ev\ e \in \text{ready-set } P\ then\ (\ ?f, \ ?d)$ 
                         $else\ \{(s, X). s = []\}, \{ \} \rangle \rangle$ ) for  $P\ e$ 
```

```
proof (split if-split, intro conjI impI)
```

```
show  $\langle \text{is-process} \ (\{(s, X). s = []\}, \{ \}) \rangle$  by (metis STOP.rsp eq-onp-def)
```

```
next
```

```

assume ready: ⟨ev e ∈ ready-set P⟩
show ⟨is-process (?f, ?d)⟩
unfolding is-process-def FAILURES-def DIVERGENCES-def fst-conv snd-conv
proof (intro conjI impI allI)
  show ⟨([], {}) ∈ ?f⟩
    using ready[unfolded ready-set-def T-F-spec[symmetric]] by force
next
  show ⟨(s, X) ∈ ?f ⇒ front-tickFree s⟩ for s X
    by simp (metis butlast-rev butlast-tl front-tickFree-def is-processT2 tick-Free-butlast)
next
  show ⟨(s @ t, {}) ∈ ?f ⇒ (s, {}) ∈ ?f⟩ for s t
    by simp (metis (no-types, opaque-lifting) append-Cons is-processT3 list.sel(1, 3) neq-Nil-conv)
next
  show ⟨(s, Y) ∈ ?f ∧ X ⊆ Y ⇒ (s, X) ∈ ?f⟩ for s X Y
    using is-processT4 by simp blast
next
  show ⟨(s, X) ∈ ?f ∧ (∀ c. c ∈ Y ⇒ (s @ [c], {}) ∉ ?f) ⇒ (s, X ∪ Y) ∈ ?f⟩ for s X Y
    using ready[unfolded ready-set-def T-F-spec[symmetric]]
    by auto (metis Nil-is-append-conv hd-append2 is-processT5 tl-append2)
next
  show ⟨(s @ [✓], {}) ∈ ?f ⇒ (s, X - {✓}) ∈ ?f⟩ for s X
    by simp (metis (no-types, lifting) Cons-eq-appendI is-processT6 list.collapse list.distinct(1) list.sel(1, 3))
next
  fix s t :: ⟨α trace⟩
  assume ⟨s ∈ ?d ∧ tickFree s ∧ front-tickFree t⟩
  hence ⟨s @ t = tl (ev e # s @ t) ∧ ev e # s @ t ∈ D P ∧ ev e # s @ t ≠ [] ∧ hd (ev e # s @ t) = ev e⟩
    by simp (metis append-Cons event.distinct(1) is-processT7-S list.sel(1, 3) neq-Nil-conv tickFree-Cons)
  show ⟨s @ t ∈ ?d⟩
    apply simp
    using ⟨this⟩ by blast
next
  show ⟨s ∈ ?d ⇒ (s, X) ∈ ?f⟩ for s X
    using NF-ND by blast
next
  show ⟨s @ [✓] ∈ ?d ⇒ s ∈ ?d⟩ for s
    by auto (metis Cons-eq-appendI is-processT9-S-swap list.sel(1, 3) neq-Nil-conv)
qed
qed
qed

```

4.2 Projections

lemma *F-After*:

$\langle \mathcal{F} (P \text{ after } e) = (\text{ if } ev \ e \in \text{ ready-set } P$
 $\text{ then } \{(tl \ s, X) \mid s \ X. (s, X) \in \mathcal{F} \ P \wedge s \neq [] \wedge hd \ s = ev \ e\}$
 $\text{ else } \{(s, X). s = []\} \rangle$
by (*simp add: Failures-def After.rep-eq FAILURES-def*)

lemma *D-After*:

$\langle \mathcal{D} (P \text{ after } e) = (\text{ if } ev \ e \in \text{ ready-set } P$
 $\text{ then } \{tl \ s \mid s. s \in \mathcal{D} \ P \wedge s \neq [] \wedge hd \ s = ev \ e\}$
 $\text{ else } \{\}\rangle$
by (*simp add: Divergences-def After.rep-eq DIVERGENCES-def*)

lemma *T-After*:

$\langle \mathcal{T} (P \text{ after } e) = (\text{ if } ev \ e \in \text{ ready-set } P$
 $\text{ then } \{tl \ s \mid s. s \in \mathcal{T} \ P \wedge s \neq [] \wedge hd \ s = ev \ e\}$
 $\text{ else } \{[]\} \rangle$
by (*auto simp add: T-F-spec[symmetric] F-After*)

lemma *not-ready-After*: $\langle ev \ e \notin \text{ ready-set } P \implies P \text{ after } e = STOP \rangle$
by (*simp add: STOP-iff-T T-After*)

lemma *ready-set-After*: $\langle \text{ ready-set } (P \text{ after } e) = \{a. ev \ e \# [a] \in \mathcal{T} \ P\} \rangle$
apply (*simp add: T-After ready-set-def, safe*)
apply (*metis list.exhaust-sel*)
apply (*metis list.discI list.sel(1, 3)*)
by (*simp add: is-processT3-ST-pref le-list-def*)

4.3 Monotony

lemma *mono-After* : $\langle P \text{ after } e \sqsubseteq Q \text{ after } e \rangle$ **if** $\langle P \sqsubseteq Q \rangle$

proof (*subst le-approx-def, safe*)

from *that[THEN anti-mono-ready-set] that[THEN le-approx1]*

show $\langle s \in \mathcal{D} (Q \text{ after } e) \implies s \in \mathcal{D} (P \text{ after } e) \rangle$ **for** s

by (*simp add: D-After ready-set-def subset-iff split: if-split-asm*) *blast*

next

from *that[THEN anti-mono-ready-set] that[THEN le-approx2]*

show $\langle s \notin \mathcal{D} (P \text{ after } e) \implies X \in \mathcal{R}_a (P \text{ after } e) \ s \implies X \in \mathcal{R}_a (Q \text{ after } e) \ s \rangle$

for $s \ X$

apply (*simp add: Ra-def D-After F-After ready-set-def subset-iff split: if-split-asm*)

by (*metis F-T append-Cons append-Nil is-processT3-ST list.exhaust-sel*) *blast*

next

from *that[THEN anti-mono-ready-set] that[THEN le-approx2]*

show $\langle s \notin \mathcal{D} (P \text{ after } e) \implies X \in \mathcal{R}_a (Q \text{ after } e) \ s \implies X \in \mathcal{R}_a (P \text{ after } e) \ s \rangle$

for $s \ X$

apply (*simp add: Ra-def D-After F-After ready-set-def subset-iff split: if-split-asm*)

by *blast* (*metis T-F-spec list.distinct(1) list.sel(1, 3)*)
next
show $\langle s \in \text{min-elems } (\mathcal{D} (P \text{ after } e)) \implies s \in \mathcal{T} (Q \text{ after } e) \rangle$ **for** s
proof (*cases* $\langle P = \perp \rangle$)
 assume $\langle s \in \text{min-elems } (\mathcal{D} (P \text{ after } e)) \rangle$ **and** $\langle P = \perp \rangle$
 hence $\langle s = [] \rangle$
 by (*simp add: BOT-iff-D D-After ready-set-BOT D-UU min-elems-def*)
 (*metis front-tickFree-single less-list-def list.distinct(1) list.sel(1, 3) nil-le*)
 thus $\langle s \in \mathcal{T} (Q \text{ after } e) \rangle$ **by** (*simp add: Nil-elem-T*)
next
assume *assms* : $\langle P \neq \perp \rangle \langle s \in \text{min-elems } (\mathcal{D} (P \text{ after } e)) \rangle$
from *assms*(2)[*THEN elem-min-elems*] **have** $*$: $\langle \text{ev } e \# s \in \mathcal{D} P \rangle$
 by (*simp add: D-After split: if-split-asm*) (*metis list.collapse*)
 { **assume** $\langle \text{ev } e \# s \notin \text{min-elems } (\mathcal{D} P) \rangle$
 with *assms*(1) $*$ **obtain** $a t$ **where** $\langle a \# t \in \mathcal{D} P \rangle \langle a \# t < \text{ev } e \# s \rangle$
 by (*simp add: BOT-iff-D min-elems-def*) (*metis list.exhaust*)
 hence $\langle a = \text{ev } e \wedge t < s \wedge t \in \mathcal{D} (P \text{ after } e) \rangle$
 by (*simp add: le-list-def less-list-def D-After*)
 (*metis Cons-in-T-imp-elem-ready-set D-T list.discI list.sel(1, 3)*)
 hence *False* **by** (*metis assms(2) less-list-def min-elems-no*)
 }
hence $\langle \text{ev } e \# s \in \text{min-elems } (\mathcal{D} P) \rangle$ **by** *blast*
with *le-approx3* **that** **have** $\langle \text{ev } e \# s \in \mathcal{T} Q \rangle$ **by** *blast*
thus $\langle s \in \mathcal{T} (Q \text{ after } e) \rangle$
 by (*simp add: T-After*)
 (*metis Cons-in-T-imp-elem-ready-set list.discI list.sel(1, 3)*)
qed
qed

lemma *mono-After-T* : $\langle P \sqsubseteq_T Q \implies P \text{ after } e \sqsubseteq_T Q \text{ after } e \rangle$
by (*auto simp add: trace-refine-def T-After ready-set-def*)
 (*metis list.distinct(1) list.sel(1, 3)*)

lemma *mono-After-F* :
 $\langle P \sqsubseteq_F Q \implies \text{ev } e \notin \text{ready-set } P \vee \text{ev } e \in \text{ready-set } Q \implies$
 $P \text{ after } e \sqsubseteq_F Q \text{ after } e \rangle$
using *F-subset-imp-T-subset*
by (*auto simp add: failure-refine-def F-After ready-set-def*)

lemma *mono-After-D* : $\langle P \sqsubseteq_D Q \implies P \text{ after } e \sqsubseteq_D Q \text{ after } e \rangle$
by (*auto simp add: divergence-refine-def D-After ready-set-def*)
 (*metis Cons-eq-appendI NT-ND append-self-conv2 is-processT3-ST list.collapse*
subset-iff)

lemma *mono-After-FD* :
 $\langle P \sqsubseteq_{FD} Q \implies \text{ev } e \notin \text{ready-set } P \vee \text{ev } e \in \text{ready-set } Q \implies$
 $P \text{ after } e \sqsubseteq_{FD} Q \text{ after } e \rangle$
using *F-subset-imp-T-subset*

by (*simp add: failure-divergence-refine-def le-ref-def F-After D-After ready-set-def*)
blast

lemma *mono-After-DT* : $\langle P \sqsubseteq_{DT} Q \implies P \text{ after } e \sqsubseteq_{DT} Q \text{ after } e \rangle$
by (*simp add: mono-After-D mono-After-T trace-divergence-refine-def*)

4.4 Behaviour of *After* with *STOP*, *SKIP* and \perp

lemma *After-STOP*: $\langle STOP \text{ after } e = STOP \rangle$
by (*simp add: STOP-iff-T T-After ready-set-STOP*)

lemma *After-is-STOP-iff*:
 $\langle P \text{ after } e = STOP \iff (\forall s. \text{ev } e \# s \in \mathcal{T} P \implies s = [])\rangle$
apply (*simp add: STOP-iff-T T-After ready-set-def, safe*)
apply *fastforce*
apply (*metis list.collapse*)
using *is-processT3-ST* **by** *force+*

lemma *After-SKIP*: $\langle SKIP \text{ after } e = STOP \rangle$
by (*simp add: STOP-iff-T T-After ready-set-SKIP*)

lemma *After-BOT*: $\langle \perp \text{ after } e = \perp \rangle$
by (*force simp add: BOT-iff-D D-After ready-set-BOT D-UU*)

lemma *After-is-BOT-iff*: $\langle P \text{ after } e = \perp \iff [ev\ e] \in \mathcal{D} P \rangle$
using *hd-Cons-til* **by** (*force simp add: BOT-iff-D D-After ready-set-def D-T*)

4.5 Behaviour of *After* with Operators of HOL-CSP

4.5.1 Loss of Determinism

A first interesting observation is that the *After* operator leads to the loss of determinism.

lemma *After-Mprefix-is-After-Mndetprefix*:
 $\langle (\Box a \in A \rightarrow P\ a) \text{ after } e = (\Box a \in A \rightarrow P\ a) \text{ after } e \rangle$
by (*subst Process-eq-spec*)
(force simp add: ready-set-Mprefix ready-set-Mndetprefix F-After D-After F-Mprefix D-Mprefix F-Mndetprefix D-Mndetprefix write0-def)

lemma *After-Det-is-After-Ndet*: $\langle P \sqcap Q \text{ after } e = P \sqcap Q \text{ after } e \rangle$
by (*subst Process-eq-spec*)
(auto simp add: ready-set-Det ready-set-Ndet F-After D-After F-Det F-Ndet D-Det D-Ndet)

lemma *After-Ndet*:

$\langle P \sqcap Q \text{ after } e =$
 $($
 $\text{if } ev\ e \notin \text{ready-set } P \wedge ev\ e \notin \text{ready-set } Q \text{ then } STOP$
 $\text{else if } ev\ e \in \text{ready-set } P \wedge ev\ e \in \text{ready-set } Q \text{ then } (P \text{ after } e) \sqcap (Q \text{ after } e)$
 $\text{else if } ev\ e \in \text{ready-set } P \text{ then } P \text{ after } e \text{ else } Q \text{ after } e)$
 \rangle
(is $\langle P \sqcap Q \text{ after } e =$
 $($
 $\text{if } ?c1 \text{ then } STOP \text{ else if } ?c2 \text{ then } (P \text{ after } e) \sqcap (Q \text{ after } e) \text{ else}$
 $\text{if } ?c3 \text{ then } P \text{ after } e \text{ else } Q \text{ after } e)$
 \rangle
proof –
have $\langle ?c1 \implies P \sqcap Q \text{ after } e = STOP \rangle$
by (*simp add: Process-eq-spec F-After D-After ready-set-Ndet F-STOP D-STOP*)
moreover have $\langle ?c2 \implies P \sqcap Q \text{ after } e = (P \text{ after } e) \sqcap (Q \text{ after } e) \rangle$
by (*auto simp add: Process-eq-spec F-After D-After F-Ndet D-Ndet ready-set-Ndet*)
moreover have $\langle \neg ?c2 \implies ?c3 \implies P \sqcap Q \text{ after } e = P \text{ after } e \rangle$
and $\langle \neg ?c2 \implies \neg ?c3 \implies P \sqcap Q \text{ after } e = Q \text{ after } e \rangle$
by (*auto simp add: Process-eq-spec F-After D-After F-Ndet D-Ndet ready-set-Ndet*)
 $(metis\ Cons-in-T-imp-elem-ready-set\ F-T\ list.collapse,$
 $metis\ Cons-in-T-imp-elem-ready-set\ D-T\ list.collapse)+$
ultimately show $?thesis$ **by** *presburger*
qed

lemma *After-Mprefix*: $\langle (\sqcap a \in A \rightarrow P\ a) \text{ after } e = (\text{if } e \in A \text{ then } P\ e \text{ else } STOP) \rangle$
by (*subst Process-eq-spec, auto simp add: F-After D-After ready-set-Mprefix*
 $F-Mprefix\ D-Mprefix\ F-STOP\ D-STOP$)
 $(metis\ image-eqI\ list.distinct(1)\ list.sel(1, 3))+$

lemmas *After-Det* = *After-Ndet*[*folded After-Det-is-After-Ndet*]
and *After-Mndetprefix* = *After-Mprefix*[*unfolded After-Mprefix-is-After-Mndetprefix*]
and *After-prefix* = *After-Mprefix*[*of* $\langle \{a\} \rangle \langle \lambda a. P \rangle$, *folded write0-def, simplified*]
for $a\ P$

lemma $\langle (e \rightarrow P) \text{ after } e = P \rangle$ **by** (*simp add: After-prefix*)

This result justifies seeing $P \text{ after } e$ as the reciprocal operator of the prefix $e \rightarrow P$.

However, we lose information with *After*: in general, $e \rightarrow (P \text{ after } e) \neq P$ (even when $ev\ e \in \text{ready-set } P$ and $P \neq \perp$).

lemma $\langle \exists P. (e \rightarrow (P \text{ after } e)) \neq P \rangle$

proof (*intro exI*)

show $\langle e \rightarrow (SKIP \text{ after } e) \neq SKIP \rangle$

by (*metis Par-SKIP-SKIP SKIP-Neq-STOP prefix-Par-SKIP*)

qed

lemma $\langle \exists P. ev\ e \in \text{ready-set } P \wedge (e \rightarrow (P \text{ after } e)) \neq P \rangle$

proof (*intro exI*)

show $\langle ev\ e \in \text{ready-set } \perp \wedge (e \rightarrow (\perp \text{ after } e)) \neq \perp \rangle$

by (*simp add: ready-set-BOT Mprefix-neq-BOT write0-def*)

qed

lemma $\langle \exists P. \text{ev } e \in \text{ready-set } P \wedge P \neq \perp \wedge (e \rightarrow (P \text{ after } e) \neq P) \rangle$

proof (*intro exI*)

define P **where** $P\text{-def}$: $\langle P = (e \rightarrow \text{STOP}) \sqcap \text{SKIP} \rangle$

have $*$: $\langle \text{ev } e \in \text{ready-set } P \rangle$ **by** (*simp add: P-def ready-set-Det ready-set-prefix*)

moreover have $\langle P \neq \perp \rangle$

by (*simp add: Det-is-BOT-iff Mprefix-neq-BOT P-def SKIP-neq-BOT write0-def*)

moreover have $\langle e \rightarrow (P \text{ after } e) = (e \rightarrow \text{STOP}) \rangle$

by (*rule arg-cong[where f = $\langle \lambda P. (e \rightarrow P) \rangle$]*)

(*simp add: P-def After-Det ready-set-SKIP ready-set-prefix After-prefix*)

moreover have $\langle e \rightarrow \text{STOP} \neq P \rangle$

apply (*rule contrapos-nn[of $\langle \text{ready-set } (e \rightarrow \text{STOP}) = \text{ready-set } P \rangle \langle e \rightarrow \text{STOP} = P \rangle$]*)

by (*simp add: P-def ready-set-Det ready-set-prefix ready-set-SKIP*)

(*erule arg-cong*)

ultimately show $\langle \text{ev } e \in \text{ready-set } P \wedge P \neq \perp \wedge (e \rightarrow (P \text{ after } e) \neq P) \rangle$ **by** *presburger*

qed

4.5.2 After Sequential Composition

The first goal is to obtain an equivalent of $e \rightarrow P ; Q = e \rightarrow (P ; Q)$. But in order to be exhaustive we also have to consider the possibility of Q taking the lead when $\checkmark \in \text{ready-set } P$ in the sequential composition $P ; Q$.

lemma *not-skipable-or-not-readyR-After-Seq*: $\langle (P ; Q) \text{ after } e = P \text{ after } e ; Q \rangle$

if $\langle \checkmark \notin \text{ready-set } P \vee \text{ev } e \notin \text{ready-set } Q \rangle$

proof (*cases $\langle P = \perp \rangle$*)

show $\langle P = \perp \implies (P ; Q) \text{ after } e = P \text{ after } e ; Q \rangle$

by (*simp add: BOT-Seq After-BOT*)

next

assume *non-BOT*: $\langle P \neq \perp \rangle$

show $\langle (P ; Q) \text{ after } e = P \text{ after } e ; Q \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

fix s

assume $\langle s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$

hence $*$: $\langle \text{ev } e \# s \in \mathcal{D} (P ; Q) \rangle$

by (*simp add: D-After split: if-split-asm*) (*metis list.exhaust-sel*)

then consider $\langle \text{ev } e \# s \in \mathcal{D} P \rangle$

| $\langle \exists t1 t2. \text{ev } e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{D} Q \rangle$

by (*simp add: D-Seq blast*)

thus $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \rangle$

proof *cases*

show $\langle \text{ev } e \# s \in \mathcal{D} P \implies s \in \mathcal{D} (P \text{ after } e ; Q) \rangle$

by (*simp add: D-Seq D-After*)

(*metis Cons-in-T-imp-elem-ready-set D-T list.discI list.sel(1, 3)*)

next

assume $\langle \exists t1 t2. \text{ev } e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{D} Q \rangle$

then obtain $t1\ t2$ **where** $** : \langle ev\ e\ \# \ s = t1\ @\ t2 \rangle \langle t1\ @\ [\checkmark] \in \mathcal{T}\ P \rangle \langle t2 \in \mathcal{D}\ Q \rangle$ **by** *blast*
have $\langle t1 \neq [] \rangle$ **by** (*metis ** Cons-in-T-imp-elem-ready-set D-T self-append-conv2 that*)
with $** (1)$ **obtain** $t1'$
where $\langle t1 = ev\ e\ \# \ t1' \rangle \langle s = t1' @ t2 \rangle$ **by** (*metis Cons-eq-append-conv*)
with $** (2, 3)$ **show** $\langle s \in \mathcal{D}\ (P\ after\ e ; Q) \rangle$
by (*simp add: D-Seq T-After*)
(metis Cons-in-T-imp-elem-ready-set list.distinct(1) list.sel(1, 3))
qed
next
fix s
assume $\langle s \in \mathcal{D}\ (P\ after\ e ; Q) \rangle$
hence $\langle ev\ e\ \# \ s \in \mathcal{D}\ P \vee (\exists t1\ t2. s = t1\ @\ t2 \wedge ev\ e\ \# \ t1\ @\ [\checkmark] \in \mathcal{T}\ P \wedge t2 \in \mathcal{D}\ Q) \rangle$
by (*simp add: D-Seq D-After T-After split: if-split-asm*) (*metis list.collapse*)
hence $\langle ev\ e\ \# \ s \in \mathcal{D}\ (P ; Q) \rangle$
by (*elim disjE; simp add: D-Seq*) (*metis append-Cons*)
thus $\langle s \in \mathcal{D}\ ((P ; Q)\ after\ e) \rangle$
by (*simp add: D-After*)
(metis Cons-in-T-imp-elem-ready-set D-T list.discI list.sel(1, 3))
next
fix $s\ X$
assume *same-div* : $\langle \mathcal{D}\ ((P ; Q)\ after\ e) = \mathcal{D}\ (P\ after\ e ; Q) \rangle$
assume $\langle (s, X) \in \mathcal{F}\ ((P ; Q)\ after\ e) \rangle$
then consider $\langle ev\ e \in ready\text{-}set\ (P ; Q) \rangle \langle (ev\ e\ \# \ s, X) \in \mathcal{F}\ (P ; Q) \rangle$
 $| \langle ev\ e \notin ready\text{-}set\ (P ; Q) \rangle \langle s = [] \rangle$
by (*simp add: F-After split: if-split-asm*) (*metis list.exhaust-sel*)
thus $\langle (s, X) \in \mathcal{F}\ (P\ after\ e ; Q) \rangle$
proof cases
assume *assms* : $\langle ev\ e \in ready\text{-}set\ (P ; Q) \rangle \langle (ev\ e\ \# \ s, X) \in \mathcal{F}\ (P ; Q) \rangle$
from *assms*(2) **consider** $\langle ev\ e\ \# \ s \in \mathcal{D}\ (P ; Q) \rangle$
 $| \langle (ev\ e\ \# \ s, insert\ \checkmark\ X) \in \mathcal{F}\ P \rangle \langle tickFree\ s \rangle$
 $| \langle \exists t1\ t2. ev\ e\ \# \ s = t1\ @\ t2 \wedge t1\ @\ [\checkmark] \in \mathcal{T}\ P \wedge (t2, X) \in \mathcal{F}\ Q \rangle$
by (*simp add: F-Seq D-Seq blast*)
thus $\langle (s, X) \in \mathcal{F}\ (P\ after\ e ; Q) \rangle$
proof cases
assume $\langle ev\ e\ \# \ s \in \mathcal{D}\ (P ; Q) \rangle$
hence $\langle s \in \mathcal{D}\ ((P ; Q)\ after\ e) \rangle$
by (*simp add: D-After assms(1)*) (*metis list.distinct(1) list.sel(1, 3)*)
with *same-div D-F* **show** $\langle (s, X) \in \mathcal{F}\ (P\ after\ e ; Q) \rangle$ **by** *blast*
next
show $\langle (ev\ e\ \# \ s, insert\ \checkmark\ X) \in \mathcal{F}\ P \implies tickFree\ s \implies (s, X) \in \mathcal{F}\ (P\ after\ e ; Q) \rangle$
by (*simp add: F-Seq F-After*)
(metis Cons-in-T-imp-elem-ready-set F-T list.distinct(1) list.sel(1, 3))
next
assume $\langle \exists t1\ t2. ev\ e\ \# \ s = t1\ @\ t2 \wedge t1\ @\ [\checkmark] \in \mathcal{T}\ P \wedge (t2, X) \in \mathcal{F}\ Q \rangle$
then obtain $t1\ t2$ **where** $*$: $\langle ev\ e\ \# \ s = t1\ @\ t2 \rangle \langle t1\ @\ [\checkmark] \in \mathcal{T}\ P \rangle \langle (t2,$

$X) \in \mathcal{F} Q$ by *blast*
have $\langle t1 \neq [] \rangle$ by (*metis* * *Cons-in-T-imp-elem-ready-set F-T self-append-conv2*
that)
with $*(1)$ **obtain** $t1'$
where $\langle t1 = ev e \# t1' \rangle \langle s = t1' @ t2 \rangle$ by (*metis* *Cons-eq-append-conv*)
with $*(2, 3)$ **show** $\langle (s, X) \in \mathcal{F} (P \text{ after } e ; Q) \rangle$
by (*simp add: F-Seq T-After*)
(*metis* *Cons-in-T-imp-elem-ready-set list.distinct(1) list.sel(1, 3)*)
qed
next
show $\langle ev e \notin \text{ready-set} (P ; Q) \implies s = [] \implies (s, X) \in \mathcal{F} (P \text{ after } e ; Q) \rangle$
by (*simp add: F-Seq F-After non-BOT ready-set-Seq*)
qed
next
fix $s X$
assume *same-div* : $\langle \mathcal{D} ((P ; Q) \text{ after } e) = \mathcal{D} (P \text{ after } e ; Q) \rangle$
assume $\langle (s, X) \in \mathcal{F} (P \text{ after } e ; Q) \rangle$
then consider $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \rangle$
| $\langle (s, \text{insert } \checkmark X) \in \mathcal{F} (P \text{ after } e) \rangle \langle \text{tickFree } s \rangle$
| $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} (P \text{ after } e) \wedge (t2, X) \in \mathcal{F} Q \rangle$
by (*simp add: F-Seq D-Seq blast*)
thus $\langle (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$
proof cases
from *same-div D-F* **show** $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \implies (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ by *blast*
next
from *that* **show** $\langle (s, \text{insert } \checkmark X) \in \mathcal{F} (P \text{ after } e) \implies \text{tickFree } s \implies (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$
by (*simp add: F-After ready-set-Seq F-Seq non-BOT split: if-split-asm*)
(*metis* *event.distinct(1) list.collapse tickFree-Cons*)
next
assume $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} (P \text{ after } e) \wedge (t2, X) \in \mathcal{F} Q \rangle$
then obtain $t1 t2$
where $*$: $\langle s = t1 @ t2 \rangle \langle t1 @ [\checkmark] \in \mathcal{T} (P \text{ after } e) \rangle \langle (t2, X) \in \mathcal{F} Q \rangle$ by
blast
from $*(2)$ **have** $\langle ev e \# t1 @ [\checkmark] \in \mathcal{T} P \rangle$
by (*simp add: T-After split: if-split-asm*) (*metis* *list.exhaust-sel*)
with $*(1, 3)$ **show** $\langle (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$
by (*simp add: F-After F-Seq ready-set-Seq non-BOT*)
(*metis* *Cons-in-T-imp-elem-ready-set append-Cons list.distinct(1) list.sel(1, 3)*)
qed
qed
qed

lemma $\langle (P ; \text{STOP}) \text{ after } e = P \text{ after } e ; \text{STOP} \rangle$
by (*simp add: not-skippable-or-not-readyR-After-Seq ready-set-STOP*)

lemma *skippable-not-readyL-After-Seq*: $\langle (P ; Q) \text{ after } e = Q \text{ after } e \rangle$
if $\langle \checkmark \in \text{ready-set } P \rangle$ **and** $\langle ev \ e \notin \text{ready-set } P \rangle$
proof (*cases* $\langle P = \perp \rangle$)
from *that(2) ready-set-BOT* **show** $\langle P = \perp \implies (P ; Q) \text{ after } e = Q \text{ after } e \rangle$ **by**
blast
next
assume *non-BOT* : $\langle P \neq \perp \rangle$
show $\langle (P ; Q) \text{ after } e = Q \text{ after } e \rangle$
proof (*subst Process-eq-spec-optimized, safe*)
fix *s*
assume $\langle s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$
hence $\langle ev \ e \# s \in \mathcal{D} (P ; Q) \rangle$
by (*simp add: D-After split: if-split-asm*) (*metis list.exhaust-sel*)
with *that(2)* **obtain** *t1 t2* **where** $*$: $\langle ev \ e \# s = t1 \ @ \ t2 \rangle$ $\langle t1 \ @ \ [\checkmark] \in \mathcal{T} P \rangle$
 $\langle t2 \in \mathcal{D} Q \rangle$
by (*simp add: D-Seq*) (*meson Cons-in-T-imp-elem-ready-set NT-ND*)
from $*$ (1, 2) *that(2) Cons-in-T-imp-elem-ready-set* **have** $\langle t1 = [] \rangle$
by (*cases t1; simp*) *blast*
with $*$ **show** $\langle s \in \mathcal{D} (Q \text{ after } e) \rangle$
by (*simp add: D-After*)
(*metis Cons-in-T-imp-elem-ready-set D-T list.discI list.sel(1, 3)*)
next
show $\langle s \in \mathcal{D} (Q \text{ after } e) \implies s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$ **for** *s*
by (*simp add: D-After D-Seq ready-set-Seq non-BOT that split: if-split-asm*)
(*metis append-Nil mem-Collect-eq ready-set-def that(1)*)
next
fix *s X*
assume *same-div* : $\langle \mathcal{D} ((P ; Q) \text{ after } e) = \mathcal{D} (Q \text{ after } e) \rangle$
assume $\langle (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$
then consider $\langle ev \ e \in \text{ready-set } (P ; Q) \rangle$ $\langle (ev \ e \# s, X) \in \mathcal{F} (P ; Q) \rangle$
| $\langle ev \ e \notin \text{ready-set } (P ; Q) \rangle$ $\langle s = [] \rangle$
by (*simp add: F-After split: if-split-asm*) (*metis list.exhaust-sel*)
thus $\langle (s, X) \in \mathcal{F} (Q \text{ after } e) \rangle$
proof cases
show $\langle ev \ e \in \text{ready-set } (P ; Q) \implies (ev \ e \# s, X) \in \mathcal{F} (P ; Q) \implies$
 $(s, X) \in \mathcal{F} (Q \text{ after } e) \rangle$
by (*simp add: F-After F-Seq*)
(*metis Cons-in-T-imp-elem-ready-set D-T F-T append-Nil hd-append2*
is-processT3-ST list.exhaust-sel list.sel(1, 3) that(2))
next
show $\langle ev \ e \notin \text{ready-set } (P ; Q) \implies s = [] \implies (s, X) \in \mathcal{F} (Q \text{ after } e) \rangle$
by (*simp add: F-After ready-set-Seq non-BOT that*)
qed
next
show $\langle (s, X) \in \mathcal{F} (Q \text{ after } e) \implies (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ **for** *s X*
by (*simp add: F-After F-Seq ready-set-Seq non-BOT that split: if-split-asm*)
(*metis append-Nil mem-Collect-eq ready-set-def that(1)*)
qed

qed

lemma *skippable-readyL-readyR-After-Seq*: $\langle (P ; Q) \text{ after } e = (P \text{ after } e ; Q) \sqcap (Q \text{ after } e) \rangle$

if $\langle \checkmark \in \text{ready-set } P \rangle \langle \text{ev } e \in \text{ready-set } P \rangle \langle \text{ev } e \in \text{ready-set } Q \rangle$

proof (cases $\langle P = \perp \rangle$)

show $\langle P = \perp \implies (P ; Q) \text{ after } e = (P \text{ after } e ; Q) \sqcap (Q \text{ after } e) \rangle$

by (simp add: BOT-Seq After-BOT Ndet-commute[of \perp , simplified Ndet-BOT])

next

assume *non-BOT* : $\langle P \neq \perp \rangle$

show $\langle (P ; Q) \text{ after } e = (P \text{ after } e ; Q) \sqcap (Q \text{ after } e) \rangle$

proof (subst Process-eq-spec-optimized, safe)

fix *s*

assume $\langle s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$

hence $\langle \text{ev } e \# s \in \mathcal{D} (P ; Q) \rangle$

by (simp add: D-After ready-set-Seq non-BOT that(2)) (metis list.exhaust-sel)

then consider $\langle \text{ev } e \# s \in \mathcal{D} P \rangle$

| $\langle \exists t1 t2. \text{ev } e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{D} Q \rangle$

by (simp add: D-Seq) blast

thus $\langle s \in \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$

proof cases

show $\langle \text{ev } e \# s \in \mathcal{D} P \implies s \in \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$

by (simp add: D-After D-Seq D-Ndet non-BOT that)

(metis list.distinct(1) list.sel(1, 3))

next

assume $\langle \exists t1 t2. \text{ev } e \# s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} P \wedge t2 \in \mathcal{D} Q \rangle$

then obtain *t1 t2* **where** $\ast : \langle \text{ev } e \# s = t1 @ t2 \rangle \langle t1 @ [\checkmark] \in \mathcal{T} P \rangle \langle t2 \in \mathcal{D} Q \rangle$ **by** blast

show $\langle s \in \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$

proof (cases $\langle t1 = [] \rangle$)

from $\ast(1, 3)$ **show** $\langle t1 = [] \implies s \in \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$

by (simp add: D-After D-Ndet that(3))

(metis list.distinct(1) list.sel(1, 3))

next

assume $\langle t1 \neq [] \rangle$

with $\ast(1, 3)$ **obtain** *t1'* **where** $\langle t1 = \text{ev } e \# t1' \rangle$ **by** (cases *t1*; simp)

with \ast **show** $\langle s \in \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$

by (simp add: T-After D-Seq D-Ndet that(2))

(metis list.distinct(1) list.sel(1, 3))

qed

qed

next

fix *s*

assume $\langle s \in \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$

then consider $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \rangle \mid \langle s \in \mathcal{D} (Q \text{ after } e) \rangle$

by (simp add: D-Ndet) blast

thus $\langle s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$

proof cases

show $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \implies s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$
apply (*simp add: D-After T-After D-Seq ready-set-Seq non-BOT that(2)*),
elim disjE)
by *blast* (*metis append-Cons list.distinct(1) list.exhaust-sel list.sel(1, 3)*)
next
from *that* **show** $\langle s \in \mathcal{D} (Q \text{ after } e) \implies s \in \mathcal{D} ((P ; Q) \text{ after } e) \rangle$
by (*simp add: D-After D-Seq ready-set-Seq non-BOT*)
(metis append-Nil mem-Collect-eq ready-set-def)
qed
next
fix $s X$
assume *same-div* : $\langle \mathcal{D} ((P ; Q) \text{ after } e) = \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
assume $\langle (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$
hence $\langle (ev\ e \# s, X) \in \mathcal{F} (P ; Q) \rangle$
by (*simp add: F-After ready-set-Seq non-BOT that(2)*) (*metis list.exhaust-sel*)
then consider $\langle ev\ e \# s \in \mathcal{D} P \rangle$
| $\langle (ev\ e \# s, insert\ \checkmark\ X) \in \mathcal{F} P \rangle \langle tickFree\ s \rangle$
| $\langle \exists t1\ t2. ev\ e \# s = t1\ @\ t2 \wedge t1\ @\ [\checkmark] \in \mathcal{T} P \wedge (t2, X) \in \mathcal{F} Q \rangle$
by (*simp add: F-Seq D-Seq*) *blast*
thus $\langle (s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
proof cases
assume $\langle ev\ e \# s \in \mathcal{D} P \rangle$
hence $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \rangle$
by (*simp add: D-After D-Seq that(2)*) (*metis list.distinct(1) list.sel(1, 3)*)
with *same-div* *D-Ndet D-F* **show** $\langle (s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
by *blast*
next
show $\langle (ev\ e \# s, insert\ \checkmark\ X) \in \mathcal{F} P \implies tickFree\ s \implies$
 $(s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
by (*simp add: F-Ndet F-Seq F-After that(2)*)
(metis list.distinct(1) list.sel(1, 3))
next
assume $\langle \exists t1\ t2. ev\ e \# s = t1\ @\ t2 \wedge t1\ @\ [\checkmark] \in \mathcal{T} P \wedge (t2, X) \in \mathcal{F} Q \rangle$
then obtain $t1\ t2$
where $*$: $\langle ev\ e \# s = t1\ @\ t2 \rangle \langle t1\ @\ [\checkmark] \in \mathcal{T} P \rangle \langle (t2, X) \in \mathcal{F} Q \rangle$ **by** *blast*
show $\langle (s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
proof (*cases* $\langle t1 = [] \rangle$)
from $*(1, 3)$ **show** $\langle t1 = [] \implies (s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
by (*simp add: F-Ndet F-Seq F-After that(2, 3)*)
(metis list.distinct(1) list.sel(1, 3))
next
assume $\langle t1 \neq [] \rangle$
with $*(1, 3)$ **obtain** $t1'$ **where** $\langle t1 = ev\ e \# t1' \rangle$ **by** (*cases* $t1$; *simp*)
with $*$ **show** $\langle (s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$
by (*simp add: F-Ndet F-Seq F-After T-After that(2)*)
(metis list.distinct(1) list.sel(1, 3))
qed
qed
next


```

fix s X
assume same-div :  $\langle \mathcal{D} ((P ; Q) \text{ after } e) = \mathcal{D} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$ 
assume  $\langle (s, X) \in \mathcal{F} ((P \text{ after } e ; Q) \sqcap (Q \text{ after } e)) \rangle$ 
then consider  $\langle (s, X) \in \mathcal{F} (P \text{ after } e ; Q) \rangle \mid \langle (s, X) \in \mathcal{F} (Q \text{ after } e) \rangle$ 
  by (simp add: F-Ndet) blast
thus  $\langle (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ 
proof cases
  assume  $\langle (s, X) \in \mathcal{F} (P \text{ after } e ; Q) \rangle$ 
  then consider  $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \rangle$ 
     $\mid \langle (s, \text{insert } \checkmark X) \in \mathcal{F} (P \text{ after } e) \rangle \langle \text{tickFree } s \rangle$ 
     $\mid \langle \exists t1 t2. s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} (P \text{ after } e) \wedge (t2, X) \in \mathcal{F} Q \rangle$ 
    by (simp add: F-Seq D-Seq) blast
  thus  $\langle (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ 
  proof cases
    show  $\langle s \in \mathcal{D} (P \text{ after } e ; Q) \implies (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ 
      using same-div D-Ndet D-F by blast
    next
      show  $\langle (s, \text{insert } \checkmark X) \in \mathcal{F} (P \text{ after } e) \implies \text{tickFree } s \implies (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ 
        by (simp add: F-After F-Seq ready-set-Seq that(2))
          (metis event.distinct(1) list.collapse tickFree-Cons)
      next
        show  $\langle \exists t1 t2. s = t1 @ t2 \wedge t1 @ [\checkmark] \in \mathcal{T} (P \text{ after } e) \wedge (t2, X) \in \mathcal{F} Q \implies$ 
           $(s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ 
          by (simp add: F-After T-After F-Seq ready-set-Seq non-BOT that(2))
            (metis append-Cons list.distinct(1) list.exhaust_sel list.sel(1, 3))
        qed
      next
        show  $\langle (s, X) \in \mathcal{F} (Q \text{ after } e) \implies (s, X) \in \mathcal{F} ((P ; Q) \text{ after } e) \rangle$ 
          by (simp add: F-After F-Seq ready-set-Seq that(3))
            (metis append-Nil mem-Collect-eq ready-set-def that(1))
        qed
      qed
      qed

```

lemma not-readyL-not-skippable-or-not-readyR-After-Seq:
 $\langle \text{ev } e \notin \text{ready-set } P \implies \checkmark \notin \text{ready-set } P \vee \text{ev } e \notin \text{ready-set } Q \implies$
 $(P ; Q) \text{ after } e = \text{STOP} \rangle$
by (simp add: not-skippable-or-not-readyR-After-Seq STOP-Seq not-ready-After)

lemma After-Seq:
 $\langle (P ; Q) \text{ after } e =$
 ($\text{if } \text{ev } e \notin \text{ready-set } P \wedge \text{ev } e \notin \text{ready-set } Q \text{ then } \text{STOP}$
 $\text{else if } \text{ev } e \notin \text{ready-set } Q \text{ then } P \text{ after } e ; Q$
 $\text{else if } \text{ev } e \notin \text{ready-set } P \text{ then if } \checkmark \in \text{ready-set } P \text{ then } Q \text{ after } e \text{ else } \text{STOP}$
 $\text{else if } \checkmark \in \text{ready-set } P \text{ then } (P \text{ after } e ; Q) \sqcap (Q \text{ after } e) \text{ else } P \text{ after } e ; Q \rangle$

by (*simp add: STOP-Seq not-ready-After not-skippable-or-not-readyR-After-Seq
skippable-not-readyL-After-Seq skippable-readyL-readyR-After-Seq*)

4.5.3 After Synchronization

Now let's focus on *Sync*. We want to obtain an equivalent of

$$\llbracket ?A \cap ?S = \{\}; ?A' \subseteq ?S; ?B \cap ?S = \{\}; ?B' \subseteq ?S \rrbracket \implies Mprefix (?A \cup ?A') ?P \llbracket ?S \rrbracket Mprefix (?B \cup ?B') ?Q = (\Box x \in ?A \rightarrow ?P x \llbracket ?S \rrbracket Mprefix (?B \cup ?B') ?Q) \Box (\Box y \in ?B \rightarrow Mprefix (?A \cup ?A') ?P \llbracket ?S \rrbracket ?Q y) \Box (\Box x \in ?A' \cap ?B' \rightarrow ?P x \llbracket ?S \rrbracket ?Q x)$$

We will also divide the task.

After version of

$$\llbracket e \notin ?S; ?B' \subseteq ?S \rrbracket \implies e \rightarrow P \llbracket ?S \rrbracket Mprefix ?B' ?Q = e \rightarrow (P \llbracket ?S \rrbracket Mprefix ?B' ?Q).$$

lemma *tickFree-tl*: $\langle tickFree s \implies tickFree(tl s) \rangle$

— Remove this lemma, already in future versions of HOL-CSP.

by (*metis Nil-tl tickFree-tl*)

lemma *readyL-not-readyR-not-in-After-Sync*:

$\langle (P \llbracket S \rrbracket Q) \text{ after } e = P \text{ after } e \llbracket S \rrbracket Q \rangle$

if *ready-hyps*: $\langle ev e \in \text{ready-set } P \rangle \langle ev e \notin \text{ready-set } Q \rangle$ and *notin*: $\langle e \notin S \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

```

{ fix s X
  assume assms :  $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle (ev e \# s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$ 
    and same-div :  $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$ 
  have  $\langle (s, X) \in \mathcal{F} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$ 
  proof (cases  $\langle ev e \# s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ )
    case True
      hence  $\langle s \in \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$ 
      by (force simp add: D-After ready-set-Sync ready-hyps(1) assms(1, 2) notin)
      thus  $\langle (s, X) \in \mathcal{F} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$  using NF-ND same-div by blast
    next
      case False
      with assms(3) obtain s-P s-Q X-P X-Q
      where * :  $\langle (s-P, X-P) \in \mathcal{F} P \rangle \langle (s-Q, X-Q) \in \mathcal{F} Q \rangle$ 
         $\langle (ev e \# s) \text{ setinterleaves } ((s-P, s-Q), \text{insert } \checkmark (ev 'S)) \rangle$ 
         $\langle X = (X-P \cup X-Q) \cap \text{insert } \checkmark (ev 'S) \cup X-P \cap X-Q \rangle$ 
      by (simp add: F-Sync D-Sync) blast
      have ** :  $\langle s-P \neq \perp \wedge hd s-P = ev e \wedge s \text{ setinterleaves } ((tl s-P, s-Q), \text{insert } \checkmark (ev 'S)) \rangle$ 
      using *(3) by (cases s-P; cases s-Q, auto split: if-split-asm)
        (metis *(2) After-is-STOP-iff CollectI F-T not-ready-After  
ready-set-def ready-hyps(2))
      hence  $\langle (tl s-P, X-P) \in \mathcal{F} (P \text{ after } e) \wedge (s-Q, X-Q) \in \mathcal{F} Q \wedge$ 
         $s \text{ setinterleaves } ((tl s-P, s-Q), \text{insert } \checkmark (ev 'S)) \wedge$ 
         $X = (X-P \cup X-Q) \cap \text{insert } \checkmark (ev 'S) \cup X-P \cap X-Q \rangle$ 

```

```

    apply (simp add: F-After ** ready-hyps(1))
    using *(1, 2, 4) ** by blast
  thus ⟨(s, X) ∈  $\mathcal{F}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
    by (simp add: F-Sync) blast
qed
} note * = this

show ⟨(s, X) ∈  $\mathcal{F}$  ((P  $\llbracket S \rrbracket$  Q) after e) ⇒ (s, X) ∈  $\mathcal{F}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
  if same-div : ⟨ $\mathcal{D}$  ((P  $\llbracket S \rrbracket$  Q) after e) =  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q)⟩ for s X
  apply (simp add: F-After ready-set-Sync ready-hyps notin image-iff
    split: if-split-asm)
  apply (erule disjE;
    simp add: After-BOT Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT
    F-UU,
    metis butlast-tl front-tickFree-butlast tickFree-tl)
  by (metis * list.exhaust-sel same-div)
next

fix s X
assume same-div : ⟨ $\mathcal{D}$  ((P  $\llbracket S \rrbracket$  Q) after e) =  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
{ assume asms : ⟨P ≠  $\perp$ ⟩ ⟨Q ≠  $\perp$ ⟩ ⟨(s, X) ∈  $\mathcal{F}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
  from asms(3) consider
    ⟨ $\exists$  s-P s-Q X-P X-Q. (s-P, X-P) ∈  $\mathcal{F}$  (P after e) ∧ (s-Q, X-Q) ∈  $\mathcal{F}$  Q ∧
      s setinterleaves ((s-P, s-Q), insert ✓ (ev ' S)) ∧
      X = (X-P ∪ X-Q) ∩ insert ✓ (ev ' S) ∪ X-P ∩ X-Q⟩ |
    ⟨s ∈  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
  by (simp add: F-Sync D-Sync) blast
hence ⟨(ev e # s, X) ∈  $\mathcal{F}$  (P  $\llbracket S \rrbracket$  Q)⟩
proof cases
case 1
then obtain s-P s-Q X-P X-Q
  where * : ⟨(ev e # s-P, X-P) ∈  $\mathcal{F}$  P⟩ ⟨(s-Q, X-Q) ∈  $\mathcal{F}$  Q⟩
    ⟨s setinterleaves ((s-P, s-Q), insert ✓ (ev ' S))⟩
    ⟨X = (X-P ∪ X-Q) ∩ insert ✓ (ev ' S) ∪ X-P ∩ X-Q⟩
  by (simp add: F-After ready-hyps(1)) (metis list.collapse)
have ⟨(s-Q, X-Q) ∈  $\mathcal{F}$  Q ∧
  (ev e # s) setinterleaves ((ev e # s-P, s-Q), insert ✓ (ev ' S)) ∧
  X = (X-P ∪ X-Q) ∩ insert ✓ (ev ' S) ∪ X-P ∩ X-Q⟩
  apply (simp add: *(2, 4))
  using *(3) by (cases s-Q; simp add: notin image-iff)
with *(1) show ⟨(ev e # s, X) ∈  $\mathcal{F}$  (P  $\llbracket S \rrbracket$  Q)⟩
  by (simp add: F-Sync) blast
next
case 2
from 2[simplified same-div[symmetric]]
have ⟨ev e # s ∈  $\mathcal{D}$  (P  $\llbracket S \rrbracket$  Q)⟩
  by (simp add: D-After ready-hyps(1) ready-set-Sync
    asms(1, 2) notin image-iff, metis list.collapse)
thus ⟨(ev e # s, X) ∈  $\mathcal{F}$  (P  $\llbracket S \rrbracket$  Q)⟩ using NF-ND by blast

```

```

    qed
  }
  thus ⟨s, X⟩ ∈  $\mathcal{F}$  (P after e  $\llbracket S \rrbracket$  Q)  $\implies$  ⟨s, X⟩ ∈  $\mathcal{F}$  ((P  $\llbracket S \rrbracket$  Q) after e)
  by (simp add: F-After ready-set-Sync ready-hyps After-BOT F-UU image-iff
      Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT notin)
      (metis butlast.simps(2) event.distinct(1) front-tickFree-butlast front-tickFree-single
      list.distinct(1) list.sel(1, 3) process-chnr tickFree-Cons)
next

{ fix s
  assume assms : ⟨P ≠  $\perp$ ⟩ ⟨Q ≠  $\perp$ ⟩ ⟨ev e # s ∈  $\mathcal{D}$  (P  $\llbracket S \rrbracket$  Q)⟩
  from assms(3) obtain t u r v
  where * : ⟨front-tickFree v⟩ ⟨tickFree r ∨ v = []⟩ ⟨ev e # s = r @ v⟩
          ⟨r setinterleaves ((t, u), insert ✓ (ev ' S))⟩
          ⟨t ∈  $\mathcal{D}$  P ∧ u ∈  $\mathcal{T}$  Q ∨ t ∈  $\mathcal{D}$  Q ∧ u ∈  $\mathcal{T}$  P⟩ by (simp add: D-Sync)
blast
  have ** : ⟨r ≠ [] ∧ hd r = ev e⟩
  by (metis *(3, 4, 5) BOT-iff-D assms(1, 2) empty-setinterleaving hd-append
  list.sel(1))
  hence *** : ⟨(t ∈  $\mathcal{D}$  P ∧ u ∈  $\mathcal{T}$  Q  $\longrightarrow$  t ≠ [] ∧ hd t = ev e ∧
              tl r setinterleaves ((tl t, u), insert ✓ (ev ' S))) ∧
              (t ∈  $\mathcal{D}$  Q ∧ u ∈  $\mathcal{T}$  P  $\longrightarrow$  u ≠ [] ∧ hd u = ev e ∧
              tl r setinterleaves ((t, tl u), insert ✓ (ev ' S)))⟩
  using *(4) assms(1, 2)[simplified BOT-iff-D] ready-hyps[simplified ready-set-def]
  apply (cases t; cases u; simp split: if-split-asm)
  by (safe; simp; metis [[metis-verbose = false]] After-is-STOP-iff D-T
      not-ready-After ready-hyps(2))+
  from *(5) have ⟨s ∈  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
  proof (elim disjE)
    assume ⟨t ∈  $\mathcal{D}$  P ∧ u ∈  $\mathcal{T}$  Q⟩
    hence ⟨front-tickFree v ∧ (tickFree (tl r) ∨ v = []) ∧ s = tl r @ v ∧
          tl r setinterleaves ((tl t, u), insert ✓ (ev ' S)) ∧
          tl t ∈  $\mathcal{D}$  (P after e) ∧ u ∈  $\mathcal{T}$  Q⟩
    by (simp add: D-After ready-hyps,
        metis *(1, 2, 3) ** *** list.sel(3) tickFree-tl tl-append2)
    thus ⟨s ∈  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q)⟩ by (simp add: D-Sync) blast
  next
    assume ⟨t ∈  $\mathcal{D}$  Q ∧ u ∈  $\mathcal{T}$  P⟩
    hence ⟨front-tickFree v ∧ (tickFree (tl r) ∨ v = []) ∧ s = tl r @ v ∧
          tl r setinterleaves ((t, tl u), insert ✓ (ev ' S)) ∧
          t ∈  $\mathcal{D}$  Q ∧ tl u ∈  $\mathcal{T}$  (P after e)⟩
    by (simp add: D-After T-After ready-hyps,
        metis *(1, 2, 3) ** *** list.sel(3) tickFree-tl tl-append2)
    thus ⟨s ∈  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q)⟩
    by (simp add: D-Sync) blast
  qed
} note * = this

show ⟨s ∈  $\mathcal{D}$  ((P  $\llbracket S \rrbracket$  Q) after e)  $\implies$  s ∈  $\mathcal{D}$  (P after e  $\llbracket S \rrbracket$  Q) for s

```

```

apply (simp add: D-After ready-set-Sync ready-hyps notin image-iff
        split: if-split-asm)
apply (erule disjE;
        simp add: After-BOT Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT
        D-UU,
        metis butlast-tl front-tickFree-butlast tickFree-tl)
by (metis * list.collapse)
next

fix s
{ assume assms :  $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle s \in \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$ 
  from assms(3) obtain t u r v
    where * :  $\langle \text{front-tickFree } v \rangle \langle \text{tickFree } r \vee v = [] \rangle \langle s = r @ v \rangle$ 
       $\langle r \text{ setinterleaves } ((t, u), \text{insert } \checkmark (ev \text{ ' } S)) \rangle$ 
       $\langle t \in \mathcal{D} (P \text{ after } e) \wedge u \in \mathcal{T} Q \vee t \in \mathcal{D} Q \wedge u \in \mathcal{T} (P \text{ after } e) \rangle$ 
    by (simp add: D-Sync) blast
  from *(5) have  $\langle ev \ e \ \# \ s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ 
  proof (elim disjE)
    assume  $\langle t \in \mathcal{D} (P \text{ after } e) \wedge u \in \mathcal{T} Q \rangle$ 
    with *(1, 2, 3, 4) ready-hyps(1)
    have ** :  $\langle \text{front-tickFree } v \wedge (\text{tickFree } (ev \ e \ \# \ r) \vee v = []) \wedge s = r @ v \wedge$ 
       $(ev \ e \ \# \ r) \text{ setinterleaves } ((t, ev \ e \ \# \ u), \text{insert } \checkmark (ev \text{ ' } S)) \wedge$ 
       $ev \ e \ \# \ t \in \mathcal{D} P \wedge u \in \mathcal{T} Q \rangle$ 
    by (cases u; simp add: D-After notin image-iff, metis list.collapse)
    show  $\langle ev \ e \ \# \ s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ 
    by (simp add: D-Sync) (metis ** Cons-eq-appendI)
  next
    assume  $\langle t \in \mathcal{D} Q \wedge u \in \mathcal{T} (P \text{ after } e) \rangle$ 
    with *(1, 2, 3, 4) ready-hyps(1)
    have ** :  $\langle \text{front-tickFree } v \wedge (\text{tickFree } (ev \ e \ \# \ r) \vee v = []) \wedge s = r @ v \wedge$ 
       $(ev \ e \ \# \ r) \text{ setinterleaves } ((t, ev \ e \ \# \ u), \text{insert } \checkmark (ev \text{ ' } S)) \wedge$ 
       $t \in \mathcal{D} Q \wedge ev \ e \ \# \ u \in \mathcal{T} P \rangle$ 
    by (cases t; simp add: T-After notin image-iff, metis list.collapse)
    show  $\langle ev \ e \ \# \ s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ 
    by (simp add: D-Sync) (metis ** Cons-eq-appendI)
  qed
}
thus  $\langle s \in \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q) \implies s \in \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$ 
by (simp add: D-After ready-set-Sync ready-hyps After-BOT D-UU
        Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT notin
        image-iff)
  (metis D-imp-front-tickFree butlast.simps(2) event.distinct(1) front-tickFree-butlast
        list.discI list.sel(1, 3) tickFree-Cons tickFree-butlast)
qed

```

lemma *not-readyL-in-After-Sync*:

$\langle ev\ e \notin ready\text{-}set\ P \implies e \in S \implies$
 $(P \llbracket S \rrbracket Q) \text{ after } e = (\text{if } Q = \perp \text{ then } \perp \text{ else } STOP)\rangle$
apply (*simp*, *intro conjI impI*)
by (*simp add: BOT-iff-D D-After Sync-BOT ready-set-BOT D-UU*,
metis front-tickFree-single list.sel(1) list.sel(3) not-Cons-self)
(auto simp add: STOP-iff-T T-After ready-set-BOT ready-set-Sync)

After version of $\llbracket e \in ?S; e \in ?S \rrbracket \implies e \rightarrow P \llbracket ?S \rrbracket e \rightarrow Q = e \rightarrow (P \llbracket ?S \rrbracket Q)$.

lemma *readyL-readyR-in-After-Sync*:

$\langle (P \llbracket S \rrbracket Q) \text{ after } e = P \text{ after } e \llbracket S \rrbracket Q \text{ after } e \rangle$

if *ready-hyps*: $\langle ev\ e \in ready\text{-}set\ P \rangle$ $\langle ev\ e \in ready\text{-}set\ Q \rangle$ **and** *inside*: $\langle e \in S \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

{ fix $s\ X$
assume *assms* : $\langle P \neq \perp \rangle$ $\langle Q \neq \perp \rangle$ $\langle (ev\ e \# s, X) \in \mathcal{F}\ (P \llbracket S \rrbracket Q) \rangle$
and *same-div* : $\langle \mathcal{D}\ ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D}\ (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$
from *assms*(3) **consider**
 $\langle \exists s\text{-}P\ s\text{-}Q\ X\text{-}P\ X\text{-}Q. (s\text{-}P, X\text{-}P) \in \mathcal{F}\ P \wedge (s\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \wedge$
 $(ev\ e \# s) \text{ setinterleaves } ((s\text{-}P, s\text{-}Q), \text{insert } \checkmark (ev\ 'S)) \wedge$
 $X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark (ev\ 'S) \cup X\text{-}P \cap X\text{-}Q \rangle$
 $\langle ev\ e \# s \in \mathcal{D}\ (P \llbracket S \rrbracket Q) \rangle$
by (*simp add: F-Sync D-Sync*) *blast*
hence $\langle (s, X) \in \mathcal{F}\ (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$
proof *cases*
case 1
then obtain $s\text{-}P\ s\text{-}Q\ X\text{-}P\ X\text{-}Q$
where $*$: $\langle (s\text{-}P, X\text{-}P) \in \mathcal{F}\ P \rangle$ $\langle (s\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \rangle$
 $\langle (ev\ e \# s) \text{ setinterleaves } ((s\text{-}P, s\text{-}Q), \text{insert } \checkmark (ev\ 'S)) \rangle$
 $\langle X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark (ev\ 'S) \cup X\text{-}P \cap X\text{-}Q \rangle$ **by** *blast*
from $*$ (3) **have** $\langle s\text{-}P \neq [] \wedge hd\ s\text{-}P = ev\ e \wedge s\text{-}Q \neq [] \wedge hd\ s\text{-}Q = ev\ e \wedge$
 $s \text{ setinterleaves } ((tl\ s\text{-}P, tl\ s\text{-}Q), \text{insert } \checkmark (ev\ 'S)) \rangle$
using *inside* **by** (*cases s-P; cases s-Q, auto simp add: split: if-split-asm*)
hence $\langle (tl\ s\text{-}P, X\text{-}P) \in \mathcal{F}\ (P \text{ after } e) \wedge (tl\ s\text{-}Q, X\text{-}Q) \in \mathcal{F}\ (Q \text{ after } e) \wedge$
 $s \text{ setinterleaves } ((tl\ s\text{-}P, tl\ s\text{-}Q), \text{insert } \checkmark (ev\ 'S)) \wedge$
 $X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark (ev\ 'S) \cup X\text{-}P \cap X\text{-}Q \rangle$
using $*$ (1, 2, 4) *ready-hyps* **by** (*simp add: F-After*) *blast*
thus $\langle (s, X) \in \mathcal{F}\ (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$
by (*simp add: F-Sync*) *blast*
next
assume $\langle ev\ e \# s \in \mathcal{D}\ (P \llbracket S \rrbracket Q) \rangle$
hence $\langle s \in \mathcal{D}\ ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$
by (*force simp add: D-After ready-set-Sync ready-hyps assms(1, 2)*)
from *this*[*simplified same-div*]
show $\langle (s, X) \in \mathcal{F}\ (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$ **using** *NF-ND* **by** *blast*
qed
} note $*$ = *this*

show $\langle (s, X) \in \mathcal{F}\ ((P \llbracket S \rrbracket Q) \text{ after } e) \implies (s, X) \in \mathcal{F}\ (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$

```

if same-div :  $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$  for  $s \ X$ 
apply (simp add: F-After ready-set-Sync ready-hyps split: if-split-asm)
  apply (erule disjE;
    simp add: After-BOT Sync-commute[of  $\perp$ , simplified Sync-BOT])
Sync-BOT F-UU,
    metis butlast-tl front-tickFree-butlast tickFree-tl)
  by (metis * list.exhaust-sel same-div)
next

fix  $s \ X$ 
assume same-div :  $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$ 
{ assume assms :  $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle (s, X) \in \mathcal{F} (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$ 
  from assms(3) consider
     $\langle \exists s\text{-}P \ s\text{-}Q \ X\text{-}P \ X\text{-}Q. (ev \ e \ \# \ s\text{-}P, X\text{-}P) \in \mathcal{F} \ P \wedge (ev \ e \ \# \ s\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q$ 
 $\wedge$ 
     $s \ \text{setinterleaves} ((s\text{-}P, s\text{-}Q), \text{insert } \checkmark (ev \ ' \ S)) \wedge$ 
 $X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark (ev \ ' \ S) \cup X\text{-}P \cap X\text{-}Q \rangle$  |
     $\langle s \in \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$ 
  by (auto simp add: F-Sync D-Sync F-After ready-hyps) (metis list.collapse)
hence  $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$ 
proof cases
  case 1
  then obtain  $s\text{-}P \ s\text{-}Q \ X\text{-}P \ X\text{-}Q$ 
    where  $*$  :  $\langle (ev \ e \ \# \ s\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle \langle (ev \ e \ \# \ s\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle$ 
     $\langle s \ \text{setinterleaves} ((s\text{-}P, s\text{-}Q), \text{insert } \checkmark (ev \ ' \ S)) \rangle$ 
     $\langle X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark (ev \ ' \ S) \cup X\text{-}P \cap X\text{-}Q \rangle$  by blast
  have  $**$  :  $\langle (ev \ e \ \# \ s) \ \text{setinterleaves} ((ev \ e \ \# \ s\text{-}P, ev \ e \ \# \ s\text{-}Q), \text{insert } \checkmark (ev \ ' \ S)) \rangle$ 
    by (simp add: inside *(3))
  show  $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$ 
  apply (simp add: F-Sync)
  using  $*(1, 2, 4) **$  by blast
next
  assume  $\langle s \in \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \rangle$ 
  with same-div[symmetric] have  $\langle ev \ e \ \# \ s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ 
    by (simp add: D-After ready-hyps ready-set-Sync assms(1, 2)) (metis list.collapse)
  with D-F show  $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$  by blast
  qed
}
thus  $\langle (s, X) \in \mathcal{F} (P \text{ after } e \llbracket S \rrbracket Q \text{ after } e) \implies (s, X) \in \mathcal{F} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$ 
by (simp add: F-After ready-set-Sync ready-hyps After-BOT F-UU
  Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT)
  (metis butlast.simps(2) event.distinct(1) front-tickFree-butlast is-processT2
  list.distinct(1) list.sel(1, 3) tickFree-Cons tickFree-butlast)
next

{ fix  $s$ 
  assume assms :  $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle ev \ e \ \# \ s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ 

```

from $assms(3)$ **obtain** $t\ u\ r\ v$
where $*$: $\langle front\ tickFree\ v \rangle \langle tickFree\ r \vee v = [] \rangle \langle ev\ e \# s = r @ v \rangle$
 $\langle r\ setinterleaves\ ((t, u), insert\ \checkmark\ (ev\ 'S)) \rangle$
 $\langle t \in \mathcal{D}\ P \wedge u \in \mathcal{T}\ Q \vee t \in \mathcal{D}\ Q \wedge u \in \mathcal{T}\ P \rangle$ **by** ($simp\ add: D-Sync$)

$blast$
have $**$: $\langle r \neq [] \wedge hd\ r = ev\ e \wedge t \neq [] \wedge hd\ t = ev\ e \wedge u \neq [] \wedge hd\ u = ev\ e$
 \wedge
 $tl\ r\ setinterleaves\ ((tl\ t, tl\ u), insert\ \checkmark\ (ev\ 'S)) \rangle$
using $*(3, 4, 5)$ *inside* $assms(1, 2)[simplified\ BOT-iff-D]$
by ($cases\ t$; $cases\ u$; $force\ split: if-split-asm$)
have $\langle tickFree\ (tl\ r) \vee v = [] \rangle \wedge s = tl\ r @ v \wedge$
 $(tl\ t \in \mathcal{D}\ (P\ after\ e) \wedge tl\ u \in \mathcal{T}\ (Q\ after\ e) \vee$
 $tl\ t \in \mathcal{D}\ (Q\ after\ e) \wedge tl\ u \in \mathcal{T}\ (P\ after\ e)) \rangle$
using $*(2, 3, 5)$ **** apply** ($simp\ add: D-After\ ready-hyps\ T-After$)
by ($metis\ tickFree-tl\ list.sel(3)\ tl-append2$)
with $*(1)$ **** have** $\langle s \in \mathcal{D}\ (P\ after\ e\ [S])\ Q\ after\ e \rangle$ **by** ($simp\ add: D-Sync$)

$blast$
} note $*$ = *this*

show $\langle s \in \mathcal{D}\ ((P\ [S])\ Q)\ after\ e \rangle \implies s \in \mathcal{D}\ (P\ after\ e\ [S])\ Q\ after\ e \rangle$ **for** s
apply ($simp\ add: D-After\ ready-set-Sync\ ready-hyps\ After-BOT\ D-UU$
 $split: if-split-asm$)
apply ($erule\ disjE$;
 $simp\ add: After-BOT\ Sync-commute[of\ \perp, simplified\ Sync-BOT]\ Sync-BOT$
 $D-UU,$
 $metis\ butlast-tl\ front-tickFree-butlast\ tickFree-tl$)
by ($metis\ * list.exhaust-sel$)

next

fix s
{ assume $\langle s \in \mathcal{D}\ (P\ after\ e\ [S])\ Q\ after\ e \rangle$
from *this* **obtain** $t\ u\ r\ v$
where $*$: $\langle front\ tickFree\ v \rangle \langle tickFree\ r \vee v = [] \rangle \langle s = r @ v \rangle$
 $\langle r\ setinterleaves\ ((t, u), insert\ \checkmark\ (ev\ 'S)) \rangle$
 $\langle ev\ e \# t \in \mathcal{D}\ P \wedge ev\ e \# u \in \mathcal{T}\ Q \vee ev\ e \# t \in \mathcal{D}\ Q \wedge ev\ e \# u \in$
 $\mathcal{T}\ P \rangle$
by ($simp\ add: D-Sync\ D-After\ T-After\ ready-hyps$) ($metis\ list.collapse$)
have $**$: $\langle ev\ e \# r \rangle setinterleaves\ ((ev\ e \# t, ev\ e \# u), insert\ \checkmark\ (ev\ 'S)) \rangle$
by ($simp\ add: inside\ *(4)$)
have $\langle ev\ e \# s \in \mathcal{D}\ (P\ [S])\ Q \rangle$
by ($simp\ add: D-Sync\ inside$)
 $(metis\ *(1, 2, 3, 5)\ **\ append-Cons\ event.distinct(1)\ tickFree-Cons)$
}

thus $\langle s \in \mathcal{D}\ (P\ after\ e\ [S])\ Q\ after\ e \rangle \implies s \in \mathcal{D}\ ((P\ [S])\ Q)\ after\ e \rangle$
by ($simp\ add: D-After\ ready-set-Sync\ ready-hyps\ After-BOT\ D-UU$
 $Sync-commute[of\ \perp, simplified\ Sync-BOT]\ Sync-BOT\ inside$)
 $(metis\ D-imp-front-tickFree\ list.distinct(1)\ list.sel(1, 3))$

qed

After version of

$\llbracket e \notin ?S; e \notin ?S \rrbracket \implies e \rightarrow P \llbracket ?S \rrbracket e \rightarrow Q = (e \rightarrow (P \llbracket ?S \rrbracket e \rightarrow Q)) \sqcap (e \rightarrow (e \rightarrow P \llbracket ?S \rrbracket Q)).$

lemma *readyL-readyR-not-in-After-Sync:*

$\langle (P \llbracket S \rrbracket Q) \text{ after } e = (P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e) \rangle$

if *ready-hyps*: $\langle \text{ev } e \in \text{ready-set } P \rangle \langle \text{ev } e \in \text{ready-set } Q \rangle$ **and** *notin*: $\langle e \notin S \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

{ **fix** $P Q s X s-P s-Q X-P X-Q$

assume *assms* : $\langle (s-P, X-P) \in \mathcal{F} P \rangle \langle (s-Q, X-Q) \in \mathcal{F} Q \rangle$

$\langle X = (X-P \cup X-Q) \cap \text{insert } \checkmark (ev \text{ ' } S) \cup X-P \cap X-Q \rangle$

$\langle s-P \neq [] \rangle \langle \text{hd } s-P = \text{ev } e \rangle$

$\langle s \text{ setinterleaves } ((\text{tl } s-P, s-Q), \text{insert } \checkmark (ev \text{ ' } S)) \rangle$

$\langle \text{ev } e \in \text{ready-set } P \rangle$

from *assms*(1, 4, 5, 7) **have** $\langle (\text{tl } s-P, X-P) \in \mathcal{F} (P \text{ after } e) \rangle$

by (*simp add: F-After*) *blast*

with *assms*(2, 3, 6) **have** $\langle (s, X) \in \mathcal{F} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$

by (*simp add: F-Sync*) *blast*

} **note** * = *this*

{ **fix** $s X$

assume *assms* : $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle (\text{ev } e \# s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$

and *same-div* : $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$

from *assms*(3) **consider**

$\langle \exists s-P s-Q X-P X-Q. (s-P, X-P) \in \mathcal{F} P \wedge (s-Q, X-Q) \in \mathcal{F} Q \wedge$

$(\text{ev } e \# s) \text{ setinterleaves } ((s-P, s-Q), \text{insert } \checkmark (ev \text{ ' } S)) \wedge$

$X = (X-P \cup X-Q) \cap \text{insert } \checkmark (ev \text{ ' } S) \cup X-P \cap X-Q \rangle$

$\langle s \in \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$

by (*simp add: F-Sync D-After D-Sync ready-set-Sync assms*(1, 2) *ready-hyps*)

(*metis* (*no-types, opaque-lifting*) *list.distinct*(1) *list.sel*(1, 3))

hence $\langle (s, X) \in \mathcal{F} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$

proof *cases*

case 1

then obtain $s-P s-Q X-P X-Q$

where ** : $\langle (s-P, X-P) \in \mathcal{F} P \rangle \langle (s-Q, X-Q) \in \mathcal{F} Q \rangle$

$\langle (\text{ev } e \# s) \text{ setinterleaves } ((s-P, s-Q), \text{insert } \checkmark (ev \text{ ' } S)) \rangle$

$\langle X = (X-P \cup X-Q) \cap \text{insert } \checkmark (ev \text{ ' } S) \cup X-P \cap X-Q \rangle$ **by** *blast*

have $\langle s-P \neq [] \wedge \text{hd } s-P = \text{ev } e \wedge s \text{ setinterleaves } ((\text{tl } s-P, s-Q), \text{insert } \checkmark (ev \text{ ' } S)) \vee$

$s-Q \neq [] \wedge \text{hd } s-Q = \text{ev } e \wedge s \text{ setinterleaves } ((s-P, \text{tl } s-Q), \text{insert } \checkmark (ev \text{ ' } S)) \rangle$

using ** (3) **by** (*cases* *s-P*; *cases* *s-Q*; *simp add: notin image-iff split: if-split-asm*) *blast*

thus $\langle (s, X) \in \mathcal{F} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$

apply (*elim disjE*; *simp add: F-Ndet*)

subgoal using * ** (1, 2, 4) *ready-hyps*(1) **by** *blast*

apply (*rule disjI2, subst Sync-commute, rule* *[*OF* ** (2, 1)])

by (*simp-all add: ***(4) *Int-commute Un-commute Sync-commute Sync.sym ready-hyps*(2))

```

next
  assume  $\langle s \in \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$ 
  from this[simplified same-div]
  show  $\langle (s, X) \in \mathcal{F} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$  using NF-ND
by blast
  qed
} note ** = this

show  $\langle (s, X) \in \mathcal{F} ((P \llbracket S \rrbracket Q) \text{ after } e) \implies (s, X) \in \mathcal{F} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$ 
  if same-div:  $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$  for s X
  apply (simp add: F-After ready-set-Sync ready-hyps split: if-split-asm)
  apply (erule disjE;
    simp add: After-BOT Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT F-UU Ndet-id,
    metis butlast-tl front-tickFree-butlast tickFree-tl)
  by (metis ** same-div list.exhaust-sel)
next

{ fix s X P Q
  assume assms :  $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle (s, X) \in \mathcal{F} (P \text{ after } e \llbracket S \rrbracket Q) \rangle \langle ev \ e \in \text{ready-set } P \rangle$ 
  and same-div :  $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$ 
  from assms(3)[simplified F-Sync, simplified] consider
     $\langle \exists s\text{-}P \ s\text{-}Q \ X\text{-}P \ X\text{-}Q. (ev \ e \ \# \ s\text{-}P, X\text{-}P) \in \mathcal{F} \ P \wedge (s\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \wedge$ 
       $s \ \text{setinterleaves} \ ((s\text{-}P, s\text{-}Q), \text{insert } \checkmark \ (ev \ ' \ S)) \wedge$ 
       $X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark \ (ev \ ' \ S) \cup X\text{-}P \cap X\text{-}Q \rangle$  |
     $\langle s \in \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$ 
  by (simp add: F-Sync F-After assms(4) D-Sync)
    (metis (no-types, opaque-lifting) list.exhaust-sel)
  hence  $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$ 
proof cases
  case 1
  then obtain s-P s-Q X-P X-Q
  where * :  $\langle (ev \ e \ \# \ s\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle \langle (s\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle$ 
     $\langle s \ \text{setinterleaves} \ ((s\text{-}P, s\text{-}Q), \text{insert } \checkmark \ (ev \ ' \ S)) \rangle$ 
     $\langle X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert } \checkmark \ (ev \ ' \ S) \cup X\text{-}P \cap X\text{-}Q \rangle$  by blast
  have  $\langle (ev \ e \ \# \ s) \ \text{setinterleaves} \ ((ev \ e \ \# \ s\text{-}P, s\text{-}Q), \text{insert } \checkmark \ (ev \ ' \ S)) \rangle$ 
    using *(3) by (cases s-Q; simp add: notin image-iff)
  with *(1, 2, 4) show  $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$ 
    by (simp add: F-Sync) blast
next
  assume  $\langle s \in \mathcal{D} (P \text{ after } e \llbracket S \rrbracket Q) \rangle$ 
  hence  $\langle s \in \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$  using same-div[simplified D-Ndet] by fast
  hence  $\langle (s, X) \in \mathcal{F} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$  using process-cha by blast
  thus  $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} (P \llbracket S \rrbracket Q) \rangle$ 
    by (simp add: F-After ready-set-Sync assms(1, 2, 4) notin image-iff)

```

```

      (metis list.collapse)
    qed
  } note * = this

  show  $\langle (s, X) \in \mathcal{F} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \implies (s, X) \in \mathcal{F} ((P \llbracket S \rrbracket Q) \text{ after } e) \rangle$ 
  if same-div :  $\langle \mathcal{D} ((P \llbracket S \rrbracket Q) \text{ after } e) = \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$  for s X
  apply (simp add: F-After ready-set-Sync ready-hyps After-BOT F-UU
    Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT Ndet-id)
  apply (intro conjI impI)
  apply ((metis butlast.simps(2) event.simps(3) front-tickFree-butlast tick-Free-Cons
    front-tickFree-single is-processT2 list.distinct(1) list.sel(1, 3))+)[2]
  by (simp add: F-Ndet)
  (metis * Ndet-commute Sync-commute list.distinct(1) list.sel(1, 3) ready-hyps same-div)
next

  { fix s
    assume assms :  $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle \langle ev \ e \ \# \ s \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$ 
    from assms(3) obtain t u r v
      where ** :  $\langle \text{front-tickFree } v \rangle \langle \text{tickFree } r \vee v = [] \rangle \langle ev \ e \ \# \ s = r \ @ \ v \rangle$ 
       $\langle r \ \text{setinterleaves } ((t, u), \text{insert } \checkmark (ev \ 'S)) \rangle$ 
       $\langle t \in \mathcal{D} P \wedge u \in \mathcal{T} Q \vee t \in \mathcal{D} Q \wedge u \in \mathcal{T} P \rangle$  by (simp add: D-Sync)
  blast
  have *** :  $\langle r \neq [] \rangle$  using **(4, 5) BOT-iff-D assms(1, 2) empty-setinterleaving
  by blast
  have  $\langle t \neq [] \wedge hd \ t = ev \ e \wedge tl \ r \ \text{setinterleaves } ((tl \ t, u), \text{insert } \checkmark (ev \ 'S)) \vee$ 
     $u \neq [] \wedge hd \ u = ev \ e \wedge tl \ r \ \text{setinterleaves } ((t, tl \ u), \text{insert } \checkmark (ev \ 'S)) \rangle$ 
  using **(3, 4) by (cases t; cases u, auto simp add: *** notin split: if-split-asm)
  with **(5) have  $\langle s \in \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$ 
  proof (elim disjE)
    assume * :  $\langle t \in \mathcal{D} P \wedge u \in \mathcal{T} Q \rangle$ 
       $\langle t \neq [] \wedge hd \ t = ev \ e \wedge tl \ r \ \text{setinterleaves } ((tl \ t, u), \text{insert } \checkmark (ev \ 'S)) \rangle$ 
    hence  $\langle s = tl \ r \ @ \ v \wedge tl \ t \in \mathcal{D} (P \text{ after } e) \wedge u \in \mathcal{T} Q \rangle$ 
      using **(3) *** ready-hyps(1) apply (simp add: D-After, intro conjI)
      by (metis list.sel(3) tl-append2) blast
    thus  $\langle s \in \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$ 
      using *(2) **(1, 2) tickFree-tl by (simp add: D-Ndet D-Sync) blast
  next
    assume * :  $\langle t \in \mathcal{D} P \wedge u \in \mathcal{T} Q \rangle$ 
       $\langle u \neq [] \wedge hd \ u = ev \ e \wedge tl \ r \ \text{setinterleaves } ((t, tl \ u), \text{insert } \checkmark (ev \ 'S)) \rangle$ 
  S))
  hence  $\langle s = tl \ r \ @ \ v \wedge t \in \mathcal{D} P \wedge tl \ u \in \mathcal{T} (Q \text{ after } e) \rangle$ 
    using **(3) ready-hyps(2) *** apply (simp add: T-After, intro conjI)
    by (metis list.sel(3) tl-append2) blast
  thus  $\langle s \in \mathcal{D} ((P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)) \rangle$ 
    using *(2) **(1, 2) tickFree-tl by (simp add: D-Ndet D-Sync) blast
  }

```

```

next
  assume * : ⟨t ∈ D Q ∧ u ∈ T P⟩
    ⟨t ≠ [] ∧ hd t = ev e ∧ tl r setinterleaves ((tl t, u), insert ✓ (ev ' S))⟩
  hence ⟨s = tl r @ v ∧ tl t ∈ D (Q after e) ∧ u ∈ T P⟩
    using **(1, 2, 3) ready-hyps apply (simp add: D-After, intro conjI)
    by (metis *** list.sel(3) tl-append2) blast
  thus ⟨s ∈ D ((P after e [S] Q) □ (P [S] Q after e))⟩
    using *(2) **(1, 2) tickFree-tl by (simp add: D-Ndet D-Sync) blast
next
  assume * : ⟨t ∈ D Q ∧ u ∈ T P⟩
    ⟨u ≠ [] ∧ hd u = ev e ∧ tl r setinterleaves ((t, tl u), insert ✓ (ev '
S))⟩
  hence ⟨s = tl r @ v ∧ t ∈ D Q ∧ tl u ∈ T (P after e)⟩
    using **(1, 2, 3) ready-hyps apply (simp add: T-After, intro conjI)
    by (metis *** list.sel(3) tl-append2) blast
  thus ⟨s ∈ D ((P after e [S] Q) □ (P [S] Q after e))⟩
    using *(2) **(1, 2) tickFree-tl by (simp add: D-Ndet D-Sync) blast
qed
} note * = this

show ⟨s ∈ D ((P [S] Q) after e) ⟹ s ∈ D ((P after e [S] Q) □ (P [S] Q after
e))⟩ for s
  apply (simp add: D-After ready-set-Sync ready-hyps After-BOT D-UU
    split: if-split-asm)
  apply (erule disjE;
    simp add: After-BOT Sync-commute[of ⊥, simplified Sync-BOT] Sync-BOT
Ndet-id D-UU,
    metis butlast-tl front-tickFree-butlast tickFree-tl)
  by (metis * list.collapse)
next

{ fix s P Q
  assume assms : ⟨P ≠ ⊥⟩ ⟨Q ≠ ⊥⟩ ⟨s ∈ D ((P after e [S] Q))⟩ ⟨ev e ∈ ready-set
P⟩
  from assms(3)[simplified D-Sync, simplified] obtain t u r v
  where * : ⟨front-tickFree v⟩ ⟨tickFree r ∨ v = []⟩ ⟨s = r @ v⟩
    ⟨r setinterleaves ((t, u), insert ✓ (ev ' S))⟩
    ⟨ev e # t ∈ D P ∧ u ∈ T Q ∨ t ∈ D Q ∧ ev e # u ∈ T P⟩
  by (simp add: D-Sync D-After T-After assms(4)) (metis list.collapse)
  from *(5) have ⟨ev e # s ∈ D (P [S] Q)⟩
  proof (elim disjE)
  assume ** : ⟨ev e # t ∈ D P ∧ u ∈ T Q⟩
  have *** : ⟨(ev e # r) setinterleaves ((ev e # t, u), insert ✓ (ev ' S))⟩
    using *(4) by (cases u; simp add: notin image-iff *(1, 2, 3))
  show ⟨ev e # s ∈ D (P [S] Q)⟩
    by (simp add: D-Sync)
    (metis *(1, 2, 3) ** *** Cons-eq-appendI event.distinct(1) tickFree-Cons)
  next
  assume ** : ⟨t ∈ D Q ∧ ev e # u ∈ T P⟩

```

```

have *** : ⟨(ev e # r) setinterleaves ((t, ev e # u), insert ✓ (ev ‘ S))⟩
using *(4) by (cases t; simp add: notin image-iff *(1, 2, 3))
show ⟨ev e # s ∈  $\mathcal{D}$  (P  $\llbracket$ S $\rrbracket$  Q)⟩
by (simp add: D-Sync)
      (metis *(1, 2, 3) ** *** Cons-eq-appendI event.distinct(1) tickFree-Cons)
qed
note * = this

show ⟨s ∈  $\mathcal{D}$  ((P after e  $\llbracket$ S $\rrbracket$  Q)  $\sqcap$  (P  $\llbracket$ S $\rrbracket$  Q after e))  $\implies$  s ∈  $\mathcal{D}$  ((P  $\llbracket$ S $\rrbracket$  Q)
after e)⟩ for s
apply (simp add: D-After ready-set-Sync ready-hyps After-BOT D-UU
      Sync-commute[of  $\perp$ , simplified Sync-BOT] Sync-BOT notin
Ndet-id)
apply (intro conjI impI)
apply ((metis D-imp-front-tickFree butlast.simps(2) event.distinct(1) tick-
Free-Cons
      front-tickFree-single list.discI list.sel(1, 3) front-tickFree-butlast+)[2])
by (simp add: D-Ndet)
      (metis * list.distinct(1) list.sel(1, 3) mono-D-Sync ready-hyps)
qed

```

lemma *not-readyL-not-readyR-After-Sync*: ⟨(P \llbracket S \rrbracket Q) after e = STOP⟩
if ready-hyps: ⟨ev e \notin ready-set P⟩ ⟨ev e \notin ready-set Q⟩
apply (subst not-ready-After, simp add: ready-set-Sync)
using ready-set-BOT ready-hyps **by** auto

Finally, the monster theorem !

theorem *After-Sync*:
⟨(P \llbracket S \rrbracket Q) after e =
(if P = \perp \vee Q = \perp then \perp
else if ev e ∈ ready-set P \wedge ev e ∈ ready-set Q
then if e ∈ S then P after e \llbracket S \rrbracket Q after e
else (P after e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q after e)
else if e ∈ S then STOP
else if ev e ∈ ready-set P then P after e \llbracket S \rrbracket Q
else if ev e ∈ ready-set Q then P \llbracket S \rrbracket Q after e
else STOP)⟩
by (simp add: Sync-commute[of \perp , simplified Sync-BOT] Sync-BOT After-BOT
 readyL-readyR-in-After-Sync readyL-readyR-not-in-After-Sync
 not-readyL-in-After-Sync readyL-not-readyR-not-in-After-Sync
 not-readyL-not-readyR-After-Sync)
 (metis Sync-commute not-readyL-in-After-Sync readyL-not-readyR-not-in-After-Sync)

4.5.4 After Hiding Operator

$P \setminus A$ is harder to deal with, we will only obtain refinements results.

lemma *Hiding-FD-Hiding-After-if-ready-inside*:
⟨e ∈ B \implies (P \setminus B) \sqsubseteq_{FD} (P after e \setminus B)⟩

and *After-Hiding-FD-Hiding-After-if-ready-notin*:
 $\langle e \notin B \implies (P \setminus B) \text{ after } e \sqsubseteq_{FD} (P \text{ after } e \setminus B) \rangle$
if ready: $\langle \text{ev } e \in \text{ready-set } P \rangle$
supply $\text{ready}' = \text{ready-notin-imp-ready-Hiding}[OF \text{ ready}]$
proof –
{ **fix** s
assume $\langle s \in \mathcal{D} (P \text{ after } e \setminus B) \rangle$
with *D-Hiding* **obtain** $t \ u$
where $*$: $\langle \text{front-tickFree } u \ \langle \text{tickFree } t \ \langle s = \text{trace-hide } t (ev \ 'B) @ u \rangle$
 $\langle t \in \mathcal{D} (P \text{ after } e) \vee (\exists f. \text{isInfHiddenRun } f (P \text{ after } e) B \wedge t \in \text{range}$
 $f) \rangle$
by *blast*
from $*(4)$ **have** $\langle s \in (\text{if } e \in B \text{ then } \mathcal{D} (P \setminus B) \text{ else } \mathcal{D} ((P \setminus B) \text{ after } e)) \rangle$
proof (*elim disjE*)
assume $\langle t \in \mathcal{D} (P \text{ after } e) \rangle$
hence $**$: $\langle \text{ev } e \# t \in \mathcal{D} P \rangle$ **by** (*simp add: D-After ready*) (*metis list.exhaust-sel*)
show $\langle s \in (\text{if } e \in B \text{ then } \mathcal{D} (P \setminus B) \text{ else } \mathcal{D} ((P \setminus B) \text{ after } e)) \rangle$
proof (*split if-split, intro conjI impI*)
assume $\langle e \in B \rangle$
with $*(3)$ **have** $***$: $\langle s = \text{trace-hide } (ev \ e \# t) (ev \ 'B) @ u \rangle$ **by** *simp*
show $\langle s \in \mathcal{D} (P \setminus B) \rangle$
by (*simp add: D-Hiding*)
(*metis *(1, 2) ** *** event.distinct(1) tickFree-Cons*)
next
assume $\langle e \notin B \rangle$
with $*(3)$ **have** $***$: $\langle \text{ev } e \# s = \text{trace-hide } (ev \ e \# t) (ev \ 'B) @ u \rangle$
by (*simp add: image-iff*)
have $\langle \text{ev } e \# s \in \mathcal{D} (P \setminus B) \rangle$
by (*simp add: D-Hiding*)
(*metis *(1, 2) ** *** event.distinct(1) tickFree-Cons*)
thus $\langle s \in \mathcal{D} ((P \setminus B) \text{ after } e) \rangle$
by (*simp add: D-After ready' e e \notin B*)
(*metis list.distinct(1) list.sel(1, 3)*)
qed
next
assume $\langle \exists f. \text{isInfHiddenRun } f (P \text{ after } e) B \wedge t \in \text{range } f \rangle$
then obtain f **where** $\langle \text{isInfHiddenRun } f (P \text{ after } e) B \ \langle t \in \text{range } f \rangle$ **by**
blast
hence $**$: $\langle \text{isInfHiddenRun } (\lambda i. \text{ev } e \# f \ i) P B \wedge$
 $\text{ev } e \# t \in \text{range } (\lambda i. \text{ev } e \# f \ i) \rangle$
by (*simp add: less-cons monotone-on-def T-After ready*)
(*metis list.exhaust-sel rangeE rangeI*)
show $\langle s \in (\text{if } e \in B \text{ then } \mathcal{D} (P \setminus B) \text{ else } \mathcal{D} ((P \setminus B) \text{ after } e)) \rangle$
proof (*split if-split, intro conjI impI*)
assume $\langle e \in B \rangle$
with $*(3)$ **have** $***$: $\langle s = \text{trace-hide } (ev \ e \# t) (ev \ 'B) @ u \rangle$ **by** *simp*
show $\langle s \in \mathcal{D} (P \setminus B) \rangle$
by (*simp add: D-Hiding*)
(*metis *(1, 2) ** *** event.distinct(1) tickFree-Cons*)

```

next
  assume  $\langle e \notin B \rangle$ 
  with  $*(3)$  have  $*** : \langle ev\ e \# s = trace\text{-}hide\ (ev\ e \# t)\ (ev\ 'B) \ @\ u \rangle$ 
    by (simp add: image-iff)
  have  $\langle ev\ e \# s \in \mathcal{D}\ (P \setminus B) \rangle$ 
    by (simp add: D-Hiding)
      (metis  $*(1, 2)$   $**\ ***\ event.distinct(1)\ tickFree-Cons$ )
  thus  $\langle s \in \mathcal{D}\ ((P \setminus B)\ after\ e) \rangle$ 
    by (simp add: D-After ready'  $\langle e \notin B \rangle$ )
      (metis list.distinct(1) list.sel(1, 3))
qed
qed
} note div-ref = this

{ fix  $s\ X$ 
  assume  $\langle (s, X) \in \mathcal{F}\ (P\ after\ e \setminus B) \rangle$ 
  with F-Hiding D-Hiding consider
     $\langle \exists t. s = trace\text{-}hide\ t\ (ev\ 'B) \wedge (t, X \cup ev\ 'B) \in \mathcal{F}\ (P\ after\ e) \rangle$ 
    |  $\langle s \in \mathcal{D}\ (P\ after\ e \setminus B) \rangle$  by blast
  hence  $\langle (s, X) \in (if\ e \in B\ then\ \mathcal{F}\ (P \setminus B)\ else\ \mathcal{F}\ ((P \setminus B)\ after\ e)) \rangle$ 
  proof cases
    assume  $\langle \exists t. s = trace\text{-}hide\ t\ (ev\ 'B) \wedge (t, X \cup ev\ 'B) \in \mathcal{F}\ (P\ after\ e) \rangle$ 
    then obtain  $t$  where  $* : \langle s = trace\text{-}hide\ t\ (ev\ 'B) \rangle \langle (ev\ e \# t, X \cup ev\ 'B) \in \mathcal{F}\ P \rangle$ 
      by (simp add: F-After ready) (metis list.exhaust-sel)
    show  $\langle (s, X) \in (if\ e \in B\ then\ \mathcal{F}\ (P \setminus B)\ else\ \mathcal{F}\ ((P \setminus B)\ after\ e)) \rangle$ 
    proof (split if-split, intro conjI impI)
      from * show  $\langle e \in B \implies (s, X) \in \mathcal{F}\ (P \setminus B) \rangle$ 
        by (simp add: F-Hiding) (metis filter.simps(2) image-eqI)
    next
      assume  $\langle e \notin B \rangle$ 
      with  $*(1)$  have  $** : \langle ev\ e \# s = trace\text{-}hide\ (ev\ e \# t)\ (ev\ 'B) \rangle$ 
        by (simp add: image-iff)
      show  $\langle (s, X) \in \mathcal{F}\ ((P \setminus B)\ after\ e) \rangle$ 
        by (simp add: F-After ready'  $\langle e \notin B \rangle$  F-Hiding)
          (metis  $*(2)$   $**\ list.discI\ list.sel(1, 3)$ )
    qed
  next
    show  $\langle s \in \mathcal{D}\ (P\ after\ e \setminus B) \implies (s, X) \in (if\ e \in B\ then\ \mathcal{F}\ (P \setminus B)\ else\ \mathcal{F}\ ((P \setminus B)\ after\ e)) \rangle$ 
      by (drule div-ref, simp split: if-split-asm; use NF-ND in blast)
    qed
  } note fail-ref = this

show  $\langle e \in B \implies (P \setminus B) \sqsubseteq_{FD} (P\ after\ e \setminus B) \rangle$ 
and  $\langle e \notin B \implies (P \setminus B)\ after\ e \sqsubseteq_{FD} (P\ after\ e \setminus B) \rangle$ 
  unfolding failure-divergence-refine-def le-ref-def using div-ref fail-ref by auto
qed

```

lemmas *Hiding-F-Hiding-After-if-ready-inside* =
 Hiding-FD-Hiding-After-if-ready-inside[*THEN leFD-imp-leF*]
and *After-Hiding-F-Hiding-After-if-ready-notin* =
 After-Hiding-FD-Hiding-After-if-ready-notin[*THEN leFD-imp-leF*]
and *Hiding-D-Hiding-After-if-ready-inside* =
 Hiding-FD-Hiding-After-if-ready-inside[*THEN leFD-imp-leD*]
and *After-Hiding-D-Hiding-After-if-ready-notin* =
 After-Hiding-FD-Hiding-After-if-ready-notin[*THEN leFD-imp-leD*]
and *Hiding-T-Hiding-After-if-ready-inside* =
 Hiding-FD-Hiding-After-if-ready-inside[*THEN leFD-imp-leF, THEN leF-imp-leT*]

and *After-Hiding-T-Hiding-After-if-ready-notin* =
 After-Hiding-FD-Hiding-After-if-ready-notin[*THEN leFD-imp-leF, THEN leF-imp-leT*]

corollary *Hiding-DT-Hiding-After-if-ready-inside*:
 $\langle ev e \in \text{ready-set } P \implies e \in B \implies (P \setminus B) \sqsubseteq_{DT} (P \text{ after } e \setminus B) \rangle$
and *After-Hiding-DT-Hiding-After-if-ready-notin*:
 $\langle ev e \in \text{ready-set } P \implies e \notin B \implies (P \setminus B) \text{ after } e \sqsubseteq_{DT} (P \text{ after } e \setminus B) \rangle$
by (*simp add: Hiding-D-Hiding-After-if-ready-inside*
 Hiding-T-Hiding-After-if-ready-inside leD-leT-imp-leDT)
 (*simp add: After-Hiding-D-Hiding-After-if-ready-notin*
 After-Hiding-T-Hiding-After-if-ready-notin leD-leT-imp-leDT)

This is the best we can obtain: even by restricting ourselves to two events, we can already construct a counterexample.

lemma defines *P-def*: $\langle P \equiv (Suc\ 0 \rightarrow (0 \rightarrow STOP)) \sqcap (0 \rightarrow SKIP) \rangle$
and *B-def*: $\langle B \equiv \{Suc\ 0\} \rangle$ **and** *e-def*: $\langle e \equiv 0 \rangle$ **and** *f-def*: $\langle f \equiv Suc\ 0 \rangle$
shows $\langle ev e \in \text{ready-set } P \wedge f \in B \wedge P \setminus B \neq P \text{ after } f \setminus B \rangle$
and $\langle ev e \in \text{ready-set } P \wedge e \notin B \wedge (P \setminus B) \text{ after } e \neq P \text{ after } e \setminus B \rangle$
unfolding *e-def f-def P-def B-def*
apply (*simp-all add: ready-set-Ndet ready-set-prefix*)
apply (*simp-all add: After-Ndet ready-set-prefix After-prefix Hiding-set-SKIP*)
apply (*simp-all add: Hiding-Ndet After-Ndet ready-set-prefix After-prefix Hiding-set-SKIP*
 Hiding-set-STOP no-Hiding-write0 Hiding-write0)
by (*metis After-prefix Ndet-is-STOP-iff SKIP-Neq-STOP write0-Ndet*)
 (*metis mono-Ndet-FD-right Ndet-commute SKIP-FD-iff SKIP-Neq-STOP STOP-FD-iff*)

4.5.5 After Renaming

lemma *After-Renaming*:
 $\langle \text{Renaming } P f \text{ after } e =$
 (*if* $P = \perp$ *then* \perp *else*
 $\sqcap a \in \{a. ev a \in \text{ready-set } P \wedge f a = e\}. \text{Renaming } (P \text{ after } a) f \rangle$
 (**is** $\langle ?lhs = (\text{if } P = \perp \text{ then } \perp \text{ else } ?rhs) \rangle$)
 — We treat the case $P = \perp$ separately because the set $\{a. f a = e\}$ may be empty,
 which implies $\sqcap a \in \{a. ev a \in \text{ready-set } P \wedge f a = e\}. \text{Renaming } (P \text{ after } a) f =$

STOP while Renaming $P f$ after $e = \perp$.

```

proof (split if-split, intro conjI impI)
  show  $\langle P = \perp \implies ?lhs = \perp \rangle$ 
    by (simp add: Renaming-BOT After-BOT GlobalNdet-id)
next
  assume non-BOT:  $\langle P \neq \perp \rangle$ 
  show  $\langle ?lhs = ?rhs \rangle$ 
  proof (subst Process-eq-spec-optimized, safe)
    fix  $s$ 
    assume  $\langle s \in \mathcal{D} ?lhs \rangle$ 
    hence  $*$  :  $\langle ev\ e \in \text{ready-set } (Renaming\ P\ f) \rangle \langle ev\ e \# s \in \mathcal{D} (Renaming\ P\ f) \rangle$ 
      by (auto simp add: D-After split: if-split-asm) (metis list.exhaust-sel)
    from  $*(2)$  obtain  $t1\ t2$ 
      where  $**$  :  $\langle tickFree\ t1 \rangle \langle front-tickFree\ t2 \rangle$ 
         $\langle ev\ e \# s = \text{map } (EvExt\ f)\ t1\ @\ t2 \rangle \langle t1 \in \mathcal{D}\ P \rangle$ 
      by (simp add: D-Renaming) blast
    from  $**(1, 3, 4)$  non-BOT obtain  $a\ t1'$ 
      where  $***$  :  $\langle t1 = ev\ a \# t1' \rangle \langle f\ a = e \rangle$ 
      by (cases  $t1$ ; simp add: BOT-iff-D EvExt-def)
        (metis comp-apply event.exhaust event.inject event.simps(4))
    have  $\langle ev\ a \in \text{ready-set } P \rangle$ 
      using  $**(4)\ ***(1)$  Cons-in-T-imp-elem-ready-set D-T by blast
    also have  $\langle s \in \mathcal{D} (Renaming\ (P\ \text{after}\ a)\ f) \rangle$ 
      using  $**\ ***(1)$  by (simp add: D-Renaming D-After calculation)
        (metis list.discI list.sel(1, 3))
    ultimately show  $\langle s \in \mathcal{D} ?rhs \rangle$ 
      using  $**(2)$  by (simp add: D-GlobalNdet) blast
next
  show  $\langle s \in \mathcal{D} ?rhs \implies s \in \mathcal{D} ?lhs \rangle$  for  $s$ 
    apply (simp add: D-GlobalNdet D-Renaming D-After
      ready-set-Renaming non-BOT EvExt-def image-iff
      split: if-split-asm event.split, intro conjI)
    by (metis (no-types, lifting) Nil-is-append-conv event.distinct(1)
      hd-append2 hd-map-EvExt list.collapse list.map-disc-iff
      map-tl tickFree-Cons tl-append2) blast
next
  fix  $s\ X$ 
  assume same-div :  $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$ 
  assume  $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ 
  then consider  $\langle ev\ e \notin \text{ready-set } (Renaming\ P\ f) \rangle \langle s = [] \rangle$ 
    |  $\langle ev\ e \in \text{ready-set } (Renaming\ P\ f) \rangle \langle (ev\ e \# s, X) \in \mathcal{F} (Renaming\ P\ f) \rangle$ 
      by (simp add: F-After split: if-split-asm) (metis list.exhaust-sel)
  thus  $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ 
  proof cases
    show  $\langle ev\ e \notin \text{ready-set } (Renaming\ P\ f) \implies s = [] \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
      by (simp add: ready-set-Renaming non-BOT F-GlobalNdet F-Renaming
        F-After)
        (metis ev-elem-anteced1 imageI vimage-eq)

```

next
assume $\langle ev\ e \in ready\text{-}set\ (Renaming\ P\ f) \rangle$
 $\langle (ev\ e\ \# \ s, X) \in \mathcal{F}\ (Renaming\ P\ f) \rangle$
from $assms(2)$ **consider** $\langle ev\ e\ \# \ s \in \mathcal{D}\ (Renaming\ P\ f) \rangle$
 $| \langle \exists s1. (s1, EvExt\ f\ -' X) \in \mathcal{F}\ P \wedge ev\ e\ \# \ s = map\ (EvExt\ f)\ s1 \rangle$
by $(simp\ add: F\text{-}Renaming\ D\text{-}Renaming)$ **blast**
thus $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
proof cases
assume $\langle ev\ e\ \# \ s \in \mathcal{D}\ (Renaming\ P\ f) \rangle$
hence $\langle s \in \mathcal{D}\ ?lhs \rangle$ **by** $(force\ simp\ add: D\text{-}After\ assms(1))$
with $D\text{-}F\ same\text{-}div$ **show** $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$ **by blast**
next
assume $\langle \exists s1. (s1, EvExt\ f\ -' X) \in \mathcal{F}\ P \wedge ev\ e\ \# \ s = map\ (EvExt\ f)\ s1 \rangle$
then obtain $s1$ **where** $*$: $\langle (s1, EvExt\ f\ -' X) \in \mathcal{F}\ P \rangle$
 $\langle ev\ e\ \# \ s = map\ (EvExt\ f)\ s1 \rangle$ **by meson**
from $*(2)$ **obtain** $a\ s1'$
where $**$: $\langle s1 = ev\ a\ \# \ s1' \rangle$ $\langle f\ a = e \rangle$
by $(cases\ s1; simp; metis\ *(2)\ EvExt\text{-}ev1\ event.inject$
 $hd\text{-}map\text{-}EvExt\ list.distinct(1)\ list.sel(1))$
have $\langle ev\ a \in ready\text{-}set\ P \rangle$
using $*(1)\ ** (1)$ $Cons\text{-}in\text{-}T\text{-}imp\text{-}elem\text{-}ready\text{-}set\ F\text{-}T$ **by blast**
also have $\langle (s, X) \in \mathcal{F}\ (Renaming\ (P\ after\ a)\ f) \rangle$
using $*(1, 2)\ ** (1)$ **by** $(simp\ add: F\text{-}Renaming\ F\text{-}After\ calculation)$
 $(metis\ list.distinct(1)\ list.sel(1, 3))$
ultimately show $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
using $** (2)$ **by** $(simp\ add: F\text{-}GlobalNdet)$ **blast**
qed
qed
next
fix $s\ X$
assume $same\text{-}div : \langle \mathcal{D}\ ?lhs = \mathcal{D}\ ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
then consider $\langle \forall a. ev\ a \in ready\text{-}set\ P \longrightarrow f\ a \neq e \rangle$ $\langle s = [] \rangle$
 $| \langle \exists a. f\ a = e \wedge ev\ a \in ready\text{-}set\ P \wedge (s, X) \in \mathcal{F}\ (Renaming\ (P\ after\ a)\ f) \rangle$
by $(auto\ simp\ add: F\text{-}GlobalNdet\ split: if\text{-}split\text{-}asm)$
thus $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
proof cases
show $\langle \forall a. ev\ a \in ready\text{-}set\ P \longrightarrow f\ a \neq e \implies s = [] \implies (s, X) \in \mathcal{F}\ ?lhs \rangle$
by $(auto\ simp\ add: F\text{-}After\ F\text{-}Renaming\ ready\text{-}set\ Renaming\ non\text{-}BOT)$
 $(metis\ (no\text{-}types, opaque\text{-}lifting)\ EvExt\text{-}ev1\ event.inject$
 $hd\text{-}map\text{-}EvExt\ list.distinct(1)\ list.sel(1)\ list.simps(9))$
next
assume $\langle \exists a. f\ a = e \wedge ev\ a \in ready\text{-}set\ P \wedge$
 $(s, X) \in \mathcal{F}\ (Renaming\ (P\ after\ a)\ f) \rangle$
then obtain a
where $*$: $\langle f\ a = e \rangle$ $\langle ev\ a \in ready\text{-}set\ P \rangle$
 $\langle (s, X) \in \mathcal{F}\ (Renaming\ (P\ after\ a)\ f) \rangle$ **by blast**
from $*(3)$ **consider** $\langle s \in \mathcal{D}\ (Renaming\ (P\ after\ a)\ f) \rangle$
 $| \langle \exists s1. (s1, EvExt\ f\ -' X) \in \mathcal{F}\ (P\ after\ a) \wedge s = map\ (EvExt\ f)\ s1 \rangle$

```

    by (simp add: F-Renaming D-Renaming) blast
  thus  $\langle s, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
  proof cases
    assume  $\langle s \in \mathcal{D} \text{ (Renaming (P after a) f)} \rangle$ 
    with  $\ast(1, 2)$  have  $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$  by (simp add: D-GlobalNdet) blast
    with D-F same-div show  $\langle s, X \rangle \in \mathcal{F} \text{ ?lhs}$  by blast
  next
    assume  $\langle \exists s1. (s1, \text{EvExt } f - ' X) \in \mathcal{F} (P \text{ after } a) \wedge s = \text{map } (\text{EvExt } f)$ 
  s1  $\rangle$ 
    then obtain s1 where  $\ast\ast : \langle s1, \text{EvExt } f - ' X \rangle \in \mathcal{F} (P \text{ after } a)$ 
       $\langle s = \text{map } (\text{EvExt } f) s1 \rangle$  by blast
    with  $\ast(1)$  have  $\ast\ast\ast : \langle \text{ev } e \notin \text{ready-set } (\text{Renaming } P f) \implies s = [] \rangle$ 
      by (simp add: ready-set-Renaming non-BOT image-iff F-After
        split: if-split-asm) (metis hd-map hd-map-EvExt)
    { assume  $\langle \text{ev } e \in \text{ready-set } (\text{Renaming } P f) \rangle$ 
      hence  $\langle (\text{ev } a \# s1, \text{EvExt } f - ' X) \in \mathcal{F} P \wedge \text{ev } e \# s = \text{map } (\text{EvExt } f)$ 
    (ev a # s1)  $\rangle$ 
      using  $\ast(1, 2) \ast\ast$  by (simp add: F-After  $\ast(2)$ )
        (metis hd-map hd-map-EvExt list.collapse)
      hence  $\langle (\text{ev } e \# s, X) \in \mathcal{F} (\text{Renaming } P f) \rangle$ 
        by (auto simp add: F-Renaming)
    }
    thus  $\langle s, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
      by (simp add: F-After  $\ast\ast\ast$ ) (metis list.discI list.sel(1, 3))
  qed
qed
qed
qed

```

4.6 Behaviour of After with Operators of HOL-CSPM

lemma *After-MultiDet-is-After-MultiNdet*:

$\langle \text{finite } A \implies (\Box a \in A. P a) \text{ after } e = (\Box a \in A. P a) \text{ after } e \rangle$

apply (cases $\langle A = \{\} \rangle$, simp)

apply (rotate-tac, induct rule: finite-set-induct-nonempty, simp)

by (simp add: After-Det-is-After-Ndet After-Ndet

ready-set-MultiDet ready-set-MultiNdet)

lemma *After-GlobalNdet*: $\langle (\Box a \in A. P a) \text{ after } e =$

(if $\text{ev } e \notin (\bigcup a \in A. \text{ready-set } (P a))$ then STOP

else $(\Box a \in \{a \in A. \text{ev } e \in \text{ready-set } (P a)\}. P a) \text{ after } e \rangle$

apply (subst Process-eq-spec, auto simp add: not-ready-After ready-set-GlobalNdet)

apply (auto simp add: F-After F-GlobalNdet D-After D-GlobalNdet ready-set-GlobalNdet

split: if-split-asm,

auto simp add: ready-set-def)

by (metis F-T append-Cons append-Nil is-processT3-ST list.exhaust list.sel(1))

(metis D-T append-Cons append-Nil hd-Cons-tl is-processT3-ST)

lemma *After-MultiNdet*: $\langle \text{finite } A \implies (\prod a \in A. P a) \text{ after } e =$
 $(\text{if } ev\ e \notin (\bigcup a \in A. \text{ready-set } (P a)) \text{ then } STOP$
 $\text{else } (\prod a \in \{a \in A. ev\ e \in \text{ready-set } (P a)\}. P a) \text{ after } e \rangle$
by (*subst* (1 2) *finite-GlobalNdet-is-MultiNdet[symmetric]*, *simp-all add: After-GlobalNdet*)

lemma *After-MultiDet*: $\langle \text{finite } A \implies$
 $(\prod a \in A. P a) \text{ after } e =$
 $(\text{if } ev\ e \notin (\bigcup a \in A. \text{ready-set } (P a)) \text{ then } STOP$
 $\text{else } (\prod a \in \{a \in A. ev\ e \in \text{ready-set } (P a)\}. P a) \text{ after } e \rangle$
by (*simp add: After-MultiDet-is-After-MultiNdet After-MultiNdet*)

4.7 Behaviour of *After* with Operators of HOL-CSP_OpSem

4.7.1 *After* Sliding

lemma *After-Sliding*:
 $\langle P \triangleright Q \text{ after } e =$
 $(\text{if } ev\ e \notin \text{ready-set } P \wedge ev\ e \notin \text{ready-set } Q \text{ then } STOP$
 $\text{else if } ev\ e \in \text{ready-set } P \wedge ev\ e \in \text{ready-set } Q \text{ then } (P \text{ after } e) \sqcap (Q \text{ after } e)$
 $\text{else if } ev\ e \in \text{ready-set } P \text{ then } P \text{ after } e \text{ else } Q \text{ after } e \rangle$
by (*simp add: Sliding-def After-Ndet After-Det ready-set-Det Ndet-id Ndet-assoc*)

An interesting corollary is that (\triangleright) is also concerned by the loss of determinism (see $?P \sqcap ?Q \text{ after } ?e = ?P \sqcap ?Q \text{ after } ?e$).

lemma *After-Sliding-is-After-Ndet*: $\langle P \triangleright Q \text{ after } e = P \sqcap Q \text{ after } e \rangle$
by (*simp add: After-Ndet After-Sliding*)

4.7.2 *After* Throwing

lemma *After-Throw*:
 $\langle (P \Theta a \in A. Q a) \text{ after } e =$
 $(\text{if } P = \perp \text{ then } \perp$
 $\text{else if } ev\ e \in \text{ready-set } P \text{ then if } e \in A \text{ then } Q\ e$
 $\text{else } (P \text{ after } e) \Theta a \in A. Q\ a$
 $\text{else } STOP \rangle$
(is $\langle ?lhs = ?rhs \rangle$)

proof –

have $\langle ?lhs = Q\ e \rangle$ **if** $\langle P \neq \perp \rangle$ **and** $\langle ev\ e \in \text{ready-set } P \rangle$ **and** $\langle e \in A \rangle$

proof (*subst Process-eq-spec, safe*)

fix s

assume $\langle s \in \mathcal{D}\ ?lhs \rangle$

with *that*(2) **have** $\langle ev\ e \neq s \in \mathcal{D}\ (P \Theta a \in A. Q a) \rangle$

by (*simp add: D-After ready-set-Throw*) (*metis list.exhaust-sel*)

with *that*(1) **show** $\langle s \in \mathcal{D}\ (Q\ e) \rangle$

apply (*simp add: D-Throw disjoint-iff BOT-iff-D, elim disjE*)

by (*metis hd-append2 hd-in-set image-eqI list.sel(1) that(3)*)

$(metis\ append-self-conv2\ event.inject\ hd-append2$
 $hd-in-set\ image-eqI\ list.sel(1, 3)\ that(3))$

next

have $\langle [ev\ e] \in \mathcal{T}\ P \wedge e \in A \rangle$ **using** *ready-set-def that(2, 3)* **by** *blast*

thus $\langle s \in \mathcal{D}\ (Q\ e) \implies s \in \mathcal{D}\ ((P\ \Theta\ a \in A.\ Q\ a)\ after\ e) \rangle$ **for** s

by (*simp add: D-After ready-set-Throw that(2) D-Throw*)

$(metis\ append-Nil\ empty-set\ inf-bot-left\ list.distinct(1)\ list.sel(1, 3))$

next

fix $s\ X$

assume $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$

with *that(2)* **have** $\langle (ev\ e \# s, X) \in \mathcal{F}\ (P\ \Theta\ a \in A.\ Q\ a) \rangle$

by (*simp add: F-After ready-set-Throw (metis list.exhaust-sel)*)

with *that(1, 3)* **show** $\langle (s, X) \in \mathcal{F}\ (Q\ e) \rangle$

apply (*simp add: F-Throw image-iff BOT-iff-D, elim disjE*)

by (*metis disjoint-iff hd-append2 hd-in-set image-eqI list.sel(1)*)

$(metis\ append-Nil\ event.inject\ hd-append2\ imageI\ insert-disjoint(2)$
 $list.exhaust-sel\ list.sel(1, 3)\ list.simps(15))$

next

have $\langle [ev\ e] \in \mathcal{T}\ P \wedge e \in A \rangle$ **using** *ready-set-def that(2, 3)* **by** *blast*

thus $\langle (s, X) \in \mathcal{F}\ (Q\ e) \implies (s, X) \in \mathcal{F}\ ?lhs \rangle$ **for** $s\ X$

by (*simp add: F-After ready-set-Throw that(2) F-Throw*)

$(metis\ append-Nil\ empty-set\ inf-bot-left\ list.distinct(1)\ list.sel(1, 3))$

qed

also have $\langle ?lhs = (P\ after\ e) \Theta\ a \in A.\ Q\ a \rangle$

if $\langle P \neq \perp \rangle$ **and** $\langle ev\ e \in ready-set\ P \rangle$ **and** $\langle e \notin A \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

fix s

assume $\langle s \in \mathcal{D}\ ?lhs \rangle$

with *that(2)* **have** $\langle ev\ e \# s \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q\ a) \rangle$

by (*simp add: D-After ready-set-Throw (metis list.exhaust-sel)*)

then consider $\langle \exists t1\ t2.\ ev\ e \# s = t1\ @\ t2 \wedge t1 \in \mathcal{D}\ P \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \rangle$

$| \langle \exists t1\ a\ t2.\ ev\ e \# s = t1\ @\ ev\ a \# t2 \wedge t1\ @\ [ev\ a] \in \mathcal{T}\ P \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D}\ (Q\ a) \rangle$

by (*simp add: D-Throw*) *blast*

thus $\langle s \in \mathcal{D}\ ((P\ after\ e) \Theta\ a \in A.\ Q\ a) \rangle$

proof cases

assume $\langle \exists t1\ t2.\ ev\ e \# s = t1\ @\ t2 \wedge t1 \in \mathcal{D}\ P \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \rangle$

then obtain $t1\ t2$

where $*$: $\langle ev\ e \# s = t1\ @\ t2 \rangle \langle t1 \in \mathcal{D}\ P \rangle \langle tickFree\ t1 \rangle$
 $\langle set\ t1 \cap ev\ 'A = \{\} \rangle \langle front-tickFree\ t2 \rangle$ **by** *blast*

from *that(1)* $*$ (1, 2) *BOT-iff-D* **obtain** $t1'$

where $\langle t1 = ev\ e \# t1' \rangle$ **by** (*cases t1*) *auto*

with $*$ (1, 2, 3, 4) **have** $\langle s = t1' @ t2 \wedge t1' \in \mathcal{D}\ (P\ after\ e) \wedge$
 $tickFree\ t1' \wedge set\ t1' \cap ev\ 'A = \{\} \rangle$

by (*simp add: D-After that(2)*) (*metis list.discI list.sel(1, 3)*)

with $*$ (5) **show** $\langle s \in \mathcal{D}\ ((P\ after\ e) \Theta\ a \in A.\ Q\ a) \rangle$

by (*auto simp add: D-Throw*)
 next
 assume $\langle \exists t1\ a\ t2. ev\ e\ \# s = t1\ @\ ev\ a\ \# t2 \wedge t1\ @\ [ev\ a] \in \mathcal{T}\ P \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D}\ (Q\ a) \rangle$
 then obtain $t1\ a\ t2$
 where $*$: $\langle ev\ e\ \# s = t1\ @\ ev\ a\ \# t2 \rangle \langle t1\ @\ [ev\ a] \in \mathcal{T}\ P \rangle$
 $\langle set\ t1 \cap ev\ 'A = \{\} \rangle \langle a \in A \rangle \langle t2 \in \mathcal{D}\ (Q\ a) \rangle$ by *blast*
 have $\langle e \neq a \rangle$ using $*(4)$ that(3) by *blast*
 with $*(1)$ obtain $t1'$ where $**$: $\langle t1 = ev\ e\ \# t1' \rangle$ by (*cases t1*) *auto*
 with $*(2)$ that(2) have $\langle t1' @ [ev\ a] \in \mathcal{T}\ (P\ after\ e) \rangle$
 by (*simp add: T-After*) (*metis list.distinct(1) list.sel(1, 3)*)
 thus $\langle s \in \mathcal{D}\ ((P\ after\ e) \Theta\ a \in A. Q\ a) \rangle$
 using $*(1, 3, 4, 5)$ $**$ by (*simp add: D-Throw*) *blast*
 qed
 next
 fix s
 assume $\langle s \in \mathcal{D}\ ((P\ after\ e) \Theta\ a \in A. Q\ a) \rangle$
 then consider $\langle \exists t1\ t2. s = t1\ @\ t2 \wedge t1 \in \mathcal{D}\ (P\ after\ e) \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \rangle$
 | $\langle \exists t1\ a\ t2. s = t1\ @\ ev\ a\ \# t2 \wedge t1\ @\ [ev\ a] \in \mathcal{T}\ (P\ after\ e) \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D}\ (Q\ a) \rangle$
 by (*simp add: D-Throw*) *blast*
 thus $\langle s \in \mathcal{D}\ ?lhs \rangle$
 proof *cases*
 assume $\langle \exists t1\ t2. s = t1\ @\ t2 \wedge t1 \in \mathcal{D}\ (P\ after\ e) \wedge tickFree\ t1 \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge front-tickFree\ t2 \rangle$
 then obtain $t1\ t2$
 where $*$: $\langle s = t1\ @\ t2 \rangle \langle t1 \in \mathcal{D}\ (P\ after\ e) \rangle \langle tickFree\ t1 \rangle$
 $\langle set\ t1 \cap ev\ 'A = \{\} \rangle \langle front-tickFree\ t2 \rangle$ by *blast*
 from $*(2)$ that(2) have $**$: $\langle ev\ e\ \# t1 \in \mathcal{D}\ P \rangle$
 by (*simp add: D-After*) (*metis list.exhaust-sel*)
 have $***$: $\langle tickFree\ (ev\ e\ \# t1) \wedge set\ (ev\ e\ \# t1) \cap ev\ 'A = \{\} \rangle$
 by (*simp add: image-iff *(3, 4) that(3)*)
 show $\langle s \in \mathcal{D}\ ?lhs \rangle$
 by (*simp add: D-After D-Throw ready-set-Throw that(2)*)
 (*metis *(1, 5) ** *** Cons-eq-appendI list.discI list.sel(1, 3)*)
 next
 assume $\langle \exists t1\ a\ t2. s = t1\ @\ ev\ a\ \# t2 \wedge t1\ @\ [ev\ a] \in \mathcal{T}\ (P\ after\ e) \wedge$
 $set\ t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D}\ (Q\ a) \rangle$
 then obtain $t1\ a\ t2$
 where $*$: $\langle s = t1\ @\ ev\ a\ \# t2 \rangle \langle t1\ @\ [ev\ a] \in \mathcal{T}\ (P\ after\ e) \rangle$
 $\langle set\ t1 \cap ev\ 'A = \{\} \rangle \langle a \in A \rangle \langle t2 \in \mathcal{D}\ (Q\ a) \rangle$ by *blast*
 from $*(2)$ that(2) have $**$: $\langle ev\ e\ \# t1 @ [ev\ a] \in \mathcal{T}\ P \rangle$
 by (*simp add: T-After*) (*metis list.exhaust-sel*)
 have $***$: $\langle set\ (ev\ e\ \# t1) \cap ev\ 'A = \{\} \rangle$
 by (*simp add: image-iff *(3) that(3)*)
 show $\langle s \in \mathcal{D}\ ?lhs \rangle$
 by (*simp add: D-After D-Throw ready-set-Throw that(2)*)
 (*metis *(1, 4, 5) ** *** Cons-eq-appendI list.discI list.sel(1, 3)*)

qed
next
fix $s X$
assume $\langle \text{same-div} : \langle \mathcal{D} \text{ ?lhs} = \mathcal{D} (\text{Throw } (P \text{ after } e) A Q) \rangle$
assume $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
with $\text{that}(2)$ **have** $\langle (ev\ e \# s, X) \in \mathcal{F} (P \Theta a \in A. Q\ a) \rangle$
by (*simp add: F-After ready-set-Throw*) (*metis list.exhaust-sel*)
then consider $\langle ev\ e \# s \in \mathcal{D} (P \Theta a \in A. Q\ a) \rangle$
| $\langle (ev\ e \# s, X) \in \mathcal{F} P \rangle \langle \text{set } s \cap ev\ 'A = \{\} \rangle$
| $\langle \exists t1\ a\ t2. ev\ e \# s = t1 \ @\ ev\ a \# t2 \wedge t1 \ @\ [ev\ a] \in \mathcal{T} P \wedge$
| $\text{set } t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q\ a) \rangle$
by (*simp add: F-Throw D-Throw*) *blast*
thus $\langle (s, X) \in \mathcal{F} ((P \text{ after } e) \Theta a \in A. Q\ a) \rangle$
proof cases
assume $\langle ev\ e \# s \in \mathcal{D} (P \Theta a \in A. Q\ a) \rangle$
hence $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$ **by** (*force simp add: D-After ready-set-Throw that(2)*)
with $\text{same-div } D\text{-}F$ **show** $\langle (s, X) \in \mathcal{F} ((P \text{ after } e) \Theta a \in A. Q\ a) \rangle$ **by** *blast*
next
show $\langle (ev\ e \# s, X) \in \mathcal{F} P \implies \text{set } s \cap ev\ 'A = \{\} \implies$
 $(s, X) \in \mathcal{F} ((P \text{ after } e) \Theta a \in A. Q\ a) \rangle$
by (*simp add: F-Throw F-After that(2)*)
(*metis list.distinct(1) list.sel(1, 3)*)
next
assume $\langle \exists t1\ a\ t2. ev\ e \# s = t1 \ @\ ev\ a \# t2 \wedge t1 \ @\ [ev\ a] \in \mathcal{T} P \wedge$
 $\text{set } t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q\ a) \rangle$
then obtain $t1\ a\ t2$
where $*$: $\langle ev\ e \# s = t1 \ @\ ev\ a \# t2 \rangle \langle t1 \ @\ [ev\ a] \in \mathcal{T} P \rangle$
 $\langle \text{set } t1 \cap ev\ 'A = \{\} \rangle \langle a \in A \rangle \langle (t2, X) \in \mathcal{F} (Q\ a) \rangle$ **by** *blast*
have $\langle e \neq a \rangle$ **using** $*(4)$ $\text{that}(3)$ **by** *blast*
with $*(1)$ **obtain** $t1'$ **where** $\langle t1 = ev\ e \# t1' \rangle$ **by** (*cases t1*) *auto*
also have $\langle t1' \ @\ [ev\ a] \in \mathcal{T} (P \text{ after } e) \wedge \text{set } t1' \cap ev\ 'A = \{\} \rangle$
using $*(2, 3)$ **by** (*simp add: image-iff T-After that(2) calculation*)
(*metis list.distinct(1) list.sel(1, 3)*)
ultimately show $\langle (s, X) \in \mathcal{F} ((P \text{ after } e) \Theta a \in A. Q\ a) \rangle$
using $*(1, 4, 5)$ **by** (*simp add: F-Throw*) *blast*
qed
next
fix $s X$
assume $\langle \text{same-div} : \langle \mathcal{D} \text{ ?lhs} = \mathcal{D} (\text{Throw } (P \text{ after } e) A Q) \rangle$
assume $\langle (s, X) \in \mathcal{F} ((P \text{ after } e) \Theta a \in A. Q\ a) \rangle$
then consider $\langle s \in \mathcal{D} ((P \text{ after } e) \Theta a \in A. Q\ a) \rangle$
| $\langle (s, X) \in \mathcal{F} (P \text{ after } e) \rangle \langle \text{set } s \cap ev\ 'A = \{\} \rangle$
| $\langle \exists t1\ a\ t2. s = t1 \ @\ ev\ a \# t2 \wedge t1 \ @\ [ev\ a] \in \mathcal{T} (P \text{ after } e) \wedge$
| $\text{set } t1 \cap ev\ 'A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q\ a) \rangle$
by (*simp add: F-Throw D-Throw*) *blast*
thus $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof cases
show $\langle s \in \mathcal{D} (\text{Throw } (P \text{ after } e) A Q) \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
using $\text{same-div } D\text{-}F$ **by** *blast*

next
assume $assms : \langle (s, X) \in \mathcal{F} (P \text{ after } e) \rangle \langle \text{set } s \cap \text{ev } 'A = \{\} \rangle$
from $assms(2)$ **have** $* : \langle \text{set } (ev \ e \ \# \ s) \cap \text{ev } 'A = \{\} \rangle$
by (*simp add: image-iff that(3) assms(2)*)
from $assms(1)$ **have** $\langle (ev \ e \ \# \ s, X) \in \mathcal{F} \ P \rangle$
by (*simp add: F-After that(2)*) (*metis list.exhaust-sel*)
thus $\langle (s, X) \in \mathcal{F} \ ?lhs \rangle$
by (*simp add: F-After F-Throw ready-set-Throw that(2)*)
(*metis * list.discI list.sel(1, 3)*)
next
assume $\langle \exists t1 \ a \ t2. \ s = t1 \ @ \ ev \ a \ \# \ t2 \wedge \ t1 \ @ \ [ev \ a] \in \mathcal{T} (P \text{ after } e) \wedge$
 $\text{set } t1 \cap \text{ev } 'A = \{\} \wedge \ a \in A \wedge (t2, X) \in \mathcal{F} (Q \ a) \rangle$
then obtain $t1 \ a \ t2$
where $* : \langle s = t1 \ @ \ ev \ a \ \# \ t2 \rangle \langle t1 \ @ \ [ev \ a] \in \mathcal{T} (P \text{ after } e) \rangle$
 $\langle \text{set } t1 \cap \text{ev } 'A = \{\} \rangle \langle a \in A \rangle \langle (t2, X) \in \mathcal{F} (Q \ a) \rangle$ **by** *blast*
from $*(2)$ *that(2)* **have** $** : \langle ev \ e \ \# \ t1 \ @ \ [ev \ a] \in \mathcal{T} \ P \rangle$
by (*simp add: T-After*) (*metis list.exhaust-sel*)
have $*** : \langle \text{set } (ev \ e \ \# \ t1) \cap \text{ev } 'A = \{\} \rangle$
by (*simp add: image-iff *(3) that(3)*)
from $*(1, 4, 5)$ **show** $\langle (s, X) \in \mathcal{F} \ ?lhs \rangle$
by (*simp add: F-After F-Throw ready-set-Throw that(2)*)
(*metis ** *** append-Cons list.distinct(1) list.sel(1, 3)*)
qed
qed
ultimately show $\langle ?lhs = ?rhs \rangle$
by (*simp add: Throw-BOT After-BOT STOP-iff-T T-After ready-set-Throw*)
qed

4.7.3 After Interrupting

theorem *After-Interrupt*:

$\langle (P \ \Delta \ Q) \text{ after } e =$
 $($ *if* $ev \ e \in \text{ready-set } P \wedge ev \ e \in \text{ready-set } Q$
 $\text{then } (Q \text{ after } e) \sqcap (P \text{ after } e \ \Delta \ Q)$
 $\text{else } \text{if } ev \ e \in \text{ready-set } P \wedge ev \ e \notin \text{ready-set } Q$
 $\text{then } P \text{ after } e \ \Delta \ Q$
 $\text{else } \text{if } ev \ e \notin \text{ready-set } P \wedge ev \ e \in \text{ready-set } Q$
 $\text{then } Q \text{ after } e$
 $\text{else } STOP) \rangle$

proof –

have $\langle (P \ \Delta \ Q) \text{ after } e \sqsubseteq_{FD} Q \text{ after } e \rangle$ **if** *ready*: $\langle ev \ e \in \text{ready-set } Q \rangle$

proof (*unfold failure-divergence-refine-def le-ref-def, safe*)

fix s

assume $\langle s \in \mathcal{D} (Q \text{ after } e) \rangle$

hence $\langle ev \ e \ \# \ s \in \mathcal{D} \ Q \rangle$

by (*simp add: D-After ready*) (*metis list.exhaust-sel*)

thus $\langle s \in \mathcal{D} ((P \ \Delta \ Q) \text{ after } e) \rangle$

by (*simp add: D-After ready ready-set-Interrupt D-Interrupt*)

$(metis\ Nil\ elem\ T\ append\ Nil\ list.\ distinct(1)\ list.sel(1, 3)\ tickFree\ Nil)$
next
show $\langle (s, X) \in \mathcal{F} (Q\ after\ e) \implies (s, X) \in \mathcal{F} ((P \Delta Q)\ after\ e) \rangle$ **for** $s\ X$
by $(simp\ add:\ F\ After\ ready\ set\ Interrupt\ ready\ F\ Interrupt)$
 $(metis\ Nil\ elem\ T\ append\ Nil\ tickFree\ Nil)$
qed

moreover have $\langle (P \Delta Q)\ after\ e \sqsubseteq_{FD} P\ after\ e \Delta Q \rangle$ **if** $ready:\ \langle ev\ e \in ready\ set\ P \rangle$
proof $(unfold\ failure\ divergence\ refine\ def\ le\ ref\ def,\ safe)$
show $\langle s \in \mathcal{D} (P\ after\ e \Delta Q) \implies s \in \mathcal{D} ((P \Delta Q)\ after\ e) \rangle$ **for** s
apply $(simp\ add:\ D\ Interrupt\ D\ After\ ready\ T\ After\ ready\ set\ Interrupt,$
 $elim\ disjE)$
by $blast\ (metis\ append\ Cons\ event.\ simps(3)\ list.sel(1, 3)\ neq\ Nil\ conv\ tick\ Free\ Cons)$
next
show $\langle (s, X) \in \mathcal{F} (P\ after\ e \Delta Q) \implies (s, X) \in \mathcal{F} ((P \Delta Q)\ after\ e) \rangle$ **for** $s\ X$
apply $(simp\ add:\ F\ Interrupt\ F\ After\ ready\ ready\ set\ Interrupt\ T\ After\ D\ After,$
 $elim\ disjE)$
by $(metis\ (no\ types,\ opaque\ lifting)\ [[metis\ verbose = false]]$
 $append\ Cons\ list.\ distinct(1)\ event.\ distinct(1)$
 $list.\ exhaust\ sel\ list.sel(1, 3)\ tickFree\ Cons)+$
qed

moreover have $\langle Q\ after\ e \sqsubseteq_{FD} (P \Delta Q)\ after\ e \rangle$ **if** $not\ ready:\ \langle ev\ e \notin ready\ set\ P \rangle$
proof $(unfold\ failure\ divergence\ refine\ def\ le\ ref\ def,\ safe)$
show $\langle s \in \mathcal{D} ((P \Delta Q)\ after\ e) \implies s \in \mathcal{D} (Q\ after\ e) \rangle$ **for** s
by $(simp\ add:\ D\ After\ not\ ready\ ready\ set\ Interrupt\ D\ Interrupt$
 $split:\ if\ split\ asm)$
 $(metis\ Cons\ in\ T\ imp\ elem\ ready\ set\ D\ T\ not\ ready$
 $append\ Nil\ hd\ append2\ list.\ exhaust\ sel)$
next
show $\langle (s, X) \in \mathcal{F} ((P \Delta Q)\ after\ e) \implies (s, X) \in \mathcal{F} (Q\ after\ e) \rangle$ **for** $s\ X$
by $(simp\ add:\ F\ After\ not\ ready\ ready\ set\ Interrupt\ F\ Interrupt$
 $split:\ if\ split\ asm,\ elim\ exE\ disjE\ conjE)$
 $(metis\ (no\ types,\ opaque\ lifting)\ [[metis\ verbose = false]]$
 $Cons\ in\ T\ imp\ elem\ ready\ set\ F\ T\ NF\ ND\ hd\ Cons\ tl$
 $snoc\ eq\ iff\ butlast\ append\ self\ conv2\ hd\ append2\ not\ ready)+$
qed

moreover have $\langle P\ after\ e \Delta Q \sqsubseteq_{FD} (P \Delta Q)\ after\ e \rangle$
if $ready\ hysps:\ \langle ev\ e \in ready\ set\ P \rangle\ \langle ev\ e \notin ready\ set\ Q \rangle$
proof $(unfold\ failure\ divergence\ refine\ def\ le\ ref\ def,\ safe)$
show $\langle s \in \mathcal{D} ((P \Delta Q)\ after\ e) \implies s \in \mathcal{D} (P\ after\ e \Delta Q) \rangle$ **for** s
by $(simp\ add:\ D\ After\ T\ After\ ready\ hysps\ ready\ set\ Interrupt\ D\ Interrupt)$
 $(metis\ tickFree\ tl\ Cons\ in\ T\ imp\ elem\ ready\ set\ D\ T\ eq\ Nil\ appendI$
 $hd\ append\ list.\ exhaust\ sel\ ready\ hysps(2)\ tl\ append\ if)$

```

next
  fix s X
  assume ⟨(s, X) ∈  $\mathcal{F}$  ((P  $\Delta$  Q) after e)⟩
  with ready-hyps(1) have ⟨(ev e # s, X) ∈  $\mathcal{F}$  (P  $\Delta$  Q)⟩
  by (simp add: F-After ready-set-Interrupt) (metis list.exhaust-sel)
  thus ⟨(s, X) ∈  $\mathcal{F}$  (P after e  $\Delta$  Q)⟩
  by (simp add: F-Interrupt F-After D-After T-After ready-hyps, elim disjE)
      (metis (no-types, opaque-lifting) [[metis-verbose = false]] tickFree-tl
        Cons-in-T-imp-elem-ready-set F-T NT-ND append-self-conv2 hd-append
        event.distinct(1) list.sel(1, 3) list.distinct(1) ready-hyps(2) tl-append2)+
qed

moreover have ⟨(Q after e)  $\sqcap$  (P after e  $\Delta$  Q)  $\sqsubseteq_{FD}$  (P  $\Delta$  Q) after e⟩
if both-ready: ⟨ev e ∈ ready-set P⟩ ⟨ev e ∈ ready-set Q⟩
proof (unfold failure-divergence-refine-def le-ref-def, safe)
  fix s
  assume ⟨s ∈  $\mathcal{D}$  ((P  $\Delta$  Q) after e)⟩
  with both-ready(1) have ⟨ev e # s ∈  $\mathcal{D}$  (P  $\Delta$  Q)⟩
  by (simp add: D-After ready-set-Interrupt) (metis list.exhaust-sel)
  thus ⟨s ∈  $\mathcal{D}$  ((Q after e)  $\sqcap$  (P after e  $\Delta$  Q))⟩
  by (simp add: D-Interrupt D-After T-After D-Ndet both-ready)
      (metis tickFree-tl append-Cons append-Nil
        list.distinct(1) list.exhaust-sel list.sel(1, 3))
next
  fix s X
  assume ⟨(s, X) ∈  $\mathcal{F}$  ((P  $\Delta$  Q) after e)⟩
  with both-ready(1) have ⟨(ev e # s, X) ∈  $\mathcal{F}$  (P  $\Delta$  Q)⟩
  by (simp add: F-After ready-set-Interrupt) (metis list.exhaust-sel)
  thus ⟨(s, X) ∈  $\mathcal{F}$  ((Q after e)  $\sqcap$  (P after e  $\Delta$  Q))⟩
  by (simp add: F-Interrupt F-After F-Ndet D-After T-After both-ready, elim
    disjE)
      (metis (no-types, opaque-lifting) [[metis-verbose = false]]
        event.distinct(1) hd-append2 list.distinct(1) list.sel(1, 3)
        process-chaos self-append-conv2 tickFree-tl tl-append2)+
qed

ultimately show ?thesis
  by (auto simp add: STOP-FD-iff not-ready-After intro: FD-antisym)
      (metis mono-Ndet-FD FD-antisym Ndet-id)
qed

```

4.8 Behaviour of *After* with Reference Processes

lemma *After-DF*: ⟨DF A after e = (if e ∈ A then DF A else STOP)⟩
 by (subst DF-unfold, subst After-Mndetprefix, simp)

lemma *After-DF_{SKIP}*:
 ⟨DF_{SKIP} A after e = (if e ∈ A then DF_{SKIP} A else STOP)⟩
 by (subst DF_{SKIP}-unfold)

(simp add: ready-set-SKIP ready-set-Mndetprefix After-Ndet After-Mndetprefix)

lemma *After-RUN*: $\langle \text{RUN } A \text{ after } e = (\text{if } e \in A \text{ then RUN } A \text{ else STOP}) \rangle$
by (subst RUN-unfold, subst After-Mprefix, simp)

lemma *After-CHAOS*: $\langle \text{CHAOS } A \text{ after } e = (\text{if } e \in A \text{ then CHAOS } A \text{ else STOP}) \rangle$
by (subst CHAOS-unfold)
(simp add: After-Ndet ready-set-STOP ready-set-Mprefix After-Mprefix)

lemma *After-CHAOS_{SKIP}*:
 $\langle \text{CHAOS}_{\text{SKIP}} A \text{ after } e = (\text{if } e \in A \text{ then CHAOS}_{\text{SKIP}} A \text{ else STOP}) \rangle$
by (subst CHAOS_{SKIP}-unfold)
(simp add: ready-set-Ndet ready-set-STOP ready-set-SKIP
ready-set-Mprefix After-Ndet After-Mprefix)

lemma *DF-FD-After*: $\langle \text{DF } A \sqsubseteq_{\text{FD}} P \text{ after } e \rangle$
if $\langle \text{DF } A \sqsubseteq_{\text{FD}} P \rangle$ **and** $\langle \text{ev } e \in \text{ready-set } P \rangle$
proof –
have $\langle \text{DF } A \text{ after } e \sqsubseteq_{\text{FD}} P \text{ after } e \rangle$ **by** (rule mono-After-FD[OF that(1)])
(use that(2) **in** $\langle \text{simp add: ready-set-DF image-iff} \rangle$)
also have $\langle e \in A \rangle$
by (metis After-DF DF-Univ-freeness DF-unfold STOP-FD-iff UNIV-I deadlock-free-def empty-iff
mono-After-FD mt-Mndetprefix non-deadlock-free-STOP ready-set-STOP
that)
ultimately show $\langle \text{DF } A \sqsubseteq_{\text{FD}} P \text{ after } e \rangle$ **by** (subst (asm) After-DF, simp split:
if-splits)
qed

lemma *DF_{SKIP}-FD-After*: $\langle \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \text{ after } e \rangle$
if $\langle \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \rangle$ **and** $\langle \text{ev } e \in \text{ready-set } P \rangle$
proof –
have $\langle \text{DF}_{\text{SKIP}} A \text{ after } e \sqsubseteq_{\text{FD}} P \text{ after } e \rangle$ **by** (rule mono-After-FD[OF that(1)])
(use that(2) **in** $\langle \text{simp add: ready-set-DF image-iff} \rangle$)
also have $\langle e \in A \rangle$ **using** anti-mono-ready-set-FD ready-set-DF_{SKIP} that **by**
fastforce

ultimately show $\langle \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \text{ after } e \rangle$ **by** (subst (asm) After-DF_{SKIP},
simp split: if-splits)
qed

We have corollaries on *deadlock-free* and *deadlock-free_{SKIP}*.

corollary *deadlock-free-After*:
 $\langle \text{deadlock-free } P \implies \text{deadlock-free } (P \text{ after } e) \iff (\text{if } \text{ev } e \in \text{ready-set } P \text{ then True else False}) \rangle$

apply (*simp add: non-deadlock-free-STOP not-ready-After*)
unfolding *deadlock-free-def* **by** (*intro impI DF-FD-After*)

corollary *deadlock-free_{SKIP}-After*:

$\langle \text{deadlock-free}_{SKIP} P \implies$

$\text{deadlock-free}_{SKIP} (P \text{ after } e) \longleftrightarrow (\text{if } ev \ e \in \text{ready-set } P \text{ then True else False}) \rangle$

apply (*simp add: non-deadlock-free_{SKIP}-STOP not-ready-After*)

unfolding *deadlock-free_{SKIP}-FD* **by** (*intro impI DF_{SKIP}-FD-After*)

end

Chapter 5

Extension of the After Operator

5.1 The AfterExt Operator

```
theory AfterExt
  imports After
begin
```

5.1.1 Definition

We just defined $P \text{ after } e$ for $P::'\alpha$ process and $e::'\alpha$; in other words we cannot handle \checkmark . We now introduce a generalisation for $e::'\alpha$ event.

```
definition AfterExt ::  $\langle ['\alpha \text{ process}, '\alpha \text{ event}] \Rightarrow '\alpha \text{ process} \rangle$  (infixl  $\langle \text{afterExt} \rangle$  77)
  where  $\langle P \text{ afterExt } e \equiv \text{case } e \text{ of } \text{ev } x \Rightarrow P \text{ after } x$ 
      |  $\checkmark \Rightarrow \text{if } P = \perp \text{ then } \perp \text{ else } \text{STOP} \rangle$ 
```

lemma *not-ready-AfterExt:*

```
 $\langle e \notin \text{ready-set } P \Longrightarrow P \text{ afterExt } e = (\text{if } P = \perp \text{ then } \perp \text{ else } \text{STOP}) \rangle$ 
by (auto simp add: AfterExt-def After-BOT intro: not-ready-After split: event.split)
```

lemma *ready-set-AfterExt:*

```
 $\langle \text{ready-set } (P \text{ afterExt } e) =$ 
   $(\text{if } P = \perp \text{ then } \text{UNIV} \text{ else if } e = \checkmark \text{ then } \{\} \text{ else } \{a. e \# [a] \in \mathcal{T} P\}) \rangle$ 
by (simp add: AfterExt-def ready-set-After ready-set-BOT
    ready-set-STOP T-UU front-tickFree-butlast
    split: event.split)
```

5.1.2 Projections

lemma *F-AfterExt:*

```
 $\langle \mathcal{F} (P \text{ afterExt } e) =$ 
   $( \text{if } e = \checkmark \wedge P = \perp \text{ then } \{(s, X). \text{front-tickFree } s\}$ 
```

else if $e \in \text{ready-set } P$ then $\{(tl\ s, X) \mid s\ X. (s, X) \in \mathcal{F}\ P \wedge s \neq [] \wedge hd\ s = e\}$
 else $\{(s, X). s = []\}$
 (is $\langle - = ?rhs \rangle$)
proof (unfold *AfterExt-def*, *split event.split*, *intro conjI allI impI*)
show $\langle e = ev\ x \implies \mathcal{F}\ (P\ \text{after}\ x) = ?rhs \rangle$ **for** x
by (*simp add: F-After*)
next
show $\langle e = \checkmark \implies \mathcal{F}\ (\text{if } P = \perp \text{ then } \perp \text{ else } STOP) = ?rhs \rangle$
by (*simp add: F-UU F-STOP BOT-iff-D ready-set-def set-eq-iff*)
 (*metis F-T T-nonTickFree-imp-decomp append-single-T-imp-tickFree hd-append2*
hd-in-set list.discI list.sel(1) list.sel(3) tickFree-def tick-T-F)
qed

lemma *D-AfterExt*:

$\langle \mathcal{D}\ (P\ \text{afterExt}\ e) = (\text{if } e = \checkmark \wedge P = \perp \text{ then } \{s.\ \text{front-tickFree}\ s\}$
 else $\{tl\ s \mid s . s \in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = e\}) \rangle$
 (is $\langle - = ?rhs \rangle$)
proof (unfold *AfterExt-def*, *split event.split*, *intro conjI allI impI*)
show $\langle e = ev\ x \implies \mathcal{D}\ (P\ \text{after}\ x) = ?rhs \rangle$ **for** x
by (*simp add: D-After*)
 (*metis Cons-in-T-imp-elem-ready-set D-T list.exhaust-sel*)
next
show $\langle e = \checkmark \implies \mathcal{D}\ (\text{if } P = \perp \text{ then } \perp \text{ else } STOP) = ?rhs \rangle$
by (*simp add: D-UU D-STOP BOT-iff-D*)
 (*metis D-imp-front-tickFree front-tickFree-implies-tickFree hd-append2*
hd-in-set is-processT9 nonTickFree-n-frontTickFree tickFree-def)
qed

lemma *T-AfterExt*:

$\langle \mathcal{T}\ (P\ \text{afterExt}\ e) = (\text{if } e = \checkmark \wedge P = \perp \text{ then } \{s.\ \text{front-tickFree}\ s\}$
 else $\text{insert } []\ \{tl\ s \mid s . s \in \mathcal{T}\ P \wedge s \neq [] \wedge hd\ s = e\}) \rangle$
 (is $\langle - = ?rhs \rangle$)
proof (unfold *AfterExt-def*, *split event.split*, *intro conjI allI impI*)
show $\langle e = ev\ x \implies \mathcal{T}\ (P\ \text{after}\ x) = ?rhs \rangle$ **for** x
by (*simp add: T-After set-eq-iff subset-iff*)
 (*metis Cons-in-T-imp-elem-ready-set list.collapse*
list.distinct(1) list.sel(1, 3) mem-Collect-eq ready-set-def)
next
show $\langle e = \checkmark \implies \mathcal{T}\ (\text{if } P = \perp \text{ then } \perp \text{ else } STOP) = ?rhs \rangle$
by (*simp add: T-After T-UU T-STOP subset-iff*)
 (*metis front-tickFree-charn hd-append2 hd-in-set*
is-processT2-TR list.sel(3) tickFree-def tl-append-if)
qed

5.1.3 Monotony

lemma *mono-AfterExt* : $\langle P \sqsubseteq Q \implies P\ \text{afterExt}\ e \sqsubseteq Q\ \text{afterExt}\ e \rangle$

by (*auto simp add: AfterExt-def mono-After split: event.split*)

lemma *mono-AfterExt-T* : $\langle P \sqsubseteq_T Q \implies e \neq \checkmark \vee P = \perp \vee Q \neq \perp \implies P \text{ afterExt } e \sqsubseteq_T Q \text{ afterExt } e \rangle$

by (*auto simp add: AfterExt-def mono-After-T split: event.split*)

lemma *mono-AfterExt-F* :

$\langle P \sqsubseteq_F Q \implies ev \ e \notin \text{ready-set } P \vee ev \ e \in \text{ready-set } Q \implies P \text{ afterExt } ev \ e \sqsubseteq_F Q \text{ afterExt } ev \ e \rangle$

by (*simp add: AfterExt-def mono-After-F*)

lemma *mono-AfterExt-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterExt } e \sqsubseteq_D Q \text{ afterExt } e \rangle$

by (*auto simp add: AfterExt-def mono-After-D split: event.split*)
(*meson BOT-iff-D divergence-refine-def subset-iff*)

lemma *mono-AfterExt-FD* :

$\langle P \sqsubseteq_{FD} Q \implies e \notin \text{ready-set } P \vee e \in \text{ready-set } Q \implies P \text{ afterExt } e \sqsubseteq_{FD} Q \text{ afterExt } e \rangle$

by (*auto simp add: AfterExt-def mono-After-FD FD-antisym split: event.split*)

lemma *mono-AfterExt-DT* : $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterExt } e \sqsubseteq_{DT} Q \text{ afterExt } e \rangle$

by (*auto simp add: AfterExt-def mono-After-DT split: event.split*)
(*meson BOT-iff-D divergence-refine-def leDT-imp-leD subsetD*)

5.1.4 Behaviour of *AfterExt* with *STOP*, *SKIP* and \perp

lemma *AfterExt-STOP*: $\langle \text{STOP afterExt } e = \text{STOP} \rangle$

by (*simp add: STOP-neq-BOT STOP-iff-T T-AfterExt ready-set-STOP T-STOP*)

lemma *AfterExt-is-STOP-iff*:

$\langle P \text{ afterExt } e = \text{STOP} \iff P \neq \perp \wedge (\forall s. e \# s \in \mathcal{T} P \longrightarrow s = []) \rangle$

apply (*cases e; simp add: AfterExt-def After-is-STOP-iff*)

by (*metis After-BOT After-is-STOP-iff STOP-neq-BOT*)

(*metis STOP-neq-BOT butlast.simps(2) front-tickFree-butlast is-processT2-TR tickFree-Cons*)

lemma *AfterExt-SKIP*: $\langle \text{SKIP afterExt } e = \text{STOP} \rangle$

by (*auto simp add: SKIP-neq-BOT STOP-iff-T T-AfterExt ready-set-SKIP T-SKIP*)

lemma *AfterExt-BOT* : $\langle \perp \text{ afterExt } e = \perp \rangle$

by (*force simp add: BOT-iff-D D-AfterExt D-UU*)

lemma *AfterExt-is-BOT-iff*: $\langle P \text{ afterExt } e = \perp \iff [e] \in \mathcal{D} P \rangle$

apply (*cases e; simp add: AfterExt-def After-is-BOT-iff*)

using *BOT-iff-D D-UU STOP-neq-BOT is-processT9-tick* **by** *fastforce*

5.1.5 Behaviour of *AfterExt* with Operators of HOL-CSP

Here again, we lose determinism.

lemma *AfterExt-Mprefix-is-AfterExt-Mndetprefix*:
 $\langle (\Box a \in A \rightarrow P a) \text{ afterExt } e = (\Box a \in A \rightarrow P a) \text{ afterExt } e \rangle$
by (*simp add: AfterExt-def After-Mprefix-is-After-Mndetprefix*
Mprefix-neq-BOT Mndetprefix-neq-BOT split: event.split)

lemma *AfterExt-Det-is-AfterExt-Ndet*: $\langle P \Box Q \text{ afterExt } e = P \Box Q \text{ afterExt } e \rangle$
by (*simp add: AfterExt-def After-Det-is-After-Ndet Det-is-BOT-iff Ndet-is-BOT-iff*
split: event.split)

lemma *AfterExt-Ndet*:
 $\langle P \Box Q \text{ afterExt } e = (\text{ if } e \notin \text{ ready-set } P \cup \text{ ready-set } Q \text{ then } STOP$
 $\text{ else case } e \text{ of } ev \ x \Rightarrow \text{ if } ev \ x \in \text{ ready-set } P \cap \text{ ready-set } Q$
 $\text{ then } (P \text{ afterExt } ev \ x) \Box (Q \text{ afterExt } ev \ x)$
 $\text{ else if } ev \ x \in \text{ ready-set } P$
 $\text{ then } P \text{ afterExt } ev \ x$
 $\text{ else } Q \text{ afterExt } ev \ x$
 $| \checkmark \Rightarrow \text{ if } P = \perp \vee Q = \perp \text{ then } \perp \text{ else } STOP \rangle$
by (*simp add: AfterExt-def Ndet-is-BOT-iff After-Ndet ready-set-BOT split:*
event.split)

lemma *AfterExt-Mprefix*:
 $\langle (\Box a \in A \rightarrow P a) \text{ afterExt } e =$
 $(\text{ case } e \text{ of } ev \ x \Rightarrow \text{ if } x \in A \text{ then } P x \text{ else } STOP | \checkmark \Rightarrow STOP) \rangle$
by (*simp add: AfterExt-def Mprefix-neq-BOT After-Mprefix split: event.split*)

corollary *AfterExt-prefix*:
 $\langle (a \rightarrow P) \text{ afterExt } e =$
 $(\text{ case } e \text{ of } ev \ x \Rightarrow \text{ if } x = a \text{ then } P \text{ else } STOP | \checkmark \Rightarrow STOP) \rangle$
unfolding *write0-def* **by** (*simp add: AfterExt-Mprefix split: event.split*)

lemmas *AfterExt-Det = AfterExt-Ndet* [*folded AfterExt-Det-is-AfterExt-Ndet*]
and *AfterExt-Mndetprefix = AfterExt-Mprefix*
[*unfolded AfterExt-Mprefix-is-AfterExt-Mndetprefix*]

lemma *Renaming-is-BOT-iff*: $\langle \text{Renaming } P \ f = \perp \longleftrightarrow P = \perp \rangle$
apply (*intro iffI*)
apply (*simp add: BOT-iff-D D-Renaming*)
by (*simp add: Renaming-BOT*)

lemma *Renaming-is-STOP-iff*: $\langle \text{Renaming } P \ f = STOP \longleftrightarrow P = STOP \rangle$
apply (*intro iffI, simp-all add: Renaming-STOP*)

by (auto simp add: STOP-iff-T T-Renaming intro: Nil-lem-T)

lemma *AfterExt-Renaming*:

⟨Renaming P f afterExt e =
 (if $P = \perp$ then \perp else
 $\sqcap a \in \{a. \text{ev } a \in \text{ready-set } P \wedge \text{ev } (f \ a) = e\}. \text{Renaming } (P \text{ afterExt } \text{ev } a) \ f)$ ⟩
 by (simp add: AfterExt-def After-Renaming Renaming-is-BOT-iff split: event.split)

— Move this result in HOL-CSP

lemma *Seq-is-BOT-iff*: ⟨ $P ; Q = \perp \iff P = \perp \vee ([\checkmark] \in \mathcal{T} \ P \wedge Q = \perp)$ ⟩
 by (auto simp add: BOT-iff-D D-Seq D-UU)

lemma *AfterExt-Seq*:

⟨ $(P ; Q)$ afterExt e =
 (if $e \notin \text{ready-set } P \wedge e \notin \text{ready-set } Q$ then STOP
 else if $e \notin \text{ready-set } Q$ then P afterExt $e ; Q$
 else if $e \notin \text{ready-set } P$ then if $\checkmark \in \text{ready-set } P$ then Q afterExt e else STOP
 else if $\checkmark \in \text{ready-set } P$ then $(P \text{ afterExt } e ; Q) \sqcap (Q \text{ afterExt } e)$
 else P afterExt $e ; Q$)⟩
 by (simp add: AfterExt-def After-Seq Ndet-id Ndet-is-BOT-iff
 Seq-is-BOT-iff ready-set-def T-UU STOP-Seq split: event.split)

theorem *AfterExt-Sync*:

⟨ $(P \llbracket S \rrbracket Q)$ afterExt e =
 (if $P = \perp \vee Q = \perp$ then \perp
 else case e of $\checkmark \Rightarrow$ STOP
 | $\text{ev } x \Rightarrow$ if $e \in \text{ready-set } P \wedge e \in \text{ready-set } Q$
 then if $x \in S$
 then P afterExt $e \llbracket S \rrbracket Q$ afterExt e
 else $(P \text{ afterExt } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ afterExt } e)$
 else if $e \in \text{ready-set } P$
 then if $x \in S$ then STOP else P afterExt $e \llbracket S \rrbracket Q$
 else if $e \in \text{ready-set } Q$
 then if $x \in S$ then STOP else $P \llbracket S \rrbracket Q$ afterExt e
 else STOP)⟩
 by (simp add: AfterExt-def After-Sync Sync-is-BOT-iff split: event.split)

lemma *Hiding-FD-Hiding-AfterExt-if-ready-inside*:

⟨ $e \in B \implies (P \setminus B) \sqsubseteq_{FD} (P \text{ afterExt } \text{ev } e \setminus B)$ ⟩

and *AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin*:

⟨ $e \notin B \implies (P \setminus B) \text{ afterExt } \text{ev } e \sqsubseteq_{FD} (P \text{ afterExt } \text{ev } e \setminus B)$ ⟩

if ready: ⟨ $\text{ev } e \in \text{ready-set } P$ ⟩

by (*simp add: AfterExt-def Hiding-FD-Hiding-After-if-ready-inside that*)
(*simp add: AfterExt-def After-Hiding-FD-Hiding-After-if-ready-notin that*)

lemmas *Hiding-F-Hiding-AfterExt-if-ready-inside* =
Hiding-FD-Hiding-AfterExt-if-ready-inside[THEN leFD-imp-leF]
and *AfterExt-Hiding-F-Hiding-AfterExt-if-ready-notin* =
AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin[THEN leFD-imp-leF]
and *Hiding-D-Hiding-AfterExt-if-ready-inside* =
Hiding-FD-Hiding-AfterExt-if-ready-inside[THEN leFD-imp-leD]
and *AfterExt-Hiding-D-Hiding-AfterExt-if-ready-notin* =
AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin[THEN leFD-imp-leD]
and *Hiding-T-Hiding-AfterExt-if-ready-inside* =
Hiding-FD-Hiding-AfterExt-if-ready-inside[THEN leFD-imp-leF, THEN leF-imp-leT]

and *AfterExt-Hiding-T-Hiding-AfterExt-if-ready-notin* =
AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin[THEN leFD-imp-leF, THEN leF-imp-leT]

corollary *Hiding-DT-Hiding-AfterExt-if-ready-inside*:
 $\langle ev\ e \in\ ready\text{-set}\ P \implies e \in B \implies (P \setminus B) \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
and *AfterExt-Hiding-DT-Hiding-AfterExt-if-ready-notin*:
 $\langle ev\ e \in\ ready\text{-set}\ P \implies e \notin B \implies (P \setminus B)\ afterExt\ ev\ e \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
by (*simp add: Hiding-D-Hiding-AfterExt-if-ready-inside*
Hiding-T-Hiding-AfterExt-if-ready-inside leD-leT-imp-leDT)
(*simp add: AfterExt-Hiding-D-Hiding-AfterExt-if-ready-notin*
AfterExt-Hiding-T-Hiding-AfterExt-if-ready-notin leD-leT-imp-leDT)

5.1.6 Behaviour of *AfterExt* with Operators of HOL-CSPM

lemma *AfterExt-MultiDet-is-AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\Box\ a \in A.\ P\ a)\ afterExt\ e = (\Box\ a \in A.\ P\ a)\ afterExt\ e \rangle$
by (*auto simp add: AfterExt-def After-MultiDet-is-After-MultiNdet*
MultiDet-is-BOT-iff MultiNdet-is-BOT-iff split: event.split)

lemma *AfterExt-GlobalNdet*:
 $\langle (\Box\ a \in A.\ P\ a)\ afterExt\ e = (if\ e \notin (\bigcup\ a \in A.\ ready\text{-set}\ (P\ a))\ then\ STOP\ else\ (\Box\ a \in \{a \in A.\ e \in ready\text{-set}\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
and *AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\Box\ a \in A.\ P\ a)\ afterExt\ e = (if\ e \notin (\bigcup\ a \in A.\ ready\text{-set}\ (P\ a))\ then\ STOP\ else\ (\Box\ a \in \{a \in A.\ e \in ready\text{-set}\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
and *AfterExt-MultiDet*:
 $\langle finite\ A \implies (\Box\ a \in A.\ P\ a)\ afterExt\ e = (if\ e \notin (\bigcup\ a \in A.\ ready\text{-set}\ (P\ a))\ then\ STOP\ else\ (\Box\ a \in \{a \in A.\ e \in ready\text{-set}\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$

by ((cases e, simp-all add: AfterExt-def,
auto simp add: AfterExt-def ready-set-BOT After-GlobalNdet GlobalNdet-is-BOT-iff
After-MultiNdet After-MultiDet MultiDet-is-BOT-iff MultiN-
det-is-BOT-iff,
metis UNIV-I ready-set-BOT)[1])+

5.1.7 Behaviour of AfterExt with Operators of HOL-CSP_OpSem

lemma *AfterExt-Sliding-is-AfterExt-Ndet*:

$\langle P \triangleright Q \text{ afterExt } e = P \sqcap Q \text{ afterExt } e \rangle$

by (simp add: AfterExt-def After-Sliding-is-After-Ndet Sliding-is-BOT-iff Ndet-is-BOT-iff
split: event.split)

lemmas *AfterExt-Sliding = AfterExt-Ndet*[folded *AfterExt-Sliding-is-AfterExt-Ndet*]

lemma *AfterExt-Throw*:

$\langle (P \Theta a \in A. Q a) \text{ afterExt } e =$
(if $P = \perp$ then \perp
else case e of $\checkmark \Rightarrow STOP$
| $ev x \Rightarrow$ if $x \in \text{ready-set } P$ then if $x \in A$ then $Q x$
else $(P \text{ after } x) \Theta a \in A. Q a$ else $STOP) \rangle$

by (simp add: AfterExt-def After-Throw Throw-is-BOT-iff split: event.split)

lemma *AfterExt-Interrupt*:

$\langle (P \Delta Q) \text{ afterExt } e =$
(if $P = \perp \vee Q = \perp$ then \perp
else case e of $\checkmark \Rightarrow STOP$
| $ev x \Rightarrow$ if $ev x \in \text{ready-set } P \wedge ev x \in \text{ready-set } Q$
then $(Q \text{ after } x) \sqcap (P \text{ after } x \Delta Q)$
else if $ev x \in \text{ready-set } P \wedge ev x \notin \text{ready-set } Q$
then $P \text{ after } x \Delta Q$
else if $ev x \notin \text{ready-set } P \wedge ev x \in \text{ready-set } Q$
then $Q \text{ after } x$
else $STOP) \rangle$

by (simp add: AfterExt-def After-Interrupt Interrupt-is-BOT-iff
Ndet-is-BOT-iff ready-set-BOT After-is-BOT-iff D-UU split: event.split)

5.1.8 Behaviour of AfterExt with Reference Processes

lemma *AfterExt-DF*:

$\langle DF A \text{ afterExt } e =$
(case e of $ev x \Rightarrow$ if $x \in A$ then $DF A$ else $STOP$ | $\checkmark \Rightarrow STOP) \rangle$

by (cases e) (simp-all add: AfterExt-def After-DF BOT-iff-D div-free-DF)

lemma *AfterExt-DF_SKIP*:

$\langle DF_{SKIP} A \text{ afterExt } e =$
(case e of $ev x \Rightarrow$ if $x \in A$ then $DF_{SKIP} A$ else $STOP$ | $\checkmark \Rightarrow STOP) \rangle$

by (cases e) (simp-all add: AfterExt-def After-DF_{SKIP} BOT-iff-D div-free-DF_{SKIP})

lemma AfterExt-RUN:

⟨RUN A afterExt e =
 (case e of ev x ⇒ if x ∈ A then RUN A else STOP | ✓ ⇒ STOP)⟩
by (cases e) (simp-all add: AfterExt-def After-RUN BOT-iff-D div-free-RUN)

lemma AfterExt-CHAOS:

⟨CHAOS A afterExt e =
 (case e of ev x ⇒ if x ∈ A then CHAOS A else STOP | ✓ ⇒ STOP)⟩
by (cases e) (simp-all add: AfterExt-def After-CHAOS BOT-iff-D div-free-CHAOS)

lemma AfterExt-CHAOS_{SKIP}:

⟨CHAOS_{SKIP} A afterExt e =
 (case e of ev x ⇒ if x ∈ A then CHAOS_{SKIP} A else STOP | ✓ ⇒ STOP)⟩
by (cases e) (simp-all add: AfterExt-def After-CHAOS_{SKIP} BOT-iff-D div-free-CHAOS_{SKIP})

lemma DF-FD-AfterExt:

⟨DF A ⊆_{FD} P ⇒ e ∈ ready-set P ⇒ DF A ⊆_{FD} P afterExt e⟩
apply (cases e, simp add: AfterExt-def DF-FD-After)
by (metis anti-mono-ready-set-FD event.distinct(1) image-iff ready-set-DF subsetD)

lemma DF_{SKIP}-FD-AfterExt:

⟨DF_{SKIP} A ⊆_{FD} P ⇒ ev e ∈ ready-set P ⇒ DF_{SKIP} A ⊆_{FD} P afterExt ev e⟩
by (simp add: AfterExt-def DF_{SKIP}-FD-After)

lemma deadlock-free-AfterExt:

⟨deadlock-free P ⇒ deadlock-free (P afterExt e) ↔
 (if e ∈ ready-set P ∧ e ≠ ✓ then True else False)⟩
by (cases e)
 (simp add: AfterExt-def deadlock-free-After,
 simp add: AfterExt-def BOT-iff-D deadlock-free-implies-div-free non-deadlock-free-STOP)

lemma deadlock-free_{SKIP}-AfterExt:

⟨deadlock-free_{SKIP} P ⇒ deadlock-free_{SKIP} (P afterExt e) ↔
 (if e ∈ ready-set P ∧ e ≠ ✓ then True else False)⟩
by (cases e)
 (simp add: AfterExt-def deadlock-free_{SKIP}-After,
 simp add: AfterExt-def BOT-iff-D deadlock-free_{SKIP}-implies-div-free non-deadlock-free_{SKIP}-STOP)

end

5.2 The AfterTrace Operator

```

theory AfterTrace
  imports AfterExt HOL-CSPM.DeadlockResults
begin

```

5.2.1 Definition

We just defined $P \text{ afterExt } e$ for $P::'\alpha$ process and $e::'\alpha$ event. Since a trace of a P is just an $'\alpha$ event list, the following inductive definition is natural.

```

fun AfterTrace ::  $\langle '\alpha \text{ process} \Rightarrow '\alpha \text{ trace} \Rightarrow '\alpha \text{ process} \rangle$  (infixl  $\langle \text{afterTrace} \rangle$  77)
  where  $\langle P \text{ afterTrace } [] = P \rangle$ 
  |  $\langle P \text{ afterTrace } (e \# s) = P \text{ afterExt } e \text{ afterTrace } s \rangle$ 

```

We can also induct backward.

```

lemma AfterTrace-append:  $\langle P \text{ afterTrace } (s @ t) = P \text{ afterTrace } s \text{ afterTrace } t \rangle$ 
  apply (induct t rule: rev-induct, simp)
  apply (induct s rule: rev-induct, simp)
  by (metis AfterTrace.simps append.assoc append.right-neutral append-Cons append-Nil)

```

```

lemma AfterTrace-snoc :  $\langle P \text{ afterTrace } (s @ [e]) = P \text{ afterTrace } s \text{ afterExt } e \rangle$ 
  by (simp add: AfterTrace-append)

```

We have some immediate properties.

```

lemma AfterTrace-BOT :  $\langle \perp \text{ afterTrace } s = \perp \rangle$ 
  by (induct s) (simp-all add: AfterExt-BOT)

```

```

lemma AfterTrace-STOP :  $\langle \text{STOP} \text{ afterTrace } s = \text{STOP} \rangle$ 
  by (induct s) (simp-all add: AfterExt-STOP)

```

```

lemma AfterTrace-SKIP :  $\langle \text{SKIP} \text{ afterTrace } s = (\text{if } s = [] \text{ then SKIP else STOP}) \rangle$ 
  by (induct s) (simp-all add: AfterExt-SKIP AfterTrace-STOP)

```

5.2.2 Projections

```

lemma F-AfterTrace :  $\langle (s @ t, X) \in \mathcal{F} P \Longrightarrow (t, X) \in \mathcal{F} (P \text{ afterTrace } s) \rangle$ 
  apply (induct s arbitrary: t rule: rev-induct, simp)
  apply (simp add: AfterTrace-snoc F-AfterExt)
  by (metis Cons-in-T-imp-elem-ready-set F-T butlast-tl front-tickFree-butlast
    is-processT2 list.distinct(1) list.sel(1, 3) tickFree-tl)

```

```

lemma D-AfterTrace :  $\langle s @ t \in \mathcal{D} P \Longrightarrow t \in \mathcal{D} (P \text{ afterTrace } s) \rangle$ 
  apply (induct s arbitrary: t rule: rev-induct, simp)
  apply (simp add: AfterTrace-snoc D-AfterExt)
  by (metis D-imp-front-tickFree butlast-tl front-tickFree-butlast
    list.distinct(1) list.sel(1, 3) tickFree-tl)

```

lemma *T-AfterTrace* : $\langle s @ t \in \mathcal{T} P \implies t \in \mathcal{T} (P \text{ afterTrace } s) \rangle$
using *F-AfterTrace T-F-spec* **by** *blast*

corollary *ready-set-AfterTrace* :
 $\langle s @ e \# t \in \mathcal{T} P \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
by (*metis Cons-in-T-imp-elem-ready-set T-AfterTrace*)

corollary *F-imp-R-AfterTrace*: $\langle (s, X) \in \mathcal{F} P \implies X \in \mathcal{R} (P \text{ afterTrace } s) \rangle$
by (*simp add: F-AfterTrace Refusals-iff*)

corollary *D-imp-AfterTrace-is-BOT*: $\langle s \in \mathcal{D} P \implies P \text{ afterTrace } s = \perp \rangle$
by (*simp add: BOT-iff-D D-AfterTrace*)

5.2.3 Monotony

lemma *mono-AfterTrace* : $\langle P \sqsubseteq Q \implies P \text{ afterTrace } s \sqsubseteq Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct*) (*simp-all add: mono-AfterExt AfterTrace-snoc*)

lemma *mono-AfterTrace-T* :
 $\langle P \sqsubseteq_T Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_T Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc*)
(rule mono-AfterExt-T; use is-processT3-ST in blast)

lemma *mono-AfterTrace-F* :
 $\langle P \sqsubseteq_F Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_F Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc*)
(metis event.exhaust is-processT3-ST mono-AfterExt-F ready-set-AfterTrace)

lemma *mono-AfterTrace-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterTrace } s \sqsubseteq_D Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct*) (*simp-all add: mono-AfterExt-D AfterTrace-snoc*)

lemma *mono-AfterTrace-FD* :
 $\langle P \sqsubseteq_{FD} Q \implies s \in \mathcal{T} Q \implies P \text{ afterTrace } s \sqsubseteq_{FD} Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc*)
(meson Cons-in-T-imp-elem-ready-set T-AfterTrace is-processT3-ST mono-AfterExt-FD)

lemma *mono-AfterTrace-DT* :
 $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterTrace } s \sqsubseteq_{DT} Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc mono-AfterExt-DT*)

5.2.4 Another Definition of *events-of*

inductive *reachable-event* :: $\langle 'a \text{ process} \Rightarrow 'a \Rightarrow \text{bool} \rangle$
where $\langle \text{ev } e \in \text{ready-set } P \implies \text{reachable-event } P \ e \rangle$
 $\mid \langle \text{reachable-event } (P \text{ after } f) \ e \implies \text{reachable-event } P \ e \rangle$

lemma *events-of-iff-reachable-event*: $\langle e \in \text{events-of } P \longleftrightarrow \text{reachable-event } P \ e \rangle$
proof (*intro iffI*)

```

show  $\langle \text{reachable-event } P \ e \implies e \in \text{events-of } P \rangle$ 
  apply (induct rule: reachable-event.induct;
    simp add: T-After events-of-def ready-set-def split: if-split-asm)
  by (meson list.set-intros(1)) (meson list.set-sel(2))
next
  assume  $\langle e \in \text{events-of } P \rangle$ 
  then obtain  $s$  where  $*$  :  $\langle s \in \mathcal{T} \ P \rangle \langle \text{ev } e \in \text{set } s \rangle$  unfolding events-of-def by
blast
  thus  $\langle \text{reachable-event } P \ e \rangle$ 
  proof (induct s arbitrary: P)
    show  $\langle \bigwedge P \ e. \text{ev } e \in \text{set } [] \implies \text{reachable-event } P \ e \rangle$  by simp
  next
    case (Cons f s)
    from Cons.prem(1) is-processT3-ST
    have  $*$  :  $\langle f \in \text{ready-set } P \rangle$  unfolding ready-set-def by force
    from Cons.prem(2) consider  $\langle f = \text{ev } e \rangle \mid \langle \text{ev } e \in \text{set } s \rangle$  by auto
    thus  $\langle \text{reachable-event } P \ e \rangle$ 
    proof cases
      assume  $\langle f = \text{ev } e \rangle$ 
      from  $*$ [simplified this]
      show  $\langle \text{reachable-event } P \ e \rangle$  by (rule reachable-event.intros(1))
    next
      assume  $\langle \text{ev } e \in \text{set } s \rangle$ 
      show  $\langle \text{reachable-event } P \ e \rangle$ 
      proof (cases f)
        fix  $f'$ 
        assume  $\langle f = \text{ev } f' \rangle$ 
        with  $*$  Cons.prem(1) have  $\langle s \in \mathcal{T} \ (P \ \text{after } f') \rangle$  by (force simp add:
T-After)
        show  $\langle \text{reachable-event } P \ e \rangle$ 
        apply (rule reachable-event.intros(2))
        by (rule Cons.hyps[OF  $\langle s \in \mathcal{T} \ (P \ \text{after } f') \rangle \langle \text{ev } e \in \text{set } s \rangle$ ])
      next
        from Cons.prem(1) have  $\langle f = \checkmark \implies s = [] \rangle$ 
        by simp (metis butlast.simps(2) front-tickFree-butlast is-processT2-TR
tickFree-Cons)
        thus  $\langle f = \checkmark \implies \text{reachable-event } P \ e \rangle$ 
        using  $\langle \text{ev } e \in \text{set } s \rangle$  by force
      qed
    qed
  qed

```

lemma *reachable-event-BOT*: $\langle \text{reachable-event } \perp \ e \rangle$
by (*simp add: reachable-event.intros(1) ready-set-BOT*)

lemma *not-reachable-event-STOP*: $\langle \neg \text{reachable-event } \text{STOP} \ e \rangle$
by (*subst events-of-iff-reachable-event[symmetric]*)

(*unfold events-of-def, simp add: T-STOP*)

lemma *reachable-tick-SKIP*: $\langle \neg \text{reachable-event SKIP } \checkmark \rangle$
by (*subst events-of-iff-reachable-event[symmetric]*)
(*unfold events-of-def, simp add: T-SKIP*)

lemma *reachable-event-iff-in-ready-set-AfterTrace*:

$\langle \text{reachable-event } P e \longleftrightarrow e \in \{e \mid e s. \text{ev } e \in \text{ready-set } (P \text{ afterTrace } s)\} \rangle$

proof –

have $\langle \text{reachable-event } P e \implies \exists s. \text{ev } e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$

proof (*induct rule: reachable-event.induct*)

case (1 *e P*)

thus ?*case* **by** (*metis AfterTrace.simps(1)*)

next

case (2 *P f e*)

from 2.*hyps*(2) **obtain** *s* **where** $\langle \text{ev } e \in \text{ready-set } (P \text{ after } f \text{ afterTrace } s) \rangle$ **by**
blast

hence $\langle \text{ev } e \in \text{ready-set } (P \text{ afterTrace } (\text{ev } f \# s)) \rangle$ **by** (*simp add: AfterExt-def*)

thus ?*case* **by** *blast*

qed

also have $\langle \text{ev } e \in \text{ready-set } (P \text{ afterTrace } s) \implies \text{reachable-event } P e \rangle$ **for** *s*

proof (*induct s arbitrary: P*)

show $\langle \text{ev } e \in \text{ready-set } (P \text{ afterTrace } []) \implies \text{reachable-event } P e \rangle$ **for** *P*

by (*simp add: reachable-event.intros(1)*)

next

fix *f s P*

assume *hyp* : $\langle \text{ev } e \in \text{ready-set } (P \text{ afterTrace } s) \implies \text{reachable-event } P e \rangle$ **for**
P

assume *prem* : $\langle \text{ev } e \in \text{ready-set } (P \text{ afterTrace } (f \# s)) \rangle$

show $\langle \text{reachable-event } P e \rangle$

proof (*cases f*)

fix *f'*

assume $\langle f = \text{ev } f' \rangle$

show $\langle \text{reachable-event } P e \rangle$ **by** (*rule reachable-event.intros(2)[OF hyp]*)

(*use prem in <simp add: AfterExt-def <f = ev f'>*)

next

case *tick*

with *prem* **show** $\langle f = \checkmark \implies \text{reachable-event } P e \rangle$

by (*simp add: AfterExt-def reachable-event-BOT AfterTrace-STOP ready-set-STOP*)

split: if-split-asm)

qed

qed

ultimately show ?*thesis* **by** *blast*

qed

5.2.5 Characterizations for Deadlock Freeness

lemma *deadlock-free-AfterExt-characterization:*

$\langle \text{deadlock-free } P \longleftrightarrow \text{range } ev \notin \mathcal{R} P \wedge$
 $(\forall e \in \text{ready-set } P. \text{ deadlock-free } (P \text{ afterExt } e)) \rangle$
 (is $\langle \text{deadlock-free } P \longleftrightarrow ?rhs \rangle$)

proof (intro iffI)

have $\langle \text{range } ev \notin \mathcal{R} P \longleftrightarrow \text{UNIV} - \{\checkmark\} \notin \mathcal{R} P \rangle$

by (metis Diff-insert-absorb UNIV-eq-I event.simps(3) event-set image-iff)

thus $\langle \text{deadlock-free } P \implies ?rhs \rangle$

by (metis DF-FD-AfterExt Diff-Diff-Int Diff-partition Diff-subset F-T
 deadlock-free_SKIP-is-right deadlock-free-def
 deadlock-free-implies-non-terminating deadlock-free-is-deadlock-free_SKIP
 inf-top-left is-processT5-S2a non-tickFree-tick Refusals-iff self-append-conv2)

next

assume *assm* : $\langle ?rhs \rangle$

with BOT-iff-D process-charn **have** non-BOT : $\langle P \neq \perp \rangle$ **by** (metis Refusals-iff)

show $\langle \text{deadlock-free } P \rangle$

proof (unfold deadlock-free-F failure-refine-def, safe)

from *assm* **have** * : $\langle \text{range } ev \notin \mathcal{R} P \rangle$

$\langle e \in \text{ready-set } P \implies$

$\{(tl\ s, X) \mid s\ X. (s, X) \in \mathcal{F} P \wedge s \neq [] \wedge hd\ s = e\} \subseteq \mathcal{F} (DF\ UNIV)\rangle$ **for** *e*

by (simp-all add: deadlock-free-F failure-refine-def F-AfterExt non-BOT)

fix *s X*

assume ** : $\langle (s, X) \in \mathcal{F} P \rangle$

show $\langle (s, X) \in \mathcal{F} (DF\ UNIV) \rangle$

proof (cases *s*)

show $\langle s = [] \implies (s, X) \in \mathcal{F} (DF\ UNIV) \rangle$

by (subst F-DF, simp)

(metis *(1) ** Refusals-iff image-subset-iff is-processT4)

next

fix *e s'*

assume *** : $\langle s = e \# s' \rangle$

with ** Cons-in-T-imp-elem-ready-set F-T **have** $\langle e \in \text{ready-set } P \rangle$ **by** blast

with *(2)[OF this] **show** $\langle (s, X) \in \mathcal{F} (DF\ UNIV) \rangle$

by (subst F-DF, simp add: subset-iff)

(metis (no-types, lifting) *(1) ** *** CollectD Refusals-iff
 event-set hd-append2 hd-in-set in-set-conv-decomp-first insert-iff
 is-processT6-S2 list.sel(1) list.set-intros(1) rangeE ready-set-def)

qed

qed

qed

lemma *deadlock-free_SKIP-AfterExt-characterization:*

$\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow$
 $\text{UNIV} \notin \mathcal{R} P \wedge (\forall e \in \text{ready-set } P - \{\checkmark\}. \text{ deadlock-free}_{SKIP} (P \text{ afterExt } e)) \rangle$
 (is $\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow ?rhs \rangle$)

proof (intro iffI)

```

show ⟨deadlock-freeSKIP P ⇒ ?rhs⟩
  by (metis Diff-iff Nil-elem-T deadlock-freeSKIP-AfterExt
      deadlock-freeSKIP-is-right insertI1 Refusals-iff tickFree-Nil)
next
assume assm : ⟨?rhs⟩
with BOT-iff-D process-charn have non-BOT : ⟨P ≠ ⊥⟩ by (metis Refusals-iff)
show ⟨deadlock-freeSKIP P⟩
proof (unfold deadlock-freeSKIP-def failure-refine-def, safe)
  from assm have * : ⟨UNIV ∉ ℛ P⟩
  ⟨e ∈ ready-set P ∧ e ≠ ✓ ⇒
    {(tl s, X) | s X. (s, X) ∈ ℱ P ∧ s ≠ [] ∧ hd s = e} ⊆ ℱ (DFSKIP UNIV)⟩
for e
  by (simp-all add: deadlock-freeSKIP-def failure-refine-def F-AfterExt non-BOT)

fix s X
assume ** : ⟨(s, X) ∈ ℱ P⟩
show ⟨(s, X) ∈ ℱ (DFSKIP UNIV)⟩
proof (cases s)
  show ⟨s = [] ⇒ (s, X) ∈ ℱ (DFSKIP UNIV)⟩
  by (subst F-DFSKIP, simp)
  (metis *(1) ** UNIV-eq-I event.exhaust Refusals-iff)
next
fix e s'
assume *** : ⟨s = e # s'⟩
show ⟨(s, X) ∈ ℱ (DFSKIP UNIV)⟩
proof (cases e)
  fix e'
  assume ⟨e = ev e'⟩
  with ** *** Cons-in-T-imp-elem-ready-set F-T
  have ⟨e ∈ ready-set P ∧ e ≠ ✓⟩ by blast
  with *(2)[OF this] show ⟨(s, X) ∈ ℱ (DFSKIP UNIV)⟩
  by (subst F-DFSKIP, simp add: subset-iff)
  (metis ** *** ⟨e = ev e'⟩ list.distinct(1) list.sel(1))
next
assume ⟨e = ✓⟩
hence ⟨s = [✓]⟩
  by (metis ** *** F-T append-butlast-last-id append-single-T-imp-tickFree
      butlast.simps(2) list.distinct(1) tickFree-Cons)
thus ⟨(s, X) ∈ ℱ (DFSKIP UNIV)⟩
  by (subst F-DFSKIP, simp)
qed
qed
qed
qed
end

```

Chapter 6

Motivations for our Definitions

```
theory Motivations
  imports AfterExt
begin
```

To construct our bridge between denotational and operational semantics, we want to define two kind of transitions:

- without external event: $P \rightsquigarrow_{\tau} P'$
- with an external event e (which can possibly be \checkmark): $P \rightsquigarrow_e P'$.

We will discuss in this theory some fundamental properties that we want $P \rightsquigarrow_{\tau} Q$ and $P \rightsquigarrow_e Q$ to verify, and on the consequences that this will have.

Let's say we want to define the τ transition as an inductive predicate with three introduction rules:

- we allow a process to make a τ transition towards itself: $P \rightsquigarrow_{\tau} P$
- the non-deterministic choice (\sqcap) can make a τ transition to the left side $P \sqcap Q \rightsquigarrow_{\tau} P$
- the non-deterministic choice (\sqcap) can make a τ transition to the right side $P \sqcap Q \rightsquigarrow_{\tau} Q$.

```
inductive  $\tau$ -trans ::  $\langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$  (infixl  $\langle \rightsquigarrow_{\tau} \rangle$  50)
  where  $\tau$ -trans-eq   :  $\langle P \rightsquigarrow_{\tau} P \rangle$ 
  |  $\tau$ -trans-NdetL :  $\langle P \sqcap Q \rightsquigarrow_{\tau} P \rangle$ 
```

| $\tau\text{-trans-NdetR} : \langle P \sqcap Q \rightsquigarrow_{\tau} Q \rangle$

— We can obtain the same inductive predicate by removing $\tau\text{-trans-eq}$ and $\tau\text{-trans-NdetR}$ clauses (because of (\sqcap) properties).

With this definition, we immediately show that the τ transition is the FD-refinement (\sqsubseteq_{FD}) .

lemma $\tau\text{-trans-is-FD} : \langle (\rightsquigarrow_{\tau}) = (\sqsubseteq_{FD}) \rangle$

apply (*intro ext iffI*)

by (*metis mono-Ndet-FD-left Ndet-commute $\tau\text{-trans.simps idem-FD}$*)

(*metis mono-Ndet-FD mono-Ndet-FD-right*)

FD-antisym Ndet-id $\tau\text{-trans-NdetL idem-FD}$)

The definition of the event transition will be a little bit more complex.

First of all we want to prevent a process $P::'\alpha$ process to make a transition with $e::'\alpha$ event if P can not begin with e .

More formally, we want $P \rightsquigarrow_e P' \implies e \in \text{ready-set } P$.

Moreover, we want the event transitions to absorb the τ transitions.

Finally, when $e \in \text{ready-set } P$, we want to have $P \rightsquigarrow_e P \text{ afterExt } e$.

This brings us to the following inductive definition:

inductive *event-trans-prem* :: $\langle '\alpha \text{ process} \Rightarrow '\alpha \text{ event} \Rightarrow '\alpha \text{ process} \Rightarrow \text{bool} \rangle$

where

$\tau\text{-left-absorb} : \langle [e \in \text{ready-set } P'; P \rightsquigarrow_{\tau} P'; \text{event-trans-prem } P' e P''] \implies \text{event-trans-prem } P e P'' \rangle$

| $\tau\text{-right-absorb} : \langle [e \in \text{ready-set } P; \text{event-trans-prem } P e P'; P' \rightsquigarrow_{\tau} P''] \implies \text{event-trans-prem } P e P'' \rangle$

| *ready-trans-to-AfterExt* : $\langle e \in \text{ready-set } P \implies \text{event-trans-prem } P e (P \text{ afterExt } e) \rangle$

abbreviation *event-trans* :: $\langle '\alpha \text{ process} \Rightarrow '\alpha \text{ event} \Rightarrow '\alpha \text{ process} \Rightarrow \text{bool} \rangle$

($\langle _ \rightsquigarrow _ / _ \rightarrow [50, 3, 51] 50 \rangle$)

where $\langle P \rightsquigarrow_e P' \equiv e \in \text{ready-set } P \wedge \text{event-trans-prem } P e P' \rangle$

We immediately show that this event transition definition is equivalent to the following:

lemma *startable-imp-ev-trans-is-startable-and-FD-After*:

$\langle (P \rightsquigarrow_e P') \longleftrightarrow e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow_{\tau} P' \rangle$

proof *safe*

show $\langle e \in \text{ready-set } P \implies \text{event-trans-prem } P e P' \implies P \text{ afterExt } e \rightsquigarrow_{\tau} P' \rangle$

apply (*rotate-tac, induct rule: event-trans-prem.induct*)

apply (*metis $\tau\text{-trans-is-FD mono-AfterExt-FD trans-FD}$*)

apply (*metis $\tau\text{-trans-is-FD trans-FD}$*)

by (*simp add: $\tau\text{-trans-is-FD}$*)

next

show $\langle e \in \text{ready-set } P \implies P \text{ afterExt } e \rightsquigarrow_{\tau} P' \implies \text{event-trans-prem } P e P' \rangle$

by (*rule $\tau\text{-right-absorb}$ [of $e P \langle P \text{ afterExt } e \rangle P'$]*)

(*simp-all add: ready-trans-to-AfterExt τ -trans-is-FD*)

qed

With these two results, we are encouraged in the following theories to define:

- $P \rightsquigarrow_{\tau} Q \equiv P \sqsubseteq_{FD} Q$
- $P \rightsquigarrow_e Q \equiv e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow_{\tau} Q$

and possible variations with other refinements.

end

Chapter 7

Generic Operational Semantics as a Locale

```
theory OpSemGeneric
  imports AfterTrace
begin
```

7.1 Definition

We define a correspondence pattern using a locale in order to factorize the redundant proofs.

```
locale OpSemGeneric =
  fixes  $\tau$ -trans ::  $\langle [\alpha \text{ process}, \alpha \text{ process}] \Rightarrow \text{bool} \rangle$  (infixl  $\langle \rightsquigarrow_\tau \rangle$  50)
  assumes  $\tau$ -trans-NdetL:  $\langle P \sqcap Q \rightsquigarrow_\tau P \rangle$ 
    and  $\tau$ -trans-transitivity:  $\langle P \rightsquigarrow_\tau Q \Longrightarrow Q \rightsquigarrow_\tau R \Longrightarrow P \rightsquigarrow_\tau R \rangle$ 
    and  $\tau$ -trans-anti-mono-ready-set:  $\langle P \rightsquigarrow_\tau Q \Longrightarrow \text{ready-set } Q \subseteq \text{ready-set } P \rangle$ 
    and  $\tau$ -trans-mono-AfterExt:
       $\langle e \in \text{ready-set } Q \Longrightarrow P \rightsquigarrow_\tau Q \Longrightarrow P \text{ afterExt } e \rightsquigarrow_\tau Q \text{ afterExt } e \rangle$ 
begin
```

This locale needs to be instantiated with a binary relation (\rightsquigarrow_τ) which:

- is compatible with (\sqcap)
- is transitive
- makes *ready-set* anti-monotonic
- makes *AfterExt* monotonic.

From the τ transition $P \rightsquigarrow_\tau Q$ we derive the event transition as follows:

```
abbreviation event-trans ::  $\langle [\alpha \text{ process}, \alpha \text{ event}, \alpha \text{ process}] \Rightarrow \text{bool} \rangle$ 
  ( $\langle - / \rightsquigarrow - / \rightarrow [50, 3, 51] 50 \rangle$ )
```

where $\langle P \rightsquigarrow_e Q \equiv e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow_\tau Q \rangle$

From idempotence, commutativity and \perp absorbance of (\sqcap) , we get the following free of charge.

lemma τ -trans-eq: $\langle P \rightsquigarrow_\tau P \rangle$
and τ -trans-NdetR: $\langle P \sqcap Q \rightsquigarrow_\tau Q \rangle$
and BOT- τ -trans-anything: $\langle \perp \rightsquigarrow_\tau P \rangle$
and BOT-event-trans-anything: $\langle \perp \rightsquigarrow_e P \rangle$
by (*metis Ndet-id* τ -trans-NdetL,
metis Ndet-commute τ -trans-NdetL,
metis Ndet-BOT τ -trans-NdetL,
metis AfterExt-BOT Ndet-BOT UNIV-I τ -trans-NdetL *ready-set-BOT*)

As immediate consequences of the axioms, we prove that event transitions absorb τ transitions on right and on left.

lemma *event-trans- τ -trans*: $\langle P \rightsquigarrow_e P' \implies P' \rightsquigarrow_\tau P'' \implies P \rightsquigarrow_e P'' \rangle$
by (*meson* τ -trans-transitivity)

lemma τ -trans-event-trans: $\langle P \rightsquigarrow_\tau P' \implies P' \rightsquigarrow_e P'' \implies P \rightsquigarrow_e P'' \rangle$
using τ -trans-mono-AfterExt τ -trans-transitivity τ -trans-anti-mono-ready-set **by** *blast*

We can now define the concept of transition with a trace and demonstrate the first properties.

inductive *trace-trans* :: $\langle 'a \text{ process} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$
 $(\langle - / \rightsquigarrow^* - / \rightarrow [50, 3, 51] 50 \rangle)$

where *trace- τ -trans* : $\langle P \rightsquigarrow_\tau P' \implies P \rightsquigarrow^* \sqcap P' \rangle$
| *trace-tick-trans* : $\langle P \rightsquigarrow_\tau P' \implies P \rightsquigarrow^* [\checkmark] P' \rangle$
| *trace-Cons-ev-trans* :
 $\langle P \rightsquigarrow_\tau (ev \ e) \ P' \implies P' \rightsquigarrow^* s \ P'' \implies P \rightsquigarrow^* (ev \ e) \ \# \ s \ P'' \rangle$

lemma *trace-trans- τ -trans*: $\langle P \rightsquigarrow^* s \ P' \implies P' \rightsquigarrow_\tau P'' \implies P \rightsquigarrow^* s \ P'' \rangle$
apply (*induct rule: trace-trans.induct*)
subgoal using τ -trans-transitivity *trace- τ -trans* **by** *blast*
subgoal using *event-trans- τ -trans* *trace-tick-trans* **by** *blast*
using *trace-Cons-ev-trans* **by** *blast*

lemma τ -trans-trace-trans: $\langle P \rightsquigarrow_\tau P' \implies P' \rightsquigarrow^* s \ P'' \implies P \rightsquigarrow^* s \ P'' \rangle$
apply (*rotate-tac, induct rule: trace-trans.induct*)
subgoal using τ -trans-transitivity *trace- τ -trans* **by** *blast*
subgoal using τ -trans-event-trans *trace-tick-trans* **by** *blast*
using τ -trans-event-trans *trace-Cons-ev-trans* **by** *blast*

lemma *BOT-trace-trans-anything* : $\langle \text{front-tickFree } s \implies \perp \rightsquigarrow^* s \ P \rangle$
proof (*induct s arbitrary: P*)
show $\langle \bigwedge P. \text{front-tickFree } \sqcap \implies \perp \rightsquigarrow^* \sqcap P \rangle$
by (*simp add: BOT- τ -trans-anything trace- τ -trans*)


```

next
  fix e s P
  assume prem: ⟨front-tickFree (e # s)⟩ and hyp: ⟨front-tickFree s ⟹ ⊥ ∼*s P⟩
for P
  have * : ⟨front-tickFree s⟩
  by (metis prem butlast.simps(2) front-tickFree-butlast front-tickFree-def tick-
Free-Cons)
  show ⟨⊥ ∼*e # s P⟩
  proof (cases e)
  fix e'
  assume ⟨e = ev e'⟩
  show ⟨⊥ ∼*e # s P⟩
  by (simp add: ⟨e = ev e'⟩
(rule trace-Cons-ev-trans[OF - hyp];
simp add: * AfterExt-BOT BOT-τ-trans-anything ready-set-BOT))
next
  have ⟨e = ✓ ⟹ s = []⟩
  by (metis prem butlast.simps(2) front-tickFree-butlast tickFree-Cons)
  thus ⟨e = ✓ ⟹ ⊥ ∼*e # s P⟩
  by (simp add: AfterExt-BOT BOT-τ-trans-anything ready-set-BOT trace-tick-trans)
qed
qed

```

7.2 Consequences of $P \rightsquigarrow^* s Q$ on \mathcal{F} , \mathcal{T} and \mathcal{D}

lemma *trace-trans-imp-F-if-τ-trans-imp-leF*:
 $\langle P \rightsquigarrow^* s Q \implies X \in \mathcal{R} Q \implies (s, X) \in \mathcal{F} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_F Q \rangle$
proof (*induct rule: trace-trans.induct*)
show $\langle P \rightsquigarrow_\tau Q \implies X \in \mathcal{R} Q \implies ([], X) \in \mathcal{F} P \rangle$ **for** $P Q$
by (*meson failure-refine-def in-mono Refusals-iff that*)
next
show $\langle P \rightsquigarrow_\checkmark Q \implies X \in \mathcal{R} Q \implies ([\checkmark], X) \in \mathcal{F} P \rangle$ **for** $P Q$
by (*metis append-Nil mem-Collect-eq ready-set-def tick-T-F*)
next
fix $P e Q s Q'$
assume * : $\langle P \rightsquigarrow (ev e) Q \rangle \langle X \in \mathcal{R} Q' \implies (s, X) \in \mathcal{F} Q \rangle \langle X \in \mathcal{R} Q' \rangle$
have $\langle P \text{ afterExt } ev e \sqsubseteq_F Q \rangle$ **using** *(1) **that by** *blast*
hence $\langle (s, X) \in \mathcal{F} (P \text{ afterExt } ev e) \rangle$ **by** (*simp add: failure-refine-def subsetD*
*(2, 3))
thus $\langle (ev e \# s, X) \in \mathcal{F} P \rangle$ **by** (*simp add: F-AfterExt *(1)*) (*metis list.exhaust-sel*)
qed

lemma *trace-trans-imp-T-if-τ-trans-imp-leT*: $\langle P \rightsquigarrow^* s Q \implies s \in \mathcal{T} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_T Q \rangle$
proof (*induct rule: trace-trans.induct*)
show $\langle P \rightsquigarrow_\tau Q \implies [] \in \mathcal{T} P \rangle$ **for** $P Q$
by (*simp add: Nil-lem-T*)

next
show $\langle P \rightsquigarrow \checkmark Q \implies [\checkmark] \in \mathcal{T} P \rangle$ **for** $P Q$
by (*simp add: ready-set-def*)
next
fix $P e Q s Q'$
assume $*$: $\langle P \rightsquigarrow (ev e) Q \rangle \langle s \in \mathcal{T} Q \rangle$
have $\langle P \text{ afterExt } ev e \sqsubseteq_T Q \rangle$ **using** $*(1)$ **that by** *blast*
hence $\langle s \in \mathcal{T} (P \text{ afterExt } ev e) \rangle$ **by** (*simp add: *(2) subsetD trace-refine-def*)
with $*(1)$ *list.collapse* **show** $\langle ev e \# s \in \mathcal{T} P \rangle$
by (*force simp add: T-AfterExt ready-set-def*)
qed

lemma *trace-trans-BOT-imp-D-if- τ -trans-imp-leD*: $\langle P \rightsquigarrow^* s \perp \implies s \in \mathcal{D} P \rangle$

if $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_D Q \rangle$

proof (*induct s arbitrary: P*)

show $\langle \bigwedge P. P \rightsquigarrow^* [] \perp \implies [] \in \mathcal{D} P \rangle$

by (*subst (asm) trace-trans.simps, auto*)

(*meson BOT-iff-D divergence-refine-def subsetD that*)

next

fix $e s P$

assume *prem* : $\langle P \rightsquigarrow^* e \# s \perp \rangle$

assume *hyp*: $\langle P \rightsquigarrow^* s \perp \implies s \in \mathcal{D} P \rangle$ **for** P

have $\langle P \text{ afterExt } e \rightsquigarrow^* s \perp \rangle$

using *prem* **by** (*cases rule: trace-trans.cases*)

(*auto simp add: trace- τ -trans intro: τ -trans-trace-trans*)

from *hyp*[*OF this*] **show** $\langle e \# s \in \mathcal{D} P \rangle$

by (*fastforce intro: prem trace-trans.cases simp add: D-AfterExt D-UU split:*

if-split-asm)

qed

7.3 Characterizations for $P \rightsquigarrow^* s Q$

lemma *trace-trans-iff* :

$\langle P \rightsquigarrow^* [] Q \longleftrightarrow P \rightsquigarrow_\tau Q \rangle$

$\langle P \rightsquigarrow^* [\checkmark] Q \longleftrightarrow P \rightsquigarrow \checkmark Q \rangle$

$\langle P \rightsquigarrow^* (ev e) \# s Q' \longleftrightarrow (\exists Q. P \rightsquigarrow (ev e) Q \wedge Q \rightsquigarrow^* s Q') \rangle$

$\langle \text{tickFree } s \implies (P \rightsquigarrow^* s @ [f] Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow f Q') \rangle$

$\langle \text{front-tickFree } (s @ t) \implies (P \rightsquigarrow^* s @ t Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t Q') \rangle$

proof –

show $f1$: $\langle \bigwedge P Q. P \rightsquigarrow^* [] Q \longleftrightarrow P \rightsquigarrow_\tau Q \rangle$

and $f2$: $\langle \bigwedge P Q. P \rightsquigarrow^* [\checkmark] Q \longleftrightarrow P \rightsquigarrow \checkmark Q \rangle$

and $f3$: $\langle \bigwedge P Q' e. P \rightsquigarrow^* (ev e) \# s Q' \longleftrightarrow (\exists Q. P \rightsquigarrow (ev e) Q \wedge Q \rightsquigarrow^* s Q') \rangle$

by (*((subst trace-trans.simps, auto)[1])+*)

show $f4$: $\langle \text{tickFree } s \implies (P \rightsquigarrow^* s @ [f] Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow f Q') \rangle$

for $s f P Q'$

proof *safe*

show $\langle P \rightsquigarrow^* s @ [f] Q' \implies \exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow f Q' \rangle$

```

proof (induct s arbitrary: P)
  case Nil
  thus ?case
    apply (subst (asm) trace-trans.simps, simp)
    using  $\tau$ -trans-eq  $\tau$ -trans-transitivity f1 by blast
next
  case (Cons e s)
  from Cons.prem1 have * :  $\langle e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow^* s @ [f] Q' \rangle$ 
    by (subst (asm) trace-trans.simps)
    (auto simp add: intro:  $\tau$ -trans-trace-trans)
  with Cons.hyps obtain Q where ** :  $\langle P \text{ afterExt } e \rightsquigarrow^* s Q \rangle \langle Q \rightsquigarrow f Q' \rangle$  by
blast
  have  $\langle P \rightsquigarrow^* e \# s Q \rangle$ 
  proof (cases e)
    fix e'
    assume  $\langle e = \text{ev } e' \rangle$ 
    thus  $\langle P \rightsquigarrow^* e \# s Q \rangle$ 
    apply simp
    by (rule trace-Cons-ev-trans[OF - ** (1)]) (use *  $\tau$ -trans-eq in blast)
  next
  from Cons.prem2 have  $\langle e = \checkmark \implies s = [] \rangle$  by (subst (asm) trace-trans.simps)
auto
  thus  $\langle e = \checkmark \implies P \rightsquigarrow^* e \# s Q \rangle$  using * ** (1) f1 f2 by blast
  qed
  with ** (2) show  $\langle \exists Q. P \rightsquigarrow^* e \# s Q \wedge Q \rightsquigarrow f Q' \rangle$  by blast
  qed
next
  show  $\langle \text{tickFree } s \implies P \rightsquigarrow^* s Q \implies f \in \text{ready-set } Q \implies Q \text{ afterExt } f \rightsquigarrow_\tau Q' \rangle$ 
 $\implies$ 
   $P \rightsquigarrow^* s @ [f] Q'$  for Q
  proof (induct s arbitrary: P Q)
    show  $\langle \text{tickFree } [] \implies P \rightsquigarrow^* [] Q \implies f \in \text{ready-set } Q \implies Q \text{ afterExt } f \rightsquigarrow_\tau Q' \rangle$ 
 $\implies$ 
 $P \rightsquigarrow^* [] @ [f] Q'$  for P Q
    by (simp add: f1)
    (metis (full-types)  $\tau$ -trans-eq  $\tau$ -trans-event-trans event.exhaust
trace-Cons-ev-trans trace- $\tau$ -trans trace-tick-trans)
  next
  case (Cons e s)
  from Cons.prem2 (2) have * :  $\langle e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow^* s Q \rangle$ 
    by (subst (asm) trace-trans.simps)
    (auto simp add: f1 intro:  $\tau$ -trans-trace-trans)
  show ?case
  proof (cases e)
    fix e'
    assume  $\langle e = \text{ev } e' \rangle$ 
    thus  $\langle P \rightsquigarrow^* (e \# s) @ [f] Q' \rangle$ 
    apply simp
    by (rule trace-Cons-ev-trans

```

```

      [OF - Cons.hyps[OF tickFree-tl[OF Cons.prem(1), simplified]
        * [THEN conjunct2] Cons.prem(3)]]
      (use *  $\tau$ -trans-eq Cons.prem(4) in <blast+>)
    next
      show <e = ✓  $\implies$  P  $\rightsquigarrow^*$ (e # s) @ [f] Q'> using Cons.prem(1) by auto
    qed
  qed
qed

show <front-tickFree (s @ t)  $\implies$  P  $\rightsquigarrow^*$ s @ t Q'  $\longleftrightarrow$  ( $\exists$  Q. P  $\rightsquigarrow^*$ s Q  $\wedge$  Q  $\rightsquigarrow^*$ t
Q')>
proof (induct t arbitrary: Q' rule: rev-induct)
  show <P  $\rightsquigarrow^*$ s @ [] Q'  $\longleftrightarrow$  ( $\exists$  Q. P  $\rightsquigarrow^*$ s Q  $\wedge$  Q  $\rightsquigarrow^*$ [] Q')> for Q'
    by (metis  $\tau$ -trans-eq append.right-neutral trace-trans- $\tau$ -trans f1)
  next
    case (snoc e t)
    show <P  $\rightsquigarrow^*$ s @ t @ [e] Q'  $\longleftrightarrow$  ( $\exists$  Q. P  $\rightsquigarrow^*$ s Q  $\wedge$  Q  $\rightsquigarrow^*$ t @ [e] Q')>
    proof (intro iffI)
      assume assm : <P  $\rightsquigarrow^*$ s @ t @ [e] Q'>
      from assm obtain Q where <P  $\rightsquigarrow^*$ s @ t Q> <Q  $\rightsquigarrow^*$ e Q'>
      by (metis append.assoc f4 front-tickFree-implies-tickFree snoc.prem)
      also obtain R where **: <P  $\rightsquigarrow^*$ s R> <R  $\rightsquigarrow^*$ t Q>
      by (metis calculation(1) append.assoc front-tickFree-dw-closed snoc.hyps
snoc.prem)
      ultimately show < $\exists$  Q. P  $\rightsquigarrow^*$ s Q  $\wedge$  Q  $\rightsquigarrow^*$ t @ [e] Q'>
      by (metis append-is-Nil-conv f4 front-tickFree-mono list.distinct(1) snoc.prem)
    next
      assume < $\exists$  Q. P  $\rightsquigarrow^*$ s Q  $\wedge$  Q  $\rightsquigarrow^*$ t @ [e] Q'>
      then obtain Q where <P  $\rightsquigarrow^*$ s Q> <Q  $\rightsquigarrow^*$ t @ [e] Q'> by blast
      also obtain R where <Q  $\rightsquigarrow^*$ t R> <R  $\rightsquigarrow^*$ e Q'>
      by (metis append-is-Nil-conv calculation(2) f4
front-tickFree-mono list.distinct(1) snoc.prem)
      ultimately show <P  $\rightsquigarrow^*$ s @ t @ [e] Q'>
      by (metis append-assoc f4 front-tickFree-implies-tickFree snoc.hyps
snoc.prem tickFree-implies-front-tickFree)
    qed
  qed
qed
qed

```

7.4 Finally: $P \rightsquigarrow^* s Q$ is P afterTrace $s \rightsquigarrow_{\tau} Q$

```

theorem T-imp-trace-trans-iff-AfterTrace- $\tau$ -trans :
  <s  $\in$   $\mathcal{T}$  P  $\implies$  (P  $\rightsquigarrow^*$ s Q)  $\longleftrightarrow$  P afterTrace s  $\rightsquigarrow_{\tau}$  Q>
proof (intro iffI)
  show <P  $\rightsquigarrow^*$ s Q  $\implies$  s  $\in$   $\mathcal{T}$  P  $\implies$  P afterTrace s  $\rightsquigarrow_{\tau}$  Q>
  proof (induct s arbitrary: P Q rule: rev-induct)
    show < $\bigwedge$  P Q. P  $\rightsquigarrow^*$ [] Q  $\implies$  []  $\in$   $\mathcal{T}$  P  $\implies$  P afterTrace []  $\rightsquigarrow_{\tau}$  Q>
      using trace-trans.cases by auto
  next

```

```

fix s e P Q
assume hyp : ⟨P  $\rightsquigarrow^*$ s Q  $\implies$  s  $\in$   $\mathcal{T}$  P  $\implies$  P afterTrace s  $\rightsquigarrow_\tau$  Q⟩ for P Q
assume prems : ⟨P  $\rightsquigarrow^*$ s @ [e] Q⟩ ⟨s @ [e]  $\in$   $\mathcal{T}$  P⟩
have * : ⟨e  $\in$  ready-set (P afterTrace s)⟩
  using prems(2) ready-set-AfterTrace by blast
show ⟨P afterTrace (s @ [e])  $\rightsquigarrow_\tau$  Q⟩
  by (metis AfterTrace-snoc  $\tau$ -trans-event-trans append-single-T-imp-tickFree
    hyp is-processT3-ST prems trace-trans-iff(4))
qed
next
show ⟨P afterTrace s  $\rightsquigarrow_\tau$  Q  $\implies$  s  $\in$   $\mathcal{T}$  P  $\implies$  P  $\rightsquigarrow^*$ s Q⟩
proof (induct arbitrary: P Q rule: AfterTrace.induct)
  show ⟨ $\bigwedge$ P Q. P afterTrace []  $\rightsquigarrow_\tau$  Q  $\implies$  []  $\in$   $\mathcal{T}$  P  $\implies$  P  $\rightsquigarrow^*$ [] Q⟩
    by (simp add: trace- $\tau$ -trans)
next
fix e and s :: ⟨ $\alpha$  trace⟩ and Q P
assume hyp : ⟨P afterTrace s  $\rightsquigarrow_\tau$  Q  $\implies$  s  $\in$   $\mathcal{T}$  P  $\implies$  P  $\rightsquigarrow^*$ s Q⟩ for P Q
assume prems : ⟨P afterTrace (e # s)  $\rightsquigarrow_\tau$  Q⟩ ⟨e # s  $\in$   $\mathcal{T}$  P⟩
have * : ⟨e  $\in$  ready-set P  $\wedge$  s  $\in$   $\mathcal{T}$  (P afterExt e)⟩
  by (metis AfterTrace.simps(1, 2) Cons-in-T-imp-elem-ready-set
    T-AfterTrace append-Cons append-Nil prems(2))
show ⟨P  $\rightsquigarrow^*$ e # s Q⟩
proof (cases e)
  fix e'
  assume ** : ⟨e = ev e'⟩
  from ** have ⟨P  $\rightsquigarrow$ (ev e') P afterExt (ev e')⟩
    by (simp add:  $\tau$ -trans-eq)
  thus ⟨P  $\rightsquigarrow^*$ e # s Q⟩
    by (subst **, rule trace-Cons-ev-trans[OF - hyp[OF prems(1)[simplified]
      *[THEN conjunct2], simplified **]])
next
have ⟨e =  $\checkmark$   $\implies$  s = []⟩ by (metis butlast.simps(2) front-tickFree-butlast
  is-processT2-TR tickFree-Cons prems(2))
thus ⟨e =  $\checkmark$   $\implies$  P  $\rightsquigarrow^*$ e # s Q⟩
  using * prems(1) trace-tick-trans by force
qed
qed
qed

```

As corollaries we obtain the reciprocal results of

$$\llbracket \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q; ?P \rightsquigarrow^* ?s ?Q; ?X \in \mathcal{R} ?Q \rrbracket \implies (?s, ?X) \in \mathcal{F} ?P$$

$$\llbracket \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q; ?P \rightsquigarrow^* ?s ?Q \rrbracket \implies ?s \in \mathcal{T} ?P$$

$$\llbracket \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_D Q; ?P \rightsquigarrow^* ?s \perp \rrbracket \implies ?s \in \mathcal{D} ?P$$

lemma *F-imp-exists-trace-trans*: ⟨(s, X) \in \mathcal{F} P \implies \exists Q. (P \rightsquigarrow^* s Q) \wedge X \in \mathcal{R} Q⟩
by (meson F-T F-imp-R-AfterTrace T-imp-trace-trans-iff-AfterTrace- τ -trans τ -trans-eq)

lemma *T-imp-exists-trace-trans*: ⟨s \in \mathcal{T} P \implies \exists Q. P \rightsquigarrow^* s Q⟩

using *F-imp-exists-trace-trans T-F* by *blast*

lemma *D-imp-trace-trans-BOT*: $\langle s \in \mathcal{D} P \implies P \rightsquigarrow^* s \perp \rangle$
 by (*subst T-imp-trace-trans-iff-AfterTrace- τ -trans*, *simp add: D-T*)
 (*metis BOT-iff-D D-AfterTrace τ -trans-eq self-append-conv*)

When we have more information on $P \rightsquigarrow_\tau Q$, we obtain:

lemma *τ -trans-imp-leT-imp-STOP-trace-trans-iff*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \implies STOP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = STOP \rangle$
 using *STOP-T-iff* by (*subst trace-trans.simps*)
 (*auto simp add: ready-set-STOP τ -trans-eq*)

lemma *τ -trans-imp-leF-imp-SKIP-trace-trans-iff*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q \implies$
 $SKIP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle$
 using *SKIP-F-iff STOP-F-iff* by (*subst trace-trans.simps*)
 (*auto simp add: AfterExt-SKIP ready-set-SKIP τ -trans-eq*)

lemma *τ -trans-imp-leT-imp-trace-trans-ready-set-subset-ready-set-AfterTrace*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \implies P \rightsquigarrow^* s Q \implies$
 $ready-set Q \subseteq ready-set (P \text{ afterTrace } s) \rangle$
 using *T-imp-trace-trans-iff-AfterTrace- τ -trans*
anti-mono-ready-set-T trace-trans-imp-T-if- τ -trans-imp-leT by *blast*

lemma *τ -trans-imp-leT-imp-trace-trans-imp-ready-set*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \implies P \rightsquigarrow^*(s @ e \# t) Q \implies$
 $e \in ready-set (P \text{ afterTrace } s) \rangle$
 using *ready-set-AfterTrace trace-trans-imp-T-if- τ -trans-imp-leT* by *blast*

lemma *trace-trans-iff-T-and-AfterTrace- τ -trans-if- τ -trans-imp-leT*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \implies$
 $(P \rightsquigarrow^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \rightsquigarrow_\tau Q \rangle$
 using *T-imp-trace-trans-iff-AfterTrace- τ -trans trace-trans-imp-T-if- τ -trans-imp-leT*
 by *blast*

7.5 General Rules of Operational Semantics

Some rules of operational semantics are consequences of *OpSemGeneric*'s axioms without needing to specify more what $P \rightsquigarrow_\tau Q$ is.

lemma *SKIP-trans-tick*: $\langle SKIP \rightsquigarrow \checkmark STOP \rangle$
 by (*simp add: AfterExt-SKIP τ -trans-eq ready-set-SKIP*)

lemma *tick-trans-imp-BOT-L-or-STOP-R*: $\langle P \rightsquigarrow \checkmark Q \implies P = \perp \vee Q = STOP \rangle$

by (*metis* τ -trans-anti-mono-ready-set ready-set-AfterExt ready-set-empty-iff-STOP subset-empty)

lemma *STOP-trace-trans-iff* : $\langle STOP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = STOP \rangle$

by (*metis* AfterExt-SKIP SKIP-neq-BOT SKIP-trans-tick empty-iff trace-trans.cases ready-set-STOP tick-trans-imp-BOT-L-or-STOP-R trace-trans-iff(1))

lemma *ready-tick-imp- τ -trans-SKIP*: $\langle P \rightsquigarrow_\tau SKIP \rangle$ **if** $\langle \checkmark \in \text{ready-set } P \rangle$

proof –

from that **have** $\langle P \sqsubseteq_{FD} SKIP \rangle$

unfolding failure-divergence-refine-def le-ref-def

by (*auto simp add: F-SKIP D-SKIP subset-iff ready-set-def is-processT6-S2*)
(*metis* append-Nil tick-T-F)

then obtain Q **where** $\langle P = Q \sqcap SKIP \rangle$

by (*metis* mono-Ndet-FD mono-Ndet-FD-left FD-antisym Ndet-id idem-FD)

thus $\langle P \rightsquigarrow_\tau SKIP \rangle$ **by** (*simp add: τ -trans-NdetR*)

qed

lemma *exists-tick-trans-is-ready-tick*: $\langle (\exists P'. P \rightsquigarrow_{\checkmark} P') \longleftrightarrow \checkmark \in \text{ready-set } P \rangle$

using τ -trans-eq **by** *blast*

lemma *tick-trans-iff* : $\langle P \rightsquigarrow_{\checkmark} P' \longleftrightarrow P = \perp \vee P \rightsquigarrow_\tau SKIP \wedge P' = STOP \rangle$

by (*metis* AfterExt-BOT BOT- τ -trans-anything SKIP-trans-tick τ -trans-event-trans

ready-tick-imp- τ -trans-SKIP tick-trans-imp-BOT-L-or-STOP-R)

lemma *SKIP-cant-ev-trans*: $\langle \neg SKIP \rightsquigarrow (ev e) STOP \rangle$

by (*simp add: ready-set-SKIP*)

lemma *STOP-cant-event-trans*: $\langle \neg STOP \rightsquigarrow e P \rangle$

by (*simp add: ready-set-STOP*)

lemma *ev-trans-Mprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P a \rightsquigarrow (ev e) (P e) \rangle$

by (*simp add: AfterExt-def After-Mprefix τ -trans-eq ready-set-Mprefix*)

lemma *ev-trans-Mndetprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P a \rightsquigarrow (ev e) (P e) \rangle$

by (*simp add: AfterExt-def After-Mndetprefix τ -trans-eq ready-set-Mndetprefix*)

lemma *ev-trans-prefix*: $\langle e \rightarrow P \rightsquigarrow (ev e) P \rangle$

by (*metis* ev-trans-Mprefix insertI1 write0-def)

lemma τ -trans-MultiNdet: $\langle \text{finite } A \implies x \in A \implies (\prod a \in A. P a) \rightsquigarrow_{\tau} P x \rangle$
by (metis MultiNdet-insert' τ -trans-NdetL emptyE insert-absorb)

lemma τ -trans-GlobalNdet: $\langle (\prod a \in A. P a) \rightsquigarrow_{\tau} P e \rangle$ **if** $\langle e \in A \rangle$

proof –

have $\langle (\prod a \in A. P a) = P e \sqcap (\prod a \in A. P a) \rangle$

by (metis GlobalNdet-factorization-union GlobalNdet-unit
empty-iff insertI1 insert-absorb insert-is-Un that)

thus $\langle (\prod a \in A. P a) \rightsquigarrow_{\tau} P e \rangle$ **by** (metis τ -trans-NdetL)

qed

lemma fix-point- τ -trans: $\langle \text{cont } f \implies P = (\mu X. f X) \implies P \rightsquigarrow_{\tau} f P \rangle$
by (metis τ -trans-eq cont-process-rec)

lemma event-trans-DetL: $\langle P \rightsquigarrow_e P' \implies P \sqcap Q \rightsquigarrow_e P' \rangle$

by (metis AfterExt-Det-is-AfterExt-Ndet Un-iff τ -trans-NdetL τ -trans-event-trans
ready-set-Det)

lemma event-trans-DetR: $\langle Q \rightsquigarrow_e Q' \implies P \sqcap Q \rightsquigarrow_e Q' \rangle$

by (metis Det-commute event-trans-DetL)

lemma event-trans-MultiDet:

$\langle \text{finite } A \implies a \in A \implies P a \rightsquigarrow_e Q \implies (\prod a \in A. P a) \rightsquigarrow_e Q \rangle$

by (metis MultiDet-insert' event-trans-DetL insert-absorb)

lemma Sliding-event-transL: $\langle P \rightsquigarrow_e P' \implies P \triangleright Q \rightsquigarrow_e P' \rangle$

unfolding Sliding-def

apply (drule event-trans-DetL[of e P P' Q])

using τ -trans-NdetL τ -trans-event-trans **by** blast

lemma Sliding- τ -transR: $\langle P \triangleright Q \rightsquigarrow_{\tau} Q \rangle$

unfolding Sliding-def **by** (simp add: τ -trans-NdetR)

lemma $\langle \exists P P' Q. P \rightsquigarrow_{\checkmark} P' \wedge \neg P ; Q \rightsquigarrow_{\checkmark} P' ; Q \rangle$

proof (intro exI)

show $\langle \text{SKIP} \rightsquigarrow_{\checkmark} \text{STOP} \wedge \neg \text{SKIP} ; \text{STOP} \rightsquigarrow_{\checkmark} \text{STOP} ; \text{STOP} \rangle$

by (simp add: SKIP-Seq SKIP-trans-tick STOP-cant-event-trans)

qed

lemma *ev-trans-SeqR*:
 $\langle \checkmark \in \text{ready-set } P \implies Q \rightsquigarrow (ev\ e) Q' \implies P ; Q \rightsquigarrow (ev\ e) Q' \rangle$
apply (*simp add: AfterExt-Seq ready-set-Seq AfterExt-BOT BOT-Seq*)
using τ -trans-NdetR τ -trans-transitivity **by** *blast*

lemma $\langle \text{SKIP } \llbracket S \rrbracket \text{ SKIP } \rightsquigarrow \checkmark \text{ STOP} \rangle$
by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma $\langle \text{SKIP } \llbracket S \rrbracket \text{ SKIP } \rightsquigarrow_{\tau} \text{SKIP} \rangle$
by (*simp add: Sync-SKIP-SKIP τ -trans-eq*)

lemma *tick-trans-Hiding*: $\langle P \setminus B \rightsquigarrow \checkmark \text{ STOP} \rangle$ **if** $\langle P \rightsquigarrow \checkmark P' \rangle$
proof –
have $\langle (P \setminus B) \text{ afterExt } \checkmark = \text{STOP} \sqcap (P \setminus B) \text{ afterExt } \checkmark \rangle$
by (*simp add: AfterExt-def Ndet-is-BOT-iff*)
thus $\langle P \setminus B \rightsquigarrow \checkmark \text{ STOP} \rangle$
by (*simp add: AfterExt-def BOT- τ -trans-anything τ -trans-eq
ready-tick-imp-ready-tick-Hiding that*)

qed

The following lemma is useless since the locale mechanism forces f to be of type $'\alpha \Rightarrow '\alpha$ while it could be $'\alpha \Rightarrow '\beta$. We will have to prove it again on each instantiation of the locale.

lemma $\langle \text{Renaming } P\ f \rightsquigarrow \checkmark \text{ STOP} \rangle$ **if** $\langle P \rightsquigarrow \checkmark P' \rangle$
proof –
have $\langle \text{Renaming } P\ f \text{ afterExt } \checkmark = \text{STOP} \sqcap \text{Renaming } P\ f \text{ afterExt } \checkmark \rangle$
by (*metis AfterExt-Sliding-is-AfterExt-Ndet STOP-Sliding*)
thus $\langle \text{Renaming } P\ f \rightsquigarrow \checkmark \text{ STOP} \rangle$
by (*simp add: that tick-eq-EvExt AfterExt-def τ -trans-eq
ready-set-Renaming BOT- τ -trans-anything*)

qed

end

end

Chapter 8

Failure Divergence Operational Semantics

theory *OpSemFD*
 imports *OpSemGeneric HOL-Library.LaTeXsugar*
begin

As announced in the motivations, the first definition we want to try is with (\sqsubseteq_{FD}) .

abbreviation $\tau\text{-trans} :: \langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$ (**infixl** $\langle_{FD} \rightsquigarrow_{\tau} \rangle$ 50)
 where $\langle P \text{ }_{FD} \rightsquigarrow_{\tau} Q \equiv P \sqsubseteq_{FD} Q \rangle$

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGeneric*.

interpretation *OpSemGeneric* $\langle_{(FD} \rightsquigarrow_{\tau})} \rangle$
 using *trans-FD* **by** *unfold-locales*
 (*auto simp add: anti-mono-ready-set-FD mono-AfterExt-FD*)

notation *event-trans* $\langle \langle - \rangle_{FD} \rightsquigarrow - \rangle \rightarrow [50, 3, 51]$ 50)

notation *trace-trans* $\langle \langle - \rangle_{FD} \rightsquigarrow^* - \rangle \rightarrow [50, 3, 51]$ 50)

lemma $\langle P \text{ }_{FD} \rightsquigarrow e P' \Longrightarrow P' \text{ }_{FD} \rightsquigarrow_{\tau} P'' \Longrightarrow P \text{ }_{FD} \rightsquigarrow e P'' \rangle$
 $\langle P \text{ }_{FD} \rightsquigarrow_{\tau} P' \Longrightarrow P' \text{ }_{FD} \rightsquigarrow e P'' \Longrightarrow P \text{ }_{FD} \rightsquigarrow e P'' \rangle$
 by (*fact event-trans- τ -trans τ -trans-event-trans*) $+$

8.1 Operational Semantics Laws

SKIP law

lemma $\langle \text{SKIP} \text{ }_{FD} \rightsquigarrow \checkmark \text{STOP} \rangle$
 by (*fact SKIP-trans-tick*)

$e \rightarrow P$ laws

lemma $\langle e \in A \Longrightarrow \Box a \in A \rightarrow P a \text{ }_{FD} \rightsquigarrow (ev e) (P e) \rangle$

by (*fact ev-trans-Mprefix*)

lemma $\langle e \in A \implies \sqcap a \in A \rightarrow P a \text{ }_{FD \rightsquigarrow} (ev\ e) (P\ e) \rangle$
by (*fact ev-trans-Mndetprefix*)

lemma $\langle e \rightarrow P \text{ }_{FD \rightsquigarrow} (ev\ e) P \rangle$
by (*fact ev-trans-prefix*)

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow} P \rangle$
and $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow} Q \rangle$
by (*fact τ -trans-NdetL τ -trans-NdetR*)₊

lemma $\langle a \in A \implies (\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} P a \rangle$
by (*fact τ -trans-GlobalNdet*)

lemma $\langle \text{finite } A \implies a \in A \implies (\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} P a \rangle$
by (*fact τ -trans-MultiNdet*)

$\mu x. f x$ law

lemma $\langle \text{cont } f \implies P = (\mu X. f X) \implies P \text{ }_{FD \rightsquigarrow} f P \rangle$
by (*fact fix-point- τ -trans*)

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \text{ }_{FD \rightsquigarrow} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \text{ }_{FD \rightsquigarrow} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P \sqcap Q' \rangle$
by *simp-all*

lemma τ -trans-MultiDet:
 $\langle \text{finite } A \implies \forall a \in A. P a \text{ }_{FD \rightsquigarrow} P' a \implies$
 $(\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} (\sqcap a \in A. P' a) \rangle$
by (*fact mono-MultiDet-FD*)

lemma $\langle P \text{ }_{FD \rightsquigarrow} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P' \sqcap Q \rangle$
and $\langle Q \text{ }_{FD \rightsquigarrow} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P \sqcap Q' \rangle$
by (*fact event-trans-DetL event-trans-DetR*)₊

lemma $\langle \text{finite } A \implies a \in A \implies P a \text{ }_{FD \rightsquigarrow} Q \implies (\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} Q \rangle$
by (*fact event-trans-MultiDet*)

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{FD \rightsquigarrow} P' \implies P ; Q \text{ }_{FD \rightsquigarrow} P' ; Q \rangle$
by *simp*

lemma *ev-trans-SeqL*: $\langle P \text{ }_{FD \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{FD \rightsquigarrow} (ev\ e) P' ; Q \rangle$
by (*auto simp add: ready-set-Seq AfterExt-Seq*)

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ FD} \rightsquigarrow \checkmark P' \implies P ; Q \text{ FD} \rightsquigarrow_{\tau} Q \rangle$
by (*metis mono-Seq-FD SKIP-Seq τ -trans-eq ready-tick-imp- τ -trans-SKIP*)

lemma $\checkmark \in \text{ready-set } P \implies Q \text{ FD} \rightsquigarrow (ev \ e) \ Q' \implies P ; Q \text{ FD} \rightsquigarrow (ev \ e) \ Q'$

by (*fact ev-trans-SeqR*)

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ FD} \rightsquigarrow_{\tau} P' \implies P \setminus B \text{ FD} \rightsquigarrow_{\tau} P' \setminus B \rangle$
by (*fact mono-Hiding-FD*)

lemma *ev-trans-Hiding-notin*:

$\langle P \text{ FD} \rightsquigarrow (ev \ e) \ P' \implies e \notin B \implies P \setminus B \text{ FD} \rightsquigarrow (ev \ e) \ P' \setminus B \rangle$

by (*metis AfterExt-def After-Hiding-FD-Hiding-After-if-ready-notin mono-Hiding-FD*

event-trans- τ -trans event.simps(4) ready-notin-imp-ready-Hiding)

lemma $\langle P \text{ FD} \rightsquigarrow \checkmark P' \implies P \setminus B \text{ FD} \rightsquigarrow \checkmark STOP \rangle$

by (*fact tick-trans-Hiding*)

lemma *ev-trans-Hiding-inside*:

$\langle P \text{ FD} \rightsquigarrow (ev \ e) \ P' \implies e \in B \implies P \setminus B \text{ FD} \rightsquigarrow_{\tau} P' \setminus B \rangle$

by (*metis AfterExt-def Hiding-FD-Hiding-After-if-ready-inside mono-Hiding-FD event.simps(4) trans-FD*)

Renaming P f laws

lemma τ -trans-Renaming:

$\langle P \text{ FD} \rightsquigarrow_{\tau} P' \implies \text{Renaming } P \ f \ \text{FD} \rightsquigarrow_{\tau} \text{Renaming } P' \ f \rangle$

by (*fact mono-Renaming-FD*)

lemma *tick-trans-Renaming*: $\langle P \text{ FD} \rightsquigarrow \checkmark P' \implies \text{Renaming } P \ f \ \text{FD} \rightsquigarrow \checkmark STOP \rangle$

by (*simp add: AfterExt-def ready-set-Renaming tick-eq-EvExt*)

lemma *ev-trans-Renaming*:

$\langle f \ a = b \implies P \text{ FD} \rightsquigarrow (ev \ a) \ P' \implies \text{Renaming } P \ f \ \text{FD} \rightsquigarrow (ev \ b) \ (\text{Renaming } P' \ f) \rangle$

apply (*simp add: AfterExt-Renaming Renaming-BOT ready-set-BOT ready-set-Renaming*)

apply (*intro conjI impI*)

apply (*meson ev-elem-anteced1 imageI vimageI2*)

apply (*rule τ -trans-transitivity[of - $\langle \text{Renaming } (P \ \text{afterExt } ev \ a) \ f \rangle]$*)

apply (*solves $\langle \text{rule } \tau\text{-trans-GlobalNdet, simp} \rangle$*)

by (*simp add: τ -trans-Renaming*)

lemma $\langle P \text{ FD} \rightsquigarrow_{\tau} P' \implies P \llbracket a := b \rrbracket \text{ FD} \rightsquigarrow_{\tau} P' \llbracket a := b \rrbracket \rangle$

by (*fact τ -trans-Renaming*)

lemma $\langle P \xrightarrow{FD} \checkmark P' \implies P \llbracket a := b \rrbracket \xrightarrow{FD} \checkmark STOP \rangle$
by (*fact tick-trans-Renaming*)

lemma *ev-trans-RenamingF*:
 $\langle P \xrightarrow{FD} (ev\ a) P' \implies P \llbracket a := b \rrbracket \xrightarrow{FD} (ev\ b) P' \llbracket a := b \rrbracket \rangle$
by (*metis ev-trans-Renaming fun-upd-same*)

$P \llbracket S \rrbracket Q$ laws

lemma τ -*trans-SyncL*: $\langle P \xrightarrow{FD} \tau P' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau P' \llbracket S \rrbracket Q \rangle$
and τ -*trans-SyncR*: $\langle Q \xrightarrow{FD} \tau Q' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau P \llbracket S \rrbracket Q' \rangle$
by *simp-all*

lemma *ev-trans-SyncL*:
 $\langle e \notin S \implies P \xrightarrow{FD} (ev\ e) P' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} (ev\ e) P' \llbracket S \rrbracket Q \rangle$
and *ev-trans-SyncR*:
 $\langle e \notin S \implies Q \xrightarrow{FD} (ev\ e) Q' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} (ev\ e) P \llbracket S \rrbracket Q' \rangle$
by (*simp-all add: AfterExt-Sync ready-set-Sync image-iff*)

lemma *ev-trans-SyncLR*:
 $\langle \llbracket e \in S; P \xrightarrow{FD} (ev\ e) P'; Q \xrightarrow{FD} (ev\ e) Q' \rrbracket \implies P \llbracket S \rrbracket Q \xrightarrow{FD} (ev\ e) P' \llbracket S \rrbracket Q' \rangle$
by (*simp add: AfterExt-Sync ready-set-Sync*)

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

lemma *tick-trans-SyncL*: $\langle P \xrightarrow{FD} \checkmark P' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \xrightarrow{FD} \checkmark Q' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau P \llbracket S \rrbracket SKIP \rangle$
by (*simp-all add: ready-tick-imp- τ -trans-SKIP*)

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{FD} \checkmark STOP \rangle$
by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma τ -*trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{FD} \tau SKIP \rangle$
by (*simp add: Sync-SKIP-SKIP*)

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \xrightarrow{FD} \tau P' \implies P \triangleright Q \xrightarrow{FD} \tau P' \triangleright Q \rangle$
by (*simp add: Sliding-def*)

lemma $\langle P \xrightarrow{FD} e P' \implies P \triangleright Q \xrightarrow{FD} e P' \rangle$
by (*fact Sliding-event-transL*)

lemma $\langle P \triangleright Q \xrightarrow{FD} \tau Q \rangle$
by (*fact Sliding- τ -transR*)

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \xrightarrow{FD} \tau P' \implies P \triangle Q \xrightarrow{FD} \tau P' \triangle Q \rangle$

by (simp add: mono-Interrupt-FD)

lemma *Interrupt- τ -trans-right*: $\langle Q \text{ FD}\rightsquigarrow_{\tau} Q' \implies P \Delta Q \text{ FD}\rightsquigarrow_{\tau} P \Delta Q' \rangle$
 by (simp add: mono-Interrupt-FD)

lemma *Interrupt-ev-trans-left*:
 $\langle P \text{ FD}\rightsquigarrow (ev\ e) P' \implies P \Delta Q \text{ FD}\rightsquigarrow (ev\ e) P' \Delta Q \rangle$
 by (simp add: AfterExt-def After-Interrupt Interrupt- τ -trans-left ready-set-Interrupt)

lemma *Interrupt-ev-trans-right*: $\langle Q \text{ FD}\rightsquigarrow (ev\ e) Q' \implies P \Delta Q \text{ FD}\rightsquigarrow (ev\ e) Q' \rangle$
 by (simp add: AfterExt-def After-Interrupt ready-set-Interrupt)

Throw P A Q laws

lemma *Throw- τ -trans-left*:
 $\langle P \text{ FD}\rightsquigarrow_{\tau} P' \implies P \Theta a \in A. Q\ a \text{ FD}\rightsquigarrow_{\tau} P' \Theta a \in A. Q\ a \rangle$
 by (simp add: mono-Throw-FD)

lemma *Throw- τ -trans-right*:
 $\langle \forall a \in A. Q\ a \text{ FD}\rightsquigarrow_{\tau} Q'\ a \implies P \Theta a \in A. Q\ a \text{ FD}\rightsquigarrow_{\tau} P \Theta a \in A. Q'\ a \rangle$
 by (simp add: mono-Throw-FD)

lemma *Throw-event-trans-left*:
 $\langle P \text{ FD}\rightsquigarrow e P' \implies e \notin ev\ A \implies P \Theta a \in A. Q\ a \text{ FD}\rightsquigarrow e (P' \Theta a \in A. Q\ a) \rangle$
 apply (simp add: AfterExt-Throw ready-set-Throw image-iff split: event.split)
 apply (intro conjI impI)
 by (metis AfterExt-def Throw- τ -trans-left event.simps(4))
 (solves $\langle \text{simp add: Throw-STOP tick-trans-iff} \rangle$)

lemma *Throw-ev-trans-right*:
 $\langle P \text{ FD}\rightsquigarrow (ev\ e) P' \implies e \in A \implies P \Theta a \in A. Q\ a \text{ FD}\rightsquigarrow (ev\ e) (Q\ e) \rangle$
 by (simp add: AfterExt-Throw ready-set-Throw split: event.split)

lemma $\langle \text{front-tickFree } s \implies \perp \text{ FD}\rightsquigarrow^* s P \rangle$
 by (fact BOT-trace-trans-anything)

8.2 Reality Checks

lemma $\langle \text{STOP } \text{FD}\rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = \text{STOP} \rangle$
 by (fact STOP-trace-trans-iff)

lemma *SKIP-trace-trans-iff* :
 $\langle \text{SKIP } \text{FD}\rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = \text{SKIP} \vee s = [\checkmark] \wedge P = \text{STOP} \rangle$
 by (simp add: τ -trans-imp-leF-imp-SKIP-trace-trans-iff leFD-imp-leF)

lemma *F-iff-exists-trans* :

$$\langle (s, X) \in \mathcal{F} P \longleftrightarrow (\exists P'. (P \text{ FD}\rightsquigarrow^* s P') \wedge X \in \mathcal{R} P') \rangle$$

using *F-imp-exists-trace-trans leFD-imp-leF trace-trans-imp-F-if- τ -trans-imp-leF*
by *blast*

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \text{ FD}\rightsquigarrow^* s P') \rangle$

by (*meson T-imp-exists-trace-trans leFD-imp-leF leF-imp-leT trace-trans-imp-T-if- τ -trans-imp-leT*)

lemma *D-iff-trace-trans-BOT*: $\langle s \in \mathcal{D} P \longleftrightarrow P \text{ FD}\rightsquigarrow^* s \perp \rangle$

using *D-imp-trace-trans-BOT leFD-imp-leD trace-trans-BOT-imp-D-if- τ -trans-imp-leD*
by *blast*

8.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:

$$\langle P \text{ FD}\rightsquigarrow^* s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$$

by (*metis T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace- τ -trans τ -trans-anti-mono-ready-set*)

lemma *trace-trans-imp-ready-set*:

$$\langle P \text{ FD}\rightsquigarrow^* (s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$$

using *T-iff-exists-trans ready-set-AfterTrace* **by** *blast*

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT* :

$$\langle (P \text{ FD}\rightsquigarrow^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \text{ FD}\rightsquigarrow_{\tau} Q \rangle$$

using *T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace- τ -trans* **by** *blast*

lemma $\langle \text{deadlock-free } P \longleftrightarrow DF \text{ UNIV } \text{ FD}\rightsquigarrow_{\tau} P \rangle$

by (*simp add: deadlock-free-def*)

lemma $\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow DF_{SKIP} \text{ UNIV } \text{ FD}\rightsquigarrow_{\tau} P \rangle$

by (*fact deadlock-free_{SKIP}-FD*)

8.4 Summary: Operational Rules

In this section, we will just write down the operational laws that we have proven in a fancy way.

Absorbance rules

$$\frac{\frac{?P \text{ FD}\rightsquigarrow ?e ?P' \quad ?P' \text{ FD}\rightsquigarrow_{\tau} ?P''}{?P \text{ FD}\rightsquigarrow ?e ?P''}}{?P \text{ FD}\rightsquigarrow_{\tau} ?P' \quad ?P' \text{ FD}\rightsquigarrow ?e ?P''}{?P \text{ FD}\rightsquigarrow ?e ?P''}$$

SKIP rule

$$\overline{SKIP \text{ }_{FD \rightsquigarrow} \checkmark \text{ } STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ }_a \text{ }_{FD \rightsquigarrow} ev \text{ } ?e \text{ } ?P \text{ } ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ }_a \text{ }_{FD \rightsquigarrow} ev \text{ } ?e \text{ } ?P \text{ } ?e}$$

$$\overline{?e \rightarrow ?P \text{ }_{FD \rightsquigarrow} ev \text{ } ?e \text{ } ?P}$$

(\sqcap) rules

$$\frac{\overline{?P \sqcap ?Q \text{ }_{FD \rightsquigarrow \tau} ?P} \quad \overline{?P \sqcap ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}}{?e \in ?A}$$

$$\frac{}{\square a \in ?A. ?P \text{ }_a \text{ }_{FD \rightsquigarrow \tau} ?P \text{ } ?e}$$

$\mu x. f x$ rule

$$\frac{cont \text{ } ?f \quad ?P = (\mu x. ?f x)}{?P \text{ }_{FD \rightsquigarrow \tau} ?f \text{ } ?P}$$

(\square) rules

$$\frac{\overline{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}}{?P \sqcap ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \sqcap ?Q} \quad \frac{\overline{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}}{?P \sqcap ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \sqcap ?Q'}$$

$$\frac{\overline{?P \text{ }_{FD \rightsquigarrow} ?e \text{ } ?P'}}{?P \sqcap ?Q \text{ }_{FD \rightsquigarrow} ?e \text{ } ?P'} \quad \frac{\overline{?Q \text{ }_{FD \rightsquigarrow} ?e \text{ } ?Q'}}{?P \sqcap ?Q \text{ }_{FD \rightsquigarrow} ?e \text{ } ?Q'}$$

(;) rules

$$\frac{\overline{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}}{?P ; ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' ; ?Q} \quad \frac{\overline{?P \text{ }_{FD \rightsquigarrow} ev \text{ } ?e \text{ } ?P'}}{?P ; ?Q \text{ }_{FD \rightsquigarrow} ev \text{ } ?e \text{ } ?P' ; ?Q}$$

$$\frac{\exists P'. ?P \text{ }_{FD \rightsquigarrow} \checkmark \text{ } P'}{?P ; ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}$$

Hiding rules

$$\frac{\frac{\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{?P \setminus ?B \text{ FD}\rightsquigarrow_{\tau} ?P' \setminus ?B}}{?P \text{ FD}\rightsquigarrow_{ev} ?e ?P' \quad ?e \notin ?B}}{?P \setminus ?B \text{ FD}\rightsquigarrow_{ev} ?e ?P' \setminus ?B}}{\quad} \quad \frac{\frac{\frac{?P \text{ FD}\rightsquigarrow_{\checkmark} ?P'}{?P \setminus ?B \text{ FD}\rightsquigarrow_{\checkmark} STOP}}{?P \text{ FD}\rightsquigarrow_{ev} ?e ?P' \quad ?e \in ?B}}{?P \setminus ?B \text{ FD}\rightsquigarrow_{\tau} ?P' \setminus ?B}}$$

Renaming rules

$$\frac{\frac{\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{\text{Renaming } ?P \text{ ?f FD}\rightsquigarrow_{\tau} \text{Renaming } ?P' \text{ ?f}}{?P \text{ FD}\rightsquigarrow_{\checkmark} ?P'}}{\text{Renaming } ?P \text{ ?f FD}\rightsquigarrow_{\checkmark} STOP}}{?f ?a = ?b \quad ?P \text{ FD}\rightsquigarrow_{ev} ?a ?P'}}{\text{Renaming } ?P \text{ ?f FD}\rightsquigarrow_{ev} ?b \text{ Renaming } ?P' \text{ ?f}}$$

Sync rules

$$\frac{\frac{\frac{\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} ?P' \llbracket ?S \rrbracket ?Q} \quad \frac{?Q \text{ FD}\rightsquigarrow_{\tau} ?Q'}{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} ?P \llbracket ?S \rrbracket ?Q'}}{?e \notin ?S \quad ?P \text{ FD}\rightsquigarrow_{ev} ?e ?P'}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?P' \llbracket ?S \rrbracket ?Q}{?e \notin ?S \quad ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?Q'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?P \llbracket ?S \rrbracket ?Q'}{?e \in ?S \quad ?P \text{ FD}\rightsquigarrow_{ev} ?e ?P' \quad ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?Q'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?P' \llbracket ?S \rrbracket ?Q'}{?P \text{ FD}\rightsquigarrow_{\checkmark} ?P'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} SKIP \llbracket ?S \rrbracket ?Q}{?Q \text{ FD}\rightsquigarrow_{\checkmark} ?Q'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} ?P \llbracket ?S \rrbracket SKIP}{SKIP \llbracket ?S \rrbracket SKIP \text{ FD}\rightsquigarrow_{\tau} SKIP}}$$

(▷) rules

$$\frac{\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{?P \triangleright ?Q \text{ FD}\rightsquigarrow_{\tau} ?P' \triangleright ?Q}}{\quad} \quad \frac{\frac{?P \text{ FD}\rightsquigarrow_{?e} ?P'}{?P \triangleright ?Q \text{ FD}\rightsquigarrow_{?e} ?P'}}$$

$$\overline{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}$$

(Δ) rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \Delta ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \Delta ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}{?P \Delta ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \Delta ?Q'}}{\frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P'}{?P \Delta ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?P' \Delta ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow ev} ?e ?Q'}{?P \Delta ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?Q'}}$$

Throw rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \Theta a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow \tau} ?P' \Theta a \in ?A. ?Q a} \quad \frac{\forall a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow \tau} ?Q' a}{?P \Theta a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow \tau} ?P \Theta a \in ?A. ?Q' a}}{\frac{?P \text{ }_{FD \rightsquigarrow ?e} ?P' \quad ?e \notin ev \text{ ' } ?A}{?P \Theta a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow ?e} ?P' \Theta a \in ?A. ?Q a} \quad \frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P' \quad ?e \in ?A}{?P \Theta a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow ev} ?e ?Q ?e}}$$

end

Chapter 9

Trace Divergence Operational Semantics

```
theory OpSemDT
  imports OpSemGeneric HOL-Library.LaTeXsugar
begin
```

The definition with (\sqsubseteq_{FD}) motivates us to try with other refinements.

```
abbreviation  $\tau$ -trans :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool' (infixl  $\langle_{DT \rightsquigarrow \tau}$  50)
  where  $\langle P \rangle_{DT \rightsquigarrow \tau} Q \equiv P \sqsubseteq_{DT} Q$ 
```

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGeneric*.

```
interpretation OpSemGeneric  $\langle_{DT \rightsquigarrow \tau}$ 
  using trans-DT by unfold-locales
  (auto simp add: anti-mono-ready-set-DT mono-AfterExt-DT)
```

```
notation event-trans ( $\langle_{DT \rightsquigarrow -} - \rangle \rightarrow$  [50, 3, 51] 50)
```

```
notation trace-trans ( $\langle_{DT \rightsquigarrow * -} - \rangle \rightarrow$  [50, 3, 51] 50)
```

```
lemma  $\langle P \rangle_{DT \rightsquigarrow} e P' \Longrightarrow P' \langle_{DT \rightsquigarrow \tau} P'' \Longrightarrow P \langle_{DT \rightsquigarrow} e P''$ 
   $\langle P \rangle_{DT \rightsquigarrow \tau} P' \Longrightarrow P' \langle_{DT \rightsquigarrow} e P'' \Longrightarrow P \langle_{DT \rightsquigarrow} e P''$ 
  by (fact event-trans- $\tau$ -trans  $\tau$ -trans-event-trans)+
```

9.1 Operational Semantics Laws

SKIP law

```
lemma  $\langle SKIP \rangle_{DT \rightsquigarrow} \checkmark STOP$ 
  by (fact SKIP-trans-tick)
```

$e \rightarrow P$ laws

```
lemma  $\langle e \in A \Longrightarrow \Box a \in A \rightarrow P a \rangle_{DT \rightsquigarrow} (ev e) (P e)$ 
  by (fact ev-trans-Mprefix)
```

lemma $\langle e \in A \implies \prod a \in A \rightarrow P a \text{ }_{DT \rightsquigarrow} (ev\ e) (P\ e) \rangle$
by (*fact ev-trans-Mndetprefix*)

lemma $\langle e \rightarrow P \text{ }_{DT \rightsquigarrow} (ev\ e) P \rangle$
by (*fact ev-trans-prefix*)

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
by (*fact τ -trans-NdetL τ -trans-NdetR*)⁺

lemma $\langle a \in A \implies (\prod a \in A. P a) \text{ }_{DT \rightsquigarrow \tau} P a \rangle$
by (*fact τ -trans-GlobalNdet*)

lemma $\langle \text{finite } A \implies a \in A \implies (\prod a \in A. P a) \text{ }_{DT \rightsquigarrow \tau} P a \rangle$
by (*fact τ -trans-MultiNdet*)

$\mu\ x. f\ x$ law

lemma $\langle \text{cont } f \implies P = (\mu X. f X) \implies P \text{ }_{DT \rightsquigarrow \tau} f P \rangle$
by (*fact fix-point- τ -trans*)

$P \sqcap Q$ laws, more powerful

lemma τ -trans-DetL: $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \rangle$
and τ -trans-DetR: $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
by (*metis Det-STOP τ -trans-eq leDT-STOP mono-Det-DT*)
(*metis Det-STOP Det-commute τ -trans-eq leDT-STOP mono-Det-DT*)

lemma τ -trans-MultiDet: $\langle \text{finite } A \implies a \in A \implies (\prod a \in A. P a) \text{ }_{DT \rightsquigarrow \tau} P a \rangle$
by (*metis MultiDet-insert' τ -trans-DetL insert-absorb*)

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} P' ; Q \rangle$
by *simp*

lemma *ev-trans-SeqL*: $\langle P \text{ }_{DT \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev\ e) P' ; Q \rangle$

by (*auto simp add: ready-set-Seq AfterExt-Seq*)

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ }_{DT \rightsquigarrow} \checkmark P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
by (*metis SKIP-Seq mono-Seq-DT-left ready-tick-imp- τ -trans-SKIP*)

lemma $\langle \checkmark \in \text{ready-set } P \implies Q \text{ }_{DT \rightsquigarrow} (ev\ e) Q' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev\ e) Q' \rangle$

by (*fact ev-trans-SeqR*)

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \setminus B \text{ }_{DT\rightsquigarrow\tau} P' \setminus B \rangle$
by (fact mono-Hiding-DT)

lemma *ev-trans-Hiding-notin*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \implies e \notin B \implies P \setminus B \text{ }_{DT\rightsquigarrow}(ev\ e) P' \setminus B \rangle$
by (meson AfterExt-Hiding-DT-Hiding-AfterExt-if-ready-notin
mono-Hiding-DT ready-notin-imp-ready-Hiding trans-DT)

lemma $\langle P \text{ }_{DT\rightsquigarrow}\checkmark P' \implies P \setminus B \text{ }_{DT\rightsquigarrow}\checkmark STOP \rangle$
by (fact tick-trans-Hiding)

lemma *ev-trans-Hiding-inside*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \implies e \in B \implies P \setminus B \text{ }_{DT\rightsquigarrow\tau} P' \setminus B \rangle$
by (meson Hiding-DT-Hiding-AfterExt-if-ready-inside mono-Hiding-DT trans-DT)

Renaming P f laws

lemma τ -trans-Renaming:
 $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow\tau} \text{Renaming } P'\ f \rangle$
by (fact mono-Renaming-DT)

lemma *tick-trans-Renaming*: $\langle P \text{ }_{DT\rightsquigarrow}\checkmark P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow}\checkmark STOP \rangle$
by (simp add: AfterExt-def ready-set-Renaming tick-eq-EvExt)

lemma *ev-trans-Renaming*:
 $\langle f\ a = b \implies P \text{ }_{DT\rightsquigarrow}(ev\ a) P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow}(ev\ b) (\text{Renaming } P'\ f) \rangle$
apply (simp add: AfterExt-Renaming Renaming-BOT ready-set-BOT ready-set-Renaming)
apply (intro conjI impI)
apply (meson ev-elem-anteced1 imageI vimageI2)
apply (rule τ -trans-transitivity[of - $\langle \text{Renaming } (P\ \text{afterExt}\ ev\ a)\ f \rangle$])
apply (solves $\langle \text{rule } \tau$ -trans-GlobalNdet, simp \rangle)
by (simp add: τ -trans-Renaming)

lemma $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow\tau} P' \llbracket a := b \rrbracket \rangle$
by (fact τ -trans-Renaming)

lemma $\langle P \text{ }_{DT\rightsquigarrow}\checkmark P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow}\checkmark STOP \rangle$
by (fact tick-trans-Renaming)

lemma *ev-trans-RenamingF*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ a) P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow}(ev\ b) P' \llbracket a := b \rrbracket \rangle$
by (metis ev-trans-Renaming fun-upd-same)

P $\llbracket S \rrbracket$ Q laws

lemma τ -trans-SyncL: $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \llbracket S \rrbracket Q \text{ }_{DT\rightsquigarrow\tau} P' \llbracket S \rrbracket Q \rangle$
and τ -trans-SyncR: $\langle Q \text{ }_{DT\rightsquigarrow\tau} Q' \implies P \llbracket S \rrbracket Q \text{ }_{DT\rightsquigarrow\tau} P \llbracket S \rrbracket Q' \rangle$
by simp-all

lemma *ev-trans-SyncL*:

$$\langle e \notin S \implies P \xrightarrow{DT} (ev\ e)\ P' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} (ev\ e)\ P' \llbracket S \rrbracket Q \rangle$$

and *ev-trans-SyncR*:

$$\langle e \notin S \implies Q \xrightarrow{DT} (ev\ e)\ Q' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} (ev\ e)\ P \llbracket S \rrbracket Q' \rangle$$

by (*simp-all add: AfterExt-Sync ready-set-Sync image-iff*)

lemma *ev-trans-SyncLR*:

$$\langle [e \in S; P \xrightarrow{DT} (ev\ e)\ P'; Q \xrightarrow{DT} (ev\ e)\ Q'] \implies$$

$$P \llbracket S \rrbracket Q \xrightarrow{DT} (ev\ e)\ P' \llbracket S \rrbracket Q' \rangle$$

by (*simp add: AfterExt-Sync ready-set-Sync*)

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

lemma *tick-trans-SyncL*: $\langle P \xrightarrow{DT} \checkmark P' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} \tau \text{ SKIP} \llbracket S \rrbracket Q \rangle$

and *tick-trans-SyncR*: $\langle Q \xrightarrow{DT} \checkmark Q' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} \tau P \llbracket S \rrbracket \text{ SKIP} \rangle$

by (*simp-all add: ready-tick-imp- τ -trans-SKIP*)

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle \text{SKIP} \llbracket S \rrbracket \text{ SKIP} \xrightarrow{DT} \checkmark \text{ STOP} \rangle$

by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma τ -*trans-SKIP-Sync-SKIP*: $\langle \text{SKIP} \llbracket S \rrbracket \text{ SKIP} \xrightarrow{DT} \tau \text{ SKIP} \rangle$

by (*simp add: Sync-SKIP-SKIP*)

P \triangleright *Q* laws

lemma *Sliding- τ -trans-left*: $\langle P \xrightarrow{DT} \tau P' \implies P \triangleright Q \xrightarrow{DT} \tau P' \triangleright Q \rangle$

unfolding *Sliding-def* **by** *simp*

lemma $\langle P \xrightarrow{DT} e P' \implies P \triangleright Q \xrightarrow{DT} e P' \rangle$

by (*fact Sliding-event-transL*)

lemma $\langle P \triangleright Q \xrightarrow{DT} \tau Q \rangle$

by (*fact Sliding- τ -transR*)

P \triangle *Q* laws

lemma *Interrupt- τ -trans-left*: $\langle P \xrightarrow{DT} \tau P' \implies P \triangle Q \xrightarrow{DT} \tau P' \triangle Q \rangle$

by (*simp add: mono-Interrupt-DT*)

lemma *Interrupt- τ -trans-right*: $\langle Q \xrightarrow{DT} \tau Q' \implies P \triangle Q \xrightarrow{DT} \tau P \triangle Q' \rangle$

by (*simp add: mono-Interrupt-DT*)

lemma *Interrupt-ev-trans-left*:

$$\langle P \xrightarrow{DT} (ev\ e)\ P' \implies P \triangle Q \xrightarrow{DT} (ev\ e)\ P' \triangle Q \rangle$$

by (*simp add: AfterExt-def After-Interrupt Interrupt- τ -trans-left ready-set-Interrupt*)

lemma *Interrupt-ev-trans-right*: $\langle Q \xrightarrow{DT} (ev\ e)\ Q' \implies P \triangle Q \xrightarrow{DT} (ev\ e)\ P \triangle Q' \rangle$

by (*simp add: AfterExt-def After-Interrupt ready-set-Interrupt*)

Throw P A Q laws

lemma *Throw- τ -trans-left*:

$\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \Theta a \in A. Q a \text{ }_{DT \rightsquigarrow \tau} P' \Theta a \in A. Q a \rangle$
by (*simp add: mono-Throw-DT*)

lemma *Throw- τ -trans-right:*

$\langle \forall a \in A. Q a \text{ }_{DT \rightsquigarrow \tau} Q' a \implies P \Theta a \in A. Q a \text{ }_{DT \rightsquigarrow \tau} P \Theta a \in A. Q' a \rangle$
by (*simp add: mono-Throw-DT*)

lemma *Throw-event-trans-left:*

$\langle P \text{ }_{DT \rightsquigarrow e} P' \implies e \notin \text{ev } A \implies P \Theta a \in A. Q a \text{ }_{DT \rightsquigarrow e} (P' \Theta a \in A. Q a) \rangle$
apply (*simp add: AfterExt-Throw ready-set-Throw image-iff split: event.split*)
apply (*intro conjI impI*)
by (*metis AfterExt-def Throw- τ -trans-left event.simps(4)*)
(solves <simp add: Throw-STOP tick-trans-iff>)

lemma *Throw-ev-trans-right:*

$\langle P \text{ }_{DT \rightsquigarrow} (\text{ev } e) P' \implies e \in A \implies P \Theta a \in A. Q a \text{ }_{DT \rightsquigarrow} (\text{ev } e) (Q e) \rangle$
by (*simp add: AfterExt-Throw ready-set-Throw split: event.split*)

lemma $\langle \text{front-tickFree } s \implies \perp \text{ }_{DT \rightsquigarrow^*} s P \rangle$

by (*fact BOT-trace-trans-anything*)

9.2 Reality Checks

lemma $\langle \text{STOP } \text{ }_{DT \rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = \text{STOP} \rangle$

by (*fact STOP-trace-trans-iff*)

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \text{ }_{DT \rightsquigarrow^*} s P') \rangle$

using *T-imp-exists-trace-trans leDT-imp-leT trace-trans-imp-T-if- τ -trans-imp-leT*

by *blast*

lemma *D-iff-trace-trans-BOT*: $\langle s \in \mathcal{D} P \longleftrightarrow P \text{ }_{DT \rightsquigarrow^*} s \perp \rangle$

using *D-imp-trace-trans-BOT leDT-imp-leD trace-trans-BOT-imp-D-if- τ -trans-imp-leD*

by *blast*

9.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace:*

$\langle P \text{ }_{DT \rightsquigarrow^*} s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$

by (*metis T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace- τ -trans τ -trans-anti-mono-ready-set*)

lemma *trace-trans-imp-ready-set:*

$\langle P \text{ }_{DT \rightsquigarrow^*} (s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$

using *T-iff-exists-trans ready-set-AfterTrace* **by** *blast*

lemma *AfterTrace- τ -trans-iff- τ -trans-imp-leT* :
 $\langle (P \text{ }_{DT\rightsquigarrow}^* s \text{ } Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \text{ }_{DT\rightsquigarrow\tau} Q \rangle$
using *T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace- τ -trans* **by** *blast*

9.4 Summary: Operational Rules

In this section, we will just write down the operational laws that we have proven in a fancy way.

Absorbance rules

$$\frac{\frac{?P \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P' \quad ?P' \text{ }_{DT\rightsquigarrow\tau} ?P''}{?P \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P''}}{?P \text{ }_{DT\rightsquigarrow\tau} ?P' \quad ?P' \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P''}{?P \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P''}$$

SKIP rule

$$\overline{\text{SKIP }_{DT\rightsquigarrow} \checkmark \text{ STOP}}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ }_a \text{ }_{DT\rightsquigarrow} \text{ev } ?e \text{ } ?P \text{ } ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ }_a \text{ }_{DT\rightsquigarrow} \text{ev } ?e \text{ } ?P \text{ } ?e}$$

$$\overline{?e \rightarrow ?P \text{ }_{DT\rightsquigarrow} \text{ev } ?e \text{ } ?P}$$

(\square) rules

$$\frac{\frac{?P \square ?Q \text{ }_{DT\rightsquigarrow\tau} ?P \quad ?P \square ?Q \text{ }_{DT\rightsquigarrow\tau} ?Q}{?e \in ?A}}{\square a \in ?A. ?P \text{ }_a \text{ }_{DT\rightsquigarrow\tau} ?P \text{ } ?e}$$

$\mu x. f x$ rule

$$\frac{\text{cont } ?f \quad ?P = (\mu x. ?f x)}{?P \text{ }_{DT\rightsquigarrow\tau} ?f \text{ } ?P}$$

(□) rules (more powerful than in OpSemFD)

$$?P \square ?Q \xrightarrow{DT} ?P \quad ?P \square ?Q \xrightarrow{DT} ?Q$$

(;) rules

$$\frac{?P \xrightarrow{DT} ?P'}{?P ; ?Q \xrightarrow{DT} ?P' ; ?Q} \quad \frac{?P \xrightarrow{DT} ev \ ?e \ ?P'}{?P ; ?Q \xrightarrow{DT} ev \ ?e \ ?P' ; ?Q}$$

$$\frac{\exists P'. ?P \xrightarrow{DT} \checkmark P'}{?P ; ?Q \xrightarrow{DT} ?Q}$$

Hiding rules

$$\frac{?P \xrightarrow{DT} ?P'}{?P \setminus ?B \xrightarrow{DT} ?P' \setminus ?B} \quad \frac{?P \xrightarrow{DT} \checkmark ?P'}{?P \setminus ?B \xrightarrow{DT} \checkmark STOP}$$

$$\frac{?P \xrightarrow{DT} ev \ ?e \ ?P' \quad ?e \notin ?B}{?P \setminus ?B \xrightarrow{DT} ev \ ?e \ ?P' \setminus ?B} \quad \frac{?P \xrightarrow{DT} ev \ ?e \ ?P' \quad ?e \in ?B}{?P \setminus ?B \xrightarrow{DT} ?P' \setminus ?B}$$

Renaming rules

$$\frac{?P \xrightarrow{DT} ?P'}{\text{Renaming } ?P \ ?f \xrightarrow{DT} \text{Renaming } ?P' \ ?f}$$

$$\frac{\text{Renaming } ?P \ ?f \xrightarrow{DT} \checkmark STOP}{?f \ ?a = ?b \quad ?P \xrightarrow{DT} ev \ ?a \ ?P'}$$

$$\frac{\text{Renaming } ?P \ ?f \xrightarrow{DT} ev \ ?b \ \text{Renaming } ?P' \ ?f}{?P \xrightarrow{DT} \checkmark ?P'}$$

Sync rules

$$\frac{?P \xrightarrow{DT} ?P'}{?P \ [?S] \ ?Q \xrightarrow{DT} ?P' \ [?S] \ ?Q} \quad \frac{?Q \xrightarrow{DT} ?Q'}{?P \ [?S] \ ?Q \xrightarrow{DT} ?P \ [?S] \ ?Q'}$$

$$\frac{?e \notin ?S \quad ?P \xrightarrow{DT} ev \ ?e \ ?P'}{?P \ [?S] \ ?Q \xrightarrow{DT} ev \ ?e \ ?P' \ [?S] \ ?Q}$$

$$\frac{?e \notin ?S \quad ?Q \xrightarrow{DT} ev \ ?e \ ?Q'}{?P \ [?S] \ ?Q \xrightarrow{DT} ev \ ?e \ ?P \ [?S] \ ?Q'}$$

$$\frac{?e \in ?S \quad ?P \xrightarrow{DT} ev \ ?e \ ?P' \quad ?Q \xrightarrow{DT} ev \ ?e \ ?Q'}{?P \ [?S] \ ?Q \xrightarrow{DT} ev \ ?e \ ?P' \ [?S] \ ?Q'}$$

$$\frac{\frac{\frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow \tau} \text{SKIP} \llbracket ?S \rrbracket ?Q}}{?Q \text{ }_{DT \rightsquigarrow \checkmark} ?Q'}}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \llbracket ?S \rrbracket \text{SKIP}}$$

$$\frac{}{\text{SKIP} \llbracket ?S \rrbracket \text{SKIP} \text{ }_{DT \rightsquigarrow \tau} \text{SKIP}}$$

(▷) rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \triangleright ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \triangleright ?Q} \quad \frac{?P \text{ }_{DT \rightsquigarrow ?e} ?P'}{?P \triangleright ?Q \text{ }_{DT \rightsquigarrow ?e} ?P'}$$

$$\frac{}{?P \triangleright ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q}$$

(△) rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \triangle ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \triangle ?Q'}$$

$$\frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \triangle ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}$$

Throw rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \ominus a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?P' \ominus a \in ?A. ?Q a}$$

$$\frac{\forall a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?Q' a}{?P \ominus a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?P \ominus a \in ?A. ?Q' a}$$

$$\frac{?P \text{ }_{DT \rightsquigarrow ?e} ?P' \quad ?e \notin ev \text{ ' } ?A}{?P \ominus a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow ?e} ?P' \ominus a \in ?A. ?Q a}$$

$$\frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?e \in ?A}{?P \ominus a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow ev} ?e ?Q ?e}$$

end

Chapter 10

Extension of the After Operator, bis

```
theory AfterExtBis
  imports After
begin
```

10.1 The AfterExt Operator, bis

10.1.1 Definition

The refinements (\sqsubseteq_F) and (\sqsubseteq_T) are not verifying the locale axioms. In order to make the constructions available for these refinements, we will slightly modify the definition of AfterExt.

If the event is \checkmark we obtain *STOP* anyway, even if the process was \perp .

At first this appears a little weird, but can be interpreted as the fact that even if a process is diverging, after accepting \checkmark it has to stop.

definition *AfterExt* :: $\langle ['\alpha \text{ process}, '\alpha \text{ event}] \Rightarrow '\alpha \text{ process} \rangle$ (**infixl** $\langle \text{afterExt} \rangle$ 77)
where $\langle P \text{ afterExt } e \equiv \text{case } e \text{ of } \text{ev } x \Rightarrow P \text{ after } x \mid \checkmark \Rightarrow \text{STOP} \rangle$

lemma *not-ready-AfterExt*: $\langle e \notin \text{ready-set } P \Longrightarrow P \text{ afterExt } e = \text{STOP} \rangle$
by (*auto simp add: AfterExt-def intro: not-ready-After split: event.split*)

lemma *ready-set-AfterExt*:
 $\langle \text{ready-set } (P \text{ afterExt } e) = (\text{if } e = \checkmark \text{ then } \{\} \text{ else } \{a. e \# [a] \in \mathcal{T} P\}) \rangle$
by (*simp add: AfterExt-def ready-set-After ready-set-STOP split: event.split*)

10.1.2 Projections

lemma *F-AfterExt*:
 $\langle \mathcal{F} (P \text{ afterExt } e) =$

(if $e \in \text{ready-set } P$ then $\{(tl\ s, X) \mid s\ X. (s, X) \in \mathcal{F}\ P \wedge s \neq [] \wedge hd\ s = e\}$
 else $\{(s, X). s = []\}$)
 (is $\langle - = ?rhs \rangle$)
proof (unfold *AfterExt-def*, *split event.split*, *intro conjI allI impI*)
show $\langle e = ev\ x \implies \mathcal{F}\ (P\ \text{after}\ x) = ?rhs \rangle$ **for** x
by (*simp add: F-After*)
next
show $\langle e = \checkmark \implies \mathcal{F}\ STOP = ?rhs \rangle$
by (*simp add: F-UU F-STOP BOT-iff-D ready-set-def set-eq-iff*)
 (*metis F-T T-nonTickFree-imp-decomp append-single-T-imp-tickFree hd-append2*
hd-in-set list.discI list.sel(1) list.sel(3) tickFree-def tick-T-F)
qed

lemma *D-AfterExt*: $\langle \mathcal{D}\ (P\ \text{afterExt}\ e) = (\text{ if } e = \checkmark \wedge P = \perp \text{ then } \{\} \text{ else } \{tl\ s \mid s . s \in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = e\}) \rangle$
 (is $\langle - = ?rhs \rangle$)
proof (unfold *AfterExt-def*, *split event.split*, *intro conjI allI impI*)
show $\langle e = ev\ x \implies \mathcal{D}\ (P\ \text{after}\ x) = ?rhs \rangle$ **for** x
by (*simp add: D-After*)
 (*metis Cons-in-T-imp-elem-ready-set D-T list.exhaust-sel*)
next
show $\langle e = \checkmark \implies \mathcal{D}\ STOP = ?rhs \rangle$
by (*simp add: D-UU D-STOP BOT-iff-D*)
 (*metis D-imp-front-tickFree front-tickFree-implies-tickFree hd-append2*
hd-in-set is-processT9 nonTickFree-n-frontTickFree tickFree-def)
qed

lemma *T-AfterExt*: $\langle \mathcal{T}\ (P\ \text{afterExt}\ e) = (\text{ if } e = \checkmark \wedge P = \perp \text{ then } \{\} \text{ else } \text{insert } [] \{tl\ s \mid s . s \in \mathcal{T}\ P \wedge s \neq [] \wedge hd\ s = e\}) \rangle$
 (is $\langle - = ?rhs \rangle$)
proof (unfold *AfterExt-def*, *split event.split*, *intro conjI allI impI*)
show $\langle e = ev\ x \implies \mathcal{T}\ (P\ \text{after}\ x) = ?rhs \rangle$ **for** x
by (*simp add: T-After set-eq-iff subset-iff*)
 (*metis Cons-in-T-imp-elem-ready-set list.collapse*
list.distinct(1) list.sel(1, 3) mem-Collect-eq ready-set-def)
next
show $\langle e = \checkmark \implies \mathcal{T}\ STOP = ?rhs \rangle$
by (*simp add: T-After T-UU T-STOP subset-iff*)
 (*metis front-tickFree-charn hd-append2 hd-in-set*
is-processT2-TR list.sel(3) tickFree-def tl-append-if)
qed

10.1.3 Monotony

lemma *mono-AfterExt* : $\langle P \sqsubseteq Q \implies P\ \text{afterExt}\ e \sqsubseteq Q\ \text{afterExt}\ e \rangle$
by (*auto simp add: AfterExt-def mono-After split: event.split*)

lemma *mono-AfterExt-T* : $\langle P \sqsubseteq_T Q \implies P \text{ afterExt } e \sqsubseteq_T Q \text{ afterExt } e \rangle$
by (*auto simp add: AfterExt-def mono-After-T split: event.split*)

lemma *mono-AfterExt-F* :
 $\langle P \sqsubseteq_F Q \implies ev \ e \notin \text{ready-set } P \vee ev \ e \in \text{ready-set } Q \implies$
 $P \text{ afterExt } ev \ e \sqsubseteq_F Q \text{ afterExt } ev \ e \rangle$
by (*simp add: AfterExt-def mono-After-F*)

lemma *mono-AfterExt-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterExt } e \sqsubseteq_D Q \text{ afterExt } e \rangle$
by (*simp add: AfterExt-def mono-After-D split: event.split*)

lemma *mono-AfterExt-FD* :
 $\langle P \sqsubseteq_{FD} Q \implies e \notin \text{ready-set } P \vee e \in \text{ready-set } Q \implies$
 $P \text{ afterExt } e \sqsubseteq_{FD} Q \text{ afterExt } e \rangle$
by (*auto simp add: AfterExt-def mono-After-FD split: event.split*)

lemma *mono-AfterExt-DT* : $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterExt } e \sqsubseteq_{DT} Q \text{ afterExt } e \rangle$
by (*simp add: AfterExt-def mono-After-DT split: event.split*)

10.1.4 Behaviour of *AfterExt* with *STOP*, *SKIP* and \perp

lemma *AfterExt-STOP*: $\langle STOP \text{ afterExt } e = STOP \rangle$
by (*simp add: STOP-neq-BOT STOP-iff-T T-AfterExt ready-set-STOP T-STOP*)

lemma *AfterExt-is-STOP-iff*:
 $\langle P \text{ afterExt } e = STOP \iff P = \perp \wedge e = \checkmark \vee (\forall s. e \# s \in \mathcal{T} \ P \longrightarrow s = []) \rangle$
by (*simp add: AfterExt-def After-is-STOP-iff split: event.split*)
(metis T-nonTickFree-imp-decomp append-single-T-imp-tickFree
butlast.simps(2) butlast-snoc tickFree-Cons)

lemma *AfterExt-SKIP*: $\langle SKIP \text{ afterExt } e = STOP \rangle$
by (*auto simp add: SKIP-neq-BOT STOP-iff-T T-AfterExt ready-set-SKIP T-SKIP*)

lemma *AfterExt-BOT* : $\langle \perp \text{ afterExt } e = (\text{if } e = \checkmark \text{ then } STOP \text{ else } \perp) \rangle$
by (*metis AfterExt-def After-BOT event.case event.exhaust*)

lemma *AfterExt-is-BOT-iff*: $\langle P \text{ afterExt } e = \perp \iff e \neq \checkmark \wedge [e] \in \mathcal{D} \ P \rangle$
by (*simp add: AfterExt-def After-is-BOT-iff STOP-neq-BOT split: event.split*)

10.1.5 Behaviour of *AfterExt* with Operators of HOL-CSP

Here again, we lose determinism.

lemma *AfterExt-Mprefix-is-AfterExt-Mndetprefix*:
 $\langle (\Box a \in A \rightarrow P \ a) \text{ afterExt } e = (\Box a \in A \rightarrow P \ a) \text{ afterExt } e \rangle$
by (*simp add: AfterExt-def After-Mprefix-is-After-Mndetprefix*
Mprefix-neq-BOT Mndetprefix-neq-BOT split: event.split)

lemma *AfterExt-Det-is-AfterExt-Ndet*: $\langle P \sqcap Q \text{ afterExt } e = P \sqcap Q \text{ afterExt } e \rangle$
by (*simp add: AfterExt-def After-Det-is-After-Ndet Det-is-BOT-iff Ndet-is-BOT-iff split: event.split*)

lemma *AfterExt-Ndet*:
 $\langle P \sqcap Q \text{ afterExt } e = (\text{case } e \text{ of } ev \ x \Rightarrow \text{if } ev \ x \in \text{ready-set } P \cap \text{ready-set } Q$
 $\text{then } (P \text{ afterExt } ev \ x) \sqcap (Q \text{ afterExt } ev \ x)$
 $\text{else if } ev \ x \in \text{ready-set } P$
 $\text{then } P \text{ afterExt } ev \ x$
 $\text{else } Q \text{ afterExt } ev \ x$
 $| \checkmark \Rightarrow STOP) \rangle$
by (*auto simp add: AfterExt-def After-Ndet not-ready-After split: event.split*)

lemma *AfterExt-Mprefix*:
 $\langle (\sqcap a \in A \rightarrow P \ a) \text{ afterExt } e =$
 $(\text{case } e \text{ of } ev \ x \Rightarrow \text{if } x \in A \text{ then } P \ x \text{ else } STOP | \checkmark \Rightarrow STOP) \rangle$
by (*simp add: AfterExt-def Mprefix-neq-BOT After-Mprefix split: event.split*)

corollary *AfterExt-prefix*:
 $\langle (a \rightarrow P) \text{ afterExt } e =$
 $(\text{case } e \text{ of } ev \ x \Rightarrow \text{if } x = a \text{ then } P \text{ else } STOP | \checkmark \Rightarrow STOP) \rangle$
unfolding *write0-def* **by** (*simp add: AfterExt-Mprefix split: event.split*)

lemmas *AfterExt-Det = AfterExt-Ndet*[*folded AfterExt-Det-is-AfterExt-Ndet*]
and *AfterExt-Mndetprefix = AfterExt-Mprefix*[*unfolded AfterExt-Mprefix-is-AfterExt-Mndetprefix*]

lemma *Renaming-is-BOT-iff*: $\langle \text{Renaming } P \ f = \perp \longleftrightarrow P = \perp \rangle$
apply (*intro iffI*)
apply (*simp add: BOT-iff-D D-Renaming*)
by (*simp add: Renaming-BOT*)

lemma *Renaming-is-STOP-iff*: $\langle \text{Renaming } P \ f = STOP \longleftrightarrow P = STOP \rangle$
apply (*intro iffI, simp-all add: Renaming-STOP*)
by (*auto simp add: STOP-iff-T T-Renaming intro: Nil-elem-T*)

lemma *AfterExt-Renaming*:
 $\langle \text{Renaming } P \ f \text{ afterExt } e =$
 $(\text{case } e \text{ of } \checkmark \Rightarrow STOP$
 $| ev \ a \Rightarrow \text{if } P = \perp \text{ then } \perp \text{ else}$
 $\sqcap a \in \{a. ev \ a \in \text{ready-set } P \wedge ev \ (f \ a) = e\}. \text{ Renaming } (P \text{ afterExt } ev \ a) \ f) \rangle$
by (*simp add: AfterExt-def After-Renaming Renaming-is-BOT-iff split: event.split*)

— Move this result in HOL-CSP

lemma *Seq-is-BOT-iff*: $\langle P ; Q = \perp \longleftrightarrow P = \perp \vee ([\checkmark] \in \mathcal{T} P \wedge Q = \perp) \rangle$
by (*auto simp add: BOT-iff-D D-Seq D-UU*)

lemma *AfterExt-Seq*:

$\langle (P ; Q) \text{ afterExt } e =$
 (*if* $e \notin \text{ready-set } P \wedge e \notin \text{ready-set } Q$ *then* *STOP*
else if $e \notin \text{ready-set } Q$ *then* $P \text{ afterExt } e ; Q$
else if $e \notin \text{ready-set } P$ *then if* $\checkmark \in \text{ready-set } P$ *then* $Q \text{ afterExt } e$ *else* *STOP*
else if $\checkmark \in \text{ready-set } P$ *then* $(P \text{ afterExt } e ; Q) \sqcap (Q \text{ afterExt } e)$
else $P \text{ afterExt } e ; Q \rangle$
by (*simp add: AfterExt-def After-Seq Ndet-id Ndet-is-BOT-iff*
Seq-is-BOT-iff ready-set-def T-UU STOP-Seq split: event.split)

theorem *AfterExt-Sync*:

$\langle (P \llbracket S \rrbracket Q) \text{ afterExt } e =$
 (*case* e *of* $\checkmark \Rightarrow \text{STOP}$
 | $ev\ x \Rightarrow$ *if* $P = \perp \vee Q = \perp$ *then* \perp
 else if $e \in \text{ready-set } P \wedge e \in \text{ready-set } Q$
 then if $x \in S$
 then $P \text{ afterExt } e \llbracket S \rrbracket Q \text{ afterExt } e$
 else $(P \text{ afterExt } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ afterExt } e)$
 else if $e \in \text{ready-set } P$
 then if $x \in S$ *then* *STOP* *else* $P \text{ afterExt } e \llbracket S \rrbracket Q$
 else if $e \in \text{ready-set } Q$
 then if $x \in S$ *then* *STOP* *else* $P \llbracket S \rrbracket Q \text{ afterExt } e$
 else *STOP*)
by (*simp add: AfterExt-def After-Sync split: event.split*)

lemma *Hiding-FD-Hiding-AfterExt-if-ready-inside*:

$\langle e \in B \Longrightarrow (P \setminus B) \sqsubseteq_{FD} (P \text{ afterExt } ev\ e \setminus B) \rangle$
and *AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin*:
 $\langle e \notin B \Longrightarrow (P \setminus B) \text{ afterExt } ev\ e \sqsubseteq_{FD} (P \text{ afterExt } ev\ e \setminus B) \rangle$
if *ready*: $\langle ev\ e \in \text{ready-set } P \rangle$

by (*simp add: AfterExt-def Hiding-FD-Hiding-After-if-ready-inside that*)
 (*simp add: AfterExt-def After-Hiding-FD-Hiding-After-if-ready-notin that*)

lemmas *Hiding-F-Hiding-AfterExt-if-ready-inside* =

Hiding-FD-Hiding-AfterExt-if-ready-inside[*THEN leFD-imp-leF*]

and *AfterExt-Hiding-F-Hiding-AfterExt-if-ready-notin* =

AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin[*THEN leFD-imp-leF*]

and *Hiding-D-Hiding-AfterExt-if-ready-inside* =
 $Hiding-FD-Hiding-AfterExt-if-ready-inside[THEN\ leFD-imp-leD]$
and *AfterExt-Hiding-D-Hiding-AfterExt-if-ready-notin* =
 $AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin[THEN\ leFD-imp-leD]$
and *Hiding-T-Hiding-AfterExt-if-ready-inside* =
 $Hiding-FD-Hiding-AfterExt-if-ready-inside[THEN\ leFD-imp-leF,\ THEN\ leF-imp-leT]$

and *AfterExt-Hiding-T-Hiding-AfterExt-if-ready-notin* =
 $AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin[THEN\ leFD-imp-leF,\ THEN\ leF-imp-leT]$

corollary *Hiding-DT-Hiding-AfterExt-if-ready-inside*:

$\langle [ev\ e \in ready-set\ P; e \in B] \implies (P \setminus B) \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
and *AfterExt-Hiding-DT-Hiding-AfterExt-if-ready-notin*:
 $\langle [ev\ e \in ready-set\ P; e \notin B] \implies (P \setminus B)\ afterExt\ ev\ e \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
by (*simp add: Hiding-D-Hiding-AfterExt-if-ready-inside*
 $Hiding-T-Hiding-AfterExt-if-ready-inside\ leD-leT-imp-leDT$)
(*simp add: AfterExt-Hiding-D-Hiding-AfterExt-if-ready-notin*
 $AfterExt-Hiding-T-Hiding-AfterExt-if-ready-notin\ leD-leT-imp-leDT$)

10.1.6 Behaviour of *AfterExt* with Operators of HOL-CSPM

lemma *AfterExt-MultiDet-is-AfterExt-MultiNdet*:

$\langle finite\ A \implies (\sqcap\ a \in A.\ P\ a)\ afterExt\ e = (\sqcap\ a \in A.\ P\ a)\ afterExt\ e \rangle$
by (*auto simp add: AfterExt-def After-MultiDet-is-After-MultiNdet*
 $MultiDet-is-BOT-iff\ MultiNdet-is-BOT-iff\ split:\ event.split$)

lemma *AfterExt-GlobalNdet*:

$\langle (\sqcap\ a \in A.\ P\ a)\ afterExt\ e =$
 $(\ if\ e = \checkmark \vee e \notin (\bigcup\ a \in A.\ ready-set\ (P\ a))\ then\ STOP$
 $\ else\ (\sqcap\ a \in \{a \in A.\ e \in ready-set\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
and *AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\sqcap\ a \in A.\ P\ a)\ afterExt\ e =$
 $(\ if\ e = \checkmark \vee e \notin (\bigcup\ a \in A.\ ready-set\ (P\ a))\ then\ STOP$
 $\ else\ (\sqcap\ a \in \{a \in A.\ e \in ready-set\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
and *AfterExt-MultiDet*:
 $\langle finite\ A \implies (\sqcap\ a \in A.\ P\ a)\ afterExt\ e =$
 $(\ if\ e = \checkmark \vee e \notin (\bigcup\ a \in A.\ ready-set\ (P\ a))\ then\ STOP$
 $\ else\ (\sqcap\ a \in \{a \in A.\ e \in ready-set\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
by (*simp-all add: AfterExt-def split: event.split*)
(*simp add: After-GlobalNdet, simp add: After-MultiNdet, simp add: After-MultiDet*)

10.1.7 Behaviour of *AfterExt* with Operators of HOL-CSP_OpSem

lemma *AfterExt-Sliding-is-AfterExt-Ndet*:

$\langle P \triangleright Q\ afterExt\ e = P \sqcap Q\ afterExt\ e \rangle$
by (*simp add: AfterExt-def After-Sliding-is-After-Ndet Sliding-is-BOT-iff Ndet-is-BOT-iff*
 $split:\ event.split$)

lemmas *AfterExt-Sliding* = *AfterExt-Ndet*[*folded AfterExt-Sliding-is-AfterExt-Ndet*]

lemma *AfterExt-Throw*:

$\langle (P \Theta a \in A. Q a) \text{ afterExt } e =$
 (case e of $\checkmark \Rightarrow STOP$
 | $ev\ x \Rightarrow$ if $P = \perp$ then \perp
 else if $ev\ x \in \text{ready-set } P$ then if $x \in A$ then $Q\ x$
 else $(P \text{ after } x) \Theta a \in A. Q a$ else $STOP$) \rangle
by (*simp add: AfterExt-def After-Throw Throw-is-BOT-iff split: event.split*)

lemma *AfterExt-Interrupt*:

$\langle (P \Delta Q) \text{ afterExt } e =$
 (case e of $\checkmark \Rightarrow STOP$
 | $ev\ x \Rightarrow$ if $P = \perp \vee Q = \perp$ then \perp
 else if $ev\ x \in \text{ready-set } P \wedge ev\ x \in \text{ready-set } Q$
 then $(Q \text{ after } x) \sqcap (P \text{ after } x \Delta Q)$
 else if $ev\ x \in \text{ready-set } P \wedge ev\ x \notin \text{ready-set } Q$
 then $P \text{ after } x \Delta Q$
 else if $ev\ x \notin \text{ready-set } P \wedge ev\ x \in \text{ready-set } Q$
 then $Q \text{ after } x$
 else $STOP$) \rangle
by (*simp add: AfterExt-def After-Interrupt Interrupt-is-BOT-iff*
Ndet-is-BOT-iff ready-set-BOT After-is-BOT-iff D-UU split: event.split)

10.1.8 Behaviour of *AfterExt* with Reference Processes

lemma *AfterExt-DF*:

$\langle DF\ A \text{ afterExt } e =$
 (case e of $ev\ x \Rightarrow$ if $x \in A$ then $DF\ A$ else $STOP$ | $\checkmark \Rightarrow STOP$) \rangle
by (*cases e*) (*simp-all add: AfterExt-def After-DF BOT-iff-D div-free-DF*)

lemma *AfterExt-DF_{SKIP}*:

$\langle DF_{SKIP}\ A \text{ afterExt } e =$
 (case e of $ev\ x \Rightarrow$ if $x \in A$ then $DF_{SKIP}\ A$ else $STOP$ | $\checkmark \Rightarrow STOP$) \rangle
by (*cases e*) (*simp-all add: AfterExt-def After-DF_{SKIP} BOT-iff-D div-free-DF_{SKIP}*)

lemma *AfterExt-RUN*:

$\langle RUN\ A \text{ afterExt } e =$
 (case e of $ev\ x \Rightarrow$ if $x \in A$ then $RUN\ A$ else $STOP$ | $\checkmark \Rightarrow STOP$) \rangle
by (*cases e*) (*simp-all add: AfterExt-def After-RUN BOT-iff-D div-free-RUN*)

lemma *AfterExt-CHAOS*:

$\langle CHAOS\ A \text{ afterExt } e =$
 (case e of $ev\ x \Rightarrow$ if $x \in A$ then $CHAOS\ A$ else $STOP$ | $\checkmark \Rightarrow STOP$) \rangle
by (*cases e*) (*simp-all add: AfterExt-def After-CHAOS BOT-iff-D div-free-CHAOS*)

lemma *AfterExt-CHAOS_{SKIP}*:
 ‹*CHAOS_{SKIP} A afterExt e =*
*(case e of ev x ⇒ if x ∈ A then CHAOS_{SKIP} A else STOP | ✓ ⇒ STOP)›
by (*cases e*) (*simp-all add: AfterExt-def After-CHAOS_{SKIP} BOT-iff-D div-free-CHAOS_{SKIP}*)*

lemma *DF-FD-AfterExt*:
 ‹*DF A ⊆_{FD} P ⇒ e ∈ ready-set P ⇒ DF A ⊆_{FD} P afterExt e*›
apply (*cases e, simp add: AfterExt-def DF-FD-After*)
by (*metis anti-mono-ready-set-FD event.distinct(1) image-iff ready-set-DF subsetD*)

lemma *DF_{SKIP}-FD-AfterExt*:
 ‹*DF_{SKIP} A ⊆_{FD} P ⇒ ev e ∈ ready-set P ⇒ DF_{SKIP} A ⊆_{FD} P afterExt ev*
e›
by (*simp add: AfterExt-def DF_{SKIP}-FD-After*)

lemma *deadlock-free-AfterExt*:
 ‹*deadlock-free P ⇒ deadlock-free (P afterExt e) ⟷*
(if e ∈ ready-set P ∧ e ≠ ✓ then True else False)›
by (*cases e*)
(simp add: AfterExt-def deadlock-free-After,
simp add: AfterExt-def BOT-iff-D deadlock-free-implies-div-free non-deadlock-free-STOP)

lemma *deadlock-free_{SKIP}-AfterExt*:
 ‹*deadlock-free_{SKIP} P ⇒ deadlock-free_{SKIP} (P afterExt e) ⟷*
(if e ∈ ready-set P ∧ e ≠ ✓ then True else False)›
by (*cases e*)
(simp add: AfterExt-def deadlock-free_{SKIP}-After,
simp add: AfterExt-def BOT-iff-D deadlock-free_{SKIP}-implies-div-free non-deadlock-free_{SKIP}-STOP)

end

10.2 The AfterTrace Operator, bis

theory *AfterTraceBis*
imports *AfterExtBis HOL-CSPM.DeadlockResults*
begin

10.2.1 Definition

We just defined $P \text{ afterExt } e$ for P an ' α process and e of type ' α event. Since traces of an ' α process are ' α event list, the following inductive definition is

natural.

fun *AfterTrace* :: $\langle ' \alpha \text{ process} \Rightarrow ' \alpha \text{ trace} \Rightarrow ' \alpha \text{ process} \rangle$ (**infixl** $\langle \text{afterTrace} \rangle$ 77)
where $\langle P \text{ afterTrace} [] = P \rangle$
| $\langle P \text{ afterTrace} (e \# s) = P \text{ afterExt } e \text{ afterTrace } s \rangle$

We can also induct backward.

lemma *AfterTrace-append*: $\langle P \text{ afterTrace} (s @ t) = P \text{ afterTrace } s \text{ afterTrace } t \rangle$
apply (*induct t rule: rev-induct, simp*)
apply (*induct s rule: rev-induct, simp*)
by (*metis AfterTrace.simps append.assoc append.right-neutral append-Cons append-Nil*)

lemma *AfterTrace-snoc*: $\langle P \text{ afterTrace} (s @ [e]) = P \text{ afterTrace } s \text{ afterExt } e \rangle$
by (*simp add: AfterTrace-append*)

We have some immediate properties.

lemma *AfterTrace-STOP*: $\langle \text{STOP} \text{ afterTrace } s = \text{STOP} \rangle$
by (*induct s (simp-all add: AfterExt-STOP)*)

lemma *AfterTrace-SKIP*: $\langle \text{SKIP} \text{ afterTrace } s = (\text{if } s = [] \text{ then } \text{SKIP} \text{ else } \text{STOP}) \rangle$
by (*induct s (simp-all add: AfterExt-SKIP AfterTrace-STOP)*)

lemma *AfterTrace-BOT*: $\langle \perp \text{ afterTrace } s = (\text{if } \text{tickFree } s \text{ then } \perp \text{ else } \text{STOP}) \rangle$
by (*induct s (simp-all add: AfterExt-BOT AfterTrace-STOP)*)

10.2.2 Projections

lemma *F-AfterTrace*: $\langle (s @ t, X) \in \mathcal{F} P \Longrightarrow (t, X) \in \mathcal{F} (P \text{ afterTrace } s) \rangle$
apply (*induct s arbitrary: t rule: rev-induct, simp*)
apply (*simp add: AfterTrace-snoc F-AfterExt*)
by (*metis Cons-in-T-imp-elem-ready-set F-T list.discI list.sel(1, 3)*)

lemma *D-AfterTrace*:
 $\langle \text{tickFree } (s @ t) \Longrightarrow s @ t \in \mathcal{D} P \Longrightarrow t \in \mathcal{D} (P \text{ afterTrace } s) \rangle$
apply (*induct s arbitrary: t rule: rev-induct, solves <simp>*)
by (*simp add: AfterTrace-snoc D-AfterExt,*
metis list.discI list.sel(1, 3) tickFree-Cons)

lemma *T-AfterTrace*: $\langle s @ t \in \mathcal{T} P \Longrightarrow t \in \mathcal{T} (P \text{ afterTrace } s) \rangle$
using *F-AfterTrace T-F-spec* **by** *blast*

corollary *ready-set-AfterTrace*:
 $\langle s @ e \# t \in \mathcal{T} P \Longrightarrow e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
by (*metis Cons-in-T-imp-elem-ready-set T-AfterTrace*)

corollary *F-imp-refusal-AfterTrace*:
 $\langle (s, X) \in \mathcal{F} P \Longrightarrow ([], X) \in \mathcal{F} (P \text{ afterTrace } s) \rangle$
by (*simp add: F-AfterTrace*)

corollary *D-imp-AfterTrace-is-BOT*:
 $\langle \text{tickFree } s \implies s \in \mathcal{D} P \implies P \text{ afterTrace } s = \perp \rangle$
by (*simp add: BOT-iff-D D-AfterTrace*)

10.2.3 Monotony

lemma *mono-AfterTrace* : $\langle P \sqsubseteq Q \implies P \text{ afterTrace } s \sqsubseteq Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct*) (*simp-all add: mono-AfterExt AfterTrace-snoc*)

lemma *mono-AfterTrace-T* :
 $\langle P \sqsubseteq_T Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_T Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc*)
(rule mono-AfterExt-T; use is-processT3-ST in blast)

lemma *mono-AfterTrace-F* :
 $\langle P \sqsubseteq_F Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_F Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc*)
(metis event.exhaust is-processT3-ST mono-AfterExt-F ready-set-AfterTrace)

lemma *mono-AfterTrace-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterTrace } s \sqsubseteq_D Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct*) (*simp-all add: mono-AfterExt-D AfterTrace-snoc*)

lemma *mono-AfterTrace-FD* :
 $\langle P \sqsubseteq_{FD} Q \implies s \in \mathcal{T} Q \implies P \text{ afterTrace } s \sqsubseteq_{FD} Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc*)
(meson Cons-in-T-imp-elem-ready-set T-AfterTrace is-processT3-ST mono-AfterExt-FD)

lemma *mono-AfterTrace-DT* :
 $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterTrace } s \sqsubseteq_{DT} Q \text{ afterTrace } s \rangle$
by (*induct s rule: rev-induct; simp add: AfterTrace-snoc mono-AfterExt-DT*)

10.2.4 Another Definition of *events-of*

inductive *reachable-event* :: $\langle 'a \text{ process} \Rightarrow 'a \Rightarrow \text{bool} \rangle$
where $\langle \text{ev } e \in \text{ready-set } P \implies \text{reachable-event } P \ e \rangle$
 $\mid \langle \text{reachable-event } (P \text{ after } f) \ e \implies \text{reachable-event } P \ e \rangle$

lemma *events-of-iff-reachable-event*: $\langle e \in \text{events-of } P \longleftrightarrow \text{reachable-event } P \ e \rangle$

proof (*intro iffI*)

show $\langle \text{reachable-event } P \ e \implies e \in \text{events-of } P \rangle$

apply (*induct rule: reachable-event.induct;*
simp add: T-After events-of-def ready-set-def split: if-split-asm)

by (*meson list.set-intros(1)*) (*meson list.set-sel(2)*)

next

assume $\langle e \in \text{events-of } P \rangle$

then obtain *s* **where** $\ast : \langle s \in \mathcal{T} P \rangle \langle \text{ev } e \in \text{set } s \rangle$ **unfolding** *events-of-def* **by**
blast

thus $\langle \text{reachable-event } P \ e \rangle$

```

proof (induct s arbitrary: P)
  show  $\langle \bigwedge P e. ev e \in set \ [] \implies reachable-event P e \rangle$  by simp
next
  case (Cons f s)
  from Cons.prem(1) is-processT3-ST
  have * :  $\langle f \in ready-set P \rangle$  unfolding ready-set-def by force
  from Cons.prem(2) consider  $\langle f = ev e \rangle$  |  $\langle ev e \in set s \rangle$  by auto
  thus  $\langle reachable-event P e \rangle$ 
  proof cases
    assume  $\langle f = ev e \rangle$ 
    from *[simplified this]
    show  $\langle reachable-event P e \rangle$  by (rule reachable-event.intros(1))
  next
    assume  $\langle ev e \in set s \rangle$ 
    show  $\langle reachable-event P e \rangle$ 
    proof (cases f)
      fix f'
      assume  $\langle f = ev f' \rangle$ 
      with * Cons.prem(1) have  $\langle s \in \mathcal{T} (P \text{ after } f') \rangle$  by (force simp add:
T-After)
      show  $\langle reachable-event P e \rangle$ 
      apply (rule reachable-event.intros(2))
      by (rule Cons.hyps[OF  $\langle s \in \mathcal{T} (P \text{ after } f') \rangle$ ,  $\langle ev e \in set s \rangle$ ])
    next
      from Cons.prem(1) have  $\langle f = \checkmark \implies s = [] \rangle$ 
      by simp (metis butlast.simps(2) front-tickFree-butlast is-processT2-TR
tickFree-Cons)
      thus  $\langle f = \checkmark \implies reachable-event P e \rangle$ 
      using  $\langle ev e \in set s \rangle$  by force
    qed
  qed
qed
qed

```

lemma *reachable-event-BOT*: $\langle reachable-event \perp e \rangle$
by (simp add: reachable-event.intros(1) ready-set-BOT)

lemma *not-reachable-event-STOP*: $\langle \neg reachable-event STOP e \rangle$
by (subst events-of-iff-reachable-event[symmetric])
(unfold events-of-def, simp add: T-STOP)

lemma *reachable-tick-SKIP*: $\langle \neg reachable-event SKIP \checkmark \rangle$
by (subst events-of-iff-reachable-event[symmetric])
(unfold events-of-def, simp add: T-SKIP)

lemma *reachable-event-iff-in-ready-set-AfterTrace*:

$\langle \text{reachable-event } P \ e \longleftrightarrow e \in \{e \mid e \ s. \ \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } s)\} \rangle$
proof –
have $\langle \text{reachable-event } P \ e \implies \exists s. \ \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } s) \rangle$
proof (*induct rule: reachable-event.induct*)
case (1 $e \ P$)
thus ?*case* **by** (*metis AfterTrace.simps(1)*)
next
case (2 $P \ f \ e$)
from 2.*hyps*(2) **obtain** s **where** $\langle \text{ev } e \in \text{ready-set } (P \ \text{after } f \ \text{afterTrace } s) \rangle$ **by**
blast
hence $\langle \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } (\text{ev } f \ \# \ s)) \rangle$ **by** (*simp add: AfterExt-def*)
thus ?*case* **by** *blast*
qed
also have $\langle \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } s) \implies \text{reachable-event } P \ e \rangle$ **for** s
proof (*induct s arbitrary: P*)
show $\langle \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } []) \implies \text{reachable-event } P \ e \rangle$ **for** P
by (*simp add: reachable-event.intros(1)*)
next
fix $f \ s \ P$
assume *hyp* : $\langle \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } s) \implies \text{reachable-event } P \ e \rangle$ **for**
 P
assume *prem* : $\langle \text{ev } e \in \text{ready-set } (P \ \text{afterTrace } (f \ \# \ s)) \rangle$
show $\langle \text{reachable-event } P \ e \rangle$
proof (*cases f*)
fix f'
assume $\langle f = \text{ev } f' \rangle$
show $\langle \text{reachable-event } P \ e \rangle$ **by** (*rule reachable-event.intros(2)[OF hyp]*)
(use prem in <simp add: AfterExt-def <f = ev f'>>)
next
case *tick*
with *prem* **show** $\langle f = \checkmark \implies \text{reachable-event } P \ e \rangle$
by (*simp add: AfterExt-def reachable-event-BOT AfterTrace-STOP ready-set-STOP*)

split: if-split-asm)
qed
qed
ultimately show ?*thesis* **by** *blast*
qed

10.2.5 Characterizations for Deadlock Freeness

lemma *deadlock-free-AfterExt-characterization:*

$\langle \text{deadlock-free } P \longleftrightarrow \text{range } \text{ev} \notin \mathcal{R} \ P \wedge$
 $(\forall e \in \text{ready-set } P. \ \text{deadlock-free } (P \ \text{afterExt } e)) \rangle$
(is <deadlock-free P $\longleftrightarrow ?rhs$>)

proof (*intro iffI*)

have $\langle \text{range } \text{ev} \notin \mathcal{R} \ P \longleftrightarrow \text{UNIV} - \{\checkmark\} \notin \mathcal{R} \ P \rangle$

by (*metis Diff-insert-absorb UNIV-eq-I event.simps(3) event-set image-iff*)

thus $\langle \text{deadlock-free } P \implies ?rhs \rangle$

by (*metis DF-FD-AfterExt Diff-Diff-Int Diff-partition Diff-subset F-T*
deadlock-free_{SKIP}-is-right deadlock-free-def
deadlock-free-implies-non-terminating deadlock-free-is-deadlock-free_{SKIP}
inf-top-left is-processT5-S2a non-tickFree-tick Refusals-iff self-append-conv2)

next
assume *assm* : $\langle ?rhs \rangle$
with *BOT-iff-D process-charn* **have** *non-BOT* : $\langle P \neq \perp \rangle$ **by** (*metis Refusals-iff*)
show $\langle \text{deadlock-free } P \rangle$
proof (*unfold deadlock-free-F failure-refine-def, safe*)
from *assm* **have** *** : $\langle \text{range } ev \notin \mathcal{R} P \rangle$
 $\langle e \in \text{ready-set } P \implies$
 $\{ (tl\ s, X) \mid s\ X. (s, X) \in \mathcal{F} P \wedge s \neq [] \wedge hd\ s = e \} \subseteq \mathcal{F} (DF\ UNIV) \rangle$ **for** *e*
by (*simp-all add: deadlock-free-F failure-refine-def F-AfterExt non-BOT*)

fix *s X*
assume **** : $\langle (s, X) \in \mathcal{F} P \rangle$
show $\langle (s, X) \in \mathcal{F} (DF\ UNIV) \rangle$
proof (*cases s*)
show $\langle s = [] \implies (s, X) \in \mathcal{F} (DF\ UNIV) \rangle$
by (*subst F-DF, simp*)
*(metis *(1) ** Refusals-iff image-subset-iff is-processT4)*

next
fix *e s'*
assume ***** : $\langle s = e \# s' \rangle$
with *** Cons-in-T-imp-elem-ready-set F-T* **have** $\langle e \in \text{ready-set } P \rangle$ **by** *blast*
with **(2)[OF this]* **show** $\langle (s, X) \in \mathcal{F} (DF\ UNIV) \rangle$
by (*subst F-DF, simp add: subset-iff*)
*(metis (no-types, lifting) *(1) ** *** CollectD Refusals-iff*
event-set hd-append2 hd-in-set in-set-conv-decomp-first insert-iff
is-processT6-S2 list.sel(1) list.set-intros(1) rangeE ready-set-def)

qed
qed
qed

lemma *deadlock-free_{SKIP}-AfterExt-characterization*:
 $\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow$
 $UNIV \notin \mathcal{R} P \wedge (\forall e \in \text{ready-set } P - \{\checkmark\}. \text{deadlock-free}_{SKIP} (P\ \text{afterExt } e)) \rangle$
(is $\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow ?rhs \rangle$)
proof (*intro iffI*)
show $\langle \text{deadlock-free}_{SKIP} P \implies ?rhs \rangle$
by (*metis Diff-iff Nil-elem-T deadlock-free_{SKIP}-AfterExt*
deadlock-free_{SKIP}-is-right insertI1 Refusals-iff tickFree-Nil)

next
assume *assm* : $\langle ?rhs \rangle$
with *BOT-iff-D process-charn* **have** *non-BOT* : $\langle P \neq \perp \rangle$ **by** (*metis Refusals-iff*)
show $\langle \text{deadlock-free}_{SKIP} P \rangle$
proof (*unfold deadlock-free_{SKIP}-def failure-refine-def, safe*)
from *assm* **have** *** : $\langle UNIV \notin \mathcal{R} P \rangle$

```

    ⟨e ∈ ready-set P ∧ e ≠ ✓ ⟹
      {(tl s, X) | s X. (s, X) ∈ F P ∧ s ≠ [] ∧ hd s = e} ⊆ F (DFSKIP UNIV)⟩
  for e
  by (simp-all add: deadlock-freeSKIP-def failure-refine-def F-AfterExt non-BOT)

  fix s X
  assume **: ⟨(s, X) ∈ F P⟩
  show ⟨(s, X) ∈ F (DFSKIP UNIV)⟩
  proof (cases s)
    show ⟨s = [] ⟹ (s, X) ∈ F (DFSKIP UNIV)⟩
      by (subst F-DFSKIP, simp)
        (metis *(1) ** UNIV-eq-I event.exhaust Refusals-iff)
  next
    fix e s'
    assume *** : ⟨s = e # s'⟩
    show ⟨(s, X) ∈ F (DFSKIP UNIV)⟩
    proof (cases e)
      fix e'
      assume ⟨e = ev e'⟩
      with ** *** Cons-in-T-imp-elem-ready-set F-T
      have ⟨e ∈ ready-set P ∧ e ≠ ✓⟩ by blast
      with *(2)[OF this] show ⟨(s, X) ∈ F (DFSKIP UNIV)⟩
        by (subst F-DFSKIP, simp add: subset-iff)
          (metis ** *** ⟨e = ev e'⟩ list.distinct(1) list.sel(1))
    next
      assume ⟨e = ✓⟩
      hence ⟨s = [✓]⟩
        by (metis ** *** F-T append-butlast-last-id append-single-T-imp-tickFree
          butlast.simps(2) list.distinct(1) tickFree-Cons)
      thus ⟨(s, X) ∈ F (DFSKIP UNIV)⟩
        by (subst F-DFSKIP, simp)
    qed
  qed
qed
qed
end

```

Chapter 11

Generic Operational Semantics as a Locale, bis

```
theory OpSemGenericBis
  imports AfterTraceBis HOL-CSPM.DeadlockResults
begin
```

11.1 Definition

```
locale OpSemGeneric =
  fixes  $\tau$ -trans :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool' (infixl  $\rightsquigarrow_\tau$  50)
  assumes  $\tau$ -trans-NdetL:  $\langle P \sqcap Q \rightsquigarrow_\tau P \rangle$ 
    and  $\tau$ -trans-transitivity:  $\langle P \rightsquigarrow_\tau Q \Longrightarrow Q \rightsquigarrow_\tau R \Longrightarrow P \rightsquigarrow_\tau R \rangle$ 
    and  $\tau$ -trans-anti-mono-ready-set:  $\langle P \rightsquigarrow_\tau Q \Longrightarrow \text{ready-set } Q \subseteq \text{ready-set } P \rangle$ 
    and  $\tau$ -trans-mono-AfterExt:
       $\langle e \in \text{ready-set } Q \Longrightarrow P \rightsquigarrow_\tau Q \Longrightarrow P \text{ afterExt } e \rightsquigarrow_\tau Q \text{ afterExt } e \rangle$ 
begin
```

This locale needs to be instantiated with a binary relation (\rightsquigarrow_τ) which:

- is compatible with (\sqcap)
- is transitive
- makes *ready-set* anti-monotonic
- makes *AfterExt* monotonic.

From the τ transition $P \rightsquigarrow_\tau Q$ we derive the event transition as follows:

```
abbreviation event-trans :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  event  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool' ( $\dashv$ -/  
 $\rightsquigarrow_\tau$ -/ $\rightarrow$  [50, 3, 51] 50)  
  where  $\langle P \rightsquigarrow_e Q \equiv e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow_\tau Q \rangle$ 
```

From idempotence, commutativity and \perp absorbance of (\sqcap), we get the following for free.

lemma τ -trans-eg: $\langle P \rightsquigarrow_{\tau} P \rangle$
and τ -trans-NdetR: $\langle P \sqcap Q \rightsquigarrow_{\tau} Q \rangle$
and BOT- τ -trans-anything: $\langle \perp \rightsquigarrow_{\tau} P \rangle$
and BOT-ev-trans-anything: $\langle \perp \rightsquigarrow (ev\ e)\ P \rangle$
by (metis Ndet-id τ -trans-NdetL,
metis Ndet-commute τ -trans-NdetL,
metis Ndet-BOT τ -trans-NdetL,
metis AfterExt-BOT Ndet-BOT UNIV-I τ -trans-NdetL event.simps(3) ready-set-BOT)

As immediate consequences of the axioms, we prove that event transitions absorb τ transitions on right and on left.

lemma event-trans- τ -trans: $\langle P \rightsquigarrow e\ P' \implies P' \rightsquigarrow_{\tau} P'' \implies P \rightsquigarrow e\ P'' \rangle$
by (meson τ -trans-transitivity)

lemma τ -trans-event-trans: $\langle P \rightsquigarrow_{\tau} P' \implies P' \rightsquigarrow e\ P'' \implies P \rightsquigarrow e\ P'' \rangle$
using τ -trans-mono-AfterExt τ -trans-transitivity τ -trans-anti-mono-ready-set **by** blast

We can now define the concept of transition with a trace and demonstrate the first properties.

inductive trace-trans :: $\langle 'a\ process \implies 'a\ trace \implies 'a\ process \implies bool \rangle$ ($\langle - / \rightsquigarrow^* - / - \rangle$
[50, 3, 51] 50)

where trace- τ -trans : $\langle P \rightsquigarrow_{\tau} P' \implies P \rightsquigarrow^* \square P' \rangle$
| trace-tick-trans : $\langle P \rightsquigarrow \checkmark P' \implies P \rightsquigarrow^* [\checkmark] P' \rangle$
| trace-Cons-ev-trans :
 $\langle P \rightsquigarrow (ev\ e)\ P' \implies P' \rightsquigarrow^* s\ P'' \implies P \rightsquigarrow^* (ev\ e)\ \# s\ P'' \rangle$

lemma trace-trans- τ -trans: $\langle P \rightsquigarrow^* s\ P' \implies P' \rightsquigarrow_{\tau} P'' \implies P \rightsquigarrow^* s\ P'' \rangle$
apply (induct rule: trace-trans.induct)
subgoal using τ -trans-transitivity trace- τ -trans **by** blast
subgoal using event-trans- τ -trans trace-tick-trans **by** blast
using trace-Cons-ev-trans **by** blast

lemma τ -trans-trace-trans: $\langle P \rightsquigarrow_{\tau} P' \implies P' \rightsquigarrow^* s\ P'' \implies P \rightsquigarrow^* s\ P'' \rangle$
apply (rotate-tac, induct rule: trace-trans.induct)
subgoal using τ -trans-transitivity trace- τ -trans **by** blast
subgoal using τ -trans-event-trans trace-tick-trans **by** blast
using τ -trans-event-trans trace-Cons-ev-trans **by** blast

lemma tickFree-if-trans-trans-not-STOP :
 $\langle P \rightsquigarrow^* s\ Q \implies Q \neq STOP \implies tickFree\ s \rangle$
by (induct rule: trace-trans.induct, simp-all)
(metis τ -trans-anti-mono-ready-set bot.extremum-uniqueI
ready-set-AfterExt ready-set-empty-iff-STOP)

lemma *BOT-trace-trans-tickFree-anything* : $\langle \text{tickFree } s \implies \perp \rightsquigarrow^* s P \rangle$
proof (*induct s arbitrary: P*)
 show $\langle \bigwedge P. \text{tickFree } [] \implies \perp \rightsquigarrow^* [] P \rangle$
 by (*simp add: BOT- τ -trans-anything trace- τ -trans*)
next
 fix $e s P$
 assume *prem*: $\langle \text{tickFree } (e \# s) \rangle$ **and** *hyp*: $\langle \text{tickFree } s \implies \perp \rightsquigarrow^* s P \rangle$ **for** P
 have $*$: $\langle \text{tickFree } s \rangle$ **using** *prem* **by** *auto*
 obtain e' **where** $\langle e = \text{ev } e' \rangle$ **using** *event.exhaust prem* **by** *auto*
 thus $\langle \perp \rightsquigarrow^* e \# s P \rangle$
 by *simp (rule trace-Cons-ev-trans[OF - hyp]; simp add: * AfterExt-BOT BOT- τ -trans-anything ready-set-BOT)*
qed

11.2 Consequences of $P \rightsquigarrow^* s Q$ on \mathcal{F} , \mathcal{T} and \mathcal{D}

lemma *trace-trans-imp-F-if- τ -trans-imp-leF*:
 $\langle P \rightsquigarrow^* s Q \implies X \in \mathcal{R} Q \implies (s, X) \in \mathcal{F} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_F Q \rangle$
proof (*induct rule: trace-trans.induct*)
 show $\langle P \rightsquigarrow_\tau Q \implies X \in \mathcal{R} Q \implies ([], X) \in \mathcal{F} P \rangle$ **for** $P Q$
 by (*meson failure-refine-def in-mono Refusals-iff that*)
next
 show $\langle P \rightsquigarrow_{\checkmark} Q \implies X \in \mathcal{R} Q \implies ([\checkmark], X) \in \mathcal{F} P \rangle$ **for** $P Q$
 by (*metis append-Nil mem-Collect-eq ready-set-def tick-T-F*)
next
 fix $P e Q s Q'$
 assume $*$: $\langle P \rightsquigarrow (ev e) Q \rangle$ $\langle X \in \mathcal{R} Q' \implies (s, X) \in \mathcal{F} Q \rangle$ $\langle X \in \mathcal{R} Q' \rangle$
 have $\langle P \text{ afterExt } ev e \sqsubseteq_F Q \rangle$ **using** $*(1)$ **that** **by** *blast*
 hence $\langle (s, X) \in \mathcal{F} (P \text{ afterExt } ev e) \rangle$ **by** (*simp add: failure-refine-def subsetD*
 $*(2, 3)$)
 thus $\langle (ev e \# s, X) \in \mathcal{F} P \rangle$ **by** (*simp add: F-AfterExt *(1)*) (*metis list.exhaust-sel*)
qed

lemma *trace-trans-imp-T-if- τ -trans-imp-leT*: $\langle P \rightsquigarrow^* s Q \implies s \in \mathcal{T} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_T Q \rangle$
proof (*induct rule: trace-trans.induct*)
 show $\langle P \rightsquigarrow_\tau Q \implies [] \in \mathcal{T} P \rangle$ **for** $P Q$
 by (*simp add: Nil-lem-T*)
next
 show $\langle P \rightsquigarrow_{\checkmark} Q \implies [\checkmark] \in \mathcal{T} P \rangle$ **for** $P Q$
 by (*simp add: ready-set-def*)
next
 fix $P e Q s Q'$
 assume $*$: $\langle P \rightsquigarrow (ev e) Q \rangle$ $\langle s \in \mathcal{T} Q \rangle$
 have $\langle P \text{ afterExt } ev e \sqsubseteq_T Q \rangle$ **using** $*(1)$ **that** **by** *blast*
 hence $\langle s \in \mathcal{T} (P \text{ afterExt } ev e) \rangle$ **by** (*simp add: *(2) subsetD trace-refine-def*)
 with $*(1)$ *list.collapse* **show** $\langle ev e \# s \in \mathcal{T} P \rangle$

by (force simp add: T-AfterExt ready-set-def)
qed

lemma trace-trans-BOT-imp-D-if- τ -trans-imp-leD: $\langle P \rightsquigarrow^* s \perp \implies s \in \mathcal{D} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_D Q \rangle$
proof (induct s arbitrary: P)
show $\langle \bigwedge P. P \rightsquigarrow^* \perp \implies \perp \in \mathcal{D} P \rangle$
by (subst (asm) trace-trans.simps, auto)
(meson BOT-iff-D divergence-refine-def subsetD that)
next
fix e s P
assume prem : $\langle P \rightsquigarrow^* e \# s \perp \rangle$
assume hyp: $\langle P \rightsquigarrow^* s \perp \implies s \in \mathcal{D} P \rangle$ **for** P
have $\langle P \text{ afterExt } e \rightsquigarrow^* s \perp \rangle$
using prem **by** (cases rule: trace-trans.cases)
(auto simp add: trace- τ -trans intro: τ -trans-trace-trans)
from hyp[OF this] **show** $\langle e \# s \in \mathcal{D} P \rangle$
by (auto simp add: D-AfterExt D-UU split: if-split-asm)
qed

11.3 Characterizations for $P \rightsquigarrow^* s Q$

The following results require a lot of work, but will be very useful.

lemma trace-trans-iff :
 $\langle P \rightsquigarrow^* \perp Q \longleftrightarrow P \rightsquigarrow_{\tau} Q \rangle$
 $\langle P \rightsquigarrow^* [\checkmark] Q \longleftrightarrow P \rightsquigarrow \checkmark Q \rangle$
 $\langle P \rightsquigarrow^* (ev e) \# s Q' \longleftrightarrow (\exists Q. P \rightsquigarrow (ev e) Q \wedge Q \rightsquigarrow^* s Q') \rangle$
 $\langle tickFree s \implies (P \rightsquigarrow^* s @ [f] Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow f Q') \rangle$
 $\langle front-tickFree (s @ t) \implies (P \rightsquigarrow^* s @ t Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t Q') \rangle$
proof –
show f1 : $\langle \bigwedge P Q. P \rightsquigarrow^* \perp Q \longleftrightarrow P \rightsquigarrow_{\tau} Q \rangle$
and f2 : $\langle \bigwedge P Q. P \rightsquigarrow^* [\checkmark] Q \longleftrightarrow P \rightsquigarrow \checkmark Q \rangle$
and f3 : $\langle \bigwedge P Q' e. P \rightsquigarrow^* (ev e) \# s Q' \longleftrightarrow (\exists Q. P \rightsquigarrow (ev e) Q \wedge Q \rightsquigarrow^* s Q') \rangle$
by ((subst trace-trans.simps, auto)[1])+

show f4 : $\langle tickFree s \implies (P \rightsquigarrow^* s @ [f] Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow f Q') \rangle$
for s f P Q'
proof safe
show $\langle P \rightsquigarrow^* s @ [f] Q' \implies \exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow f Q' \rangle$
proof (induct s arbitrary: P)
case Nil
thus ?case
apply (subst (asm) trace-trans.simps, simp)
using τ -trans-eq τ -trans-transitivity f1 **by** blast
next
case (Cons e s)
from Cons.prem **have** * : $\langle e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow^* s @ [f] Q' \rangle$

```

    by (subst (asm) trace-trans.simps)
      (auto simp add: intro:  $\tau$ -trans-trace-trans)
  with Cons.hyps obtain Q where **:  $\langle P \text{ afterExt } e \rightsquigarrow^* s Q \rangle \langle Q \rightsquigarrow f Q' \rangle$  by
blast
  have  $\langle P \rightsquigarrow^* e \# s Q \rangle$ 
  proof (cases e)
    fix e'
    assume  $\langle e = \text{ev } e' \rangle$ 
    thus  $\langle P \rightsquigarrow^* e \# s Q \rangle$ 
    apply simp
    by (rule trace-Cons-ev-trans[OF - ** (1)]) (use *  $\tau$ -trans-eq in blast)
  next
  from Cons.prem1 have  $\langle e = \checkmark \implies s = [] \rangle$  by (subst (asm) trace-trans.simps)
  auto
    thus  $\langle e = \checkmark \implies P \rightsquigarrow^* e \# s Q \rangle$  using * ** (1) f1 f2 by blast
  qed
  with ** (2) show  $\langle \exists Q. P \rightsquigarrow^* e \# s Q \wedge Q \rightsquigarrow f Q' \rangle$  by blast
  qed
  next
  show  $\langle \text{tickFree } s \implies P \rightsquigarrow^* s Q \implies f \in \text{ready-set } Q \implies Q \text{ afterExt } f \rightsquigarrow_\tau Q' \rangle$ 
 $\implies$ 
     $P \rightsquigarrow^* s @ [f] Q'$  for Q
  proof (induct s arbitrary: P Q)
    show  $\langle \text{tickFree } [] \implies P \rightsquigarrow^* [] Q \implies f \in \text{ready-set } Q \implies Q \text{ afterExt } f \rightsquigarrow_\tau Q' \rangle$ 
 $Q' \implies$ 
     $P \rightsquigarrow^* [] @ [f] Q'$  for P Q
    by (simp add: f1)
    (metis (full-types)  $\tau$ -trans-eq  $\tau$ -trans-event-trans event.exhaust
      trace-Cons-ev-trans trace- $\tau$ -trans trace-tick-trans)
  next
  case (Cons e s)
  from Cons.prem2 have * :  $\langle e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow^* s Q \rangle$ 
  by (subst (asm) trace-trans.simps)
  (auto simp add: f1 intro:  $\tau$ -trans-trace-trans)
  show ?case
  proof (cases e)
    fix e'
    assume  $\langle e = \text{ev } e' \rangle$ 
    thus  $\langle P \rightsquigarrow^* (e \# s) @ [f] Q' \rangle$ 
    apply simp
    by (rule trace-Cons-ev-trans
      [OF - Cons.hyps[OF tickFree-tl[OF Cons.prem1], simplified]
      *[THEN conjunct2] Cons.prem3]])
    (use *  $\tau$ -trans-eq Cons.prem4 in  $\langle \text{blast} \rangle$ )
  next
  show  $\langle e = \checkmark \implies P \rightsquigarrow^* (e \# s) @ [f] Q' \rangle$  using Cons.prem1 by auto
  qed
  qed
  qed

```

```

show  $\langle \text{front-tickFree } (s @ t) \implies P \rightsquigarrow^* s @ t Q' \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t Q') \rangle$ 
proof (induct t arbitrary: Q' rule: rev-induct)
  show  $\langle P \rightsquigarrow^* s @ [] Q' \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* [] Q') \rangle$  for  $Q'$ 
    by (metis  $\tau$ -trans-eq append.right-neutral trace-trans- $\tau$ -trans f1)
  next
    case (snoc e t)
    show  $\langle P \rightsquigarrow^* s @ t @ [e] Q' \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t @ [e] Q') \rangle$ 
    proof (intro iffI)
      assume assm :  $\langle P \rightsquigarrow^* s @ t @ [e] Q' \rangle$ 
      from assm obtain  $Q$  where  $\langle P \rightsquigarrow^* s @ t Q \rangle$   $\langle Q \rightsquigarrow_e Q' \rangle$ 
      by (metis append.assoc f4 front-tickFree-implies-tickFree snoc.prems)
      also obtain  $R$  where  $\langle P \rightsquigarrow^* s R \rangle$   $\langle R \rightsquigarrow^* t Q \rangle$ 
      by (metis calculation(1) append.assoc front-tickFree-dw-closed snoc.hyps snoc.prems)
      ultimately show  $\langle \exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t @ [e] Q' \rangle$ 
      by (metis append-is-Nil-conv f4 front-tickFree-mono list.distinct(1) snoc.prems)
    next
      assume  $\langle \exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t @ [e] Q' \rangle$ 
      then obtain  $Q$  where  $\langle P \rightsquigarrow^* s Q \rangle$   $\langle Q \rightsquigarrow^* t @ [e] Q' \rangle$  by blast
      also obtain  $R$  where  $\langle Q \rightsquigarrow^* t R \rangle$   $\langle R \rightsquigarrow_e Q' \rangle$ 
      by (metis append-is-Nil-conv calculation(2) f4 front-tickFree-mono list.distinct(1) snoc.prems)
      ultimately show  $\langle P \rightsquigarrow^* s @ t @ [e] Q' \rangle$ 
      by (metis append-assoc f4 front-tickFree-implies-tickFree snoc.hyps snoc.prem tickFree-implies-front-tickFree)
    qed
  qed
qed

```

11.4 Finally: $P \rightsquigarrow^* s Q$ is P afterTrace $s \rightsquigarrow_\tau Q$

```

theorem T-imp-trace-trans-iff-AfterTrace- $\tau$ -trans :
   $\langle s \in \mathcal{T} P \implies (P \rightsquigarrow^* s Q) \longleftrightarrow P \text{ afterTrace } s \rightsquigarrow_\tau Q \rangle$ 
proof (intro iffI)
  show  $\langle P \rightsquigarrow^* s Q \implies s \in \mathcal{T} P \implies P \text{ afterTrace } s \rightsquigarrow_\tau Q \rangle$ 
  proof (induct s arbitrary: P Q rule: rev-induct)
    show  $\langle \bigwedge P Q. P \rightsquigarrow^* [] Q \implies [] \in \mathcal{T} P \implies P \text{ afterTrace } [] \rightsquigarrow_\tau Q \rangle$ 
    using trace-trans.cases by auto
  next
    fix  $s e P Q$ 
    assume hyp :  $\langle P \rightsquigarrow^* s Q \implies s \in \mathcal{T} P \implies P \text{ afterTrace } s \rightsquigarrow_\tau Q \rangle$  for  $P Q$ 
    assume prems :  $\langle P \rightsquigarrow^* s @ [e] Q \rangle$   $\langle s @ [e] \in \mathcal{T} P \rangle$ 
    have  $*$  :  $\langle e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$ 
    using prems(2) ready-set-AfterTrace by blast
    show  $\langle P \text{ afterTrace } (s @ [e]) \rightsquigarrow_\tau Q \rangle$ 
    by (metis AfterTrace-snoc  $\tau$ -trans-event-trans append-single-T-imp-tickFree hyp is-processT3-ST prems trace-trans-iff(4))

```



```

qed
next
show  $\langle P \text{ afterTrace } s \rightsquigarrow_{\tau} Q \implies s \in \mathcal{T} P \implies P \rightsquigarrow^* s Q \rangle$ 
proof (induct arbitrary: P Q rule: AfterTrace.induct)
  show  $\langle \bigwedge P Q. P \text{ afterTrace } [] \rightsquigarrow_{\tau} Q \implies [] \in \mathcal{T} P \implies P \rightsquigarrow^* [] Q \rangle$ 
    by (simp add: trace- $\tau$ -trans)
next
fix e and s ::  $\langle 'a \text{ trace} \rangle$  and Q P
assume hyp :  $\langle P \text{ afterTrace } s \rightsquigarrow_{\tau} Q \implies s \in \mathcal{T} P \implies P \rightsquigarrow^* s Q \rangle$  for P Q
assume prems :  $\langle P \text{ afterTrace } (e \# s) \rightsquigarrow_{\tau} Q \rangle \langle e \# s \in \mathcal{T} P \rangle$ 
have * :  $\langle e \in \text{ready-set } P \wedge s \in \mathcal{T} (P \text{ afterExt } e) \rangle$ 
  by (metis AfterTrace.simps(1, 2) Cons-in-T-imp-elem-ready-set
      T-AfterTrace append-Cons append-Nil prems(2))
show  $\langle P \rightsquigarrow^* e \# s Q \rangle$ 
proof (cases e)
  fix e'
  assume ** :  $\langle e = \text{ev } e' \rangle$ 
  from * ** have  $\langle P \rightsquigarrow (\text{ev } e') P \text{ afterExt } (\text{ev } e') \rangle$ 
    by (simp add:  $\tau$ -trans-eq)
  thus  $\langle P \rightsquigarrow^* e \# s Q \rangle$ 
    by (subst **, rule trace-Cons-ev-trans[OF - hyp[OF prems(1)][simplified]
        *[THEN conjunct2], simplified **]])
next
have  $\langle e = \checkmark \implies s = [] \rangle$  by (metis butlast.simps(2) front-tickFree-butlast
    is-processT2-TR tickFree-Cons prems(2))
thus  $\langle e = \checkmark \implies P \rightsquigarrow^* e \# s Q \rangle$ 
  using * prems(1) trace-tick-trans by force
qed
qed
qed

```

As corollaries we obtain the reciprocal results of

$\llbracket \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_F Q; ?P \rightsquigarrow^* ?s ?Q; ?X \in \mathcal{R} ?Q \rrbracket \implies (?s, ?X) \in \mathcal{F} ?P$

$\llbracket \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q; ?P \rightsquigarrow^* ?s ?Q \rrbracket \implies ?s \in \mathcal{T} ?P$

$\llbracket \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_D Q; ?P \rightsquigarrow^* ?s \perp \rrbracket \implies ?s \in \mathcal{D} ?P$

lemma *F-imp-exists-trace-trans*: $\langle (s, X) \in \mathcal{F} P \implies \exists Q. (P \rightsquigarrow^* s Q) \wedge X \in \mathcal{R} Q \rangle$
 by (meson F-T F-imp-refusal-AfterTrace Refusals-iff
 T-imp-trace-trans-iff-AfterTrace- τ -trans τ -trans-eq)

lemma *T-imp-exists-trace-trans*: $\langle s \in \mathcal{T} P \implies \exists Q. P \rightsquigarrow^* s Q \rangle$
 using *F-imp-exists-trace-trans* T-F by blast

lemma *D-imp-trace-trans-BOT*: $\langle \text{tickFree } s \implies s \in \mathcal{D} P \implies P \rightsquigarrow^* s \perp \rangle$
 by (simp add: D-T D-imp-AfterTrace-is-BOT
 T-imp-trace-trans-iff-AfterTrace- τ -trans τ -trans-eq)

When we have more information on $P \rightsquigarrow_{\tau} Q$, we obtain:

lemma τ -trans-imp-leT-imp-STOP-trace-trans-iff:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow STOP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = STOP \rangle$
using *STOP-T-iff* **by** (*subst trace-trans.simps*)
(auto simp add: ready-set-STOP τ -trans-eq)

lemma τ -trans-imp-leF-imp-SKIP-trace-trans-iff:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_F Q \Longrightarrow$
 $SKIP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle$
using *SKIP-F-iff* *STOP-F-iff* **by** (*subst trace-trans.simps*)
(auto simp add: AfterExt-SKIP ready-set-SKIP τ -trans-eq)

lemma τ -trans-imp-leT-imp-trace-trans-ready-set-subset-ready-set-AfterTrace:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow P \rightsquigarrow^* s Q \Longrightarrow$
 $ready\text{-set } Q \subseteq ready\text{-set } (P \text{ afterTrace } s) \rangle$
using *T-imp-trace-trans-iff-AfterTrace- τ -trans*
anti-mono-ready-set-T trace-trans-imp-T-if- τ -trans-imp-leT **by** *blast*

lemma τ -trans-imp-leT-imp-trace-trans-imp-ready-set:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow P \rightsquigarrow^* (s @ e \# t) Q \Longrightarrow$
 $e \in ready\text{-set } (P \text{ afterTrace } s) \rangle$
using *ready-set-AfterTrace trace-trans-imp-T-if- τ -trans-imp-leT* **by** *blast*

lemma *trace-trans-iff-T-and-AfterTrace- τ -trans-if- τ -trans-imp-leT*:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow$
 $(P \rightsquigarrow^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \rightsquigarrow_{\tau} Q \rangle$
using *T-imp-trace-trans-iff-AfterTrace- τ -trans* *trace-trans-imp-T-if- τ -trans-imp-leT*
by *blast*

11.5 General Rules of Operational Semantics

Some rules of operational semantics are consequences of *OpSemGeneric*'s axioms without needing to specify more what $P \rightsquigarrow_{\tau} Q$ is.

lemma *SKIP-trans-tick*: $\langle SKIP \rightsquigarrow \checkmark STOP \rangle$
by (*simp add: AfterExt-SKIP τ -trans-eq ready-set-SKIP*)

lemma *tick-trans-imp-BOT-L-or-STOP-R*: $\langle P \rightsquigarrow \checkmark Q \Longrightarrow P = \perp \vee Q = STOP \rangle$
by (*metis τ -trans-anti-mono-ready-set ready-set-AfterExt ready-set-empty-iff-STOP subset-empty*)

lemma *STOP-trace-trans-iff* : $\langle STOP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = STOP \rangle$
by (*metis AfterExt-SKIP SKIP-neq-BOT SKIP-trans-tick empty-iff trace-trans.cases ready-set-STOP tick-trans-imp-BOT-L-or-STOP-R trace-trans-iff(1)*)

lemma *ready-tick-imp- τ -trans-SKIP*: $\langle P \rightsquigarrow_{\tau} SKIP \rangle$ **if** $\langle \checkmark \in \text{ready-set } P \rangle$
proof –
from *that have* $\langle P \sqsubseteq_{FD} SKIP \rangle$
unfolding *failure-divergence-refine-def le-ref-def*
by (*auto simp add: F-SKIP D-SKIP subset-iff ready-set-def is-processT6-S2*)
(*metis append-Nil tick-T-F*)
then obtain Q **where** $\langle P = Q \sqcap SKIP \rangle$
by (*metis mono-Ndet-FD mono-Ndet-FD-left FD-antisym Ndet-id idem-FD*)
thus $\langle P \rightsquigarrow_{\tau} SKIP \rangle$ **by** (*simp add: τ -trans-NdetR*)
qed

lemma *exists-tick-trans-is-ready-tick*: $\langle (\exists P'. P \rightsquigarrow_{\checkmark} P') \longleftrightarrow \checkmark \in \text{ready-set } P \rangle$
using τ -trans-eq **by** *blast*

lemma *tick-trans-iff* : $\langle P \rightsquigarrow_{\checkmark} P' \longleftrightarrow P \rightsquigarrow_{\tau} SKIP \wedge P' = STOP \rangle$
by (*metis AfterExt-BOT AfterExt-SKIP SKIP-neq-BOT SKIP-trans-tick*)
(*τ -trans-event-trans ready-tick-imp- τ -trans-SKIP tick-trans-imp-BOT-L-or-STOP-R*)

lemma *SKIP-cant-ev-trans*: $\langle \neg SKIP \rightsquigarrow (ev\ e)\ STOP \rangle$
by (*simp add: ready-set-SKIP*)

lemma *STOP-cant-event-trans*: $\langle \neg STOP \rightsquigarrow_e P \rangle$
by (*simp add: ready-set-STOP*)

lemma *ev-trans-Mprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P\ a \rightsquigarrow (ev\ e)\ (P\ e) \rangle$
by (*simp add: AfterExt-def After-Mprefix τ -trans-eq ready-set-Mprefix*)

lemma *ev-trans-Mndetprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P\ a \rightsquigarrow (ev\ e)\ (P\ e) \rangle$
by (*simp add: AfterExt-def After-Mndetprefix τ -trans-eq ready-set-Mndetprefix*)

lemma *ev-trans-prefix*: $\langle e \rightarrow P \rightsquigarrow (ev\ e)\ P \rangle$
by (*metis ev-trans-Mprefix insertI1 write0-def*)

lemma *τ -trans-MultiNdet*: $\langle \text{finite } A \implies x \in A \implies (\Box a \in A. P\ a) \rightsquigarrow_{\tau} P\ x \rangle$
by (*metis MultiNdet-insert' τ -trans-NdetL emptyE insert-absorb*)

lemma *τ -trans-GlobalNdet*: $\langle (\Box a \in A. P\ a) \rightsquigarrow_{\tau} P\ e \rangle$ **if** $\langle e \in A \rangle$
proof –
have $\langle (\Box a \in A. P\ a) = P\ e \sqcap (\Box a \in A. P\ a) \rangle$

by (metis GlobalNdet-factorization-union GlobalNdet-unit
empty-iff insertI1 insert-absorb insert-is-Un that)
thus $\langle (\prod a \in A. P a) \rightsquigarrow_{\tau} P e \rangle$ by (metis τ -trans-NdetL)
qed

lemma *fix-point- τ -trans*: $\langle cont f \implies P = (\mu X. f X) \implies P \rightsquigarrow_{\tau} f P \rangle$
by (metis τ -trans-eq cont-process-rec)

lemma *event-trans-DetL*: $\langle P \rightsquigarrow_e P' \implies P \square Q \rightsquigarrow_e P' \rangle$
by (metis AfterExt-Det-is-AfterExt-Ndet Un-iff τ -trans-NdetL τ -trans-event-trans
ready-set-Det)

lemma *event-trans-DetR*: $\langle Q \rightsquigarrow_e Q' \implies P \square Q \rightsquigarrow_e Q' \rangle$
by (metis Det-commute event-trans-DetL)

lemma *event-trans-MultiDet*:
 $\langle finite A \implies a \in A \implies P a \rightsquigarrow_e Q \implies (\prod a \in A. P a) \rightsquigarrow_e Q \rangle$
by (metis MultiDet-insert' event-trans-DetL insert-absorb)

lemma *Sliding-event-transL*: $\langle P \rightsquigarrow_e P' \implies P \triangleright Q \rightsquigarrow_e P' \rangle$
unfolding *Sliding-def*
apply (drule event-trans-DetL[of e P P' Q])
using τ -trans-NdetL τ -trans-event-trans by blast

lemma *Sliding- τ -transR*: $\langle P \triangleright Q \rightsquigarrow_{\tau} Q \rangle$
unfolding *Sliding-def* by (simp add: τ -trans-NdetR)

lemma $\langle \exists P P' Q. P \rightsquigarrow_{\checkmark} P' \wedge \neg P ; Q \rightsquigarrow_{\checkmark} P' ; Q \rangle$
by (metis SKIP-Seq SKIP-trans-tick STOP-cant-event-trans)

lemma *ev-trans-SeqR*:
 $\langle \checkmark \in ready-set P \implies Q \rightsquigarrow (ev e) Q' \implies P ; Q \rightsquigarrow (ev e) Q' \rangle$
by (simp add: AfterExt-Seq ready-set-Seq AfterExt-BOT BOT-Seq)
(metis Ndet-commute τ -trans-NdetL τ -trans-transitivity)

lemma $\langle SKIP \llbracket S \rrbracket SKIP \rightsquigarrow_{\checkmark} STOP \rangle$

by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma $\langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \rightsquigarrow_{\tau} \text{SKIP} \rangle$
by (*simp add: Sync-SKIP-SKIP τ -trans-eq*)

lemma *tick-trans-Hiding*: $\langle P \setminus B \rightsquigarrow_{\checkmark} \text{STOP} \rangle$ **if** $\langle P \rightsquigarrow_{\checkmark} P' \rangle$
proof –
have $\langle (P \setminus B) \text{afterExt } \checkmark = \text{STOP} \sqcap (P \setminus B) \text{afterExt } \checkmark \rangle$
by (*simp add: AfterExt-def*)
thus $\langle P \setminus B \rightsquigarrow_{\checkmark} \text{STOP} \rangle$
by (*simp add: AfterExt-def τ -trans-eq ready-tick-imp-ready-tick-Hiding that*)
qed

The following lemma is useless since the locale mechanism forces f to be of type $'\alpha \Rightarrow '\alpha$ while it could be $'\alpha \Rightarrow '\beta$. We will have to prove it again on each instantiation.

lemma $\langle \text{Renaming } P f \rightsquigarrow_{\checkmark} \text{STOP} \rangle$ **if** $\langle P \rightsquigarrow_{\checkmark} P' \rangle$
proof –
have $\langle \text{Renaming } P f \text{afterExt } \checkmark = \text{STOP} \sqcap \text{Renaming } P f \text{afterExt } \checkmark \rangle$
by (*metis τ -trans-eq not-ready-AfterExt tick-trans-iff*)
thus $\langle \text{Renaming } P f \rightsquigarrow_{\checkmark} \text{STOP} \rangle$
by (*simp add: that tick-eq-EvExt AfterExt-def τ -trans-eq ready-set-Renaming BOT- τ -trans-anything*)
qed

end

end

Chapter 12

Failure Divergence Operational Semantics, bis

theory *OpSemFDBis*
 imports *OpSemGenericBis HOL-Library.LaTeXsugar*
begin

As announced in the motivations, the first definition we want to try is with (\sqsubseteq_{FD}) .

abbreviation $\tau\text{-trans} :: \langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$ (**infixl** $\langle_{FD} \rightsquigarrow_{\tau} \rangle$ 50)
 where $\langle P \text{ }_{FD} \rightsquigarrow_{\tau} Q \equiv P \sqsubseteq_{FD} Q \rangle$

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGenericBis*.

interpretation *OpSemGeneric* $\langle_{(FD} \rightsquigarrow_{\tau})} \rangle$
 using *trans-FD* **by** *unfold-locales*
 (auto simp add: anti-mono-ready-set-FD mono-AfterExt-FD)

notation *event-trans* $\langle \langle - \rangle_{FD} \rightsquigarrow - \rangle \rightarrow [50, 3, 51] 50$

notation *trace-trans* $\langle \langle - \rangle_{FD} \rightsquigarrow^* - \rangle \rightarrow [50, 3, 51] 50$

lemma $\langle P \text{ }_{FD} \rightsquigarrow e P' \Longrightarrow P' \text{ }_{FD} \rightsquigarrow_{\tau} P'' \Longrightarrow P \text{ }_{FD} \rightsquigarrow e P'' \rangle$
 $\langle P \text{ }_{FD} \rightsquigarrow_{\tau} P' \Longrightarrow P' \text{ }_{FD} \rightsquigarrow e P'' \Longrightarrow P \text{ }_{FD} \rightsquigarrow e P'' \rangle$
 by *(fact event-trans- τ -trans τ -trans-event-trans)+*

12.1 Operational Semantics Laws

SKIP law

lemma $\langle \text{SKIP} \text{ }_{FD} \rightsquigarrow \checkmark \text{STOP} \rangle$
 by *(fact SKIP-trans-tick)*

$e \rightarrow P$ laws

lemma $\langle e \in A \Longrightarrow \square a \in A \rightarrow P a \text{ }_{FD} \rightsquigarrow (ev e) (P e) \rangle$

by (*fact ev-trans-Mprefix*)

lemma $\langle e \in A \implies \sqcap a \in A \rightarrow P a \text{ }_{FD \rightsquigarrow} (ev\ e) (P\ e) \rangle$
by (*fact ev-trans-Mndetprefix*)

lemma $\langle e \rightarrow P \text{ }_{FD \rightsquigarrow} (ev\ e) P \rangle$
by (*fact ev-trans-prefix*)

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow} P \rangle$
and $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow} Q \rangle$
by (*fact τ -trans-NdetL τ -trans-NdetR*)₊

lemma $\langle a \in A \implies (\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} P a \rangle$
by (*fact τ -trans-GlobalNdet*)

lemma $\langle \text{finite } A \implies a \in A \implies (\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} P a \rangle$
by (*fact τ -trans-MultiNdet*)

$\mu x. f x$ law

lemma $\langle \text{cont } f \implies P = (\mu X. f X) \implies P \text{ }_{FD \rightsquigarrow} f P \rangle$
by (*fact fix-point- τ -trans*)

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \text{ }_{FD \rightsquigarrow} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \text{ }_{FD \rightsquigarrow} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P \sqcap Q' \rangle$
by *simp-all*

lemma τ -trans-MultiDet:
 $\langle \text{finite } A \implies \forall a \in A. P a \text{ }_{FD \rightsquigarrow} P' a \implies$
 $(\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} (\sqcap a \in A. P' a) \rangle$
by (*fact mono-MultiDet-FD*)

lemma $\langle P \text{ }_{FD \rightsquigarrow} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P' \sqcap Q \rangle$
and $\langle Q \text{ }_{FD \rightsquigarrow} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow} P \sqcap Q' \rangle$
by (*fact event-trans-DetL event-trans-DetR*)₊

lemma $\langle \text{finite } A \implies a \in A \implies P a \text{ }_{FD \rightsquigarrow} Q \implies (\sqcap a \in A. P a) \text{ }_{FD \rightsquigarrow} Q \rangle$
by (*fact event-trans-MultiDet*)

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{FD \rightsquigarrow} P' \implies P ; Q \text{ }_{FD \rightsquigarrow} P' ; Q \rangle$
by *simp*

lemma *ev-trans-SeqL*: $\langle P \text{ }_{FD \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{FD \rightsquigarrow} (ev\ e) P' ; Q \rangle$
by (*auto simp add: ready-set-Seq AfterExt-Seq*)

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ FD}\rightsquigarrow\checkmark P' \implies P ; Q \text{ FD}\rightsquigarrow_{\tau} Q \rangle$
by (*metis mono-Seq-FD SKIP-Seq τ -trans-eq ready-tick-imp- τ -trans-SKIP*)

lemma $\checkmark \in \text{ready-set } P \implies Q \text{ FD}\rightsquigarrow(\text{ev } e) Q' \implies P ; Q \text{ FD}\rightsquigarrow(\text{ev } e) Q'$

by (*fact ev-trans-SeqR*)

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ FD}\rightsquigarrow_{\tau} P' \implies P \setminus B \text{ FD}\rightsquigarrow_{\tau} P' \setminus B \rangle$
by (*fact mono-Hiding-FD*)

lemma *ev-trans-Hiding-notin*:

$\langle P \text{ FD}\rightsquigarrow(\text{ev } e) P' \implies e \notin B \implies P \setminus B \text{ FD}\rightsquigarrow(\text{ev } e) P' \setminus B \rangle$

by (*metis AfterExt-def After-Hiding-FD-Hiding-After-if-ready-notin mono-Hiding-FD*)

event-trans- τ -trans event.simps(4) ready-notin-imp-ready-Hiding)

lemma $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies P \setminus B \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$

by (*fact tick-trans-Hiding*)

lemma *ev-trans-Hiding-inside*:

$\langle P \text{ FD}\rightsquigarrow(\text{ev } e) P' \implies e \in B \implies P \setminus B \text{ FD}\rightsquigarrow_{\tau} P' \setminus B \rangle$

by (*metis AfterExt-def Hiding-FD-Hiding-After-if-ready-inside mono-Hiding-FD event.simps(4) trans-FD*)

Renaming P f laws

lemma τ -trans-Renaming: $\langle P \text{ FD}\rightsquigarrow_{\tau} P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow_{\tau} \text{Renaming } P' f \rangle$

by (*fact mono-Renaming-FD*)

lemma *tick-trans-Renaming*: $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$

by (*simp add: AfterExt-def ready-set-Renaming tick-eq-EvExt*)

lemma *ev-trans-Renaming*:

$\langle f a = b \implies P \text{ FD}\rightsquigarrow(\text{ev } a) P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow(\text{ev } b) (\text{Renaming } P' f) \rangle$

apply (*simp add: AfterExt-Renaming Renaming-BOT ready-set-BOT ready-set-Renaming*)

apply (*intro conjI impI*)

apply (*meson ev-elem-anteced1 imageI vimageI2*)

apply (*rule τ -trans-transitivity[of - $\langle \text{Renaming } (P \text{ afterExt } \text{ev } a) f \rangle]$*)

apply (*solves $\langle \text{rule } \tau$ -trans-GlobalNdet, simp \rangle*)

by (*simp add: τ -trans-Renaming*)

lemma $\langle P \text{ FD}\rightsquigarrow_{\tau} P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow_{\tau} P' \llbracket a := b \rrbracket \rangle$

by (*fact τ -trans-Renaming*)

lemma $\langle P \xrightarrow{FD} \checkmark P' \implies P \llbracket a := b \rrbracket \xrightarrow{FD} \checkmark STOP \rangle$
by (*fact tick-trans-Renaming*)

lemma *ev-trans-RenamingF*:
 $\langle P \xrightarrow{FD} (ev\ a)\ P' \implies P \llbracket a := b \rrbracket \xrightarrow{FD} (ev\ b)\ P' \llbracket a := b \rrbracket \rangle$
by (*metis ev-trans-Renaming fun-upd-same*)

$P \llbracket S \rrbracket Q$ laws

lemma τ -*trans-SyncL*: $\langle P \xrightarrow{FD} \tau P' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau P' \llbracket S \rrbracket Q \rangle$
and τ -*trans-SyncR*: $\langle Q \xrightarrow{FD} \tau Q' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau P \llbracket S \rrbracket Q' \rangle$
by *simp-all*

lemma *ev-trans-SyncL*:
 $\langle e \notin S \implies P \xrightarrow{FD} (ev\ e)\ P' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} (ev\ e)\ P' \llbracket S \rrbracket Q \rangle$
and *ev-trans-SyncR*:
 $\langle e \notin S \implies Q \xrightarrow{FD} (ev\ e)\ Q' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} (ev\ e)\ P \llbracket S \rrbracket Q' \rangle$
by (*simp-all add: AfterExt-Sync ready-set-Sync image-iff*)

lemma *ev-trans-SyncLR*:
 $\langle \llbracket e \in S; P \xrightarrow{FD} (ev\ e)\ P'; Q \xrightarrow{FD} (ev\ e)\ Q' \rrbracket \implies P \llbracket S \rrbracket Q \xrightarrow{FD} (ev\ e)\ P' \llbracket S \rrbracket Q' \rangle$
by (*simp add: AfterExt-Sync ready-set-Sync*)

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

lemma *tick-trans-SyncL*: $\langle P \xrightarrow{FD} \checkmark P' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \xrightarrow{FD} \checkmark Q' \implies P \llbracket S \rrbracket Q \xrightarrow{FD} \tau P \llbracket S \rrbracket SKIP \rangle$
by (*simp-all add: ready-tick-imp- τ -trans-SKIP*)

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{FD} \checkmark STOP \rangle$
by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma τ -*trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{FD} \tau SKIP \rangle$
by (*simp add: Sync-SKIP-SKIP*)

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \xrightarrow{FD} \tau P' \implies P \triangleright Q \xrightarrow{FD} \tau P' \triangleright Q \rangle$
unfolding *Sliding-def* **by** *simp*

lemma $\langle P \xrightarrow{FD} e P' \implies P \triangleright Q \xrightarrow{FD} e P' \rangle$
by (*fact Sliding-event-transL*)

lemma $\langle P \triangleright Q \xrightarrow{FD} \tau Q \rangle$
by (*fact Sliding- τ -transR*)

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \xrightarrow{FD} \tau P' \implies P \triangle Q \xrightarrow{FD} \tau P' \triangle Q \rangle$

by (simp add: mono-Interrupt-FD)

lemma *Interrupt- τ -trans-right*: $\langle Q \text{ }_{FD \rightsquigarrow \tau} Q' \implies P \Delta Q \text{ }_{FD \rightsquigarrow \tau} P \Delta Q' \rangle$
 by (simp add: mono-Interrupt-FD)

lemma *Interrupt-ev-trans-left*:
 $\langle P \text{ }_{FD \rightsquigarrow} (ev \ e) P' \implies P \Delta Q \text{ }_{FD \rightsquigarrow} (ev \ e) P' \Delta Q \rangle$
 by (simp add: AfterExt-def After-Interrupt Interrupt- τ -trans-left ready-set-Interrupt)

lemma *Interrupt-ev-trans-right*: $\langle Q \text{ }_{FD \rightsquigarrow} (ev \ e) Q' \implies P \Delta Q \text{ }_{FD \rightsquigarrow} (ev \ e) Q' \rangle$
 by (simp add: AfterExt-def After-Interrupt ready-set-Interrupt)

Throw $P \ A \ Q$ laws

lemma *Throw- τ -trans-left*:
 $\langle P \text{ }_{FD \rightsquigarrow \tau} P' \implies P \ \Theta \ a \in A. \ Q \ a \text{ }_{FD \rightsquigarrow \tau} P' \ \Theta \ a \in A. \ Q \ a \rangle$
 by (simp add: mono-Throw-FD)

lemma *Throw- τ -trans-right*:
 $\langle \forall a \in A. \ Q \ a \text{ }_{FD \rightsquigarrow \tau} Q' \ a \implies P \ \Theta \ a \in A. \ Q \ a \text{ }_{FD \rightsquigarrow \tau} P \ \Theta \ a \in A. \ Q' \ a \rangle$
 by (simp add: mono-Throw-FD)

lemma *Throw-event-trans-left*:
 $\langle P \text{ }_{FD \rightsquigarrow} e P' \implies e \notin ev \ A \implies P \ \Theta \ a \in A. \ Q \ a \text{ }_{FD \rightsquigarrow} e (P' \ \Theta \ a \in A. \ Q \ a) \rangle$
 apply (simp add: AfterExt-Throw ready-set-Throw image-iff split: event.split)
 apply (intro conjI impI)
 by (metis AfterExt-def Throw- τ -trans-left event.simps(4))
 (solves \langle simp add: Throw-STOP tick-trans-iff \rangle)

lemma *Throw-ev-trans-right*:
 $\langle P \text{ }_{FD \rightsquigarrow} (ev \ e) P' \implies e \in A \implies P \ \Theta \ a \in A. \ Q \ a \text{ }_{FD \rightsquigarrow} (ev \ e) (Q \ e) \rangle$
 by (simp add: AfterExt-Throw ready-set-Throw split: event.split)

lemma $\langle tickFree \ s \implies \perp \text{ }_{FD \rightsquigarrow^*} s \ P \rangle$
 by (fact BOT-trace-trans-tickFree-anything)

12.2 Reality Checks

lemma $\langle STOP \text{ }_{FD \rightsquigarrow^*} s \ P \longleftrightarrow s = [] \wedge P = STOP \rangle$
 by (fact STOP-trace-trans-iff)

lemma *SKIP-trace-trans-iff* :
 $\langle SKIP \text{ }_{FD \rightsquigarrow^*} s \ P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle$
 by (simp add: τ -trans-imp-leF-imp-SKIP-trace-trans-iff leFD-imp-leF)

lemma *F-iff-exists-trans* :
 $\langle (s, X) \in \mathcal{F} P \longleftrightarrow (\exists P'. (P \text{ FD}\rightsquigarrow^* s P') \wedge X \in \mathcal{R} P') \rangle$
using *F-imp-exists-trace-trans leFD-imp-leF trace-trans-imp-F-if-τ-trans-imp-leF*
by *blast*

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \text{ FD}\rightsquigarrow^* s P') \rangle$
by (*meson T-imp-exists-trace-trans leFD-imp-leF leF-imp-leT trace-trans-imp-T-if-τ-trans-imp-leT*)

lemma *tickFree-imp-D-iff-trace-trans-BOT*:
 $\langle \text{tickFree } s \implies s \in \mathcal{D} P \longleftrightarrow P \text{ FD}\rightsquigarrow^* s \perp \rangle$
using *D-imp-trace-trans-BOT leFD-imp-leD trace-trans-BOT-imp-D-if-τ-trans-imp-leD*
by *blast*

lemma *D-iff-trace-trans-BOT*:
 $\langle s \in \mathcal{D} P \longleftrightarrow (\text{if tickFree } s \text{ then } P \text{ FD}\rightsquigarrow^* s \perp \text{ else } P \text{ FD}\rightsquigarrow^* (\text{butlast } s) \perp) \rangle$
by (*metis tickFree-if-trans-trans-not-STOP STOP-neq-BOT*
append-butlast-last-id front-tickFree-butlast
front-tickFree-single is-processT tickFree-butlast
tickFree-imp-D-iff-trace-trans-BOT)

12.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
 $\langle P \text{ FD}\rightsquigarrow^* s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$
by (*metis T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace-τ-trans τ-trans-anti-mono-ready-set*)

lemma *trace-trans-imp-ready-set*:
 $\langle P \text{ FD}\rightsquigarrow^* (s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
using *T-iff-exists-trans ready-set-AfterTrace* **by** *blast*

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
 $\langle (P \text{ FD}\rightsquigarrow^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \text{ FD}\rightsquigarrow_{\tau} Q \rangle$
using *T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace-τ-trans* **by** *blast*

lemma $\langle \text{deadlock-free } P \longleftrightarrow DF \text{ UNIV } \text{FD}\rightsquigarrow_{\tau} P \rangle$
by (*simp add: deadlock-free-def*)

lemma $\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow DF_{SKIP} \text{ UNIV } \text{FD}\rightsquigarrow_{\tau} P \rangle$
by (*fact deadlock-free_{SKIP}-FD*)

12.4 Summary: Operational Rules

In this section, we will just write down the operational laws that we have proven in a fancy way.

Absorbance rules

$$\frac{\frac{?P \text{ FD} \rightsquigarrow ?e \ ?P' \quad ?P' \text{ FD} \rightsquigarrow_{\tau} \ ?P''}{?P \text{ FD} \rightsquigarrow ?e \ ?P''}}{?P \text{ FD} \rightsquigarrow_{\tau} \ ?P' \quad ?P' \text{ FD} \rightsquigarrow ?e \ ?P''} \frac{}{?P \text{ FD} \rightsquigarrow ?e \ ?P''}$$

SKIP rule

$$\overline{\text{SKIP FD} \rightsquigarrow \checkmark \text{ STOP}}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \text{ FD} \rightsquigarrow \text{ev} \ ?e \ ?P \ ?e} \quad \frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \text{ FD} \rightsquigarrow \text{ev} \ ?e \ ?P \ ?e}$$

$$\overline{?e \rightarrow ?P \text{ FD} \rightsquigarrow \text{ev} \ ?e \ ?P}$$

(\Box) rules

$$\frac{\frac{?P \ \Box \ ?Q \text{ FD} \rightsquigarrow_{\tau} \ ?P \quad ?P \ \Box \ ?Q \text{ FD} \rightsquigarrow_{\tau} \ ?Q}{?e \in ?A}}{\Box a \in ?A. \ ?P \ a \text{ FD} \rightsquigarrow_{\tau} \ ?P \ ?e}$$

$\mu x. f x$ rule

$$\frac{\text{cont } ?f \quad ?P = (\mu x. \ ?f \ x)}{?P \text{ FD} \rightsquigarrow_{\tau} \ ?f \ ?P}$$

(\Box) rules

$$\frac{\frac{?P \text{ FD} \rightsquigarrow_{\tau} \ ?P'}{?P \ \Box \ ?Q \text{ FD} \rightsquigarrow_{\tau} \ ?P' \ \Box \ ?Q} \quad \frac{?Q \text{ FD} \rightsquigarrow_{\tau} \ ?Q'}{?P \ \Box \ ?Q \text{ FD} \rightsquigarrow_{\tau} \ ?P \ \Box \ ?Q'}}{\frac{?P \text{ FD} \rightsquigarrow ?e \ ?P'}{?P \ \Box \ ?Q \text{ FD} \rightsquigarrow ?e \ ?P'} \quad \frac{?Q \text{ FD} \rightsquigarrow ?e \ ?Q'}{?P \ \Box \ ?Q \text{ FD} \rightsquigarrow ?e \ ?Q'}}$$

(;) rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P ; ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' ; ?Q} \quad \frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P'}{?P ; ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?P' ; ?Q}}{\frac{\exists P'. ?P \text{ }_{FD \rightsquigarrow \checkmark} P'}{?P ; ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}}$$

Hiding rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \setminus ?B \text{ }_{FD \rightsquigarrow \tau} ?P' \setminus ?B} \quad \frac{?P \text{ }_{FD \rightsquigarrow \checkmark} ?P'}{?P \setminus ?B \text{ }_{FD \rightsquigarrow \checkmark} STOP}}{\frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P' \quad ?e \notin ?B}{?P \setminus ?B \text{ }_{FD \rightsquigarrow ev} ?e ?P' \setminus ?B}} \quad \frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P' \quad ?e \in ?B}{?P \setminus ?B \text{ }_{FD \rightsquigarrow \tau} ?P' \setminus ?B}}$$

Renaming rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{FD \rightsquigarrow \tau} \text{Renaming } ?P' \text{ } ?f} \quad \frac{?P \text{ }_{FD \rightsquigarrow \checkmark} ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{FD \rightsquigarrow \checkmark} STOP}}{\frac{?f \text{ } ?a = ?b \quad ?P \text{ }_{FD \rightsquigarrow ev} ?a ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{FD \rightsquigarrow ev} ?b \text{Renaming } ?P' \text{ } ?f}}$$

Sync rules

$$\frac{\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \text{ } [?S] \text{ } ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \text{ } [?S] \text{ } ?Q'}}{\frac{?e \notin ?S \quad ?P \text{ }_{FD \rightsquigarrow ev} ?e ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?P' \text{ } [?S] \text{ } ?Q} \quad \frac{?e \notin ?S \quad ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?P \text{ } [?S] \text{ } ?Q'}}{\frac{?e \in ?S \quad ?P \text{ }_{FD \rightsquigarrow ev} ?e ?P' \quad ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?P' \text{ } [?S] \text{ } ?Q'}} \quad \frac{?P \text{ }_{FD \rightsquigarrow \checkmark} ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow \tau} \text{SKIP} \text{ } [?S] \text{ } ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow \checkmark} ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \text{ } [?S] \text{ } \text{SKIP}}}$$

$$\overline{SKIP \llbracket ?S \rrbracket SKIP \text{ }_{FD \rightsquigarrow \tau} SKIP}$$

(▷) rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \triangleright ?Q} \quad \frac{?P \text{ }_{FD \rightsquigarrow ?e} ?P'}{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow ?e} ?P'}}{\frac{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}}$$

(△) rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \triangle ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \triangle ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}{?P \triangle ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \triangle ?Q'}}{\frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P'}{?P \triangle ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?P' \triangle ?Q}} \quad \frac{\frac{?Q \text{ }_{FD \rightsquigarrow ev} ?e ?Q'}{?P \triangle ?Q \text{ }_{FD \rightsquigarrow ev} ?e ?Q'}}$$

Throw rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \ominus a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow \tau} ?P' \ominus a \in ?A. ?Q a} \quad \frac{\forall a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow \tau} ?Q' a}{?P \ominus a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow \tau} ?P \ominus a \in ?A. ?Q' a}}{\frac{?P \text{ }_{FD \rightsquigarrow ?e} ?P' \quad ?e \notin ev \text{ } ?A}{?P \ominus a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow ?e} ?P' \ominus a \in ?A. ?Q a} \quad \frac{?P \text{ }_{FD \rightsquigarrow ev} ?e ?P' \quad ?e \in ?A}{?P \ominus a \in ?A. ?Q a \text{ }_{FD \rightsquigarrow ev} ?e ?Q ?e}}$$

end

Chapter 13

Trace Divergence Operational Semantics, bis

theory *OpSemDTBis*
 imports *OpSemGenericBis HOL-Library.LaTeXsugar*
begin

The definition with (\sqsubseteq_{FD}) motivates us to try with other refinements.

abbreviation $\tau\text{-trans} :: \langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$ (**infixl** $\langle_{DT\rightsquigarrow\tau} \rangle$ 50)
 where $\langle P \text{ }_{DT\rightsquigarrow\tau} Q \equiv P \sqsubseteq_{DT} Q \rangle$

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGenericBis*.

interpretation *OpSemGeneric* $\langle_{(DT\rightsquigarrow\tau)} \rangle$
 using *trans-DT* **by** *unfold-locales*
 (*auto simp add: anti-mono-ready-set-DT mono-AfterExt-DT*)

notation *event-trans* ($\langle_{- /_{DT\rightsquigarrow} - /} \rightarrow$ [50, 3, 51] 50)

notation *trace-trans* ($\langle_{- /_{DT\rightsquigarrow^*} - /} \rightarrow$ [50, 3, 51] 50)

lemma $\langle P \text{ }_{DT\rightsquigarrow} e P' \Longrightarrow P' \text{ }_{DT\rightsquigarrow\tau} P'' \Longrightarrow P \text{ }_{DT\rightsquigarrow} e P'' \rangle$
 $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \Longrightarrow P' \text{ }_{DT\rightsquigarrow} e P'' \Longrightarrow P \text{ }_{DT\rightsquigarrow} e P'' \rangle$
 by (*fact event-trans- τ -trans τ -trans-event-trans*) $+$

13.1 Operational Semantics Laws

SKIP law

lemma $\langle \text{SKIP} \text{ }_{DT\rightsquigarrow} \checkmark \text{ STOP} \rangle$
 by (*fact SKIP-trans-tick*)

$e \rightarrow P$ laws

lemma $\langle e \in A \Longrightarrow \square a \in A \rightarrow P a \text{ }_{DT\rightsquigarrow} (ev e) (P e) \rangle$
 by (*fact ev-trans-Mprefix*)

lemma $\langle e \in A \implies \sqcap a \in A \rightarrow P a \text{ }_{DT \rightsquigarrow} (ev\ e) (P\ e) \rangle$
by (*fact ev-trans-Mndetprefix*)

lemma $\langle e \rightarrow P \text{ }_{DT \rightsquigarrow} (ev\ e) P \rangle$
by (*fact ev-trans-prefix*)

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
by (*fact τ -trans-NdetL τ -trans-NdetR*) $+$

lemma $\langle a \in A \implies (\sqcap a \in A. P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
by (*fact τ -trans-GlobalNdet*)

lemma $\langle \text{finite } A \implies a \in A \implies (\sqcap a \in A. P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
by (*fact τ -trans-MultiNdet*)

$\mu\ x. f\ x$ law

lemma $\langle \text{cont } f \implies P = (\mu\ X. f\ X) \implies P \text{ }_{DT \rightsquigarrow \tau} f\ P \rangle$
by (*fact fix-point- τ -trans*)

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \text{ }_{DT \rightsquigarrow \tau} Q' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \sqcap Q' \rangle$
by *simp-all*

lemma τ -trans-MultiDet:
 $\langle \text{finite } A \implies \forall a \in A. P\ a \text{ }_{DT \rightsquigarrow \tau} P'\ a \implies$
 $(\sqcap a \in A. P\ a) \text{ }_{DT \rightsquigarrow \tau} (\sqcap a \in A. P'\ a) \rangle$
by (*fact mono-MultiDet-DT*)

lemma $\langle P \text{ }_{DT \rightsquigarrow e} P' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow e} P' \rangle$
and $\langle Q \text{ }_{DT \rightsquigarrow e} Q' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow e} Q' \rangle$
by (*fact event-trans-DetL event-trans-DetR*) $+$

lemma $\langle \text{finite } A \implies a \in A \implies P\ a \text{ }_{DT \rightsquigarrow e} Q \implies (\sqcap a \in A. P\ a) \text{ }_{DT \rightsquigarrow e} Q \rangle$
by (*fact event-trans-MultiDet*)

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} P' ; Q \rangle$
by *simp*

lemma *ev-trans-SeqL*: $\langle P \text{ }_{DT \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev\ e) P' ; Q \rangle$
by (*auto simp add: ready-set-Seq AfterExt-Seq*)

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ }_{DT\rightsquigarrow\checkmark} P' \implies P ; Q \text{ }_{DT\rightsquigarrow\tau} Q \rangle$
by (*metis mono-Seq-DT SKIP-Seq τ -trans-eq ready-tick-imp- τ -trans-SKIP*)

lemma $\checkmark \in \text{ready-set } P \implies Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q' \implies P ; Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q'$
by (*fact ev-trans-SeqR*)

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \setminus B \text{ }_{DT\rightsquigarrow\tau} P' \setminus B \rangle$
by (*fact mono-Hiding-DT*)

lemma *ev-trans-Hiding-notin*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \implies e \notin B \implies P \setminus B \text{ }_{DT\rightsquigarrow}(ev\ e) P' \setminus B \rangle$
by (*metis AfterExt-def After-Hiding-DT-Hiding-After-if-ready-notin mono-Hiding-DT*
event-trans- τ -trans event.simps(4) ready-notin-imp-ready-Hiding)

lemma $\langle P \text{ }_{DT\rightsquigarrow\checkmark} P' \implies P \setminus B \text{ }_{DT\rightsquigarrow\checkmark} STOP \rangle$
by (*fact tick-trans-Hiding*)

lemma *ev-trans-Hiding-inside*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \implies e \in B \implies P \setminus B \text{ }_{DT\rightsquigarrow\tau} P' \setminus B \rangle$
by (*metis AfterExt-def Hiding-DT-Hiding-After-if-ready-inside*
mono-Hiding-DT event.simps(4) trans-DT)

Renaming P f laws

lemma τ -trans-Renaming:
 $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow\tau} \text{Renaming } P'\ f \rangle$
by (*fact mono-Renaming-DT*)

lemma *tick-trans-Renaming*: $\langle P \text{ }_{DT\rightsquigarrow\checkmark} P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow\checkmark} STOP \rangle$
by (*simp add: AfterExt-def ready-set-Renaming tick-eq-EvExt*)

lemma *ev-trans-Renaming*:
 $\langle f\ a = b \implies P \text{ }_{DT\rightsquigarrow}(ev\ a) P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow}(ev\ b) (\text{Renaming } P'\ f) \rangle$
apply (*simp add: AfterExt-Renaming Renaming-BOT ready-set-BOT ready-set-Renaming*)
apply (*intro conjI impI*)
apply (*meson ev-elem-anteced1 imageI vimageI2*)
apply (*rule τ -trans-transitivity[of - $\langle \text{Renaming } (P\ \text{afterExt } ev\ a)\ f \rangle$]*)
apply (*solves $\langle \text{rule } \tau\text{-trans-GlobalNdet, simp} \rangle$*)
by (*simp add: τ -trans-Renaming*)

lemma $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow\tau} P' \llbracket a := b \rrbracket \rangle$
by (*fact τ -trans-Renaming*)

lemma $\langle P \xrightarrow{DT} \checkmark P' \implies P \llbracket a := b \rrbracket \xrightarrow{DT} \checkmark STOP \rangle$
by (*fact tick-trans-Renaming*)

lemma *ev-trans-RenamingF*:
 $\langle P \xrightarrow{DT} (ev\ a)\ P' \implies P \llbracket a := b \rrbracket \xrightarrow{DT} (ev\ b)\ P' \llbracket a := b \rrbracket \rangle$
by (*metis ev-trans-Renaming fun-upd-same*)

$P \llbracket S \rrbracket Q$ laws

lemma τ -*trans-SyncL*: $\langle P \xrightarrow{DT} \tau P' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} \tau P' \llbracket S \rrbracket Q \rangle$
and τ -*trans-SyncR*: $\langle Q \xrightarrow{DT} \tau Q' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} \tau P \llbracket S \rrbracket Q' \rangle$
by *simp-all*

lemma *ev-trans-SyncL*:
 $\langle e \notin S \implies P \xrightarrow{DT} (ev\ e)\ P' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} (ev\ e)\ P' \llbracket S \rrbracket Q \rangle$
and *ev-trans-SyncR*:
 $\langle e \notin S \implies Q \xrightarrow{DT} (ev\ e)\ Q' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} (ev\ e)\ P \llbracket S \rrbracket Q' \rangle$
by (*simp-all add: AfterExt-Sync ready-set-Sync image-iff*)

lemma *ev-trans-SyncLR*:
 $\langle \llbracket e \in S; P \xrightarrow{DT} (ev\ e)\ P'; Q \xrightarrow{DT} (ev\ e)\ Q' \rrbracket \implies P \llbracket S \rrbracket Q \xrightarrow{DT} (ev\ e)\ P' \llbracket S \rrbracket Q' \rangle$
by (*simp add: AfterExt-Sync ready-set-Sync*)

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

lemma *tick-trans-SyncL*: $\langle P \xrightarrow{DT} \checkmark P' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} \tau SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \xrightarrow{DT} \checkmark Q' \implies P \llbracket S \rrbracket Q \xrightarrow{DT} \tau P \llbracket S \rrbracket SKIP \rangle$
by (*simp-all add: ready-tick-imp- τ -trans-SKIP*)

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{DT} \checkmark STOP \rangle$
by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma τ -*trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{DT} \tau SKIP \rangle$
by (*simp add: Sync-SKIP-SKIP*)

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \xrightarrow{DT} \tau P' \implies P \triangleright Q \xrightarrow{DT} \tau P' \triangleright Q \rangle$
unfolding *Sliding-def* **by** *simp*

lemma $\langle P \xrightarrow{DT} e P' \implies P \triangleright Q \xrightarrow{DT} e P' \rangle$
by (*fact Sliding-event-transL*)

lemma $\langle P \triangleright Q \xrightarrow{DT} \tau Q \rangle$
by (*fact Sliding- τ -transR*)

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \xrightarrow{DT} \tau P' \implies P \triangle Q \xrightarrow{DT} \tau P' \triangle Q \rangle$
by (*simp add: mono-Interrupt-DT*)

lemma *Interrupt- τ -trans-right*: $\langle Q \text{ }_{DT\rightsquigarrow\tau} Q' \implies P \Delta Q \text{ }_{DT\rightsquigarrow\tau} P \Delta Q' \rangle$
by (*simp add: mono-Interrupt-DT*)

lemma *Interrupt-ev-trans-left*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \implies P \Delta Q \text{ }_{DT\rightsquigarrow}(ev\ e) P' \Delta Q \rangle$
by (*simp add: AfterExt-def After-Interrupt Interrupt- τ -trans-left ready-set-Interrupt*)

lemma *Interrupt-ev-trans-right*: $\langle Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q' \implies P \Delta Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q' \rangle$
by (*simp add: AfterExt-def After-Interrupt ready-set-Interrupt*)

Throw P A Q laws

lemma *Throw- τ -trans-left*:
 $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \Theta a \in A. Q\ a \text{ }_{DT\rightsquigarrow\tau} P' \Theta a \in A. Q\ a \rangle$
by (*simp add: mono-Throw-DT*)

lemma *Throw- τ -trans-right*:
 $\langle \forall a \in A. Q\ a \text{ }_{DT\rightsquigarrow\tau} Q' a \implies P \Theta a \in A. Q\ a \text{ }_{DT\rightsquigarrow\tau} P \Theta a \in A. Q' a \rangle$
by (*simp add: mono-Throw-DT*)

lemma *Throw-event-trans-left*:
 $\langle P \text{ }_{DT\rightsquigarrow}e P' \implies e \notin ev\ 'A \implies P \Theta a \in A. Q\ a \text{ }_{DT\rightsquigarrow}e (P' \Theta a \in A. Q\ a) \rangle$
apply (*simp add: AfterExt-Throw ready-set-Throw image-iff split: event.split*)
apply (*intro conjI impI*)
by (*metis AfterExt-def Throw- τ -trans-left event.simps(4)*)
(solves $\langle simp\ add: Throw-STOP\ tick-trans-iff \rangle$)

lemma *Throw-ev-trans-right*:
 $\langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \implies e \in A \implies P \Theta a \in A. Q\ a \text{ }_{DT\rightsquigarrow}(ev\ e) (Q\ e) \rangle$
by (*simp add: AfterExt-Throw ready-set-Throw split: event.split*)

lemma $\langle tickFree\ s \implies \perp \text{ }_{DT\rightsquigarrow^*} s P \rangle$
by (*fact BOT-trace-trans-tickFree-anything*)

13.2 Reality Checks

lemma $\langle STOP \text{ }_{DT\rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = STOP \rangle$
by (*fact STOP-trace-trans-iff*)

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T}\ P \longleftrightarrow (\exists P'. P \text{ }_{DT\rightsquigarrow^*} s P') \rangle$

using *T-imp-exists-trace-trans leDT-imp-leT trace-trans-imp-T-if-τ-trans-imp-leT*
by *blast*

lemma *tickFree-imp-D-iff-trace-trans-BOT*:

$\langle \text{tickFree } s \implies s \in \mathcal{D} P \iff P \text{ }_{DT\rightsquigarrow^*} s \perp \rangle$

using *D-imp-trace-trans-BOT leDT-imp-leD trace-trans-BOT-imp-D-if-τ-trans-imp-leD*
by *blast*

lemma *D-iff-trace-trans-BOT*:

$\langle s \in \mathcal{D} P \iff (\text{if tickFree } s \text{ then } P \text{ }_{DT\rightsquigarrow^*} s \perp \text{ else } P \text{ }_{DT\rightsquigarrow^*} (\text{butlast } s) \perp) \rangle$

by (*metis tickFree-if-trans-trans-not-STOP STOP-neq-BOT*
append-butlast-last-id front-tickFree-butlast
front-tickFree-single is-processT tickFree-butlast
tickFree-imp-D-iff-trace-trans-BOT)

13.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:

$\langle P \text{ }_{DT\rightsquigarrow^*} s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$

by (*metis T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace-τ-trans τ-trans-anti-mono-ready-set*)

lemma *trace-trans-imp-ready-set*:

$\langle P \text{ }_{DT\rightsquigarrow^*} (s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$

using *T-iff-exists-trans ready-set-AfterTrace* **by** *blast*

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :

$\langle (P \text{ }_{DT\rightsquigarrow^*} s Q) \iff s \in \mathcal{T} P \wedge P \text{ afterTrace } s \text{ }_{DT\rightsquigarrow\tau} Q \rangle$

using *T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace-τ-trans* **by** *blast*

13.4 Summary: Operational Rules

In this section, we will just write down the operational laws that we have proven in a fancy way.

Absorbance rules

$$\frac{\frac{?P \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P' \quad ?P' \text{ }_{DT\rightsquigarrow\tau} ?P''}{?P \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P''}}{?P \text{ }_{DT\rightsquigarrow\tau} ?P' \quad ?P' \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P''}{?P \text{ }_{DT\rightsquigarrow} ?e \text{ } ?P''}$$

SKIP rule

$$\frac{}{SKIP \text{ }_{DT\rightsquigarrow} \checkmark STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \ a \ DT \rightsquigarrow \text{ev} \ ?e \ ?P \ ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \ a \ DT \rightsquigarrow \text{ev} \ ?e \ ?P \ ?e}$$

$$\frac{}{?e \rightarrow ?P \ DT \rightsquigarrow \text{ev} \ ?e \ ?P}$$

(\square) rules

$$\frac{\frac{}{?P \ \square \ ?Q \ DT \rightsquigarrow_{\tau} \ ?P} \quad \frac{}{?P \ \square \ ?Q \ DT \rightsquigarrow_{\tau} \ ?Q}}{?e \in ?A}}{\square a \in ?A. \ ?P \ a \ DT \rightsquigarrow_{\tau} \ ?P \ ?e}$$

$\mu x. f x$ rule

$$\frac{\text{cont } ?f \quad ?P = (\mu x. ?f x)}{?P \ DT \rightsquigarrow_{\tau} \ ?f \ ?P}$$

(\square) rules (more powerful than in OpSemFD)

$$\frac{\frac{}{?P \ DT \rightsquigarrow_{\tau} \ ?P'}}{?P \ \square \ ?Q \ DT \rightsquigarrow_{\tau} \ ?P' \ \square \ ?Q} \quad \frac{\frac{}{?Q \ DT \rightsquigarrow_{\tau} \ ?Q'}}{?P \ \square \ ?Q \ DT \rightsquigarrow_{\tau} \ ?P \ \square \ ?Q'}$$

(;) rules

$$\frac{\frac{}{?P \ DT \rightsquigarrow_{\tau} \ ?P'}}{?P \ ; \ ?Q \ DT \rightsquigarrow_{\tau} \ ?P' \ ; \ ?Q} \quad \frac{\frac{}{?P \ DT \rightsquigarrow \text{ev} \ ?e \ ?P'}}{?P \ ; \ ?Q \ DT \rightsquigarrow \text{ev} \ ?e \ ?P' \ ; \ ?Q}}{\frac{\exists P'. ?P \ DT \rightsquigarrow \checkmark P'}{?P \ ; \ ?Q \ DT \rightsquigarrow_{\tau} \ ?Q}}$$

Hiding rules

$$\frac{\frac{}{?P \ DT \rightsquigarrow_{\tau} \ ?P'}}{?P \ \setminus \ ?B \ DT \rightsquigarrow_{\tau} \ ?P' \ \setminus \ ?B} \quad \frac{\frac{}{?P \ DT \rightsquigarrow \checkmark \ ?P'}}{?P \ \setminus \ ?B \ DT \rightsquigarrow \checkmark \ \text{STOP}}}{\frac{?P \ DT \rightsquigarrow \text{ev} \ ?e \ ?P' \quad ?e \notin ?B}{?P \ \setminus \ ?B \ DT \rightsquigarrow \text{ev} \ ?e \ ?P' \ \setminus \ ?B} \quad \frac{?P \ DT \rightsquigarrow \text{ev} \ ?e \ ?P' \quad ?e \in ?B}{?P \ \setminus \ ?B \ DT \rightsquigarrow_{\tau} \ ?P' \ \setminus \ ?B}}$$

Renaming rules

$$\begin{array}{c}
\frac{\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{DT \rightsquigarrow \tau} \text{Renaming } ?P' \text{ } ?f}}{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}}{\text{Renaming } ?P \text{ } ?f \text{ }_{DT \rightsquigarrow \checkmark} \text{STOP}} \\
\frac{?f \text{ } ?a = ?b \quad ?P \text{ }_{DT \rightsquigarrow ev} ?a \text{ } ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{DT \rightsquigarrow ev} ?b \text{ } \text{Renaming } ?P' \text{ } ?f}
\end{array}$$

Sync rules

$$\begin{array}{c}
\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \text{ } [?S] \text{ } ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \text{ } [?S] \text{ } ?Q'}}{?e \notin ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?P'} \\
\frac{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?P' \text{ } [?S] \text{ } ?Q \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?Q'}{?e \notin ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?P' \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?Q'} \\
\frac{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?P \text{ } [?S] \text{ } ?Q'}{?e \in ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?P' \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?Q'} \\
\frac{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow ev} ?e \text{ } ?P' \text{ } [?S] \text{ } ?Q'}{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'} \\
\frac{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} \text{SKIP} \text{ } [?S] \text{ } ?Q \quad ?Q \text{ }_{DT \rightsquigarrow \checkmark} ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \text{ } [?S] \text{ } \text{SKIP}} \\
\frac{\text{SKIP} \text{ } [?S] \text{ } \text{SKIP} \text{ }_{DT \rightsquigarrow \tau} \text{SKIP}}{}
\end{array}$$

(▷) rules

$$\begin{array}{c}
\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \text{ } \triangleright ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \text{ } \triangleright ?Q} \quad \frac{?P \text{ }_{DT \rightsquigarrow ?e} ?P'}{?P \text{ } \triangleright ?Q \text{ }_{DT \rightsquigarrow ?e} ?P'} \\
\frac{}{?P \text{ } \triangleright ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q}
\end{array}$$

(△) rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \text{ } \triangle ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \text{ } \triangle ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \text{ } \triangle ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \text{ } \triangle ?Q'}$$

$$\frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P \Delta ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \Delta ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \Delta ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}$$

Throw rules

$$\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?P' \Theta a \in ?A. ?Q a \quad \forall a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?Q' a}}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?P \Theta a \in ?A. ?Q' a \quad ?P \text{ }_{DT \rightsquigarrow ?e} ?P' \quad ?e \notin ev \text{ ' } ?A}}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow ?e} ?P' \Theta a \in ?A. ?Q a \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?e \in ?A}}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow ev} ?e ?Q ?e}$$

end

Chapter 14

Failure Operational Semantics, bis

```
theory OpSemFBis
  imports OpSemGenericBis HOL-Library.LaTeXsugar
begin
```

The definition with (\sqsubseteq_{FD}) motivates us to try with other refinements.

```
abbreviation  $\tau$ -trans :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool' (infixl  $\langle F \rightsquigarrow_{\tau} \rangle$  50)
  where  $\langle P \ F \rightsquigarrow_{\tau} \ Q \equiv P \ \sqsubseteq_F \ Q \rangle$ 
```

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGenericBis*.

```
interpretation OpSemGeneric  $\langle (F \rightsquigarrow_{\tau}) :: '\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool'>
```

```
proof unfold-locales
```

```
  show  $\langle P :: '\alpha$  process  $\rangle \sqcap Q \ F \rightsquigarrow_{\tau} P \rangle$  for  $P \ Q$  by simp
```

```
next
```

```
  show  $\langle P :: '\alpha$  process  $\rangle F \rightsquigarrow_{\tau} Q \Longrightarrow Q \ F \rightsquigarrow_{\tau} R \Longrightarrow P \ F \rightsquigarrow_{\tau} R \rangle$  for  $P \ Q \ R$ 
    by (rule trans-F)
```

```
next
```

```
  show  $\langle P :: '\alpha$  process  $\rangle F \rightsquigarrow_{\tau} Q \Longrightarrow \text{ready-set } Q \subseteq \text{ready-set } P \rangle$  for  $P \ Q$ 
    by (simp add: anti-mono-ready-set-F)
```

```
next
```

```
  show  $\langle (e :: '\alpha$  event)  $\in$  ready-set  $Q \Longrightarrow P \ F \rightsquigarrow_{\tau} Q \Longrightarrow$ 
     $P \ \text{afterExt } e \ F \rightsquigarrow_{\tau} Q \ \text{afterExt } e \rangle$  for  $e \ P \ Q$ 
```

```
    by (auto simp add: AfterExt-def mono-After-F split: event.split)
```

```
qed
```

```
notation event-trans  $\langle - / F \rightsquigarrow - / - \rangle$  [50, 3, 51] 50)
```

```
notation trace-trans  $\langle - / F \rightsquigarrow^* - / - \rangle$  [50, 3, 51] 50)
```

```
lemma  $\langle P \ F \rightsquigarrow e \ P' \Longrightarrow P' \ F \rightsquigarrow_{\tau} P'' \Longrightarrow P \ F \rightsquigarrow e \ P'' \rangle$ 
   $\langle P \ F \rightsquigarrow_{\tau} P' \Longrightarrow P' \ F \rightsquigarrow e \ P'' \Longrightarrow P \ F \rightsquigarrow e \ P'' \rangle$ 
  by (fact event-trans- $\tau$ -trans  $\tau$ -trans-event-trans)+
```

14.1 Operational Semantics Laws

SKIP law

lemma $\langle \text{SKIP} \xrightarrow{F} \checkmark \text{STOP} \rangle$
by (*fact SKIP-trans-tick*)

$e \rightarrow P$ laws

lemma $\langle e \in A \implies \Box a \in A \rightarrow P a \xrightarrow{F} (ev\ e) (P\ e) \rangle$
by (*fact ev-trans-Mprefix*)

lemma $\langle e \in A \implies \Box a \in A \rightarrow P a \xrightarrow{F} (ev\ e) (P\ e) \rangle$
by (*fact ev-trans-Mndetprefix*)

lemma $\langle e \rightarrow P \xrightarrow{F} (ev\ e) P \rangle$
by (*fact ev-trans-prefix*)

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \xrightarrow{F} \tau P \rangle$
and $\langle P \sqcap Q \xrightarrow{F} \tau Q \rangle$
by (*fact τ -trans-NdetL τ -trans-NdetR*) $+$

lemma $\langle a \in A \implies (\Box a \in A. P\ a) \xrightarrow{F} \tau P a \rangle$
by (*fact τ -trans-GlobalNdet*)

lemma $\langle \text{finite } A \implies a \in A \implies (\Box a \in A. P\ a) \xrightarrow{F} \tau P a \rangle$
by (*fact τ -trans-MultiNdet*)

$\mu x. f\ x$ law

lemma $\langle \text{cont } f \implies P = (\mu X. f\ X) \implies P \xrightarrow{F} \tau f\ P \rangle$
by (*fact fix-point- τ -trans*)

$P \sqcap Q$ laws

lemma $\langle P \xrightarrow{F} e P' \implies P \sqcap Q \xrightarrow{F} e P' \rangle$
and $\langle Q \xrightarrow{F} e Q' \implies P \sqcap Q \xrightarrow{F} e Q' \rangle$
by (*fact event-trans-DetL event-trans-DetR*) $+$

lemma $\langle \text{finite } A \implies a \in A \implies P a \xrightarrow{F} e Q \implies (\Box a \in A. P\ a) \xrightarrow{F} e Q \rangle$
by (*fact event-trans-MultiDet*)

$P ; Q$ laws

lemma $\tau\text{-trans-SeqR}$: $\langle \exists P'. P \xrightarrow{F} \checkmark P' \implies P ; Q \xrightarrow{F} \tau Q \rangle$
apply (*elim exE conjE, drule ready-tick-imp- τ -trans-SKIP*)
by (*metis (no-types, lifting) CSP-Laws.mono-Seq-FD D-Seq SKIP-Seq divergence-refine-def*
dual-order.refl failure-divergence-refine-def leFD-imp-leF leF-leD-imp-leFD
le-sup-iff)

— not in the Roscoe's because direct consequence of $\exists P'. ?P \xrightarrow{F} \checkmark P' \implies ?P ; ?Q \xrightarrow{F} \checkmark ?Q$

lemma $\langle \checkmark \in \text{ready-set } P \implies Q \xrightarrow{F} (ev\ e) Q' \implies P ; Q \xrightarrow{F} (ev\ e) Q' \rangle$

by (*fact ev-trans-SeqR*)

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \xrightarrow{F} \checkmark P' \implies P \setminus B \xrightarrow{F} \checkmark P' \setminus B \rangle$

by (*fact mono-Hiding-F*)

lemma *ev-trans-Hiding-notin*:

$\langle P \xrightarrow{F} (ev\ e) P' \implies e \notin B \implies P \setminus B \xrightarrow{F} (ev\ e) P' \setminus B \rangle$

by (*metis AfterExt-def After-Hiding-F-Hiding-After-if-ready-notin mono-Hiding-F*

event-trans- τ -trans event.simps(4) ready-notin-imp-ready-Hiding)

lemma $\langle P \xrightarrow{F} \checkmark P' \implies P \setminus B \xrightarrow{F} \checkmark STOP \rangle$

by (*fact tick-trans-Hiding*)

lemma *ev-trans-Hiding-inside*:

$\langle P \xrightarrow{F} (ev\ e) P' \implies e \in B \implies P \setminus B \xrightarrow{F} \checkmark P' \setminus B \rangle$

by (*metis AfterExt-def Hiding-F-Hiding-After-if-ready-inside*

mono-Hiding-F event.simps(4) trans-F)

Renaming P f laws

lemma *tick-trans-Renaming*: $\langle P \xrightarrow{F} \checkmark P' \implies \text{Renaming } P\ f \xrightarrow{F} \checkmark STOP \rangle$

by (*simp add: AfterExt-def ready-set-Renaming tick-eq-EvExt*)

lemma $\langle P \xrightarrow{F} \checkmark P' \implies P \llbracket a := b \rrbracket \xrightarrow{F} \checkmark STOP \rangle$

using *tick-trans-Renaming by blast*

$P \llbracket S \rrbracket Q$ laws

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

lemma *tick-trans-SyncL*: $\langle P \xrightarrow{F} \checkmark P' \implies P \llbracket S \rrbracket Q \xrightarrow{F} \checkmark \text{SKIP} \llbracket S \rrbracket Q \rangle$

and *tick-trans-SyncR*: $\langle Q \xrightarrow{F} \checkmark Q' \implies P \llbracket S \rrbracket Q \xrightarrow{F} \checkmark P \llbracket S \rrbracket \text{SKIP} \rangle$

by (*simp-all add: D-SKIP ready-tick-imp- τ -trans-SKIP*

divergence-refine-def leFD-imp-leF leF-leD-imp-leFD)

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \xrightarrow{F} \checkmark STOP \rangle$

by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma τ -trans-SKIP-Sync-SKIP: $\langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \xrightarrow{F \rightsquigarrow \tau} \text{SKIP} \rangle$
by (*simp add: Sync-SKIP-SKIP*)

$P \triangleright Q$ laws

lemma $\langle P \xrightarrow{F \rightsquigarrow e} P' \implies P \triangleright Q \xrightarrow{F \rightsquigarrow e} P' \rangle$
by (*fact Sliding-event-transL*)

lemma $\langle P \triangleright Q \xrightarrow{F \rightsquigarrow \tau} Q \rangle$
by (*fact Sliding- τ -transR*)

$P \triangle Q$ laws

lemma *Interrupt-ev-trans-right*: $\langle Q \xrightarrow{F \rightsquigarrow} (ev\ e)\ Q' \implies P \triangle Q \xrightarrow{F \rightsquigarrow} (ev\ e)\ Q' \rangle$
by (*simp add: AfterExt-def After-Interrupt ready-set-Interrupt*)

Throw P A Q laws

lemma *Throw- τ -trans-right*:
 $\langle \forall a \in A. Q\ a \xrightarrow{F \rightsquigarrow \tau} Q'\ a \implies P\ \Theta\ a \in A. Q\ a \xrightarrow{F \rightsquigarrow \tau} P\ \Theta\ a \in A. Q'\ a \rangle$
by (*simp add: mono-right-Throw-F*)

lemma *Throw-ev-trans-right*:
 $\langle P \xrightarrow{F \rightsquigarrow} (ev\ e)\ P' \implies e \in A \implies P\ \Theta\ a \in A. Q\ a \xrightarrow{F \rightsquigarrow} (ev\ e)\ (Q\ e) \rangle$
by (*simp add: AfterExt-Throw ready-set-Throw*)

lemma $\langle \text{tickFree}\ s \implies \perp \xrightarrow{F \rightsquigarrow^*} s\ P \rangle$
by (*fact BOT-trace-trans-tickFree-anything*)

14.2 Reality Checks

lemma $\langle \text{STOP} \xrightarrow{F \rightsquigarrow^*} s\ P \longleftrightarrow s = [] \wedge P = \text{STOP} \rangle$
by (*fact STOP-trace-trans-iff*)

lemma *SKIP-trace-trans-iff* :
 $\langle \text{SKIP} \xrightarrow{F \rightsquigarrow^*} s\ P \longleftrightarrow s = [] \wedge P = \text{SKIP} \vee s = [\checkmark] \wedge P = \text{STOP} \rangle$
by (*simp add: τ -trans-imp-leF-imp-SKIP-trace-trans-iff*)

lemma *F-iff-exists-trans* :
 $\langle (s, X) \in \mathcal{F}\ P \longleftrightarrow (\exists P'. (P \xrightarrow{F \rightsquigarrow^*} s\ P') \wedge X \in \mathcal{R}\ P') \rangle$
using *F-imp-exists-trace-trans trace-trans-imp-F-if- τ -trans-imp-leF* **by** *blast*

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T}\ P \longleftrightarrow (\exists P'. P \xrightarrow{F \rightsquigarrow^*} s\ P') \rangle$
by (*meson T-imp-exists-trace-trans leF-imp-leT trace-trans-imp-T-if- τ -trans-imp-leT*)

14.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:

$$\langle P \xrightarrow{F}^* s \ Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$$

by (*metis T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace- τ -trans τ -trans-anti-mono-ready-set*)

lemma *trace-trans-imp-ready-set*:

$$\langle P \xrightarrow{F}^* (s \ @ \ e \ \# \ t) \ Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$$

using *T-iff-exists-trans ready-set-AfterTrace* **by** *blast*

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT* :

$$\langle (P \xrightarrow{F}^* s \ Q) \longleftrightarrow s \in \mathcal{T} \ P \ \wedge \ P \text{ afterTrace } s \xrightarrow{F}^*_{\tau} \ Q \rangle$$

using *T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace- τ -trans* **by** *blast*

lemma $\langle \text{deadlock-free } P \longleftrightarrow DF \ UNIV \xrightarrow{F}^*_{\tau} \ P \rangle$

by (*fact deadlock-free-F*)

lemma $\langle \text{deadlock-free}_{SKIP} \ P \longleftrightarrow DF_{SKIP} \ UNIV \xrightarrow{F}^*_{\tau} \ P \rangle$

by (*simp add: deadlock-free_{SKIP}-def*)

14.4 Nicely written operational rules

In this section, we will just write down the operational laws that we have proven in a fancy way.

Absorbance rules

$$\frac{?P \xrightarrow{F} ?e \ ?P' \quad ?P' \xrightarrow{F}^*_{\tau} ?P''}{?P \xrightarrow{F} ?e \ ?P''} \quad \frac{?P \xrightarrow{F}^*_{\tau} ?P' \quad ?P' \xrightarrow{F} ?e \ ?P''}{?P \xrightarrow{F} ?e \ ?P''}$$

SKIP rule

$$\overline{SKIP \xrightarrow{F} \checkmark \ STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \ \xrightarrow{F} \ ev \ ?e \ ?P \ ?e} \quad \frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \ \xrightarrow{F} \ ev \ ?e \ ?P \ ?e}$$

$$\overline{?e \rightarrow ?P \ \xrightarrow{F} \ ev \ ?e \ ?P}$$

(\sqcap) **rules**

$$\frac{\frac{\overline{?P \sqcap ?Q \text{ } F \rightsquigarrow_{\tau} ?P} \quad \overline{?P \sqcap ?Q \text{ } F \rightsquigarrow_{\tau} ?Q}}{?e \in ?A}}{\sqcap a \in ?A. ?P a \text{ } F \rightsquigarrow_{\tau} ?P ?e}$$

$\mu x. f x$ **rule**

$$\frac{\text{cont } ?f \quad ?P = (\mu x. ?f x)}{?P \text{ } F \rightsquigarrow_{\tau} ?f ?P}$$

(\sqcup) **rules**

$$\frac{?P \text{ } F \rightsquigarrow ?e ?P'}{?P \sqcup ?Q \text{ } F \rightsquigarrow ?e ?P'} \quad \frac{?Q \text{ } F \rightsquigarrow ?e ?Q'}{?P \sqcup ?Q \text{ } F \rightsquigarrow ?e ?Q'}$$

(;) **rule**

$$\frac{\exists P'. ?P \text{ } F \rightsquigarrow \checkmark P'}{?P ; ?Q \text{ } F \rightsquigarrow_{\tau} ?Q}$$

Hiding **rules**

$$\frac{\frac{?P \text{ } F \rightsquigarrow_{\tau} ?P'}{?P \setminus ?B \text{ } F \rightsquigarrow_{\tau} ?P' \setminus ?B}}{?P \text{ } F \rightsquigarrow_{ev} ?e ?P' \quad ?e \notin ?B}}{?P \setminus ?B \text{ } F \rightsquigarrow_{ev} ?e ?P' \setminus ?B} \quad \frac{\frac{?P \text{ } F \rightsquigarrow \checkmark ?P'}{?P \setminus ?B \text{ } F \rightsquigarrow \checkmark STOP}}{?P \text{ } F \rightsquigarrow_{ev} ?e ?P' \quad ?e \in ?B}}{?P \setminus ?B \text{ } F \rightsquigarrow_{\tau} ?P' \setminus ?B}$$

Renaming **rule**

$$\frac{?P \text{ } F \rightsquigarrow \checkmark ?P'}{\text{Renaming } ?P ?f \text{ } F \rightsquigarrow \checkmark STOP}$$

Sync rules

$$\frac{?P \xrightarrow{F} \checkmark ?P'}{?P \llbracket ?S \rrbracket ?Q \xrightarrow{F} \tau \text{ SKIP } \llbracket ?S \rrbracket ?Q} \quad \frac{?Q \xrightarrow{F} \checkmark ?Q'}{?P \llbracket ?S \rrbracket ?Q \xrightarrow{F} \tau ?P \llbracket ?S \rrbracket \text{ SKIP}}$$

$$\frac{}{\text{SKIP } \llbracket ?S \rrbracket \text{ SKIP } \xrightarrow{F} \tau \text{ SKIP}}$$

(▷) rules

$$\frac{?P \xrightarrow{F} ?e ?P'}{?P \triangleright ?Q \xrightarrow{F} ?e ?P'}$$

$$\frac{}{?P \triangleright ?Q \xrightarrow{F} \tau ?Q}$$

(△) rule

$$\frac{?Q \xrightarrow{F} ev ?e ?Q'}{?P \triangle ?Q \xrightarrow{F} ev ?e ?Q'}$$

Throw rules

$$\frac{\forall a \in ?A. ?Q a \xrightarrow{F} \tau ?Q' a}{?P \Theta a \in ?A. ?Q a \xrightarrow{F} \tau ?P \Theta a \in ?A. ?Q' a}$$

$$\frac{?P \xrightarrow{F} ev ?e ?P' \quad ?e \in ?A}{?P \Theta a \in ?A. ?Q a \xrightarrow{F} ev ?e ?Q ?e}$$

end

Chapter 15

Trace Operational Semantics, bis

```
theory OpSemTBis
  imports OpSemGenericBis HOL-Library.LaTeXsugar
begin
```

The definition with (\sqsubseteq_{FD}) motivates us to try with other refinements.

```
abbreviation  $\tau$ -trans ::  $\langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$  (infixl  $\langle T \rightsquigarrow_{\tau} \rangle$  50)
  where  $\langle P T \rightsquigarrow_{\tau} Q \equiv P \sqsubseteq_T Q \rangle$ 
```

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGenericBis*.

```
interpretation OpSemGeneric  $\langle (T \rightsquigarrow_{\tau}) \rangle$ 
  using trans-T by unfold-locales
  (auto simp add: anti-mono-ready-set-T mono-AfterExt-T)
```

```
notation event-trans ( $\langle - / T \rightsquigarrow - / - \rangle$  [50, 3, 51] 50)
```

```
notation trace-trans ( $\langle - / T \rightsquigarrow^* - / - \rangle$  [50, 3, 51] 50)
```

```
lemma  $\langle P T \rightsquigarrow e P' \Longrightarrow P' T \rightsquigarrow_{\tau} P'' \Longrightarrow P T \rightsquigarrow e P'' \rangle$ 
   $\langle P T \rightsquigarrow_{\tau} P' \Longrightarrow P' T \rightsquigarrow e P'' \Longrightarrow P T \rightsquigarrow e P'' \rangle$ 
  by (fact event-trans- $\tau$ -trans  $\tau$ -trans-event-trans)+
```

15.1 Operational Semantics Laws

SKIP law

```
lemma  $\langle \text{SKIP} T \rightsquigarrow \checkmark \text{STOP} \rangle$ 
  by (fact SKIP-trans-tick)
```

$e \rightarrow P$ laws

```
lemma  $\langle e \in A \Longrightarrow \Box a \in A \rightarrow P a T \rightsquigarrow (ev e) (P e) \rangle$ 
  by (fact ev-trans-Mprefix)
```

lemma $\langle e \in A \implies \Box a \in A \rightarrow P a \ T \rightsquigarrow (ev\ e)\ (P\ e) \rangle$
by (*fact ev-trans-Mndetprefix*)

lemma $\langle e \rightarrow P \ T \rightsquigarrow (ev\ e)\ P \rangle$
by (*fact ev-trans-prefix*)

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \ T \rightsquigarrow P \rangle$
and $\langle P \sqcap Q \ T \rightsquigarrow Q \rangle$
by (*fact τ -trans-NdetL τ -trans-NdetR*) $+$

lemma $\langle a \in A \implies (\Box a \in A.\ P\ a) \ T \rightsquigarrow P\ a \rangle$
by (*fact τ -trans-GlobalNdet*)

lemma $\langle \text{finite } A \implies a \in A \implies (\Box a \in A.\ P\ a) \ T \rightsquigarrow P\ a \rangle$
by (*fact τ -trans-MultiNdet*)

$\mu\ x.\ f\ x$ law

lemma $\langle \text{cont } f \implies P = (\mu\ X.\ f\ X) \implies P \ T \rightsquigarrow f\ P \rangle$
by (*fact fix-point- τ -trans*)

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \ T \rightsquigarrow P' \implies P \sqcap Q \ T \rightsquigarrow P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \ T \rightsquigarrow Q' \implies P \sqcap Q \ T \rightsquigarrow P \sqcap Q' \rangle$
by *simp-all*

lemma τ -trans-MultiDet:
 $\langle \text{finite } A \implies \forall a \in A.\ P\ a \ T \rightsquigarrow P' a \implies (\Box a \in A.\ P\ a) \ T \rightsquigarrow (\Box a \in A.\ P' a) \rangle$
by (*fact mono-MultiDet-T*)

lemma $\langle P \ T \rightsquigarrow e\ P' \implies P \sqcap Q \ T \rightsquigarrow e\ P' \rangle$
and $\langle Q \ T \rightsquigarrow e\ Q' \implies P \sqcap Q \ T \rightsquigarrow e\ Q' \rangle$
by (*fact event-trans-DetL event-trans-DetR*) $+$

lemma $\langle \text{finite } A \implies a \in A \implies P\ a \ T \rightsquigarrow e\ Q \implies (\Box a \in A.\ P\ a) \ T \rightsquigarrow e\ Q \rangle$
by (*fact event-trans-MultiDet*)

$P ; Q$ laws

lemma τ -trans-SegR: $\langle \exists P'. P \ T \rightsquigarrow \checkmark\ P' \implies P ; Q \ T \rightsquigarrow Q \rangle$
by (*metis D-SKIP D-STOP SKIP-Seq divergence-refine-def leDT-imp-leT
leD-STOP leD-leT-imp-leDT mono-Seq-DT-left ready-tick-imp- τ -trans-SKIP*)

lemma $\langle Q \ T \rightsquigarrow Q' \implies P ; Q \ T \rightsquigarrow P ; Q' \rangle$
by (*fact mono-Seq-T-right*)

lemma $\langle \checkmark \in \text{ready-set } P \implies Q \xrightarrow{\tau} (ev \ e) \ Q' \implies P ; Q \xrightarrow{\tau} (ev \ e) \ Q' \rangle$

by (*fact ev-trans-SeqR*)

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \xrightarrow{\tau} P' \implies P \setminus B \xrightarrow{\tau} P' \setminus B \rangle$

by (*fact mono-Hiding-T*)

lemma *ev-trans-Hiding-notin*:

$\langle P \xrightarrow{\tau} (ev \ e) \ P' \implies e \notin B \implies P \setminus B \xrightarrow{\tau} (ev \ e) \ P' \setminus B \rangle$

by (*metis AfterExt-def After-Hiding-T-Hiding-After-if-ready-notin mono-Hiding-T*

event-trans- τ -trans event.simps(4) ready-notin-imp-ready-Hiding)

lemma $\langle P \xrightarrow{\tau} \checkmark \ P' \implies P \setminus B \xrightarrow{\tau} \checkmark \ STOP \rangle$

by (*fact tick-trans-Hiding*)

lemma *ev-trans-Hiding-inside*:

$\langle P \xrightarrow{\tau} (ev \ e) \ P' \implies e \in B \implies P \setminus B \xrightarrow{\tau} P' \setminus B \rangle$

by (*metis AfterExt-def Hiding-T-Hiding-After-if-ready-inside*

mono-Hiding-T event.simps(4) trans-T)

Renaming P f laws

lemma *tick-trans-Renaming*: $\langle P \xrightarrow{\tau} \checkmark \ P' \implies \text{Renaming } P \ f \ \xrightarrow{\tau} \checkmark \ STOP \rangle$

by (*simp add: AfterExt-def ready-set-Renaming tick-eq-EvExt*)

lemma $\langle P \xrightarrow{\tau} \checkmark \ P' \implies P \llbracket a := b \rrbracket \xrightarrow{\tau} \checkmark \ STOP \rangle$

by (*fact tick-trans-Renaming*)

$P \llbracket S \rrbracket Q$ laws

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

lemma *tick-trans-SyncL*: $\langle P \xrightarrow{\tau} \checkmark \ P' \implies P \llbracket S \rrbracket Q \xrightarrow{\tau} \text{SKIP} \llbracket S \rrbracket Q \rangle$

and *tick-trans-SyncR*: $\langle Q \xrightarrow{\tau} \checkmark \ Q' \implies P \llbracket S \rrbracket Q \xrightarrow{\tau} P \llbracket S \rrbracket \text{SKIP} \rangle$

by (*simp-all add: D-SKIP divergence-refine-def leDT-imp-leT leD-leT-imp-leDT ready-tick-imp- τ -trans-SKIP*)

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \xrightarrow{\tau} \checkmark \ STOP \rangle$

by (*simp add: SKIP-trans-tick Sync-SKIP-SKIP*)

lemma τ -trans-SKIP-Sync-SKIP: $\langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \xrightarrow{\tau} \text{SKIP} \rangle$

by (*simp add: Sync-SKIP-SKIP*)

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \xrightarrow{T\rightsquigarrow\tau} P' \implies P \triangleright Q \xrightarrow{T\rightsquigarrow\tau} P' \triangleright Q \rangle$
unfolding *Sliding-def by simp*

lemma $\langle P \xrightarrow{T\rightsquigarrow e} P' \implies P \triangleright Q \xrightarrow{T\rightsquigarrow e} P' \rangle$
by (*fact Sliding-event-transL*)

lemma $\langle P \triangleright Q \xrightarrow{T\rightsquigarrow\tau} Q \rangle$
by (*fact Sliding- τ -transR*)

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \xrightarrow{T\rightsquigarrow\tau} P' \implies P \triangle Q \xrightarrow{T\rightsquigarrow\tau} P' \triangle Q \rangle$
by (*simp add: mono-Interrupt-T*)

lemma *Interrupt- τ -trans-right*: $\langle Q \xrightarrow{T\rightsquigarrow\tau} Q' \implies P \triangle Q \xrightarrow{T\rightsquigarrow\tau} P \triangle Q' \rangle$
by (*simp add: mono-Interrupt-T*)

lemma *Interrupt-ev-trans-left*: $\langle P \xrightarrow{T\rightsquigarrow}(ev\ e) P' \implies P \triangle Q \xrightarrow{T\rightsquigarrow}(ev\ e) P' \triangle Q \rangle$
by (*simp add: AfterExt-def After-Interrupt Interrupt- τ -trans-left ready-set-Interrupt*)

lemma *Interrupt-ev-trans-right*: $\langle Q \xrightarrow{T\rightsquigarrow}(ev\ e) Q' \implies P \triangle Q \xrightarrow{T\rightsquigarrow}(ev\ e) Q' \rangle$
by (*simp add: AfterExt-def After-Interrupt ready-set-Interrupt*)

Throw P A Q laws

lemma *Throw- τ -trans-right*:
 $\langle \forall a \in A. Q \ a \xrightarrow{T\rightsquigarrow\tau} Q' \ a \implies P \ \Theta \ a \in A. Q \ a \xrightarrow{T\rightsquigarrow\tau} P \ \Theta \ a \in A. Q' \ a \rangle$
by (*fact mono-right-Throw-T*)

lemma *Throw-ev-trans-right*:
 $\langle P \xrightarrow{T\rightsquigarrow}(ev\ e) P' \implies e \in A \implies P \ \Theta \ a \in A. Q \ a \xrightarrow{T\rightsquigarrow}(ev\ e) (Q \ e) \rangle$
by (*simp add: AfterExt-Throw ready-set-Throw split: event.split*)

lemma $\langle tickFree\ s \implies \perp \xrightarrow{T\rightsquigarrow^*} s \ P \rangle$
by (*fact BOT-trace-trans-tickFree-anything*)

15.2 Reality Checks

lemma $\langle STOP \xrightarrow{T\rightsquigarrow^*} s \ P \longleftrightarrow s = \square \wedge P = STOP \rangle$
by (*fact STOP-trace-trans-iff*)

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T} \ P \longleftrightarrow (\exists P'. P \xrightarrow{T\rightsquigarrow^*} s \ P') \rangle$

by (*meson T-imp-exists-trace-trans trace-trans-imp-T-if-τ-trans-imp-leT*)

15.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:

$\langle P \xrightarrow{T}^* s \ Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$

by (*metis T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace-τ-trans τ-trans-anti-mono-ready-set*)

lemma *trace-trans-imp-ready-set*:

$\langle P \xrightarrow{T}^* (s \ @ \ e \ \# \ t) \ Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$

using *T-iff-exists-trans ready-set-AfterTrace* by *blast*

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :

$\langle (P \xrightarrow{T}^* s \ Q) \longleftrightarrow s \in \mathcal{T} \ P \ \wedge \ P \text{ afterTrace } s \xrightarrow{T} \tau \ Q \rangle$

using *T-iff-exists-trans T-imp-trace-trans-iff-AfterTrace-τ-trans* by *blast*

15.4 Summary: Operational Rules

In this section, we will just write down the operational laws that we have proven in a fancy way.

Absorbance rules

$$\frac{?P \xrightarrow{T} ?e \ ?P' \quad ?P' \xrightarrow{T} \tau \ ?P''}{?P \xrightarrow{T} ?e \ ?P''} \quad \frac{?P \xrightarrow{T} \tau \ ?P' \quad ?P' \xrightarrow{T} ?e \ ?P''}{?P \xrightarrow{T} ?e \ ?P''}$$

SKIP rule

$$\overline{SKIP \xrightarrow{T} \checkmark \ STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \ \xrightarrow{T} \text{ev} \ ?e \ ?P \ ?e} \quad \frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \ \xrightarrow{T} \text{ev} \ ?e \ ?P \ ?e}$$

$$\overline{?e \rightarrow ?P \ \xrightarrow{T} \text{ev} \ ?e \ ?P}$$

(\sqcap) **rules**

$$\frac{\frac{\overline{?P \sqcap ?Q \text{ } T \rightsquigarrow_{\tau} ?P} \quad \overline{?P \sqcap ?Q \text{ } T \rightsquigarrow_{\tau} ?Q}}{?e \in ?A}}{\sqcap a \in ?A. ?P a \text{ } T \rightsquigarrow_{\tau} ?P ?e}$$

$\mu x. f x$ **rule**

$$\frac{\text{cont } ?f \quad ?P = (\mu x. ?f x)}{?P \text{ } T \rightsquigarrow_{\tau} ?f ?P}$$

(\square) **rules**

$$\frac{\frac{?P \text{ } T \rightsquigarrow_{\tau} ?P'}{?P \sqcap ?Q \text{ } T \rightsquigarrow_{\tau} ?P' \sqcap ?Q} \quad \frac{?Q \text{ } T \rightsquigarrow_{\tau} ?Q'}{?P \sqcap ?Q \text{ } T \rightsquigarrow_{\tau} ?P \sqcap ?Q'}}{\frac{?P \text{ } T \rightsquigarrow_{\tau} ?e ?P'}{?P \sqcap ?Q \text{ } T \rightsquigarrow_{\tau} ?e ?P'} \quad \frac{?Q \text{ } T \rightsquigarrow_{\tau} ?e ?Q'}{?P \sqcap ?Q \text{ } T \rightsquigarrow_{\tau} ?e ?Q'}}$$

(;) **rule**

$$\frac{\exists P'. ?P \text{ } T \rightsquigarrow_{\tau} \checkmark P'}{?P ; ?Q \text{ } T \rightsquigarrow_{\tau} ?Q}$$

Hiding rules

$$\frac{\frac{?P \text{ } T \rightsquigarrow_{\tau} ?P'}{?P \setminus ?B \text{ } T \rightsquigarrow_{\tau} ?P' \setminus ?B} \quad \frac{?P \text{ } T \rightsquigarrow_{\tau} \checkmark ?P'}{?P \setminus ?B \text{ } T \rightsquigarrow_{\tau} \checkmark STOP}}{\frac{?P \text{ } T \rightsquigarrow_{\tau} \text{ev } ?e ?P' \quad ?e \notin ?B}{?P \setminus ?B \text{ } T \rightsquigarrow_{\tau} \text{ev } ?e ?P' \setminus ?B} \quad \frac{?P \text{ } T \rightsquigarrow_{\tau} \text{ev } ?e ?P' \quad ?e \in ?B}{?P \setminus ?B \text{ } T \rightsquigarrow_{\tau} ?P' \setminus ?B}}$$

Renaming rule

$$\frac{?P \text{ } T \rightsquigarrow_{\tau} \checkmark ?P'}{\text{Renaming } ?P ?f \text{ } T \rightsquigarrow_{\tau} \checkmark STOP}$$

Sync rules

$$\frac{?P \xrightarrow{\tau} \checkmark ?P'}{?P \llbracket ?S \rrbracket ?Q \xrightarrow{\tau} \text{SKIP} \llbracket ?S \rrbracket ?Q} \quad \frac{?Q \xrightarrow{\tau} \checkmark ?Q'}{?P \llbracket ?S \rrbracket ?Q \xrightarrow{\tau} ?P \llbracket ?S \rrbracket \text{SKIP}}$$

$$\frac{}{\text{SKIP} \llbracket ?S \rrbracket \text{SKIP} \xrightarrow{\tau} \text{SKIP}}$$

(▷) rules

$$\frac{?P \xrightarrow{\tau} ?P'}{?P \triangleright ?Q \xrightarrow{\tau} ?P' \triangleright ?Q} \quad \frac{?P \xrightarrow{\tau} ?e ?P'}{?P \triangleright ?Q \xrightarrow{\tau} ?e ?P'}$$

$$\frac{}{?P \triangleright ?Q \xrightarrow{\tau} ?Q}$$

(△) rules

$$\frac{?P \xrightarrow{\tau} ?P'}{?P \triangle ?Q \xrightarrow{\tau} ?P' \triangle ?Q} \quad \frac{?Q \xrightarrow{\tau} ?Q'}{?P \triangle ?Q \xrightarrow{\tau} ?P \triangle ?Q'}$$

$$\frac{?P \xrightarrow{\tau} ?e ?P'}{?P \triangle ?Q \xrightarrow{\tau} ?e ?P' \triangle ?Q} \quad \frac{?Q \xrightarrow{\tau} ?e ?Q'}{?P \triangle ?Q \xrightarrow{\tau} ?e ?Q'}$$

Throw rules

$$\frac{\forall a \in ?A. ?Q a \xrightarrow{\tau} ?Q' a}{?P \Theta a \in ?A. ?Q a \xrightarrow{\tau} ?P \Theta a \in ?A. ?Q' a}$$

$$\frac{?P \xrightarrow{\tau} ?e ?P' \quad ?e \in ?A}{?P \Theta a \in ?A. ?Q a \xrightarrow{\tau} ?e ?Q ?e}$$

end

Chapter 16

Bonus: powerful new Laws

```
theory NewLaws
  imports Interrupt Sliding Throw
begin
```

16.1 Powerful Results about *Renaming*

In this section we will provide laws about the *Renaming* operator. In the first subsection we will give slight generalizations of previous results, but in the other we prove some very powerful theorems.

16.1.1 Some Generalizations

For *Renaming*, we can obtain generalizations of the following results:

$$\text{Renaming } (\Box a \in A \rightarrow P (f a)) f = \Box b \in f ' A \rightarrow \text{Renaming } (P b) f$$
$$\text{Renaming } (\Box a \in A \rightarrow P (f a)) f = \Box b \in f ' A \rightarrow \text{Renaming } (P b) f$$

lemma *Renaming-Mprefix*:

$$\langle \text{Renaming } (\Box a \in A \rightarrow P a) f =$$
$$\Box y \in f ' A \rightarrow \Box a \in \{x \in A. y = f x\}. \text{Renaming } (P a) f \rangle \text{ (is } \langle ?lhs = ?rhs \rangle)$$

proof (*subst Process-eq-spec-optimized, safe*)

show $\langle s \in \mathcal{D} \ ?lhs \implies s \in \mathcal{D} \ ?rhs \rangle$ **for** s

by (*auto simp add: D-Renaming D-Mprefix D-GlobalNdet hd-map-EvExt*)

(*use list.map-sel(2) tickFree-tl in blast*)

next

fix s

assume $\langle s \in \mathcal{D} \ ?rhs \rangle$

then obtain $a \ s'$ **where** $*$: $\langle a \in A \rangle \langle s = \text{EvExt } f (ev a) \# s' \rangle$

$$\langle s' \in \mathcal{D} (\text{Renaming } (P a) f) \rangle$$

by (*simp add: D-Mprefix EvExt-def D-GlobalNdet split: if-split-asm*)

(*metis list.collapse*)

from $*(3)$ **obtain** $s1 \ s2$

where $**$: $\langle \text{tickFree } s1 \rangle \langle \text{front-tickFree } s2 \rangle$

$$\langle s' = \text{map } (\text{EvExt } f) \ s1 \ @ \ s2 \rangle \langle s1 \in \mathcal{D} (P a) \rangle$$

```

    by (simp add: D-Renaming) blast
  have *** : ⟨tickFree (ev a # s1) ∧
    s = EvExt f (ev a) # map (EvExt f) s1 @ s2 ∧ ev a # s1 ∈ D (Mprefix
A P)⟩
    by (simp add: D-Mprefix *(1, 2) *(1, 3, 4))
  show ⟨s ∈ D ?lhs⟩
    by (simp add: D-Renaming )
      (metis *(2) *** append-Cons list.simps(9))
next
fix s X
assume same-div: ⟨D ?lhs = D ?rhs⟩
assume ⟨(s, X) ∈ F ?lhs⟩
then consider ⟨s ∈ D ?lhs⟩
  | ⟨∃ t. (t, EvExt f -' X) ∈ F (Mprefix A P) ∧ s = map (EvExt f) t⟩
    by (simp add: F-Renaming D-Renaming) blast
  thus ⟨(s, X) ∈ F ?rhs⟩
proof cases
  from D-F same-div show ⟨s ∈ D ?lhs ⟹ (s, X) ∈ F ?rhs⟩ by blast
next
assume ⟨∃ t. (t, EvExt f -' X) ∈ F (Mprefix A P) ∧ s = map (EvExt f) t⟩
then obtain t where * : ⟨(t, EvExt f -' X) ∈ F (Mprefix A P)⟩
  ⟨s = map (EvExt f) t⟩ by blast
show ⟨(s, X) ∈ F ?rhs⟩
proof (cases ⟨t = []⟩)
  from * show ⟨t = [] ⟹ (s, X) ∈ F ?rhs⟩
    by (simp add: F-Mprefix disjoint-iff-not-equal)
      (metis ev-elem-anteced1)
next
assume ⟨t ≠ []⟩
with *(1) obtain a t'
  where ** : ⟨a ∈ A⟩ ⟨t = ev a # t'⟩ ⟨(t', EvExt f -' X) ∈ F (P a)⟩
  by (simp add: F-Mprefix) (metis event.inject imageE list.exhaust-sel)
have *** : ⟨s ≠ [] ∧ hd s ∈ ev ' f ' A⟩
  using *(2) *(1, 2) hd-map-EvExt by fastforce
with ** have ⟨ev (f a) = hd s ∧
  (tl s, X) ∈ F (∏ a ∈ {x ∈ A. f a = f x}. Renaming (P a) f)⟩
  by (simp add: F-GlobalNdet *(2) F-Renaming)
  (metis (no-types, opaque-lifting) ⟨t ≠ []⟩ hd-map hd-map-EvExt list.sel(1))
with *** show ⟨(s, X) ∈ F ?rhs⟩ by (simp add: F-Mprefix) blast
qed
qed
next
show ⟨(s, X) ∈ F ?rhs ⟹ (s, X) ∈ F ?lhs⟩ for s X
proof (cases s)
  show ⟨s = [] ⟹ (s, X) ∈ F ?rhs ⟹ (s, X) ∈ F ?lhs⟩
    by (simp add: F-Mprefix F-Renaming disjoint-iff-not-equal)
      (metis ev-elem-anteced1)
next
fix b s'

```

assume $\langle s = b \# s' \rangle \langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$
then obtain a
where $*$: $\langle a \in A \rangle \langle s = (\text{EvExt } f) (ev \ a) \# s' \rangle$
 $\langle (s', X) \in \mathcal{F} (\sqcap a \in \{x \in A. f \ a = f \ x\}. \text{Renaming } (P \ a) \ f) \rangle$
by (*auto simp add: F-Mprefix EvExt-def*)
from $*(1, 3)$ **obtain** a'
where $**$: $\langle a' \in A \rangle \langle f \ a' = f \ a \rangle \langle (s', X) \in \mathcal{F} (\text{Renaming } (P \ a') \ f) \rangle$
by (*auto simp add: F-GlobalNdet split: if-split-asm*)
from $** (3)$ **consider**
 $\langle \exists t. (t, \text{EvExt } f \ -' \ X) \in \mathcal{F} (P \ a') \wedge s' = \text{map } (\text{EvExt } f) \ t \rangle$
 $| \langle \exists s1 \ s2. \text{tickFree } s1 \wedge \text{front-tickFree } s2 \wedge$
 $s' = \text{map } (\text{EvExt } f) \ s1 \ @ \ s2 \wedge s1 \in \mathcal{D} (P \ a') \rangle$
by (*simp add: F-Renaming blast*)
thus $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
proof cases
assume $\langle \exists t. (t, \text{EvExt } f \ -' \ X) \in \mathcal{F} (P \ a') \wedge s' = \text{map } (\text{EvExt } f) \ t \rangle$
then obtain t **where** $***$: $\langle (t, \text{EvExt } f \ -' \ X) \in \mathcal{F} (P \ a') \rangle$
 $\langle s' = \text{map } (\text{EvExt } f) \ t \rangle$ **by** *blast*
have $****$: $\langle (ev \ a' \ \# \ t, \text{EvExt } f \ -' \ X) \in \mathcal{F} (\text{Mprefix } A \ P) \wedge$
 $s = \text{map } (\text{EvExt } f) (ev \ a' \ \# \ t) \rangle$
by (*simp add: F-Mprefix *(2) **(1) *** (1, 2)*)
(*metis *(2) ev-elem-anteced1 vimage-singleton-eq*)
show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$
apply (*simp add: F-Renaming*)
using $****$ **by** *blast*
next
assume $\langle \exists s1 \ s2. \text{tickFree } s1 \wedge \text{front-tickFree } s2 \wedge$
 $s' = \text{map } (\text{EvExt } f) \ s1 \ @ \ s2 \wedge s1 \in \mathcal{D} (P \ a') \rangle$
then obtain $s1 \ s2$
where $***$: $\langle \text{tickFree } s1 \rangle \langle \text{front-tickFree } s2 \rangle$
 $\langle s' = \text{map } (\text{EvExt } f) \ s1 \ @ \ s2 \rangle \langle s1 \in \mathcal{D} (P \ a') \rangle$ **by** *blast*
have $\langle \text{tickFree } (ev \ a' \ \# \ s1) \wedge$
 $s = \text{EvExt } f (ev \ a') \ \# \ \text{map } (\text{EvExt } f) \ s1 \ @ \ s2 \wedge$
 $ev \ a' \ \# \ s1 \in \mathcal{D} (\text{Mprefix } A \ P) \rangle$
by (*simp add: *(2) **(1) *** D-Mprefix*)
(*metis *(2) ev-elem-anteced1 vimage-singleton-eq*)
thus $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ **by** (*auto simp add: *** (2) F-Renaming*)
qed
qed
qed

lemma *Renaming-Mprefix-Sliding*:

$\langle \text{Renaming } ((\sqcap a \in A \rightarrow P \ a) \triangleright Q) \ f =$

$(\sqcap y \in f \ ' \ A \rightarrow \sqcap a \in \{x \in A. y = f \ x\}. \text{Renaming } (P \ a) \ f) \triangleright \text{Renaming } Q \ f \rangle$

unfolding *Sliding-def*

by (*simp add: Renaming-Ndet Renaming-Det Renaming-Mprefix*)

lemma *Renaming-GlobalNdet*:
 $\langle \text{Renaming } (\sqcap a \in A. P a) f = \sqcap b \in f ' A. \sqcap a \in \{a \in A. b = f a\}. \text{Renaming } (P a) f \rangle$
by (*simp add: Process-eq-spec F-Renaming F-GlobalNdet D-Renaming D-GlobalNdet blast*)

lemma *Renaming-Mndetprefix*:
 $\langle \text{Renaming } (\sqcap a \in A \rightarrow P a) f = \sqcap y \in f ' A \rightarrow \sqcap a \in \{x \in A. y = f x\}. \text{Renaming } (P a) f \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
apply (*subst (1 2) Mndetprefix-GlobalNdet*)
apply (*simp add: Renaming-GlobalNdet Renaming-prefix*)
apply (*intro mono-GlobalNdet-eq ballI*)
apply (*subst write0-GlobalNdet[symmetric], blast*)
by (*simp add: mono-GlobalNdet-eq*)

16.1.2 Renaming and Hiding

When f is one to one, $\text{Renaming } (P \setminus S) f$ will behave like we expect it to do.

lemma *strict-mono-map*: $\langle \text{strict-mono } g \implies \text{strict-mono } (\lambda i. \text{map } f (g i)) \rangle$
unfolding *strict-mono-def less-list-def le-list-def*
by (*metis list.map-disc-iff map-append self-append-conv*)

lemma *trace-hide-map-EvExt* :
 $\langle \text{inj-on } (\text{EvExt } f) (\text{set } s \cup \text{ev } ' S) \implies \text{trace-hide } (\text{map } (\text{EvExt } f) s) (\text{ev } ' f ' S) = \text{map } (\text{EvExt } f) (\text{trace-hide } s (\text{ev } ' S)) \rangle$
proof (*induct s*)
case *Nil*
show *?case* **by** *simp*
next
case (*Cons a s*)
hence $*$: $\langle \text{trace-hide } (\text{map } (\text{EvExt } f) s) (\text{ev } ' f ' S) = \text{map } (\text{EvExt } f) (\text{trace-hide } s (\text{ev } ' S)) \rangle$ **by** *fastforce*
from *Cons.premis[unfolded inj-on-def, rule-format, of a, simplified]* **show** *?case*
apply (*simp add: **)
apply (*simp add: image-iff split: event.split*)
by (*metis ev-elem-anteced1 singletonI vimage-singleton-eq*)
qed

lemma *inj-on-EvExt-iff*: $\langle \text{inj-on } (\text{EvExt } f) (\text{insert tick } (\text{ev } ' S)) \longleftrightarrow \text{inj-on } f S \rangle$
unfolding *inj-on-def EvExt-def* **by** (*auto split: event.split*)

lemma *inj-on-EvExt-set-T*:
 $\langle \text{inj-on } f (\text{events-of } P) \implies s \in \mathcal{T} P \implies \text{inj-on } (\text{EvExt } f) (\text{insert tick } (\text{set } s)) \rangle$
unfolding *inj-on-def EvExt-def events-of-def* **by** (*auto split: event.split*)

theorem *bij-Renaming-Hiding*: $\langle \text{Renaming } (P \setminus S) f = \text{Renaming } P f \setminus f ' S \rangle$
 (is $\langle ?lhs = ?rhs \rangle$) **if** *bij-f*: $\langle \text{bij } f \rangle$
proof (*subst Process-eq-spec-optimized, safe*)
fix *s*
assume $\langle s \in \mathcal{D} \ ?lhs \rangle$
then obtain *s1 s2* **where** $*$: $\langle \text{tickFree } s1 \rangle \langle \text{front-tickFree } s2 \rangle$
 $\langle s = \text{map } (\text{EvExt } f) \ s1 \ @ \ s2 \rangle \langle s1 \in \mathcal{D} \ (P \setminus S) \rangle$
by (*simp add: D-Renaming*) *blast*
from $*(4)$ **obtain** *t u*
where $**$: $\langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle \langle s1 = \text{trace-hide } t \ (ev ' S) \ @ \ u \rangle$
 $\langle t \in \mathcal{D} \ P \vee (\exists g. \text{isInfHiddenRun } g \ P \ S \wedge t \in \text{range } g) \rangle$
by (*simp add: D-Hiding*) *blast*
from $**(4)$ **show** $\langle s \in \mathcal{D} \ ?rhs \rangle$
proof (*elim disjE*)
assume $\langle t \in \mathcal{D} \ P \rangle$
hence $\langle \text{front-tickFree } (\text{map } (\text{EvExt } f) \ u \ @ \ s2) \wedge \text{tickFree } (\text{map } (\text{EvExt } f) \ t) \wedge$
 $s = \text{trace-hide } (\text{map } (\text{EvExt } f) \ t) \ (ev ' f ' S) \ @ \ \text{map } (\text{EvExt } f) \ u \ @ \ s2 \wedge$
 $\text{map } (\text{EvExt } f) \ t \in \mathcal{D} \ (\text{Renaming } P f) \rangle$
apply (*simp add: *(3) **(2, 3) EvExt-tF, intro conjI*)
subgoal using $*(1, 2) \ ***(3) \ \text{EvExt-tF front-tickFree-append tickFree-append}$
by *blast*
subgoal by (*rule trace-hide-map-EvExt[symmetric];*
use bij-f in <unfold bij-def inj-on-def EvExt-def, auto split: event.split>)
using $**(2) \ D\text{-Renaming front-tickFree-Nil}$ **by** *blast*
thus $\langle s \in \mathcal{D} \ ?rhs \rangle$ **by** (*simp add: D-Hiding*) *blast*
next
assume $\langle \exists g. \text{isInfHiddenRun } g \ P \ S \wedge t \in \text{range } g \rangle$
then obtain *g* **where** $\langle \text{isInfHiddenRun } g \ P \ S \rangle \langle t \in \text{range } g \rangle$ **by** *blast*
hence $\langle \text{front-tickFree } (\text{map } (\text{EvExt } f) \ u \ @ \ s2) \wedge$
 $\text{tickFree } (\text{map } (\text{EvExt } f) \ t) \wedge$
 $s = \text{trace-hide } (\text{map } (\text{EvExt } f) \ t) \ (ev ' f ' S) \ @ \ \text{map } (\text{EvExt } f) \ u \ @ \ s2 \wedge$
 $\text{isInfHiddenRun } (\lambda i. \text{map } (\text{EvExt } f) \ (g \ i)) \ (\text{Renaming } P f) \ (f ' S) \wedge$
 $\text{map } (\text{EvExt } f) \ t \in \text{range } (\lambda i. \text{map } (\text{EvExt } f) \ (g \ i)) \rangle$
apply (*simp add: *(3) **(2, 3) EvExt-tF, intro conjI*)
subgoal using $*(1, 2) \ ***(3) \ \text{EvExt-tF front-tickFree-append tickFree-append}$
by *blast*
subgoal by (*rule trace-hide-map-EvExt[symmetric];*
use bij-f in <unfold bij-def inj-on-def EvExt-def, auto split: event.split>)
subgoal using *strict-mono-map* **by** *blast*
subgoal by (*solves <auto simp add: T-Renaming>*)
subgoal apply (*subst (1 2) trace-hide-map-EvExt*)
by (*use bij-f in <unfold bij-def inj-on-def EvExt-def, auto split: event.split>*)
metis
subgoal by *blast*
done
thus $\langle s \in \mathcal{D} \ ?rhs \rangle$ **by** (*simp add: D-Hiding*) *blast*
qed

```

next
  fix s
  assume ⟨s ∈ D ?rhs⟩
  then obtain t u
    where * : ⟨front-tickFree u⟩ ⟨tickFree t⟩ ⟨s = trace-hide t (ev ' f ' S) @ u⟩
      ⟨t ∈ D (Renaming P f)⟩ ∨
      ⟨∃ g. isInfHiddenRun g (Renaming P f) (f ' S) ∧ t ∈ range g⟩
    by (simp add: D-Hiding) blast
  from *(4) show ⟨s ∈ D ?lhs⟩
  proof (elim disjE)
    assume ⟨t ∈ D (Renaming P f)⟩
    then obtain t1 t2 where ** : ⟨tickFree t1⟩ ⟨front-tickFree t2⟩
      ⟨t = map (EvExt f) t1 @ t2⟩ ⟨t1 ∈ D P⟩
    by (simp add: D-Renaming) blast
    have ⟨tickFree (trace-hide t1 (ev ' S)) ∧
      front-tickFree (trace-hide t2 (ev ' f ' S) @ u) ∧
      trace-hide (map (EvExt f) t1) (ev ' f ' S) @ trace-hide t2 (ev ' f ' S) @ u
    =
      map (EvExt f) (trace-hide t1 (ev ' S)) @ trace-hide t2 (ev ' f ' S) @ u ∧
      trace-hide t1 (ev ' S) ∈ D (P \ S)⟩
    apply (simp, intro conjI)
    subgoal using *(1) Hiding-tickFree by blast
    subgoal using *(1, 2) *(3) Hiding-tickFree front-tickFree-append tick-
Free-append by blast
    subgoal by (rule trace-hide-map-EvExt;
      use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, simp split: event.split⟩)
    subgoal using *(4) elemDisemHD by blast
    done
  thus ⟨s ∈ D ?lhs⟩ by (simp add: D-Renaming *(3) *(3)) blast
next
  have inv-S: ⟨S = inv f ' f ' S⟩ by (simp add: bij-is-inj bij-f)
  have inj-inv-f: ⟨inj (inv f)⟩
  by (simp add: bij-imp-bij-inv bij-is-inj bij-f)
  have ** : ⟨map (EvExt (inv f) ∘ EvExt f) v = v⟩ for v
  by (induct v, unfold EvExt-def, auto split: event.split)
  (metis bij-inv-eq-iff bij-f)
  assume ⟨∃ g. isInfHiddenRun g (Renaming P f) (f ' S) ∧ t ∈ range g⟩
  then obtain g
    where *** : ⟨isInfHiddenRun g (Renaming P f) (f ' S)⟩ ⟨t ∈ range g⟩ by
blast
  then consider ⟨∃ t1. t1 ∈ T P ∧ t = map (EvExt f) t1⟩
  | ⟨∃ t1 t2. tickFree t1 ∧ front-tickFree t2 ∧
    t = map (EvExt f) t1 @ t2 ∧ t1 ∈ D P⟩
  by (simp add: T-Renaming) blast
  thus ⟨s ∈ D ?lhs⟩
  proof cases
    assume ⟨∃ t1. t1 ∈ T P ∧ t = map (EvExt f) t1⟩
    then obtain t1 where **** : ⟨t1 ∈ T P⟩ ⟨t = map (EvExt f) t1⟩ by blast
    have ***** : ⟨t1 = map (EvExt (inv f)) t⟩ by (simp add: ****(2) **)

```



```

have ***** : ⟨trace-hide t1 (ev ‘ S) = trace-hide t1 (ev ‘ S) ∧
  isInfHiddenRun (λi. map (EvExt (inv f)) (g i)) P S ∧
  t1 ∈ range (λi. map (EvExt (inv f)) (g i))⟩
apply (simp add: ***(1) strict-mono-map, intro conjI)
subgoal apply (subst Renaming-inv[where f = f, symmetric])
  apply (solves ⟨simp add: bij-is-inj bij-f⟩)
  using ***(1) by (subst T-Renaming, blast)
subgoal apply (subst (1 2) inv-S, subst (1 2) trace-hide-map-EvExt)
  by (use bij-f in ⟨unfold bij-def inj-on-def EvExt-def,
    auto simp add: inj-eq inj-inv-f split: event.split⟩)
    (metis ***(1))
subgoal using ***(2) ***** by blast
done
have ⟨tickFree (trace-hide t1 (ev ‘ S)) ∧ front-tickFree t1 ∧
  trace-hide (map (EvExt f) t1) (ev ‘ f ‘ S) @ u =
  map (EvExt f) (trace-hide t1 (ev ‘ S)) @ u ∧
  trace-hide t1 (ev ‘ S) ∈ D (P \ S)⟩
apply (simp, intro conjI)
subgoal using *(2) ***** (2) EvExt-tF Hiding-tickFree by blast

subgoal using *****(1) is-processT2-TR by blast
apply (rule trace-hide-map-EvExt,
  use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, auto split: event.split⟩)[1]
apply (simp add: D-Renaming D-Hiding)
using *(2) ***** EvExt-tF front-tickFree-Nil by blast
with *(1) show ⟨s ∈ D ?lhs⟩ by (simp add: D-Renaming *(3) *****(2)) blast
next
assume ⟨∃ t1 t2. tickFree t1 ∧ front-tickFree t2 ∧
  t = map (EvExt f) t1 @ t2 ∧ t1 ∈ D P⟩
then obtain t1 t2
  where ***** : ⟨tickFree t1⟩ ⟨front-tickFree t2⟩
    ⟨t = map (EvExt f) t1 @ t2⟩ ⟨t1 ∈ D P⟩ by blast
have ⟨tickFree (trace-hide t1 (ev ‘ S)) ∧
  front-tickFree (trace-hide t2 (ev ‘ f ‘ S) @ u) ∧
  trace-hide (map (EvExt f) t1) (ev ‘ f ‘ S) @ trace-hide t2 (ev ‘ f ‘ S) @
u =
  map (EvExt f) (trace-hide t1 (ev ‘ S)) @ trace-hide t2 (ev ‘ f ‘ S) @ u ∧
  trace-hide t1 (ev ‘ S) ∈ D (P \ S)⟩
apply (simp, intro conjI)
subgoal using *****(1) Hiding-tickFree by blast
subgoal using *(1, 2) *****(3) Hiding-tickFree front-tickFree-append tick-
Free-append by blast
subgoal by (rule trace-hide-map-EvExt;
  use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, simp split: event.split⟩)
subgoal using *****(4) elemDisemHD by blast
done
thus ⟨s ∈ D ?lhs⟩ by (simp add: D-Renaming *(3) *****(3)) blast
qed
qed

```

```

next
  fix s X
  assume same-div : ⟨D ?lhs = D ?rhs⟩
  assume ⟨(s, X) ∈ F ?lhs⟩
  then consider ⟨s ∈ D ?lhs⟩
    | ⟨∃ s1. (s1, EvExt f -' X) ∈ F (P \ S) ∧ s = map (EvExt f) s1⟩
      by (simp add: F-Renaming D-Renaming) blast
  thus ⟨(s, X) ∈ F ?rhs⟩
  proof cases
    from D-F same-div show ⟨s ∈ D ?lhs ⇒ (s, X) ∈ F ?rhs⟩ by blast
  next
    assume ⟨∃ s1. (s1, EvExt f -' X) ∈ F (P \ S) ∧ s = map (EvExt f) s1⟩
    then obtain s1 where * : ⟨(s1, EvExt f -' X) ∈ F (P \ S)⟩
      ⟨s = map (EvExt f) s1⟩ by blast
    from this(1) consider ⟨s1 ∈ D (P \ S)⟩
      | ⟨∃ t. s1 = trace-hide t (ev ' S) ∧ (t, EvExt f -' X ∪ ev ' S) ∈ F P⟩
        by (simp add: F-Hiding D-Hiding) blast
    thus ⟨(s, X) ∈ F ?rhs⟩
    proof cases
      assume ⟨s1 ∈ D (P \ S)⟩
      then obtain t u
        where ** : ⟨front-tickFree u⟩ ⟨tickFree t⟩ ⟨s1 = trace-hide t (ev ' S) @ u⟩
          ⟨t ∈ D P ∨ (∃ g. isInfHiddenRun g P S ∧ t ∈ range g)⟩
        by (simp add: D-Hiding) blast
      have *** : ⟨front-tickFree (map (EvExt f) u) ∧ tickFree (map (EvExt f) t) ∧
        map (EvExt f) (trace-hide t (ev ' S)) @ map (EvExt f) u =
        trace-hide (map (EvExt f) t) (ev ' f ' S) @ (map (EvExt f) u)⟩
        by (simp add: EvExt-ftF EvExt-tF **(1, 2))
          (rule trace-hide-map-EvExt[symmetric];
          use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, simp split: event.split⟩)
      from **(4) show ⟨(s, X) ∈ F ?rhs⟩
      proof (elim disjE)
        assume ⟨t ∈ D P⟩
        hence $ : ⟨map (EvExt f) t ∈ D (Renaming P f)⟩
          apply (simp add: D-Renaming)
          using **(2) front-tickFree-Nil by blast
        show ⟨(s, X) ∈ F ?rhs⟩
          by (simp add: F-Hiding) (metis $ *(2) *(3) *** map-append)
      next
        assume ⟨∃ g. isInfHiddenRun g P S ∧ t ∈ range g⟩
        then obtain g where ⟨isInfHiddenRun g P S⟩ ⟨t ∈ range g⟩ by blast
        hence $ : ⟨isInfHiddenRun (λi. map (EvExt f) (g i)) (Renaming P f) (f '
S) ∧
          map (EvExt f) t ∈ range (λi. map (EvExt f) (g i))⟩
          apply (subst (1 2) trace-hide-map-EvExt,
            (use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, auto split:
event.split⟩)[2])
          by (simp add: strict-mono-map T-Renaming image-iff) (metis (mono-tags,
lifting))

```

```

    show  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
      by (simp add: F-Hiding)
         (metis (mono-tags, lifting)  $\$ *(2) ** (3) *** \text{map-append}$ )
  qed
next
assume  $\langle \exists t. s1 = \text{trace-hide } t (ev \text{ ' } S) \wedge (t, EvExt f \text{ -' } X \cup ev \text{ ' } S) \in \mathcal{F} P \rangle$ 
then obtain t where  $** : \langle s1 = \text{trace-hide } t (ev \text{ ' } S) \rangle$ 
                     $\langle (t, EvExt f \text{ -' } X \cup ev \text{ ' } S) \in \mathcal{F} P \rangle$  by blast
have  $*** : \langle EvExt f \text{ -' } X \cup EvExt f \text{ -' } ev \text{ ' } f \text{ ' } S = EvExt f \text{ -' } X \cup ev \text{ ' } S \rangle$ 
  by (use bij-f in  $\langle \text{unfold bij-def inj-on-def EvExt-def, auto simp add: image-iff}$ 
split: event.split  $\rangle$ )
  (metis (no-types, opaque-lifting) event.distinct(1) event.exhaust event.simps(4,
5) o-apply)
have  $\langle \text{map } (EvExt f) (\text{trace-hide } t (ev \text{ ' } S)) =$ 
       $\text{trace-hide } (\text{map } (EvExt f) t) (ev \text{ ' } f \text{ ' } S) \wedge$ 
       $(\text{map } (EvExt f) t, X \cup ev \text{ ' } f \text{ ' } S) \in \mathcal{F} (\text{Renaming } P f) \rangle$ 
  apply (intro conjI)
  apply (rule trace-hide-map-EvExt[symmetric];
  use bij-f in  $\langle \text{unfold bij-def inj-on-def EvExt-def, simp split: event.split} \rangle$ )
  apply (simp add: F-Renaming)
  using  $** (2) ***$  by auto
show  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
  apply (simp add: F-Hiding  $*(2) ** (1)$ )
  using  $\langle \text{?this} \rangle$  by blast
qed
qed
next
fix s X
assume same-div :  $\langle \mathcal{D} \text{ ?lhs} = \mathcal{D} \text{ ?rhs} \rangle$ 
assume  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
then consider  $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$ 
  |  $\langle \exists t. s = \text{trace-hide } t (ev \text{ ' } f \text{ ' } S) \wedge$ 
     $(t, X \cup ev \text{ ' } f \text{ ' } S) \in \mathcal{F} (\text{Renaming } P f) \rangle$ 
  by (simp add: F-Hiding D-Hiding) blast
thus  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
proof cases
  from D-F same-div show  $\langle s \in \mathcal{D} \text{ ?rhs} \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$  by blast
next
assume  $\langle \exists t. s = \text{trace-hide } t (ev \text{ ' } f \text{ ' } S) \wedge$ 
       $(t, X \cup ev \text{ ' } f \text{ ' } S) \in \mathcal{F} (\text{Renaming } P f) \rangle$ 
then obtain t
  where  $* : \langle s = \text{trace-hide } t (ev \text{ ' } f \text{ ' } S) \rangle$ 
         $\langle (t, X \cup ev \text{ ' } f \text{ ' } S) \in \mathcal{F} (\text{Renaming } P f) \rangle$  by blast
have  $** : \langle EvExt f \text{ -' } X \cup EvExt f \text{ -' } ev \text{ ' } f \text{ ' } S = EvExt f \text{ -' } X \cup ev \text{ ' } S \rangle$ 
  by (use bij-f in  $\langle \text{unfold bij-def inj-on-def EvExt-def, auto simp add: image-iff}$ 
split: event.split  $\rangle$ )
  (metis (no-types, opaque-lifting) event.distinct(1) event.exhaust event.simps(4,
5) o-apply)
have  $\langle (\exists s1. (s1, EvExt f \text{ -' } X \cup EvExt f \text{ -' } ev \text{ ' } f \text{ ' } S) \in \mathcal{F} P \wedge t = \text{map}$ 

```

```

(EvExt f) s1) ∨
  (∃ s1 s2. tickFree s1 ∧ front-tickFree s2 ∧ t = map (EvExt f) s1 @ s2 ∧
s1 ∈ D P)
  using *(2) by (simp add: F-Renaming)
  thus ⟨s, X⟩ ∈ F ?lhs
  proof (elim disjE exE conjE)
    fix s1
    assume ⟨s1, EvExt f -' X ∪ EvExt f -' ev ' f ' S⟩ ∈ F P ⟨t = map (EvExt
f) s1⟩
    hence ⟨(trace-hide s1 (ev ' S), EvExt f -' X) ∈ F (P \ S) ∧
      s = map (EvExt f) (trace-hide s1 (ev ' S))⟩
    apply (simp add: *(1) F-Hiding **, intro conjI)
    apply blast
    by (rule trace-hide-map-EvExt;
      use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, simp split: event.split⟩)
    show ⟨s, X⟩ ∈ F ?lhs
    apply (simp add: F-Renaming)
    using ⟨?this⟩ by blast
  next
  fix s1 s2
  assume ⟨tickFree s1⟩ ⟨front-tickFree s2⟩ ⟨t = map (EvExt f) s1 @ s2⟩ ⟨s1 ∈
D P⟩
  hence ⟨tickFree (trace-hide s1 (ev ' S)) ∧
    front-tickFree (trace-hide s2 (ev ' f ' S)) ∧
    s = map (EvExt f) (trace-hide s1 (ev ' S)) @ trace-hide s2 (ev ' f ' S) ∧
    trace-hide s1 (ev ' S) ∈ D (P \ S)⟩
  apply (simp add: F-Renaming *(1), intro conjI)
  subgoal using Hiding-tickFree by blast
  subgoal using Hiding-fronttickFree by blast
  apply (rule trace-hide-map-EvExt;
    use bij-f in ⟨unfold bij-def inj-on-def EvExt-def, simp split: event.split⟩)
  using elemDIselemHD by blast
  show ⟨s, X⟩ ∈ F ?lhs
  apply (simp add: F-Renaming)
  using ⟨?this⟩ by blast
qed
qed
qed

```

16.1.3 Renaming and Sync

Idem for the synchronization: when f is one to one, $\text{Renaming } (P \llbracket S \rrbracket Q)$ will behave as expected.

lemma *exists-map-antecedent-if-subset-range*:

⟨set $u \subseteq \text{range } f \implies \exists t. u = \text{map } f \ t$ ⟩

— In particular, when f is surjective or bijective.

proof (*induct* u)

show ⟨ $\exists t. \square = \text{map } f \ t$ ⟩ by *simp*

next

```

fix a u
assume prem : ⟨set (a # u) ⊆ range f⟩
  and hyp : ⟨set u ⊆ range f ⟹ ∃ t. u = map f t⟩
then obtain t where * : ⟨u = map f t⟩
  by (meson set-subset-Cons subset-trans)
from prem obtain x where ** : ⟨f x = a⟩ by auto
show ⟨∃ t. a # u = map f t⟩
proof (intro exI)
  show ⟨a # u = map f (x # t)⟩ by (simp add: * **)
qed
qed

lemma bij-map-setinterleaving-iff-setinterleaving :
  ⟨map f r setinterleaves ((map f t, map f u), f ' S) ⟷
  r setinterleaves ((t, u), S)⟩ if bij-f : ⟨bij f⟩
proof (induct ⟨(t, S, u)⟩ arbitrary; t u r rule: setinterleaving.induct)
  case 1
  thus ?case by simp
next
  case (2 y u)
  show ?case
  proof (cases ⟨y ∈ S⟩)
    show ⟨y ∈ S ⟹ ?case⟩ by simp
  next
    assume ⟨y ∉ S⟩
    hence ⟨f y ∉ f ' S⟩ by (metis bij-betw-imp-inj-on inj-image-mem-iff bij-f)
    with 2.hyps[OF ⟨y ∉ S⟩, of ⟨tl r⟩] show ?case
    by (cases r; simp add: ⟨y ∉ S⟩) (metis bij-pointE bij-f)
  qed
next
  case (3 x t)
  show ?case
  proof (cases ⟨x ∈ S⟩)
    show ⟨x ∈ S ⟹ ?case⟩ by simp
  next
    assume ⟨x ∉ S⟩
    hence ⟨f x ∉ f ' S⟩ by (metis bij-betw-imp-inj-on inj-image-mem-iff bij-f)
    with 3.hyps[OF ⟨x ∉ S⟩, of ⟨tl r⟩] show ?case
    by (cases r; simp add: ⟨x ∉ S⟩) (metis bij-pointE bij-f)
  qed
next
  case (4 x t y u)
  have * : ⟨x ≠ y ⟹ f x ≠ f y⟩ by (metis bij-pointE bij-f)
  have ** : ⟨f z ∈ f ' S ⟷ z ∈ S⟩ for z
    by (meson bij-betw-def inj-image-mem-iff bij-f)
  show ?case
  proof (cases ⟨x ∈ S⟩; cases ⟨y ∈ S⟩)
    from 4.hyps(1)[of ⟨tl r⟩] show ⟨x ∈ S ⟹ y ∈ S ⟹ ?case⟩

```

```

    by (cases r; simp add: *) (metis bij-pointE bij-f)
  next
    from 4.hyps(2)[of <tl r>] show <x ∈ S ⇒ y ∉ S ⇒ ?case>
    by (cases r; simp add: **) (metis bij-pointE bij-f)
  next
    from 4.hyps(5)[of <tl r>] show <x ∉ S ⇒ y ∈ S ⇒ ?case>
    by (cases r; simp add: **) (metis bij-pointE bij-f)
  next
    from 4.hyps(3, 4)[of <tl r>] show <x ∉ S ⇒ y ∉ S ⇒ ?case>
    by (cases r; simp add: **) (metis bij-pointE bij-f)
qed
qed

```

theorem *bij-Renaming-Sync*:

$\langle \text{Renaming } (P \llbracket S \rrbracket Q) f = \text{Renaming } P f \llbracket f ' S \rrbracket \text{Renaming } Q f \rangle$
 $(\text{is } \langle ?\text{lhs } P Q = ?\text{rhs } P Q \rangle \text{ if } \text{bij-}f : \langle \text{bij } f \rangle$

proof –

– Three intermediate results.

have *bij-EvExt-f* : $\langle \text{bij } (\text{EvExt } f) \rangle$

proof (unfold *bij-iff EvExt-def*, intro *allI ex1I*)

show $\langle (\text{case } (\text{EvExt } (\text{inv } f)) e \text{ of } \text{ev } x \Rightarrow (\text{ev } \circ f) x \mid \text{tick} \Rightarrow \text{tick}) = e \rangle \text{ for } e$

by (*simp add: EvExt-def split: event.split*)

(*meson bij-inv-eq-iff bij-f*)

next

show $\langle (\text{case } e' \text{ of } \text{ev } x \Rightarrow (\text{ev } \circ f) x \mid \text{tick} \Rightarrow \text{tick}) = e \Rightarrow e' = \text{EvExt } (\text{inv } f) e \rangle \text{ for } e e'$

by (*simp add: EvExt-def split: event.splits*)

(*metis bij-inv-eq-iff event.inject event.simps(3) bij-f*)

qed

have *EvExt-inv-f-is-inv-EvExt-f* : $\langle \text{inv } (\text{EvExt } f) = \text{EvExt } (\text{inv } f) \rangle$

proof –

have $\langle \text{EvExt } (\text{inv } f) \circ \text{EvExt } f = \text{id} \rangle$

by (*rule ext, simp add: EvExt-def split: event.split*)

(*meson bij-betw-imp-inj-on inv-f-eq bij-f*)

thus $\langle \text{inv } (\text{EvExt } f) = \text{EvExt } (\text{inv } f) \rangle$

by (*metis EvExt-comp EvExt-id bij-betw-def inv-unique-comp surj-iff that*)

qed

have *sets-S-eq* : $\langle (\text{EvExt } f) ' (\text{insert tick } (\text{ev } ' S)) = \text{insert tick } (\text{ev } ' f ' S) \rangle$

unfolding *EvExt-def image-def* **by** *auto*

show $\langle ?\text{lhs } P Q = ?\text{rhs } P Q \rangle$

proof (*subst Process-eq-spec-optimized, safe*)

fix *s*

assume $\langle s \in \mathcal{D} (?\text{lhs } P Q) \rangle$

then obtain *s1 s2* **where** $\ast : \langle \text{tickFree } s1 \rangle \langle \text{front-tickFree } s2 \rangle$

$\langle s = \text{map } (\text{EvExt } f) s1 @ s2 \rangle \langle s1 \in \mathcal{D} (P \llbracket S \rrbracket Q) \rangle$

by (*simp add: D-Renaming*) *blast*

from $\ast(4)$ **obtain** *t u r v*

```

where ** : ⟨front-tickFree v⟩ ⟨tickFree r ∨ v = []⟩
          ⟨s1 = r @ v⟩ ⟨r setinterleaves ((t, u), insert tick (ev ‘ S))⟩
          ⟨t ∈ D P ∧ u ∈ T Q ∨ t ∈ D Q ∧ u ∈ T P⟩
by (simp add: D-Sync) blast
{ fix t u P Q
  assume assms : ⟨r setinterleaves ((t, u), insert tick (ev ‘ S))⟩
          ⟨t ∈ D P⟩ ⟨u ∈ T Q⟩
  have ⟨map (EvExt f) r setinterleaves
        ((map (EvExt f) t, map (EvExt f) u), insert tick (ev ‘ f ‘ S))⟩
    by (metis assms(1) bij-EvExt-f
            bij-map-setinterleaving-iff-setinterleaving sets-S-eq)
  moreover have ⟨map (EvExt f) t ∈ D (Renaming P f)⟩
  apply (cases ⟨tickFree t⟩; simp add: D-Renaming)
  using assms(2) front-tickFree-Nil apply blast
  by (metis (no-types, lifting) assms(2) butlast-snoc front-tickFree-butlast
            front-tickFree-single map-EvExt-tick map-append
            nonTickFree-n-frontTickFree process-charn)
  moreover have ⟨map (EvExt f) u ∈ T (Renaming Q f)⟩
  using assms(3) by (simp add: T-Renaming) blast
  ultimately have ⟨s ∈ D (?rhs P Q)⟩
  by (simp add: D-Sync *(3) **(3))
      (metis *(1, 2) *(3) EvExt-tF front-tickFree-append tickFree-append)
} note *** = this

from *(4, 5) *** show ⟨s ∈ D (?rhs P Q)⟩
apply (elim disjE)
using *(4) *** apply blast
using *(4) *** by (subst Sync-commute) blast
next
fix s
assume ⟨s ∈ D (?rhs P Q)⟩
then obtain t u r v
  where * : ⟨front-tickFree v⟩ ⟨tickFree r ∨ v = []⟩ ⟨s = r @ v⟩
          ⟨r setinterleaves ((t, u), insert tick (ev ‘ f ‘ S))⟩
          ⟨t ∈ D (Renaming P f) ∧ u ∈ T (Renaming Q f) ∨
            t ∈ D (Renaming Q f) ∧ u ∈ T (Renaming P f)⟩
  by (simp add: D-Sync) blast

{ fix t u P Q
  assume assms : ⟨r setinterleaves ((t, u), insert tick (ev ‘ f ‘ S))⟩
          ⟨t ∈ D (Renaming P f)⟩ ⟨u ∈ T (Renaming Q f)⟩
  have ** : ⟨(map (inv (EvExt f)) r) setinterleaves
            ((map (inv (EvExt f)) t, map (inv (EvExt f)) u), insert tick (ev ‘
S))⟩
    by (metis (no-types) assms(1) bij-EvExt-f bij-betw-imp-inj-on
            bij-betw-inv-into image-inv-f-f sets-S-eq
            bij-map-setinterleaving-iff-setinterleaving)
  have ⟨map (EvExt (inv f)) t ∈ D (Renaming (Renaming P f) (inv f))⟩
  by (subst D-Renaming, simp)

```

```

    (metis EvExt-ftF append.right-neutral assms(2) front-tickFree-implies-tickFree

      front-tickFree-single map-append nonTickFree-n-frontTickFree
process-charn)
  hence *** : ⟨map (inv (EvExt f)) t ∈ D P⟩
  by (simp add: Renaming-inv bij-is-inj bij-f EvExt-inv-f-is-inv-EvExt-f)
  have ⟨map (EvExt (inv f)) u ∈ T (Renaming (Renaming Q f) (inv f))⟩
  using assms(3) by (subst T-Renaming, simp) blast
  hence **** : ⟨map (inv (EvExt f)) u ∈ T Q⟩
  by (simp add: Renaming-inv bij-is-inj bij-f EvExt-inv-f-is-inv-EvExt-f)
  have ***** : ⟨map (EvExt f ∘ inv (EvExt f)) r = r⟩
  by (metis bij-EvExt-f bij-betw-imp-surj list.map-id surj-iff)
  have ⟨s ∈ D (?lhs P Q)⟩
  proof (cases ⟨tickFree r⟩)
    assume ⟨tickFree r⟩
    have $ : ⟨r @ v = map (EvExt f) (map (inv (EvExt f)) r) @ v⟩
    by (simp add: *****)
    show ⟨s ∈ D (?lhs P Q)⟩
    apply (simp add: D-Renaming D-Sync *(3))
    by (metis $(1) ** *** **** EvExt-tF ⟨tickFree r⟩
      append.right-neutral append-same-eq front-tickFree-Nil)
  next
  assume ⟨¬ tickFree r⟩
  then obtain r' where $ : ⟨r = r' @ [tick]⟩ ⟨tickFree r'⟩
  by (metis D-imp-front-tickFree assms front-tickFree-implies-tickFree ftf-Sync
    is-processT2-TR nonTickFree-n-frontTickFree)
  then obtain t' u'
  where $$ : ⟨t = t' @ [tick]⟩ ⟨u = u' @ [tick]⟩
  using SyncWithTick-imp-NTF D-T assms by blast
  hence $$$ : ⟨(map (inv (EvExt f)) r') setinterleaves
    ((map (inv (EvExt f)) t', map (inv (EvExt f)) u'),
    insert tick (ev ' S))⟩
  by (metis $(1) assms(1) bij-EvExt-f bij-betw-imp-inj-on
    bij-betw-inv-into bij-map-setinterleaving-iff-setinterleaving
    butlast-snoc ftf-Sync32 image-inv-f-f sets-S-eq)
  from *** $$ (1) have *** : ⟨map (inv (EvExt f)) t' ∈ D P⟩
  by simp (metis EvExt-inv-f-is-inv-EvExt-f is-processT9 tick-eq-EvExt)
  from **** $$ (2) have **** : ⟨map (inv (EvExt f)) u' ∈ T Q⟩
  using is-processT3-ST by simp blast
  have $$$$ : ⟨r = map (EvExt f) (map (inv (EvExt f)) r') @ [tick]⟩
  using $ ***** by auto
  show ⟨s ∈ D (?lhs P Q)⟩
  by (simp add: D-Renaming D-Sync *(3) $$$)
    (metis $(1) $(2) $$$ $$$$ *(2) *** **** EvExt-tF ⟨¬ tickFree r⟩
      append.right-neutral append-same-eq front-tickFree-Nil front-tickFree-single)
  qed
} note ** = this
show ⟨s ∈ D (?lhs P Q)⟩ by (metis *(4, 5) ** Sync-commute)
next

```


fix $s\ X$
assume $\text{same-div} : \langle \mathcal{D} (?lhs\ P\ Q) = \mathcal{D} (?rhs\ P\ Q) \rangle$
assume $\langle (s, X) \in \mathcal{F} (?lhs\ P\ Q) \rangle$
then consider $\langle s \in \mathcal{D} (?lhs\ P\ Q) \rangle$
| $\langle \exists s1. (s1, \text{EvExt}\ f\ -' X) \in \mathcal{F} (P\ \llbracket S \rrbracket\ Q) \wedge s = \text{map}\ (\text{EvExt}\ f)\ s1 \rangle$
by (*simp add: F-Renaming D-Renaming*) *blast*
thus $\langle (s, X) \in \mathcal{F} (?rhs\ P\ Q) \rangle$
proof cases
from *same-div D-F* **show** $\langle s \in \mathcal{D} (?lhs\ P\ Q) \implies (s, X) \in \mathcal{F} (?rhs\ P\ Q) \rangle$ **by**
blast
next
assume $\langle \exists s1. (s1, \text{EvExt}\ f\ -' X) \in \mathcal{F} (P\ \llbracket S \rrbracket\ Q) \wedge s = \text{map}\ (\text{EvExt}\ f)\ s1 \rangle$
then obtain $s1$ **where** $*$: $\langle (s1, \text{EvExt}\ f\ -' X) \in \mathcal{F} (P\ \llbracket S \rrbracket\ Q) \rangle$
 $\langle s = \text{map}\ (\text{EvExt}\ f)\ s1 \rangle$ **by** *blast*
from $*(1)$ **consider** $\langle s1 \in \mathcal{D} (P\ \llbracket S \rrbracket\ Q) \rangle$
| $\langle \exists t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q.$
 $(t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \wedge$
 $s1\ \text{setinterleaves}\ ((t\text{-}P, t\text{-}Q), \text{insert}\ \text{tick}\ (\text{ev}\ ' S)) \wedge$
 $\text{EvExt}\ f\ -' X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert}\ \text{tick}\ (\text{ev}\ ' S) \cup X\text{-}P \cap X\text{-}Q \rangle$
by (*simp add: F-Sync D-Sync*) *blast*
thus $\langle (s, X) \in \mathcal{F} (?rhs\ P\ Q) \rangle$
proof cases
assume $\langle s1 \in \mathcal{D} (P\ \llbracket S \rrbracket\ Q) \rangle$
hence $\langle s \in \mathcal{D} (?lhs\ P\ Q) \rangle$
apply (*cases tickFree s1*); *simp add: D-Renaming *(2)*
using *front-tickFree-Nil apply blast*
by (*metis (no-types, lifting) EvExt-ftF butlast-snoc front-tickFree-butlast*
front-tickFree-single map-butlast nonTickFree-n-frontTickFree
process-charn)
with *same-div D-F* **show** $\langle (s, X) \in \mathcal{F} (?rhs\ P\ Q) \rangle$ **by** *blast*
next
assume $\langle \exists t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q.$
 $(t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \wedge$
 $s1\ \text{setinterleaves}\ ((t\text{-}P, t\text{-}Q), \text{insert}\ \text{tick}\ (\text{ev}\ ' S)) \wedge$
 $\text{EvExt}\ f\ -' X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert}\ \text{tick}\ (\text{ev}\ ' S) \cup X\text{-}P \cap X\text{-}Q \rangle$
then obtain $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$
where $**$: $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \rangle$
 $\langle s1\ \text{setinterleaves}\ ((t\text{-}P, t\text{-}Q), \text{insert}\ \text{tick}\ (\text{ev}\ ' S)) \rangle$
 $\langle \text{EvExt}\ f\ -' X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert}\ \text{tick}\ (\text{ev}\ ' S) \cup X\text{-}P \cap$
 $X\text{-}Q \rangle$ **by** *blast*
have $\langle (\text{map}\ (\text{EvExt}\ f)\ t\text{-}P, (\text{EvExt}\ f)\ ' X\text{-}P) \in \mathcal{F} (\text{Renaming}\ P\ f) \rangle$
by (*simp add: F-Renaming*)
 $(\text{metis}\ ** (1)\ \text{bij-EvExt-f}\ \text{bij-betw-imp-inj-on}\ \text{inj-vimage-image-eq})$
moreover have $\langle (\text{map}\ (\text{EvExt}\ f)\ t\text{-}Q, (\text{EvExt}\ f)\ ' X\text{-}Q) \in \mathcal{F} (\text{Renaming}\ Q$
 $f) \rangle$
by (*simp add: F-Renaming*)
 $(\text{metis}\ ** (2)\ \text{bij-EvExt-f}\ \text{bij-betw-imp-inj-on}\ \text{inj-vimage-image-eq})$
moreover have $\langle s\ \text{setinterleaves}\ ((\text{map}\ (\text{EvExt}\ f)\ t\text{-}P, \text{map}\ (\text{EvExt}\ f)\ t\text{-}Q),$
 $\text{insert}\ \text{tick}\ (\text{ev}\ ' f\ ' S)) \rangle$

by (*metis* $*(2)$ $*(3)$ *bij-EvExt-f sets-S-eq*
bij-map-setinterleaving-iff-setinterleaving)
moreover have $\langle X = ((\text{EvExt } f) \text{ ` } X\text{-}P \cup (\text{EvExt } f) \text{ ` } X\text{-}Q) \cap \text{insert tick}$
 $(\text{ev ` } f \text{ ` } S) \cup$
 $(\text{EvExt } f) \text{ ` } X\text{-}P \cap (\text{EvExt } f) \text{ ` } X\text{-}Q \rangle$
by (*metis* $*(4)$ *bij-EvExt-f bij-betw-def image-Int image-Un*
image-vimage-eq inf-top.right-neutral sets-S-eq)
ultimately show $\langle (s, X) \in \mathcal{F} (?rhs \ P \ Q) \rangle$
by (*simp add: F-Sync*) *blast*
qed
qed
next
fix $s \ X$
assume *same-div* : $\langle \mathcal{D} (?lhs \ P \ Q) = \mathcal{D} (?rhs \ P \ Q) \rangle$
assume $\langle (s, X) \in \mathcal{F} (?rhs \ P \ Q) \rangle$
then consider $\langle s \in \mathcal{D} (?rhs \ P \ Q) \rangle$
| $\langle \exists t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q.$
 $(t\text{-}P, X\text{-}P) \in \mathcal{F} (\text{Renaming } P \ f) \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F} (\text{Renaming } Q \ f) \wedge$
 $s \text{ setinterleaves } ((t\text{-}P, t\text{-}Q), \text{insert tick } (\text{ev ` } f \text{ ` } S)) \wedge$
 $X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert tick } (\text{ev ` } f \text{ ` } S) \cup X\text{-}P \cap X\text{-}Q \rangle$
by (*simp add: F-Sync D-Sync*) *blast*
thus $\langle (s, X) \in \mathcal{F} (?lhs \ P \ Q) \rangle$
proof cases
from *same-div D-F* **show** $\langle s \in \mathcal{D} (?rhs \ P \ Q) \implies (s, X) \in \mathcal{F} (?lhs \ P \ Q) \rangle$ **by**
blast
next
assume $\langle \exists t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q.$
 $(t\text{-}P, X\text{-}P) \in \mathcal{F} (\text{Renaming } P \ f) \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F} (\text{Renaming } Q \ f) \wedge$
 $s \text{ setinterleaves } ((t\text{-}P, t\text{-}Q), \text{insert tick } (\text{ev ` } f \text{ ` } S)) \wedge$
 $X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert tick } (\text{ev ` } f \text{ ` } S) \cup X\text{-}P \cap X\text{-}Q \rangle$
then obtain $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$
where $*$: $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} (\text{Renaming } P \ f) \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} (\text{Renaming } Q \ f) \rangle$
 $\langle s \text{ setinterleaves } ((t\text{-}P, t\text{-}Q), \text{insert tick } (\text{ev ` } f \text{ ` } S)) \rangle$
 $\langle X = (X\text{-}P \cup X\text{-}Q) \cap \text{insert tick } (\text{ev ` } f \text{ ` } S) \cup X\text{-}P \cap X\text{-}Q \rangle$ **by** *blast*
from $*(1, 2)$ **consider** $\langle t\text{-}P \in \mathcal{D} (\text{Renaming } P \ f) \vee t\text{-}Q \in \mathcal{D} (\text{Renaming } Q \ f) \rangle$
| $\langle \exists t\text{-}P1 \ t\text{-}Q1. (t\text{-}P1, \text{EvExt } f \text{ ` } X\text{-}P) \in \mathcal{F} \ P \wedge t\text{-}P = \text{map } (\text{EvExt } f) \ t\text{-}P1$
 \wedge
 $(t\text{-}Q1, \text{EvExt } f \text{ ` } X\text{-}Q) \in \mathcal{F} \ Q \wedge t\text{-}Q = \text{map } (\text{EvExt } f) \ t\text{-}Q1 \rangle$
by (*simp add: F-Renaming D-Renaming*) *blast*
thus $\langle (s, X) \in \mathcal{F} (?lhs \ P \ Q) \rangle$
proof cases
assume $\langle t\text{-}P \in \mathcal{D} (\text{Renaming } P \ f) \vee t\text{-}Q \in \mathcal{D} (\text{Renaming } Q \ f) \rangle$
hence $\langle s \in \mathcal{D} (?rhs \ P \ Q) \rangle$
apply (*simp add: D-Sync*)
using $*(1, 2, 3)$ *F-T Sync.sym front-tickFree-Nil* **by** *blast*
with *same-div D-F* **show** $\langle (s, X) \in \mathcal{F} (?lhs \ P \ Q) \rangle$ **by** *blast*
next

assume $\langle \exists t-P1\ t-Q1. (t-P1, EvExt\ f\ -' X-P) \in \mathcal{F}\ P \wedge t-P = map\ (EvExt\ f)\ t-P1 \wedge$
 $(t-Q1, EvExt\ f\ -' X-Q) \in \mathcal{F}\ Q \wedge t-Q = map\ (EvExt\ f)\ t-Q1 \rangle$
then obtain $t-P1\ t-Q1$
where $** : \langle (t-P1, EvExt\ f\ -' X-P) \in \mathcal{F}\ P \rangle \langle t-P = map\ (EvExt\ f)\ t-P1 \rangle$
 $\langle (t-Q1, EvExt\ f\ -' X-Q) \in \mathcal{F}\ Q \rangle \langle t-Q = map\ (EvExt\ f)\ t-Q1 \rangle$
by *blast*
from $**(2, 4)$ **have** $*** : \langle t-P1 = map\ (inv\ (EvExt\ f))\ t-P \rangle$
 $\langle t-Q1 = map\ (inv\ (EvExt\ f))\ t-Q \rangle$
by (*simp, metis bij-EvExt-f bij-betw-imp-inj-on inv-o-cancel list.map-id*)
have $**** : \langle map\ (EvExt\ f)\ (map\ (inv\ (EvExt\ f))\ s) = s \rangle$
by (*metis bij-EvExt-f bij-betw-imp-surj list.map-comp list.map-id surj-iff*)
from *bij-map-setinterleaving-iff-setinterleaving*
 $[of\ \langle inv\ (EvExt\ f) \rangle s\ t-P\ \langle insert\ tick\ (ev\ ' f\ ' S) \rangle t-Q, simplified\ *(3)]$
have $\langle map\ (inv\ (EvExt\ f))\ s\ setinterleaves\ ((t-P1, t-Q1), insert\ tick\ (ev\ ' S)) \rangle$
by (*metis *** bij-EvExt-f bij-betw-imp-inj-on*
bij-betw-inv-into image-inv-f-f sets-S-eq)
moreover have $\langle EvExt\ f\ -' X = (EvExt\ f\ -' X-P \cup EvExt\ f\ -' X-Q) \cap$
 $insert\ tick\ (ev\ ' S) \cup$
 $EvExt\ f\ -' X-P \cap EvExt\ f\ -' X-Q \rangle$
by (*metis *(4) bij-EvExt-f bij-betw-imp-inj-on*
inj-vimage-image-eq sets-S-eq vimage-Int vimage-Un)
ultimately show $\langle (s, X) \in \mathcal{F}\ (?lhs\ P\ Q) \rangle$
by (*simp add: F-Renaming F-Sync*)
 $(metis\ ** (1, 3) ****)$
qed
qed
qed
qed

16.2 Hiding and Mprefix

We already have a way to distribute the *Hiding* operator on the *Mprefix* operator with $S \cap A = \{\} \implies Mprefix\ A\ ?P \setminus S = \square x \in A \rightarrow (?P\ x \setminus S)$. But this is only usable when $A \cap S = \{\}$. With the (\triangleright) operator, we can now handle the case $A \cap S \neq \{\}$.

16.2.1 Two intermediate Results

lemma *D-Hiding-Mprefix-dir2*:

assumes $\langle s \in \mathcal{D}\ (\square a \in A - S \rightarrow (P\ a \setminus S)) \rangle$

shows $\langle s \in \mathcal{D}\ (\square a \in A \rightarrow P\ a \setminus S) \rangle$

proof –

from *assms* **obtain** $a\ s'$

where $* : \langle a \in A \rangle \langle a \notin S \rangle \langle s = ev\ a \ \# \ s' \rangle \langle s' \in \mathcal{D}\ (P\ a \setminus S) \rangle$

by (*auto simp add: D-Mprefix*) (*metis list.collapse*)

from $*(4)$ **obtain** $t u$
where $** : \langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle \langle s' = \text{trace-hide } t (ev \text{ ' } S) @ u \rangle$
 $\langle t \in \mathcal{D} (P a) \vee (\exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f) \rangle$
by (*simp add: D-Hiding*) *blast*
from $*(4)$ **show** $\langle s \in \mathcal{D} (\Box a \in A \rightarrow P a \setminus S) \rangle$
proof (*elim disjE*)
assume $\langle t \in \mathcal{D} (P a) \rangle$
hence $\langle \text{tickFree } (ev a \# t) \wedge s = \text{trace-hide } (ev a \# t) (ev \text{ ' } S) @ u \wedge$
 $ev a \# t \in \mathcal{D} (Mprefix A P) \rangle$
by (*simp add: *(1, 2, 3) *(2, 3) image-iff[of ev a] D-Mprefix*)
show $\langle s \in \mathcal{D} (Mprefix A P \setminus S) \rangle$
apply (*simp add: D-Hiding*)
using $*(1)$ $\langle ?this \rangle$ **by** *blast*
next
assume $\langle \exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f \rangle$
then obtain f **where** $*** : \langle \text{isInfHiddenRun } f (P a) S \rangle \langle t \in \text{range } f \rangle$ **by** *blast*
hence $\langle \text{tickFree } (ev a \# t) \wedge s = \text{trace-hide } (ev a \# t) (ev \text{ ' } S) @ u \wedge$
 $\text{isInfHiddenRun } (\lambda i. ev a \# f i) (Mprefix A P) S \wedge$
 $ev a \# t \in \text{range } (\lambda i. ev a \# f i) \rangle$
by (*auto simp add: *(1, 2, 3) *(2, 3) image-iff[of ev a]*)
 $T\text{-Mprefix less-cons strict-mono-Suc-iff}$
show $\langle s \in \mathcal{D} (Mprefix A P \setminus S) \rangle$
apply (*simp add: D-Hiding*)
using $*(1)$ $\langle ?this \rangle$ **by** *blast*
qed
qed

lemma *F-Hiding-Mprefix-dir2*:

assumes $\langle s \neq [] \rangle$ **and** $\langle (s, X) \in \mathcal{F} (\Box a \in A - S \rightarrow (P a \setminus S)) \rangle$
shows $\langle (s, X) \in \mathcal{F} (\Box a \in A \rightarrow P a \setminus S) \rangle$

proof –

from *assms* **obtain** $a s'$

where $* : \langle a \in A \rangle \langle a \notin S \rangle \langle s = ev a \# s' \rangle \langle (s', X) \in \mathcal{F} (P a \setminus S) \rangle$

by (*auto simp add: F-Mprefix (metis list.collapse)*)

from $*(4)$ **consider** $\langle s' \in \mathcal{D} (P a \setminus S) \rangle$

$| \langle \exists t. s' = \text{trace-hide } t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} (P a) \rangle$

by (*simp add: F-Hiding D-Hiding*) *blast*

thus $\langle (s, X) \in \mathcal{F} (Mprefix A P \setminus S) \rangle$

proof *cases*

assume $\langle s' \in \mathcal{D} (P a \setminus S) \rangle$

then obtain $t u$

where $** : \langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle$

$\langle s' = \text{trace-hide } t (ev \text{ ' } S) @ u \rangle$

$\langle t \in \mathcal{D} (P a) \vee (\exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f) \rangle$

by (*simp add: D-Hiding*) *blast*

from $*(4)$ **show** $\langle (s, X) \in \mathcal{F} (Mprefix A P \setminus S) \rangle$

proof (*elim disjE*)

assume $\langle t \in \mathcal{D} (P a) \rangle$

```

hence ⟨tickFree (ev a # t) ∧ s = trace-hide (ev a # t) (ev ' S) @ u ∧
      ev a # t ∈ D (Mprefix A P)⟩
  by (simp add: *(1, 2, 3) **(2, 3) D-Mprefix image-iff[of ⟨ev a⟩])
show ⟨(s, X) ∈ F (Mprefix A P \ S)⟩
  apply (simp add: F-Hiding)
  using *(1) ⟨?this⟩ by blast
next
assume ⟨∃ f. isInfHiddenRun f (P a) S ∧ t ∈ range f⟩
then obtain f where ⟨isInfHiddenRun f (P a) S⟩ ⟨t ∈ range f⟩ by blast
hence ⟨tickFree (ev a # t) ∧ s = trace-hide (ev a # t) (ev ' S) @ u ∧
      (isInfHiddenRun (λi. ev a # f i) (Mprefix A P) S ∧
       ev a # t ∈ range (λi. ev a # f i))⟩
  by (auto simp add: *(1, 2, 3) **(2, 3) less-cons monotone-on-def T-Mprefix)
show ⟨(s, X) ∈ F (Mprefix A P \ S)⟩
  apply (simp add: F-Hiding)
  using *(1) ⟨?this⟩ by blast
qed
next
assume ⟨∃ t. s' = trace-hide t (ev ' S) ∧ (t, X ∪ ev ' S) ∈ F (P a)⟩
then obtain t where **: ⟨s' = trace-hide t (ev ' S)⟩
      ⟨(t, X ∪ ev ' S) ∈ F (P a)⟩ by blast
have ⟨s = trace-hide (ev a # t) (ev ' S) ∧ (ev a # t, X ∪ ev ' S) ∈ F (Mprefix
A P)⟩
  by (simp add: *(1, 2, 3) ** F-Mprefix image-iff)
show ⟨(s, X) ∈ F (Mprefix A P \ S)⟩
  apply (simp add: F-Hiding)
  using ⟨?this⟩ by blast
qed
qed

```

16.2.2 Hiding and Mprefix for disjoint Sets

Now we can give a more readable proof of the following result (already proven in *HOL-CSP.CSP-Laws*).

theorem *Hiding-Mprefix-disjoint*:

⟨ $\Box a \in A \rightarrow P a \setminus S = \Box a \in A \rightarrow (P a \setminus S)$ ⟩
 (is ⟨?lhs = ?rhs⟩) if disjoint: ⟨ $A \cap S = \{\}$ ⟩

proof (subst Process-eq-spec-optimized, safe)

fix s

assume ⟨s ∈ D ?lhs⟩

then obtain t u

where *: ⟨front-tickFree u⟩ ⟨tickFree t⟩ ⟨s = trace-hide t (ev ' S) @ u⟩
 ⟨t ∈ D (Mprefix A P) ∨
 (∃ f. isInfHiddenRun f (Mprefix A P) S ∧ t ∈ range f)⟩

by (simp add: D-Hiding) blast

from *(4) show ⟨s ∈ D ?rhs⟩

proof (elim disjE)

assume ⟨t ∈ D (Mprefix A P)⟩

then obtain a t' where **: ⟨a ∈ A⟩ ⟨a ∉ S⟩ ⟨t = ev a # t'⟩ ⟨t' ∈ D (P a)⟩

```

    by (auto simp add: D-Mprefix)
      (metis list.exhaust-sel disjoint-iff disjoint)
  have ⟨front-tickFree u ∧ tickFree t' ∧
    trace-hide t' (ev ' S) @ u = trace-hide t' (ev ' S) @ u ∧ t' ∈ D (P a)⟩
    apply (simp add: *(1) *(4))
    using *(2) *(3) tickFree-Cons by blast
  show ⟨s ∈ D ?rhs⟩
    apply (simp add: D-Mprefix *(3) *(1, 2, 3) image-iff[of ⟨ev -⟩] D-Hiding)
    using ⟨?this⟩ by blast
next
assume ⟨∃ f. isInfHiddenRun f (Mprefix A P) S ∧ t ∈ range f⟩
then obtain f where **: ⟨isInfHiddenRun f (Mprefix A P) S⟩
  ⟨t ∈ range f⟩ by blast
from *(1) T-Mprefix obtain a
  where *** : ⟨a ∈ A⟩ ⟨a ∉ S⟩ ⟨f (Suc 0) ≠ []⟩ ⟨hd (f (Suc 0)) = ev a⟩
  by (simp add: T-Mprefix)
  (metis disjoint-iff disjoint nil-less strict-mono-Suc-iff)
from *(1)[THEN conjunct2, THEN conjunct2, rule-format, of 1]
  *(1)[simplified isInfHiddenRun-1] *** (1, 4) disjoint
have **** : ⟨f j ≠ [] ∧ hd (f j) = ev a⟩ for j
  using *(1)[THEN conjunct2, THEN conjunct1, rule-format, of j]
  apply (cases ⟨f 1⟩; simp add: T-Mprefix *(3) split: if-split-asm)
  by (metis Nil-is-append-conv filter.simps(1)
    hd-append2 list.distinct(1) list.sel(1)) blast
then obtain t' where ⟨t = ev a # t'⟩
  by (metis *(2) list.exhaust-sel rangeE)
hence ⟨tickFree t' ∧ trace-hide t' (ev ' S) @ u = trace-hide t' (ev ' S) @ u ∧
  isInfHiddenRun (λi. tl (f i)) (P a) S ∧ t' ∈ range (λi. tl (f i))⟩
  apply (simp, intro conjI)
  subgoal using *(2) tickFree-Cons by blast
  subgoal by (meson *(1) **** less-tail strict-mono-Suc-iff)
  subgoal using *(1) by (simp add: T-Mprefix ****)
  subgoal by (metis (no-types, lifting) *(1) **** filter.simps(2) list.exhaust-sel
    list.sel(3))
  by (metis (no-types, lifting) *(2) image-iff list.sel(3))
show ⟨s ∈ D ?rhs⟩
  apply (simp add: D-Mprefix *(3) image-iff[of ⟨ev a⟩] *(1, 2) D-Hiding ⟨t
    = ev a # t'⟩)
  using *(1) ⟨?this⟩ by blast
qed
next
show ⟨s ∈ D ?rhs ⟹ s ∈ D ?lhs⟩ for s
  by (simp add: D-Hiding-Mprefix-dir2 Diff-triv that)
next
fix s X
assume same-div : ⟨D ?lhs = D ?rhs⟩
assume ⟨(s, X) ∈ F ?lhs⟩
then consider ⟨s ∈ D ?lhs⟩
  | ⟨∃ t. s = trace-hide t (ev ' S) ∧ (t, X ∪ ev ' S) ∈ F (Mprefix A P)⟩

```

```

  by (simp add: F-Hiding D-Hiding) blast
thus  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
proof cases
  from same-div D-F show  $\langle s \in \mathcal{D} \text{ ?lhs} \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$  by blast
next
  assume  $\langle \exists t. s = \text{trace-hide } t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$ 
  then obtain  $t$  where  $*$  :  $\langle s = \text{trace-hide } t (ev \text{ ' } S) \rangle$ 
     $\langle (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$  by blast
  show  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
  proof (cases  $\langle t = [] \rangle$ )
    from  $*$  show  $\langle t = [] \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
    by (auto simp add: F-Mprefix)
  next
    assume  $\langle t \neq [] \rangle$ 
    with  $*(2)$  disjoint obtain  $a t'$ 
      where  $**$  :  $\langle a \in A \rangle \langle a \notin S \rangle \langle t = ev a \# t' \rangle \langle (t', X \cup ev \text{ ' } S) \in \mathcal{F} (P a) \rangle$ 
      by (cases  $t$ , auto simp add: F-Mprefix)
    show  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
    apply (simp add: F-Mprefix  $*(1)$   $**(1, 2, 3)$  image-iff[of  $\langle ev a \rangle$ ])
    by (simp add: F-Hiding, rule disjI1, auto simp add:  $**(4)$ )
  qed
qed
next
  from disjoint show  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \implies (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$  for  $s X$ 
  apply (cases  $\langle s = [] \rangle$ )
  apply (simp add: F-Mprefix F-Hiding disjoint-iff,
    metis event.inject filter.simps(1) imageE)
  by (simp add: Diff-triv F-Hiding-Mprefix-dir2)
qed

```

And we obtain a similar result when adding a (\triangleright) in the expression.

theorem *Hiding-Mprefix-Sliding-disjoint*:

$\langle ((\Box a \in A \rightarrow P a) \triangleright Q) \setminus S = (\Box a \in A \rightarrow (P a \setminus S)) \triangleright (Q \setminus S) \rangle$
 if disjoint: $\langle A \cap S = \{\} \rangle$

proof (*subst Hiding-Mprefix-disjoint[OF disjoint, symmetric]*)

show $\langle ((\Box a \in A \rightarrow P a) \triangleright Q) \setminus S = (\Box a \in A \rightarrow P a \setminus S) \triangleright (Q \setminus S) \rangle$
 (is $\langle \text{?lhs} = \text{?rhs} \rangle$)

proof (*subst Process-eq-spec-optimized, safe*)

fix s

assume $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$

hence $\langle s \in \mathcal{D} (Mprefix A P \sqcap Q \setminus S) \rangle$

by (*simp add: D-Hiding D-Sliding D-Ndet T-Sliding T-Ndet*)

thus $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$

by (*rule set-rev-mp*)

(*simp add: D-Ndet D-Sliding Hiding-Ndet subsetI*)

next

fix s

assume $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$

hence $\langle s \in \mathcal{D} (Q \setminus S) \vee s \in \mathcal{D} (\Box a \in A \rightarrow P a \setminus S) \rangle$

by (*simp add: Hiding-Mprefix-disjoint[OF disjoint]*
D-Ndet D-Sliding) *blast*
thus $\langle s \in \mathcal{D} \ ?lhs \rangle$
 by (*auto simp add: D-Hiding D-Sliding T-Sliding*)
next
fix $s \ X$
assume *same-div* : $\langle \mathcal{D} \ ?lhs = \mathcal{D} \ ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} \ ?lhs \rangle$
then consider $\langle s \in \mathcal{D} \ ?lhs \rangle$
 | $\langle \exists t. s = \text{trace-hide } t \ (ev \ ' S) \wedge (t, X \cup ev \ ' S) \in \mathcal{F} \ (Mprefix \ A \ P \triangleright \ Q) \rangle$
 by (*simp add: F-Hiding D-Hiding*) *blast*
thus $\langle (s, X) \in \mathcal{F} \ ?rhs \rangle$
proof cases
from *same-div D-F* **show** $\langle s \in \mathcal{D} \ ?lhs \implies (s, X) \in \mathcal{F} \ ?rhs \rangle$ by *blast*
next
assume $\langle \exists t. s = \text{trace-hide } t \ (ev \ ' S) \wedge$
 $(t, X \cup ev \ ' S) \in \mathcal{F} \ (Mprefix \ A \ P \triangleright \ Q) \rangle$
then obtain t
where $*$: $\langle s = \text{trace-hide } t \ (ev \ ' S)$
 $\langle (t, X \cup ev \ ' S) \in \mathcal{F} \ (Mprefix \ A \ P \triangleright \ Q) \rangle$ by *blast*
from $*$ (2) **consider** $\langle (t, X \cup ev \ ' S) \in \mathcal{F} \ Q \rangle$
 | $\langle t \neq [] \rangle \langle (t, X \cup ev \ ' S) \in \mathcal{F} \ (Mprefix \ A \ P) \rangle$
 by (*simp add: F-Sliding D-Mprefix*) *blast*
thus $\langle (s, X) \in \mathcal{F} \ ?rhs \rangle$
proof cases
have $\langle (t, X \cup ev \ ' S) \in \mathcal{F} \ Q \implies (s, X) \in \mathcal{F} \ (Q \setminus S) \rangle$
 by (*auto simp add: F-Hiding *(1)*)
thus $\langle (t, X \cup ev \ ' S) \in \mathcal{F} \ Q \implies (s, X) \in \mathcal{F} \ ?rhs \rangle$
 by (*simp add: F-Ndet F-Sliding *(1)*)
next
assume *assms* : $\langle t \neq [] \rangle \langle (t, X \cup ev \ ' S) \in \mathcal{F} \ (Mprefix \ A \ P) \rangle$
with disjoint **have** $\langle \text{trace-hide } t \ (ev \ ' S) \neq [] \rangle$
 by (*cases t, auto simp add: F-Mprefix*)
also have $\langle (s, X) \in \mathcal{F} \ (Mprefix \ A \ P \setminus S) \rangle$
 using *assms* by (*auto simp: F-Hiding *(1)*)
ultimately show $\langle (s, X) \in \mathcal{F} \ ?rhs \rangle$
 by (*simp add: F-Sliding *(1)*)
qed
qed
next
have $*$: $\langle t \in \mathcal{T} \ (Mprefix \ A \ P) \implies \text{trace-hide } t \ (ev \ ' S) = [] \longleftrightarrow t = [] \rangle$ for t
 using *disjoint* by (*cases t, auto simp add: T-Mprefix*)
have $**$: $\langle [] \notin \mathcal{D} \ (Mprefix \ A \ P \setminus S) \rangle$
apply (*rule ccontr, simp add: D-Hiding*)
by (*metis (mono-tags, lifting) * NT-ND Nil-Nin-D-Mprefix*
UNIV-I f-inv-into-f lessI nless-le strict-mono-on-eqD)
fix $s \ X$
assume *same-div* : $\langle \mathcal{D} \ ?lhs = \mathcal{D} \ ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} \ ?rhs \rangle$


```

with ** consider ⟨s, X⟩ ∈  $\mathcal{F} (Q \setminus S)$ 
  | ⟨s ≠ []⟩ ⟨s, X⟩ ∈  $\mathcal{F} (Mprefix A P \setminus S)$ 
  by (simp add: Hiding-Mprefix-disjoint[OF disjoint] F-Sliding D-Mprefix) blast
thus ⟨s, X⟩ ∈  $\mathcal{F} ?lhs$ 
proof cases
  assume ⟨s, X⟩ ∈  $\mathcal{F} (Q \setminus S)$ 
  then consider ⟨s ∈  $\mathcal{D} (Q \setminus S)$ ⟩
    | ⟨∃ t. s = trace-hide t (ev ' S) ∧ (t, X ∪ ev ' S) ∈  $\mathcal{F} Q$ ⟩
    by (simp add: F-Hiding D-Hiding) blast
  thus ⟨s, X⟩ ∈  $\mathcal{F} ?lhs$ 
proof cases
  assume ⟨s ∈  $\mathcal{D} (Q \setminus S)$ ⟩
  hence ⟨s ∈  $\mathcal{D} ?rhs$ ⟩ by (simp add: D-Ndet D-Sliding)
  with same-div D-F show ⟨s, X⟩ ∈  $\mathcal{F} ?lhs$  by blast
next
  assume ⟨∃ t. s = trace-hide t (ev ' S) ∧ (t, X ∪ ev ' S) ∈  $\mathcal{F} Q$ ⟩
  thus ⟨s, X⟩ ∈  $\mathcal{F} ?lhs$ 
  by (simp add: F-Hiding F-Sliding) blast
qed
next
  assume assms : ⟨s ≠ []⟩ ⟨s, X⟩ ∈  $\mathcal{F} (Mprefix A P \setminus S)$ 
  then consider ⟨s ∈  $\mathcal{D} (Mprefix A P \setminus S)$ ⟩
    | ⟨∃ t. t ≠ [] ∧ s = trace-hide t (ev ' S) ∧ (t, X ∪ ev ' S) ∈  $\mathcal{F} (Mprefix A P)$ ⟩
    by (simp add: F-Hiding D-Hiding) (metis filter.simps(1))
  thus ⟨s, X⟩ ∈  $\mathcal{F} ?lhs$ 
proof cases
  assume ⟨s ∈  $\mathcal{D} (Mprefix A P \setminus S)$ ⟩
  hence ⟨s ∈  $\mathcal{D} ?rhs$ ⟩ by (simp add: D-Ndet D-Sliding)
  with same-div D-F show ⟨s, X⟩ ∈  $\mathcal{F} ?lhs$  by blast
next
  show ⟨∃ t. t ≠ [] ∧ s = trace-hide t (ev ' S) ∧
    (t, X ∪ ev ' S) ∈  $\mathcal{F} (Mprefix A P)$  ⟹ (s, X) ∈  $\mathcal{F} ?lhs$ ⟩
  by (auto simp add: F-Sliding F-Hiding)
qed
qed
qed
qed

```

16.2.3 Hiding and Mprefix for non-disjoint Sets

Finally the new version, when $A \cap S \neq \{\}$.

theorem *Hiding-Mprefix-non-disjoint*:

— Rework this proof

⟨ $\Box a \in A \rightarrow P a \setminus S = (\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright (\Box a \in A \cap S. (P a \setminus S))$ ⟩
 (is ⟨?lhs = ?rhs⟩) if non-disjoint: ⟨ $A \cap S \neq \{\}$ ⟩

proof (subst Process-eq-spec-optimized, safe)

fix s

assume ⟨s ∈ $\mathcal{D} ?lhs$ ⟩

then obtain $t u$
where $*$: $\langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle \langle s = \text{trace-hide } t (ev \text{ ' } S) @ u \rangle$
 $\langle t \in \mathcal{D} (Mprefix A P) \vee$
 $(\exists f. \text{isInfHiddenRun } f (Mprefix A P) S \wedge t \in \text{range } f) \rangle$
by (*simp add: D-Hiding*) *blast*
from $*(4)$ **show** $\langle s \in \mathcal{D} ?rhs \rangle$
proof (*elim disjE*)
assume $\langle t \in \mathcal{D} (Mprefix A P) \rangle$
then obtain $a t'$ **where** $**$: $\langle a \in A \rangle \langle t = ev a \# t' \rangle \langle t' \in \mathcal{D} (P a) \rangle$
by (*simp add: D-Mprefix*)
 $(metis \text{event.inject image-iff list.exhaust-sel})$
have $\langle \text{tickFree } t' \wedge$
 $(\text{if } a \in S \text{ then } s \text{ else } tl s) = \text{trace-hide } t' (ev \text{ ' } S) @ u \wedge$
 $(t' \in \mathcal{D} (P a) \vee (\exists f. \text{isInfHiddenRun } f (P a) S \wedge t' \in \text{range } f)) \rangle$
using $*(2) ** (2, 3)$ **by** (*auto simp add: *(1, 3) ** (2) image-iff*)
with $*(1)$ **have** $***$: $\langle (\text{if } a \in S \text{ then } s \text{ else } tl s) \in \mathcal{D} (P a \setminus S) \rangle$
by (*simp add: D-Hiding*) *blast*
show $\langle s \in \mathcal{D} ?rhs \rangle$
proof (*cases* $\langle a \in S \rangle$)
assume $\langle a \in S \rangle$
hence $\langle a \in A \cap S \rangle$ **by** (*simp add: ** (1)*)
with $***$ **show** $\langle s \in \mathcal{D} ?rhs \rangle$
by (*auto simp add: D-Sliding D-GlobalNdet*)
next
assume $\langle a \notin S \rangle$
hence $\langle a \in A - S \rangle$ **by** (*simp add: ** (1)*)
with $***$ **show** $\langle s \in \mathcal{D} ?rhs \rangle$
by (*auto simp add: D-Sliding D-Mprefix *(3) ** (2) image-iff*)
qed
next
assume $\langle \exists f. \text{isInfHiddenRun } f (Mprefix A P) S \wedge t \in \text{range } f \rangle$
then obtain f
where $**$: $\langle \text{isInfHiddenRun } f (Mprefix A P) S \rangle \langle t \in \text{range } f \rangle$ **by** *blast*
obtain k **where** $\langle t = f k \rangle$ **using** $*(2)$ **by** *blast*
show $\langle s \in \mathcal{D} ?rhs \rangle$
proof (*cases* $\langle f 0 = [] \rangle$)
assume $\langle f 0 = [] \rangle$
hence $\langle f 1 \neq [] \rangle$
by (*metis ** (1) One-nat-def monotoneD nil-less zero-less-Suc*)
with $*(1)$ [*THEN conjunct2, THEN conjunct1, rule-format, of 1*]
obtain a **where** $***$: $\langle a \in A \rangle \langle f 1 \neq [] \rangle \langle hd (f 1) = ev a \rangle$
by (*simp add: T-Mprefix*) *blast*
have $****$: $\langle 0 < j \implies f j \neq [] \wedge hd (f j) = ev a \rangle$ **for** j
proof (*induct j rule: nat-induct-non-zero*)
from $*** (2, 3)$ **show** $\langle f 1 \neq [] \wedge hd (f 1) = ev a \rangle$ **by** *blast*
next
case (*Suc j*)
have $\langle j < Suc j \rangle$ **by** *simp*
from $*(1)$ [*THEN conjunct1, THEN strict-monoD, OF this*]

```

obtain  $v$  where  $\langle f (Suc\ j) = f\ j\ @\ v \rangle$ 
  by (metis le-list-def order-less-imp-le)
thus ?case by (simp add: Suc.hyps(2))
qed
from  $**(1)$  have  $*****: \langle a \in A \cap S \rangle$ 
  by simp (metis (no-types, lifting) **(1) **(2, 3)
     $\langle f\ 0 = [] \rangle$  empty-filter-conv event.inject
    filter.simps(1) image-iff list.set-sel(1))
have  $\langle (if\ i = 0 \wedge t = []\ then\ []\ else\ tl\ (f\ (Suc\ i))) \in \mathcal{T}\ (P\ a) \rangle$  for  $i$ 
proof –
  from  $**(1)$   $*****[of\ \langle Suc\ i \rangle]$  have  $\langle tl\ (f\ (Suc\ i)) \in \mathcal{T}\ (P\ a) \rangle$ 
    by (simp add: T-Mprefix) (metis event.inject)
  thus  $\langle (if\ i = 0 \wedge t = []\ then\ []\ else\ tl\ (f\ (Suc\ i))) \in \mathcal{T}\ (P\ a) \rangle$ 
    by (simp add: Nil-elem-T)
qed
hence  $***** :$ 
   $\langle front\ tickFree\ u \wedge tickFree\ (tl\ t) \wedge$ 
     $s = trace\ hide\ (tl\ t)\ (ev\ 'S)\ @\ u \wedge$ 
     $isInfHiddenRun\ (\lambda i. if\ i = 0 \wedge t = []\ then\ []\ else\ tl\ (f\ (Suc\ i)))\ (P\ a)\ S \wedge$ 
     $tl\ t \in range\ (\lambda i. if\ i = 0 \wedge t = []\ then\ []\ else\ tl\ (f\ (Suc\ i))) \rangle$ 
  apply (intro conjI)
  subgoal by (solves  $\langle simp\ add: *(1) \rangle$ )
  subgoal by (metis *(2) list.sel(2) tickFree-tl)
  subgoal by (metis *(3) *(1)  $\langle f\ 0 = [] \rangle$   $\langle t = f\ k \rangle$  empty-filter-conv
    filter.simps(1) list.sel(2) list.set-sel(2))
  subgoal by (simp add: monotone-on-def,
    metis *(1) Suc-less-eq less-list-def less-tail monotoneD
    nil-le nil-less zero-less-Suc)
  subgoal by blast
  subgoal by (simp, metis *(1) ****  $\langle f\ 0 = [] \rangle$  empty-filter-conv
    filter.simps(1) list.set-sel(2) zero-less-Suc)
  by (simp add: image-iff,
    metis Suc-pred  $\langle f\ 0 = [] \rangle$   $\langle t = f\ k \rangle$  bot-nat-0.not-eq-extremum)
have  $\langle a \in A \cap S \wedge s \in \mathcal{D}\ (P\ a \setminus S) \rangle$ 
  apply (simp add: D-Hiding *****[simplified])
  using  $*****$  by blast
hence  $\langle s \in \mathcal{D}\ (\bigcap a \in A \cap S. (P\ a \setminus S)) \rangle$  by (simp add: D-GlobalNdet) blast
thus  $\langle s \in \mathcal{D}\ ?rhs \rangle$  by (simp add: D-Sliding)
next
assume  $\langle f\ 0 \neq [] \rangle$ 
with  $**(1)[THEN\ conjunct2, THEN\ conjunct1, rule-format, of\ 0]$ 
obtain  $a$  where  $***** : \langle a \in A \rangle \langle f\ 0 \neq [] \rangle \langle hd\ (f\ 0) = ev\ a \rangle$ 
  by (simp add: T-Mprefix) blast
have  $***** : \langle f\ j \neq [] \wedge hd\ (f\ j) = ev\ a \rangle$  for  $j$ 
proof (induct j)
  from  $**(2, 3)$  show  $\langle f\ 0 \neq [] \wedge hd\ (f\ 0) = ev\ a \rangle$  by blast
next
  case (Suc j)
  have  $\langle j < Suc\ j \rangle$  by simp

```

```

    from **(1)[THEN conjunct1, THEN strict-monoD, OF this]
    obtain  $v$  where  $\langle f (Suc\ j) = f\ j\ @\ v \rangle$ 
      by (metis le-list-def order-less-imp-le)
    thus ?case by (simp add: Suc.hyps(1))
  qed
  show  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
  proof (cases  $\langle a \in S \rangle$ )
    assume  $\langle a \in S \rangle$ 
    hence  $\langle a \in A \cap S \rangle$  by (simp add: **(1))
    have  $\langle tickFree\ (tl\ t) \wedge s = trace\text{-}hide\ (tl\ t)\ (ev\ 'S)\ @\ u \wedge$ 
       $isInfHiddenRun\ (\lambda i. tl\ (f\ i))\ (P\ a)\ S \wedge tl\ t \in range\ (\lambda i. tl\ (f\ i)) \rangle$ 
      apply (simp add: *(3), intro conjI)
      subgoal by (metis *(2) list.sel(2) tickFree-tl)
    subgoal by (cases t; simp; metis ****  $\langle a \in S \rangle \langle t = f\ k \rangle image\text{-}iff\ list.sel(1)$ )
      subgoal by (meson **(1) **** less-tail strict-mono-Suc-iff)
      subgoal using **(1) by (simp add: T-Mprefix ****)
    subgoal by (metis (no-types, lifting) **(1) **** filter.simps(2) list.exhaust-sel
       $list.sel(3)$ )
      using **(2) by blast
    have  $\langle s \in \mathcal{D}\ (\Box a \in A \cap S. (P\ a \setminus S)) \rangle$ 
      apply (simp add: D-GlobalNdet D-Hiding)
      using *(1)  $\langle a \in A \cap S \rangle$  <?this> by blast
    thus  $\langle s \in \mathcal{D}\ ?rhs \rangle$  by (simp add: D-Sliding)
  next
    assume  $\langle a \notin S \rangle$ 
    have  $\langle tickFree\ (tl\ t) \wedge$ 
       $trace\text{-}hide\ (tl\ t)\ (ev\ 'S)\ @\ u = trace\text{-}hide\ (tl\ t)\ (ev\ 'S)\ @\ u \wedge$ 
       $isInfHiddenRun\ (\lambda i. tl\ (f\ i))\ (P\ a)\ S \wedge tl\ t \in range\ (\lambda i. tl\ (f\ i)) \rangle$ 
      apply (simp add: *(3), intro conjI)
      subgoal by (metis *(2) list.sel(2) tickFree-tl)
      subgoal by (meson **(1) **** less-tail strict-mono-Suc-iff)
      subgoal using **(1) by (simp add: T-Mprefix ****)
    subgoal by (metis (no-types, lifting) **(1) **** filter.simps(2) list.exhaust-sel
       $list.sel(3)$ )
      using **(2) by blast
    from  $\langle a \notin S \rangle$  have  $\langle s \in \mathcal{D}\ (\Box a \in A - S \rightarrow (P\ a \setminus S)) \rangle$ 
      apply (simp add: D-Mprefix *(3)  $\langle t = f\ k \rangle$ )
      using ****(1) ****[of k]
      apply (cases  $\langle f\ k \rangle$ ; simp add:  $\langle a \notin S \rangle image\text{-}iff[of\ \langle ev\ a \rangle]$  D-Hiding)
      using <?this> by (metis *(1)  $\langle t = f\ k \rangle list.sel(3)$ )
    thus  $\langle s \in \mathcal{D}\ ?rhs \rangle$  by (simp add: D-Sliding)
  qed
  qed
  next
  fix  $s$ 
  assume  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
  then consider  $\langle s \in \mathcal{D}\ (\Box a \in A - S \rightarrow (P\ a \setminus S)) \rangle$ 
    |  $\langle s \in \mathcal{D}\ (\Box a \in A \cap S. (P\ a \setminus S)) \rangle$  by (simp add: D-Sliding) blast

```

```

thus  $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$ 
proof cases
  show  $\langle s \in \mathcal{D} (\Box a \in A - S \rightarrow (P a \setminus S)) \implies s \in \mathcal{D} \text{ ?lhs} \rangle$ 
    by (rule D-Hiding-Mprefix-dir2)
next
  assume  $\langle s \in \mathcal{D} (\Box a \in A \cap S. (P a \setminus S)) \rangle$ 
  then obtain  $a$  where  $*$  :  $\langle a \in A \rangle \langle a \in S \rangle \langle s \in \mathcal{D} (P a \setminus S) \rangle$ 
    by (simp add: D-GlobalNdet)
    (metis (no-types, lifting) IntD1 IntD2 UN-iff empty-iff)
  from  $*(3)$  obtain  $t u$ 
    where  $**$  :  $\langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle \langle s = \text{trace-hide } t (ev \text{ ' } S) @ u \rangle$ 
       $\langle t \in \mathcal{D} (P a) \vee (\exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f) \rangle$ 
    by (simp add: D-Hiding) blast
  from  $*(4)$  show  $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$ 
proof (elim disjE)
  assume  $\langle t \in \mathcal{D} (P a) \rangle$ 
  hence  $\$$  :  $\langle \text{tickFree } (ev a \# t) \wedge ev a \# t \in \mathcal{D} (Mprefix A P) \wedge$ 
     $s = \text{trace-hide } (ev a \# t) (ev \text{ ' } S) @ u \rangle$ 
  by (simp add:  $*(1, 2) ** (2, 3)$  D-Mprefix)
  show  $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$ 
  apply (simp add: D-Hiding)
  using  $\$ ** (1)$  by blast
next
  assume  $\langle \exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f \rangle$ 
  then obtain  $f$  where  $\langle \text{isInfHiddenRun } f (P a) S \rangle \langle t \in \text{range } f \rangle$  by blast
  hence  $\$$  :  $\langle \text{tickFree } (ev a \# t) \wedge$ 
     $s = \text{trace-hide } (ev a \# t) (ev \text{ ' } S) @ u \wedge$ 
     $\text{isInfHiddenRun } (\lambda i. ev a \# f i) (Mprefix A P) S \wedge$ 
     $ev a \# t \in \text{range } (\lambda i. ev a \# f i) \rangle$ 
  by (auto simp add: *(1, 2) ** (2, 3) less-cons monotone-on-def T-Mprefix)
  show  $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$ 
  apply (simp add: D-Hiding)
  using  $\$ ** (1)$  by blast
qed
qed
next
  fix  $s X$ 
  assume same-div :  $\langle \mathcal{D} \text{ ?lhs} = \mathcal{D} \text{ ?rhs} \rangle$ 
  assume  $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
  then consider  $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$ 
    |  $\langle \exists t. s = \text{trace-hide } t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$ 
    by (simp add: F-Hiding D-Hiding) blast
  thus  $\langle (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
proof cases
  from D-F same-div show  $\langle s \in \mathcal{D} \text{ ?lhs} \implies (s, X) \in \mathcal{F} \text{ ?rhs} \rangle$  by blast
next
  assume  $\langle \exists t. s = \text{trace-hide } t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$ 
  then obtain  $t$  where  $*$  :  $\langle s = \text{trace-hide } t (ev \text{ ' } S) \rangle$ 
     $\langle (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$  by blast

```

```

from *(2) consider  $\langle t = [] \rangle \langle (X \cup \text{ev } ' S) \cap \text{ev } ' A = \{\} \rangle$ 
  |  $\langle \exists a t'. a \in A \wedge t = \text{ev } a \# t' \wedge (t', X \cup \text{ev } ' S) \in \mathcal{F} (P a) \rangle$ 
  by (simp add: F-Mprefix) (metis event.inject image-iff list.collapse)
thus  $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ 
proof cases
  show  $\langle t = [] \implies (X \cup \text{ev } ' S) \cap \text{ev } ' A = \{\} \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
    using *(1) by (auto simp add: F-Sliding F-GlobalNdet)
next
  assume  $\langle \exists a t'. a \in A \wedge t = \text{ev } a \# t' \wedge (t', X \cup \text{ev } ' S) \in \mathcal{F} (P a) \rangle$ 
  then obtain  $a t'$  where  $** : \langle a \in A \rangle \langle t = \text{ev } a \# t' \rangle$ 
     $\langle (t', X \cup \text{ev } ' S) \in \mathcal{F} (P a) \rangle$  by blast
  show  $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ 
  proof (cases  $\langle a \in S \rangle$ )
    show  $\langle a \in S \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
      using *(1) ** by (auto simp add: F-Sliding F-GlobalNdet F-Hiding)
    next
      show  $\langle a \notin S \implies (s, X) \in \mathcal{F} ?rhs \rangle$ 
      using *(1) ** (1, 2, 3) by (auto simp add: F-Sliding F-Mprefix F-Hiding)
  qed
qed
qed
next
fix  $s X$ 
assume same-div :  $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$ 
assume  $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ 
then consider  $\langle (s, X) \in \mathcal{F} (\bigcap a \in A \cap S. (P a \setminus S)) \rangle$ 
  |  $\langle s \neq [] \rangle \langle (s, X) \in \mathcal{F} (\bigcap a \in A - S \rightarrow (P a \setminus S)) \rangle$ 
  by (auto simp add: F-Sliding)
thus  $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ 
proof cases
  assume  $\langle (s, X) \in \mathcal{F} (\bigcap a \in A \cap S. (P a \setminus S)) \rangle$ 
  then obtain  $a$  where  $* : \langle a \in A \rangle \langle a \in S \rangle \langle (s, X) \in \mathcal{F} (P a \setminus S) \rangle$ 
    by (simp add: F-GlobalNdet non-disjoint) blast
  from *(3) consider  $\langle s \in \mathcal{D} (P a \setminus S) \rangle$ 
  |  $\langle \exists t. s = \text{trace-hide } t (\text{ev } ' S) \wedge (t, X \cup \text{ev } ' S) \in \mathcal{F} (P a) \rangle$ 
  by (simp add: F-Hiding D-Hiding) blast
thus  $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ 
proof cases
  assume  $\langle s \in \mathcal{D} (P a \setminus S) \rangle$ 
  then obtain  $t u$ 
    where  $** : \langle \text{front-tickFree } u \rangle \langle \text{tickFree } t \rangle$ 
       $\langle s = \text{trace-hide } t (\text{ev } ' S) @ u \rangle$ 
       $\langle t \in \mathcal{D} (P a) \vee (\exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f) \rangle$ 
    by (simp add: D-Hiding) blast
  from ** (4) show  $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ 
proof (elim disjE)
  assume  $\langle t \in \mathcal{D} (P a) \rangle$ 
  hence  $\$ : \langle \text{tickFree } (\text{ev } a \# t) \wedge s = \text{trace-hide } (\text{ev } a \# t) (\text{ev } ' S) @ u \wedge$ 
     $\text{ev } a \# t \in \mathcal{D} (\text{Mprefix } A P) \rangle$ 

```

```

    by (simp add: D-Mprefix *(1, 2) **(2, 3) image-iff[of ⟨ev a⟩])
  show ⟨(s, X) ∈  $\mathcal{F}$  ?lhs⟩
    apply (simp add: F-Hiding)
    using $ **(1) by blast
next
  assume ⟨ $\exists f. \text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f$ ⟩
  then obtain f where ⟨ $\text{isInfHiddenRun } f (P a) S \wedge t \in \text{range } f$ ⟩ by blast
  hence $ : ⟨ $\text{tickFree } (ev a \# t) \wedge s = \text{trace-hide } (ev a \# t) (ev ' S) @ u \wedge$   

     $\text{isInfHiddenRun } (\lambda i. ev a \# f i) (Mprefix A P) S \wedge$   

     $ev a \# t \in \text{range } (\lambda i. ev a \# f i)$ ⟩
    by (simp add: T-Mprefix *(1, 2) **(2, 3)
      image-iff[of ⟨ev a⟩] less-cons monotone-on-def) blast
  show ⟨(s, X) ∈  $\mathcal{F}$  ?lhs⟩
    apply (simp add: F-Hiding)
    using $ **(1) by blast
qed
next
  assume ⟨ $\exists t. s = \text{trace-hide } t (ev ' S) \wedge (t, X \cup ev ' S) \in \mathcal{F} (P a)$ ⟩
  then obtain t where ⟨ $s = \text{trace-hide } t (ev ' S)$ ⟩
    ⟨ $(t, X \cup ev ' S) \in \mathcal{F} (P a)$ ⟩ by blast
  hence ⟨ $s = \text{trace-hide } (ev a \# t) (ev ' S) \wedge$   

     $(ev a \# t, X \cup ev ' S) \in \mathcal{F} (Mprefix A P)$ ⟩
    by (simp add: *(1, 2) F-Mprefix)
  show ⟨(s, X) ∈  $\mathcal{F}$  ?lhs⟩
    apply (simp add: F-Hiding, rule disjI1)
    using ⟨?this⟩ by blast
qed
next
  show ⟨ $s \neq [] \implies (s, X) \in \mathcal{F} (\Box a \in A - S \rightarrow (P a \setminus S)) \implies (s, X) \in \mathcal{F} ?lhs$ ⟩
    by (rule F-Hiding-Mprefix-dir2)
qed
qed

```

— Just a small lemma to understand why the nonempty hypothesis is necessary.

lemma $\langle \exists A P S. A \cap S = \{\} \wedge$
 $\Box a \in A :: \text{nat set} \rightarrow P a \setminus S \neq$
 $(\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright (\Box a \in A \cap S. (P a \setminus S)) \rangle$

proof (intro exI)

show $\langle \{0\} \cap \{\text{Suc } 0\} = \{\} \wedge$
 $\Box a \in \{0\} :: \text{nat set} \rightarrow \text{SKIP} \setminus \{\text{Suc } 0\} \neq$
 $(\Box a \in \{0\} - \{\text{Suc } 0\} \rightarrow (\text{SKIP} \setminus \{\text{Suc } 0\})) \triangleright (\Box a \in \{0\} \cap \{\text{Suc } 0\}. (\text{SKIP}$
 $\setminus \{\text{Suc } 0\})) \rangle$

apply (simp add: write0-def[symmetric] no-Hiding-write0 Hiding-set-SKIP)

by (simp add: Process-eq-spec write0-def
 F-Mprefix F-Sliding F-STOP set-eq-iff) blast

qed

And we obtain a similar result when adding a (\triangleright) in the expression.

lemma *Hiding-Mprefix-Sliding-non-disjoint*:

$\langle (\Box a \in A \rightarrow P a) \triangleright Q \rangle \setminus S = (\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright$
 $(Q \setminus S) \sqcap (\Box a \in A \cap S. (P a \setminus S))$

if *non-disjoint*: $\langle A \cap S \neq \{\} \rangle$

proof (*subst Sliding-Ndet*(2),
subst Hiding-Mprefix-non-disjoint[*OF non-disjoint, symmetric*])

show $\langle Mprefix A P \triangleright Q \setminus S =$
 $((\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright (Q \setminus S)) \sqcap (\Box a \in A \rightarrow P a \setminus S) \rangle$
(is $\langle ?lhs = ?rhs \rangle$)

proof (*subst Process-eq-spec-optimized, safe*)

fix s
assume $\langle s \in \mathcal{D} ?lhs \rangle$
hence $\langle s \in \mathcal{D} (Mprefix A P \sqcap Q \setminus S) \rangle$
by (*simp add: D-Hiding D-Sliding D-Ndet T-Sliding T-Ndet*)
thus $\langle s \in \mathcal{D} ?rhs \rangle$
by (*rule set-rev-mp*)
(simp add: D-Ndet D-Sliding Hiding-Ndet subsetI)

next

fix s
assume $\langle s \in \mathcal{D} ?rhs \rangle$
hence $\langle s \in \mathcal{D} (Q \setminus S) \vee s \in \mathcal{D} (\Box a \in A \rightarrow P a \setminus S) \rangle$
by (*simp add: Hiding-Mprefix-non-disjoint*[*OF non-disjoint*]
D-Ndet D-Sliding) *blast*

thus $\langle s \in \mathcal{D} ?lhs \rangle$
by (*auto simp add: D-Hiding D-Sliding T-Sliding*)

next

fix $s X$
assume *same-div* : $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
then consider $\langle s \in \mathcal{D} ?lhs \rangle$
 $| \langle \exists t. s = trace\text{-}hide\ t (ev\ 'S) \wedge (t, X \cup ev\ 'S) \in \mathcal{F} (Mprefix A P \triangleright Q) \rangle$
by (*simp add: F-Hiding D-Hiding*) *blast*

thus $\langle (s, X) \in \mathcal{F} ?rhs \rangle$

proof cases

from same-div D-F show $\langle s \in \mathcal{D} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$ **by** *blast*

next

assume $\langle \exists t. s = trace\text{-}hide\ t (ev\ 'S) \wedge$
 $(t, X \cup ev\ 'S) \in \mathcal{F} (Mprefix A P \triangleright Q) \rangle$

then obtain t

where $*$: $\langle s = trace\text{-}hide\ t (ev\ 'S) \rangle$
 $\langle (t, X \cup ev\ 'S) \in \mathcal{F} (Mprefix A P \triangleright Q) \rangle$ **by** *blast*

from $*$ (2) **consider** $\langle (t, X \cup ev\ 'S) \in \mathcal{F} Q \rangle$
 $| \langle (t, X \cup ev\ 'S) \in \mathcal{F} (Mprefix A P) \rangle$
by (*simp add: F-Sliding D-Mprefix*) *blast*

thus $\langle (s, X) \in \mathcal{F} ?rhs \rangle$

proof cases

have $\langle (t, X \cup ev\ 'S) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} (Q \setminus S) \rangle$
by (*auto simp add: F-Hiding* $*$ (1))

thus $\langle (t, X \cup ev\ 'S) \in \mathcal{F} Q \implies (s, X) \in \mathcal{F} ?rhs \rangle$
by (*simp add: F-Ndet F-Sliding* $*$ (1))

next
assume $\langle t, X \cup ev \text{ ' } S \rangle \in \mathcal{F} (Mprefix A P)$
hence $\langle (s, X) \in \mathcal{F} (Mprefix A P \setminus S) \rangle$ **by** (*auto simp: F-Hiding *(1)*)
thus $\langle (s, X) \in \mathcal{F} ?rhs \rangle$ **by** (*simp add: F-Ndet *(1)*)
qed
qed
next
fix $s X$
have $*$: $\langle s \neq [] \implies (s, X) \in \mathcal{F} (\Box a \in A - S \rightarrow (P a \setminus S)) \implies$
 $(s, X) \in \mathcal{F} (\Box a \in A \rightarrow P a \setminus S) \rangle$
by (*simp add: Hiding-Mprefix-non-disjoint[OF non-disjoint] F-Sliding*)
assume *same-div* : $\langle \mathcal{D} ?lhs = \mathcal{D} ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F} ?rhs \rangle$
with $*$ **consider** $\langle (s, X) \in \mathcal{F} (Q \setminus S) \rangle$ | $\langle (s, X) \in \mathcal{F} (Mprefix A P \setminus S) \rangle$
by (*auto simp add: F-Ndet F-Sliding*)
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof cases
assume $\langle (s, X) \in \mathcal{F} (Q \setminus S) \rangle$
then consider $\langle s \in \mathcal{D} (Q \setminus S) \rangle$
| $\langle \exists t. s = trace\text{-}hide t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} Q \rangle$
by (*simp add: F-Hiding D-Hiding blast*)
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof cases
assume $\langle s \in \mathcal{D} (Q \setminus S) \rangle$
hence $\langle s \in \mathcal{D} ?rhs \rangle$ **by** (*simp add: D-Ndet D-Sliding*)
with *same-div D-F* **show** $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ **by** *blast*
next
assume $\langle \exists t. s = trace\text{-}hide t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} Q \rangle$
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Hiding F-Sliding blast*)
qed
next
assume $\langle (s, X) \in \mathcal{F} (Mprefix A P \setminus S) \rangle$
then consider $\langle s \in \mathcal{D} (Mprefix A P \setminus S) \rangle$
| $\langle \exists t. s = trace\text{-}hide t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$
by (*simp add: F-Hiding D-Hiding blast*)
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof cases
assume $\langle s \in \mathcal{D} (Mprefix A P \setminus S) \rangle$
hence $\langle s \in \mathcal{D} ?rhs \rangle$ **by** (*simp add: D-Ndet D-Sliding*)
with *same-div D-F* **show** $\langle (s, X) \in \mathcal{F} ?lhs \rangle$ **by** *blast*
next
assume $\langle \exists t. s = trace\text{-}hide t (ev \text{ ' } S) \wedge (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$
then obtain t **where** $*$: $\langle s = trace\text{-}hide t (ev \text{ ' } S) \rangle$
 $\langle (t, X \cup ev \text{ ' } S) \in \mathcal{F} (Mprefix A P) \rangle$ **by** *blast*
from $*(2)$ *non-disjoint* **have** $\langle t \neq [] \rangle$ **by** (*simp add: F-Mprefix blast*)
with $*$ **show** $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
by (*simp add: F-Hiding F-Sliding blast*)
qed

qed
 qed
 qed

16.3 (\triangleright) behaviour

We already proved several laws for the (\triangleright) operator. Here we give other results in the same spirit as *Hiding-Mprefix-Sliding-disjoint* and *Hiding-Mprefix-Sliding-non-disjoint*.

lemma *Mprefix-Sliding-Mprefix-Sliding*:

$\langle (\Box a \in A \rightarrow P a) \triangleright (\Box b \in B \rightarrow Q b) \triangleright R =$
 $(\Box x \in A \cup B \rightarrow (\text{if } x \in A \cap B \text{ then } P x \sqcap Q x \text{ else if } x \in A \text{ then } P x \text{ else } Q$
 $x)) \triangleright R \rangle$

(is $\langle (\Box a \in A \rightarrow P a) \triangleright (\Box b \in B \rightarrow Q b) \triangleright R = ?term \triangleright R \rangle$)

proof (*subst Sliding-def, subst Mprefix-Det-distr*)

have $\langle Mprefix B Q \sqcap (Mprefix A P \sqcap Mprefix B Q) \triangleright R = Mprefix A P \sqcap$
 $Mprefix B Q \triangleright R \rangle$

by (*metis Det-STOP Ndet-commute Ndet-distrib Sliding-STOP-Det Sliding-assoc Sliding-def*)

thus $\langle ?term \sqcap Mprefix B Q \triangleright R = ?term \triangleright R \rangle$

by (*simp add: Mprefix-Det-distr Ndet-commute*)

qed

lemma *Mprefix-Sliding-Seq*:

$\langle ((\Box a \in A \rightarrow P a) \triangleright P') ; Q = (\Box a \in A \rightarrow P a ; Q) \triangleright (P' ; Q) \rangle$

proof (*subst Mprefix-Seq[symmetric]*)

show $\langle ((\Box a \in A \rightarrow P a) \triangleright P') ; Q =$
 $((\Box a \in A \rightarrow P a) ; Q) \triangleright (P' ; Q) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)

proof (*subst Process-eq-spec, safe*)

show $\langle s \in \mathcal{D} ?lhs \implies s \in \mathcal{D} ?rhs \rangle$ **for** s

by (*auto simp add: D-Sliding D-Seq T-Sliding*)

next

show $\langle s \in \mathcal{D} ?rhs \implies s \in \mathcal{D} ?lhs \rangle$ **for** s

by (*auto simp add: D-Sliding D-Seq T-Sliding*)

next

show $\langle (s, X) \in \mathcal{F} ?lhs \implies (s, X) \in \mathcal{F} ?rhs \rangle$ **for** $s X$

apply (*simp add: F-Seq, elim disjE exE*)

apply (*solves $\langle auto simp add: F-Sliding F-Seq D-Mprefix \rangle$*)

apply (*solves $\langle auto simp add: F-Sliding T-Sliding F-Seq \rangle$*)

by (*solves $\langle auto simp add: D-Sliding D-Seq F-Sliding F-Seq \rangle$*)

next

show $\langle (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} ?lhs \rangle$ **for** $s X$

apply (*simp add: F-Sliding, elim disjE*)

apply (*solves $\langle auto simp add: F-Seq D-Sliding F-Sliding T-Sliding \rangle$*)

apply (*solves $\langle auto simp add: F-Seq D-Sliding F-Sliding T-Sliding \rangle$*)

by (*simp add: D-Seq D-Mprefix T-Seq T-Mprefix*)

(*metis append-is-Nil-conv event.simps(3) hd-append list.sel(1)*)

qed

qed

lemma *Throw-Sliding* :

$\langle (\Box a \in A \rightarrow P a) \triangleright P' \Theta b \in B. Q b =$
 $(\Box a \in A \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } (P a \Theta b \in B. Q b))) \triangleright (P' \Theta b \in B. Q b) \rangle$
(is $\langle ?lhs = ?rhs \rangle$)

proof (*subst Process-eq-spec-optimized, safe*)

fix s

assume $\langle s \in \mathcal{D} \ ?lhs \rangle$

then consider $\langle \exists t1 \ t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} (Mprefix \ A \ P \triangleright P') \wedge$
 $tickFree \ t1 \wedge set \ t1 \cap ev \ 'B = \{\} \wedge front-tickFree \ t2 \rangle$
 $| \langle \exists t1 \ b \ t2. s = t1 @ ev \ b \ \# \ t2 \wedge t1 @ [ev \ b] \in \mathcal{T} (Mprefix \ A \ P \triangleright P') \wedge$
 $set \ t1 \cap ev \ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D} (Q \ b) \rangle$

by (*simp add: D-Throw*) *blast*

thus $\langle s \in \mathcal{D} \ ?rhs \rangle$

proof *cases*

assume $\langle \exists t1 \ t2. s = t1 @ t2 \wedge t1 \in \mathcal{D} (Mprefix \ A \ P \triangleright P') \wedge$
 $tickFree \ t1 \wedge set \ t1 \cap ev \ 'B = \{\} \wedge front-tickFree \ t2 \rangle$

then obtain $t1 \ t2$

where $*$: $\langle s = t1 @ t2 \rangle \langle t1 \in \mathcal{D} (Mprefix \ A \ P \triangleright P') \rangle \langle tickFree \ t1 \rangle$
 $\langle set \ t1 \cap ev \ 'B = \{\} \rangle \langle front-tickFree \ t2 \rangle$ **by** *blast*

from $*(2)$ **consider** $\langle t1 \in \mathcal{D} (Mprefix \ A \ P) \rangle | \langle t1 \in \mathcal{D} \ P' \rangle$

by (*simp add: D-Sliding*) *blast*

thus $\langle s \in \mathcal{D} \ ?rhs \rangle$

proof *cases*

assume $\langle t1 \in \mathcal{D} (Mprefix \ A \ P) \rangle$

then obtain $a \ t1'$ **where** $\langle t1 = ev \ a \ \# \ t1' \rangle \langle a \in A \rangle \langle t1' \in \mathcal{D} (P \ a) \rangle$

by (*simp add: D-Mprefix image-iff*)
(metis event.inject list.collapse)

with $*(1, 3, 4, 5)$ **show** $\langle s \in \mathcal{D} \ ?rhs \rangle$

by (*simp add: D-Sliding D-Mprefix D-Throw*) (*metis image-eq1*)

next

from $*(1, 3, 4, 5)$ **show** $\langle t1 \in \mathcal{D} \ P' \implies s \in \mathcal{D} \ ?rhs \rangle$

by (*simp add: D-Sliding D-Throw*) *blast*

qed

next

assume $\langle \exists t1 \ b \ t2. s = t1 @ ev \ b \ \# \ t2 \wedge t1 @ [ev \ b] \in \mathcal{T} (Mprefix \ A \ P \triangleright P') \rangle$

\wedge

$set \ t1 \cap ev \ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D} (Q \ b) \rangle$

then obtain $t1 \ b \ t2$

where $*$: $\langle s = t1 @ ev \ b \ \# \ t2 \rangle \langle t1 @ [ev \ b] \in \mathcal{T} (Mprefix \ A \ P \triangleright P') \rangle$
 $\langle set \ t1 \cap ev \ 'B = \{\} \rangle \langle b \in B \rangle \langle t2 \in \mathcal{D} (Q \ b) \rangle$ **by** *blast*

from $*(2)$ **consider** $\langle t1 @ [ev \ b] \in \mathcal{T} (Mprefix \ A \ P) \rangle | \langle t1 @ [ev \ b] \in \mathcal{T} \ P' \rangle$

by (*simp add: T-Sliding*) *blast*

thus $\langle s \in \mathcal{D} \ ?rhs \rangle$

proof *cases*

assume $\langle t1 @ [ev \ b] \in \mathcal{T} (Mprefix \ A \ P) \rangle$

```

then obtain  $a\ t1'$ 
  where  $\langle t1\ @\ [ev\ b] = ev\ a\ \#\ t1' \rangle \langle a \in A \rangle \langle t1' \in \mathcal{T}(P\ a) \rangle$ 
  by (simp add: T-Mprefix)
      (metis list.exhaust-sel snoc-eq-iff-butlast)
with  $\ast(1, 3, 4, 5)$  show  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
  by (cases t1; simp add:  $\ast(1)$  D-Sliding D-Mprefix D-Throw)
      (metis image-eqI)
next
  from  $\ast(1, 3, 4, 5)$  show  $\langle t1\ @\ [ev\ b] \in \mathcal{T}\ P' \implies s \in \mathcal{D}\ ?rhs \rangle$ 
  by (simp add: D-Sliding D-Mprefix D-Throw) blast
qed
qed
next
fix  $s$ 
assume  $\langle s \in \mathcal{D}\ ?rhs \rangle$ 
then consider  $\langle s \in \mathcal{D}\ (Throw\ P'\ B\ Q) \rangle$ 
  |  $\langle s \in \mathcal{D}\ (\Box a \in A \rightarrow (if\ a \in B\ then\ Q\ a\ else\ Throw\ (P\ a)\ B\ Q)) \rangle$ 
  by (simp add: D-Sliding) blast
thus  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 
proof cases
  show  $\langle s \in \mathcal{D}\ (Throw\ P'\ B\ Q) \implies s \in \mathcal{D}\ ?lhs \rangle$ 
  by (simp add: D-Throw D-Sliding T-Sliding) blast
next
assume  $\langle s \in \mathcal{D}\ (\Box a \in A \rightarrow (if\ a \in B\ then\ Q\ a\ else\ Throw\ (P\ a)\ B\ Q)) \rangle$ 
then obtain  $a\ s'$ 
  where  $\ast : \langle s = ev\ a\ \#\ s' \rangle \langle a \in A \rangle$ 
       $\langle s' \in \mathcal{D}\ (if\ a \in B\ then\ Q\ a\ else\ Throw\ (P\ a)\ B\ Q) \rangle$ 
  by (cases s; simp add: D-Mprefix) blast
show  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 
proof (cases  $\langle a \in B \rangle$ )
  from  $\ast$  show  $\langle a \in B \implies s \in \mathcal{D}\ ?lhs \rangle$ 
  by (simp add: D-Throw T-Sliding T-Mprefix disjoint-iff)
      (metis Nil-elem-T emptyE empty-set list.sel(1, 3) self-append-conv2)
next
assume  $\langle a \notin B \rangle$ 
with  $\ast(3)$  consider
   $\langle \exists t1\ t2. s' = t1\ @\ t2 \wedge t1 \in \mathcal{D}\ (P\ a) \wedge tickFree\ t1 \wedge$ 
     $set\ t1 \cap ev\ 'B = \{\} \wedge front-tickFree\ t2 \rangle$ 
  |  $\langle \exists t1\ b\ t2. s' = t1\ @\ ev\ b\ \#\ t2 \wedge t1\ @\ [ev\ b] \in \mathcal{T}\ (P\ a) \wedge$ 
     $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D}\ (Q\ b) \rangle$ 
  by (simp add: D-Throw) blast
thus  $\langle s \in \mathcal{D}\ ?lhs \rangle$ 
proof cases
  assume  $\langle \exists t1\ t2. s' = t1\ @\ t2 \wedge t1 \in \mathcal{D}\ (P\ a) \wedge tickFree\ t1 \wedge$ 
     $set\ t1 \cap ev\ 'B = \{\} \wedge front-tickFree\ t2 \rangle$ 
then obtain  $t1\ t2$ 
  where  $\ast\ast : \langle s' = t1\ @\ t2 \rangle \langle t1 \in \mathcal{D}\ (P\ a) \rangle \langle tickFree\ t1 \rangle$ 
       $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle front-tickFree\ t2 \rangle$ 
  by blast

```

have $*** : \langle s = (ev\ a \# t1) @ t2 \wedge set\ (ev\ a \# t1) \cap ev\ 'B = \{\} \rangle$
by (*simp add: *(1) **(1, 4) image-iff* $\langle a \notin B \rangle$)
from $*$ $**(1, 2, 3, 5)$ **show** $\langle s \in \mathcal{D}\ ?lhs \rangle$
by (*simp add: D-Throw D-Sliding D-Mprefix image-iff*)
(*metis *** event.distinct(1) list.discI list.sel(1, 3) tickFree-Cons*)
next
assume $\langle \exists t1\ b\ t2. s' = t1 @ ev\ b \# t2 \wedge t1 @ [ev\ b] \in \mathcal{T}\ (P\ a) \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge t2 \in \mathcal{D}\ (Q\ b) \rangle$
then obtain $t1\ b\ t2$
where $** : \langle s' = t1 @ ev\ b \# t2 \rangle \langle t1 @ [ev\ b] \in \mathcal{T}\ (P\ a) \rangle$
 $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle b \in B \rangle \langle t2 \in \mathcal{D}\ (Q\ b) \rangle$ **by** *blast*
have $*** : \langle s = (ev\ a \# t1 @ [ev\ b]) @ t2 \wedge set\ (ev\ a \# t1) \cap ev\ 'B = \{\} \rangle$
by (*simp add: *(1) **(1, 3) image-iff* $\langle a \notin B \rangle$)
from $*$ $**(1, 2, 4, 5)$ **show** $\langle s \in \mathcal{D}\ ?lhs \rangle$
by (*simp add: D-Throw T-Sliding T-Mprefix*)
(*metis *** Cons-eq-appendI list.sel(1, 3)*)
qed
qed
qed
next
fix $s\ X$
assume *same-div* : $\langle \mathcal{D}\ ?lhs = \mathcal{D}\ ?rhs \rangle$
assume $\langle (s, X) \in \mathcal{F}\ ?lhs \rangle$
then consider $\langle s \in \mathcal{D}\ ?lhs \rangle$
| $\langle (s, X) \in \mathcal{F}\ (Mprefix\ A\ P \triangleright P') \rangle \langle set\ s \cap ev\ 'B = \{\} \rangle$
| $\langle \exists t1\ b\ t2. s = t1 @ ev\ b \# t2 \wedge t1 @ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P \triangleright P') \wedge$
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge (t2, X) \in \mathcal{F}\ (Q\ b) \rangle$
by (*simp add: F-Throw D-Throw*) *blast*
thus $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
proof cases
from *same-div D-F* **show** $\langle s \in \mathcal{D}\ ?lhs \implies (s, X) \in \mathcal{F}\ ?rhs \rangle$ **by** *blast*
next
show $\langle (s, X) \in \mathcal{F}\ (Mprefix\ A\ P \triangleright P') \implies set\ s \cap ev\ 'B = \{\} \implies (s, X) \in$
 $\mathcal{F}\ ?rhs \rangle$
by (*cases s; simp add: F-Sliding F-Mprefix F-Throw*) *blast*
next
assume $\langle \exists t1\ b\ t2. s = t1 @ ev\ b \# t2 \wedge t1 @ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P \triangleright P') \rangle$
 \wedge
 $set\ t1 \cap ev\ 'B = \{\} \wedge b \in B \wedge (t2, X) \in \mathcal{F}\ (Q\ b) \rangle$
then obtain $t1\ b\ t2$
where $* : \langle s = t1 @ ev\ b \# t2 \rangle \langle t1 @ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P \triangleright P') \rangle$
 $\langle set\ t1 \cap ev\ 'B = \{\} \rangle \langle b \in B \rangle \langle (t2, X) \in \mathcal{F}\ (Q\ b) \rangle$ **by** *blast*
from $*(2)$ **consider** $\langle t1 @ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \rangle$ | $\langle t1 @ [ev\ b] \in \mathcal{T}\ P' \rangle$
by (*simp add: T-Sliding*) *blast*
thus $\langle (s, X) \in \mathcal{F}\ ?rhs \rangle$
proof cases
assume $\langle t1 @ [ev\ b] \in \mathcal{T}\ (Mprefix\ A\ P) \rangle$
then obtain $a\ t1'$
where $\langle t1 @ [ev\ b] = ev\ a \# t1' \rangle \langle a \in A \rangle \langle t1' \in \mathcal{T}\ (P\ a) \rangle$

```

    by (simp add: T-Mprefix)
      (metis list.exhaust-sel snoc-eq-iff-butlast)
  with *(1, 3, 4, 5) show ⟨(s, X) ∈ F ?rhs⟩
    by (cases t1; simp add: *(1) F-Sliding F-Mprefix F-Throw) blast
next
  from *(1, 3, 4, 5) show ⟨t1 @ [ev b] ∈ T P' ⟹ (s, X) ∈ F ?rhs⟩
    by (simp add: F-Sliding F-Mprefix F-Throw) blast
qed
qed
next
fix s X
assume same-div : ⟨D ?lhs = D ?rhs⟩
assume ⟨(s, X) ∈ F ?rhs⟩
then consider ⟨s ∈ D ?rhs⟩ | ⟨(s, X) ∈ F (Throw P' B Q)⟩
  | ⟨s ≠ []⟩ ⟨(s, X) ∈ F (□a ∈ A → (if a ∈ B then Q a else Throw (P a) B Q))⟩
    by (simp add: F-Sliding D-Sliding) blast
thus ⟨(s, X) ∈ F ?lhs⟩
proof cases
  from same-div D-F show ⟨s ∈ D ?rhs ⟹ (s, X) ∈ F ?lhs⟩ by blast
next
  show ⟨(s, X) ∈ F (Throw P' B Q) ⟹ (s, X) ∈ F ?lhs⟩
    by (simp add: F-Throw F-Sliding D-Sliding T-Sliding) blast
next
  assume ⟨s ≠ []⟩ ⟨(s, X) ∈ F (□a ∈ A → (if a ∈ B then Q a else Throw (P a)
B Q))⟩
  then obtain a s'
    where * : ⟨s = ev a # s'⟩ ⟨a ∈ A⟩
      ⟨(s', X) ∈ F (if a ∈ B then Q a else Throw (P a) B Q)⟩
    by (simp add: F-Mprefix image-iff) (metis event.inject list.collapse)
  show ⟨(s, X) ∈ F ?lhs⟩
  proof (cases ⟨a ∈ B⟩)
    assume ⟨a ∈ B⟩
    have ⟨[ev a] ∈ T (Mprefix A P ▷ P')⟩
      by (simp add: T-Sliding T-Mprefix *(2) Nil-lem-T)

    with *(1, 3) ⟨a ∈ B⟩ show ⟨(s, X) ∈ F ?lhs⟩
      by (simp add: F-Throw) (metis append-Nil empty-set inf-bot-left)
  next
    assume ⟨a ∉ B⟩
    with *(3) consider ⟨s' ∈ D (Throw (P a) B Q)⟩
      | ⟨(s', X) ∈ F (P a)⟩ ⟨set s' ∩ ev ' B = {}⟩
      | ⟨∃ t1 b t2. s' = t1 @ ev b # t2 ∧ t1 @ [ev b] ∈ T (P a) ∧
        set t1 ∩ ev ' B = {} ∧ b ∈ B ∧ (t2, X) ∈ F (Q b)⟩
      by (simp add: F-Throw D-Throw) blast
    thus ⟨(s, X) ∈ F ?lhs⟩
  proof cases
    assume ⟨s' ∈ D (Throw (P a) B Q)⟩
    hence ⟨s ∈ D ?rhs⟩ by (simp add: *(1, 2) D-Sliding D-Mprefix ⟨a ∉ B⟩)
    with same-div D-F show ⟨(s, X) ∈ F ?lhs⟩ by blast

```

```

next
  show  $\langle (s', X) \in \mathcal{F}(P a) \implies \text{set } s' \cap \text{ev } 'B = \{\} \implies (s, X) \in \mathcal{F} ?\text{lhs} \rangle$ 
    using  $*(1, 2) \langle a \notin B \rangle$  by (simp add: F-Throw F-Sliding F-Mprefix) blast
next
  assume  $\langle \exists t1 b t2. s' = t1 @ \text{ev } b \# t2 \wedge t1 @ [\text{ev } b] \in \mathcal{T}(P a) \wedge$ 
     $\text{set } t1 \cap \text{ev } 'B = \{\} \wedge b \in B \wedge (t2, X) \in \mathcal{F}(Q b) \rangle$ 
  then obtain  $t1 b t2$ 
    where  $** : \langle s' = t1 @ \text{ev } b \# t2 \rangle \langle t1 @ [\text{ev } b] \in \mathcal{T}(P a) \rangle$ 
     $\langle \text{set } t1 \cap \text{ev } 'B = \{\} \rangle \langle b \in B \rangle \langle (t2, X) \in \mathcal{F}(Q b) \rangle$  by blast
  from  $**$  have  $*** : \langle (\text{ev } a \# t1) @ [\text{ev } b] \in \mathcal{T}(Mprefix A P) \wedge$ 
     $\text{set } (\text{ev } a \# t1) \cap \text{ev } 'B = \{\} \rangle$ 
    by (simp add: T-Mprefix  $*(2)$  image-iff  $\langle a \notin B \rangle$ )
  from  $*(1) ** (1, 4, 5) ** (5)$  show  $\langle (s, X) \in \mathcal{F} ?\text{lhs} \rangle$ 
    by (simp add: F-Throw T-Sliding) (metis  $***$  Cons-eq-appendI)
  qed
  qed
  qed
  qed
end

```


Chapter 17

Conclusion

theory *Conclusion*

imports *OpSemFD OpSemDT OpSemFDBis OpSemDTBis OpSemFBis OpSemTBis NewLaws*

begin

We started by defining the operators (\triangleright), *Throw* and (\triangle) and provided on them several new laws, especially monotony, "step-law" (behaviour with $\square a \in A \rightarrow P a$) and continuity.

We defined the *ready-set* notion, and described its behaviour with the reference processes and the operators of CSP (which is already a minor contribution).

As main contribution, we defined the *After* operator which represents a bridge between the denotational and the versions of operational semantics for CSP. Therefore we derive the correspondence between denotational and operational semantics by construction. Based on failure divergence or trace divergence refinements, the two operational semantics correspond to the versions described in [3, 5].

We only have a slight variation for the *Sync* operator: *STOP* is replaced by *SKIP*, probably because of the operator definition in HOL-CSP.

Thus, we provided a formal theory of operational behaviour for CSP, which is, to our knowledge, done for the first time for the entire language and the complete FD-Semantics model. Some of the proofs turned out to be extremely complex and out of reach of paper-and-pencil reasoning.

A notable point is that the experimental order ($_{DT} \rightsquigarrow_{\tau}$) behaves surprisingly well: initially pushed in HOL-CSP for pure curiosity, it looks promising for future applications, since it gives a direct handle for an operational trace semantics for non-diverging processes which is executable.

Another take-away is the development of alternatives allowing ($_{F} \rightsquigarrow_{\tau}$) and

$(T \rightsquigarrow \tau)$ orders to be used operational construction by modifying the definition of *AfterExt.AfterExt*.

But even if $(FD \rightsquigarrow \tau)$ and $(DT \rightsquigarrow \tau)$ constructions are (almost) not impacted by this change, this remains a bit disappointing because the monotony of $(F \rightsquigarrow \tau)$ and $(T \rightsquigarrow \tau)$ w.r.t. to some operators does not allow to recover all the laws of [3, 5].

As a bonus we provided in *HOL-CSP-OpSem.NewLaws* some powerful laws for CSP. Here, we recall only the most important ones:

$$bij\ ?f \implies Renaming\ (?P \setminus ?S)\ ?f = Renaming\ ?P\ ?f \setminus ?f\ ' ?S$$

$$bij\ ?f \implies Renaming\ (?P\ [\![?S]\!] ?Q)\ ?f = Renaming\ ?P\ ?f\ [\![?f\ ' ?S]\!] Renaming\ ?Q\ ?f$$

$$?A \cap ?S \neq \emptyset \implies \Box a \in ?A \rightarrow ?P\ a \setminus ?S = (\Box a \in ?A - ?S \rightarrow (?P\ a \setminus ?S))$$

$$\triangleright (\Box a \in ?A \cap ?S.\ ?P\ a \setminus ?S)$$

end

Bibliography

- [1] B. Ballenghien, S. Taha, and B. Wolff. Hol-cspm - architectural operators for hol-csp. *Archive of Formal Proofs*, December 2023. <https://isa-afp.org/entries/HOL-CSPM.html>, Formal proof development.
- [2] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 281–305, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
- [3] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.
- [4] A. W. Roscoe. Understanding concurrent systems. In *Texts in Computer Science*, 2010.
- [5] A. W. Roscoe. The expressiveness of CSP with priority. In D. R. Ghica, editor, *The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015*, volume 319 of *Electronic Notes in Theoretical Computer Science*, pages 387–401. Elsevier, 2015.
- [6] S. Taha, L. Ye, and B. Wolff. Hol-csp version 2.0. *Archive of Formal Proofs*, April 2019. <https://isa-afp.org/entries/HOL-CSP.html>, Formal proof development.