

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	15
2.2.2	Projections	16
2.2.3	Monotony	17
2.2.4	Properties	17
2.2.5	Key Property	18
2.2.6	Continuity	19
2.3	The Interrupt Operator	19
2.3.1	Definition	19
2.3.2	Projections	20
2.3.3	Monotony	20
2.3.4	Properties	21
2.3.5	Key Property	21
2.3.6	Continuity	21
3	The Ready Set Notion	23
3.1	Definition	23
3.2	Anti-Mono Rules	24
3.3	Behaviour of <i>ready-set</i> with <i>STOP</i> , <i>SKIP</i> and \perp	24
3.4	Behaviour of <i>ready-set</i> with Operators of HOL-CSP	25
3.5	Behaviour of <i>ready-set</i> with Operators of HOL-CSPM	27
3.6	Behaviour of <i>ready-set</i> with Operators of HOL-CSP_OpSem	28

3.7	Behaviour of <i>ready-set</i> with Reference Processes	28
4	Construction of the After Operator	29
4.1	Definition	29
4.2	Projections	29
4.3	Monotony	30
4.4	Behaviour of <i>After</i> with <i>STOP</i> , <i>SKIP</i> and \perp	31
4.5	Behaviour of <i>After</i> with Operators of HOL-CSP	31
4.5.1	Loss of Determinism	31
4.5.2	<i>After</i> Sequential Composition	32
4.5.3	<i>After</i> Synchronization	33
4.5.4	<i>After</i> Hiding Operator	34
4.5.5	<i>After</i> Renaming	35
4.6	Behaviour of <i>After</i> with Operators of HOL-CSPM	35
4.7	Behaviour of <i>After</i> with Operators of HOL-CSP_OpSem	36
4.7.1	<i>After</i> Sliding	36
4.7.2	<i>After</i> Throwing	36
4.7.3	<i>After</i> Interrupting	36
4.8	Behaviour of <i>After</i> with Reference Processes	37
5	Extension of the After Operator	39
5.1	The AfterExt Operator	39
5.1.1	Definition	39
5.1.2	Projections	39
5.1.3	Monotony	40
5.1.4	Behaviour of <i>AfterExt</i> with <i>STOP</i> , <i>SKIP</i> and \perp	40
5.1.5	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP	41
5.1.6	Behaviour of <i>AfterExt</i> with Operators of HOL-CSPM	43
5.1.7	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP_OpSem	43
5.1.8	Behaviour of <i>AfterExt</i> with Reference Processes	44
5.2	The AfterTrace Operator	45
5.2.1	Definition	45
5.2.2	Projections	46
5.2.3	Monotony	46
5.2.4	Another Definition of <i>events-of</i>	47
5.2.5	Characterizations for Deadlock Freeness	47
6	Motivations for our Definitions	49
7	Generic Operational Semantics as a Locale	51
7.1	Definition	51
7.2	Consequences of $P \rightsquigarrow^*_s Q$ on \mathcal{F} , \mathcal{T} and \mathcal{D}	52
7.3	Characterizations for $P \rightsquigarrow^*_s Q$	53
7.4	Finally: $P \rightsquigarrow^*_s Q$ is $P \text{ afterTrace } s \rightsquigarrow_{\mathcal{T}} Q$	53

7.5	General Rules of Operational Semantics	54
8	Failure Divergence Operational Semantics	57
8.1	Operational Semantics Laws	57
8.2	Reality Checks	61
8.3	Other Results	62
8.4	Summary: Operational Rules	62
9	Trace Divergence Operational Semantics	67
9.1	Operational Semantics Laws	67
9.2	Reality Checks	71
9.3	Other Results	71
9.4	Summary: Operational Rules	71
10	Extension of the After Operator, bis	75
10.1	The AfterExt Operator, bis	75
10.1.1	Definition	75
10.1.2	Projections	75
10.1.3	Monotony	76
10.1.4	Behaviour of <i>AfterExt</i> with <i>STOP</i> , <i>SKIP</i> and \perp	76
10.1.5	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP	77
10.1.6	Behaviour of <i>AfterExt</i> with Operators of HOL-CSPM	79
10.1.7	Behaviour of <i>AfterExt</i> with Operators of HOL-CSP_OpSem	79
10.1.8	Behaviour of <i>AfterExt</i> with Reference Processes	80
10.2	The AfterTrace Operator, bis	81
10.2.1	Definition	81
10.2.2	Projections	82
10.2.3	Monotony	82
10.2.4	Another Definition of <i>events-of</i>	83
10.2.5	Characterizations for Deadlock Freeness	83
11	Generic Operational Semantics as a Locale, bis	85
11.1	Definition	85
11.2	Consequences of $P \rightsquigarrow^*_s Q$ on \mathcal{F} , \mathcal{T} and \mathcal{D}	86
11.3	Characterizations for $P \rightsquigarrow^*_s Q$	87
11.4	Finally: $P \rightsquigarrow^*_s Q$ is P <i>afterTrace</i> $s \rightsquigarrow_{\mathcal{T}} Q$	87
11.5	General Rules of Operational Semantics	88
12	Failure Divergence Operational Semantics, bis	91
12.1	Operational Semantics Laws	91
12.2	Reality Checks	95
12.3	Other Results	96
12.4	Summary: Operational Rules	96

13 Trace Divergence Operational Semantics, bis	101
13.1 Operational Semantics Laws	101
13.2 Reality Checks	105
13.3 Other Results	105
13.4 Summary: Operational Rules	106
14 Failure Operational Semantics, bis	111
14.1 Operational Semantics Laws	111
14.2 Reality Checks	114
14.3 Other Results	114
14.4 Nicely written operational rules	115
15 Trace Operational Semantics, bis	119
15.1 Operational Semantics Laws	119
15.2 Reality Checks	122
15.3 Other Results	123
15.4 Summary: Operational Rules	123
16 Bonus: powerful new Laws	127
16.1 Powerful Results about <i>Renaming</i>	127
16.1.1 Some Generalizations	127
16.1.2 <i>Renaming</i> and <i>Hiding</i>	128
16.1.3 <i>Renaming</i> and <i>Sync</i>	128
16.2 <i>Hiding</i> and <i>Mprefix</i>	129
16.2.1 Two intermediate Results	129
16.2.2 <i>Hiding</i> and <i>Mprefix</i> for disjoint Sets	129
16.2.3 <i>Hiding</i> and <i>Mprefix</i> for non-disjoint Sets	129
16.3 (\triangleright) behaviour	130
17 Conclusion	131

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

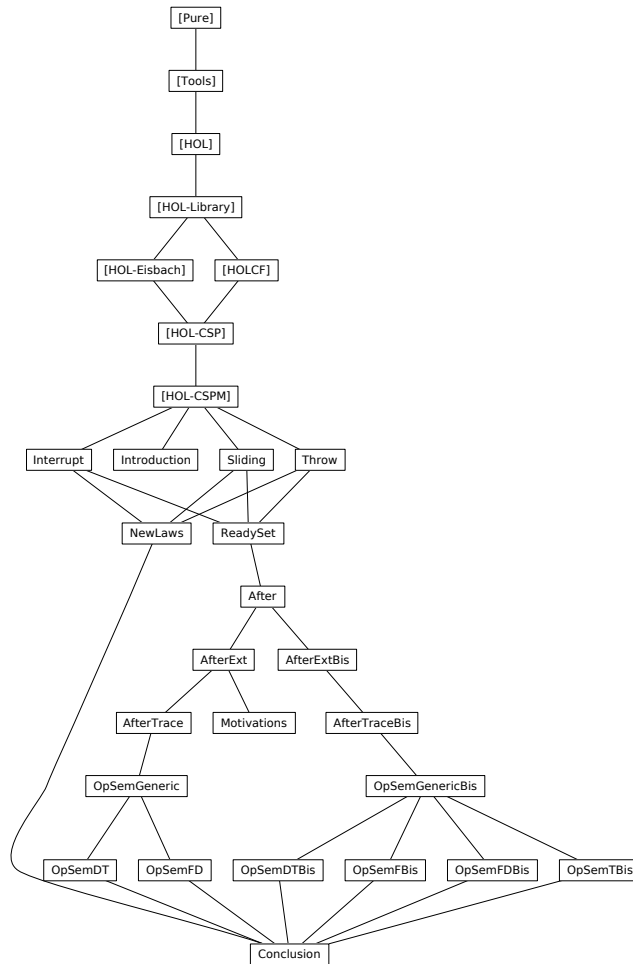


Figure 1.1: The overall architecture

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*:

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

<proof>

corollary $\langle \mathcal{F} (Mprefix A P \triangleright Q) = \mathcal{F} Q \cup \{(s, X) \in \mathcal{F} (Mprefix A P). s \neq []\} \rangle$

<proof>

lemma *D-Sliding*: $\langle \mathcal{D} (P \triangleright Q) = \mathcal{D} P \cup \mathcal{D} Q \rangle$

<proof>

lemma *T-Sliding*: $\langle \mathcal{T} (P \triangleright Q) = \mathcal{T} P \cup \mathcal{T} Q \rangle$
<proof>

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*: $\langle P \sqsubseteq_{DT} P' \implies Q \sqsubseteq_{DT} Q' \implies P \triangleright Q \sqsubseteq_{DT} P' \triangleright Q' \rangle$
<proof>

2.1.4 Properties

lemma *Sliding-id*: $\langle P \triangleright P = P \rangle$
<proof>

lemma *STOP-Sliding*: $\langle STOP \triangleright P = P \rangle$
<proof>

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>

But we still have this result.

lemma *Sliding-is-STOP-iff*: $\langle P \triangleright Q = STOP \iff P = STOP \wedge Q = STOP \rangle$
<proof>

lemma *Sliding-STOP-Det*: $\langle (P \triangleright STOP) \sqcap Q = P \triangleright Q \rangle$
<proof>

lemma *BOT-Sliding*: $\langle \perp \triangleright P = \perp \rangle$
and *Sliding-BOT*: $\langle P \triangleright \perp = \perp \rangle$
<proof>

lemma *Sliding-is-BOT-iff*: $\langle P \triangleright Q = \perp \iff P = \perp \vee Q = \perp \rangle$
<proof>

lemma *Sliding-assoc*: $\langle P1 \triangleright P2 \triangleright P3 = P1 \triangleright (P2 \triangleright P3) \rangle$
<proof>

lemma *SKIP-Sliding*: $\langle SKIP \triangleright P = P \sqcap SKIP \rangle$
<proof>

lemma *Sliding-SKIP*: $\langle P \triangleright SKIP = P \sqcap SKIP \rangle$

<proof>

lemma *Sliding-Det*: $\langle (P \triangleright P') \sqcap Q = P \triangleright P' \sqcap Q \rangle$
<proof>

lemma *Sliding-Ndet*: $\langle (P \sqcap P') \triangleright Q = (P \triangleright Q) \sqcap (P' \triangleright Q) \rangle$
 $\langle P \triangleright (Q \sqcap Q') = (P \triangleright Q) \sqcap (P \triangleright Q') \rangle$
<proof>

lemma *Renaming-Sliding*:
 $\langle \text{Renaming } (P \triangleright Q) f = \text{Renaming } P f \triangleright \text{Renaming } Q f \rangle$
<proof>

lemma *events-Sliding*: $\langle \text{events-of } (P \triangleright Q) = \text{events-of } P \cup \text{events-of } Q \rangle$
<proof>

2.1.5 Continuity

From the definition, continuity is obvious.

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

end

2.2 The Throw Operator

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

<proof>

2.2.1 Definition

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

This operator can somehow be seen as a generalization of sequential composition ($;$): P terminates on any event in $ev \ A$ 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 $\langle \lambda P A Q.$
 $\{(t1, X) \in \mathcal{F} P. \text{set } t1 \cap \text{ev } \alpha A = \{\}\} \cup$
 $\{(t1 @ t2, X) \mid t1 t2 X. t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge$
 $\text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{(t1 @ \text{ev } a \# t2, X) \mid t1 a t2 X. t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge$
 $a \in A \wedge (t2, X) \in \mathcal{F} (Q a)\},$
 $\{t1 @ t2 \mid t1 t2. t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge$
 $\text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{t1 @ \text{ev } a \# t2 \mid t1 a t2. t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge$
 $a \in A \wedge t2 \in \mathcal{D} (Q a)\}\rangle$
 $\langle \text{proof} \rangle$

We add some syntactic sugar.

syntax $-Throw :: \langle [\alpha \text{ process}, \text{ptrn}, \alpha \text{ set}, \alpha \Rightarrow \alpha \text{ process}] \Rightarrow \alpha \text{ process} \rangle$
 $\langle \langle (-) \Theta (-) \cdot (-) \rangle [73, 0, 0, 73] 72 \rangle$
translations $P \Theta a \in A. Q \Rightarrow \text{CONST } Throw P A (\lambda a. Q)$

abbreviation $Throw\text{-without-free-var} ::$
 $\langle [\alpha \text{ process}, \alpha \text{ set}, \alpha \text{ process}] \Rightarrow \alpha \text{ process} \rangle \langle \langle (-) \Theta (-) \cdot (-) \rangle [73, 0, 73] 72 \rangle$
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 $\langle P \Theta a \in A. Q = P \Theta A Q \rangle \langle \text{proof} \rangle$

2.2.2 Projections

lemma $F\text{-Throw}$:
 $\langle \mathcal{F} (P \Theta a \in A. Q a) =$
 $\{(t1, X) \in \mathcal{F} P. \text{set } t1 \cap \text{ev } \alpha A = \{\}\} \cup$
 $\{(t1 @ t2, X) \mid t1 t2 X.$
 $t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{(t1 @ \text{ev } a \# t2, X) \mid t1 a t2 X.$
 $t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge a \in A \wedge (t2, X) \in \mathcal{F} (Q a)\}\rangle$
 $\langle \text{proof} \rangle$

lemma $D\text{-Throw}$:
 $\langle \mathcal{D} (P \Theta a \in A. Q a) =$
 $\{t1 @ t2 \mid t1 t2.$
 $t1 \in \mathcal{D} P \wedge \text{tickFree } t1 \wedge \text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge \text{front-tickFree } t2\} \cup$
 $\{t1 @ \text{ev } a \# t2 \mid t1 a t2.$
 $t1 @ [\text{ev } a] \in \mathcal{T} P \wedge \text{set } t1 \cap \text{ev } \alpha A = \{\} \wedge a \in A \wedge t2 \in \mathcal{D} (Q a)\}\rangle$
 $\langle \text{proof} \rangle$

lemma *T-Throw*:

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

2.2.3 Monotony

lemma *min-elems-Un-subset*:

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

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** $\langle \forall a \in A. Q a \sqsubseteq Q' a \rangle$

$\langle \text{proof} \rangle$

lemma *mono-right-Throw-F* :

$$\langle \forall a \in A. Q a \sqsubseteq_F Q' a \implies P \Theta a \in A. Q a \sqsubseteq_F P \Theta a \in A. Q' a \rangle \\ \langle \text{proof} \rangle$$

lemma *mono-right-Throw-T* :

$$\langle \forall a \in A. Q a \sqsubseteq_T Q' a \implies P \Theta a \in A. Q a \sqsubseteq_T P \Theta a \in A. Q' a \rangle \\ \langle \text{proof} \rangle$$

lemma *mono-right-Throw-D*:

$$\langle \forall a \in A. Q a \sqsubseteq_D Q' a \implies P \Theta a \in A. Q a \sqsubseteq_D P \Theta a \in A. Q' a \rangle \\ \langle \text{proof} \rangle$$

lemma *mono-Throw-FD* : $\langle P \sqsubseteq_{FD} P' \implies \forall a \in A. Q a \sqsubseteq_{FD} Q' a \implies \\ P \Theta a \in A. Q a \sqsubseteq_{FD} P' \Theta a \in A. Q' a \rangle$

$\langle \text{proof} \rangle$

lemma *mono-Throw-DT* : $\langle P \sqsubseteq_{DT} P' \implies \forall a \in A. Q a \sqsubseteq_{DT} Q' a \implies \\ P \Theta a \in A. Q a \sqsubseteq_{DT} P' \Theta a \in A. Q' a \rangle$

$\langle \text{proof} \rangle$

2.2.4 Properties

lemma *Throw-STOP*: $\langle \text{STOP} \Theta a \in A. Q a = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *Throw-SKIP*: $\langle \text{SKIP} \Theta a \in A. Q a = \text{SKIP} \rangle$
 $\langle \text{proof} \rangle$

lemma *Throw-BOT*: $\langle \perp \Theta a \in A. Q a = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *Throw-is-BOT-iff*: $\langle P \Theta a \in A. Q a = \perp \longleftrightarrow P = \perp \rangle$
 $\langle proof \rangle$

lemma *Throw-empty-set*: $\langle P \Theta a \in \{\}, Q a = P \rangle$
 $\langle proof \rangle$

lemma *Throw-Ndet*:
 $\langle P \sqcap P' \Theta a \in A. Q a = (P \Theta a \in A. Q a) \sqcap (P' \Theta a \in A. Q a) \rangle$
 $\langle P \Theta a \in A. Q a \sqcap Q' a = (P \Theta a \in A. Q a) \sqcap (P' \Theta a \in A. Q' a) \rangle$
 $\langle proof \rangle$

lemma *Throw-Det*:
 $\langle P \sqcup P' \Theta a \in A. Q a = (P \Theta a \in A. Q a) \sqcup (P' \Theta a \in A. Q a) \rangle$
 $\langle P \Theta a \in A. Q a \sqcup Q' a = (P \Theta a \in A. Q a) \sqcup (P' \Theta a \in A. Q' a) \rangle$
 $\langle proof \rangle$

lemma *Throw-GlobalNdet*:
 $\langle (\sqcap a \in A. P a) \Theta b \in B. Q b = \sqcap a \in A. (P a \Theta b \in B. Q b) \rangle$
 $\langle P' \Theta a \in A. (\sqcap b \in B. Q' a b) =$
 $(if B = \{\} then P' \Theta A STOP else \sqcap b \in B. (P' \Theta a \in A. Q' a b)) \rangle$
 $\langle proof \rangle$

lemma *Throw-disjoint-events*: $\langle A \cap events-of P = \{\} \implies P \Theta a \in A. Q a = P \rangle$
 $\langle proof \rangle$

lemma *events-Throw*:
 $\langle events-of (P \Theta a \in A. Q a) \subseteq events-of P \cup (\bigcup a \in (A \cap events-of P). events-of (Q a)) \rangle$
 $\langle proof \rangle$

2.2.5 Key Property

lemma *Throw-Mprefix*:
 $\langle (\sqcap a \in A \rightarrow P a) \Theta b \in B. Q b =$
 $\sqcap a \in A \rightarrow (if a \in B then Q a else P a \Theta b \in B. Q b) \rangle$
 $(is \langle ?lhs = ?rhs \rangle)$
 $\langle proof \rangle$

corollary *Throw-prefix*: $\langle (a \rightarrow P) \Theta b \in B. Q b =$
 $(a \rightarrow (if a \in B then Q a else (P \Theta b \in B. Q b))) \rangle$
 $\langle proof \rangle$

corollary *Throw-Mndetprefix*:

$\langle (\prod a \in A \rightarrow P a) \Theta b \in B. Q b =$
 $\prod a \in A \rightarrow (if a \in B then Q a else P a \Theta b \in B. Q b) \rangle$
 $\langle proof \rangle$

2.2.6 Continuity

lemma *chain-left-Throw*: $\langle chain Y \implies chain (\lambda i. Y i \Theta a \in A. Q a) \rangle$
 $\langle proof \rangle$

lemma *chain-right-Throw*: $\langle chain Y \implies chain (\lambda i. P \Theta a \in A. Y i a) \rangle$
 $\langle proof \rangle$

lemma *cont-left-prem-Throw* :
 $\langle (\bigsqcup i. Y i) \Theta a \in A. Q a = (\bigsqcup i. Y i \Theta a \in A. Q a) \rangle$
 $(is \langle ?lhs = ?rhs \rangle) \text{ if } chain : \langle chain Y \rangle$
 $\langle proof \rangle$

lemma *cont-right-prem-Throw* :
 $\langle P \Theta a \in A. (\bigsqcup i. Y i a) = (\bigsqcup i. P \Theta a \in A. Y i a) \rangle$
 $(is \langle ?lhs = ?rhs \rangle) \text{ if } chain : \langle chain Y \rangle$
 $\langle proof \rangle$

lemma *Throw-cont[simp]* :
assumes *cont-f* : $\langle cont f \rangle$ **and** *cont-g* : $\langle \forall a. cont (g a) \rangle$
shows $\langle cont (\lambda x. f x \Theta a \in A. g a x) \rangle$
 $\langle proof \rangle$

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* :: $\langle [\alpha process, \alpha process] \Rightarrow \alpha process \rangle$ (**infixl** $\langle \Delta \rangle$ 75)

is $\langle \lambda P Q.$

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

2.3.2 Projections

lemma *F-Interrupt* :

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

lemma *D-Interrupt* :

$$\langle \mathcal{D} (P \Delta Q) = \mathcal{D} P \cup \{t1 @ t2 \mid t1 t2. t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{D} Q\} \rangle$$

$\langle proof \rangle$

lemma *T-Interrupt* :

$$\langle \mathcal{T} (P \Delta Q) = \mathcal{T} P \cup \{t1 @ t2 \mid t1 t2. t1 \in \mathcal{T} P \wedge tickFree t1 \wedge t2 \in \mathcal{T} Q\} \rangle$$

$\langle proof \rangle$

2.3.3 Monotony

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

$\langle proof \rangle$

lemma *mono-Interrupt-T*: $\langle P \sqsubseteq_T P' \implies Q \sqsubseteq_T Q' \implies P \Delta Q \sqsubseteq_T P' \Delta Q' \rangle$

$\langle proof \rangle$

lemma *mono-right-Interrupt-D*: $\langle Q \sqsubseteq_D Q' \implies P \Delta Q \sqsubseteq_D P \Delta Q' \rangle$

$\langle proof \rangle$

lemma *mono-Interrupt-FD*:

$$\langle P \sqsubseteq_{FD} P' \implies Q \sqsubseteq_{FD} Q' \implies P \Delta Q \sqsubseteq_{FD} P' \Delta Q' \rangle$$

$\langle proof \rangle$

lemma *mono-Interrupt-DT*:

$$\langle P \sqsubseteq_{DT} P' \implies Q \sqsubseteq_{DT} Q' \implies P \triangle Q \sqsubseteq_{DT} P' \triangle Q' \rangle$$

<proof>

2.3.4 Properties

lemma *Interrupt-STOP-neutral* : $\langle P \triangle STOP = P \rangle$ $\langle STOP \triangle P = P \rangle$
<proof>

lemma *Interrupt-BOT-absorb* : $\langle P \triangle \perp = \perp \rangle$ $\langle \perp \triangle P = \perp \rangle$
<proof>

lemma *Interrupt-is-BOT-iff* : $\langle P \triangle Q = \perp \iff P = \perp \vee Q = \perp \rangle$
<proof>

lemma *events-Interrupt*: $\langle \text{events-of } (P \triangle Q) = \text{events-of } P \cup \text{events-of } Q \rangle$
<proof>

lemma *Interrupt-Ndet-distrib* : $\langle P \triangle Q1 \sqcap Q2 = (P \triangle Q1) \sqcap (P \triangle Q2) \rangle$
(is <?lhs = ?rhs>)
<proof>

lemma *Ndet-Interrupt-distrib* : $\langle P1 \sqcap P2 \triangle Q = (P1 \triangle Q) \sqcap (P2 \triangle Q) \rangle$
(is <?lhs = ?rhs>)
<proof>

lemma *Interrupt-assoc*: $\langle P \triangle (Q \triangle R) = P \triangle Q \triangle R \rangle$ *(is <?lhs = ?rhs>)*
<proof>

2.3.5 Key Property

lemma *Interrupt-Mprefix*:

$$\langle (\Box a \in A \rightarrow P a) \triangle Q = Q \sqcap (\Box a \in A \rightarrow P a \triangle Q) \rangle$$
 (is <?lhs = ?rhs>)
<proof>

corollary $\langle (\Box a \in A \rightarrow P a) \triangle (\Box b \in B \rightarrow Q b) =$

$$\Box x \in A \cup B \rightarrow (\begin{array}{l} \text{if } x \in A \cap B \text{ then } (P x \triangle (\Box b \in B \rightarrow Q b)) \sqcap Q x \\ \text{else if } x \in A \text{ then } P x \triangle (\Box b \in B \rightarrow Q b) \\ \text{else } Q x \end{array}) \rangle$$

<proof>

2.3.6 Continuity

lemma *chain-left-Interrupt*: $\langle \text{chain } Y \implies \text{chain } (\lambda i. Y i \triangle Q) \rangle$
<proof>

lemma *chain-right-Interrupt*: $\langle \text{chain } Y \implies \text{chain } (\lambda i. P \Delta Y i) \rangle$
<proof>

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>

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* : $\langle \text{chain } Y \rangle$
<proof>

lemma *Interrupt-cont[simp]* :
 assumes *cont-f* : $\langle \text{cont } f \rangle$ **and** *cont-g* : $\langle \text{cont } g \rangle$
 shows $\langle \text{cont } (\lambda x. f x \Delta g x) \rangle$
<proof>

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 ::  $\langle 'a\ process \Rightarrow 'a\ event\ set \rangle$ 
  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:  $\langle e \# s \in \mathcal{T}\ P \Longrightarrow e \in ready\text{-}set\ P \rangle$ 
   $\langle proof \rangle$ 
```

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
   $\langle proof \rangle$ 
end

```

3.2 Anti-Mono Rules

lemma *anti-mono-ready-set-T*: $\langle P \sqsubseteq_T Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
 $\langle proof \rangle$

lemma *anti-mono-ready-set-F*: $\langle P \sqsubseteq_F Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
 $\langle proof \rangle$

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

lemma *anti-mono-ready-set-FD*: $\langle P \sqsubseteq_{FD} Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
 $\langle proof \rangle$

lemma *anti-mono-ready-set*: $\langle P \sqsubseteq Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
 $\langle proof \rangle$

lemma *anti-mono-ready-set-DT*: $\langle P \sqsubseteq_{DT} Q \implies \text{ready-set } Q \subseteq \text{ready-set } P \rangle$
 $\langle proof \rangle$

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

lemma *ready-set-STOP*: $\langle \text{ready-set } STOP = \{\} \rangle$
 $\langle proof \rangle$

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

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

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

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

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

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

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

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

lemma *ready-set-Det*: $\langle \text{ready-set } (P \Box Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$
and *ready-set-Ndet*: $\langle \text{ready-set } (P \sqcap Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$
 $\langle \text{proof} \rangle$

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

lemma *ready-set-Sync*:
 $\langle \text{ready-set } (P \llbracket S \rrbracket Q) =$
 $(\text{if } P = \perp \vee Q = \perp \text{ then UNIV}$
 $\text{else } (\text{ready-set } P - \text{insert } \checkmark \text{ (ev } \langle S \rangle)) \cup$
 $(\text{ready-set } Q - \text{insert } \checkmark \text{ (ev } \langle S \rangle)) \cup$
 $\text{ready-set } P \cap \text{ready-set } Q \cap \text{insert } \checkmark \text{ (ev } \langle S \rangle) \rangle$
(is $\langle \text{ready-set } (P \llbracket S \rrbracket Q) = (\text{if } P = \perp \vee Q = \perp \text{ then UNIV else ?rhs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *ready-set-Renaming*:
 $\langle \text{ready-set } (\text{Renaming } P f) = (\text{if } P = \perp \text{ then UNIV else EvExt } f \langle \text{ready-set } P \rangle) \rangle$
 $\langle \text{proof} \rangle$

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* :

$$\langle P \setminus S = \perp \longleftrightarrow (\exists t. \text{set } t \subseteq \text{ev } 'S \wedge (t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f P S \wedge t \in \text{range } f))) \rangle$$

(is $\langle P \setminus S = \perp \longleftrightarrow ?rhs \rangle$)
 $\langle proof \rangle$

lemma *event-in-D-Hiding-iff* :

$$\langle [e] \in \mathcal{D} (P \setminus S) \longleftrightarrow P \setminus S = \perp \vee (\exists x t. e = \text{ev } x \wedge x \notin S \wedge [\text{ev } x] = \text{trace-hide } t (\text{ev } 'S) \wedge (t \in \mathcal{D} P \vee (\exists f. \text{isInfHiddenRun } f P S \wedge t \in \text{range } f))) \rangle$$

(is $\langle [e] \in \mathcal{D} (P \setminus S) \longleftrightarrow P \setminus S = \perp \vee ?ugly\text{-assertion} \rangle$)
 $\langle proof \rangle$

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*:

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

(is $\langle \text{ready-set } (P \setminus S) = (\text{if } P \setminus S = \perp \text{ then UNIV else ?set} \rangle$)
 $\langle proof \rangle$

In the end the result would look something like this:

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

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

corollary *ready-tick-Hiding-iff* :

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

$\langle proof \rangle$

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

$$\langle \checkmark \in \text{ready-set } P \implies \checkmark \in \text{ready-set } (P \setminus B) \rangle$$

$\langle proof \rangle$

corollary *ready-inside-Hiding-iff* :

$$\langle e \in S \implies \text{ev } e \in \text{ready-set } (P \setminus S) \longleftrightarrow P \setminus S = \perp \rangle$$

$\langle proof \rangle$

corollary *ready-notin-Hiding-iff* :

$\langle e \notin S \implies ev\ e \in \text{ready-set } (P \setminus S) \longleftrightarrow$
 $P \setminus S = \perp \vee$
 $(\exists t. [ev\ e] = \text{trace-hide } t\ (ev\ 'S) \wedge$
 $(t \in \mathcal{D}\ P \vee (\exists f. \text{isInfHiddenRun } f\ P\ S \wedge t \in \text{range } f) \vee (t, ev\ 'S) \in \mathcal{F}$
 $P)) \rangle$
 ⟨proof⟩

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

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

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

lemma *ready-set-MultiDet*:

$\langle \text{finite } A \implies \text{ready-set } (\text{MultiDet } A\ P) = (\bigcup a \in A. \text{ready-set } (P\ a)) \rangle$
 ⟨proof⟩

lemma *ready-set-MultiNdet*:

$\langle \text{finite } A \implies \text{ready-set } (\text{MultiNdet } A\ P) = (\bigcup a \in A. \text{ready-set } (P\ a)) \rangle$
 ⟨proof⟩

lemma *ready-set-GlobalNdet*:

$\langle \text{ready-set } (\text{GlobalNdet } A\ P) = (\bigcup a \in A. \text{ready-set } (P\ a)) \rangle$
 ⟨proof⟩

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 $\text{ready-set } (P\ (\text{hd } L)) \rangle$
 ⟨proof⟩

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⟩

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

lemma *ready-set-Sliding*:

$$\langle \text{ready-set } (P \triangleright Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$$

<proof>

lemma *ready-set-Throw*: $\langle \text{ready-set } (P \ominus a \in A. Q a) = \text{ready-set } P \rangle$

<proof>

corollary *Throw-is-STOP-iff*: $\langle P \ominus a \in A. Q a = \text{STOP} \iff P = \text{STOP} \rangle$

<proof>

lemma *ready-set-Interrupt*: $\langle \text{ready-set } (P \triangle Q) = \text{ready-set } P \cup \text{ready-set } Q \rangle$

<proof>

corollary *Interrupt-is-STOP-iff*: $\langle P \triangle Q = \text{STOP} \iff P = \text{STOP} \wedge Q = \text{STOP} \rangle$

<proof>

3.7 Behaviour of *ready-set* with Reference Processes

lemma *ready-set-DF*: $\langle \text{ready-set } (DF A) = \text{ev } 'A \rangle$

<proof>

lemma *ready-set-DF_SKIP*: $\langle \text{ready-set } (DF_{SKIP} A) = \text{insert } \checkmark (\text{ev } 'A) \rangle$

<proof>

lemma *ready-set-RUN*: $\langle \text{ready-set } (RUN A) = \text{ev } 'A \rangle$

<proof>

lemma *ready-set-CHAOS*: $\langle \text{ready-set } (CHAOS A) = \text{ev } 'A \rangle$

<proof>

lemma *ready-set-CHAOS_SKIP*: $\langle \text{ready-set } (CHAOS_{SKIP} A) = \text{insert } \checkmark (\text{ev } 'A) \rangle$

<proof>

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 $ev\ e \notin ready\text{-set}\ P$ 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 $ev\ e \in ready\text{-set}\ P$ to define operational semantics.

lift-definition *After* :: $\langle [\alpha\ process, \alpha] \Rightarrow \alpha\ process \rangle$ (**infixl** $\langle after \rangle$ 77)

```
is  $\langle \lambda P\ e.\ if\ ev\ e \in ready\text{-set}\ P$ 
  then  $\langle \{(tl\ s, X) \mid s\ X.\ (s, X) \in \mathcal{F}\ P \wedge s \neq [] \wedge hd\ s = ev\ e\},$ 
         $\{tl\ s \mid s.\ s \in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = ev\ e\}$ 
  else  $\langle \{(s, X).\ s = []\}, \{ \} \rangle$ 
```

$\langle proof \rangle$

4.2 Projections

lemma *F-After*:

$\langle \mathcal{F} (P \text{ after } e) = (\text{ if } ev \ e \in \text{ ready-set } P$
 $\text{ then } \{(tl \ s, X) \mid s \ X. (s, X) \in \mathcal{F} \ P \wedge s \neq [] \wedge hd \ s = ev \ e\}$
 $\text{ else } \{(s, X). s = []\}) \rangle$
 $\langle \text{proof} \rangle$

lemma *D-After*:

$\langle \mathcal{D} (P \text{ after } e) = (\text{ if } ev \ e \in \text{ ready-set } P$
 $\text{ then } \{tl \ s \mid s. s \in \mathcal{D} \ P \wedge s \neq [] \wedge hd \ s = ev \ e\}$
 $\text{ else } \{\}\rangle$
 $\langle \text{proof} \rangle$

lemma *T-After*:

$\langle \mathcal{T} (P \text{ after } e) = (\text{ if } ev \ e \in \text{ ready-set } P$
 $\text{ then } \{tl \ s \mid s. s \in \mathcal{T} \ P \wedge s \neq [] \wedge hd \ s = ev \ e\}$
 $\text{ else } \{\{\}\}\rangle$
 $\langle \text{proof} \rangle$

lemma *not-ready-After*: $\langle ev \ e \notin \text{ ready-set } P \implies P \text{ after } e = STOP \rangle$
 $\langle \text{proof} \rangle$

lemma *ready-set-After*: $\langle \text{ready-set} (P \text{ after } e) = \{a. ev \ e \# [a] \in \mathcal{T} \ P\} \rangle$
 $\langle \text{proof} \rangle$

4.3 Monotony

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

lemma *mono-After-T* : $\langle P \sqsubseteq_T Q \implies P \text{ after } e \sqsubseteq_T Q \text{ after } e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-After-F* :
 $\langle P \sqsubseteq_F Q \implies ev \ e \notin \text{ ready-set } P \vee ev \ e \in \text{ ready-set } Q \implies$
 $P \text{ after } e \sqsubseteq_F Q \text{ after } e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-After-D* : $\langle P \sqsubseteq_D Q \implies P \text{ after } e \sqsubseteq_D Q \text{ after } e \rangle$
 $\langle \text{proof} \rangle$

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

lemma *mono-After-DT* : $\langle P \sqsubseteq_{DT} Q \implies P \text{ after } e \sqsubseteq_{DT} Q \text{ after } e \rangle$
 $\langle \text{proof} \rangle$

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

lemma *After-STOP*: $\langle STOP \text{ after } e = STOP \rangle$
 $\langle \text{proof} \rangle$

lemma *After-is-STOP-iff*:
 $\langle P \text{ after } e = STOP \iff (\forall s. \text{ev } e \# s \in \mathcal{T} P \implies s = []) \rangle$
 $\langle \text{proof} \rangle$

lemma *After-SKIP*: $\langle SKIP \text{ after } e = STOP \rangle$
 $\langle \text{proof} \rangle$

lemma *After-BOT*: $\langle \perp \text{ after } e = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *After-is-BOT-iff*: $\langle P \text{ after } e = \perp \iff [ev \ e] \in \mathcal{D} P \rangle$
 $\langle \text{proof} \rangle$

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*:
 $\langle (\Box a \in A \rightarrow P \ a) \text{ after } e = (\Box a \in A \rightarrow P \ a) \text{ after } e \rangle$
 $\langle \text{proof} \rangle$

lemma *After-Det-is-After-Ndet*: $\langle P \sqcap Q \text{ after } e = P \sqcap Q \text{ after } e \rangle$
 $\langle \text{proof} \rangle$

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** $\langle P \sqcap Q \text{ after } e =$
 ($\text{if } ?c1 \text{ then } STOP \text{ else if } ?c2 \text{ then } (P \text{ after } e) \sqcap (Q \text{ after } e) \text{ else}$
 $\text{if } ?c3 \text{ then } P \text{ after } e \text{ else } Q \text{ after } e \rangle$)
 $\langle \text{proof} \rangle$

lemma *After-Mprefix*: $\langle (\Box a \in A \rightarrow P a) \text{ after } e = (\text{if } e \in A \text{ then } P e \text{ else } STOP) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (e \rightarrow P) \text{ after } e = P \rangle \langle \text{proof} \rangle$

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

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

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

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-skippable-or-not-readyR-After-Seq*: $\langle (P ; Q) \text{ after } e = P \text{ after } e ; Q \rangle$
if $\langle \checkmark \notin \text{ready-set } P \vee ev e \notin \text{ready-set } Q \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle (P ; STOP) \text{ after } e = P \text{ after } e ; STOP \rangle$
 $\langle \text{proof} \rangle$

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** $\langle ev e \notin \text{ready-set } P \rangle$
 $\langle \text{proof} \rangle$

lemma *skippable-readyL-readyR-After-Seq*: $\langle (P ; Q) \text{ after } e = (P \text{ after } e ; Q) \sqcap (Q \text{ after } e) \rangle$
if $\langle \checkmark \in \text{ready-set } P \rangle \langle ev e \in \text{ready-set } P \rangle \langle ev e \in \text{ready-set } Q \rangle$
 $\langle \text{proof} \rangle$

lemma *not-readyL-not-skippable-or-not-readyR-After-Seq:*

$\langle ev\ e \notin \text{ready-set } P \implies \checkmark \notin \text{ready-set } P \vee ev\ e \notin \text{ready-set } Q \implies$
 $(P ; Q)\ \text{after } e = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *After-Seq:*

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

4.5.3 After Synchronization

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

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

We will also divide the task.

After version of

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

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

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

$\langle \text{proof} \rangle$

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

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

$\langle ev\ e \notin \text{ready-set } P \implies e \in S \implies$
 $(P \llbracket S \rrbracket Q)\ \text{after } e = (\text{if } Q = \perp \text{ then } \perp \text{ else } \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma not-readyL-not-readyR-After-Sync: $\langle (P \llbracket S \rrbracket Q) \text{ after } e = STOP \rangle$

if ready-hyps: $\langle ev \ e \notin \text{ready-set } P \rangle \langle ev \ e \notin \text{ready-set } Q \rangle$
 $\langle \text{proof} \rangle$

Finally, the monster theorem !

theorem After-Sync:

$\langle (P \llbracket S \rrbracket Q) \text{ after } e =$
 $($
 if $P = \perp \vee Q = \perp$ **then** \perp
 else if $ev \ e \in \text{ready-set } P \wedge ev \ e \in \text{ready-set } Q$
 then if $e \in S$ **then** $P \text{ after } e \llbracket S \rrbracket Q \text{ after } e$
 else $(P \text{ after } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ after } e)$
 else if $e \in S$ **then** $STOP$
 else if $ev \ e \in \text{ready-set } P$ **then** $P \text{ after } e \llbracket S \rrbracket Q$
 else if $ev \ e \in \text{ready-set } Q$ **then** $P \llbracket S \rrbracket Q \text{ after } e$
 else $STOP$)
 \rangle
 $\langle \text{proof} \rangle$

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:

$\langle e \in B \implies (P \setminus B) \sqsubseteq_{FD} (P \text{ after } e \setminus B) \rangle$

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

$\langle \text{proof} \rangle$

lemmas Hiding-F-Hiding-After-if-ready-inside =

$Hiding-FD-Hiding-After-if-ready-inside[THEN \text{ leFD-imp-leF}]$

and After-Hiding-F-Hiding-After-if-ready-notin =

$After-Hiding-FD-Hiding-After-if-ready-notin[THEN \text{ leFD-imp-leF}]$

and Hiding-D-Hiding-After-if-ready-inside =

$Hiding-FD-Hiding-After-if-ready-inside[THEN \text{ leFD-imp-leD}]$

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*:
 $\langle ev\ e \in ready\text{-}set\ P \implies e \in B \implies (P \setminus B) \sqsubseteq_{DT} (P\ after\ e \setminus B) \rangle$
and *After-Hiding-DT-Hiding-After-if-ready-notin*:
 $\langle ev\ e \in ready\text{-}set\ P \implies e \notin B \implies (P \setminus B)\ after\ e \sqsubseteq_{DT} (P\ after\ e \setminus B) \rangle$
<proof>

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 $\langle ev\ e \in ready\text{-}set\ P \wedge f \in B \wedge P \setminus B \neq P\ after\ f \setminus B \rangle$
and $\langle ev\ e \in ready\text{-}set\ P \wedge e \notin B \wedge (P \setminus B)\ after\ e \neq P\ after\ e \setminus B \rangle$
<proof>

4.5.5 After Renaming

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

<proof>

4.6 Behaviour of After with Operators of HOL-CSPM

lemma *After-MultiDet-is-After-MultiNdet*:
 $\langle finite\ A \implies (\sqcap a \in A.\ P\ a)\ after\ e = (\sqcap a \in A.\ P\ a)\ after\ e \rangle$
<proof>

lemma *After-GlobalNdet*: $\langle (\sqcap a \in A.\ P\ a)\ after\ e =$
 $(if\ ev\ e \notin (\bigcup a \in A.\ ready\text{-}set\ (P\ a))\ then\ STOP$
 $else\ (\sqcap a \in \{a \in A.\ ev\ e \in ready\text{-}set\ (P\ a)\}.\ P\ a)\ after\ e) \rangle$
<proof>

lemma *After-MultiNdet*: $\langle finite\ A \implies (\sqcap a \in A.\ P\ a)\ after\ e =$

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

⟨proof⟩

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

⟨proof⟩

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

4.7.1 *After* Sliding

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

⟨proof⟩

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*: $\langle P \triangleright Q \text{ after } e = P \sqcap Q \text{ after } e \rangle$
 ⟨proof⟩

4.7.2 *After* Throwing

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

(is ⟨?lhs = ?rhs⟩)

⟨proof⟩

4.7.3 *After* Interrupting

theorem *After-Interrupt*:
 $\langle (P \triangle Q) \text{ after } e =$
 $(\text{if } ev\ e \in \text{ready-set } P \wedge ev\ e \in \text{ready-set } Q$
 $\text{then } (Q \text{ after } e) \sqcap (P \text{ after } e \triangle Q)$
 $\text{else if } ev\ e \in \text{ready-set } P \wedge ev\ e \notin \text{ready-set } Q$
 $\text{then } P \text{ after } e \triangle Q$
 $\text{else if } ev\ e \notin \text{ready-set } P \wedge ev\ e \in \text{ready-set } Q$
 $\text{then } Q \text{ after } e$
 $\text{else } STOP)$

<proof>

4.8 Behaviour of *After* with Reference Processes

lemma *After-DF*: $\langle DF\ A\ after\ e = (if\ e \in A\ then\ DF\ A\ else\ STOP) \rangle$
<proof>

lemma *After-DF_{SKIP}*:
 $\langle DF_{SKIP}\ A\ after\ e = (if\ e \in A\ then\ DF_{SKIP}\ A\ else\ STOP) \rangle$
<proof>

lemma *After-RUN*: $\langle RUN\ A\ after\ e = (if\ e \in A\ then\ RUN\ A\ else\ STOP) \rangle$
<proof>

lemma *After-CHAOS*: $\langle CHAOS\ A\ after\ e = (if\ e \in A\ then\ CHAOS\ A\ else\ STOP) \rangle$
<proof>

lemma *After-CHAOS_{SKIP}*:
 $\langle CHAOS_{SKIP}\ A\ after\ e = (if\ e \in A\ then\ CHAOS_{SKIP}\ A\ else\ STOP) \rangle$
<proof>

lemma *DF-FD-After*: $\langle DF\ A\ \sqsubseteq_{FD}\ P\ after\ e \rangle$
if $\langle DF\ A\ \sqsubseteq_{FD}\ P \rangle$ **and** $\langle ev\ e \in ready\text{-}set\ P \rangle$
<proof>

lemma *DF_{SKIP}-FD-After*: $\langle DF_{SKIP}\ A\ \sqsubseteq_{FD}\ P\ after\ e \rangle$
if $\langle DF_{SKIP}\ A\ \sqsubseteq_{FD}\ P \rangle$ **and** $\langle ev\ e \in ready\text{-}set\ P \rangle$
<proof>

We have corollaries on *deadlock-free* and *deadlock-free_{SKIP}*.

corollary *deadlock-free-After*:
 $\langle deadlock\text{-}free\ P \implies$
 $deadlock\text{-}free\ (P\ after\ e) \iff (if\ ev\ e \in ready\text{-}set\ P\ then\ True\ else\ False) \rangle$
<proof>

corollary *deadlock-free_{SKIP}-After*:
 $\langle deadlock\text{-}free_{SKIP}\ P \implies$
 $deadlock\text{-}free_{SKIP}\ (P\ after\ e) \iff (if\ ev\ e \in ready\text{-}set\ P\ then\ True\ else\ False) \rangle$
<proof>

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 ::  $\langle ['\alpha$  process,  $'\alpha$  event]  $\Rightarrow$   $'\alpha$  process  $\rangle$  (infixl  $\langle$ 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:*

```
 $\langle e \notin \text{ready-set } P \Longrightarrow P \text{ afterExt } e = (\text{if } P = \perp \text{ then } \perp \text{ else } \text{STOP}) \rangle$ 
 $\langle \text{proof} \rangle$ 
```

lemma *ready-set-AfterExt:*

```
 $\langle \text{ready-set } (P \text{ afterExt } e) =$ 
   $(\text{if } P = \perp \text{ then } \text{UNIV} \text{ else if } e = \checkmark \text{ then } \{\} \text{ else } \{a. e \# [a] \in \mathcal{T} P\}) \rangle$ 
 $\langle \text{proof} \rangle$ 
```

5.1.2 Projections

lemma *F-AfterExt:*

```
 $\langle \mathcal{F} (P \text{ afterExt } e) =$ 
   $( \text{if } e = \checkmark \wedge P = \perp \text{ then } \{(s, X). \text{front-tickFree } s\}$ 
   $\text{else if } e \in \text{ready-set } P \text{ then } \{(tl \ s, X) \mid s \ X. (s, X) \in \mathcal{F} P \wedge s \neq [] \wedge \text{hd } s = e\}$ 
   $\text{else } \{(s, X). s = []\} \rangle$ 
```

(is $\langle - = ?rhs \rangle$)
 $\langle proof \rangle$

lemma *D-AfterExt*:

$\langle \mathcal{D} (P \text{ afterExt } e) = (\text{ if } e = \checkmark \wedge P = \perp \text{ then } \{s. \text{ front-tickFree } s\}$
 $\text{ else } \{tl \ s \mid s . s \in \mathcal{D} P \wedge s \neq [] \wedge hd \ s = e\} \rangle$

(is $\langle - = ?rhs \rangle$)
 $\langle proof \rangle$

lemma *T-AfterExt*:

$\langle \mathcal{T} (P \text{ afterExt } e) = (\text{ if } e = \checkmark \wedge P = \perp \text{ then } \{s. \text{ front-tickFree } s\}$
 $\text{ else insert } [] \ \{tl \ s \mid s . s \in \mathcal{T} P \wedge s \neq [] \wedge hd \ s = e\} \rangle$

(is $\langle - = ?rhs \rangle$)
 $\langle proof \rangle$

5.1.3 Monotony

lemma *mono-AfterExt* : $\langle P \sqsubseteq Q \implies P \text{ afterExt } e \sqsubseteq Q \text{ afterExt } e \rangle$
 $\langle proof \rangle$

lemma *mono-AfterExt-T* : $\langle P \sqsubseteq_T Q \implies e \neq \checkmark \vee P = \perp \vee Q \neq \perp \implies$
 $P \text{ afterExt } e \sqsubseteq_T Q \text{ afterExt } e \rangle$
 $\langle proof \rangle$

lemma *mono-AfterExt-F* :

$\langle P \sqsubseteq_F Q \implies ev \ e \notin \text{ ready-set } P \vee ev \ e \in \text{ ready-set } Q \implies$
 $P \text{ afterExt } ev \ e \sqsubseteq_F Q \text{ afterExt } ev \ e \rangle$

$\langle proof \rangle$

lemma *mono-AfterExt-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterExt } e \sqsubseteq_D Q \text{ afterExt } e \rangle$
 $\langle proof \rangle$

lemma *mono-AfterExt-FD* :

$\langle P \sqsubseteq_{FD} Q \implies e \notin \text{ ready-set } P \vee e \in \text{ ready-set } Q \implies$
 $P \text{ afterExt } e \sqsubseteq_{FD} Q \text{ afterExt } e \rangle$

$\langle proof \rangle$

lemma *mono-AfterExt-DT* : $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterExt } e \sqsubseteq_{DT} Q \text{ afterExt } e \rangle$
 $\langle proof \rangle$

5.1.4 Behaviour of AfterExt with STOP, SKIP and \perp

lemma *AfterExt-STOP*: $\langle STOP \text{ afterExt } e = STOP \rangle$
 $\langle proof \rangle$

lemma *AfterExt-is-STOP-iff*:

$\langle P \text{ afterExt } e = STOP \iff P \neq \perp \wedge (\forall s. e \# s \in \mathcal{T} P \longrightarrow s = []) \rangle$

$\langle proof \rangle$

lemma *AfterExt-SKIP*: $\langle \text{SKIP afterExt } e = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-BOT* : $\langle \perp \text{ afterExt } e = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-is-BOT-iff*: $\langle P \text{ afterExt } e = \perp \iff [e] \in \mathcal{D} P \rangle$
 $\langle \text{proof} \rangle$

5.1.5 Behaviour of *AfterExt* with Operators of HOL-CSP

Here again, we lose determinism.

lemma *AfterExt-Mprefix-is-AfterExt-Mndetprefix*:
 $\langle (\Box a \in A \rightarrow P a) \text{ afterExt } e = (\Box a \in A \rightarrow P a) \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Det-is-AfterExt-Ndet*: $\langle P \sqcap Q \text{ afterExt } e = P \sqcap Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Ndet*:
 $\langle P \sqcap Q \text{ afterExt } e = (\text{if } e \notin \text{ready-set } P \cup \text{ready-set } Q \text{ then } \text{STOP}$
 $\quad \text{else case } e \text{ of } ev \ x \Rightarrow \text{if } ev \ x \in \text{ready-set } P \cap \text{ready-set } Q$
 $\quad \text{then } (P \text{ afterExt } ev \ x) \sqcap (Q \text{ afterExt } ev \ x)$
 $\quad \text{else if } ev \ x \in \text{ready-set } P$
 $\quad \text{then } P \text{ afterExt } ev \ x$
 $\quad \text{else } Q \text{ afterExt } ev \ x$
 $\quad | \checkmark \Rightarrow \text{if } P = \perp \vee Q = \perp \text{ then } \perp \text{ else } \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Mprefix*:
 $\langle (\Box a \in A \rightarrow P a) \text{ afterExt } e =$
 $\quad (\text{case } e \text{ of } ev \ x \Rightarrow \text{if } x \in A \text{ then } P x \text{ else } \text{STOP} | \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

corollary *AfterExt-prefix*:
 $\langle (a \rightarrow P) \text{ afterExt } e =$
 $\quad (\text{case } e \text{ of } ev \ x \Rightarrow \text{if } x = a \text{ then } P \text{ else } \text{STOP} | \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemmas *AfterExt-Det = AfterExt-Ndet*[folded *AfterExt-Det-is-AfterExt-Ndet*]
and *AfterExt-Mndetprefix = AfterExt-Mprefix*
 \quad [unfolded *AfterExt-Mprefix-is-AfterExt-Mndetprefix*]

lemma *Renaming-is-BOT-iff*: $\langle \text{Renaming } P f = \perp \longleftrightarrow P = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *Renaming-is-STOP-iff*: $\langle \text{Renaming } P f = \text{STOP} \longleftrightarrow P = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Renaming*:
 $\langle \text{Renaming } P f \text{ afterExt } e =$
 (if $P = \perp$ then \perp else
 $\sqcap a \in \{a. \text{ ev } a \in \text{ready-set } P \wedge \text{ ev } (f a) = e\}. \text{ Renaming } (P \text{ afterExt } \text{ev } a) f) \rangle$
 $\langle \text{proof} \rangle$

lemma *Seq-is-BOT-iff*: $\langle P ; Q = \perp \longleftrightarrow P = \perp \vee ([\checkmark] \in \mathcal{T} P \wedge Q = \perp) \rangle$
 $\langle \text{proof} \rangle$

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 $P \text{ afterExt } e ; Q) \rangle$
 $\langle \text{proof} \rangle$

theorem *AfterExt-Sync*:
 $\langle (P \llbracket S \rrbracket Q) \text{ afterExt } e =$
 (if $P = \perp \vee Q = \perp$ then \perp
 else case e of $\checkmark \Rightarrow \text{STOP}$
 | $\text{ev } x \Rightarrow$ if $e \in \text{ready-set } P \wedge e \in \text{ready-set } Q$
 then if $x \in S$
 then $P \text{ afterExt } e \llbracket S \rrbracket Q \text{ afterExt } e$
 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 $\text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *Hiding-FD-Hiding-AfterExt-if-ready-inside*:
 $\langle e \in B \implies (P \setminus B) \sqsubseteq_{FD} (P \text{ afterExt } \text{ev } e \setminus B) \rangle$
and *AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin*:
 $\langle e \notin B \implies (P \setminus B) \text{ afterExt } \text{ev } e \sqsubseteq_{FD} (P \text{ afterExt } \text{ev } e \setminus B) \rangle$
if ready: $\langle \text{ev } e \in \text{ready-set } P \rangle$

<proof>

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*:
 $\langle ev\ e \in ready\text{-set}\ P \implies e \notin B \implies (P \setminus B)\ afterExt\ ev\ e \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
<proof>

5.1.6 Behaviour of *AfterExt* with Operators of HOL-CSPM

lemma *AfterExt-MultiDet-is-AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\Box a \in A.\ P\ a)\ afterExt\ e = (\prod a \in A.\ P\ a)\ afterExt\ e \rangle$
<proof>

lemma *AfterExt-GlobalNdet*:
 $\langle (\prod a \in A.\ P\ a)\ afterExt\ e = (\text{if } e \notin (\bigcup a \in A.\ ready\text{-set}\ (P\ a)) \text{ then } STOP \text{ else } (\prod a \in \{a \in A.\ e \in ready\text{-set}\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
and *AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\prod a \in A.\ P\ a)\ afterExt\ e = (\text{if } e \notin (\bigcup a \in A.\ ready\text{-set}\ (P\ a)) \text{ then } STOP \text{ else } (\prod a \in \{a \in A.\ e \in ready\text{-set}\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
and *AfterExt-MultiDet*:
 $\langle finite\ A \implies (\Box a \in A.\ P\ a)\ afterExt\ e = (\text{if } e \notin (\bigcup a \in A.\ ready\text{-set}\ (P\ a)) \text{ then } STOP \text{ else } (\prod a \in \{a \in A.\ e \in ready\text{-set}\ (P\ a)\}.\ P\ a)\ afterExt\ e) \rangle$
<proof>

5.1.7 Behaviour of *AfterExt* with Operators of HOL-CSP_OpSem

lemma *AfterExt-Sliding-is-AfterExt-Ndet*:
 $\langle P \triangleright Q\ afterExt\ e = P \sqcap Q\ afterExt\ e \rangle$

<proof>

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 $ev\ x \in \text{ready-set } P$ then if $x \in A$ then $Q\ x$
else $(P \text{ after } x) \Theta a \in A. Q a$ else $STOP$) \rangle

<proof>

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

<proof>

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
<proof>

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
<proof>

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
<proof>

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
<proof>

lemma *AfterExt-CHAOS_{SKIP}*:
 $\langle \text{CHAOS}_{\text{SKIP}} A \text{ afterExt } e =$
 $(\text{case } e \text{ of } \text{ev } x \Rightarrow \text{if } x \in A \text{ then } \text{CHAOS}_{\text{SKIP}} A \text{ else } \text{STOP} \mid \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *DF-FD-AfterExt*:
 $\langle \text{DF } A \sqsubseteq_{\text{FD}} P \Longrightarrow e \in \text{ready-set } P \Longrightarrow \text{DF } A \sqsubseteq_{\text{FD}} P \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *DF_{SKIP}-FD-AfterExt*:
 $\langle \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \Longrightarrow \text{ev } e \in \text{ready-set } P \Longrightarrow \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *deadlock-free-AfterExt*:
 $\langle \text{deadlock-free } P \Longrightarrow \text{deadlock-free } (P \text{ afterExt } e) \longleftrightarrow$
 $(\text{if } e \in \text{ready-set } P \wedge e \neq \checkmark \text{ then True else False}) \rangle$
 $\langle \text{proof} \rangle$

lemma *deadlock-free_{SKIP}-AfterExt*:
 $\langle \text{deadlock-free}_{\text{SKIP}} P \Longrightarrow \text{deadlock-free}_{\text{SKIP}} (P \text{ afterExt } e) \longleftrightarrow$
 $(\text{if } e \in \text{ready-set } P \wedge e \neq \checkmark \text{ then True else False}) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle P \text{ afterTrace } [] = P \rangle$
 $| \quad \langle P \text{ afterTrace } (e \# s) = P \text{ afterExt } e \text{ afterTrace } s \rangle$

We can also induct backward.

lemma *AfterTrace-append*: $\langle P \text{ afterTrace } (s @ t) = P \text{ afterTrace } s \text{ afterTrace } t \rangle$

<proof>

lemma *AfterTrace-snoc* : $\langle P \text{ afterTrace } (s @ [e]) = P \text{ afterTrace } s \text{ afterExt } e \rangle$
<proof>

We have some immediate properties.

lemma *AfterTrace-BOT* : $\langle \perp \text{ afterTrace } s = \perp \rangle$
<proof>

lemma *AfterTrace-STOP* : $\langle STOP \text{ afterTrace } s = STOP \rangle$
<proof>

lemma *AfterTrace-SKIP* : $\langle SKIP \text{ afterTrace } s = (\text{if } s = [] \text{ then } SKIP \text{ else } STOP) \rangle$
<proof>

5.2.2 Projections

lemma *F-AfterTrace* : $\langle (s @ t, X) \in \mathcal{F} P \implies (t, X) \in \mathcal{F} (P \text{ afterTrace } s) \rangle$
<proof>

lemma *D-AfterTrace* : $\langle s @ t \in \mathcal{D} P \implies t \in \mathcal{D} (P \text{ afterTrace } s) \rangle$
<proof>

lemma *T-AfterTrace* : $\langle s @ t \in \mathcal{T} P \implies t \in \mathcal{T} (P \text{ afterTrace } s) \rangle$
<proof>

corollary *ready-set-AfterTrace* :
 $\langle s @ e \# t \in \mathcal{T} P \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
<proof>

corollary *F-imp-R-AfterTrace*: $\langle (s, X) \in \mathcal{F} P \implies X \in \mathcal{R} (P \text{ afterTrace } s) \rangle$
<proof>

corollary *D-imp-AfterTrace-is-BOT*: $\langle s \in \mathcal{D} P \implies P \text{ afterTrace } s = \perp \rangle$
<proof>

5.2.3 Monotony

lemma *mono-AfterTrace* : $\langle P \sqsubseteq Q \implies P \text{ afterTrace } s \sqsubseteq Q \text{ afterTrace } s \rangle$
<proof>

lemma *mono-AfterTrace-T* :
 $\langle P \sqsubseteq_T Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_T Q \text{ afterTrace } s \rangle$
<proof>

lemma *mono-AfterTrace-F* :
 $\langle P \sqsubseteq_F Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_F Q \text{ afterTrace } s \rangle$
<proof>

lemma *mono-AfterTrace-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterTrace } s \sqsubseteq_D Q \text{ afterTrace } s \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterTrace-FD* :
 $\langle P \sqsubseteq_{FD} Q \implies s \in \mathcal{T} Q \implies P \text{ afterTrace } s \sqsubseteq_{FD} Q \text{ afterTrace } s \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterTrace-DT* :
 $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterTrace } s \sqsubseteq_{DT} Q \text{ afterTrace } s \rangle$
 $\langle \text{proof} \rangle$

5.2.4 Another Definition of *events-of*

inductive *reachable-event* :: $\langle 'a \text{ process} \implies 'a \implies \text{bool} \rangle$
where $\langle \text{ev } e \in \text{ready-set } P \implies \text{reachable-event } P \ e \rangle$
 $| \quad \langle \text{reachable-event } (P \text{ after } f) \ e \implies \text{reachable-event } P \ e \rangle$

lemma *events-of-iff-reachable-event*: $\langle e \in \text{events-of } P \iff \text{reachable-event } P \ e \rangle$
 $\langle \text{proof} \rangle$

lemma *reachable-event-BOT*: $\langle \text{reachable-event } \perp \ e \rangle$
 $\langle \text{proof} \rangle$

lemma *not-reachable-event-STOP*: $\langle \neg \text{reachable-event } \text{STOP} \ e \rangle$
 $\langle \text{proof} \rangle$

lemma *reachable-tick-SKIP*: $\langle \neg \text{reachable-event } \text{SKIP} \ \checkmark \rangle$
 $\langle \text{proof} \rangle$

lemma *reachable-event-iff-in-ready-set-AfterTrace*:
 $\langle \text{reachable-event } P \ e \iff e \in \{e \mid e \text{ s. } \text{ev } e \in \text{ready-set } (P \text{ afterTrace } s)\} \rangle$
 $\langle \text{proof} \rangle$

5.2.5 Characterizations for Deadlock Freeness

lemma *deadlock-free-AfterExt-characterization*:
 $\langle \text{deadlock-free } P \iff \text{range } \text{ev} \notin \mathcal{R} \ P \wedge$
 $(\forall e \in \text{ready-set } P. \text{deadlock-free } (P \text{ afterExt } e)) \rangle$
(is $\langle \text{deadlock-free } P \iff ?\text{rhs} \rangle$
 $\langle \text{proof} \rangle$

lemma *deadlock-free_{SKIP}-AfterExt-characterization*:
 $\langle \text{deadlock-free}_{\text{SKIP}} \ P \iff$
 $\text{UNIV} \notin \mathcal{R} \ P \wedge (\forall e \in \text{ready-set } P - \{\checkmark\}. \text{deadlock-free}_{\text{SKIP}} (P \text{ afterExt } e)) \rangle$

(is $\langle \text{deadlock-free}_{SKIP} P \longleftrightarrow ?rhs \rangle$)
 $\langle proof \rangle$

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 $\tau\text{-trans-is-FD}$: $\langle (\rightsquigarrow_{\tau}) = (\sqsubseteq_{FD}) \rangle$
 $\langle \text{proof} \rangle$

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 $\text{event-trans-prem} :: \langle '\alpha \text{ process} \Rightarrow '\alpha \text{ event} \Rightarrow '\alpha \text{ process} \Rightarrow \text{bool} \rangle$
where

$\tau\text{-left-absorb} : \langle \llbracket e \in \text{ready-set } P'; P \rightsquigarrow_{\tau} P'; \text{event-trans-prem } P' e P'' \rrbracket \implies \text{event-trans-prem } P e P'' \rangle$

| $\tau\text{-right-absorb} : \langle \llbracket e \in \text{ready-set } P; \text{event-trans-prem } P e P'; P' \rightsquigarrow_{\tau} P'' \rrbracket \implies \text{event-trans-prem } P e P'' \rangle$

| $\text{ready-trans-to-AfterExt} : \langle e \in \text{ready-set } P \implies \text{event-trans-prem } P e (P \text{ afterExt } e) \rangle$

abbreviation $\text{event-trans} :: \langle '\alpha \text{ process} \Rightarrow '\alpha \text{ event} \Rightarrow '\alpha \text{ process} \Rightarrow \text{bool} \rangle$
 $(\langle _ \rightsquigarrow _ / _ \rangle [50, 3, 51] 50)$

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 $\text{startable-imp-ev-trans-is-startable-and-FD-After}$:

$\langle (P \rightsquigarrow_e P') \iff e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow_{\tau} P' \rangle$
 $\langle \text{proof} \rangle$

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-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-event-trans-anything: $\langle \perp \rightsquigarrow_e P \rangle$
 $\langle \text{proof} \rangle$

As immediate consequences of the axioms, we prove that event transitions absorb τ transitions on right and on left.

lemma event-trans- τ -trans: $\langle P \rightsquigarrow_e P' \implies P' \rightsquigarrow_\tau P'' \implies P \rightsquigarrow_e P'' \rangle$
 $\langle \text{proof} \rangle$

lemma τ -trans-event-trans: $\langle P \rightsquigarrow_\tau P' \implies P' \rightsquigarrow_e P'' \implies P \rightsquigarrow_e P'' \rangle$
 $\langle \text{proof} \rangle$

We can now define the concept of transition with a trace and demonstrate the first properties.

inductive trace-trans :: $\langle 'a \text{ process} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$
 $(\langle - / \rightsquigarrow^* - / - \rangle [50, 3, 51] 50)$
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 :
 $\langle P \rightsquigarrow(\text{ev } e) P' \implies P' \rightsquigarrow^* s P'' \implies P \rightsquigarrow^* (\text{ev } e) \# s P'' \rangle$

lemma trace-trans- τ -trans: $\langle P \rightsquigarrow^* s P' \implies P' \rightsquigarrow_\tau P'' \implies P \rightsquigarrow^* s P'' \rangle$
 $\langle \text{proof} \rangle$

lemma τ -trans-trace-trans: $\langle P \rightsquigarrow_\tau P' \implies P' \rightsquigarrow^* s P'' \implies P \rightsquigarrow^* s P'' \rangle$
 $\langle \text{proof} \rangle$

lemma BOT-trace-trans-anything : $\langle \text{front-tickFree } s \implies \perp \rightsquigarrow^* s P \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q \rangle$
 $\langle \text{proof} \rangle$

lemma trace-trans-imp-T-if- τ -trans-imp-leT: $\langle P \rightsquigarrow^* s Q \implies s \in \mathcal{T} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-trans-BOT-imp-D-if- τ -trans-imp-leD*: $\langle P \rightsquigarrow^* s \perp \implies s \in \mathcal{D} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_D Q \rangle$
 $\langle \text{proof} \rangle$

7.3 Characterizations for $P \rightsquigarrow^* s Q$

lemma *trace-trans-iff* :
 $\langle P \rightsquigarrow^* [] Q \iff P \rightsquigarrow_\tau Q \rangle$
 $\langle P \rightsquigarrow^* [\checkmark] Q \iff P \rightsquigarrow_{\checkmark} Q \rangle$
 $\langle P \rightsquigarrow^* (ev\ e) \# s Q' \iff (\exists Q. P \rightsquigarrow (ev\ e) Q \wedge Q \rightsquigarrow^* s Q') \rangle$
 $\langle tickFree\ s \implies (P \rightsquigarrow^* s @ [f] Q') \iff (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow_f Q') \rangle$
 $\langle front-tickFree\ (s @ t) \implies (P \rightsquigarrow^* s @ t Q') \iff (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t Q') \rangle$
 $\langle \text{proof} \rangle$

7.4 Finally: $P \rightsquigarrow^* s Q$ is P afterTrace $s \rightsquigarrow_\tau Q$

theorem *T-imp-trace-trans-iff-AfterTrace- τ -trans* :
 $\langle s \in \mathcal{T} P \implies (P \rightsquigarrow^* s Q) \iff P \text{ afterTrace } s \rightsquigarrow_\tau Q \rangle$
 $\langle \text{proof} \rangle$

As corollaries we obtain the reciprocal results of

$\llbracket \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_F Q; ?P \rightsquigarrow^* ?s ?Q; ?X \in \mathcal{R} ?Q \rrbracket \implies (?s, ?X) \in \mathcal{F} ?P$

$\llbracket \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_T Q; ?P \rightsquigarrow^* ?s ?Q \rrbracket \implies ?s \in \mathcal{T} ?P$

$\llbracket \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_D Q; ?P \rightsquigarrow^* ?s \perp \rrbracket \implies ?s \in \mathcal{D} ?P$

lemma *F-imp-exists-trace-trans*: $\langle (s, X) \in \mathcal{F} P \implies \exists Q. (P \rightsquigarrow^* s Q) \wedge X \in \mathcal{R} Q \rangle$
 $\langle \text{proof} \rangle$

lemma *T-imp-exists-trace-trans*: $\langle s \in \mathcal{T} P \implies \exists Q. P \rightsquigarrow^* s Q \rangle$
 $\langle \text{proof} \rangle$

lemma *D-imp-trace-trans-BOT*: $\langle s \in \mathcal{D} P \implies P \rightsquigarrow^* s \perp \rangle$
 $\langle \text{proof} \rangle$

When we have more information on $P \rightsquigarrow_\tau Q$, we obtain:

lemma *τ -trans-imp-leT-imp-STOP-trace-trans-iff*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_T Q \implies STOP \rightsquigarrow^* s P \iff s = [] \wedge P = STOP \rangle$
 $\langle \text{proof} \rangle$

lemma *τ -trans-imp-leF-imp-SKIP-trace-trans-iff*:
 $\langle \forall P Q. P \rightsquigarrow_\tau Q \implies P \sqsubseteq_F Q \implies$
 $SKIP \rightsquigarrow^* s P \iff s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle$
 $\langle \text{proof} \rangle$

lemma τ -trans-imp-leT-imp-trace-trans-ready-set-subset-ready-set-AfterTrace:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow P \rightsquigarrow^* s Q \Longrightarrow$
 $\text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

lemma τ -trans-imp-leT-imp-trace-trans-imp-ready-set:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow P \rightsquigarrow^*(s @ e \# t) Q \Longrightarrow$
 $e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

lemma trace-trans-iff-T-and-AfterTrace- τ -trans-if- τ -trans-imp-leT:
 $\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow$
 $(P \rightsquigarrow^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \rightsquigarrow_{\tau} Q \rangle$
 $\langle \text{proof} \rangle$

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: $\langle \text{SKIP} \rightsquigarrow \checkmark \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma tick-trans-imp-BOT-L-or-STOP-R: $\langle P \rightsquigarrow \checkmark Q \Longrightarrow P = \perp \vee Q = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma STOP-trace-trans-iff : $\langle \text{STOP} \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma ready-tick-imp- τ -trans-SKIP: $\langle P \rightsquigarrow_{\tau} \text{SKIP} \rangle$ **if** $\langle \checkmark \in \text{ready-set } P \rangle$
 $\langle \text{proof} \rangle$

lemma exists-tick-trans-is-ready-tick: $\langle (\exists P'. P \rightsquigarrow \checkmark P') \longleftrightarrow \checkmark \in \text{ready-set } P \rangle$
 $\langle \text{proof} \rangle$

lemma tick-trans-iff : $\langle P \rightsquigarrow \checkmark P' \longleftrightarrow P = \perp \vee P \rightsquigarrow_{\tau} \text{SKIP} \wedge P' = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma SKIP-cant-ev-trans: $\langle \neg \text{SKIP} \rightsquigarrow (ev e) \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *STOP-cant-event-trans*: $\langle \neg STOP \rightsquigarrow_e P \rangle$
 $\langle \text{proof} \rangle$

lemma *ev-trans-Mprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P a \rightsquigarrow (ev\ e) (P\ e) \rangle$
 $\langle \text{proof} \rangle$

lemma *ev-trans-Mndetprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P a \rightsquigarrow (ev\ e) (P\ e) \rangle$
 $\langle \text{proof} \rangle$

lemma *ev-trans-prefix*: $\langle e \rightarrow P \rightsquigarrow (ev\ e) P \rangle$
 $\langle \text{proof} \rangle$

lemma *τ -trans-MultiNdet*: $\langle \text{finite } A \implies x \in A \implies (\Box a \in A. P\ a) \rightsquigarrow_\tau P\ x \rangle$
 $\langle \text{proof} \rangle$

lemma *τ -trans-GlobalNdet*: $\langle (\Box a \in A. P\ a) \rightsquigarrow_\tau P\ e \rangle$ **if** $\langle e \in A \rangle$
 $\langle \text{proof} \rangle$

lemma *fix-point- τ -trans*: $\langle \text{cont } f \implies P = (\mu X. f\ X) \implies P \rightsquigarrow_\tau f\ P \rangle$
 $\langle \text{proof} \rangle$

lemma *event-trans-DetL*: $\langle P \rightsquigarrow_e P' \implies P \Box Q \rightsquigarrow_e P' \rangle$
 $\langle \text{proof} \rangle$

lemma *event-trans-DetR*: $\langle Q \rightsquigarrow_e Q' \implies P \Box Q \rightsquigarrow_e Q' \rangle$
 $\langle \text{proof} \rangle$

lemma *event-trans-MultiDet*:
 $\langle \text{finite } A \implies a \in A \implies P a \rightsquigarrow_e Q \implies (\Box a \in A. P\ a) \rightsquigarrow_e Q \rangle$
 $\langle \text{proof} \rangle$

lemma *Sliding-event-transL*: $\langle P \rightsquigarrow_e P' \implies P \triangleright Q \rightsquigarrow_e P' \rangle$
 $\langle \text{proof} \rangle$

lemma *Sliding- τ -transR*: $\langle P \triangleright Q \rightsquigarrow_\tau Q \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle \exists P P' Q. P \rightsquigarrow_{\checkmark} P' \wedge \neg P ; Q \rightsquigarrow_{\checkmark} P' ; Q \rangle$
 $\langle proof \rangle$

lemma *ev-trans-SeqR*:
 $\langle \checkmark \in \text{ready-set } P \implies Q \rightsquigarrow (ev\ e) Q' \implies P ; Q \rightsquigarrow (ev\ e) Q' \rangle$
 $\langle proof \rangle$

lemma $\langle SKIP \llbracket S \rrbracket SKIP \rightsquigarrow_{\checkmark} STOP \rangle$
 $\langle proof \rangle$

lemma $\langle SKIP \llbracket S \rrbracket SKIP \rightsquigarrow_{\tau} SKIP \rangle$
 $\langle proof \rangle$

lemma *tick-trans-Hiding*: $\langle P \setminus B \rightsquigarrow_{\checkmark} STOP \rangle$ **if** $\langle P \rightsquigarrow_{\checkmark} P' \rangle$
 $\langle proof \rangle$

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 $\langle Renaming\ P\ f \rightsquigarrow_{\checkmark} STOP \rangle$ **if** $\langle P \rightsquigarrow_{\checkmark} P' \rangle$
 $\langle proof \rangle$

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  $\tau$ -trans :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool' (infixl  $\langle_{FD \rightsquigarrow \tau} \rangle$  50)
  where  $\langle P \rangle_{FD \rightsquigarrow \tau} Q \equiv P \sqsubseteq_{FD} Q$ 
```

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGeneric*.

```
interpretation OpSemGeneric  $\langle_{(FD \rightsquigarrow \tau)} \rangle$ 
   $\langle$ proof $\rangle$ 
```

```
notation event-trans ( $\langle - \rangle_{FD \rightsquigarrow} - \rangle$  [50, 3, 51] 50)
```

```
notation trace-trans ( $\langle - \rangle_{FD \rightsquigarrow^*} - \rangle$  [50, 3, 51] 50)
```

```
lemma  $\langle P \rangle_{FD \rightsquigarrow} e P' \Longrightarrow P' \rangle_{FD \rightsquigarrow \tau} P'' \Longrightarrow P \rangle_{FD \rightsquigarrow} e P''$ 
   $\langle P \rangle_{FD \rightsquigarrow \tau} P' \Longrightarrow P' \rangle_{FD \rightsquigarrow} e P'' \Longrightarrow P \rangle_{FD \rightsquigarrow} e P''$ 
   $\langle$ proof $\rangle$ 
```

8.1 Operational Semantics Laws

SKIP law

```
lemma  $\langle SKIP \rangle_{FD \rightsquigarrow} \checkmark STOP$ 
   $\langle$ proof $\rangle$ 
```

$e \rightarrow P$ laws

```
lemma  $\langle e \in A \Longrightarrow \square a \in A \rightarrow P a \rangle_{FD \rightsquigarrow} (ev e) (P e)$ 
   $\langle$ proof $\rangle$ 
```

lemma $\langle e \in A \implies \prod a \in A \rightarrow P a \text{ }_{FD \rightsquigarrow} (ev\ e) (P\ e) \rangle$
 $\langle proof \rangle$

lemma $\langle e \rightarrow P \text{ }_{FD \rightsquigarrow} (ev\ e) P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle a \in A \implies (\prod a \in A. P a) \text{ }_{FD \rightsquigarrow \tau} P a \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies (\prod a \in A. P a) \text{ }_{FD \rightsquigarrow \tau} P a \rangle$
 $\langle proof \rangle$

$\mu\ x. f\ x$ law

lemma $\langle cont\ f \implies P = (\mu\ X. f\ X) \implies P \text{ }_{FD \rightsquigarrow \tau} f\ P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \text{ }_{FD \rightsquigarrow \tau} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \text{ }_{FD \rightsquigarrow \tau} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} P \sqcap Q' \rangle$
 $\langle proof \rangle$

lemma τ -trans-MultiDet:
 $\langle finite\ A \implies \forall a \in A. P a \text{ }_{FD \rightsquigarrow \tau} P' a \implies$
 $(\prod a \in A. P a) \text{ }_{FD \rightsquigarrow \tau} (\prod a \in A. P' a) \rangle$
 $\langle proof \rangle$

lemma $\langle P \text{ }_{FD \rightsquigarrow e} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow e} P' \rangle$
and $\langle Q \text{ }_{FD \rightsquigarrow e} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow e} Q' \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies P a \text{ }_{FD \rightsquigarrow e} Q \implies (\prod a \in A. P a) \text{ }_{FD \rightsquigarrow e} Q \rangle$
 $\langle proof \rangle$

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{FD \rightsquigarrow \tau} P' \implies P ; Q \text{ }_{FD \rightsquigarrow \tau} P' ; Q \rangle$
 $\langle proof \rangle$

lemma ev -trans-SeqL: $\langle P \text{ }_{FD \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{FD \rightsquigarrow} (ev\ e) P' ; Q \rangle$
 $\langle proof \rangle$

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ FD}\rightsquigarrow\checkmark P' \implies P ; Q \text{ FD}\rightsquigarrow\tau Q \rangle$
 ⟨proof⟩

lemma $\checkmark \in \text{ready-set } P \implies Q \text{ FD}\rightsquigarrow(\text{ev } e) Q' \implies P ; Q \text{ FD}\rightsquigarrow(\text{ev } e) Q' \rangle$
 ⟨proof⟩

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ FD}\rightsquigarrow\tau P' \implies P \setminus B \text{ FD}\rightsquigarrow\tau P' \setminus B \rangle$
 ⟨proof⟩

lemma *ev-trans-Hiding-notin*:
 $\langle P \text{ FD}\rightsquigarrow(\text{ev } e) P' \implies e \notin B \implies P \setminus B \text{ FD}\rightsquigarrow(\text{ev } e) P' \setminus B \rangle$
 ⟨proof⟩

lemma $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies P \setminus B \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$
 ⟨proof⟩

lemma *ev-trans-Hiding-inside*:
 $\langle P \text{ FD}\rightsquigarrow(\text{ev } e) P' \implies e \in B \implies P \setminus B \text{ FD}\rightsquigarrow\tau P' \setminus B \rangle$
 ⟨proof⟩

Renaming P f laws

lemma τ -trans-Renaming:
 $\langle P \text{ FD}\rightsquigarrow\tau P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow\tau \text{Renaming } P' f \rangle$
 ⟨proof⟩

lemma *tick-trans-Renaming*: $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$
 ⟨proof⟩

lemma *ev-trans-Renaming*:
 $\langle f a = b \implies P \text{ FD}\rightsquigarrow(\text{ev } a) P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow(\text{ev } b) (\text{Renaming } P' f) \rangle$
 ⟨proof⟩

lemma $\langle P \text{ FD}\rightsquigarrow\tau P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow\tau P' \llbracket a := b \rrbracket \rangle$
 ⟨proof⟩

lemma $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$
 ⟨proof⟩

lemma *ev-trans-RenamingF*:
 $\langle P \text{ FD}\rightsquigarrow(\text{ev } a) P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow(\text{ev } b) P' \llbracket a := b \rrbracket \rangle$
 ⟨proof⟩

$P \llbracket S \rrbracket Q$ laws

lemma τ -trans-SyncL: $\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} P' \llbracket S \rrbracket Q \rangle$
and τ -trans-SyncR: $\langle Q \text{ }_{FD\rightsquigarrow\tau} Q' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} P \llbracket S \rrbracket Q' \rangle$
<proof>

lemma *ev-trans-SyncL*:
 $\langle e \notin S \implies P \text{ }_{FD\rightsquigarrow}(ev\ e) P' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow}(ev\ e) P' \llbracket S \rrbracket Q \rangle$
and *ev-trans-SyncR*:
 $\langle e \notin S \implies Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow}(ev\ e) P \llbracket S \rrbracket Q' \rangle$
<proof>

lemma *ev-trans-SyncLR*:
 $\langle \llbracket e \in S; P \text{ }_{FD\rightsquigarrow}(ev\ e) P'; Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \rrbracket \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow}(ev\ e) P' \llbracket S \rrbracket Q' \rangle$
<proof>

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*: $\langle P \text{ }_{FD\rightsquigarrow}\checkmark P' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \text{ }_{FD\rightsquigarrow}\checkmark Q' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} P \llbracket S \rrbracket SKIP \rangle$
<proof>

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \text{ }_{FD\rightsquigarrow}\checkmark STOP \rangle$
<proof>

lemma τ -trans-SKIP-Sync-SKIP: $\langle SKIP \llbracket S \rrbracket SKIP \text{ }_{FD\rightsquigarrow\tau} SKIP \rangle$
<proof>

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \triangleright Q \text{ }_{FD\rightsquigarrow\tau} P' \triangleright Q \rangle$
<proof>

lemma $\langle P \text{ }_{FD\rightsquigarrow}e P' \implies P \triangleright Q \text{ }_{FD\rightsquigarrow}e P' \rangle$
<proof>

lemma $\langle P \triangleright Q \text{ }_{FD\rightsquigarrow\tau} Q \rangle$
<proof>

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \triangle Q \text{ }_{FD\rightsquigarrow\tau} P' \triangle Q \rangle$
<proof>

lemma *Interrupt- τ -trans-right*: $\langle Q \text{ }_{FD\rightsquigarrow\tau} Q' \implies P \triangle Q \text{ }_{FD\rightsquigarrow\tau} P \triangle Q' \rangle$
<proof>

lemma *Interrupt-ev-trans-left*:
 $\langle P \text{ }_{FD\rightsquigarrow}(ev\ e) P' \implies P \triangle Q \text{ }_{FD\rightsquigarrow}(ev\ e) P' \triangle Q \rangle$
<proof>

lemma *Interrupt-ev-trans-right*: $\langle Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \implies P \triangle Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \rangle$

⟨proof⟩

Throw $P A Q$ laws

lemma *Throw- τ -trans-left:*

$\langle P \text{ }_{FD \rightsquigarrow \tau} P' \implies P \Theta a \in A. Q a \text{ }_{FD \rightsquigarrow \tau} P' \Theta a \in A. Q a \rangle$
⟨proof⟩

lemma *Throw- τ -trans-right:*

$\langle \forall a \in A. Q a \text{ }_{FD \rightsquigarrow \tau} Q' a \implies P \Theta a \in A. Q a \text{ }_{FD \rightsquigarrow \tau} P \Theta a \in A. Q' a \rangle$
⟨proof⟩

lemma *Throw-event-trans-left:*

$\langle P \text{ }_{FD \rightsquigarrow e} P' \implies e \notin \text{ev } A \implies P \Theta a \in A. Q a \text{ }_{FD \rightsquigarrow e} (P' \Theta a \in A. Q a) \rangle$
⟨proof⟩

lemma *Throw-ev-trans-right:*

$\langle P \text{ }_{FD \rightsquigarrow (ev e)} P' \implies e \in A \implies P \Theta a \in A. Q a \text{ }_{FD \rightsquigarrow (ev e)} (Q e) \rangle$
⟨proof⟩

lemma *⟨front-tickFree $s \implies \perp \text{ }_{FD \rightsquigarrow^*} s P \rangle$*
⟨proof⟩

8.2 Reality Checks

lemma $\langle STOP \text{ }_{FD \rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = STOP \rangle$
⟨proof⟩

lemma *SKIP-trace-trans-iff :*

$\langle SKIP \text{ }_{FD \rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle$
⟨proof⟩

lemma *F-iff-exists-trans :*

$\langle (s, X) \in \mathcal{F} P \longleftrightarrow (\exists P'. (P \text{ }_{FD \rightsquigarrow^*} s P') \wedge X \in \mathcal{R} P') \rangle$
⟨proof⟩

lemma *T-iff-exists-trans :* $\langle s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \text{ }_{FD \rightsquigarrow^*} s P') \rangle$
⟨proof⟩

lemma *D-iff-trace-trans-BOT:* $\langle s \in \mathcal{D} P \longleftrightarrow P \text{ }_{FD \rightsquigarrow^*} s \perp \rangle$
⟨proof⟩

8.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace:*

$$\langle P \text{ }_{FD\rightsquigarrow}^* s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$$

\langle proof \rangle

lemma *trace-trans-imp-ready-set:*

$$\langle P \text{ }_{FD\rightsquigarrow}^*(s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$$

\langle proof \rangle

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT :*

$$\langle (P \text{ }_{FD\rightsquigarrow}^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \text{ }_{FD\rightsquigarrow\tau} Q \rangle$$

\langle proof \rangle

lemma *\langle deadlock-free P \longleftrightarrow DF UNIV \text{ }_{FD\rightsquigarrow\tau} P \rangle*

\langle proof \rangle

lemma *\langle deadlock-free_{SKIP} P \longleftrightarrow DF_{SKIP} UNIV \text{ }_{FD\rightsquigarrow\tau} P \rangle*

\langle proof \rangle

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

$$\overline{\text{SKIP }_{FD\rightsquigarrow} \checkmark \text{ STOP}}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P a \text{ }_{FD\rightsquigarrow} ev ?e ?P ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P a \text{ }_{FD\rightsquigarrow} ev ?e ?P ?e}$$

$$\overline{?e \rightarrow ?P \text{ }_{FD\rightsquigarrow} ev ?e ?P}$$

(\sqcap) rules

$$\frac{\frac{\overline{?P \sqcap ?Q}_{FD \rightsquigarrow \tau} ?P} \quad \frac{\overline{?P \sqcap ?Q}_{FD \rightsquigarrow \tau} ?Q}}{?e \in ?A}}{\sqcap a \in ?A. ?P a_{FD \rightsquigarrow \tau} ?P ?e}$$

$\mu x. f x$ rule

$$\frac{cont ?f \quad ?P = (\mu x. ?f x)}{?P_{FD \rightsquigarrow \tau} ?f ?P}$$

(\sqcup) rules

$$\frac{\frac{\overline{?P}_{FD \rightsquigarrow \tau} ?P'}}{?P \sqcup ?Q_{FD \rightsquigarrow \tau} ?P' \sqcup ?Q} \quad \frac{\overline{?Q}_{FD \rightsquigarrow \tau} ?Q'}}{?P \sqcup ?Q_{FD \rightsquigarrow \tau} ?P \sqcup ?Q'}}{\frac{\overline{?P}_{FD \rightsquigarrow ?e} ?P'}{?P \sqcup ?Q_{FD \rightsquigarrow ?e} ?P'} \quad \frac{\overline{?Q}_{FD \rightsquigarrow ?e} ?Q'}}{?P \sqcup ?Q_{FD \rightsquigarrow ?e} ?Q'}}$$

(;) rules

$$\frac{\overline{?P}_{FD \rightsquigarrow \tau} ?P'}}{?P ; ?Q_{FD \rightsquigarrow \tau} ?P' ; ?Q} \quad \frac{\overline{?P}_{FD \rightsquigarrow ev} ?e ?P'}}{?P ; ?Q_{FD \rightsquigarrow ev} ?e ?P' ; ?Q}}{\frac{\exists P'. ?P_{FD \rightsquigarrow \checkmark} P'}{?P ; ?Q_{FD \rightsquigarrow \tau} ?Q}}$$

Hiding rules

$$\frac{\overline{?P}_{FD \rightsquigarrow \tau} ?P'}}{?P \setminus ?B_{FD \rightsquigarrow \tau} ?P' \setminus ?B} \quad \frac{\overline{?P}_{FD \rightsquigarrow \checkmark} ?P'}}{?P \setminus ?B_{FD \rightsquigarrow \checkmark} STOP}}{\frac{\overline{?P}_{FD \rightsquigarrow ev} ?e ?P'}{?P \setminus ?B_{FD \rightsquigarrow ev} ?e ?P' \setminus ?B} \quad \frac{\overline{?P}_{FD \rightsquigarrow ev} ?e ?P'}{?P \setminus ?B_{FD \rightsquigarrow \tau} ?P' \setminus ?B} \quad ?e \in ?B}}$$

Renaming rules

$$\begin{array}{c}
\frac{\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{\text{Renaming } ?P \text{ }_{?f \text{ }_{FD \rightsquigarrow \tau}} \text{Renaming } ?P' \text{ }_{?f}}{?P \text{ }_{FD \rightsquigarrow \checkmark} ?P'}}{\text{Renaming } ?P \text{ }_{?f \text{ }_{FD \rightsquigarrow \checkmark}} \text{STOP}}}{?f \text{ }_{?a = ?b} \quad ?P \text{ }_{FD \rightsquigarrow ev} ?a \text{ }_{?P'}} \\
\hline
\text{Renaming } ?P \text{ }_{?f \text{ }_{FD \rightsquigarrow ev} ?b} \text{Renaming } ?P' \text{ }_{?f}
\end{array}$$

Sync rules

$$\begin{array}{c}
\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \text{ }_{[[?S]]} ?Q} \quad \frac{\frac{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}{?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \text{ }_{[[?S]]} ?Q'}}{\frac{?e \notin ?S \quad ?P \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?P'}}{?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?P' \text{ }_{[[?S]]} ?Q}}}{\frac{?e \notin ?S \quad ?Q \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?Q'}}{?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?P \text{ }_{[[?S]]} ?Q'}}} \\
?e \in ?S \quad ?P \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?P'} \quad ?Q \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?Q'} \\
\hline
?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow ev} ?e \text{ }_{?P' \text{ }_{[[?S]]} ?Q'} \\
\frac{?P \text{ }_{FD \rightsquigarrow \checkmark} ?P'}{?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow \tau} \text{SKIP} \text{ }_{[[?S]]} ?Q} \\
\frac{?Q \text{ }_{FD \rightsquigarrow \checkmark} ?Q'}{?P \text{ }_{[[?S]]} ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \text{ }_{[[?S]]} \text{SKIP}} \\
\hline
\text{SKIP} \text{ }_{[[?S]]} \text{SKIP} \text{ }_{FD \rightsquigarrow \tau} \text{SKIP}
\end{array}$$

(\triangleright) rules

$$\begin{array}{c}
\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \triangleright ?Q} \quad \frac{?P \text{ }_{FD \rightsquigarrow ?e} ?P'}{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow ?e} ?P'} \\
\hline
?P \triangleright ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q
\end{array}$$

(\triangle) rules

$$\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \triangle ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \triangle ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}{?P \triangle ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \triangle ?Q'}$$

$$\frac{?P \text{ FD} \rightsquigarrow \text{ev } ?e \text{ } ?P'}{?P \Delta ?Q \text{ FD} \rightsquigarrow \text{ev } ?e \text{ } ?P' \Delta ?Q} \quad \frac{?Q \text{ FD} \rightsquigarrow \text{ev } ?e \text{ } ?Q'}{?P \Delta ?Q \text{ FD} \rightsquigarrow \text{ev } ?e \text{ } ?Q'}$$

Throw rules

$$\frac{\frac{?P \text{ FD} \rightsquigarrow_{\tau} ?P'}{?P \Theta a \in ?A. ?Q a \text{ FD} \rightsquigarrow_{\tau} ?P' \Theta a \in ?A. ?Q a \quad \forall a \in ?A. ?Q a \text{ FD} \rightsquigarrow_{\tau} ?Q' a}}{?P \Theta a \in ?A. ?Q a \text{ FD} \rightsquigarrow_{\tau} ?P \Theta a \in ?A. ?Q' a} \quad \frac{?P \text{ FD} \rightsquigarrow ?e \text{ } ?P' \quad ?e \notin \text{ev } ' ?A}{?P \Theta a \in ?A. ?Q a \text{ FD} \rightsquigarrow ?e \text{ } ?P' \Theta a \in ?A. ?Q a \quad ?P \text{ FD} \rightsquigarrow \text{ev } ?e \text{ } ?P' \quad ?e \in ?A}}{?P \Theta a \in ?A. ?Q a \text{ FD} \rightsquigarrow \text{ev } ?e \text{ } ?Q ?e}$$

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 ::  $\langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$  (infixl  $\langle_{DT \rightsquigarrow \tau} \rangle$  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  $\langle_{(DT \rightsquigarrow \tau)} \rangle$ 
   $\langle \text{proof} \rangle$ 
```

```
notation event-trans ( $\langle - /_{DT \rightsquigarrow} - / \rightarrow [50, 3, 51] 50$ )
```

```
notation trace-trans ( $\langle - /_{DT \rightsquigarrow^*} - / \rightarrow [50, 3, 51] 50$ )
```

```
lemma  $\langle P \rangle_{DT \rightsquigarrow} e P' \Longrightarrow P'_{DT \rightsquigarrow \tau} P'' \Longrightarrow P \rangle_{DT \rightsquigarrow} e P'' \rangle$ 
   $\langle P \rangle_{DT \rightsquigarrow \tau} P' \Longrightarrow P'_{DT \rightsquigarrow} e P'' \Longrightarrow P \rangle_{DT \rightsquigarrow} e P'' \rangle$ 
   $\langle \text{proof} \rangle$ 
```

9.1 Operational Semantics Laws

SKIP law

```
lemma  $\langle \text{SKIP} \rangle_{DT \rightsquigarrow} \checkmark \text{STOP} \rangle$ 
   $\langle \text{proof} \rangle$ 
```

$e \rightarrow P$ laws

```
lemma  $\langle e \in A \Longrightarrow \square a \in A \rightarrow P a \rangle_{DT \rightsquigarrow} (ev e) (P e) \rangle$ 
   $\langle \text{proof} \rangle$ 
```

lemma $\langle e \in A \implies \prod a \in A \rightarrow P a \text{ }_{DT \rightsquigarrow} (ev\ e)\ (P\ e) \rangle$
 $\langle proof \rangle$

lemma $\langle e \rightarrow P \text{ }_{DT \rightsquigarrow} (ev\ e)\ P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle a \in A \implies (\prod a \in A.\ P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies (\prod a \in A.\ P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
 $\langle proof \rangle$

$\mu\ x.\ f\ x$ law

lemma $\langle cont\ f \implies P = (\mu\ X.\ f\ X) \implies P \text{ }_{DT \rightsquigarrow \tau} f\ P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws, more powerful

lemma τ -trans-DetL: $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \rangle$
and τ -trans-DetR: $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma τ -trans-MultiDet: $\langle finite\ A \implies a \in A \implies (\prod a \in A.\ P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
 $\langle proof \rangle$

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} P' ; Q \rangle$
 $\langle proof \rangle$

lemma ev -trans-SeqL: $\langle P \text{ }_{DT \rightsquigarrow} (ev\ e)\ P' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev\ e)\ P' ; Q \rangle$
 $\langle proof \rangle$

lemma τ -trans-SeqR: $\langle \exists P'.\ P \text{ }_{DT \rightsquigarrow} \checkmark\ P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle \checkmark \in ready\ set\ P \implies Q \text{ }_{DT \rightsquigarrow} (ev\ e)\ Q' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev\ e)\ Q' \rangle$
 $\langle proof \rangle$

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \setminus B \text{ }_{DT \rightsquigarrow \tau} P' \setminus B \rangle$

<proof>

lemma *ev-trans-Hiding-notin:*

$\langle P \text{ }_{DT\rightsquigarrow}(ev\ e)\ P' \implies e \notin B \implies P \setminus B \text{ }_{DT\rightsquigarrow}(ev\ e)\ P' \setminus B \rangle$
<proof>

lemma $\langle P \text{ }_{DT\rightsquigarrow}\checkmark\ P' \implies P \setminus B \text{ }_{DT\rightsquigarrow}\checkmark\ STOP \rangle$

<proof>

lemma *ev-trans-Hiding-inside:*

$\langle P \text{ }_{DT\rightsquigarrow}(ev\ e)\ P' \implies e \in B \implies P \setminus B \text{ }_{DT\rightsquigarrow\tau}\ P' \setminus B \rangle$
<proof>

Renaming P f laws

lemma *τ -trans-Renaming:*

$\langle P \text{ }_{DT\rightsquigarrow\tau}\ P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow\tau}\ \text{Renaming } P'\ f \rangle$
<proof>

lemma *tick-trans-Renaming:* $\langle P \text{ }_{DT\rightsquigarrow}\checkmark\ P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow}\checkmark\ STOP \rangle$

<proof>

lemma *ev-trans-Renaming:*

$\langle f\ a = b \implies P \text{ }_{DT\rightsquigarrow}(ev\ a)\ P' \implies \text{Renaming } P\ f \text{ }_{DT\rightsquigarrow}(ev\ b)\ (\text{Renaming } P'\ f) \rangle$
<proof>

lemma $\langle P \text{ }_{DT\rightsquigarrow\tau}\ P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow\tau}\ P' \llbracket a := b \rrbracket \rangle$

<proof>

lemma $\langle P \text{ }_{DT\rightsquigarrow}\checkmark\ P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow}\checkmark\ STOP \rangle$

<proof>

lemma *ev-trans-RenamingF:*

$\langle P \text{ }_{DT\rightsquigarrow}(ev\ a)\ P' \implies P \llbracket a := b \rrbracket \text{ }_{DT\rightsquigarrow}(ev\ b)\ P' \llbracket a := b \rrbracket \rangle$
<proof>

P [[S]] 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:* $\langle Q \text{ }_{DT\rightsquigarrow\tau}\ Q' \implies P \llbracket S \rrbracket\ Q \text{ }_{DT\rightsquigarrow\tau}\ P \llbracket S \rrbracket\ Q' \rangle$
<proof>

lemma *ev-trans-SyncL:*

$\langle e \notin S \implies P \text{ }_{DT\rightsquigarrow}(ev\ e)\ P' \implies P \llbracket S \rrbracket\ Q \text{ }_{DT\rightsquigarrow}(ev\ e)\ P' \llbracket S \rrbracket\ Q \rangle$

and *ev-trans-SyncR:*

$\langle e \notin S \implies Q \text{ }_{DT\rightsquigarrow}(ev\ e)\ Q' \implies P \llbracket S \rrbracket\ Q \text{ }_{DT\rightsquigarrow}(ev\ e)\ P \llbracket S \rrbracket\ Q' \rangle$

<proof>

lemma *ev-trans-SyncLR:*

$$\langle \llbracket e \in S; P \text{ }_{DT\rightsquigarrow}(ev\ e) P'; Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q \rrbracket \Longrightarrow \\ P \llbracket S \rrbracket Q \text{ }_{DT\rightsquigarrow}(ev\ e) P' \llbracket S \rrbracket Q' \rangle \\ \langle \text{proof} \rangle$$

From here we slightly defer from Roscoe's laws for *Sync*: we obtain the following rules for *SKIP* instead of *STOP*.

$$\text{lemma } \textit{tick-trans-SyncL}: \langle P \text{ }_{DT\rightsquigarrow}\checkmark P' \Longrightarrow P \llbracket S \rrbracket Q \text{ }_{DT\rightsquigarrow\tau} \text{SKIP} \llbracket S \rrbracket Q \rangle \\ \text{and } \textit{tick-trans-SyncR}: \langle Q \text{ }_{DT\rightsquigarrow}\checkmark Q' \Longrightarrow P \llbracket S \rrbracket Q \text{ }_{DT\rightsquigarrow\tau} P \llbracket S \rrbracket \text{SKIP} \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \textit{tick-trans-SKIP-Sync-SKIP}: \langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \text{ }_{DT\rightsquigarrow}\checkmark \text{STOP} \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \textit{\tau-trans-SKIP-Sync-SKIP}: \langle \text{SKIP} \llbracket S \rrbracket \text{SKIP} \text{ }_{DT\rightsquigarrow\tau} \text{SKIP} \rangle \\ \langle \text{proof} \rangle$$

P \triangleright *Q* laws

$$\text{lemma } \textit{Sliding-\tau-trans-left}: \langle P \text{ }_{DT\rightsquigarrow\tau} P' \Longrightarrow P \triangleright Q \text{ }_{DT\rightsquigarrow\tau} P' \triangleright Q \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \langle P \text{ }_{DT\rightsquigarrow} e P' \Longrightarrow P \triangleright Q \text{ }_{DT\rightsquigarrow} e P' \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \langle P \triangleright Q \text{ }_{DT\rightsquigarrow\tau} Q \rangle \\ \langle \text{proof} \rangle$$

P \triangle *Q* laws

$$\text{lemma } \textit{Interrupt-\tau-trans-left}: \langle P \text{ }_{DT\rightsquigarrow\tau} P' \Longrightarrow P \triangle Q \text{ }_{DT\rightsquigarrow\tau} P' \triangle Q \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \textit{Interrupt-\tau-trans-right}: \langle Q \text{ }_{DT\rightsquigarrow\tau} Q' \Longrightarrow P \triangle Q \text{ }_{DT\rightsquigarrow\tau} P \triangle Q' \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \textit{Interrupt-ev-trans-left}: \\ \langle P \text{ }_{DT\rightsquigarrow}(ev\ e) P' \Longrightarrow P \triangle Q \text{ }_{DT\rightsquigarrow}(ev\ e) P' \triangle Q \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \textit{Interrupt-ev-trans-right}: \langle Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q' \Longrightarrow P \triangle Q \text{ }_{DT\rightsquigarrow}(ev\ e) Q' \rangle \\ \langle \text{proof} \rangle$$

Throw P A Q laws

$$\text{lemma } \textit{Throw-\tau-trans-left}: \\ \langle P \text{ }_{DT\rightsquigarrow\tau} P' \Longrightarrow P \Theta a \in A. Q a \text{ }_{DT\rightsquigarrow\tau} P' \Theta a \in A. Q a \rangle \\ \langle \text{proof} \rangle$$

$$\text{lemma } \textit{Throw-\tau-trans-right}: \\ \langle \forall a \in A. Q a \text{ }_{DT\rightsquigarrow\tau} Q' a \Longrightarrow P \Theta a \in A. Q a \text{ }_{DT\rightsquigarrow\tau} P \Theta a \in A. Q' a \rangle \\ \langle \text{proof} \rangle$$

lemma *Throw-event-trans-left:*

$$\langle P \xrightarrow{DT} e P' \implies e \notin \text{ev } A \implies P \ominus a \in A. Q a \xrightarrow{DT} e (P' \ominus a \in A. Q a) \rangle$$

<proof>

lemma *Throw-ev-trans-right:*

$$\langle P \xrightarrow{DT} (ev e) P' \implies e \in A \implies P \ominus a \in A. Q a \xrightarrow{DT} (ev e) (Q e) \rangle$$

<proof>

lemma *<front-tickFree s \implies \perp \xrightarrow{DT}^* s P>*

<proof>

9.2 Reality Checks

lemma *<STOP \xrightarrow{DT}^* s P \longleftrightarrow s = [] \wedge P = STOP>*

<proof>

lemma *T-iff-exists-trans : <s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \xrightarrow{DT}^* s P')>*

<proof>

lemma *D-iff-trace-trans-BOT: <s \in \mathcal{D} P \longleftrightarrow P \xrightarrow{DT}^* s \perp>*

<proof>

9.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace:*

$$\langle P \xrightarrow{DT}^* s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$$

<proof>

lemma *trace-trans-imp-ready-set:*

$$\langle P \xrightarrow{DT}^* (s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$$

<proof>

lemma *AfterTrace-\tau-trans-if-\tau-trans-imp-leT :*

$$\langle (P \xrightarrow{DT}^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \xrightarrow{DT} \tau Q \rangle$$

<proof>

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

$$\overline{SKIP \text{ }_{DT \rightsquigarrow} \checkmark \text{ } STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ } a \text{ }_{DT \rightsquigarrow} ev \text{ } ?e \text{ } ?P \text{ } ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ } a \text{ }_{DT \rightsquigarrow} ev \text{ } ?e \text{ } ?P \text{ } ?e}$$

$$\overline{?e \rightarrow ?P \text{ }_{DT \rightsquigarrow} ev \text{ } ?e \text{ } ?P}$$

(\square) rules

$$\frac{\overline{?P \square ?Q \text{ }_{DT \rightsquigarrow \tau} ?P} \quad \overline{?P \square ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q}}{?e \in ?A} \quad \frac{}{\square a \in ?A. \text{ } ?P \text{ } a \text{ }_{DT \rightsquigarrow \tau} ?P \text{ } ?e}$$

$\mu x. f x$ rule

$$\frac{cont \text{ } ?f \quad ?P = (\mu x. \text{ } ?f \text{ } x)}{?P \text{ }_{DT \rightsquigarrow \tau} ?f \text{ } ?P}$$

(\square) rules (more powerful than in OpSemFD)

$$?P \square ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \quad ?P \square ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q$$

(;) rules

$$\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P ; ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' ; ?Q} \quad \frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P ; ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' ; ?Q}}{\frac{\exists P'. ?P \text{ }_{DT \rightsquigarrow \checkmark} P'}{?P ; ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q}}$$

Hiding rules

$$\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \setminus ?B \text{ }_{DT \rightsquigarrow \tau} ?P' \setminus ?B} \quad \frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{?P \setminus ?B \text{ }_{DT \rightsquigarrow \checkmark} STOP}}{\frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?e \notin ?B}{?P \setminus ?B \text{ }_{DT \rightsquigarrow ev} ?e ?P' \setminus ?B}} \quad \frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?e \in ?B}{?P \setminus ?B \text{ }_{DT \rightsquigarrow \tau} ?P' \setminus ?B}}$$

Renaming rules

$$\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{DT \rightsquigarrow \tau} \text{Renaming } ?P' \text{ } ?f} \quad \frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{DT \rightsquigarrow \checkmark} STOP}}{\frac{?f \text{ } ?a = ?b \quad ?P \text{ }_{DT \rightsquigarrow ev} ?a ?P'}{\text{Renaming } ?P \text{ } ?f \text{ }_{DT \rightsquigarrow ev} ?b \text{Renaming } ?P' \text{ } ?f}}$$

Sync rules

$$\frac{\frac{\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \text{ } [?S] \text{ } ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \text{ } [?S] \text{ } ?Q'}}{\frac{?e \notin ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \text{ } [?S] \text{ } ?Q} \quad \frac{?e \notin ?S \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P \text{ } [?S] \text{ } ?Q'}}}{\frac{?e \in ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \text{ } [?S] \text{ } ?Q'}} \quad \frac{\frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} SKIP \text{ } [?S] \text{ } ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \checkmark} ?Q'}{?P \text{ } [?S] \text{ } ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \text{ } [?S] \text{ } SKIP}}$$

$$\overline{SKIP \llbracket ?S \rrbracket SKIP \quad DT \rightsquigarrow_{\tau} SKIP}$$

(▷) rules

$$\frac{?P \quad DT \rightsquigarrow_{\tau} \quad ?P'}{?P \triangleright ?Q \quad DT \rightsquigarrow_{\tau} \quad ?P' \triangleright ?Q} \quad \frac{?P \quad DT \rightsquigarrow ?e \quad ?P'}{?P \triangleright ?Q \quad DT \rightsquigarrow ?e \quad ?P'}$$

$$\frac{}{?P \triangleright ?Q \quad DT \rightsquigarrow_{\tau} \quad ?Q}$$

(△) rules

$$\frac{?P \quad DT \rightsquigarrow_{\tau} \quad ?P'}{?P \triangle ?Q \quad DT \rightsquigarrow_{\tau} \quad ?P' \triangle ?Q} \quad \frac{?Q \quad DT \rightsquigarrow_{\tau} \quad ?Q'}{?P \triangle ?Q \quad DT \rightsquigarrow_{\tau} \quad ?P \triangle ?Q'}$$

$$\frac{?P \quad DT \rightsquigarrow ev \quad ?e \quad ?P'}{?P \triangle ?Q \quad DT \rightsquigarrow ev \quad ?e \quad ?P' \triangle ?Q} \quad \frac{?Q \quad DT \rightsquigarrow ev \quad ?e \quad ?Q'}{?P \triangle ?Q \quad DT \rightsquigarrow ev \quad ?e \quad ?Q'}$$

Throw rules

$$\frac{?P \quad DT \rightsquigarrow_{\tau} \quad ?P'}{?P \ominus a \in ?A. \quad ?Q \ a \quad DT \rightsquigarrow_{\tau} \quad ?P' \ominus a \in ?A. \quad ?Q \ a}$$

$$\frac{\forall a \in ?A. \quad ?Q \ a \quad DT \rightsquigarrow_{\tau} \quad ?Q' \ a}{?P \ominus a \in ?A. \quad ?Q \ a \quad DT \rightsquigarrow_{\tau} \quad ?P \ominus a \in ?A. \quad ?Q' \ a}$$

$$\frac{?P \quad DT \rightsquigarrow ?e \quad ?P' \quad ?e \notin ev \ ' \ ?A}{?P \ominus a \in ?A. \quad ?Q \ a \quad DT \rightsquigarrow ?e \quad ?P' \ominus a \in ?A. \quad ?Q \ a}$$

$$\frac{?P \quad DT \rightsquigarrow ev \quad ?e \quad ?P' \quad ?e \in ?A}{?P \ominus a \in ?A. \quad ?Q \ a \quad DT \rightsquigarrow ev \quad ?e \quad ?Q \ ?e}$$

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*: $\langle e \notin \text{ready-set } P \Longrightarrow P \text{ afterExt } e = \text{STOP} \rangle$
<proof>

lemma *ready-set-AfterExt*:
 $\langle \text{ready-set } (P \text{ afterExt } e) = (\text{if } e = \checkmark \text{ then } \{\} \text{ else } \{a. e \# [a] \in \mathcal{T} P\}) \rangle$
<proof>

10.1.2 Projections

lemma *F-AfterExt*:
 $\langle \mathcal{F} (P \text{ afterExt } e) =$
 $(\text{if } e \in \text{ready-set } P \text{ then } \{(tl \ s, X) \mid s \ X. (s, X) \in \mathcal{F} P \wedge s \neq [] \wedge hd \ s = e\}$

$\text{else } \{(s, X). s = []\}$
 $(\text{is } \langle - = ?rhs \rangle)$
 $\langle \text{proof} \rangle$

lemma *D-AfterExt*: $\langle \mathcal{D} (P \text{ afterExt } e) = (\text{ if } e = \checkmark \wedge P = \perp \text{ then } \{\} \text{ else } \{tl\ s \mid s . s \in \mathcal{D} P \wedge s \neq [] \wedge hd\ s = e\}) \rangle$
 $(\text{is } \langle - = ?rhs \rangle)$
 $\langle \text{proof} \rangle$

lemma *T-AfterExt*: $\langle \mathcal{T} (P \text{ afterExt } e) = (\text{ if } e = \checkmark \wedge P = \perp \text{ then } \{\} \text{ else insert } [] \{tl\ s \mid s . s \in \mathcal{T} P \wedge s \neq [] \wedge hd\ s = e\}) \rangle$
 $(\text{is } \langle - = ?rhs \rangle)$
 $\langle \text{proof} \rangle$

10.1.3 Monotony

lemma *mono-AfterExt* : $\langle P \sqsubseteq Q \implies P \text{ afterExt } e \sqsubseteq Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterExt-T* : $\langle P \sqsubseteq_T Q \implies P \text{ afterExt } e \sqsubseteq_T Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterExt-F* :
 $\langle P \sqsubseteq_F Q \implies ev\ e \notin \text{ready-set } P \vee ev\ e \in \text{ready-set } Q \implies$
 $P \text{ afterExt } ev\ e \sqsubseteq_F Q \text{ afterExt } ev\ e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterExt-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterExt } e \sqsubseteq_D Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterExt-FD* :
 $\langle P \sqsubseteq_{FD} Q \implies e \notin \text{ready-set } P \vee e \in \text{ready-set } Q \implies$
 $P \text{ afterExt } e \sqsubseteq_{FD} Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterExt-DT* : $\langle P \sqsubseteq_{DT} Q \implies P \text{ afterExt } e \sqsubseteq_{DT} Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

10.1.4 Behaviour of *AfterExt* with *STOP*, *SKIP* and \perp

lemma *AfterExt-STOP*: $\langle \text{STOP afterExt } e = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-is-STOP-iff*:
 $\langle P \text{ afterExt } e = \text{STOP} \iff P = \perp \wedge e = \checkmark \vee (\forall s. e \# s \in \mathcal{T} P \longrightarrow s = []) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-SKIP*: $\langle \text{SKIP afterExt } e = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-BOT* : $\langle \perp \text{ afterExt } e = (\text{if } e = \checkmark \text{ then STOP else } \perp) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-is-BOT-iff*: $\langle P \text{ afterExt } e = \perp \longleftrightarrow e \neq \checkmark \wedge [e] \in \mathcal{D} P \rangle$
 $\langle \text{proof} \rangle$

10.1.5 Behaviour of *AfterExt* with Operators of HOL-CSP

Here again, we lose determinism.

lemma *AfterExt-Mprefix-is-AfterExt-Mndetprefix*:
 $\langle (\Box a \in A \rightarrow P a) \text{ afterExt } e = (\Box a \in A \rightarrow P a) \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Det-is-AfterExt-Ndet*: $\langle P \sqcap Q \text{ afterExt } e = P \sqcap Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Ndet*:
 $\langle P \sqcap Q \text{ afterExt } e = (\text{case } e \text{ of } ev \ x \Rightarrow \begin{array}{l} \text{if } ev \ x \in \text{ready-set } P \cap \text{ready-set } Q \\ \text{then } (P \text{ afterExt } ev \ x) \sqcap (Q \text{ afterExt } ev \ x) \\ \text{else } \text{if } ev \ x \in \text{ready-set } P \\ \text{then } P \text{ afterExt } ev \ x \\ \text{else } Q \text{ afterExt } ev \ x \\ | \checkmark \Rightarrow \text{STOP} \end{array}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Mprefix*:
 $\langle (\Box a \in A \rightarrow P a) \text{ afterExt } e = \langle \text{case } e \text{ of } ev \ x \Rightarrow \text{if } x \in A \text{ then } P \ x \text{ else STOP } | \checkmark \Rightarrow \text{STOP} \rangle \rangle$
 $\langle \text{proof} \rangle$

corollary *AfterExt-prefix*:
 $\langle (a \rightarrow P) \text{ afterExt } e = \langle \text{case } e \text{ of } ev \ x \Rightarrow \text{if } x = a \text{ then } P \text{ else STOP } | \checkmark \Rightarrow \text{STOP} \rangle \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle \text{Renaming } P \ f = \perp \longleftrightarrow P = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *Renaming-is-STOP-iff*: $\langle \text{Renaming } P \text{ } f = \text{STOP} \iff P = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Renaming*:
 $\langle \text{Renaming } P \text{ } f \text{ afterExt } e =$
 $(\text{case } e \text{ of } \checkmark \Rightarrow \text{STOP}$
 $| \text{ev } a \Rightarrow \text{if } P = \perp \text{ then } \perp \text{ else}$
 $\sqcap a \in \{a. \text{ev } a \in \text{ready-set } P \wedge \text{ev } (f \ a) = e\}. \text{Renaming } (P \text{ afterExt } \text{ev } a) \ f) \rangle$
 $\langle \text{proof} \rangle$

lemma *Seq-is-BOT-iff*: $\langle P ; Q = \perp \iff P = \perp \vee ([\checkmark] \in \mathcal{T} \ P \wedge Q = \perp) \rangle$
 $\langle \text{proof} \rangle$

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

theorem *AfterExt-Sync*:
 $\langle (P \llbracket S \rrbracket Q) \text{ afterExt } e =$
 $(\text{case } e \text{ of } \checkmark \Rightarrow \text{STOP}$
 $| \text{ev } x \Rightarrow \text{if } P = \perp \vee Q = \perp \text{ then } \perp$
 $\text{else if } e \in \text{ready-set } P \wedge e \in \text{ready-set } Q$
 $\text{then if } x \in S$
 $\text{then } P \text{ afterExt } e \llbracket S \rrbracket Q \text{ afterExt } e$
 $\text{else } (P \text{ afterExt } e \llbracket S \rrbracket Q) \sqcap (P \llbracket S \rrbracket Q \text{ afterExt } e)$
 $\text{else if } e \in \text{ready-set } P$
 $\text{then if } x \in S \text{ then } \text{STOP} \text{ else } P \text{ afterExt } e \llbracket S \rrbracket Q$
 $\text{else if } e \in \text{ready-set } Q$
 $\text{then if } x \in S \text{ then } \text{STOP} \text{ else } P \llbracket S \rrbracket Q \text{ afterExt } e$
 $\text{else } \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *Hiding-FD-Hiding-AfterExt-if-ready-inside*:
 $\langle e \in B \implies (P \setminus B) \sqsubseteq_{FD} (P \text{ afterExt } \text{ev } e \setminus B) \rangle$
and *AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin*:
 $\langle e \notin B \implies (P \setminus B) \text{ afterExt } \text{ev } e \sqsubseteq_{FD} (P \text{ afterExt } \text{ev } e \setminus B) \rangle$
if ready: $\langle \text{ev } e \in \text{ready-set } P \rangle$

<proof>

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*:
 $\langle \llbracket ev\ e \in ready\text{-}set\ P; e \in B \rrbracket \implies (P \setminus B) \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
and *AfterExt-Hiding-DT-Hiding-AfterExt-if-ready-notin*:
 $\langle \llbracket ev\ e \in ready\text{-}set\ P; e \notin B \rrbracket \implies (P \setminus B)\ afterExt\ ev\ e \sqsubseteq_{DT} (P\ afterExt\ ev\ e \setminus B) \rangle$
<proof>

10.1.6 Behaviour of *AfterExt* with Operators of HOL-CSPM

lemma *AfterExt-MultiDet-is-AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\Box a \in A.\ P\ a)\ afterExt\ e = (\prod a \in A.\ P\ a)\ afterExt\ e \rangle$
<proof>

lemma *AfterExt-GlobalNdet*:
 $\langle (\Box a \in A.\ P\ a)\ afterExt\ e =$
 (if $e = \checkmark \vee e \notin (\bigcup a \in A.\ ready\text{-}set\ (P\ a))$ then *STOP*
 else $(\Box a \in \{a \in A.\ e \in ready\text{-}set\ (P\ a)\}.\ P\ a)\ afterExt\ e \rangle$
and *AfterExt-MultiNdet*:
 $\langle finite\ A \implies (\prod a \in A.\ P\ a)\ afterExt\ e =$
 (if $e = \checkmark \vee e \notin (\bigcup a \in A.\ ready\text{-}set\ (P\ a))$ then *STOP*
 else $(\prod a \in \{a \in A.\ e \in ready\text{-}set\ (P\ a)\}.\ P\ a)\ afterExt\ e \rangle$
and *AfterExt-MultiDet*:
 $\langle finite\ A \implies (\Box a \in A.\ P\ a)\ afterExt\ e =$
 (if $e = \checkmark \vee e \notin (\bigcup a \in A.\ ready\text{-}set\ (P\ a))$ then *STOP*
 else $(\prod a \in \{a \in A.\ e \in ready\text{-}set\ (P\ a)\}.\ P\ a)\ afterExt\ e \rangle$
<proof>

10.1.7 Behaviour of *AfterExt* with Operators of HOL-CSP_OpSem

lemma *AfterExt-Sliding-is-AfterExt-Ndet*:

$\langle P \triangleright Q \text{ afterExt } e = P \sqcap Q \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemmas *AfterExt-Sliding* = *AfterExt-Ndet*[folded *AfterExt-Sliding-is-AfterExt-Ndet*]

lemma *AfterExt-Throw*:

$\langle (P \Theta a \in A. Q a) \text{ afterExt } e =$
 $(\text{case } e \text{ of } \checkmark \Rightarrow \text{STOP}$
 $\quad | \text{ev } x \Rightarrow \text{if } P = \perp \text{ then } \perp$
 $\quad \quad \text{else if } \text{ev } x \in \text{ready-set } P \text{ then if } x \in A \text{ then } Q x$
 $\quad \quad \quad \text{else } (P \text{ after } x) \Theta a \in A. Q a \text{ else } \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-Interrupt*:

$\langle (P \Delta Q) \text{ afterExt } e =$
 $(\text{case } e \text{ of } \checkmark \Rightarrow \text{STOP}$
 $\quad | \text{ev } x \Rightarrow \text{if } P = \perp \vee Q = \perp \text{ then } \perp$
 $\quad \quad \text{else if } \text{ev } x \in \text{ready-set } P \wedge \text{ev } x \in \text{ready-set } Q$
 $\quad \quad \quad \text{then } (Q \text{ after } x) \sqcap (P \text{ after } x \Delta Q)$
 $\quad \quad \quad \text{else if } \text{ev } x \in \text{ready-set } P \wedge \text{ev } x \notin \text{ready-set } Q$
 $\quad \quad \quad \text{then } P \text{ after } x \Delta Q$
 $\quad \quad \quad \text{else if } \text{ev } x \notin \text{ready-set } P \wedge \text{ev } x \in \text{ready-set } Q$
 $\quad \quad \quad \text{then } Q \text{ after } x$
 $\quad \quad \quad \text{else } \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

10.1.8 Behaviour of *AfterExt* with Reference Processes

lemma *AfterExt-DF*:

$\langle \text{DF } A \text{ afterExt } e =$
 $(\text{case } e \text{ of } \text{ev } x \Rightarrow \text{if } x \in A \text{ then } \text{DF } A \text{ else } \text{STOP} \mid \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-DF_{SKIP}*:

$\langle \text{DF}_{\text{SKIP}} A \text{ afterExt } e =$
 $(\text{case } e \text{ of } \text{ev } x \Rightarrow \text{if } x \in A \text{ then } \text{DF}_{\text{SKIP}} A \text{ else } \text{STOP} \mid \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-RUN*:

$\langle \text{RUN } A \text{ afterExt } e =$
 $(\text{case } e \text{ of } \text{ev } x \Rightarrow \text{if } x \in A \text{ then } \text{RUN } A \text{ else } \text{STOP} \mid \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-CHAOS*:

$\langle \text{CHAOS } A \text{ afterExt } e =$
 $(\text{case } e \text{ of } \text{ev } x \Rightarrow \text{if } x \in A \text{ then } \text{CHAOS } A \text{ else } \text{STOP} \mid \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterExt-CHAOS_{SKIP}*:
 $\langle \text{CHAOS}_{\text{SKIP}} A \text{ afterExt } e =$
 $(\text{case } e \text{ of } \text{ev } x \Rightarrow \text{if } x \in A \text{ then } \text{CHAOS}_{\text{SKIP}} A \text{ else } \text{STOP} \mid \checkmark \Rightarrow \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *DF-FD-AfterExt*:
 $\langle \text{DF } A \sqsubseteq_{\text{FD}} P \Longrightarrow e \in \text{ready-set } P \Longrightarrow \text{DF } A \sqsubseteq_{\text{FD}} P \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *DF_{SKIP}-FD-AfterExt*:
 $\langle \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \Longrightarrow \text{ev } e \in \text{ready-set } P \Longrightarrow \text{DF}_{\text{SKIP}} A \sqsubseteq_{\text{FD}} P \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

lemma *deadlock-free-AfterExt*:
 $\langle \text{deadlock-free } P \Longrightarrow \text{deadlock-free } (P \text{ afterExt } e) \longleftrightarrow$
 $(\text{if } e \in \text{ready-set } P \wedge e \neq \checkmark \text{ then True else False}) \rangle$
 $\langle \text{proof} \rangle$

lemma *deadlock-free_{SKIP}-AfterExt*:
 $\langle \text{deadlock-free}_{\text{SKIP}} P \Longrightarrow \text{deadlock-free}_{\text{SKIP}} (P \text{ afterExt } e) \longleftrightarrow$
 $(\text{if } e \in \text{ready-set } P \wedge e \neq \checkmark \text{ then True else False}) \rangle$
 $\langle \text{proof} \rangle$

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 $'\alpha$ process and e of type $'\alpha$ event. Since traces of an $'\alpha$ process are $'\alpha$ 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 $\langle P \text{ afterTrace } [] = P \rangle$
 $\mid \langle P \text{ afterTrace } (e \# s) = P \text{ afterExt } e \text{ afterTrace } s \rangle$

We can also induct backward.

lemma *AfterTrace-append*: $\langle P \text{ afterTrace } (s @ t) = P \text{ afterTrace } s \text{ afterTrace } t \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterTrace-snoc* : $\langle P \text{ afterTrace } (s @ [e]) = P \text{ afterTrace } s \text{ afterExt } e \rangle$
 $\langle \text{proof} \rangle$

We have some immediate properties.

lemma *AfterTrace-STOP* : $\langle \text{STOP afterTrace } s = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterTrace-SKIP* : $\langle \text{SKIP afterTrace } s = (\text{if } s = [] \text{ then SKIP else STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *AfterTrace-BOT* : $\langle \perp \text{ afterTrace } s = (\text{if tickFree } s \text{ then } \perp \text{ else STOP}) \rangle$
 $\langle \text{proof} \rangle$

10.2.2 Projections

lemma *F-AfterTrace* : $\langle (s @ t, X) \in \mathcal{F} P \implies (t, X) \in \mathcal{F} (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

lemma *D-AfterTrace* :
 $\langle \text{tickFree } (s @ t) \implies s @ t \in \mathcal{D} P \implies t \in \mathcal{D} (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

lemma *T-AfterTrace* : $\langle s @ t \in \mathcal{T} P \implies t \in \mathcal{T} (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

corollary *ready-set-AfterTrace* :
 $\langle s @ e \# t \in \mathcal{T} P \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

corollary *F-imp-refusal-AfterTrace*:
 $\langle (s, X) \in \mathcal{F} P \implies ([], X) \in \mathcal{F} (P \text{ afterTrace } s) \rangle$
 $\langle \text{proof} \rangle$

corollary *D-imp-AfterTrace-is-BOT*:
 $\langle \text{tickFree } s \implies s \in \mathcal{D} P \implies P \text{ afterTrace } s = \perp \rangle$
 $\langle \text{proof} \rangle$

10.2.3 Monotony

lemma *mono-AfterTrace* : $\langle P \sqsubseteq Q \implies P \text{ afterTrace } s \sqsubseteq Q \text{ afterTrace } s \rangle$
 $\langle \text{proof} \rangle$

lemma *mono-AfterTrace-T* :
 $\langle P \sqsubseteq_T Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_T Q \text{ afterTrace } s \rangle$

<proof>

lemma *mono-AfterTrace-F* :

$\langle P \sqsubseteq_F Q \implies s \in \mathcal{T} Q \implies \text{tickFree } s \implies P \text{ afterTrace } s \sqsubseteq_F Q \text{ afterTrace } s \rangle$
<proof>

lemma *mono-AfterTrace-D* : $\langle P \sqsubseteq_D Q \implies P \text{ afterTrace } s \sqsubseteq_D Q \text{ afterTrace } s \rangle$

<proof>

lemma *mono-AfterTrace-FD* :

$\langle P \sqsubseteq_{FD} Q \implies s \in \mathcal{T} Q \implies P \text{ afterTrace } s \sqsubseteq_{FD} Q \text{ afterTrace } s \rangle$
<proof>

lemma *mono-AfterTrace-DT* :

$\langle P \sqsubseteq_{DT} Q \implies P \text{ afterTrace } s \sqsubseteq_{DT} Q \text{ afterTrace } s \rangle$
<proof>

10.2.4 Another Definition of *events-of*

inductive *reachable-event* :: $\langle 'a \text{ process} \implies 'a \implies \text{bool} \rangle$

where $\langle \text{ev } e \in \text{ready-set } P \implies \text{reachable-event } P \ e \rangle$

| $\langle \text{reachable-event } (P \text{ after } f) \ e \implies \text{reachable-event } P \ e \rangle$

lemma *events-of-iff-reachable-event*: $\langle e \in \text{events-of } P \longleftrightarrow \text{reachable-event } P \ e \rangle$

<proof>

lemma *reachable-event-BOT*: $\langle \text{reachable-event } \perp \ e \rangle$

<proof>

lemma *not-reachable-event-STOP*: $\langle \neg \text{reachable-event } \text{STOP} \ e \rangle$

<proof>

lemma *reachable-tick-SKIP*: $\langle \neg \text{reachable-event } \text{SKIP} \ \checkmark \rangle$

<proof>

lemma *reachable-event-iff-in-ready-set-AfterTrace*:

$\langle \text{reachable-event } P \ e \longleftrightarrow e \in \{e \mid e \text{ s. } \text{ev } e \in \text{ready-set } (P \text{ afterTrace } s)\} \rangle$

<proof>

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 $\langle \text{deadlock-free } P \longleftrightarrow ?\text{rhs} \rangle$)

<proof>

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)

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' ( $\dashv$ -/  
 $\rightsquigarrow_{\dashv}$ -/  $\rightarrow$  [50, 3, 51] 50)  
  where  $\langle P \rightsquigarrow_{\dashv} e Q \equiv e \in \text{ready-set } P \wedge P \text{ afterExt } e \rightsquigarrow_\tau Q \rangle$ 
```

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: $\langle \perp \rightsquigarrow (ev\ e)\ P \rangle$
 $\langle proof \rangle$

As immediate consequences of the axioms, we prove that event transitions absorb τ transitions on right and on left.

lemma event-trans- τ -trans: $\langle P \rightsquigarrow e\ P' \implies P' \rightsquigarrow_{\tau} P'' \implies P \rightsquigarrow e\ P'' \rangle$
 $\langle proof \rangle$

lemma τ -trans-event-trans: $\langle P \rightsquigarrow_{\tau} P' \implies P' \rightsquigarrow e\ P'' \implies P \rightsquigarrow e\ P'' \rangle$
 $\langle proof \rangle$

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 : $\langle P \rightsquigarrow_{\tau} P' \implies P \rightsquigarrow^* \sqcap P' \rangle$
| 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: $\langle P \rightsquigarrow^* s\ P' \implies P' \rightsquigarrow_{\tau} P'' \implies P \rightsquigarrow^* s\ P'' \rangle$
 $\langle proof \rangle$

lemma τ -trans-trace-trans: $\langle P \rightsquigarrow_{\tau} P' \implies P' \rightsquigarrow^* s\ P'' \implies P \rightsquigarrow^* s\ P'' \rangle$
 $\langle proof \rangle$

lemma tickFree-if-trans-trans-not-STOP :
 $\langle P \rightsquigarrow^* s\ Q \implies Q \neq STOP \implies tickFree\ s \rangle$
 $\langle proof \rangle$

lemma BOT-trace-trans-tickFree-anything : $\langle tickFree\ s \implies \perp \rightsquigarrow^* s\ P \rangle$
 $\langle proof \rangle$

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 $\langle \forall P\ Q. P \rightsquigarrow_{\tau} Q \implies P \sqsubseteq_F Q \rangle$
 $\langle proof \rangle$

lemma *trace-trans-imp-T-if- τ -trans-imp-leT*: $\langle P \rightsquigarrow^* s Q \implies s \in \mathcal{T} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-trans-BOT-imp-D-if- τ -trans-imp-leD*: $\langle P \rightsquigarrow^* s \perp \implies s \in \mathcal{D} P \rangle$
if $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_D Q \rangle$
 $\langle \text{proof} \rangle$

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* :

- $\langle P \rightsquigarrow^* [] Q \longleftrightarrow P \rightsquigarrow_\tau Q \rangle$
- $\langle P \rightsquigarrow^* [\checkmark] Q \longleftrightarrow P \rightsquigarrow_{\checkmark} Q \rangle$
- $\langle P \rightsquigarrow^* (ev\ e) \# s Q' \longleftrightarrow (\exists Q. P \rightsquigarrow (ev\ e) Q \wedge Q \rightsquigarrow^* s Q') \rangle$
- $\langle tickFree\ s \implies (P \rightsquigarrow^* s @ [f] Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow_f Q') \rangle$
- $\langle front-tickFree (s @ t) \implies (P \rightsquigarrow^* s @ t Q') \longleftrightarrow (\exists Q. P \rightsquigarrow^* s Q \wedge Q \rightsquigarrow^* t Q') \rangle$

 $\langle \text{proof} \rangle$

11.4 Finally: $P \rightsquigarrow^* s Q$ is P afterTrace $s \rightsquigarrow_\tau Q$

theorem *T-imp-trace-trans-iff-AfterTrace- τ -trans* :

- $\langle s \in \mathcal{T} P \implies (P \rightsquigarrow^* s Q) \longleftrightarrow P \text{ afterTrace } s \rightsquigarrow_\tau Q \rangle$

 $\langle \text{proof} \rangle$

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*: $\langle (s, X) \in \mathcal{F} P \implies \exists Q. (P \rightsquigarrow^* s Q) \wedge X \in \mathcal{R} Q \rangle$
 $\langle \text{proof} \rangle$

lemma *T-imp-exists-trace-trans*: $\langle s \in \mathcal{T} P \implies \exists Q. P \rightsquigarrow^* s Q \rangle$
 $\langle \text{proof} \rangle$

lemma *D-imp-trace-trans-BOT*: $\langle tickFree\ s \implies s \in \mathcal{D} P \implies P \rightsquigarrow^* s \perp \rangle$
 $\langle \text{proof} \rangle$

When we have more information on $P \rightsquigarrow_\tau Q$, we obtain:

lemma *τ -trans-imp-leT-imp-STOP-trace-trans-iff*:

- $\langle \forall P Q. P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \implies STOP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = STOP \rangle$

 $\langle \text{proof} \rangle$

lemma τ -trans-imp-leF-imp-SKIP-trace-trans-iff:

$$\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_F Q \implies \\ SKIP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle \\ \langle \text{proof} \rangle$$

lemma τ -trans-imp-leT-imp-trace-trans-ready-set-subset-ready-set-AfterTrace:

$$\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \implies P \rightsquigarrow^* s Q \implies \\ \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle \\ \langle \text{proof} \rangle$$

lemma τ -trans-imp-leT-imp-trace-trans-imp-ready-set:

$$\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \implies P \rightsquigarrow^*(s @ e \# t) Q \implies \\ e \in \text{ready-set } (P \text{ afterTrace } s) \rangle \\ \langle \text{proof} \rangle$$

lemma trace-trans-iff-T-and-AfterTrace- τ -trans-if- τ -trans-imp-leT:

$$\langle \forall P Q. P \rightsquigarrow_{\tau} Q \longrightarrow P \sqsubseteq_T Q \implies \\ (P \rightsquigarrow^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \rightsquigarrow_{\tau} Q \rangle \\ \langle \text{proof} \rangle$$

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: $\langle SKIP \rightsquigarrow \checkmark STOP \rangle$

$\langle \text{proof} \rangle$

lemma tick-trans-imp-BOT-L-or-STOP-R: $\langle P \rightsquigarrow \checkmark Q \implies P = \perp \vee Q = STOP \rangle$

$\langle \text{proof} \rangle$

lemma STOP-trace-trans-iff : $\langle STOP \rightsquigarrow^* s P \longleftrightarrow s = [] \wedge P = STOP \rangle$

$\langle \text{proof} \rangle$

lemma ready-tick-imp- τ -trans-SKIP: $\langle P \rightsquigarrow_{\tau} SKIP \rangle$ **if** $\langle \checkmark \in \text{ready-set } P \rangle$

$\langle \text{proof} \rangle$

lemma exists-tick-trans-is-ready-tick: $\langle (\exists P'. P \rightsquigarrow \checkmark P') \longleftrightarrow \checkmark \in \text{ready-set } P \rangle$

$\langle \text{proof} \rangle$

lemma tick-trans-iff : $\langle P \rightsquigarrow \checkmark P' \longleftrightarrow P \rightsquigarrow_{\tau} SKIP \wedge P' = STOP \rangle$

<proof>

lemma *SKIP-cant-ev-trans*: $\langle \neg \text{SKIP} \rightsquigarrow (ev\ e)\ \text{STOP} \rangle$
<proof>

lemma *STOP-cant-event-trans*: $\langle \neg \text{STOP} \rightsquigarrow_e P \rangle$
<proof>

lemma *ev-trans-Mprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P\ a \rightsquigarrow (ev\ e)\ (P\ e) \rangle$
<proof>

lemma *ev-trans-Mndetprefix*: $\langle e \in A \implies \Box a \in A \rightarrow P\ a \rightsquigarrow (ev\ e)\ (P\ e) \rangle$
<proof>

lemma *ev-trans-prefix*: $\langle e \rightarrow P \rightsquigarrow (ev\ e)\ P \rangle$
<proof>

lemma τ -*trans-MultiNdet*: $\langle \text{finite } A \implies x \in A \implies (\Box a \in A. P\ a) \rightsquigarrow_\tau P\ x \rangle$
<proof>

lemma τ -*trans-GlobalNdet*: $\langle (\Box a \in A. P\ a) \rightsquigarrow_\tau P\ e \rangle$ **if** $\langle e \in A \rangle$
<proof>

lemma *fix-point- τ -trans*: $\langle \text{cont } f \implies P = (\mu X. f\ X) \implies P \rightsquigarrow_\tau f\ P \rangle$
<proof>

lemma *event-trans-DetL*: $\langle P \rightsquigarrow_e P' \implies P \Box Q \rightsquigarrow_e P' \rangle$
<proof>

lemma *event-trans-DetR*: $\langle Q \rightsquigarrow_e Q' \implies P \Box Q \rightsquigarrow_e Q' \rangle$
<proof>

lemma *event-trans-MultiDet*:
 $\langle \text{finite } A \implies a \in A \implies P\ a \rightsquigarrow_e Q \implies (\Box a \in A. P\ a) \rightsquigarrow_e Q \rangle$
<proof>

lemma *Sliding-event-transL*: $\langle P \rightsquigarrow_e P' \implies P \triangleright Q \rightsquigarrow_e P' \rangle$
<proof>

lemma *Sliding- τ -transR*: $\langle P \triangleright Q \rightsquigarrow_{\tau} Q \rangle$
<proof>

lemma $\langle \exists P P' Q. P \rightsquigarrow_{\checkmark} P' \wedge \neg P ; Q \rightsquigarrow_{\checkmark} P' ; Q \rangle$
<proof>

lemma *ev-trans-SeqR*:
 $\langle \checkmark \in \text{ready-set } P \implies Q \rightsquigarrow(\text{ev } e) Q' \implies P ; Q \rightsquigarrow(\text{ev } e) Q' \rangle$
<proof>

lemma $\langle \text{SKIP } \llbracket S \rrbracket \text{ SKIP } \rightsquigarrow_{\checkmark} \text{STOP} \rangle$
<proof>

lemma $\langle \text{SKIP } \llbracket S \rrbracket \text{ SKIP } \rightsquigarrow_{\tau} \text{SKIP} \rangle$
<proof>

lemma *tick-trans-Hiding*: $\langle P \setminus B \rightsquigarrow_{\checkmark} \text{STOP} \rangle$ **if** $\langle P \rightsquigarrow_{\checkmark} P' \rangle$
<proof>

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>

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  $\tau$ -trans :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool' (infixl  $\langle_{FD \rightsquigarrow \tau} \rangle$  50)
  where  $\langle P \rangle_{FD \rightsquigarrow \tau} Q \equiv P \sqsubseteq_{FD} Q$ 
```

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGenericBis*.

```
interpretation OpSemGeneric  $\langle_{(FD \rightsquigarrow \tau)} \rangle$ 
   $\langle proof \rangle$ 
```

```
notation event-trans ( $\langle - \rangle_{FD \rightsquigarrow -} - \rangle$  [50, 3, 51] 50)
```

```
notation trace-trans ( $\langle - \rangle_{FD \rightsquigarrow^* -} - \rangle$  [50, 3, 51] 50)
```

```
lemma  $\langle P \rangle_{FD \rightsquigarrow} e P' \Longrightarrow P' \rangle_{FD \rightsquigarrow \tau} P'' \Longrightarrow P \rangle_{FD \rightsquigarrow} e P'' \rangle$ 
   $\langle P \rangle_{FD \rightsquigarrow \tau} P' \Longrightarrow P' \rangle_{FD \rightsquigarrow} e P'' \Longrightarrow P \rangle_{FD \rightsquigarrow} e P'' \rangle$ 
   $\langle proof \rangle$ 
```

12.1 Operational Semantics Laws

SKIP law

```
lemma  $\langle SKIP \rangle_{FD \rightsquigarrow} \checkmark STOP \rangle$ 
   $\langle proof \rangle$ 
```

$e \rightarrow P$ laws

```
lemma  $\langle e \in A \Longrightarrow \square a \in A \rightarrow P a \rangle_{FD \rightsquigarrow} (ev e) (P e) \rangle$ 
   $\langle proof \rangle$ 
```

lemma $\langle e \in A \implies \prod a \in A \rightarrow P a \text{ }_{FD \rightsquigarrow} (ev\ e) (P\ e) \rangle$
 $\langle proof \rangle$

lemma $\langle e \rightarrow P \text{ }_{FD \rightsquigarrow} (ev\ e) P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle a \in A \implies (\prod a \in A. P a) \text{ }_{FD \rightsquigarrow \tau} P a \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies (\prod a \in A. P a) \text{ }_{FD \rightsquigarrow \tau} P a \rangle$
 $\langle proof \rangle$

$\mu\ x. f\ x$ law

lemma $\langle cont\ f \implies P = (\mu\ X. f\ X) \implies P \text{ }_{FD \rightsquigarrow \tau} f\ P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \text{ }_{FD \rightsquigarrow \tau} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \text{ }_{FD \rightsquigarrow \tau} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow \tau} P \sqcap Q' \rangle$
 $\langle proof \rangle$

lemma τ -trans-MultiDet:
 $\langle finite\ A \implies \forall a \in A. P a \text{ }_{FD \rightsquigarrow \tau} P' a \implies$
 $(\prod a \in A. P a) \text{ }_{FD \rightsquigarrow \tau} (\prod a \in A. P' a) \rangle$
 $\langle proof \rangle$

lemma $\langle P \text{ }_{FD \rightsquigarrow e} P' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow e} P' \rangle$
and $\langle Q \text{ }_{FD \rightsquigarrow e} Q' \implies P \sqcap Q \text{ }_{FD \rightsquigarrow e} Q' \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies P a \text{ }_{FD \rightsquigarrow e} Q \implies (\prod a \in A. P a) \text{ }_{FD \rightsquigarrow e} Q \rangle$
 $\langle proof \rangle$

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{FD \rightsquigarrow \tau} P' \implies P ; Q \text{ }_{FD \rightsquigarrow \tau} P' ; Q \rangle$
 $\langle proof \rangle$

lemma ev -trans-SeqL: $\langle P \text{ }_{FD \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{FD \rightsquigarrow} (ev\ e) P' ; Q \rangle$
 $\langle proof \rangle$

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ FD}\rightsquigarrow\checkmark P' \implies P ; Q \text{ FD}\rightsquigarrow\tau Q \rangle$
 ⟨proof⟩

lemma $\checkmark \in \text{ready-set } P \implies Q \text{ FD}\rightsquigarrow(\text{ev } e) Q' \implies P ; Q \text{ FD}\rightsquigarrow(\text{ev } e) Q'$
 ⟨proof⟩

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \text{ FD}\rightsquigarrow\tau P' \implies P \setminus B \text{ FD}\rightsquigarrow\tau P' \setminus B \rangle$
 ⟨proof⟩

lemma *ev-trans-Hiding-notin*:
 $\langle P \text{ FD}\rightsquigarrow(\text{ev } e) P' \implies e \notin B \implies P \setminus B \text{ FD}\rightsquigarrow(\text{ev } e) P' \setminus B \rangle$
 ⟨proof⟩

lemma $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies P \setminus B \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$
 ⟨proof⟩

lemma *ev-trans-Hiding-inside*:
 $\langle P \text{ FD}\rightsquigarrow(\text{ev } e) P' \implies e \in B \implies P \setminus B \text{ FD}\rightsquigarrow\tau P' \setminus B \rangle$
 ⟨proof⟩

Renaming P f laws

lemma τ -trans-Renaming: $\langle P \text{ FD}\rightsquigarrow\tau P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow\tau \text{Renaming } P' f \rangle$
 ⟨proof⟩

lemma *tick-trans-Renaming*: $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$
 ⟨proof⟩

lemma *ev-trans-Renaming*:
 $\langle f a = b \implies P \text{ FD}\rightsquigarrow(\text{ev } a) P' \implies \text{Renaming } P f \text{ FD}\rightsquigarrow(\text{ev } b) (\text{Renaming } P' f) \rangle$
 ⟨proof⟩

lemma $\langle P \text{ FD}\rightsquigarrow\tau P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow\tau P' \llbracket a := b \rrbracket \rangle$
 ⟨proof⟩

lemma $\langle P \text{ FD}\rightsquigarrow\checkmark P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow\checkmark \text{STOP} \rangle$
 ⟨proof⟩

lemma *ev-trans-RenamingF*:
 $\langle P \text{ FD}\rightsquigarrow(\text{ev } a) P' \implies P \llbracket a := b \rrbracket \text{ FD}\rightsquigarrow(\text{ev } b) P' \llbracket a := b \rrbracket \rangle$
 ⟨proof⟩

$P \llbracket S \rrbracket Q$ laws

lemma τ -trans-SyncL: $\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} P' \llbracket S \rrbracket Q \rangle$
and τ -trans-SyncR: $\langle Q \text{ }_{FD\rightsquigarrow\tau} Q' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} P \llbracket S \rrbracket Q' \rangle$
<proof>

lemma *ev-trans-SyncL*:
 $\langle e \notin S \implies P \text{ }_{FD\rightsquigarrow}(ev\ e) P' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow}(ev\ e) P' \llbracket S \rrbracket Q \rangle$
and *ev-trans-SyncR*:
 $\langle e \notin S \implies Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow}(ev\ e) P \llbracket S \rrbracket Q' \rangle$
<proof>

lemma *ev-trans-SyncLR*:
 $\langle \llbracket e \in S; P \text{ }_{FD\rightsquigarrow}(ev\ e) P'; Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \rrbracket \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow}(ev\ e) P' \llbracket S \rrbracket Q' \rangle$
<proof>

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*: $\langle P \text{ }_{FD\rightsquigarrow}\checkmark P' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \text{ }_{FD\rightsquigarrow}\checkmark Q' \implies P \llbracket S \rrbracket Q \text{ }_{FD\rightsquigarrow\tau} P \llbracket S \rrbracket SKIP \rangle$
<proof>

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \text{ }_{FD\rightsquigarrow}\checkmark STOP \rangle$
<proof>

lemma τ -trans-SKIP-Sync-SKIP: $\langle SKIP \llbracket S \rrbracket SKIP \text{ }_{FD\rightsquigarrow\tau} SKIP \rangle$
<proof>

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \triangleright Q \text{ }_{FD\rightsquigarrow\tau} P' \triangleright Q \rangle$
<proof>

lemma $\langle P \text{ }_{FD\rightsquigarrow}e P' \implies P \triangleright Q \text{ }_{FD\rightsquigarrow}e P' \rangle$
<proof>

lemma $\langle P \triangleright Q \text{ }_{FD\rightsquigarrow\tau} Q \rangle$
<proof>

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \triangle Q \text{ }_{FD\rightsquigarrow\tau} P' \triangle Q \rangle$
<proof>

lemma *Interrupt- τ -trans-right*: $\langle Q \text{ }_{FD\rightsquigarrow\tau} Q' \implies P \triangle Q \text{ }_{FD\rightsquigarrow\tau} P \triangle Q' \rangle$
<proof>

lemma *Interrupt-*ev-trans-left**:
 $\langle P \text{ }_{FD\rightsquigarrow}(ev\ e) P' \implies P \triangle Q \text{ }_{FD\rightsquigarrow}(ev\ e) P' \triangle Q \rangle$
<proof>

lemma *Interrupt-*ev-trans-right**: $\langle Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \implies P \triangle Q \text{ }_{FD\rightsquigarrow}(ev\ e) Q' \rangle$

<proof>

Throw P A Q laws

lemma *Throw- τ -trans-left:*

$\langle P \text{ }_{FD\rightsquigarrow\tau} P' \implies P \Theta a \in A. Q a \text{ }_{FD\rightsquigarrow\tau} P' \Theta a \in A. Q a \rangle$
<proof>

lemma *Throw- τ -trans-right:*

$\langle \forall a \in A. Q a \text{ }_{FD\rightsquigarrow\tau} Q' a \implies P \Theta a \in A. Q a \text{ }_{FD\rightsquigarrow\tau} P \Theta a \in A. Q' a \rangle$
<proof>

lemma *Throw-event-trans-left:*

$\langle P \text{ }_{FD\rightsquigarrow e} P' \implies e \notin \text{ev } A \implies P \Theta a \in A. Q a \text{ }_{FD\rightsquigarrow e} (P' \Theta a \in A. Q a) \rangle$
<proof>

lemma *Throw-ev-trans-right:*

$\langle P \text{ }_{FD\rightsquigarrow} (ev \ e) P' \implies e \in A \implies P \Theta a \in A. Q a \text{ }_{FD\rightsquigarrow} (ev \ e) (Q \ e) \rangle$
<proof>

lemma $\langle \text{tickFree } s \implies \perp \text{ }_{FD\rightsquigarrow^*} s P \rangle$

<proof>

12.2 Reality Checks

lemma $\langle \text{STOP } \text{ }_{FD\rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = \text{STOP} \rangle$

<proof>

lemma *SKIP-trace-trans-iff :*

$\langle \text{SKIP } \text{ }_{FD\rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = \text{SKIP} \vee s = [\checkmark] \wedge P = \text{STOP} \rangle$
<proof>

lemma *F-iff-exists-trans :*

$\langle (s, X) \in \mathcal{F} P \longleftrightarrow (\exists P'. (P \text{ }_{FD\rightsquigarrow^*} s P') \wedge X \in \mathcal{R} P') \rangle$
<proof>

lemma *T-iff-exists-trans :* $\langle s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \text{ }_{FD\rightsquigarrow^*} s P') \rangle$

<proof>

lemma *tickFree-imp-D-iff-trace-trans-BOT:*

$\langle \text{tickFree } s \implies s \in \mathcal{D} P \longleftrightarrow P \text{ }_{FD\rightsquigarrow^*} s \perp \rangle$
<proof>

lemma *D-iff-trace-trans-BOT:*

$\langle s \in \mathcal{D} P \longleftrightarrow (\text{if } \text{tickFree } s \text{ then } P \text{ }_{FD\rightsquigarrow^*} s \perp \text{ else } P \text{ }_{FD\rightsquigarrow^*} (\text{butlast } s) \perp) \rangle$

<proof>

12.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace:*

$\langle P \text{ }_{FD} \rightsquigarrow^* s \ Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$
<proof>

lemma *trace-trans-imp-ready-set:*

$\langle P \text{ }_{FD} \rightsquigarrow^* (s \ @ \ e \ # \ t) \ Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$
<proof>

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT :*

$\langle (P \text{ }_{FD} \rightsquigarrow^* s \ Q) \longleftrightarrow s \in \mathcal{T} \ P \wedge P \text{ afterTrace } s \text{ }_{FD} \rightsquigarrow_{\tau} \ Q \rangle$
<proof>

lemma $\langle \text{deadlock-free } P \longleftrightarrow DF \ UNIV \text{ }_{FD} \rightsquigarrow_{\tau} \ P \rangle$

<proof>

lemma $\langle \text{deadlock-free}_{SKIP} \ P \longleftrightarrow DF_{SKIP} \ UNIV \text{ }_{FD} \rightsquigarrow_{\tau} \ P \rangle$

<proof>

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

$$\frac{\frac{?P \text{ }_{FD} \rightsquigarrow ?e \ ?P' \quad ?P' \text{ }_{FD} \rightsquigarrow_{\tau} \ ?P''}{?P \text{ }_{FD} \rightsquigarrow ?e \ ?P''}}{?P \text{ }_{FD} \rightsquigarrow_{\tau} \ ?P' \quad ?P' \text{ }_{FD} \rightsquigarrow ?e \ ?P''}{?P \text{ }_{FD} \rightsquigarrow ?e \ ?P''}$$

SKIP rule

$$\overline{SKIP \text{ }_{FD} \rightsquigarrow \checkmark \ STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \ a \text{ }_{FD \rightsquigarrow ev} ?e \ ?P \ ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \ a \text{ }_{FD \rightsquigarrow ev} ?e \ ?P \ ?e}$$

$$\frac{}{?e \rightarrow ?P \text{ }_{FD \rightsquigarrow ev} ?e \ ?P}$$

(\square) rules

$$\frac{\frac{}{?P \ \square \ ?Q \text{ }_{FD \rightsquigarrow \tau} ?P} \quad \frac{}{?P \ \square \ ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}}{?e \in ?A}}{\square a \in ?A. \ ?P \ a \text{ }_{FD \rightsquigarrow \tau} ?P \ ?e}$$

$\mu x. f x$ rule

$$\frac{cont \ ?f \quad ?P = (\mu x. \ ?f \ x)}{?P \text{ }_{FD \rightsquigarrow \tau} ?f \ ?P}$$

(\square) rules

$$\frac{\frac{?P \text{ }_{FD \rightsquigarrow \tau} ?P'}{?P \ \square \ ?Q \text{ }_{FD \rightsquigarrow \tau} ?P' \ \square \ ?Q} \quad \frac{?Q \text{ }_{FD \rightsquigarrow \tau} ?Q'}{?P \ \square \ ?Q \text{ }_{FD \rightsquigarrow \tau} ?P \ \square \ ?Q'}}{\frac{?P \text{ }_{FD \rightsquigarrow ?e} ?P'}{?P \ \square \ ?Q \text{ }_{FD \rightsquigarrow ?e} ?P'}} \quad \frac{\frac{?Q \text{ }_{FD \rightsquigarrow ?e} ?Q'}{?P \ \square \ ?Q \text{ }_{FD \rightsquigarrow ?e} ?Q'}}$$

(;) 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

$$\frac{\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{?P \setminus ?B \text{ FD}\rightsquigarrow_{\tau} ?P' \setminus ?B}}{?P \text{ FD}\rightsquigarrow_{ev} ?e ?P' \quad ?e \notin ?B}}{?P \setminus ?B \text{ FD}\rightsquigarrow_{ev} ?e ?P' \setminus ?B} \quad \frac{\frac{?P \text{ FD}\rightsquigarrow_{\checkmark} ?P'}{?P \setminus ?B \text{ FD}\rightsquigarrow_{\checkmark} STOP}}{?P \text{ FD}\rightsquigarrow_{ev} ?e ?P' \quad ?e \in ?B}}{?P \setminus ?B \text{ FD}\rightsquigarrow_{\tau} ?P' \setminus ?B}$$

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

$$\frac{\frac{\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} ?P' \llbracket ?S \rrbracket ?Q} \quad \frac{\frac{?Q \text{ FD}\rightsquigarrow_{\tau} ?Q'}{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} ?P \llbracket ?S \rrbracket ?Q'}}{?e \notin ?S \quad ?P \text{ FD}\rightsquigarrow_{ev} ?e ?P'}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?P' \llbracket ?S \rrbracket ?Q}{?e \notin ?S \quad ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?Q'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?P \llbracket ?S \rrbracket ?Q'}{?e \in ?S \quad ?P \text{ FD}\rightsquigarrow_{ev} ?e ?P' \quad ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?Q'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{ev} ?e ?P' \llbracket ?S \rrbracket ?Q'}{?P \text{ FD}\rightsquigarrow_{\checkmark} ?P'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} SKIP \llbracket ?S \rrbracket ?Q}{?Q \text{ FD}\rightsquigarrow_{\checkmark} ?Q'}}}{\frac{?P \llbracket ?S \rrbracket ?Q \text{ FD}\rightsquigarrow_{\tau} ?P \llbracket ?S \rrbracket SKIP}{SKIP \llbracket ?S \rrbracket SKIP \text{ FD}\rightsquigarrow_{\tau} SKIP}}$$

(▷) rules

$$\frac{?P \text{ FD}\rightsquigarrow_{\tau} ?P'}{?P \triangleright ?Q \text{ FD}\rightsquigarrow_{\tau} ?P' \triangleright ?Q} \quad \frac{?P \text{ FD}\rightsquigarrow_{?e} ?P'}{?P \triangleright ?Q \text{ FD}\rightsquigarrow_{?e} ?P'}$$

$$\overline{?P \triangleright ?Q \text{ }_{FD \rightsquigarrow \tau} ?Q}$$

(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 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* $\langle_{(DT\rightsquigarrow\tau)} \rangle$
 $\langle \text{proof} \rangle$

notation *event-trans* $\langle \langle - / \text{ }_{DT\rightsquigarrow} - / \rightarrow [50, 3, 51] 50 \rangle$

notation *trace-trans* $\langle \langle - / \text{ }_{DT\rightsquigarrow^*} - / \rightarrow [50, 3, 51] 50 \rangle$

lemma $\langle P \text{ }_{DT\rightsquigarrow} e P' \Longrightarrow P' \text{ }_{DT\rightsquigarrow\tau} P'' \Longrightarrow P \text{ }_{DT\rightsquigarrow} e P'' \rangle$
 $\langle P \text{ }_{DT\rightsquigarrow\tau} P' \Longrightarrow P' \text{ }_{DT\rightsquigarrow} e P'' \Longrightarrow P \text{ }_{DT\rightsquigarrow} e P'' \rangle$
 $\langle \text{proof} \rangle$

13.1 Operational Semantics Laws

SKIP law

lemma $\langle \text{SKIP} \text{ }_{DT\rightsquigarrow} \checkmark \text{ STOP} \rangle$
 $\langle \text{proof} \rangle$

$e \rightarrow P$ laws

lemma $\langle e \in A \Longrightarrow \square a \in A \rightarrow P a \text{ }_{DT\rightsquigarrow} (ev e) (P e) \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle e \in A \implies \prod a \in A \rightarrow P a \text{ }_{DT \rightsquigarrow} (ev\ e) (P\ e) \rangle$
 $\langle proof \rangle$

lemma $\langle e \rightarrow P \text{ }_{DT \rightsquigarrow} (ev\ e) P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle a \in A \implies (\prod a \in A. P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies (\prod a \in A. P\ a) \text{ }_{DT \rightsquigarrow \tau} P\ a \rangle$
 $\langle proof \rangle$

$\mu\ x. f\ x$ law

lemma $\langle cont\ f \implies P = (\mu\ X. f\ X) \implies P \text{ }_{DT \rightsquigarrow \tau} f\ P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ 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: $\langle Q \text{ }_{DT \rightsquigarrow \tau} Q' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow \tau} P \sqcap Q' \rangle$
 $\langle proof \rangle$

lemma τ -trans-MultiDet:
 $\langle finite\ A \implies \forall a \in A. P\ a \text{ }_{DT \rightsquigarrow \tau} P'\ a \implies$
 $(\prod a \in A. P\ a) \text{ }_{DT \rightsquigarrow \tau} (\prod a \in A. P'\ a) \rangle$
 $\langle proof \rangle$

lemma $\langle P \text{ }_{DT \rightsquigarrow e} P' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow e} P' \rangle$
and $\langle Q \text{ }_{DT \rightsquigarrow e} Q' \implies P \sqcap Q \text{ }_{DT \rightsquigarrow e} Q' \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies P\ a \text{ }_{DT \rightsquigarrow e} Q \implies (\prod a \in A. P\ a) \text{ }_{DT \rightsquigarrow e} Q \rangle$
 $\langle proof \rangle$

$P ; Q$ laws

lemma τ -trans-SeqL: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} P' ; Q \rangle$
 $\langle proof \rangle$

lemma ev -trans-SeqL: $\langle P \text{ }_{DT \rightsquigarrow} (ev\ e) P' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev\ e) P' ; Q \rangle$
 $\langle proof \rangle$

lemma τ -trans-SeqR: $\langle \exists P'. P \text{ }_{DT \rightsquigarrow} P' \implies P ; Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$

<proof>

lemma $\langle \checkmark \in \text{ready-set } P \implies Q \text{ }_{DT \rightsquigarrow} (ev \ e) \ Q' \implies P ; Q \text{ }_{DT \rightsquigarrow} (ev \ e) \ Q' \rangle$

<proof>

P \ B laws

lemma τ -trans-Hiding: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \setminus B \text{ }_{DT \rightsquigarrow \tau} P' \setminus B \rangle$
<proof>

lemma *ev-trans-Hiding-notin:*

$\langle P \text{ }_{DT \rightsquigarrow} (ev \ e) \ P' \implies e \notin B \implies P \setminus B \text{ }_{DT \rightsquigarrow} (ev \ e) \ P' \setminus B \rangle$
<proof>

lemma $\langle P \text{ }_{DT \rightsquigarrow \checkmark} P' \implies P \setminus B \text{ }_{DT \rightsquigarrow \checkmark} STOP \rangle$
<proof>

lemma *ev-trans-Hiding-inside:*

$\langle P \text{ }_{DT \rightsquigarrow} (ev \ e) \ P' \implies e \in B \implies P \setminus B \text{ }_{DT \rightsquigarrow \tau} P' \setminus B \rangle$
<proof>

Renaming P f laws

lemma τ -trans-Renaming:

$\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies \text{Renaming } P \ f \text{ }_{DT \rightsquigarrow \tau} \text{Renaming } P' \ f \rangle$
<proof>

lemma *tick-trans-Renaming:* $\langle P \text{ }_{DT \rightsquigarrow \checkmark} P' \implies \text{Renaming } P \ f \text{ }_{DT \rightsquigarrow \checkmark} STOP \rangle$
<proof>

lemma *ev-trans-Renaming:*

$\langle f \ a = b \implies P \text{ }_{DT \rightsquigarrow} (ev \ a) \ P' \implies \text{Renaming } P \ f \text{ }_{DT \rightsquigarrow} (ev \ b) \ (\text{Renaming } P' \ f) \rangle$
<proof>

lemma $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \llbracket a := b \rrbracket \text{ }_{DT \rightsquigarrow \tau} P' \llbracket a := b \rrbracket \rangle$
<proof>

lemma $\langle P \text{ }_{DT \rightsquigarrow \checkmark} P' \implies P \llbracket a := b \rrbracket \text{ }_{DT \rightsquigarrow \checkmark} STOP \rangle$
<proof>

lemma *ev-trans-RenamingF:*

$\langle P \text{ }_{DT \rightsquigarrow} (ev \ a) \ P' \implies P \llbracket a := b \rrbracket \text{ }_{DT \rightsquigarrow} (ev \ b) \ P' \llbracket a := b \rrbracket \rangle$
<proof>

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: $\langle Q \text{ }_{DT \rightsquigarrow \tau} Q' \implies P \llbracket S \rrbracket Q \text{ }_{DT \rightsquigarrow \tau} P \llbracket S \rrbracket Q' \rangle$
 $\langle \text{proof} \rangle$

lemma *ev-trans-SyncL*:

$\langle e \notin S \implies P \text{ }_{DT \rightsquigarrow} (ev \ e) P' \implies P \llbracket S \rrbracket Q \text{ }_{DT \rightsquigarrow} (ev \ e) P' \llbracket S \rrbracket Q \rangle$
and *ev-trans-SyncR*:
 $\langle e \notin S \implies Q \text{ }_{DT \rightsquigarrow} (ev \ e) Q' \implies P \llbracket S \rrbracket Q \text{ }_{DT \rightsquigarrow} (ev \ e) P \llbracket S \rrbracket Q' \rangle$
 $\langle \text{proof} \rangle$

lemma *ev-trans-SyncLR*:

$\langle \llbracket e \in S; P \text{ }_{DT \rightsquigarrow} (ev \ e) P'; Q \text{ }_{DT \rightsquigarrow} (ev \ e) Q' \rrbracket \implies$
 $P \llbracket S \rrbracket Q \text{ }_{DT \rightsquigarrow} (ev \ e) P' \llbracket S \rrbracket Q' \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle P \text{ }_{DT \rightsquigarrow} \checkmark P' \implies P \llbracket S \rrbracket Q \text{ }_{DT \rightsquigarrow \tau} SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \text{ }_{DT \rightsquigarrow} \checkmark Q' \implies P \llbracket S \rrbracket Q \text{ }_{DT \rightsquigarrow \tau} P \llbracket S \rrbracket SKIP \rangle$
 $\langle \text{proof} \rangle$

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \text{ }_{DT \rightsquigarrow} \checkmark STOP \rangle$
 $\langle \text{proof} \rangle$

lemma τ -trans-SKIP-Sync-SKIP: $\langle SKIP \llbracket S \rrbracket SKIP \text{ }_{DT \rightsquigarrow \tau} SKIP \rangle$
 $\langle \text{proof} \rangle$

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \triangleright Q \text{ }_{DT \rightsquigarrow \tau} P' \triangleright Q \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle P \text{ }_{DT \rightsquigarrow} e P' \implies P \triangleright Q \text{ }_{DT \rightsquigarrow} e P' \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle P \triangleright Q \text{ }_{DT \rightsquigarrow \tau} Q \rangle$
 $\langle \text{proof} \rangle$

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \text{ }_{DT \rightsquigarrow \tau} P' \implies P \triangle Q \text{ }_{DT \rightsquigarrow \tau} P' \triangle Q \rangle$
 $\langle \text{proof} \rangle$

lemma *Interrupt- τ -trans-right*: $\langle Q \text{ }_{DT \rightsquigarrow \tau} Q' \implies P \triangle Q \text{ }_{DT \rightsquigarrow \tau} P \triangle Q' \rangle$
 $\langle \text{proof} \rangle$

lemma *Interrupt-ev-trans-left*:

$\langle P \text{ }_{DT \rightsquigarrow} (ev \ e) P' \implies P \triangle Q \text{ }_{DT \rightsquigarrow} (ev \ e) P' \triangle Q \rangle$
 $\langle \text{proof} \rangle$

lemma *Interrupt-ev-trans-right*: $\langle Q \text{ }_{DT \rightsquigarrow} (ev \ e) Q' \implies P \triangle Q \text{ }_{DT \rightsquigarrow} (ev \ e) Q' \rangle$
 $\langle \text{proof} \rangle$

Throw $P A Q$ laws

lemma *Throw- τ -trans-left:*

$$\langle P \text{ }_{DT\rightsquigarrow\tau} P' \implies P \Theta a \in A. Q a \text{ }_{DT\rightsquigarrow\tau} P' \Theta a \in A. Q a \rangle$$

\langle proof \rangle

lemma *Throw- τ -trans-right:*

$$\langle \forall a \in A. Q a \text{ }_{DT\rightsquigarrow\tau} Q' a \implies P \Theta a \in A. Q a \text{ }_{DT\rightsquigarrow\tau} P \Theta a \in A. Q' a \rangle$$

\langle proof \rangle

lemma *Throw-event-trans-left:*

$$\langle P \text{ }_{DT\rightsquigarrow e} P' \implies e \notin \text{ev } A \implies P \Theta a \in A. Q a \text{ }_{DT\rightsquigarrow e} (P' \Theta a \in A. Q a) \rangle$$

\langle proof \rangle

lemma *Throw-ev-trans-right:*

$$\langle P \text{ }_{DT\rightsquigarrow} (\text{ev } e) P' \implies e \in A \implies P \Theta a \in A. Q a \text{ }_{DT\rightsquigarrow} (\text{ev } e) (Q e) \rangle$$

\langle proof \rangle

lemma *\langle tickFree s \implies \perp \text{ }_{DT\rightsquigarrow^*} s P \rangle*

\langle proof \rangle

13.2 Reality Checks

lemma *\langle STOP \text{ }_{DT\rightsquigarrow^*} s P \longleftrightarrow s = [] \wedge P = STOP \rangle*

\langle proof \rangle

lemma *T-iff-exists-trans : \langle s \in \mathcal{T} P \longleftrightarrow (\exists P'. P \text{ }_{DT\rightsquigarrow^*} s P') \rangle*

\langle proof \rangle

lemma *tickFree-imp-D-iff-trace-trans-BOT:*

$$\langle \text{tickFree } s \implies s \in \mathcal{D} P \longleftrightarrow P \text{ }_{DT\rightsquigarrow^*} s \perp \rangle$$

\langle proof \rangle

lemma *D-iff-trace-trans-BOT:*

$$\langle s \in \mathcal{D} P \longleftrightarrow (\text{if } \text{tickFree } s \text{ then } P \text{ }_{DT\rightsquigarrow^*} s \perp \text{ else } P \text{ }_{DT\rightsquigarrow^*} (\text{butlast } s) \perp) \rangle$$

\langle proof \rangle

13.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace:*

$$\langle P \text{ }_{DT\rightsquigarrow^*} s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$$

<proof>

lemma *trace-trans-imp-ready-set:*

$\langle P \text{ }_{DT\rightsquigarrow}^*(s @ e \# t) Q \rangle \implies e \in \text{ready-set} (P \text{ afterTrace } s)$
<proof>

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT :*

$\langle (P \text{ }_{DT\rightsquigarrow}^* s Q) \rangle \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \text{ }_{DT\rightsquigarrow\tau} Q$
<proof>

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

$$\frac{\frac{?P \text{ }_{DT\rightsquigarrow} ?e ?P' \quad ?P' \text{ }_{DT\rightsquigarrow\tau} ?P''}{?P \text{ }_{DT\rightsquigarrow} ?e ?P''}}{?P \text{ }_{DT\rightsquigarrow\tau} ?P' \quad ?P' \text{ }_{DT\rightsquigarrow} ?e ?P''}{?P \text{ }_{DT\rightsquigarrow} ?e ?P''}$$

SKIP rule

$$\overline{\overline{SKIP \text{ }_{DT\rightsquigarrow} \checkmark STOP}}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ }_a \text{ }_{DT\rightsquigarrow} ev ?e ?P ?e} \quad \frac{?e \in ?A}{\square a \in ?A \rightarrow ?P \text{ }_a \text{ }_{DT\rightsquigarrow} ev ?e ?P ?e}$$

$$\overline{?e \rightarrow ?P \text{ }_{DT\rightsquigarrow} ev ?e ?P}$$

(\sqcap) rules

$$\frac{\frac{?P \sqcap ?Q \text{ }_{DT\rightsquigarrow\tau} ?P \quad ?P \sqcap ?Q \text{ }_{DT\rightsquigarrow\tau} ?Q}{?e \in ?A}}{\square a \in ?A. ?P \text{ }_a \text{ }_{DT\rightsquigarrow\tau} ?P ?e}$$

$\mu x. f x$ rule

$$\frac{\text{cont } ?f \quad ?P = (\mu x. ?f x)}{?P \text{ }_{DT \rightsquigarrow \tau} ?f ?P}$$

(\square) rules (more powerful than in OpSemFD)

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \square ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \square ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \square ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \square ?Q'}$$

(;) rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P ; ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' ; ?Q} \quad \frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P ; ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' ; ?Q}$$

$$\frac{\exists P'. ?P \text{ }_{DT \rightsquigarrow \checkmark} P'}{?P ; ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q}$$

Hiding rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \setminus ?B \text{ }_{DT \rightsquigarrow \tau} ?P' \setminus ?B} \quad \frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{?P \setminus ?B \text{ }_{DT \rightsquigarrow \checkmark} STOP}$$

$$\frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?e \notin ?B}{?P \setminus ?B \text{ }_{DT \rightsquigarrow ev} ?e ?P' \setminus ?B} \quad \frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?e \in ?B}{?P \setminus ?B \text{ }_{DT \rightsquigarrow \tau} ?P' \setminus ?B}$$

Renaming rules

$$\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{\text{Renaming } ?P ?f \text{ }_{DT \rightsquigarrow \tau} \text{Renaming } ?P' ?f}$$

$$\frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{\text{Renaming } ?P ?f \text{ }_{DT \rightsquigarrow \checkmark} STOP}$$

$$\frac{?f ?a = ?b \quad ?P \text{ }_{DT \rightsquigarrow ev} ?a ?P'}{\text{Renaming } ?P ?f \text{ }_{DT \rightsquigarrow ev} ?b \text{Renaming } ?P' ?f}$$

Sync rules

$$\begin{array}{c}
\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \llbracket ?S \rrbracket ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \llbracket ?S \rrbracket ?Q'} \\
\frac{?e \notin ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \llbracket ?S \rrbracket ?Q} \quad \frac{?e \notin ?S \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P \llbracket ?S \rrbracket ?Q'} \\
\frac{?e \in ?S \quad ?P \text{ }_{DT \rightsquigarrow ev} ?e ?P' \quad ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \llbracket ?S \rrbracket ?Q'} \\
\frac{?P \text{ }_{DT \rightsquigarrow \checkmark} ?P'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow \tau} \text{SKIP} \llbracket ?S \rrbracket ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \checkmark} ?Q'}{?P \llbracket ?S \rrbracket ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \llbracket ?S \rrbracket \text{SKIP}} \\
\frac{}{\text{SKIP} \llbracket ?S \rrbracket \text{SKIP} \text{ }_{DT \rightsquigarrow \tau} \text{SKIP}}
\end{array}$$

(▷) rules

$$\begin{array}{c}
\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \triangleright ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \triangleright ?Q} \quad \frac{?P \text{ }_{DT \rightsquigarrow ?e} ?P'}{?P \triangleright ?Q \text{ }_{DT \rightsquigarrow ?e} ?P'} \\
\frac{}{?P \triangleright ?Q \text{ }_{DT \rightsquigarrow \tau} ?Q}
\end{array}$$

(△) rules

$$\begin{array}{c}
\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow \tau} ?P' \triangle ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow \tau} ?Q'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow \tau} ?P \triangle ?Q'} \\
\frac{?P \text{ }_{DT \rightsquigarrow ev} ?e ?P'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?P' \triangle ?Q} \quad \frac{?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}{?P \triangle ?Q \text{ }_{DT \rightsquigarrow ev} ?e ?Q'}
\end{array}$$

Throw rules

$$\begin{array}{c}
\frac{?P \text{ }_{DT \rightsquigarrow \tau} ?P'}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?P' \Theta a \in ?A. ?Q a} \\
\frac{\forall a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?Q' a}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \tau} ?P \Theta a \in ?A. ?Q' a}
\end{array}$$

$$\frac{\frac{?P \text{ }_{DT \rightsquigarrow ?e} ?P' \quad ?e \notin \text{ev} \text{ } ^{?A}}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow ?e} ?P' \Theta a \in ?A. ?Q a} \quad ?P \text{ }_{DT \rightsquigarrow \text{ev}} ?e ?P' \quad ?e \in ?A}{?P \Theta a \in ?A. ?Q a \text{ }_{DT \rightsquigarrow \text{ev}} ?e ?Q ?e}$$

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  $\langle_{F \rightsquigarrow \tau}$  50)
  where  $\langle P \rangle_{F \rightsquigarrow \tau} Q \equiv P \sqsubseteq_F Q$ 
```

We now instantiate the locale of *HOL-CSP-OpSem.OpSemGenericBis*.

```
interpretation OpSemGeneric  $\langle_{(F \rightsquigarrow \tau)}$  :: ' $\alpha$  process  $\Rightarrow$  ' $\alpha$  process  $\Rightarrow$  bool'
  <proof>
```

```
notation event-trans ( $\langle - \rangle_{F \rightsquigarrow -} - \rangle$  [50, 3, 51] 50)
```

```
notation trace-trans ( $\langle - \rangle_{F \rightsquigarrow * -} - \rangle$  [50, 3, 51] 50)
```

```
lemma  $\langle P \rangle_{F \rightsquigarrow} e P' \Longrightarrow P' \langle_{F \rightsquigarrow \tau} P'' \Longrightarrow P \langle_{F \rightsquigarrow} e P'' \rangle$ 
   $\langle P \rangle_{F \rightsquigarrow \tau} P' \Longrightarrow P' \langle_{F \rightsquigarrow} e P'' \Longrightarrow P \langle_{F \rightsquigarrow} e P'' \rangle$ 
  <proof>
```

14.1 Operational Semantics Laws

SKIP law

```
lemma  $\langle SKIP \rangle_{F \rightsquigarrow} \checkmark STOP \rangle$ 
  <proof>
```

$e \rightarrow P$ laws

```
lemma  $\langle e \in A \Longrightarrow \Box a \in A \rightarrow P a \rangle_{F \rightsquigarrow} (ev e) (P e) \rangle$ 
  <proof>
```

lemma $\langle e \in A \implies \prod a \in A \rightarrow P a \text{ }_{F \rightsquigarrow} (ev\ e) (P\ e) \rangle$
 $\langle proof \rangle$

lemma $\langle e \rightarrow P \text{ }_{F \rightsquigarrow} (ev\ e) P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ }_{F \rightsquigarrow \tau} P \rangle$
and $\langle P \sqcap Q \text{ }_{F \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle a \in A \implies (\prod a \in A. P a) \text{ }_{F \rightsquigarrow \tau} P a \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies (\prod a \in A. P a) \text{ }_{F \rightsquigarrow \tau} P a \rangle$
 $\langle proof \rangle$

$\mu\ x. f\ x$ law

lemma $\langle cont\ f \implies P = (\mu\ X. f\ X) \implies P \text{ }_{F \rightsquigarrow \tau} f\ P \rangle$
 $\langle proof \rangle$

$P \sqcup Q$ laws

lemma $\langle P \text{ }_{F \rightsquigarrow e} P' \implies P \sqcup Q \text{ }_{F \rightsquigarrow e} P' \rangle$
and $\langle Q \text{ }_{F \rightsquigarrow e} Q' \implies P \sqcup Q \text{ }_{F \rightsquigarrow e} Q' \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies P a \text{ }_{F \rightsquigarrow e} Q \implies (\prod a \in A. P a) \text{ }_{F \rightsquigarrow e} Q \rangle$
 $\langle proof \rangle$

$P ; Q$ laws

lemma $\tau\text{-trans-SeqR}$: $\langle \exists P'. P \text{ }_{F \rightsquigarrow} \checkmark P' \implies P ; Q \text{ }_{F \rightsquigarrow \tau} Q \rangle$
 $\langle proof \rangle$

lemma $\langle \checkmark \in ready\text{-set}\ P \implies Q \text{ }_{F \rightsquigarrow} (ev\ e) Q' \implies P ; Q \text{ }_{F \rightsquigarrow} (ev\ e) Q' \rangle$
 $\langle proof \rangle$

$P \setminus B$ laws

lemma $\tau\text{-trans-Hiding}$: $\langle P \text{ }_{F \rightsquigarrow \tau} P' \implies P \setminus B \text{ }_{F \rightsquigarrow \tau} P' \setminus B \rangle$
 $\langle proof \rangle$

lemma $ev\text{-trans-Hiding-notin}$:

$\langle P \text{ }_{F \rightsquigarrow} (ev\ e) P' \implies e \notin B \implies P \setminus B \text{ }_{F \rightsquigarrow} (ev\ e) P' \setminus B \rangle$
 $\langle proof \rangle$

lemma $\langle P \text{ }_{F \rightsquigarrow} \checkmark P' \implies P \setminus B \text{ }_{F \rightsquigarrow} \checkmark STOP \rangle$
 $\langle proof \rangle$

lemma *ev-trans-Hiding-inside*:

$$\langle P \xrightarrow{F} (ev\ e)\ P' \implies e \in B \implies P \setminus B \xrightarrow{F} P' \setminus B \rangle$$

<proof>

Renaming P f laws

lemma *tick-trans-Renaming*: $\langle P \xrightarrow{F} P' \implies \text{Renaming } P\ f \xrightarrow{F} STOP \rangle$
<proof>

lemma $\langle P \xrightarrow{F} P' \implies P \llbracket a := b \rrbracket \xrightarrow{F} STOP \rangle$
<proof>

P [S] 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*: $\langle P \xrightarrow{F} P' \implies P \llbracket S \rrbracket Q \xrightarrow{F} SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \xrightarrow{F} Q' \implies P \llbracket S \rrbracket Q \xrightarrow{F} P \llbracket S \rrbracket SKIP \rangle$
<proof>

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{F} STOP \rangle$
<proof>

lemma τ -*trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{F} SKIP \rangle$
<proof>

P ▷ Q laws

lemma $\langle P \xrightarrow{F} P' \implies P \triangleright Q \xrightarrow{F} P' \rangle$
<proof>

lemma $\langle P \triangleright Q \xrightarrow{F} Q \rangle$
<proof>

P △ Q laws

lemma *Interrupt-ev-trans-right*: $\langle Q \xrightarrow{F} (ev\ e)\ Q' \implies P \triangle Q \xrightarrow{F} (ev\ e)\ Q' \rangle$
<proof>

Throw P A Q laws

lemma *Throw-τ-trans-right*:
 $\langle \forall a \in A. Q\ a \xrightarrow{F} Q'\ a \implies P \ominus a \in A. Q\ a \xrightarrow{F} P \ominus a \in A. Q'\ a \rangle$
<proof>

lemma *Throw-ev-trans-right:*

$$\langle P \xrightarrow{F} (ev\ e)\ P' \implies e \in A \implies P \ominus a \in A.\ Q\ a \xrightarrow{F} (ev\ e)\ (Q\ e) \rangle$$

<proof>

lemma $\langle tickFree\ s \implies \perp \xrightarrow{F}^* s\ P \rangle$

<proof>

14.2 Reality Checks

lemma $\langle STOP \xrightarrow{F}^* s\ P \longleftrightarrow s = \square \wedge P = STOP \rangle$

<proof>

lemma *SKIP-trace-trans-iff :*

$$\langle SKIP \xrightarrow{F}^* s\ P \longleftrightarrow s = \square \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP \rangle$$

<proof>

lemma *F-iff-exists-trans :*

$$\langle (s, X) \in \mathcal{F}\ P \longleftrightarrow (\exists P'. (P \xrightarrow{F}^* s\ P') \wedge X \in \mathcal{R}\ P') \rangle$$

<proof>

lemma *T-iff-exists-trans :* $\langle s \in \mathcal{T}\ P \longleftrightarrow (\exists P'. P \xrightarrow{F}^* s\ P') \rangle$

<proof>

14.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace:*

$$\langle P \xrightarrow{F}^* s\ Q \implies ready\text{-set}\ Q \subseteq ready\text{-set}\ (P\ afterTrace\ s) \rangle$$

<proof>

lemma *trace-trans-imp-ready-set:*

$$\langle P \xrightarrow{F}^* (s\ @\ e\ \#)\ t)\ Q \implies e \in ready\text{-set}\ (P\ afterTrace\ s) \rangle$$

<proof>

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT :*

$$\langle (P \xrightarrow{F}^* s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P\ afterTrace\ s \xrightarrow{F}^* \tau\ Q \rangle$$

<proof>

lemma $\langle deadlock\text{-free}\ P \longleftrightarrow DF\ UNIV \xrightarrow{F}^* \tau\ P \rangle$

<proof>

lemma $\langle deadlock\text{-free}_{SKIP}\ P \longleftrightarrow DF_{SKIP}\ UNIV \xrightarrow{F}^* \tau\ P \rangle$

<proof>

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

$$\frac{?P \xrightarrow{F} ?e \quad ?P'}{?P \xrightarrow{F} ?e \quad ?P'} \quad \frac{?P' \xrightarrow{F} ?P''}{?P' \xrightarrow{F} ?P''} \quad \frac{?P \xrightarrow{F} ?P' \quad ?P' \xrightarrow{F} ?e \quad ?P''}{?P \xrightarrow{F} ?e \quad ?P''}$$

SKIP rule

$$\overline{SKIP \xrightarrow{F} \checkmark \text{ STOP}}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \xrightarrow{F} ev \ ?e \ ?P \ ?e} \quad \frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P \ a \xrightarrow{F} ev \ ?e \ ?P \ ?e}$$

$$\overline{?e \rightarrow ?P \ a \xrightarrow{F} ev \ ?e \ ?P}$$

(\Box) rules

$$\frac{?P \ \Box \ ?Q \xrightarrow{F} ?P \quad ?P \ \Box \ ?Q \xrightarrow{F} ?Q}{?e \in ?A}$$

$$\frac{?e \in ?A}{\Box a \in ?A. \ ?P \ a \xrightarrow{F} ?P \ ?e}$$

$\mu x. f x$ rule

$$\frac{cont \ ?f \quad ?P = (\mu x. \ ?f \ x)}{?P \xrightarrow{F} ?f \ ?P}$$

(\square) rules

$$\frac{?P \xrightarrow{F} ?e \quad ?P'}{?P \ \square \ ?Q \xrightarrow{F} ?e \quad ?P'} \quad \frac{?Q \xrightarrow{F} ?e \quad ?Q'}{?P \ \square \ ?Q \xrightarrow{F} ?e \quad ?Q'}$$

(;) rule

$$\frac{\exists P'. ?P \xrightarrow{F} \checkmark P'}{?P ; ?Q \xrightarrow{F} \tau ?Q}$$

Hiding rules

$$\frac{\frac{?P \xrightarrow{F} \tau ?P'}{?P \setminus ?B \xrightarrow{F} \tau ?P' \setminus ?B} \quad \frac{?P \xrightarrow{F} \checkmark ?P'}{?P \setminus ?B \xrightarrow{F} \checkmark STOP}}{\frac{?P \xrightarrow{F} ev ?e ?P' \quad ?e \notin ?B}{?P \setminus ?B \xrightarrow{F} ev ?e ?P' \setminus ?B}} \quad \frac{?P \xrightarrow{F} \checkmark ?P'}{?P \setminus ?B \xrightarrow{F} \checkmark STOP} \quad \frac{?P \xrightarrow{F} ev ?e ?P' \quad ?e \in ?B}{?P \setminus ?B \xrightarrow{F} \tau ?P' \setminus ?B}}$$

Renaming rule

$$\frac{?P \xrightarrow{F} \checkmark ?P'}{\text{Renaming } ?P \ ?f \xrightarrow{F} \checkmark STOP}$$

Sync rules

$$\frac{\frac{?P \xrightarrow{F} \checkmark ?P'}{?P \llbracket ?S \rrbracket ?Q \xrightarrow{F} \tau SKIP \llbracket ?S \rrbracket ?Q} \quad \frac{?Q \xrightarrow{F} \checkmark ?Q'}{?P \llbracket ?S \rrbracket ?Q \xrightarrow{F} \tau ?P \llbracket ?S \rrbracket SKIP}}{SKIP \llbracket ?S \rrbracket SKIP \xrightarrow{F} \tau SKIP}$$

(▷) rules

$$\frac{\frac{?P \xrightarrow{F} ?e ?P'}{?P \triangleright ?Q \xrightarrow{F} ?e ?P'}}{?P \triangleright ?Q \xrightarrow{F} \tau ?Q}$$

(△) rule

$$\frac{?Q \xrightarrow{F} ev ?e ?Q'}{?P \triangle ?Q \xrightarrow{F} ev ?e ?Q'}$$

Throw rules

$$\frac{\frac{\forall a \in ?A. ?Q a \text{ }_F \rightsquigarrow_{\tau} ?Q' a}{?P \Theta a \in ?A. ?Q a \text{ }_F \rightsquigarrow_{\tau} ?P \Theta a \in ?A. ?Q' a} \quad ?P \text{ }_F \rightsquigarrow_{ev} ?e \text{ } ?P' \quad ?e \in ?A}{?P \Theta a \in ?A. ?Q a \text{ }_F \rightsquigarrow_{ev} ?e \text{ } ?Q ?e}$$

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 $\tau\text{-trans} :: \langle 'a \text{ process} \Rightarrow 'a \text{ process} \Rightarrow \text{bool} \rangle$ (**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* $\langle (T \rightsquigarrow_{\tau}) \rangle$
 $\langle \text{proof} \rangle$

notation *event-trans* $\langle \langle - / T \rightsquigarrow - / - \rangle \rightarrow [50, 3, 51] 50 \rangle$

notation *trace-trans* $\langle \langle - / T \rightsquigarrow^* - / - \rangle \rightarrow [50, 3, 51] 50 \rangle$

lemma $\langle P T \rightsquigarrow e P' \Longrightarrow P' T \rightsquigarrow_{\tau} P'' \Longrightarrow P T \rightsquigarrow e P'' \rangle$
 $\langle P T \rightsquigarrow_{\tau} P' \Longrightarrow P' T \rightsquigarrow e P'' \Longrightarrow P T \rightsquigarrow e P'' \rangle$
 $\langle \text{proof} \rangle$

15.1 Operational Semantics Laws

SKIP law

lemma $\langle \text{SKIP } T \rightsquigarrow \checkmark \text{ STOP} \rangle$
 $\langle \text{proof} \rangle$

$e \rightarrow P$ laws

lemma $\langle e \in A \Longrightarrow \Box a \in A \rightarrow P a T \rightsquigarrow (ev e) (P e) \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle e \in A \implies \sqcap a \in A \rightarrow P a \text{ } T \rightsquigarrow (ev\ e) (P\ e) \rangle$
 $\langle proof \rangle$

lemma $\langle e \rightarrow P \text{ } T \rightsquigarrow (ev\ e) P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma $\langle P \sqcap Q \text{ } T \rightsquigarrow_\tau P \rangle$
and $\langle P \sqcap Q \text{ } T \rightsquigarrow_\tau Q \rangle$
 $\langle proof \rangle$

lemma $\langle a \in A \implies (\sqcap a \in A. P a) \text{ } T \rightsquigarrow_\tau P a \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies (\sqcap a \in A. P a) \text{ } T \rightsquigarrow_\tau P a \rangle$
 $\langle proof \rangle$

$\mu\ x. f\ x$ law

lemma $\langle cont\ f \implies P = (\mu\ X. f\ X) \implies P \text{ } T \rightsquigarrow_\tau f\ P \rangle$
 $\langle proof \rangle$

$P \sqcap Q$ laws

lemma τ -trans-DetL: $\langle P \text{ } T \rightsquigarrow_\tau P' \implies P \sqcap Q \text{ } T \rightsquigarrow_\tau P' \sqcap Q \rangle$
and τ -trans-DetR: $\langle Q \text{ } T \rightsquigarrow_\tau Q' \implies P \sqcap Q \text{ } T \rightsquigarrow_\tau P \sqcap Q' \rangle$
 $\langle proof \rangle$

lemma τ -trans-MultiDet:

$\langle finite\ A \implies \forall a \in A. P a \text{ } T \rightsquigarrow_\tau P' a \implies (\sqcap a \in A. P a) \text{ } T \rightsquigarrow_\tau (\sqcap a \in A. P' a) \rangle$
 $\langle proof \rangle$

lemma $\langle P \text{ } T \rightsquigarrow_e P' \implies P \sqcap Q \text{ } T \rightsquigarrow_e P' \rangle$
and $\langle Q \text{ } T \rightsquigarrow_e Q' \implies P \sqcap Q \text{ } T \rightsquigarrow_e Q' \rangle$
 $\langle proof \rangle$

lemma $\langle finite\ A \implies a \in A \implies P a \text{ } T \rightsquigarrow_e Q \implies (\sqcap a \in A. P a) \text{ } T \rightsquigarrow_e Q \rangle$
 $\langle proof \rangle$

$P ; Q$ laws

lemma τ -trans-SegR: $\langle \exists P'. P \text{ } T \rightsquigarrow \checkmark P' \implies P ; Q \text{ } T \rightsquigarrow_\tau Q \rangle$
 $\langle proof \rangle$

lemma $\langle Q \text{ } T \rightsquigarrow_\tau Q' \implies P ; Q \text{ } T \rightsquigarrow_\tau P ; Q' \rangle$
 $\langle proof \rangle$

lemma $\langle \checkmark \in ready\text{-set}\ P \implies Q \text{ } T \rightsquigarrow (ev\ e) Q' \implies P ; Q \text{ } T \rightsquigarrow (ev\ e) Q' \rangle$

<proof>

$P \setminus B$ laws

lemma τ -trans-Hiding: $\langle P \xrightarrow{\tau} P' \implies P \setminus B \xrightarrow{\tau} P' \setminus B \rangle$
<proof>

lemma *ev-trans-Hiding-notin*:

$\langle P \xrightarrow{\tau}(ev\ e) P' \implies e \notin B \implies P \setminus B \xrightarrow{\tau}(ev\ e) P' \setminus B \rangle$
<proof>

lemma $\langle P \xrightarrow{\tau} \checkmark P' \implies P \setminus B \xrightarrow{\tau} \checkmark STOP \rangle$
<proof>

lemma *ev-trans-Hiding-inside*:

$\langle P \xrightarrow{\tau}(ev\ e) P' \implies e \in B \implies P \setminus B \xrightarrow{\tau} P' \setminus B \rangle$
<proof>

Renaming $P f$ laws

lemma *tick-trans-Renaming*: $\langle P \xrightarrow{\tau} \checkmark P' \implies \text{Renaming } P f \xrightarrow{\tau} \checkmark STOP \rangle$
<proof>

lemma $\langle P \xrightarrow{\tau} \checkmark P' \implies P \llbracket a := b \rrbracket \xrightarrow{\tau} \checkmark STOP \rangle$
<proof>

$P \llbracket S \rrbracket Q$ 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*: $\langle P \xrightarrow{\tau} \checkmark P' \implies P \llbracket S \rrbracket Q \xrightarrow{\tau} \checkmark SKIP \llbracket S \rrbracket Q \rangle$
and *tick-trans-SyncR*: $\langle Q \xrightarrow{\tau} \checkmark Q' \implies P \llbracket S \rrbracket Q \xrightarrow{\tau} \checkmark P \llbracket S \rrbracket SKIP \rangle$
<proof>

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{\tau} \checkmark STOP \rangle$
<proof>

lemma τ -trans-SKIP-Sync-SKIP: $\langle SKIP \llbracket S \rrbracket SKIP \xrightarrow{\tau} SKIP \rangle$
<proof>

$P \triangleright Q$ laws

lemma *Sliding- τ -trans-left*: $\langle P \xrightarrow{\tau} P' \implies P \triangleright Q \xrightarrow{\tau} P' \triangleright Q \rangle$
<proof>

lemma $\langle P \rightsquigarrow_e P' \implies P \triangleright Q \rightsquigarrow_e P' \rangle$
<proof>

lemma $\langle P \triangleright Q \rightsquigarrow_\tau Q \rangle$
<proof>

$P \triangle Q$ laws

lemma *Interrupt- τ -trans-left*: $\langle P \rightsquigarrow_\tau P' \implies P \triangle Q \rightsquigarrow_\tau P' \triangle Q \rangle$
<proof>

lemma *Interrupt- τ -trans-right*: $\langle Q \rightsquigarrow_\tau Q' \implies P \triangle Q \rightsquigarrow_\tau P \triangle Q' \rangle$
<proof>

lemma *Interrupt-ev-trans-left*: $\langle P \rightsquigarrow(ev\ e) P' \implies P \triangle Q \rightsquigarrow(ev\ e) P' \triangle Q \rangle$
<proof>

lemma *Interrupt-ev-trans-right*: $\langle Q \rightsquigarrow(ev\ e) Q' \implies P \triangle Q \rightsquigarrow(ev\ e) Q' \rangle$
<proof>

Throw $P\ A\ Q$ laws

lemma *Throw- τ -trans-right*:
 $\langle \forall a \in A. Q\ a \rightsquigarrow_\tau Q' \ a \implies P \ \Theta \ a \in A. Q\ a \rightsquigarrow_\tau P \ \Theta \ a \in A. Q' \ a \rangle$
<proof>

lemma *Throw-ev-trans-right*:
 $\langle P \rightsquigarrow(ev\ e) P' \implies e \in A \implies P \ \Theta \ a \in A. Q\ a \rightsquigarrow(ev\ e) (Q\ e) \rangle$
<proof>

lemma $\langle tickFree\ s \implies \perp \rightsquigarrow^* s \ P \rangle$
<proof>

15.2 Reality Checks

lemma $\langle STOP \rightsquigarrow^* s \ P \longleftrightarrow s = \square \wedge P = STOP \rangle$
<proof>

lemma *T-iff-exists-trans* : $\langle s \in \mathcal{T} \ P \longleftrightarrow (\exists P'. P \rightsquigarrow^* s \ P') \rangle$
<proof>

15.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:

$$\langle P \xrightarrow{\tau}^* s Q \implies \text{ready-set } Q \subseteq \text{ready-set } (P \text{ afterTrace } s) \rangle$$

\langle proof \rangle

lemma *trace-trans-imp-ready-set*:

$$\langle P \xrightarrow{\tau}^* (s @ e \# t) Q \implies e \in \text{ready-set } (P \text{ afterTrace } s) \rangle$$

\langle proof \rangle

lemma *AfterTrace- τ -trans-if- τ -trans-imp-leT* :

$$\langle (P \xrightarrow{\tau}^* s Q) \longleftrightarrow s \in \mathcal{T} P \wedge P \text{ afterTrace } s \xrightarrow{\tau} Q \rangle$$

\langle proof \rangle

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

$$\frac{?P \xrightarrow{\tau} ?e ?P' \quad ?P' \xrightarrow{\tau} ?P''}{?P \xrightarrow{\tau} ?e ?P''} \quad \frac{?P \xrightarrow{\tau} ?P' \quad ?P' \xrightarrow{\tau} ?e ?P''}{?P \xrightarrow{\tau} ?e ?P''}$$

SKIP rule

$$\overline{SKIP \xrightarrow{\tau} \checkmark STOP}$$

$e \rightarrow P$ rules

$$\frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P a \xrightarrow{\tau} ev ?e ?P ?e} \quad \frac{?e \in ?A}{\Box a \in ?A \rightarrow ?P a \xrightarrow{\tau} ev ?e ?P ?e}$$

$$\overline{?e \rightarrow ?P \xrightarrow{\tau} ev ?e ?P}$$

(\Box) rules

$$\frac{\overline{?P \Box ?Q \xrightarrow{\tau} ?P} \quad \overline{?P \Box ?Q \xrightarrow{\tau} ?Q}}{?e \in ?A} \quad \frac{}{\Box a \in ?A. ?P a \xrightarrow{\tau} ?P ?e}$$

$\mu x. f x$ rule

$$\frac{\text{cont } ?f \quad ?P = (\mu x. ?f x)}{?P \xrightarrow{\tau} ?f ?P}$$

(□) rules

$$\frac{?P \xrightarrow{\tau} ?P'}{?P \square ?Q \xrightarrow{\tau} ?P' \square ?Q} \quad \frac{?Q \xrightarrow{\tau} ?Q'}{?P \square ?Q \xrightarrow{\tau} ?P \square ?Q'}$$

$$\frac{?P \xrightarrow{e} ?P'}{?P \square ?Q \xrightarrow{e} ?P' \square ?Q} \quad \frac{?Q \xrightarrow{e} ?Q'}{?P \square ?Q \xrightarrow{e} ?P \square ?Q'}$$

(;) rule

$$\frac{\exists P'. ?P \xrightarrow{\tau} \checkmark P'}{?P ; ?Q \xrightarrow{\tau} ?Q}$$

Hiding rules

$$\frac{?P \xrightarrow{\tau} ?P'}{?P \setminus ?B \xrightarrow{\tau} ?P' \setminus ?B} \quad \frac{?P \xrightarrow{\tau} \checkmark ?P'}{?P \setminus ?B \xrightarrow{\tau} \checkmark \text{STOP}}$$

$$\frac{?P \xrightarrow{ev} ?e ?P' \quad ?e \notin ?B}{?P \setminus ?B \xrightarrow{ev} ?e ?P' \setminus ?B} \quad \frac{?P \xrightarrow{ev} ?e ?P' \quad ?e \in ?B}{?P \setminus ?B \xrightarrow{ev} ?P' \setminus ?B}$$

Renaming rule

$$\frac{?P \xrightarrow{\tau} \checkmark ?P'}{\text{Renaming } ?P \text{ } ?f \xrightarrow{\tau} \checkmark \text{STOP}}$$

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

$$\frac{\frac{?P \text{ } T \rightsquigarrow_{\tau} ?P'}{?P \triangleright ?Q \text{ } T \rightsquigarrow_{\tau} ?P' \triangleright ?Q} \quad \frac{?P \text{ } T \rightsquigarrow ?e ?P'}{?P \triangleright ?Q \text{ } T \rightsquigarrow ?e ?P'}}{\frac{?P \triangleright ?Q \text{ } T \rightsquigarrow_{\tau} ?Q}$$

(△) rules

$$\frac{\frac{?P \text{ } T \rightsquigarrow_{\tau} ?P'}{?P \triangle ?Q \text{ } T \rightsquigarrow_{\tau} ?P' \triangle ?Q} \quad \frac{?Q \text{ } T \rightsquigarrow_{\tau} ?Q'}{?P \triangle ?Q \text{ } T \rightsquigarrow_{\tau} ?P \triangle ?Q'}}{\frac{?P \text{ } T \rightsquigarrow \text{ev } ?e ?P'}{?P \triangle ?Q \text{ } T \rightsquigarrow \text{ev } ?e ?P' \triangle ?Q} \quad \frac{?Q \text{ } T \rightsquigarrow \text{ev } ?e ?Q'}{?P \triangle ?Q \text{ } T \rightsquigarrow \text{ev } ?e ?Q'}}$$

Throw rules

$$\frac{\frac{\forall a \in ?A. ?Q \ a \text{ } T \rightsquigarrow_{\tau} ?Q' \ a}{?P \ \Theta \ a \in ?A. ?Q \ a \text{ } T \rightsquigarrow_{\tau} ?P \ \Theta \ a \in ?A. ?Q' \ a} \quad \frac{?P \text{ } T \rightsquigarrow \text{ev } ?e ?P' \quad ?e \in ?A}{?P \ \Theta \ a \in ?A. ?Q \ a \text{ } T \rightsquigarrow \text{ev } ?e ?Q \ ?e}}$$

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>

lemma *Renaming-Mprefix-Sliding*:

$$\langle \text{Renaming } ((\Box a \in A \rightarrow P a) \triangleright Q) f =$$
$$(\Box y \in f ' A \rightarrow \Box a \in \{x \in A. y = f x\}. \text{Renaming } (P a) f) \triangleright \text{Renaming } Q f \rangle$$

<proof>

lemma *Renaming-GlobalNdet*:

$$\langle \text{Renaming } (\Box a \in A. P a) f = \Box b \in f ' A. \Box a \in \{a \in A. b = f a\}. \text{Renaming } (P$$
$$a) f \rangle$$

<proof>

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** $\langle ?lhs = ?rhs \rangle$)
 $\langle \text{proof} \rangle$

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*: $\langle \text{strict-mono } g \implies \text{strict-mono } (\lambda i. \text{map } f (g i)) \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-hide-map-EvExt* :

$\langle \text{inj-on } (\text{EvExt } f) (\text{set } s \cup \text{ev } ' S) \implies$
 $\text{trace-hide } (\text{map } (\text{EvExt } f) s) (\text{ev } ' f ' S) =$
 $\text{map } (\text{EvExt } f) (\text{trace-hide } s (\text{ev } ' S)) \rangle$
 $\langle \text{proof} \rangle$

lemma *inj-on-EvExt-iff*: $\langle \text{inj-on } (\text{EvExt } f) (\text{insert tick } (\text{ev } ' S)) \longleftrightarrow \text{inj-on } f S \rangle$
 $\langle \text{proof} \rangle$

lemma *inj-on-EvExt-set-T*:

$\langle \text{inj-on } f (\text{events-of } P) \implies s \in \mathcal{T} P \implies \text{inj-on } (\text{EvExt } f) (\text{insert tick } (\text{set } s)) \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle \text{bij } f \rangle$
 $\langle \text{proof} \rangle$

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*:

$\langle \text{set } u \subseteq \text{range } f \implies \exists t. u = \text{map } f t \rangle$
 — In particular, when f is surjective or bijective.

$\langle \text{proof} \rangle$

lemma *bij-map-setinterleaving-iff-setinterleaving* :

$\langle \text{map } f r \text{ setinterleaves } ((\text{map } f t, \text{map } f u), f ' S) \longleftrightarrow$
 $r \text{ setinterleaves } ((t, u), S) \rangle$ **if** *bij-f* : $\langle \text{bij } f \rangle$

$\langle \text{proof} \rangle$

theorem *bij-Renaming-Sync*:

$\langle \text{Renaming } (P \llbracket S \rrbracket Q) f = \text{Renaming } P f \llbracket f \text{ ' } S \rrbracket \text{Renaming } Q f \rangle$
(is $\langle ?lhs \ P \ Q = ?rhs \ P \ Q \rangle$ **if** *bij-f*: $\langle \text{bij } f \rangle$
 $\langle \text{proof} \rangle$

16.2 Hiding and Mprefix

We already have a way to distribute the *Hiding* operator on the *Mprefix* operator with $S \cap A = \{\}$ \implies $Mprefix \ A \ ?P \setminus S = \Box_{x \in A} \rightarrow (?P \ x \setminus S)$. 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 $\langle s \in \mathcal{D} \ (\Box a \in A - S \rightarrow (P \ a \setminus S)) \rangle$
shows $\langle s \in \mathcal{D} \ (\Box a \in A \rightarrow P \ a \setminus S) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (s, X) \in \mathcal{F} \ (\Box a \in A \rightarrow P \ a \setminus S) \rangle$
 $\langle \text{proof} \rangle$

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*:

$\langle \Box a \in A \rightarrow P \ a \setminus S = \Box a \in A \rightarrow (P \ a \setminus S) \rangle$
(is $\langle ?lhs = ?rhs \rangle$ **if** *disjoint*: $\langle A \cap S = \{\} \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle A \cap S = \{\} \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \Box a \in A \rightarrow P a \setminus S = (\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright (\Box a \in A \cap S. (P a \setminus S)) \rangle$
(is $\langle ?lhs = ?rhs \rangle$ if non-disjoint: $\langle A \cap S \neq \{\} \rangle$)
 $\langle proof \rangle$
lemma $\langle \exists A P S. A \cap S = \{\} \wedge$
 $\Box a \in A::nat\ set \rightarrow P a \setminus S \neq$
 $(\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright (\Box a \in A \cap S. (P a \setminus S)) \rangle$
 $\langle proof \rangle$

And we obtain a similar result when adding a (\triangleright) in the expression.

lemma *Hiding-Mprefix-Sliding-non-disjoint:*
 $\langle ((\Box a \in A \rightarrow P a) \triangleright Q) \setminus S = (\Box a \in A - S \rightarrow (P a \setminus S)) \triangleright$
 $(Q \setminus S) \sqcap (\Box a \in A \cap S. (P a \setminus S)) \rangle$
if non-disjoint: $\langle A \cap S \neq \{\} \rangle$
 $\langle proof \rangle$

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 $\langle \Box a \in A \rightarrow P a \rangle \triangleright (\Box b \in B \rightarrow Q b) \triangleright R = ?term \triangleright R \rangle$)
 $\langle proof \rangle$

lemma *Mprefix-Sliding-Seq:*
 $\langle ((\Box a \in A \rightarrow P a) \triangleright P') ; Q = (\Box a \in A \rightarrow P a ; Q) \triangleright (P' ; Q) \rangle$
 $\langle proof \rangle$

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 $\langle ?lhs = ?rhs \rangle$)
 $\langle proof \rangle$

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.