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

Benoît Ballenghien and Burkhart Wolff

May 26, 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 simultors 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 *ready-set* with *STOP*, *SKIP* and $\bot$ . . . 24
- 3.4 Behaviour of *ready-set* with Operators of HOL-CSP . . . 25
- 3.5 Behaviour of *ready-set* with Operators of HOL-CSPM . . . 27
- 3.6 Behaviour of *ready-set* with Operators of HOL-CSP_OpSem . . . 28

3.7 Behaviour of *ready-set* 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 *After* with *STOP*, *SKIP* and $\bot$ 31
4.5 Behaviour of *After* with Operators of `HOL-CSP` 31
4.5.1 Loss of Determinism 31
4.5.2 *After* Sequential Composition 32
4.5.3 *After* Synchronization 33
4.5.4 *After* Hiding Operator 34
4.5.5 *After* Renaming 35
4.6 Behaviour of *After* with Operators of `HOL-CSPM` 35
4.7 Behaviour of *After* with Operators of `HOL-CSP_OpSem` 36
4.7.1 *After* Sliding 36
4.7.2 *After* Throwing 36
4.7.3 *After* Interrupting 36
4.8 Behaviour of *After* 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 *AfterExt* with *STOP*, *SKIP* and $\bot$ . . . 40
5.1.5 Behaviour of *AfterExt* with Operators of `HOL-CSP` . . 41
5.1.6 Behaviour of *AfterExt* with Operators of `HOL-CSPM` . . 43
5.1.7 Behaviour of *AfterExt* with Operators of `HOL-CSP_OpSem` 43
5.1.8 Behaviour of *AfterExt* 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 *events-of* 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 $P\ afterTrace\ s \rightsquigarrow_{\mathcal{T}} Q$ 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 *AfterExt* with *STOP*, *SKIP* and $\bot$ . . . 76
10.1.5 Behaviour of *AfterExt* with Operators of `HOL-CSP` . . 77
10.1.6 Behaviour of *AfterExt* with Operators of `HOL-CSPM` . . 79
10.1.7 Behaviour of *AfterExt* with Operators of `HOL-CSP_OpSem` 79
10.1.8 Behaviour of *AfterExt* 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 *events-of* 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 $P\ afterTrace\ s \rightsquigarrow_{\mathcal{T}} Q$ 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 *Renaming* . . . 127
16.1.1 Some Generalizations . . . 127
16.1.2 *Renaming* and *Hiding* . . . 128
16.1.3 *Renaming* and *Sync* . . . 128
16.2 *Hiding* and *Mprefix* . . . 129
16.2.1 Two intermediate Results . . . 129
16.2.2 *Hiding* and *Mprefix* for disjoint Sets . . . 129
16.2.3 *Hiding* and *Mprefix* for non-disjoint Sets . . . 129
16.3 (▷) 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

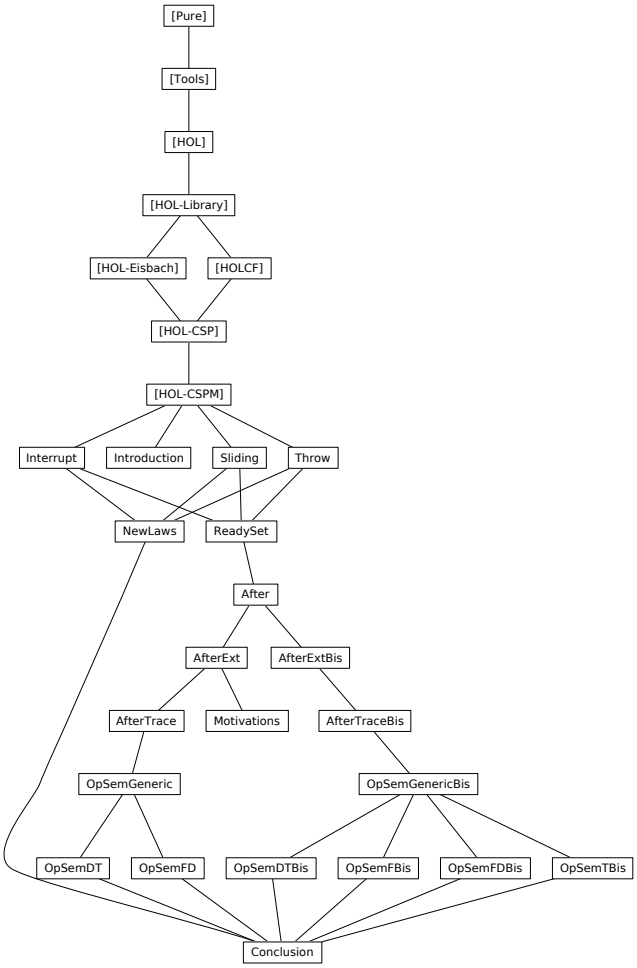

Figure 1.1: The overall architecture

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* :: ‹'α *process* ⇒ 'α *process* ⇒ 'α *process*› (**infixl** ‹▷› 78)
where ‹*P* ▷ *Q* ≡ (*P* □ *Q*) ⊓ *Q*›

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

2.1.2 Projections

lemma *F-Sliding*:
‹ℱ (*P* ▷ *Q*) = ℱ *Q* ∪ {(*s*, *X*). *s* ≠ [] ∧ (*s*, *X*) ∈ ℱ *P* ∨
s = [] ∧ (*s* ∈ 𝒟 *P* ∨ *tick* ∉ *X* ∧ [*tick*] ∈ 𝒯 *P*)}›
⟨*proof*⟩

corollary ‹ℱ (*Mprefix A P* ▷ *Q*) = ℱ *Q* ∪ {(*s*, *X*) ∈ ℱ (*Mprefix A P*). *s* ≠ []}›
⟨*proof*⟩

lemma *D-Sliding*: ‹𝒟 (*P* ▷ *Q*) = 𝒟 *P* ∪ 𝒟 *Q*›
⟨*proof*⟩

lemma *T-Sliding*: ‹$\mathcal{T}\ (P \rhd Q) = \mathcal{T}\ P \cup \mathcal{T}\ Q$›
⟨*proof*⟩

2.1.3 Monotony

lemma *mono-right-Sliding-F*: ‹$Q \sqsubseteq_F Q' \Longrightarrow P \rhd Q \sqsubseteq_F P \rhd Q'$›
and *mono-Sliding-D*: ‹$P \sqsubseteq_D P' \Longrightarrow Q \sqsubseteq_D Q' \Longrightarrow P \rhd Q \sqsubseteq_D P' \rhd Q'$›
and *mono-Sliding-T*: ‹$P \sqsubseteq_T P' \Longrightarrow Q \sqsubseteq_T Q' \Longrightarrow P \rhd Q \sqsubseteq_T P' \rhd Q'$›
and *mono-Sliding-FD*: ‹$P \sqsubseteq_{FD} P' \Longrightarrow Q \sqsubseteq_{FD} Q' \Longrightarrow P \rhd Q \sqsubseteq_{FD} P' \rhd Q'$›
and *mono-Sliding-DT*: ‹$P \sqsubseteq_{DT} P' \Longrightarrow Q \sqsubseteq_{DT} Q' \Longrightarrow P \rhd Q \sqsubseteq_{DT} P' \rhd Q'$›
⟨*proof*⟩

2.1.4 Properties

lemma *Sliding-id*: ‹$P \rhd P = P$›
⟨*proof*⟩

lemma *STOP-Sliding*: ‹$STOP \rhd P = P$›
⟨*proof*⟩

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

lemma ‹$\exists P.\ P \rhd STOP \neq STOP \wedge P \rhd STOP \neq P$›
⟨*proof*⟩

But we still have this result.

lemma *Sliding-is-STOP-iff*: ‹$P \rhd Q = STOP \longleftrightarrow P = STOP \wedge Q = STOP$›
⟨*proof*⟩

lemma *Sliding-STOP-Det*: ‹$(P \rhd STOP) \Box Q = P \rhd Q$›
⟨*proof*⟩

lemma *BOT-Sliding*: ‹$\bot \rhd P = \bot$›
and *Sliding-BOT*: ‹$P \rhd \bot = \bot$›
⟨*proof*⟩

lemma *Sliding-is-BOT-iff*: ‹$P \rhd Q = \bot \longleftrightarrow P = \bot \vee Q = \bot$›
⟨*proof*⟩

lemma *Sliding-assoc*: ‹$P1 \rhd P2 \rhd P3 = P1 \rhd (P2 \rhd P3)$›
⟨*proof*⟩

lemma *SKIP-Sliding*: ‹$SKIP \rhd P = P \sqcap SKIP$›
⟨*proof*⟩

lemma *Sliding-SKIP*: ‹$P \rhd SKIP = P \Box SKIP$›

⟨*proof*⟩

lemma *Sliding-Det*: ‹$(P \rhd P') \mathbin{\square} Q = P \rhd P' \mathbin{\square} Q$›
⟨*proof*⟩

lemma *Sliding-Ndet*: ‹$(P \sqinter P') \rhd Q = (P \rhd Q) \sqinter (P' \rhd Q)$›
‹$P \rhd (Q \sqinter Q') = (P \rhd Q) \sqinter (P \rhd Q')$›
⟨*proof*⟩

lemma *Renaming-Sliding*:
‹*Renaming* $(P \rhd Q)\ f$ = *Renaming* $P\ f \rhd$ *Renaming* $Q\ f$›
⟨*proof*⟩

lemma *events-Sliding*: ‹*events-of* $(P \rhd Q)$ = *events-of* $P \cup$ *events-of* Q›
⟨*proof*⟩

2.1.5 Continuity

From the definition, continuity is obvious.

lemma *Sliding-cont*[*simp*] : ‹*cont* $f \Longrightarrow$ *cont* $g \Longrightarrow$ *cont* $(\lambda x.\ f\ x \rhd g\ x)$›
⟨*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* :: ‹['α process, 'α set, 'α ⇒ 'α process] ⇒ 'α process›
is ‹λ *P A Q*.
({(*t1*, *X*) ∈ ℱ *P*. *set t1* ∩ *ev* ‘ *A* = {}} ∪
{(*t1* @ *t2*, *X*)| *t1 t2 X*. *t1* ∈ 𝒟 *P* ∧ *tickFree t1* ∧
set t1 ∩ *ev* ‘ *A* = {} ∧ *front-tickFree t2*} ∪
{(*t1* @ *ev a* # *t2*, *X*)| *t1 a t2 X*. *t1* @ [*ev a*] ∈ 𝒯 *P* ∧ *set t1* ∩ *ev* ‘ *A* = {} ∧
a ∈ *A* ∧ (*t2*, *X*) ∈ ℱ (*Q a*)},
{*t1* @ *t2*| *t1 t2*. *t1* ∈ 𝒟 *P* ∧ *tickFree t1* ∧
set t1 ∩ *ev* ‘ *A* = {} ∧ *front-tickFree t2*} ∪
{*t1* @ *ev a* # *t2*| *t1 a t2*. *t1* @ [*ev a*] ∈ 𝒯 *P* ∧ *set t1* ∩ *ev* ‘ *A* = {} ∧
a ∈ *A* ∧ *t2* ∈ 𝒟 (*Q a*)})›
⟨*proof*⟩

We add some syntactic sugar.

syntax -*Throw* :: ‹['α process, *pttrn*, 'α set, 'α ⇒ 'α process] ⇒ 'α process›
(‹((-) Θ (-∈-)./ (-))› [*73*, *0*, *0*, *73*] *72*)
translations *P* Θ *a* ∈ *A*. *Q* ⇌ *CONST Throw P A* (λ*a*. *Q*)

abbreviation *Throw-without-free-var* ::
‹['α process, 'α set, 'α process] ⇒ 'α process› (‹((-) Θ (-)/ (-))› [*73*, *0*, *73*] *72*)
where ‹*P* Θ *A Q* ≡ *P* Θ *a* ∈ *A*. *Q*›

Now we can write *P* Θ *a*∈*A*. *Q a*, and when we do not want *Q* to be parameterized we can just write *P* Θ *A Q*.

lemma ‹*P* Θ *a* ∈ *A*. *Q* = *P* Θ *A Q*› ⟨*proof*⟩

2.2.2 Projections

lemma *F-Throw*:
‹ℱ (*P* Θ *a* ∈ *A*. *Q a*) =
{(*t1*, *X*) ∈ ℱ *P*. *set t1* ∩ *ev* ‘ *A* = {}} ∪
{(*t1* @ *t2*, *X*)| *t1 t2 X*.
t1 ∈ 𝒟 *P* ∧ *tickFree t1* ∧ *set t1* ∩ *ev* ‘ *A* = {} ∧ *front-tickFree t2*} ∪
{(*t1* @ *ev a* # *t2*, *X*)| *t1 a t2 X*.
t1 @ [*ev a*] ∈ 𝒯 *P* ∧ *set t1* ∩ *ev* ‘ *A* = {} ∧ *a* ∈ *A* ∧ (*t2*, *X*) ∈ ℱ (*Q a*)}›
⟨*proof*⟩

lemma *D-Throw*:
‹𝒟 (*P* Θ *a* ∈ *A*. *Q a*) =
{*t1* @ *t2*| *t1 t2*.
t1 ∈ 𝒟 *P* ∧ *tickFree t1* ∧ *set t1* ∩ *ev* ‘ *A* = {} ∧ *front-tickFree t2*} ∪
{*t1* @ *ev a* # *t2*| *t1 a t2*.
t1 @ [*ev a*] ∈ 𝒯 *P* ∧ *set t1* ∩ *ev* ‘ *A* = {} ∧ *a* ∈ *A* ∧ *t2* ∈ 𝒟 (*Q a*)}›
⟨*proof*⟩

lemma *T-Throw*:
‹$\mathcal{T}$ (P Θ $a \in A.$ Q a) =
{$t1 \in \mathcal{T}$ P. *set t1* $\cap$ *ev* ‘ $A = \{\}$} $\cup$
{$t1$ @ $t2 |$ $t1$ $t2$.
$t1 \in \mathcal{D}$ $P \wedge$ *tickFree t1* $\wedge$ *set t1* $\cap$ *ev* ‘ $A = \{\} \wedge$ *front-tickFree t2*} $\cup$
{$t1$ @ *ev a* # $t2 |$ $t1$ a $t2$.
$t1$ @ $[ev\ a] \in \mathcal{T}$ $P \wedge$ *set t1* $\cap$ *ev* ‘ $A = \{\} \wedge a \in A \wedge t2 \in \mathcal{T}$ (Q a)}›
⟨*proof*⟩

2.2.3 Monotony

lemma *min-elems-Un-subset*:
‹*min-elems* ($A \cup B$) $\subseteq$ *min-elems* $A \cup$ (*min-elems* $B - A$)›
⟨*proof*⟩

lemma *mono-Throw*[*simp*] : ‹P Θ $a \in A.$ Q $a \sqsubseteq P'$ Θ $a \in A.$ Q' a›
if ‹$P \sqsubseteq P'$› **and** ‹$\forall$ $a \in A.$ Q $a \sqsubseteq Q'$ a›
⟨*proof*⟩

lemma *mono-right-Throw-F* :
‹$\forall$ $a \in A.$ Q $a \sqsubseteq_F$ Q' $a \Longrightarrow$ P Θ $a \in A.$ Q $a \sqsubseteq_F$ P Θ $a \in A.$ Q' a›
⟨*proof*⟩

lemma *mono-right-Throw-T* :
‹$\forall$ $a \in A.$ Q $a \sqsubseteq_T$ Q' $a \Longrightarrow$ P Θ $a \in A.$ Q $a \sqsubseteq_T$ P Θ $a \in A.$ Q' a›
⟨*proof*⟩

lemma *mono-right-Throw-D*:
‹$\forall$ $a \in A.$ Q $a \sqsubseteq_D$ Q' $a \Longrightarrow$ P Θ $a \in A.$ Q $a \sqsubseteq_D$ P Θ $a \in A.$ Q' a›
⟨*proof*⟩

lemma *mono-Throw-FD* : ‹$P \sqsubseteq_{FD}$ $P' \Longrightarrow \forall$ $a \in A.$ Q $a \sqsubseteq_{FD}$ Q' $a \Longrightarrow$
P Θ $a \in A.$ Q $a \sqsubseteq_{FD}$ P' Θ $a \in A.$ Q' a›
⟨*proof*⟩

lemma *mono-Throw-DT* : ‹$P \sqsubseteq_{DT}$ $P' \Longrightarrow \forall$ $a \in A.$ Q $a \sqsubseteq_{DT}$ Q' $a \Longrightarrow$
P Θ $a \in A.$ Q $a \sqsubseteq_{DT}$ P' Θ $a \in A.$ Q' a›
⟨*proof*⟩

2.2.4 Properties

lemma *Throw-STOP*: ‹*STOP* Θ $a \in A.$ Q a = *STOP*›
⟨*proof*⟩

lemma *Throw-SKIP*: ‹*SKIP* Θ $a \in A.$ Q a = *SKIP*›
⟨*proof*⟩

lemma *Throw-BOT*: ‹$\bot$ Θ $a \in A.$ Q a = $\bot$›
⟨*proof*⟩

lemma *Throw-is-BOT-iff*: ‹$P \ominus a \in A.\ Q\ a = \bot \longleftrightarrow P = \bot$›
⟨*proof*⟩

lemma *Throw-empty-set*: ‹$P \ominus a \in \{\}.\ Q\ a = P$›
⟨*proof*⟩

lemma *Throw-Ndet*:
‹$P \sqcap P' \ominus a \in A.\ Q\ a = (P \ominus a \in A.\ Q\ a) \sqcap (P' \ominus a \in A.\ Q\ a)$›
‹$P \ominus a \in A.\ Q\ a \sqcap Q'\ a = (P \ominus a \in A.\ Q\ a) \sqcap (P \ominus a \in A.\ Q'\ a)$›
⟨*proof*⟩

lemma *Throw-Det*:
‹$P \mathbin{\square} P' \ominus a \in A.\ Q\ a = (P \ominus a \in A.\ Q\ a) \mathbin{\square} (P' \ominus a \in A.\ Q\ a)$›
‹$P \ominus a \in A.\ Q\ a \sqcap Q'\ a = (P \ominus a \in A.\ Q\ a) \mathbin{\square} (P \ominus a \in A.\ Q'\ a)$›
⟨*proof*⟩

lemma *Throw-GlobalNdet*:
‹$(\sqcap a \in A.\ P\ a) \ominus b \in B.\ Q\ b = \sqcap a \in A.\ (P\ a \ominus b \in B.\ Q\ b)$›
‹$P' \ominus a \in A.\ (\sqcap b \in B.\ Q'\ a\ b) =$
$(\mathit{if}\ B = \{\}\ \mathit{then}\ P' \ominus A\ \mathit{STOP}\ \mathit{else} \sqcap b \in B.\ (P' \ominus a \in A.\ Q'\ a\ b))$›
⟨*proof*⟩

lemma *Throw-disjoint-events*: ‹$A \cap \mathit{events\text{-}of}\ P = \{\} \Longrightarrow P \ominus a \in A.\ Q\ a = P$›
⟨*proof*⟩

lemma *events-Throw*:
‹$\mathit{events\text{-}of}\ (P \ominus a \in A.\ Q\ a) \subseteq \mathit{events\text{-}of}\ P \cup (\bigcup a \in (A \cap \mathit{events\text{-}of}\ P).\ \mathit{events\text{-}of}\ (Q\ a))$›
⟨*proof*⟩

2.2.5 Key Property

lemma *Throw-Mprefix*:
‹$(\mathbin{\square} a \in A \to P\ a) \ominus b \in B.\ Q\ b =$
$\mathbin{\square} a \in A \to (\mathit{if}\ a \in B\ \mathit{then}\ Q\ a\ \mathit{else}\ P\ a \ominus b \in B.\ Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

corollary *Throw-prefix*: ‹$(a \to P) \ominus b \in B.\ Q\ b =$
$(a \to (\mathit{if}\ a \in B\ \mathit{then}\ Q\ a\ \mathit{else}\ (P \ominus b \in B.\ Q\ b)))$›
⟨*proof*⟩

corollary *Throw-Mndetprefix*:

‹(⊓a ∈ A → P a) Θ b ∈ B. Q b =
⊓a ∈ A → (if a ∈ B then Q a else P a Θ b ∈ B. Q b)›
⟨proof⟩

2.2.6 Continuity

lemma *chain-left-Throw*: ‹*chain Y* ⟹ *chain* (λ*i*. *Y i* Θ *a* ∈ *A*. *Q a*)›
⟨*proof*⟩

lemma *chain-right-Throw*: ‹*chain Y* ⟹ *chain* (λ*i*. *P* Θ *a* ∈ *A*. *Y i a*)›
⟨*proof*⟩

lemma *cont-left-prem-Throw* :
‹(⨆ *i*. *Y i*) Θ *a* ∈ *A*. *Q a* = (⨆ *i*. *Y i* Θ *a* ∈ *A*. *Q a*)›
(**is** ‹*?lhs* = *?rhs*›) **if** *chain* : ‹*chain Y*›
⟨*proof*⟩

lemma *cont-right-prem-Throw* :
‹*P* Θ *a* ∈ *A*. (⨆ *i*. *Y i a*) = (⨆ *i*. *P* Θ *a* ∈ *A*. *Y i a*)›
(**is** ‹*?lhs* = *?rhs*›) **if** *chain* : ‹*chain Y*›
⟨*proof*⟩

lemma *Throw-cont*[*simp*] :
assumes *cont-f* : ‹*cont f*› **and** *cont-g* : ‹∀ *a*. *cont* (*g a*)›
shows ‹*cont* (λ*x*. *f x* Θ *a* ∈ *A*. *g a x*)›
⟨*proof*⟩

end

2.3 The Interrupt Operator

theory *Interrupt*
imports *HOL−CSPM.CSPM*
begin

2.3.1 Definition

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

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

We propose here our corrected version.

lift-definition *Interrupt* :: ‹['α *process*, 'α *process*] ⇒ 'α *process*› (**infixl** ‹△› *75*)

is $\langle \lambda P\ Q.$
$(\{(t1\ @\ [tick],\ X) |\ t1\ X.\ t1\ @\ [tick] \in \mathcal{T}\ P\} \cup$
$\{(t1,\ X - \{tick\}) |\ t1\ X.\ t1\ @\ [tick] \in \mathcal{T}\ P\} \cup$
$\{(t1,\ X) \in \mathcal{F}\ P.\ tickFree\ t1 \wedge ([],\ X) \in \mathcal{F}\ Q\} \cup$
$\{(t1\ @\ t2,\ X) |\ t1\ t2\ X.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge (t2,\ X) \in \mathcal{F}\ Q \wedge t2 \neq []\} \cup$
$\{(t1,\ X - \{tick\}) |\ t1\ X.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge [tick] \in \mathcal{T}\ Q\} \cup$
$\{(t1,\ X).\ t1 \in \mathcal{D}\ P\} \cup$
$\{(t1\ @\ t2,\ X) |\ t1\ t2\ X.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D}\ Q\},$
$\mathcal{D}\ P \cup \{t1\ @\ t2 |\ t1\ t2.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D}\ Q\})\rangle$
⟨*proof*⟩

2.3.2 Projections

lemma *F-Interrupt* :
$\langle \mathcal{F}\ (P \triangle Q) =$
$\{(t1\ @\ [tick],\ X) |\ t1\ X.\ t1\ @\ [tick] \in \mathcal{T}\ P\} \cup$
$\{(t1,\ X - \{tick\}) |\ t1\ X.\ t1\ @\ [tick] \in \mathcal{T}\ P\} \cup$
$\{(t1,\ X) \in \mathcal{F}\ P.\ tickFree\ t1 \wedge ([],\ X) \in \mathcal{F}\ Q\} \cup$
$\{(t1\ @\ t2,\ X) |\ t1\ t2\ X.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge (t2,\ X) \in \mathcal{F}\ Q \wedge t2 \neq []\} \cup$
$\{(t1,\ X - \{tick\}) |\ t1\ X.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge [tick] \in \mathcal{T}\ Q\} \cup$
$\{(t1,\ X).\ t1 \in \mathcal{D}\ P\} \cup$
$\{(t1\ @\ t2,\ X) |\ t1\ t2\ X.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D}\ Q\}\rangle$
⟨*proof*⟩

lemma *D-Interrupt* :
$\langle \mathcal{D}\ (P \triangle Q) = \mathcal{D}\ P \cup \{t1\ @\ t2 |\ t1\ t2.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge t2 \in \mathcal{D}\ Q\}\rangle$
⟨*proof*⟩

lemma *T-Interrupt* :
$\langle \mathcal{T}\ (P \triangle Q) = \mathcal{T}\ P \cup \{t1\ @\ t2 |\ t1\ t2.\ t1 \in \mathcal{T}\ P \wedge tickFree\ t1 \wedge t2 \in \mathcal{T}\ Q\}\rangle$
⟨*proof*⟩

2.3.3 Monotony

lemma *mono-Interrupt*[*simp*]: $\langle P \triangle Q \sqsubseteq P' \triangle Q'\rangle$ **if** $\langle P \sqsubseteq P'\rangle$ **and** $\langle Q \sqsubseteq Q'\rangle$
⟨*proof*⟩

lemma *mono-Interrupt-T*: $\langle P \sqsubseteq_T P' \Longrightarrow Q \sqsubseteq_T Q' \Longrightarrow P \triangle Q \sqsubseteq_T P' \triangle Q'\rangle$
⟨*proof*⟩

lemma *mono-right-Interrupt-D*: $\langle Q \sqsubseteq_D Q' \Longrightarrow P \triangle Q \sqsubseteq_D P \triangle Q'\rangle$
⟨*proof*⟩

lemma *mono-Interrupt-FD*:
$\langle P \sqsubseteq_{FD} P' \Longrightarrow Q \sqsubseteq_{FD} Q' \Longrightarrow P \triangle Q \sqsubseteq_{FD} P' \triangle Q'\rangle$
⟨*proof*⟩

lemma *mono-Interrupt-DT*:
$\langle P \sqsubseteq_{DT} P' \Longrightarrow Q \sqsubseteq_{DT} Q' \Longrightarrow P \mathbin{\triangle} Q \sqsubseteq_{DT} P' \mathbin{\triangle} Q' \rangle$
⟨*proof*⟩

2.3.4 Properties

lemma *Interrupt-STOP-neutral* : $\langle P \mathbin{\triangle} STOP = P \rangle$ $\langle STOP \mathbin{\triangle} P = P \rangle$
⟨*proof*⟩

lemma *Interrupt-BOT-absorb* : $\langle P \mathbin{\triangle} \bot = \bot \rangle$ $\langle \bot \mathbin{\triangle} P = \bot \rangle$
⟨*proof*⟩

lemma *Interrupt-is-BOT-iff* : $\langle P \mathbin{\triangle} Q = \bot \longleftrightarrow P = \bot \vee Q = \bot \rangle$
⟨*proof*⟩

lemma *events-Interrupt*: $\langle$*events-of* $(P \mathbin{\triangle} Q)$ = *events-of* $P \cup$ *events-of* $Q \rangle$
⟨*proof*⟩

lemma *Interrupt-Ndet-distrib* : $\langle P \mathbin{\triangle} Q1 \sqcap Q2 = (P \mathbin{\triangle} Q1) \sqcap (P \mathbin{\triangle} Q2) \rangle$
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *Ndet-Interrupt-distrib* : $\langle P1 \sqcap P2 \mathbin{\triangle} Q = (P1 \mathbin{\triangle} Q) \sqcap (P2 \mathbin{\triangle} Q) \rangle$
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *Interrupt-assoc*: $\langle P \mathbin{\triangle} (Q \mathbin{\triangle} R) = P \mathbin{\triangle} Q \mathbin{\triangle} R \rangle$ (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

2.3.5 Key Property

lemma *Interrupt-Mprefix*:
$\langle (\Box a \in A \to P\ a) \mathbin{\triangle} Q = Q \,\Box\, (\Box a \in A \to P\ a \mathbin{\triangle} Q) \rangle$ (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

corollary $\langle (\Box a \in A \to P\ a) \mathbin{\triangle} (\Box b \in B \to Q\ b) =$
$\Box x \in A \cup B \to ($ *if* $x \in A \cap B$ *then* $(P\ x \mathbin{\triangle} (\Box b \in B \to Q\ b)) \sqcap Q\ x$
else if $x \in A$ *then* $P\ x \mathbin{\triangle} (\Box b \in B \to Q\ b)$
else $Q\ x) \rangle$
⟨*proof*⟩

2.3.6 Continuity

lemma *chain-left-Interrupt*: $\langle$*chain* $Y \Longrightarrow$ *chain* $(\lambda i.\ Y\ i \mathbin{\triangle} Q) \rangle$
⟨*proof*⟩

lemma *chain-right-Interrupt*: ‹*chain* $Y \Longrightarrow$ *chain* $(\lambda i.\ P \triangle Y\ i)$›
⟨*proof*⟩

lemma *cont-left-prem-Interrupt* : ‹$(\bigsqcup\ i.\ Y\ i) \triangle Q = (\bigsqcup\ i.\ Y\ i \triangle Q)$›
(**is** ‹*?lhs* = *?rhs*›) **if** *chain* : ‹*chain* Y›
⟨*proof*⟩

lemma *cont-right-prem-Interrupt* : ‹$P \triangle (\bigsqcup\ i.\ Y\ i) = (\bigsqcup\ i.\ P \triangle Y\ i)$›
(**is** ‹*?lhs* = *?rhs*›) **if** *chain* : ‹*chain* Y›
⟨*proof*⟩

lemma *Interrupt-cont*[*simp*] :
assumes *cont-f* : ‹*cont* f› **and** *cont-g* : ‹*cont* g›
shows ‹*cont* $(\lambda x.\ f\ x \triangle g\ x)$›
⟨*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* (‹✓›)

3.1 Definition

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

definition *ready-set* :: ‹'α *process* ⇒ 'α *event set*›
where ‹*ready-set P* ≡ {*a*. [*a*] ∈ $\mathcal{T}$ *P*}›

lemma *ready-set-def-bis*: ‹*ready-set P* = {*e*. ∃ *s*. *e* # *s* ∈ $\mathcal{T}$ *P*}›
and *Cons-in-T-imp-elem-ready-set*: ‹*e* # *s* ∈ $\mathcal{T}$ *P* ⟹ *e* ∈ *ready-set P*›
⟨*proof*⟩

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*} ∉ $\mathcal{R}$ *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
⟨*proof*⟩
end

3.2 Anti-Mono Rules

lemma *anti-mono-ready-set-T*: ‹$P \sqsubseteq_T Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
⟨*proof*⟩

lemma *anti-mono-ready-set-F*: ‹$P \sqsubseteq_F Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
⟨*proof*⟩

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

lemma *anti-mono-ready-set-FD*: ‹$P \sqsubseteq_{FD} Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
⟨*proof*⟩

lemma *anti-mono-ready-set*: ‹$P \sqsubseteq Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
⟨*proof*⟩

lemma *anti-mono-ready-set-DT*: ‹$P \sqsubseteq_{DT} Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
⟨*proof*⟩

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

lemma *ready-set-STOP*: ‹*ready-set STOP* = {}›
⟨*proof*⟩

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*: ‹*ready-set* $P = \{\} \longleftrightarrow P = STOP$›
⟨*proof*⟩

lemma *ready-set-SKIP*: ‹*ready-set SKIP* = {✓}›
⟨*proof*⟩

lemma *ready-set-BOT*: ‹*ready-set* $\bot$ = *UNIV*›
⟨*proof*⟩

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

lemma ‹∃ *P*. *ready-set P* = {✓} ∧ *P* ≠ *SKIP*›
⟨*proof*⟩

lemma ‹∃ *P*. *ready-set P* = *UNIV* ∧ *P* ≠ ⊥›
⟨*proof*⟩

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

lemma *ready-set-Mprefix*: ‹*ready-set* (□*a* ∈ *A* → *P a*) = *ev* ‘ *A*›
and *ready-set-Mndetprefix*: ‹*ready-set* (⊓*a* ∈ *A* → *P a*) = *ev* ‘ *A*›
and *ready-set-prefix*: ‹*ready-set* (*a* → *Q*) = {*ev a*}›
⟨*proof*⟩

As discussed earlier, *ready-set* behaves well with (□) and (⊓).

lemma *ready-set-Det*: ‹*ready-set* (*P* □ *Q*) = *ready-set P* ∪ *ready-set Q*›
and *ready-set-Ndet*: ‹*ready-set* (*P* ⊓ *Q*) = *ready-set P* ∪ *ready-set Q*›
⟨*proof*⟩

lemma *ready-set-Seq*:
‹*ready-set* (*P* ; *Q*) = (*if P* = ⊥ *then UNIV else ready-set P* − {✓} ∪
(*if* ✓ ∈ *ready-set P then ready-set Q else* {}))›
⟨*proof*⟩

lemma *ready-set-Sync*:
‹*ready-set* (*P* ⟦*S*⟧ *Q*) =
(*if P* = ⊥ ∨ *Q* = ⊥ *then UNIV*
else (*ready-set P* − *insert* ✓ (*ev* ‘ *S*)) ∪
(*ready-set Q* − *insert* ✓ (*ev* ‘ *S*)) ∪
ready-set P ∩ *ready-set Q* ∩ *insert* ✓ (*ev* ‘ *S*))›
(**is** ‹*ready-set* (*P* ⟦*S*⟧ *Q*) = (*if P* = ⊥ ∨ *Q* = ⊥ *then UNIV else ?rhs*)›)
⟨*proof*⟩

lemma *ready-set-Renaming*:
‹*ready-set* (*Renaming P f*) = (*if P* = ⊥ *then UNIV else EvExt f* ‘ (*ready-set P*))›
⟨*proof*⟩

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

lemma *Hiding-is-BOT-iff* :
‹$P \setminus S = \bot \longleftrightarrow (\exists\, t.\ set\ t \subseteq ev\ `\ S \wedge$
$(t \in \mathcal{D}\ P \vee (\exists\ f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f)))$›
(**is** ‹$P \setminus S = \bot \longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

lemma *event-in-D-Hiding-iff* :
‹$[e] \in \mathcal{D}\ (P \setminus S) \longleftrightarrow$
$P \setminus S = \bot \vee (\exists\, x\ t.\ e = ev\ x \wedge x \notin S \wedge [ev\ x] = trace\text{-}hide\ t\ (ev\ `\ S) \wedge$
$(t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f)))$›
(**is** ‹$[e] \in \mathcal{D}\ (P \setminus S) \longleftrightarrow P \setminus S = \bot \vee$ *?ugly-assertion*›)
⟨*proof*⟩

Now we can express *ready-set* $(P \setminus S)$. This result contains the term $P \setminus S = \bot$ 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*:
‹*ready-set* $(P \setminus S) =$
(*if* $P \setminus S = \bot$ *then UNIV else*
$\{e.$ *case* e *of* ✓ $\Rightarrow \exists\, t.\ set\ t \subseteq ev\ `\ S \wedge t\ @\ [✓] \in \mathcal{T}\ P$
$\mid ev\ x \Rightarrow x \notin S \wedge ([ev\ x] \in \mathcal{D}\ (P \setminus S) \vee$
$(\exists\, t.\ [ev\ x] = trace\text{-}hide\ t\ (ev\ `\ S) \wedge (t,\ ev\ `\ S) \in \mathcal{F}\ P))\}$)›
(**is** ‹*ready-set* $(P \setminus S) =$ (*if* $P \setminus S = \bot$ *then UNIV else ?set*)›)
⟨*proof*⟩

In the end the result would look something like this:
ready-set $(P \setminus S) =$ (*if* $\exists\, t.\ set\ t \subseteq ev\ `\ S \wedge (t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f))$ *then UNIV else* $\{e.$ *case* $e$ *of* $ev\ x \Rightarrow x \notin S \wedge (((\exists\, t.\ set\ t \subseteq ev\ `\ S \wedge (t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f))) \vee (\exists\, xa\ t.\ ev\ x = ev\ xa \wedge xa \notin S \wedge [ev\ xa] = trace\text{-}hide\ t\ (ev\ `\ S) \wedge (t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f)))) \vee (\exists\, t.\ [ev\ x] = trace\text{-}hide\ t\ (ev\ `\ S) \wedge (t,\ ev\ `\ S) \in \mathcal{F}\ P)) \mid$ ✓ $\Rightarrow \exists\, t.\ set\ t \subseteq ev\ `\ S \wedge t\ @\ [✓] \in \mathcal{T}\ P\}$)

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

corollary *ready-tick-Hiding-iff* :
‹✓ $\in$ *ready-set* $(P \setminus B) \longleftrightarrow P \setminus B = \bot \vee (\exists\, t.\ set\ t \subseteq ev\ `\ B \wedge t\ @\ [✓] \in \mathcal{T}\ P)$›
⟨*proof*⟩

corollary *ready-tick-imp-ready-tick-Hiding*:
‹✓ $\in$ *ready-set* $P \Longrightarrow$ ✓ $\in$ *ready-set* $(P \setminus B)$›
⟨*proof*⟩

corollary *ready-inside-Hiding-iff* :
‹$e \in S \Longrightarrow ev\ e \in$ *ready-set* $(P \setminus S) \longleftrightarrow P \setminus S = \bot$›
⟨*proof*⟩

corollary *ready-notin-Hiding-iff* :
‹$e \notin S \Longrightarrow ev\ e \in$ *ready-set* $(P \setminus S) \longleftrightarrow$
$P \setminus S = \bot \vee$
$(\exists t.\ [ev\ e] =$ *trace-hide* $t\ (ev\ `\ S) \wedge$
$(t \in \mathcal{D}\ P \vee (\exists f.$ *isInfHiddenRun* $f\ P\ S \wedge t \in$ *range* $f) \vee (t,\ ev\ `\ S) \in \mathcal{F}\ P))$›
⟨*proof*⟩

corollary *ready-notin-imp-ready-Hiding*:
‹*ev* $e \in$ *ready-set* $(P \setminus S)$› **if** *ready* : ‹*ev* $e \in$ *ready-set* P› **and** *notin* : ‹$e \notin S$›
⟨*proof*⟩

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

lemma *ready-set-MultiDet*:
‹*finite* $A \Longrightarrow$ *ready-set* $(MultiDet\ A\ P) = (\bigcup a \in A.$ *ready-set* $(P\ a))$›
⟨*proof*⟩

lemma *ready-set-MultiNdet*:
‹*finite* $A \Longrightarrow$ *ready-set* $(MultiNdet\ A\ P) = (\bigcup a \in A.$ *ready-set* $(P\ a))$›
⟨*proof*⟩

lemma *ready-set-GlobalNdet*:
‹*ready-set* $(GlobalNdet\ A\ P) = (\bigcup a \in A.$ *ready-set* $(P\ a))$›
⟨*proof*⟩

lemma *ready-set-MultiSeq*:
‹*ready-set* $(MultiSeq\ L\ P) =$
$($ *if* $L = []$ *then* $\{\checkmark\}$
else if $P\ (hd\ L) = \bot$ *then* $UNIV$
else if $\checkmark \in$ *ready-set* $(P\ (hd\ L))$
then ready-set $(P\ (hd\ L)) - \{\checkmark\} \cup$ *ready-set* $(MultiSeq\ (tl\ L)\ P)$
else ready-set $(P\ (hd\ L)))$›
⟨*proof*⟩

lemma *ready-set-MultiSync*:
‹*ready-set* $([\![S]\!]\ m \in\#\ M.\ P\ m) =$
$($ *if* $M = \{\#\}$ *then* $\{\}$
else if $\exists m \in\#\ M.\ P\ m = \bot$ *then* $UNIV$
else if $\exists m.\ M = \{\#m\#\}$ *then ready-set* $(P\ (THE\ m.\ M = \{\#m\#\}))$
else $\{e.\ \exists m \in\#\ M.\ e \in$ *ready-set* $(P\ m) -$ *insert* $\checkmark\ (ev\ `\ S)\} \cup$
$\{e \in$ *insert* $\checkmark\ (ev\ `\ S).\ \forall m \in\#\ M.\ e \in$ *ready-set* $(P\ m)\}$)›
⟨*proof*⟩

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

lemma *ready-set-Sliding*:
‹*ready-set* $(P \rhd Q) = \textit{ready-set}\ P \cup \textit{ready-set}\ Q$›
⟨*proof*⟩

lemma *ready-set-Throw*: ‹*ready-set* $(P\ \Theta\ a \in A.\ Q\ a) = \textit{ready-set}\ P$›
⟨*proof*⟩

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

lemma *ready-set-Interrupt*: ‹*ready-set* $(P \triangle Q) = \textit{ready-set}\ P \cup \textit{ready-set}\ Q$›
⟨*proof*⟩

corollary *Interrupt-is-STOP-iff*: ‹$P \triangle Q = STOP \longleftrightarrow P = STOP \wedge Q = STOP$›
⟨*proof*⟩

3.7 Behaviour of *ready-set* with Reference Processes

lemma *ready-set-DF*: ‹*ready-set* $(DF\ A) = ev\ `\ A$›
⟨*proof*⟩

lemma *ready-set-*DF_{SKIP}: ‹*ready-set* $(DF_{SKIP}\ A) = \textit{insert}\ ✓\ (ev\ `\ A)$›
⟨*proof*⟩

lemma *ready-set-RUN*: ‹*ready-set* $(RUN\ A) = ev\ `\ A$›
⟨*proof*⟩

lemma *ready-set-CHAOS*: ‹*ready-set* $(CHAOS\ A) = ev\ `\ A$›
⟨*proof*⟩

lemma *ready-set-*$CHAOS_{SKIP}$: ‹*ready-set* $(CHAOS_{SKIP}\ A) = \textit{insert}\ ✓\ (ev\ `\ A)$›
⟨*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-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-set P* to define operational semantics.

lift-definition *After* :: ‹['α *process*, 'α] $\Rightarrow$ 'α *process*› (**infixl** ‹*after*› *77*)
is ‹$\lambda P\ e.$ *if ev e* $\in$ *ready-set P*
then ({(*tl s*, *X*)| *s X*. (*s*, *X*) $\in \mathcal{F}\ P \wedge s \neq [] \wedge hd\ s = ev\ e$},
{ *tl s* | *s* . *s* $\in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = ev\ e$})
else ({(*s*, *X*). *s* = []}, {})›
⟨*proof*⟩

4.2 Projections

lemma *F-After*:

‹$\mathcal{F}$ (P *after* e) = (*if* *ev* $e \in$ *ready-set* P
then $\{(tl\ s,\ X) |\ s\ X.\ (s,\ X) \in \mathcal{F}\ P \wedge s \neq [] \wedge hd\ s = ev\ e\}$
else $\{(s,\ X).\ s = []\})$›
⟨*proof*⟩

lemma *D-After*:
‹$\mathcal{D}$ (P *after* e) = (*if* *ev* $e \in$ *ready-set* P
then $\{tl\ s |\ s.\ s \in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = ev\ e\}$
else $\{\}$)›
⟨*proof*⟩

lemma *T-After*:
‹$\mathcal{T}$ (P *after* e) = (*if* *ev* $e \in$ *ready-set* P
then $\{tl\ s |\ s.\ s \in \mathcal{T}\ P \wedge s \neq [] \wedge hd\ s = ev\ e\}$
else $\{[]\}$)›
⟨*proof*⟩

lemma *not-ready-After*: ‹*ev* $e \notin$ *ready-set* $P \Longrightarrow P$ *after* $e = STOP$›
⟨*proof*⟩

lemma *ready-set-After*: ‹*ready-set* (P *after* e) = $\{a.\ ev\ e\ \#\ [a] \in \mathcal{T}\ P\}$›
⟨*proof*⟩

4.3 Monotony

lemma *mono-After* : ‹P *after* $e \sqsubseteq Q$ *after* e› **if** ‹$P \sqsubseteq Q$›
⟨*proof*⟩

lemma *mono-After-T* : ‹$P \sqsubseteq_T Q \Longrightarrow P$ *after* $e \sqsubseteq_T Q$ *after* e›
⟨*proof*⟩

lemma *mono-After-F* :
‹$P \sqsubseteq_F Q \Longrightarrow$ *ev* $e \notin$ *ready-set* $P \vee$ *ev* $e \in$ *ready-set* $Q \Longrightarrow$
P *after* $e \sqsubseteq_F Q$ *after* e›
⟨*proof*⟩

lemma *mono-After-D* : ‹$P \sqsubseteq_D Q \Longrightarrow P$ *after* $e \sqsubseteq_D Q$ *after* e›
⟨*proof*⟩

lemma *mono-After-FD* :
‹$P \sqsubseteq_{FD} Q \Longrightarrow$ *ev* $e \notin$ *ready-set* $P \vee$ *ev* $e \in$ *ready-set* $Q \Longrightarrow$
P *after* $e \sqsubseteq_{FD} Q$ *after* e›
⟨*proof*⟩

lemma *mono-After-DT* : ‹$P \sqsubseteq_{DT} Q \Longrightarrow P$ *after* $e \sqsubseteq_{DT} Q$ *after* e›
⟨*proof*⟩

4.4 Behaviour of *After* with *STOP*, *SKIP* and $\bot$

lemma *After-STOP*: ‹*STOP after* e = *STOP*›
⟨*proof*⟩

lemma *After-is-STOP-iff*:
‹P *after* e = *STOP* $\longleftrightarrow$ $(\forall s.\ ev\ e\ \#\ s \in \mathcal{T}\ P \longrightarrow s = [])$›
⟨*proof*⟩

lemma *After-SKIP*: ‹*SKIP after* e = *STOP*›
⟨*proof*⟩

lemma *After-BOT*: ‹$\bot$ *after* $e = \bot$›
⟨*proof*⟩

lemma *After-is-BOT-iff*: ‹P *after* $e = \bot \longleftrightarrow [ev\ e] \in \mathcal{D}\ P$›
⟨*proof*⟩

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*:
‹$(\Box a \in A \rightarrow P\ a)$ *after* $e = (\sqcap a \in A \rightarrow P\ a)$ *after* e›
⟨*proof*⟩

lemma *After-Det-is-After-Ndet*: ‹$P \Box Q$ *after* $e = P \sqcap Q$ *after* e›
⟨*proof*⟩

lemma *After-Ndet*:
‹$P \sqcap Q$ *after* e =
(*if* $ev\ e \notin$ *ready-set* $P \wedge ev\ e \notin$ *ready-set* Q *then STOP*
else if $ev\ e \in$ *ready-set* $P \wedge ev\ e \in$ *ready-set* Q *then* $(P$ *after* $e) \sqcap (Q$ *after* $e)$
else if $ev\ e \in$ *ready-set* P *then* P *after* e *else* Q *after* $e)$›
(**is** ‹$P \sqcap Q$ *after* e =
(*if ?c1 then STOP else if ?c2 then* $(P$ *after* $e) \sqcap (Q$ *after* $e)$ *else*
if ?c3 then P *after* e *else* Q *after* $e)$›)
⟨*proof*⟩

lemma *After-Mprefix*: ‹(□ $a \in A \to P\ a$) *after* e = (*if* $e \in A$ *then* $P\ e$ *else* *STOP*)›
⟨*proof*⟩

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* ‹{a}› ‹λa. P›, *folded write0-def*, *simplified*]
for a P

lemma ‹($e \to P$) *after* e = P› ⟨*proof*⟩

This result justifies seeing P *after* e as the reciprocal operator of the prefix $e \to P$.

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

lemma ‹∃ P. ($e \to (P\ \textit{after}\ e) \neq P$)›
⟨*proof*⟩

lemma ‹∃ P. $e \in$ *ready-set* $P \wedge (e \to (P\ \textit{after}\ e) \neq P)$›
⟨*proof*⟩

lemma ‹∃ P. $e \in$ *ready-set* $P \wedge P \neq \bot \wedge (e \to (P\ \textit{after}\ e) \neq P)$›
⟨*proof*⟩

4.5.2 *After* Sequential Composition

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

lemma *not-skippable-or-not-readyR-After-Seq*: ‹($P\ ;\ Q$) *after* e = P *after* $e\ ;\ Q$›
if ‹✓ ∉ *ready-set* $P \vee e \notin$ *ready-set* Q›
⟨*proof*⟩

lemma ‹($P\ ;$ *STOP*) *after* e = P *after* $e\ ;$ *STOP*›
⟨*proof*⟩

lemma *skippable-not-readyL-After-Seq*: ‹($P\ ;\ Q$) *after* e = Q *after* e›
if ‹✓ ∈ *ready-set* P› **and** ‹e ∉ *ready-set* P›
⟨*proof*⟩

lemma *skippable-readyL-readyR-After-Seq*: ‹($P\ ;\ Q$) *after* e = (P *after* $e\ ;\ Q$) ⊓ (Q *after* e)›
if ‹✓ ∈ *ready-set* P› ‹$e \in$ *ready-set* P› ‹$e \in$ *ready-set* Q›
⟨*proof*⟩

lemma *not-readyL-not-skippable-or-not-readyR-After-Seq*:
‹*ev e* ∉ *ready-set P* ⟹ ✓ ∉ *ready-set P* ∨ *ev e* ∉ *ready-set Q* ⟹
(*P* ; *Q*) *after e* = *STOP*›
⟨*proof*⟩

lemma *After-Seq*:
‹(*P* ; *Q*) *after e* =
(*if ev e* ∉ *ready-set P* ∧ *ev e* ∉ *ready-set Q then STOP*
else if ev e ∉ *ready-set Q then P after e* ; *Q*
else if ev e ∉ *ready-set P then if* ✓ ∈ *ready-set P then Q after e else STOP*
else if ✓ ∈ *ready-set P then* (*P after e* ; *Q*) ⊓ (*Q after e*) *else P after e* ; *Q*)›
⟨*proof*⟩

4.5.3 *After* Synchronization

Now let's focus on *Sync*. We want to obtain an equivalent of
⟦*?A* ∩ *?S* = {}; *?A′* ⊆ *?S*; *?B* ∩ *?S* = {}; *?B′* ⊆ *?S*⟧ ⟹ *Mprefix* (*?A* ∪ *?A′*) *?P* ⟦*?S*⟧ *Mprefix* (*?B* ∪ *?B′*) *?Q* = (□*x*∈*?A* → *?P x* ⟦*?S*⟧ *Mprefix* (*?B* ∪ *?B′*) *?Q*) □ (□*y*∈*?B* → *Mprefix* (*?A* ∪ *?A′*) *?P* ⟦*?S*⟧ *?Q y*) □ (□*x*∈*?A′* ∩ *?B′* → *?P x* ⟦*?S*⟧ *?Q x*)
We will also divide the task.

After version of
⟦*e* ∉ *?S*; *?B′* ⊆ *?S*⟧ ⟹ *e* → *P* ⟦*?S*⟧ *Mprefix ?B′ ?Q* = *e* → (*P* ⟦*?S*⟧ *Mprefix ?B′ ?Q*).

lemma *tickFree-tl*: ‹*tickFree s* ⟹ *tickFree*(*tl s*)›
— Remove this lemma, already in future versions of `HOL-CSP`.
⟨*proof*⟩

lemma *readyL-not-readyR-not-in-After-Sync*:
‹(*P* ⟦*S*⟧ *Q*) *after e* = *P after e* ⟦*S*⟧ *Q*›
if *ready-hyps*: ‹*ev e* ∈ *ready-set P*› ‹*ev e* ∉ *ready-set Q*› **and** *notin*: ‹*e* ∉ *S*›
⟨*proof*⟩

lemma *not-readyL-in-After-Sync*:
‹*ev e* ∉ *ready-set P* ⟹ *e* ∈ *S* ⟹
(*P* ⟦*S*⟧ *Q*) *after e* = (*if Q* = ⊥ *then* ⊥ *else STOP*)›
⟨*proof*⟩

After version of ⟦*e* ∈ *?S*; *e* ∈ *?S*⟧ ⟹ *e* → *P* ⟦*?S*⟧ *e* → *Q* = *e* → (*P* ⟦*?S*⟧ *Q*).

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

‹(*P* ⟦*S*⟧ *Q*) *after e* = *P after e* ⟦*S*⟧ *Q after e*›
if *ready-hyps*: ‹*ev e* ∈ *ready-set P*› ‹*ev e* ∈ *ready-set Q*› **and** *inside*: ‹*e* ∈ *S*›
⟨*proof*⟩

After version of
⟦*e* ∉ *?S*; *e* ∉ *?S*⟧ ⟹ *e* → *P* ⟦*?S*⟧ *e* → *Q* = (*e* → (*P* ⟦*?S*⟧ *e* → *Q*)) □ (*e* → (*e* → *P* ⟦*?S*⟧ *Q*)).

lemma *readyL-readyR-not-in-After-Sync*:
‹(*P* ⟦*S*⟧ *Q*) *after e* = (*P after e* ⟦*S*⟧ *Q*) ⊓ (*P* ⟦*S*⟧ *Q after e*)›
if *ready-hyps*: ‹*ev e* ∈ *ready-set P*› ‹*ev e* ∈ *ready-set Q*› **and** *notin*: ‹*e* ∉ *S*›
⟨*proof*⟩

lemma *not-readyL-not-readyR-After-Sync*: ‹(*P* ⟦*S*⟧ *Q*) *after e* = *STOP*›
if *ready-hyps*: ‹*ev e* ∉ *ready-set P*› ‹*ev e* ∉ *ready-set Q*›
⟨*proof*⟩

Finally, the monster theorem !

theorem *After-Sync*:
‹(*P* ⟦*S*⟧ *Q*) *after e* =
(*if P* = ⊥ ∨ *Q* = ⊥ *then* ⊥
else if ev e ∈ *ready-set P* ∧ *ev e* ∈ *ready-set Q*
then if e ∈ *S then P after e* ⟦*S*⟧ *Q after e*
else (*P after e* ⟦*S*⟧ *Q*) ⊓ (*P* ⟦*S*⟧ *Q after e*)
else if e ∈ *S then STOP*
else if ev e ∈ *ready-set P then P after e* ⟦*S*⟧ *Q*
else if ev e ∈ *ready-set Q then P* ⟦*S*⟧ *Q after e*
else STOP)›
⟨*proof*⟩

4.5.4 *After* Hiding Operator

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

lemma *Hiding-FD-Hiding-After-if-ready-inside*:
‹*e* ∈ *B* ⟹ (*P* \ *B*) ⊑$_{FD}$ (*P after e* \ *B*)›
and *After-Hiding-FD-Hiding-After-if-ready-notin*:
‹*e* ∉ *B* ⟹ (*P* \ *B*) *after e* ⊑$_{FD}$ (*P after e* \ *B*)›
if *ready*: ‹*ev e* ∈ *ready-set P*›
⟨*proof*⟩

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

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

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

corollary *Hiding-DT-Hiding-After-if-ready-inside*:
‹*ev e* ∈ *ready-set P* ⟹ *e* ∈ *B* ⟹ (*P* \ *B*) ⊑$_{DT}$ (*P after e* \ *B*)›
and *After-Hiding-DT-Hiding-After-if-ready-notin*:
‹*ev e* ∈ *ready-set P* ⟹ *e* ∉ *B* ⟹ (*P* \ *B*) *after e* ⊑$_{DT}$ (*P after e* \ *B*)›
⟨*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*: ‹*P* ≡ (*Suc 0* → (*0* → *STOP*)) ⊓ (*0* → *SKIP*)›
and *B-def*: ‹*B* ≡ {*Suc 0*}› **and** *e-def* : ‹*e* ≡ *0*› **and** *f-def*: ‹*f* ≡ *Suc 0*›
shows ‹*ev e* ∈ *ready-set P* ∧ *f* ∈ *B* ∧ *P* \ *B* ≠ *P after f* \ *B*›
and ‹*ev e* ∈ *ready-set P* ∧ *e* ∉ *B* ∧ (*P* \ *B*) *after e* ≠ *P after e* \ *B*›
⟨*proof*⟩

4.5.5 *After* Renaming

lemma *After-Renaming*:
‹*Renaming P f after e* =
(*if P* = ⊥ *then* ⊥ *else*
⊓*a* ∈ {*a*. *ev a* ∈ *ready-set P* ∧ *f a* = *e*}. *Renaming* (*P after a*) *f*)›
(**is** ‹*?lhs* = (*if P* = ⊥ *then* ⊥ *else ?rhs*)›)
— We treat the case *P* = ⊥ separately because the set {*a*. *f a* = *e*} may be empty, which implies ⊓ *a*∈{*a*. *ev a* ∈ *ready-set P* ∧ *f a* = *e*}. *Renaming* (*P after a*) *f* = *STOP* while *Renaming P f after e* = ⊥.

⟨*proof*⟩

4.6 Behaviour of *After* with Operators of `HOL-CSPM`

lemma *After-MultiDet-is-After-MultiNdet*:
‹*finite A* ⟹ (□ *a* ∈ *A*. *P a*) *after e* = (⨅ *a* ∈ *A*. *P a*) *after e*›
⟨*proof*⟩

lemma *After-GlobalNdet*: ‹(⊓ *a* ∈ *A*. *P a*) *after e* =
(*if ev e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⊓ *a* ∈ {*a* ∈ *A*. *ev e* ∈ *ready-set* (*P a*)}. *P a*) *after e*)›
⟨*proof*⟩

lemma *After-MultiNdet*: ‹*finite A* ⟹ (⨅ *a* ∈ *A*. *P a*) *after e* =

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

⟨*proof*⟩

lemma *After-MultiDet*: ‹*finite* $A \Longrightarrow$
$(\Box\ a \in A.\ P\ a)\ \textit{after}\ e =$
$(\ \textit{if } ev\ e \notin (\bigcup a \in A.\ \textit{ready-set}\ (P\ a))\ \textit{then STOP}$
$\ \textit{else } (\prod\ a \in \{a \in A.\ ev\ e \in \textit{ready-set}\ (P\ a)\}.\ P\ a)\ \textit{after}\ e)\rangle$

⟨*proof*⟩

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

4.7.1 *After* Sliding

lemma *After-Sliding*:
‹$P \rhd Q\ \textit{after}\ e =$
$(\ \textit{if } ev\ e \notin \textit{ready-set}\ P \wedge ev\ e \notin \textit{ready-set}\ Q\ \textit{then STOP}$
$\ \textit{else}\quad \textit{if } ev\ e \in \textit{ready-set}\ P \wedge ev\ e \in \textit{ready-set}\ Q\ \textit{then}\ (P\ \textit{after}\ e) \sqcap (Q\ \textit{after}\ e)$
$\quad\ \textit{else if } ev\ e \in \textit{ready-set}\ P\ \textit{then}\ P\ \textit{after}\ e\ \textit{else}\ Q\ \textit{after}\ e)\rangle$

⟨*proof*⟩

An interesting corollary is that ($\rhd$) is also concerned by the loss of determinism (see *?P* $\Box$ *?Q after ?e* = *?P* $\sqcap$ *?Q after ?e*).

lemma *After-Sliding-is-After-Ndet*: ‹$P \rhd Q\ \textit{after}\ e = P \sqcap Q\ \textit{after}\ e\rangle$

⟨*proof*⟩

4.7.2 *After* Throwing

lemma *After-Throw*:
‹$(P\ \Theta\ a \in A.\ Q\ a)\ \textit{after}\ e =$
$(\ \textit{if } P = \bot\ \textit{then} \bot$
$\ \textit{else}\quad \textit{if } ev\ e \in \textit{ready-set}\ P\ \textit{then}\quad \textit{if } e \in A\ \textit{then}\ Q\ e$
$\qquad\qquad\qquad\qquad\qquad \textit{else}\ (P\ \textit{after}\ e)\ \Theta\ a \in A.\ Q\ a$
$\quad\ \textit{else STOP})\rangle$
(**is** ‹*?lhs* = *?rhs*›)

⟨*proof*⟩

4.7.3 *After* Interrupting

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

⟨*proof*⟩

4.8 Behaviour of *After* with Reference Processes

lemma *After-DF*: ‹*DF A after e* = (*if e* ∈ *A then DF A else STOP*)›
⟨*proof*⟩

lemma *After-DF*$_{SKIP}$:
‹*DF*$_{SKIP}$ *A after e* = (*if e* ∈ *A then DF*$_{SKIP}$ *A else STOP*)›
⟨*proof*⟩

lemma *After-RUN*: ‹*RUN A after e* = (*if e* ∈ *A then RUN A else STOP*)›
⟨*proof*⟩

lemma *After-CHAOS*: ‹*CHAOS A after e* = (*if e* ∈ *A then CHAOS A else STOP*)›
⟨*proof*⟩

lemma *After-CHAOS*$_{SKIP}$:
‹*CHAOS*$_{SKIP}$ *A after e* = (*if e* ∈ *A then CHAOS*$_{SKIP}$ *A else STOP*)›
⟨*proof*⟩

lemma *DF-FD-After*: ‹*DF A* ⊑$_{FD}$ *P after e*›
if ‹*DF A* ⊑$_{FD}$ *P*› **and** ‹*ev e* ∈ *ready-set P*›
⟨*proof*⟩

lemma *DF*$_{SKIP}$*-FD-After*: ‹*DF*$_{SKIP}$ *A* ⊑$_{FD}$ *P after e*›
if ‹*DF*$_{SKIP}$ *A* ⊑$_{FD}$ *P*› **and** ‹*ev e* ∈ *ready-set P*›
⟨*proof*⟩

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

corollary *deadlock-free-After*:
‹*deadlock-free P* ⟹
deadlock-free (*P after e*) ⟷ (*if ev e* ∈ *ready-set P then True else False*)›
⟨*proof*⟩

corollary *deadlock-free*$_{SKIP}$*-After*:
‹*deadlock-free*$_{SKIP}$ *P* ⟹
deadlock-free$_{SKIP}$ (*P after e*) ⟷ (*if ev e* ∈ *ready-set P then True else False*)›
⟨*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 after e* for $P::'\alpha$ *process* and $e::'\alpha$; in other words we cannot handle ✓. We now introduce a generalisation for $e::'\alpha$ *event*.

definition *AfterExt* :: ‹['α process, 'α event] ⇒ 'α process› (**infixl** ‹afterExt› 77)
where ‹*P afterExt e* ≡ *case e of ev x* ⇒ *P after x*
| ✓ ⇒ *if P* = ⊥ *then* ⊥ *else STOP*›

lemma *not-ready-AfterExt*:
‹*e* ∉ *ready-set P* ⟹ *P afterExt e* = (*if P* = ⊥ *then* ⊥ *else STOP*)›
⟨*proof*⟩

lemma *ready-set-AfterExt*:
‹*ready-set* (*P afterExt e*) =
(*if P* = ⊥ *then UNIV else if e* = ✓ *then* {} *else* {*a*. *e* # [*a*] ∈ 𝒯 *P*})›
⟨*proof*⟩

5.1.2 Projections

lemma *F-AfterExt*:
‹ℱ (*P afterExt e*) =
(*if e* = ✓ ∧ *P* = ⊥ *then* {(*s*, *X*). *front-tickFree s*}
else if e ∈ *ready-set P then* {(*tl s*, *X*)| *s X*. (*s*, *X*) ∈ ℱ *P* ∧ *s* ≠ [] ∧ *hd s* = *e*}
else {(*s*, *X*). *s* = []})›

(**is** ‹- = *?rhs*›)
⟨*proof*⟩

lemma *D-AfterExt*:
‹$\mathcal{D}$ (*P afterExt e*) = (*if e* = ✓ ∧ *P* = ⊥ *then* {*s*. *front-tickFree s*}
else {*tl s*| *s* . *s* ∈ $\mathcal{D}$ *P* ∧ *s* ≠ [] ∧ *hd s* = *e*})›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

lemma *T-AfterExt*:
‹$\mathcal{T}$ (*P afterExt e*) = (*if e* = ✓ ∧ *P* = ⊥ *then* {*s*. *front-tickFree s*}
else insert [] {*tl s*| *s* . *s* ∈ $\mathcal{T}$ *P* ∧ *s* ≠ [] ∧ *hd s* = *e*})›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

5.1.3 Monotony

lemma *mono-AfterExt* : ‹*P* ⊑ *Q* ⟹ *P afterExt e* ⊑ *Q afterExt e*›
⟨*proof*⟩

lemma *mono-AfterExt-T* : ‹*P* $\sqsubseteq_T$ *Q* ⟹ *e* ≠ ✓ ∨ *P* = ⊥ ∨ *Q* ≠ ⊥ ⟹
P afterExt e $\sqsubseteq_T$ *Q afterExt e*›
⟨*proof*⟩

lemma *mono-AfterExt-F* :
‹*P* $\sqsubseteq_F$ *Q* ⟹ *ev e* ∉ *ready-set P* ∨ *ev e* ∈ *ready-set Q* ⟹
P afterExt ev e $\sqsubseteq_F$ *Q afterExt ev e*›
⟨*proof*⟩

lemma *mono-AfterExt-D* : ‹*P* $\sqsubseteq_D$ *Q* ⟹ *P afterExt e* $\sqsubseteq_D$ *Q afterExt e*›
⟨*proof*⟩

lemma *mono-AfterExt-FD* :
‹*P* $\sqsubseteq_{FD}$ *Q* ⟹ *e* ∉ *ready-set P* ∨ *e* ∈ *ready-set Q* ⟹
P afterExt e $\sqsubseteq_{FD}$ *Q afterExt e*›
⟨*proof*⟩

lemma *mono-AfterExt-DT* : ‹*P* $\sqsubseteq_{DT}$ *Q* ⟹ *P afterExt e* $\sqsubseteq_{DT}$ *Q afterExt e*›
⟨*proof*⟩

5.1.4 Behaviour of *AfterExt* with *STOP*, *SKIP* and ⊥

lemma *AfterExt-STOP*: ‹*STOP afterExt e* = *STOP*›
⟨*proof*⟩

lemma *AfterExt-is-STOP-iff*:
‹*P afterExt e* = *STOP* ⟷ *P* ≠ ⊥ ∧ (∀ *s*. *e* # *s* ∈ $\mathcal{T}$ *P* ⟶ *s* = [])›
⟨*proof*⟩

lemma *AfterExt-SKIP*: ‹*SKIP afterExt e* = *STOP*›
⟨*proof*⟩

lemma *AfterExt-BOT* : ‹⊥ *afterExt e* = ⊥›
⟨*proof*⟩

lemma *AfterExt-is-BOT-iff*: ‹*P afterExt e* = ⊥ ⟷ [*e*] ∈ $\mathcal{D}$ *P*›
⟨*proof*⟩

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

Here again, we lose determinism.

lemma *AfterExt-Mprefix-is-AfterExt-Mndetprefix*:
‹(□*a* ∈ *A* → *P a*) *afterExt e* = (⊓*a* ∈ *A* → *P a*) *afterExt e*›
⟨*proof*⟩

lemma *AfterExt-Det-is-AfterExt-Ndet*: ‹*P* □ *Q afterExt e* = *P* ⊓ *Q afterExt e*›
⟨*proof*⟩

lemma *AfterExt-Ndet*:
‹*P* ⊓ *Q afterExt e* = (*if e* ∉ *ready-set P* ∪ *ready-set Q then STOP*
else case e of ev x ⇒ *if ev x* ∈ *ready-set P* ∩ *ready-set Q*
then (*P afterExt ev x*) ⊓ (*Q afterExt ev x*)
else if ev x ∈ *ready-set P*
then P afterExt ev x
else Q afterExt ev x
| ✓ ⇒ *if P* = ⊥ ∨ *Q* = ⊥ *then* ⊥ *else STOP*)›
⟨*proof*⟩

lemma *AfterExt-Mprefix*:
‹(□ *a* ∈ *A* → *P a*) *afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then P x else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

corollary *AfterExt-prefix*:
‹(*a* → *P*) *afterExt e* =
(*case e of ev x* ⇒ *if x* = *a then P else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

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*: ‹*Renaming P f* = ⊥ ⟷ *P* = ⊥›
⟨*proof*⟩

lemma *Renaming-is-STOP-iff*: ‹*Renaming P f* = *STOP* ⟷ *P* = *STOP*›
⟨*proof*⟩

lemma *AfterExt-Renaming*:
‹*Renaming P f afterExt e* =
(*if P* = ⊥ *then* ⊥ *else*
⊓*a* ∈ {*a*. *ev a* ∈ *ready-set P* ∧ *ev* (*f a*) = *e*}. *Renaming* (*P afterExt ev a*) *f*)›
⟨*proof*⟩

lemma *Seq-is-BOT-iff*: ‹*P* ; *Q* = ⊥ ⟷ *P* = ⊥ ∨ ([✓] ∈ *T P* ∧ *Q* = ⊥)›
⟨*proof*⟩

lemma *AfterExt-Seq*:
‹(*P* ; *Q*) *afterExt e* =
(*if e* ∉ *ready-set P* ∧ *e* ∉ *ready-set Q then STOP*
else if e ∉ *ready-set Q then P afterExt e* ; *Q*
else if e ∉ *ready-set P then if* ✓ ∈ *ready-set P then Q afterExt e else STOP*
else if ✓ ∈ *ready-set P then* (*P afterExt e* ; *Q*) ⊓ (*Q afterExt e*)
else P afterExt e ; *Q*)›
⟨*proof*⟩

theorem *AfterExt-Sync*:
‹(*P* ⟦*S*⟧ *Q*) *afterExt e* =
(*if P* = ⊥ ∨ *Q* = ⊥ *then* ⊥
else case e of ✓ ⇒ *STOP*
| *ev x* ⇒ *if e* ∈ *ready-set P* ∧ *e* ∈ *ready-set Q*
then if x ∈ *S*
then P afterExt e ⟦*S*⟧ *Q afterExt e*
else (*P afterExt e* ⟦*S*⟧ *Q*) ⊓ (*P* ⟦*S*⟧ *Q afterExt e*)
else if e ∈ *ready-set P*
then if x ∈ *S then STOP else P afterExt e* ⟦*S*⟧ *Q*
else if e ∈ *ready-set Q*
then if x ∈ *S then STOP else P* ⟦*S*⟧ *Q afterExt e*
else STOP)›
⟨*proof*⟩

lemma *Hiding-FD-Hiding-AfterExt-if-ready-inside*:
‹*e* ∈ *B* ⟹ (*P* \ *B*) ⊑$_{FD}$ (*P afterExt ev e* \ *B*)›
and *AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin*:
‹*e* ∉ *B* ⟹ (*P* \ *B*) *afterExt ev e* ⊑$_{FD}$ (*P afterExt ev e* \ *B*)›
if *ready*: ‹*ev e* ∈ *ready-set P*›

⟨*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*:
‹*ev e* ∈ *ready-set P* ⟹ *e* ∈ *B* ⟹ (*P* \ *B*) ⊑$_{DT}$ (*P afterExt ev e* \ *B*)›
and *AfterExt-Hiding-DT-Hiding-AfterExt-if-ready-notin*:
‹*ev e* ∈ *ready-set P* ⟹ *e* ∉ *B* ⟹
(*P* \ *B*) *afterExt ev e* ⊑$_{DT}$ (*P afterExt ev e* \ *B*)›
⟨*proof*⟩

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

lemma *AfterExt-MultiDet-is-AfterExt-MultiNdet*:
‹*finite A* ⟹ (□ *a* ∈ *A*. *P a*) *afterExt e* = (⨅ *a* ∈ *A*. *P a*) *afterExt e*›
⟨*proof*⟩

lemma *AfterExt-GlobalNdet*:
‹(⊓ *a* ∈ *A*. *P a*) *afterExt e* = (*if e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⊓ *a* ∈ {*a* ∈ *A*. *e* ∈ *ready-set* (*P a*)}. *P a*) *afterExt e*)›
and *AfterExt-MultiNdet*:
‹*finite A* ⟹ (⨅ *a* ∈ *A*. *P a*) *afterExt e* =
(*if e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⨅ *a* ∈ {*a* ∈ *A*. *e* ∈ *ready-set* (*P a*)}. *P a*) *afterExt e*)›
and *AfterExt-MultiDet*:
‹*finite A* ⟹ (□ *a* ∈ *A*. *P a*) *afterExt e* =
(*if e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⨅ *a* ∈ {*a* ∈ *A*. *e* ∈ *ready-set* (*P a*)}. *P a*) *afterExt e*)›
⟨*proof*⟩

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

lemma *AfterExt-Sliding-is-AfterExt-Ndet*:
‹*P* ▷ *Q afterExt e* = *P* ⊓ *Q afterExt e*›

⟨*proof*⟩

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

lemma *AfterExt-Throw*:
‹(*P* ⊖ *a* ∈ *A*. *Q a*) *afterExt e* =
(*if P* = ⊥ *then* ⊥
else case e of ✓ ⇒ *STOP*
| *ev x* ⇒ *if ev x* ∈ *ready-set P then if x* ∈ *A then Q x*
else (*P after x*) ⊖ *a* ∈ *A*. *Q a else STOP*)›
⟨*proof*⟩

lemma *AfterExt-Interrupt*:
‹(*P* △ *Q*) *afterExt e* =
(*if P* = ⊥ ∨ *Q* = ⊥ *then* ⊥
else case e of ✓ ⇒ *STOP*
| *ev x* ⇒ *if ev x* ∈ *ready-set P* ∧ *ev x* ∈ *ready-set Q*
then (*Q after x*) ⊓ (*P after x* △ *Q*)
else if ev x ∈ *ready-set P* ∧ *ev x* ∉ *ready-set Q*
then P after x △ *Q*
else if ev x ∉ *ready-set P* ∧ *ev x* ∈ *ready-set Q*
then Q after x
else STOP)›
⟨*proof*⟩

5.1.8 Behaviour of *AfterExt* with Reference Processes

lemma *AfterExt-DF*:
‹*DF A afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then DF A else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-DF*$_{SKIP}$:
‹*DF*$_{SKIP}$ *A afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then DF*$_{SKIP}$ *A else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-RUN*:
‹*RUN A afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then RUN A else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-CHAOS*:
‹*CHAOS A afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then CHAOS A else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-CHAOS$_{SKIP}$*:
‹*CHAOS$_{SKIP}$ A afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then CHAOS$_{SKIP}$ A else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *DF-FD-AfterExt*:
‹*DF A* ⊑$_{FD}$ *P* ⟹ *e* ∈ *ready-set P* ⟹ *DF A* ⊑$_{FD}$ *P afterExt e*›
⟨*proof*⟩

lemma *DF$_{SKIP}$-FD-AfterExt*:
‹*DF$_{SKIP}$ A* ⊑$_{FD}$ *P* ⟹ *ev e* ∈ *ready-set P* ⟹ *DF$_{SKIP}$ A* ⊑$_{FD}$ *P afterExt ev e*›
⟨*proof*⟩

lemma *deadlock-free-AfterExt*:
‹*deadlock-free P* ⟹ *deadlock-free* (*P afterExt e*) ⟷
(*if e* ∈ *ready-set P* ∧ *e* ≠ ✓ *then True else False*)›
⟨*proof*⟩

lemma *deadlock-free$_{SKIP}$-AfterExt*:
‹*deadlock-free$_{SKIP}$ P* ⟹ *deadlock-free$_{SKIP}$* (*P afterExt e*) ⟷
(*if e* ∈ *ready-set P* ∧ *e* ≠ ✓ *then True else False*)›
⟨*proof*⟩

end

5.2 The AfterTrace Operator

theory *AfterTrace*
imports *AfterExt HOL−CSPM.DeadlockResults*
begin

5.2.1 Definition

We just defined *P afterExt e* for *P*::*'α process* and *e*::*'α event*. Since a trace of a *P* is just an *'α event list*, the following inductive definition is natural.

fun *AfterTrace* :: ‹*'α process* ⇒ *'α trace* ⇒ *'α process*› (**infixl** ‹*afterTrace*› 77)
where ‹*P afterTrace* [] = *P*›
| ‹*P afterTrace* (*e* # *s*) = *P afterExt e afterTrace s*›

We can also induct backward.

lemma *AfterTrace-append*: ‹*P afterTrace* (*s* @ *t*) = *P afterTrace s afterTrace t*›

⟨*proof*⟩

lemma *AfterTrace-snoc* : ‹*P afterTrace* (*s* @ [*e*]) = *P afterTrace s afterExt e*›
⟨*proof*⟩

We have some immediate properties.

lemma *AfterTrace-BOT* : ‹⊥ *afterTrace s* = ⊥›
⟨*proof*⟩

lemma *AfterTrace-STOP* : ‹*STOP afterTrace s* = *STOP*›
⟨*proof*⟩

lemma *AfterTrace-SKIP* : ‹*SKIP afterTrace s* = (*if s* = [] *then SKIP else STOP*)›
⟨*proof*⟩

5.2.2 Projections

lemma *F-AfterTrace* : ‹(*s* @ *t*, *X*) ∈ ℱ *P* ⟹ (*t*, *X*) ∈ ℱ (*P afterTrace s*)›
⟨*proof*⟩

lemma *D-AfterTrace* : ‹*s* @ *t* ∈ 𝒟 *P* ⟹ *t* ∈ 𝒟 (*P afterTrace s*)›
⟨*proof*⟩

lemma *T-AfterTrace* : ‹*s* @ *t* ∈ 𝒯 *P* ⟹ *t* ∈ 𝒯 (*P afterTrace s*)›
⟨*proof*⟩

corollary *ready-set-AfterTrace* :
‹*s* @ *e* # *t* ∈ 𝒯 *P* ⟹ *e* ∈ *ready-set* (*P afterTrace s*)›
⟨*proof*⟩

corollary *F-imp-R-AfterTrace*: ‹(*s*, *X*) ∈ ℱ *P* ⟹ *X* ∈ ℛ (*P afterTrace s*)›
⟨*proof*⟩

corollary *D-imp-AfterTrace-is-BOT*: ‹*s* ∈ 𝒟 *P* ⟹ *P afterTrace s* = ⊥›
⟨*proof*⟩

5.2.3 Monotony

lemma *mono-AfterTrace* : ‹*P* ⊑ *Q* ⟹ *P afterTrace s* ⊑ *Q afterTrace s*›
⟨*proof*⟩

lemma *mono-AfterTrace-T* :
‹*P* ⊑$_T$ *Q* ⟹ *s* ∈ 𝒯 *Q* ⟹ *tickFree s* ⟹ *P afterTrace s* ⊑$_T$ *Q afterTrace s*›
⟨*proof*⟩

lemma *mono-AfterTrace-F* :
‹*P* ⊑$_F$ *Q* ⟹ *s* ∈ 𝒯 *Q* ⟹ *tickFree s* ⟹ *P afterTrace s* ⊑$_F$ *Q afterTrace s*›
⟨*proof*⟩

lemma *mono-AfterTrace-D* : ‹$P \sqsubseteq_D Q \Longrightarrow P$ *afterTrace* $s \sqsubseteq_D Q$ *afterTrace* s›
⟨*proof*⟩

lemma *mono-AfterTrace-FD* :
‹$P \sqsubseteq_{FD} Q \Longrightarrow s \in \mathcal{T}\ Q \Longrightarrow P$ *afterTrace* $s \sqsubseteq_{FD} Q$ *afterTrace* s›
⟨*proof*⟩

lemma *mono-AfterTrace-DT* :
‹$P \sqsubseteq_{DT} Q \Longrightarrow P$ *afterTrace* $s \sqsubseteq_{DT} Q$ *afterTrace* s›
⟨*proof*⟩

5.2.4 Another Definition of *events-of*

inductive *reachable-event* :: ‹$'\alpha$ *process* $\Rightarrow$ $'\alpha \Rightarrow$ *bool*›
where ‹*ev* $e \in$ *ready-set* $P \Longrightarrow$ *reachable-event* P e›
| ‹*reachable-event* $(P$ *after* $f)$ $e \Longrightarrow$ *reachable-event* P e›

lemma *events-of-iff-reachable-event*: ‹$e \in$ *events-of* $P \longleftrightarrow$ *reachable-event* P e›
⟨*proof*⟩

lemma *reachable-event-BOT*: ‹*reachable-event* $\bot$ e›
⟨*proof*⟩

lemma *not-reachable-event-STOP*: ‹$\neg$ *reachable-event STOP* e›
⟨*proof*⟩

lemma *reachable-tick-SKIP*: ‹$\neg$ *reachable-event SKIP* ✓›
⟨*proof*⟩

lemma *reachable-event-iff-in-ready-set-AfterTrace*:
‹*reachable-event* P $e \longleftrightarrow e \in \{e|\ e\ s.\ ev\ e \in$ *ready-set* $(P$ *afterTrace* $s)\}$›
⟨*proof*⟩

5.2.5 Characterizations for Deadlock Freeness

lemma *deadlock-free-AfterExt-characterization*:
‹*deadlock-free* $P \longleftrightarrow$ *range ev* $\notin \mathcal{R}\ P \wedge$
$(\forall\, e \in$ *ready-set* $P.$ *deadlock-free* $(P$ *afterExt* $e))$›
(**is** ‹*deadlock-free* $P \longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

lemma *deadlock-free*$_{SKIP}$*-AfterExt-characterization*:
‹*deadlock-free*$_{SKIP}$ $P \longleftrightarrow$
UNIV $\notin \mathcal{R}\ P \wedge (\forall\, e \in$ *ready-set* $P - \{$✓$\}.$ *deadlock-free*$_{SKIP}$ $(P$ *afterExt* $e))$›

(**is** ‹*deadlock-free*$_{SKIP}$ $P \longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

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 ✓): $P \rightsquigarrow e\ P'$.

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 τ-*trans* :: $\langle {'\alpha}\ process \Rightarrow {'\alpha}\ process \Rightarrow bool\rangle$ (**infixl** $\langle\rightsquigarrow_\tau\rangle$ *50*)
  where τ-*trans-eq* : $\langle P \rightsquigarrow_\tau P\rangle$
  | τ-*trans-NdetL* : $\langle P \sqcap Q \rightsquigarrow_\tau P\rangle$

| τ-*trans-NdetR* : $\langle P \sqcap Q \rightsquigarrow_\tau Q\rangle$

— We can obtain the same inductive predicate by removing τ-*trans-eq* and τ-*trans-NdetR* clauses (because of ($\sqcap$) properties).

With this definition, we immediately show that the τ transition is the FD-refinement ($\sqsubseteq_{FD}$).

lemma τ-*trans-is-FD*: $\langle(\rightsquigarrow_\tau) = (\sqsubseteq_{FD})\rangle$
⟨*proof*⟩

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 $P \rightsquigarrow e\ P' \Longrightarrow e \in$ *ready-set* P.

Moreover, we want the event transitions to absorb the τ transitions.

Finally, when $e \in$ *ready-set* P, we want to have $P \rightsquigarrow e\ P$ *afterExt* e.

This brings us to the following inductive definition:

inductive *event-trans-prem* :: $\langle'\alpha$ *process* $\Rightarrow$ $'\alpha$ *event* $\Rightarrow$ $'\alpha$ *process* $\Rightarrow$ *bool*$\rangle$
where
τ-*left-absorb* : $\langle[\![e \in$ *ready-set* P'; $P \rightsquigarrow_\tau P'$; *event-trans-prem* $P'\ e\ P'']\!] \Longrightarrow$ *event-trans-prem* $P\ e\ P''\rangle$
| τ-*right-absorb* : $\langle[\![e \in$ *ready-set* P; *event-trans-prem* $P\ e\ P'$; $P' \rightsquigarrow_\tau P'']\!] \Longrightarrow$ *event-trans-prem* $P\ e\ P''\rangle$
| *ready-trans-to-AfterExt* : $\langle e \in$ *ready-set* $P \Longrightarrow$ *event-trans-prem* $P\ e$ (P *afterExt* e)$\rangle$

abbreviation *event-trans* :: $\langle'\alpha$ *process* $\Rightarrow$ $'\alpha$ *event* $\Rightarrow$ $'\alpha$ *process* $\Rightarrow$ *bool*$\rangle$
($\langle$- $\rightsquigarrow$-/ -$\rangle$ [50, 3, 51] 50)
where $\langle P \rightsquigarrow e\ P' \equiv e \in$ *ready-set* $P \wedge$ *event-trans-prem* $P\ e\ P'\rangle$

We immediately show that this event transition definition is equivalent to the following:

lemma *startable-imp-ev-trans-is-startable-and-FD-After*:
$\langle(P \rightsquigarrow e\ P') \longleftrightarrow e \in$ *ready-set* $P \wedge P$ *afterExt* $e \rightsquigarrow_\tau P'\rangle$
⟨*proof*⟩

With these two results, we are encouraged in the following theories to define:

- $P \rightsquigarrow_\tau Q \equiv P \sqsubseteq_{FD} Q$
- $P \rightsquigarrow e\ Q \equiv e \in$ *ready-set* $P \wedge P$ *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 τ-*trans* :: ‹['α *process*, 'α *process*] ⇒ *bool*› (**infixl** ‹$\rightsquigarrow_\tau$› *50*)
assumes τ-*trans-NdetL*: ‹$P \sqcap Q \rightsquigarrow_\tau P$›
and τ-*trans-transitivity*: ‹$P \rightsquigarrow_\tau Q \Longrightarrow Q \rightsquigarrow_\tau R \Longrightarrow P \rightsquigarrow_\tau R$›
and τ-*trans-anti-mono-ready-set*: ‹$P \rightsquigarrow_\tau Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
and τ-*trans-mono-AfterExt*:
‹$e \in$ *ready-set* $Q \Longrightarrow P \rightsquigarrow_\tau Q \Longrightarrow P$ *afterExt* $e \rightsquigarrow_\tau Q$ *afterExt* e›
begin

This locale needs to be instantiated with a binary relation ($\rightsquigarrow_\tau$) 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* :: ‹['α *process*, 'α *event*, 'α *process*] ⇒ *bool*›
(‹-/ ⇝-/ -› [*50*, *3*, *51*] *50*)

where $\langle P \rightsquigarrow e\ Q \equiv e \in \mathit{ready\text{-}set}\ P \wedge P\ \mathit{afterExt}\ e \rightsquigarrow_\tau Q\rangle$

From idempotence, commutativity and $\bot$ absorbance of $(\sqcap)$, we get the following free of charge.

lemma *τ-trans-eq*: $\langle P \rightsquigarrow_\tau P\rangle$
and *τ-trans-NdetR*: $\langle P \sqcap Q \rightsquigarrow_\tau Q\rangle$
and *BOT-τ-trans-anything*: $\langle \bot \rightsquigarrow_\tau P\rangle$
and *BOT-event-trans-anything*: $\langle \bot \rightsquigarrow e\ P\rangle$
⟨*proof*⟩

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

lemma *event-trans-τ-trans*: $\langle P \rightsquigarrow e\ P' \Longrightarrow P' \rightsquigarrow_\tau P'' \Longrightarrow P \rightsquigarrow e\ P''\rangle$
⟨*proof*⟩

lemma *τ-trans-event-trans*: $\langle P \rightsquigarrow_\tau P' \Longrightarrow P' \rightsquigarrow e\ P'' \Longrightarrow P \rightsquigarrow e\ P''\rangle$
⟨*proof*⟩

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

inductive *trace-trans* :: $\langle 'a\ \mathit{process} \Rightarrow 'a\ \mathit{trace} \Rightarrow 'a\ \mathit{process} \Rightarrow \mathit{bool}\rangle$
$(\langle \text{-}/ \rightsquigarrow^* \text{-}/ \text{-}\rangle\ [50, 3, 51]\ 50)$
where *trace-τ-trans* : $\langle P \rightsquigarrow_\tau P' \Longrightarrow P \rightsquigarrow^* [] P'\rangle$
| *trace-tick-trans* : $\langle P \rightsquigarrow \checkmark\ P' \Longrightarrow P \rightsquigarrow^* [\checkmark] P'\rangle$
| *trace-Cons-ev-trans* :
$\langle P \rightsquigarrow (ev\ e)\ P' \Longrightarrow P' \rightsquigarrow^* s\ P'' \Longrightarrow P \rightsquigarrow^* (ev\ e) \#\ s\ P''\rangle$

lemma *trace-trans-τ-trans*: $\langle P \rightsquigarrow^* s\ P' \Longrightarrow P' \rightsquigarrow_\tau P'' \Longrightarrow P \rightsquigarrow^* s\ P''\rangle$
⟨*proof*⟩

lemma *τ-trans-trace-trans*: $\langle P \rightsquigarrow_\tau P' \Longrightarrow P' \rightsquigarrow^* s\ P'' \Longrightarrow P \rightsquigarrow^* s\ P''\rangle$
⟨*proof*⟩

lemma *BOT-trace-trans-anything* : $\langle \mathit{front\text{-}tickFree}\ s \Longrightarrow \bot \rightsquigarrow^* s\ P\rangle$
⟨*proof*⟩

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*:
$\langle P \rightsquigarrow^* s\ Q \Longrightarrow X \in \mathcal{R}\ Q \Longrightarrow (s, X) \in \mathcal{F}\ P\rangle$
if $\langle \forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q\rangle$
⟨*proof*⟩

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

lemma *trace-trans-BOT-imp-D-if-τ-trans-imp-leD*: ‹$P \rightsquigarrow^* s \perp \Longrightarrow s \in \mathcal{D}\ P$›
  if ‹$\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_D Q$›
⟨*proof*⟩

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

lemma *trace-trans-iff* :
  ‹$P \rightsquigarrow^* [] Q \longleftrightarrow P \rightsquigarrow_\tau Q$›
  ‹$P \rightsquigarrow^* [✓] Q \longleftrightarrow P \rightsquigarrow✓ Q$›
  ‹$P \rightsquigarrow^* (ev\ e) \# s\ Q' \longleftrightarrow (\exists Q.\ P \rightsquigarrow (ev\ e)\ Q \wedge Q \rightsquigarrow^* s\ Q')$›
  ‹*tickFree* $s \Longrightarrow (P \rightsquigarrow^* s @ [f]\ Q') \longleftrightarrow (\exists Q.\ P \rightsquigarrow^* s\ Q \wedge Q \rightsquigarrow f\ Q')$›
  ‹*front-tickFree* $(s @ t) \Longrightarrow (P \rightsquigarrow^* s @ t\ Q') \longleftrightarrow (\exists Q.\ P \rightsquigarrow^* s\ Q \wedge Q \rightsquigarrow^* t\ Q')$›
⟨*proof*⟩

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

theorem *T-imp-trace-trans-iff-AfterTrace-τ-trans* :
  ‹$s \in \mathcal{T}\ P \Longrightarrow (P \rightsquigarrow^* s\ Q) \longleftrightarrow P$ *afterTrace* $s \rightsquigarrow_\tau Q$›
⟨*proof*⟩

As corollaries we obtain the reciprocal results of
$[\![\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q;\ ?P \rightsquigarrow^* ?s\ ?Q;\ ?X \in \mathcal{R}\ ?Q]\!] \Longrightarrow (?s,\ ?X) \in \mathcal{F}\ ?P$
$[\![\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q;\ ?P \rightsquigarrow^* ?s\ ?Q]\!] \Longrightarrow ?s \in \mathcal{T}\ ?P$
$[\![\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_D Q;\ ?P \rightsquigarrow^* ?s \perp]\!] \Longrightarrow ?s \in \mathcal{D}\ ?P$

lemma *F-imp-exists-trace-trans*: ‹$(s, X) \in \mathcal{F}\ P \Longrightarrow \exists Q.\ (P \rightsquigarrow^* s\ Q) \wedge X \in \mathcal{R}\ Q$›
  ⟨*proof*⟩

lemma *T-imp-exists-trace-trans*: ‹$s \in \mathcal{T}\ P \Longrightarrow \exists Q.\ P \rightsquigarrow^* s\ Q$›
  ⟨*proof*⟩

lemma *D-imp-trace-trans-BOT*: ‹$s \in \mathcal{D}\ P \Longrightarrow P \rightsquigarrow^* s \perp$›
  ⟨*proof*⟩

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

lemma *τ-trans-imp-leT-imp-STOP-trace-trans-iff*:
  ‹$\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_T Q \Longrightarrow STOP \rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = STOP$›
  ⟨*proof*⟩

lemma *τ-trans-imp-leF-imp-SKIP-trace-trans-iff*:
  ‹$\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q \Longrightarrow$
   $SKIP \rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [✓] \wedge P = STOP$›
  ⟨*proof*⟩

lemma *τ-trans-imp-leT-imp-trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_T$ *Q* ⟹ *P* ⇝* *s Q* ⟹
ready-set Q ⊆ *ready-set* (*P afterTrace s*)›
⟨*proof*⟩

lemma *τ-trans-imp-leT-imp-trace-trans-imp-ready-set*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_T$ *Q* ⟹ *P* ⇝*(*s* @ *e* # *t*) *Q* ⟹
e ∈ *ready-set* (*P afterTrace s*)›
⟨*proof*⟩

lemma *trace-trans-iff-T-and-AfterTrace-τ-trans-if-τ-trans-imp-leT*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_T$ *Q* ⟹
(*P* ⇝* *s Q*) ⟷ *s* ∈ $\mathcal{T}$ *P* ∧ *P afterTrace s* ⇝$_\tau$ *Q*›
⟨*proof*⟩

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* ⇝$_\tau$ *Q* is.

lemma *SKIP-trans-tick*: ‹*SKIP* ⇝✓ *STOP*›
⟨*proof*⟩

lemma *tick-trans-imp-BOT-L-or-STOP-R*: ‹*P* ⇝✓ *Q* ⟹ *P* = ⊥ ∨ *Q* = *STOP*›
⟨*proof*⟩

lemma *STOP-trace-trans-iff* : ‹*STOP* ⇝* *s P* ⟷ *s* = [] ∧ *P* = *STOP*›
⟨*proof*⟩

lemma *ready-tick-imp-τ-trans-SKIP*: ‹*P* ⇝$_\tau$ *SKIP*› **if** ‹✓ ∈ *ready-set P*›
⟨*proof*⟩

lemma *exists-tick-trans-is-ready-tick*: ‹(∃ *P′*. *P* ⇝✓ *P′*) ⟷ ✓ ∈ *ready-set P*›
⟨*proof*⟩

lemma *tick-trans-iff* : ‹*P* ⇝✓ *P′* ⟷ *P* = ⊥ ∨ *P* ⇝$_\tau$ *SKIP* ∧ *P′* = *STOP*›
⟨*proof*⟩

lemma *SKIP-cant-ev-trans*: ‹¬ *SKIP* ⇝(*ev e*) *STOP*›
⟨*proof*⟩

lemma *STOP-cant-event-trans*: ‹$\neg$ *STOP* $\rightsquigarrow e$ *P*›
⟨*proof*⟩

lemma *ev-trans-Mprefix*: ‹$e \in A \Longrightarrow \Box a \in A \rightarrow P\ a \rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma *ev-trans-Mndetprefix*: ‹$e \in A \Longrightarrow \sqcap a \in A \rightarrow P\ a \rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma *ev-trans-prefix*: ‹$e \rightarrow P \rightsquigarrow (ev\ e)\ P$›
⟨*proof*⟩

lemma τ*-trans-MultiNdet*: ‹*finite* $A \Longrightarrow x \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a) \rightsquigarrow_\tau P\ x$›
⟨*proof*⟩

lemma τ*-trans-GlobalNdet*: ‹$(\sqcap a \in A.\ P\ a) \rightsquigarrow_\tau P\ e$› **if** ‹$e \in A$›
⟨*proof*⟩

lemma *fix-point-*τ*-trans*: ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P \rightsquigarrow_\tau f\ P$›
⟨*proof*⟩

lemma *event-trans-DetL*: ‹$P \rightsquigarrow e\ P' \Longrightarrow P \mathbin{\Box} Q \rightsquigarrow e\ P'$›
⟨*proof*⟩

lemma *event-trans-DetR*: ‹$Q \rightsquigarrow e\ Q' \Longrightarrow P \mathbin{\Box} Q \rightsquigarrow e\ Q'$›
⟨*proof*⟩

lemma *event-trans-MultiDet*:
‹*finite* $A \Longrightarrow a \in A \Longrightarrow P\ a \rightsquigarrow e\ Q \Longrightarrow (\Box a \in A.\ P\ a) \rightsquigarrow e\ Q$›
⟨*proof*⟩

lemma *Sliding-event-transL*: ‹$P \rightsquigarrow e\ P' \Longrightarrow P \rhd Q \rightsquigarrow e\ P'$›
⟨*proof*⟩

lemma *Sliding-*τ*-transR*: ‹$P \rhd Q \rightsquigarrow_\tau Q$›
⟨*proof*⟩

lemma ‹$\exists\, P\ P'\ Q.\ P \rightsquigarrow\!\!\checkmark\ P' \wedge \neg\ P\ ;;\ Q \rightsquigarrow\!\!\checkmark\ P'\ ;;\ Q$›
⟨*proof*⟩

lemma *ev-trans-SeqR*:
‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q \rightsquigarrow(ev\ e)\ Q' \Longrightarrow P\ ;;\ Q \rightsquigarrow(ev\ e)\ Q'$›
⟨*proof*⟩

lemma ‹*SKIP* $[\![S]\!]$ *SKIP* $\rightsquigarrow\!\!\checkmark$ *STOP*›
⟨*proof*⟩

lemma ‹*SKIP* $[\![S]\!]$ *SKIP* $\rightsquigarrow_\tau$ *SKIP*›
⟨*proof*⟩

lemma *tick-trans-Hiding*: ‹$P \setminus B \rightsquigarrow\!\!\checkmark$ *STOP*› **if** ‹$P \rightsquigarrow\!\!\checkmark\ P'$›
⟨*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 of the locale.

lemma ‹*Renaming* $P\ f \rightsquigarrow\!\!\checkmark$ *STOP*› **if** ‹$P \rightsquigarrow\!\!\checkmark\ P'$›
⟨*proof*⟩

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 *τ-trans* :: ‹'α *process* ⇒ 'α *process* ⇒ *bool*› (**infixl** ‹$_{FD}\!\rightsquigarrow_{\tau}$› *50*)
where ‹*P* $_{FD}\!\rightsquigarrow_{\tau}$ *Q* ≡ *P* $\sqsubseteq_{FD}$ *Q*›

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

interpretation *OpSemGeneric* ‹($_{FD}\!\rightsquigarrow_{\tau}$)›
⟨*proof*⟩

notation *event-trans* (‹-/ $_{FD}\!\rightsquigarrow$-/ -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹-/ $_{FD}\!\rightsquigarrow^{*}$-/ -› [*50*, *3*, *51*] *50*)

lemma ‹*P* $_{FD}\!\rightsquigarrow$ *e* *P'* ⟹ *P'* $_{FD}\!\rightsquigarrow_{\tau}$ *P''* ⟹ *P* $_{FD}\!\rightsquigarrow$ *e* *P''*›
‹*P* $_{FD}\!\rightsquigarrow_{\tau}$ *P'* ⟹ *P'* $_{FD}\!\rightsquigarrow$ *e* *P''* ⟹ *P* $_{FD}\!\rightsquigarrow$ *e* *P''*›
⟨*proof*⟩

8.1 Operational Semantics Laws

SKIP law

lemma ‹*SKIP* $_{FD}\!\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

$e \rightarrow P$ laws

lemma ‹*e* ∈ *A* ⟹ □*a* ∈ *A* → *P a* $_{FD}\!\rightsquigarrow$(*ev e*) (*P e*)›
⟨*proof*⟩

lemma ‹$e \in A \Longrightarrow \sqcap a \in A \rightarrow P\ a\ {}_{FD}\!\rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \rightarrow P\ {}_{FD}\!\rightsquigarrow\ (ev\ e)\ P$›
⟨*proof*⟩

$P \sqcap Q$ laws

lemma ‹$P \sqcap Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P$›
and ‹$P \sqcap Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$a \in A \Longrightarrow (\sqcap a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

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

lemma ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P\ {}_{FD}\!\rightsquigarrow_{\tau}\ f\ P$›
⟨*proof*⟩

$P \Box Q$ laws

lemma τ-*trans-DetL*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Box Q$›
and τ-*trans-DetR*: ‹$Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P \Box Q'$›
⟨*proof*⟩

lemma τ-*trans-MultiDet*:
‹*finite* $A \Longrightarrow \forall\, a \in A.\ P\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ a \Longrightarrow$
$(\Box a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow_{\tau}\ (\Box a \in A.\ P'\ a)$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow e\ P' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow e\ P'$›
and ‹$Q\ {}_{FD}\!\rightsquigarrow e\ Q' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow e\ Q'$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow P\ a\ {}_{FD}\!\rightsquigarrow e\ Q \Longrightarrow (\Box a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow e\ Q$›
⟨*proof*⟩

$P\ ;\ Q$ laws

lemma τ-*trans-SeqL*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ ;\ Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ ;\ Q$›
⟨*proof*⟩

lemma *ev-trans-SeqL*: ‹$P\ {}_{FD}\!\rightsquigarrow (ev\ e)\ P' \Longrightarrow P\ ;\ Q\ {}_{FD}\!\rightsquigarrow (ev\ e)\ P'\ ;\ Q$›

⟨*proof*⟩

lemma *τ-trans-SeqR*: ‹$\exists\, P'.\ P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P\ ;;\ Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q\ {}_{FD}\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P\ ;;\ Q\ {}_{FD}\!\rightsquigarrow(ev\ e)\ Q'$›

⟨*proof*⟩

$P \setminus B$ laws

lemma *τ-trans-Hiding*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

lemma *ev-trans-Hiding-notin*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \notin B \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \setminus B$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow\!\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Hiding-inside*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in B \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

Renaming $P\ f$ laws

lemma *τ-trans-Renaming*:
‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow$ *Renaming* $P\ f\ {}_{FD}\!\rightsquigarrow_{\tau}$ *Renaming* $P'\ f$›
⟨*proof*⟩

lemma *tick-trans-Renaming*: ‹$P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow$ *Renaming* $P\ f\ {}_{FD}\!\rightsquigarrow\!\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Renaming*:
‹$f\ a = b \Longrightarrow P\ {}_{FD}\!\rightsquigarrow(ev\ a)\ P' \Longrightarrow$ *Renaming* $P\ f\ {}_{FD}\!\rightsquigarrow(ev\ b)$ (*Renaming* $P'\ f$)›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{FD}\!\rightsquigarrow\!\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-RenamingF*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ a)\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{FD}\!\rightsquigarrow(ev\ b)\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

$P\ [\![S]\!]\ Q$ laws

lemma τ-*trans-SyncL*: $\langle P \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \ [\![S]\!] \ Q\rangle$
and τ-*trans-SyncR*: $\langle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ Q' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P \ [\![S]\!] \ Q'\rangle$
⟨*proof*⟩

lemma *ev-trans-SyncL*:
$\langle e \notin S \Longrightarrow P \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \ [\![S]\!] \ Q \ \rangle$
and *ev-trans-SyncR*:
$\langle e \notin S \Longrightarrow Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ Q' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P \ \ [\![S]\!] \ Q'\rangle$
⟨*proof*⟩

lemma *ev-trans-SyncLR*:
$\langle [\![e \in S; \ P \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P'; \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ Q']\!] \Longrightarrow$
$P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \ [\![S]\!] \ Q'\rangle$
⟨*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*: $\langle P \ {}_{FD}\!\rightsquigarrow\checkmark \ P' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP \ [\![S]\!] \ Q\rangle$
and *tick-trans-SyncR*: $\langle Q \ {}_{FD}\!\rightsquigarrow\checkmark \ Q' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P \ [\![S]\!] \ SKIP\rangle$
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \ [\![S]\!] \ SKIP \ {}_{FD}\!\rightsquigarrow\checkmark \ STOP\rangle$
⟨*proof*⟩

lemma τ-*trans-SKIP-Sync-SKIP*: $\langle SKIP \ [\![S]\!] \ SKIP \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP\rangle$
⟨*proof*⟩

$P \rhd Q$ laws

lemma *Sliding-*τ*-trans-left*: $\langle P \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \Longrightarrow P \rhd Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \rhd Q\rangle$
⟨*proof*⟩

lemma $\langle P \ {}_{FD}\!\rightsquigarrow e \ P' \Longrightarrow P \rhd Q \ {}_{FD}\!\rightsquigarrow e \ P'\rangle$
⟨*proof*⟩

lemma $\langle P \rhd Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ Q\rangle$
⟨*proof*⟩

$P \triangle Q$ laws

lemma *Interrupt-*τ*-trans-left*: $\langle P \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \Longrightarrow P \triangle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \triangle Q\rangle$
⟨*proof*⟩

lemma *Interrupt-*τ*-trans-right*: $\langle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ Q' \Longrightarrow P \triangle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P \triangle Q'\rangle$
⟨*proof*⟩

lemma *Interrupt-ev-trans-left*:
$\langle P \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \Longrightarrow P \triangle Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \triangle Q\rangle$
⟨*proof*⟩

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

⟨*proof*⟩

Throw $P\ A\ Q$ laws

lemma *Throw-τ-trans-left*:
‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ \Theta\ a \in A.\ Q\ a$›
⟨*proof*⟩

lemma *Throw-τ-trans-right*:
‹$\forall\, a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q'\ a \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ \Theta\ a \in A.\ Q'\ a$›
⟨*proof*⟩

lemma *Throw-event-trans-left*:
‹$P\ {}_{FD}\!\rightsquigarrow e\ P' \Longrightarrow e \notin ev\ `\ A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow e\ (P'\ \Theta\ a \in A.\ Q\ a)$›
⟨*proof*⟩

lemma *Throw-ev-trans-right*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow(ev\ e)\ (Q\ e)$›
⟨*proof*⟩

lemma ‹*front-tickFree* $s \Longrightarrow \bot\ {}_{FD}\!\rightsquigarrow^{*} s\ P$›
⟨*proof*⟩

8.2 Reality Checks

lemma ‹$STOP\ {}_{FD}\!\rightsquigarrow^{*} s\ P \longleftrightarrow s = [] \wedge P = STOP$›
⟨*proof*⟩

lemma *SKIP-trace-trans-iff* :
‹$SKIP\ {}_{FD}\!\rightsquigarrow^{*} s\ P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP$›
⟨*proof*⟩

lemma *F-iff-exists-trans* :
‹$(s,\ X) \in \mathcal{F}\ P \longleftrightarrow (\exists\, P'.\ (P\ {}_{FD}\!\rightsquigarrow^{*} s\ P') \wedge X \in \mathcal{R}\ P')$›
⟨*proof*⟩

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

lemma *D-iff-trace-trans-BOT*: ‹$s \in \mathcal{D}\ P \longleftrightarrow P\ {}_{FD}\!\rightsquigarrow^{*} s\ \bot$›
⟨*proof*⟩

8.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹$P\ {}_{FD}\!\rightsquigarrow^{*} s\ Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *trace-trans-imp-ready-set*:
‹$P\ {}_{FD}\!\rightsquigarrow^{*}(s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
‹$(P\ {}_{FD}\!\rightsquigarrow^{*} s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P$ *afterTrace* $s\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹*deadlock-free* $P \longleftrightarrow$ *DF UNIV* $\ {}_{FD}\!\rightsquigarrow_{\tau}\ P$›
⟨*proof*⟩

lemma ‹*deadlock-free*$_{SKIP}$ $P \longleftrightarrow DF_{SKIP}$ *UNIV* $\ {}_{FD}\!\rightsquigarrow_{\tau}\ P$›
⟨*proof*⟩

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

$$\frac{?P\ {}_{FD}\!\rightsquigarrow ?e\ ?P' \qquad ?P'\ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P''}{?P\ {}_{FD}\!\rightsquigarrow ?e\ ?P''}$$

$$\frac{?P\ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P' \qquad ?P'\ {}_{FD}\!\rightsquigarrow ?e\ ?P''}{?P\ {}_{FD}\!\rightsquigarrow ?e\ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP\ {}_{FD}\!\rightsquigarrow ✓\ STOP}$$

$e \rightarrow P$ **rules**

$$\frac{?e \in ?A}{\Box a \in ?A \rightarrow\ ?P\ a\ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P\ ?e} \qquad \frac{?e \in ?A}{\sqcap a \in ?A \rightarrow\ \ ?P\ a\ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P\ ?e}$$

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

($\sqcap$) **rules**

$$\frac{}{?P \sqcap ?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P} \qquad \frac{}{?P \sqcap ?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?Q}$$

$$\frac{?e \in ?A}{\sqcap\ a{\in}?A.\ \ ?P\ a \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P\ ?e}$$

$\mu\ x.\ f\ x$ **rule**

$$\frac{cont\ ?f \qquad ?P = (\mu\ x.\ ?f\ x)}{?P \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?f\ ?P}$$

($\Box$) **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P'}{?P \,\Box\, ?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P' \,\Box\, ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?Q'}{?P \,\Box\, ?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P \,\Box\, ?Q'}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ?e\ ?P'}{?P \,\Box\, ?Q \ {}_{FD}\!\rightsquigarrow ?e\ ?P'} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow ?e\ ?Q'}{?P \,\Box\, ?Q \ {}_{FD}\!\rightsquigarrow ?e\ ?Q'}$$

(;) **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P'}{?P\ ;\ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P'\ ;\ ?Q} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P'}{?P\ ;\ ?Q \ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P'\ ;\ ?Q}$$

$$\frac{\exists\, P'.\ ?P \ {}_{FD}\!\rightsquigarrow\checkmark\ P'}{?P\ ;\ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?Q}$$

Hiding **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P'}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P' \setminus ?B} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow\checkmark\ ?P'}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow\checkmark\ STOP}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \notin ?B}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P' \setminus ?B} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \in ?B}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P' \setminus ?B}$$

Renaming **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{Renaming \ ?P \ ?f \ {}_{FD}\!\rightsquigarrow_{\tau} \ Renaming \ ?P' \ ?f}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow\checkmark \ ?P'}{Renaming \ ?P \ ?f \ {}_{FD}\!\rightsquigarrow\checkmark \ STOP}$$

$$\frac{?f \ ?a = ?b \qquad ?P \ {}_{FD}\!\rightsquigarrow ev \ ?a \ ?P'}{Renaming \ ?P \ ?f \ {}_{FD}\!\rightsquigarrow ev \ ?b \ Renaming \ ?P' \ ?f}$$

Sync **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \ [\![?S]\!] \ ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \notin ?S \qquad ?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q}$$

$$\frac{?e \notin ?S \qquad ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \in ?S \qquad ?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q'}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow\checkmark \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP \ [\![?S]\!] \ ?Q}$$

$$\frac{?Q \ {}_{FD}\!\rightsquigarrow\checkmark \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ SKIP}$$

$$\frac{}{SKIP \ [\![?S]\!] \ SKIP \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP}$$

($\triangleright$) **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangleright ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \triangleright ?Q} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow ?e \ ?P'}{?P \triangleright ?Q \ {}_{FD}\!\rightsquigarrow ?e \ ?P'}$$

$$\frac{}{?P \triangleright ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q}$$

($\triangle$) **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangle ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \triangle ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \triangle ?Q'}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \ \triangle \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \ \triangle \ ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ \triangle \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}$$

Throw **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{\forall \, a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q' \ a}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \ \Theta \ a{\in}?A. \ ?Q' \ a}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ?e \ ?P' \qquad ?e \notin ev \ `\ ?A}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow ?e \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \in ?A}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q \ ?e}$$

end

Chapter 9

Trace Divergence Operational Semantics

theory *OpSemDT*
imports *OpSemGeneric HOL−Library.LaTeXsugar*
begin

The definition with ($\sqsubseteq_{FD}$) motivates us to try with other refinements.

abbreviation *τ-trans* :: ‹'*α process* ⇒ '*α process* ⇒ *bool*› (**infixl** ‹$_{DT}\!\rightsquigarrow_{\tau}$› *50*)
where ‹*P* $_{DT}\!\rightsquigarrow_{\tau}$ *Q* ≡ *P* $\sqsubseteq_{DT}$ *Q*›

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

interpretation *OpSemGeneric* ‹($_{DT}\!\rightsquigarrow_{\tau}$)›
⟨*proof*⟩

notation *event-trans* (‹-/ $_{DT}\!\rightsquigarrow$-/ -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹-/ $_{DT}\!\rightsquigarrow^{*}$-/ -› [*50*, *3*, *51*] *50*)

lemma ‹*P* $_{DT}\!\rightsquigarrow$ *e P'* ⟹ *P'* $_{DT}\!\rightsquigarrow_{\tau}$ *P''* ⟹ *P* $_{DT}\!\rightsquigarrow$ *e P''*›
‹*P* $_{DT}\!\rightsquigarrow_{\tau}$ *P'* ⟹ *P'* $_{DT}\!\rightsquigarrow$ *e P''* ⟹ *P* $_{DT}\!\rightsquigarrow$ *e P''*›
⟨*proof*⟩

9.1 Operational Semantics Laws

SKIP law

lemma ‹*SKIP* $_{DT}\!\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

$e \rightarrow P$ laws

lemma ‹*e* ∈ *A* ⟹ □*a* ∈ *A* → *P a* $_{DT}\!\rightsquigarrow$(*ev e*) (*P e*)›
⟨*proof*⟩

lemma ‹$e \in A \Longrightarrow \sqcap a \in A \rightarrow P\ a\ {}_{DT}\!\rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \rightarrow P\ {}_{DT}\!\rightsquigarrow\ (ev\ e)\ P$›
⟨*proof*⟩

$P \sqcap Q$ laws

lemma ‹$P \sqcap Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P$›
and ‹$P \sqcap Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$a \in A \Longrightarrow (\sqcap a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

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

lemma ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P\ {}_{DT}\!\rightsquigarrow_{\tau}\ f\ P$›
⟨*proof*⟩

$P \Box Q$ laws, more powerful

lemma τ-*trans-DetL*: ‹$P \Box Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P$›
and τ-*trans-DetR*: ‹$P \Box Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma τ-*trans-MultiDet*: ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\Box a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

P **;** Q laws

lemma τ-*trans-SeqL*: ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P$ **;** $Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P'$ **;** Q›
⟨*proof*⟩

lemma *ev-trans-SeqL*: ‹$P\ {}_{DT}\!\rightsquigarrow (ev\ e)\ P' \Longrightarrow P$ **;** $Q\ {}_{DT}\!\rightsquigarrow (ev\ e)\ P'$ **;** Q›

⟨*proof*⟩

lemma τ-*trans-SeqR*: ‹$\exists P'.\ P\ {}_{DT}\!\rightsquigarrow\checkmark\ P' \Longrightarrow P$ **;** $Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q\ {}_{DT}\!\rightsquigarrow (ev\ e)\ Q' \Longrightarrow P$ **;** $Q\ {}_{DT}\!\rightsquigarrow (ev\ e)\ Q'$›

⟨*proof*⟩

$P \setminus B$ laws

lemma τ-*trans-Hiding*: ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \setminus B\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \setminus B$›

⟨*proof*⟩

lemma *ev-trans-Hiding-notin*:
‹$P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \notin B \Longrightarrow P \setminus B\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \setminus B$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\!\rightsquigarrow\checkmark\ P' \Longrightarrow P \setminus B\ {}_{DT}\!\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Hiding-inside*:
‹$P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in B \Longrightarrow P \setminus B\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

Renaming P f laws

lemma *τ-trans-Renaming*:
‹$P\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \Longrightarrow Renaming\ P\ f\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ Renaming\ P'\ f$›
⟨*proof*⟩

lemma *tick-trans-Renaming*: ‹$P\ {}_{DT}\!\!\rightsquigarrow\checkmark\ P' \Longrightarrow Renaming\ P\ f\ {}_{DT}\!\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Renaming*:
‹$f\ a = b \Longrightarrow P\ {}_{DT}\!\!\rightsquigarrow(ev\ a)\ P' \Longrightarrow Renaming\ P\ f\ {}_{DT}\!\!\rightsquigarrow(ev\ b)\ (Renaming\ P'\ f)$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{DT}\!\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-RenamingF*:
‹$P\ {}_{DT}\!\!\rightsquigarrow(ev\ a)\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{DT}\!\!\rightsquigarrow(ev\ b)\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

$P\ [\![S]\!]\ Q$ laws

lemma *τ-trans-SyncL*: ‹$P\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P'\ [\![S]\!]\ Q$›
and *τ-trans-SyncR*: ‹$Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P\ [\![S]\!]\ Q'$›
⟨*proof*⟩

lemma *ev-trans-SyncL*:
‹$e \notin S \Longrightarrow P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P'\ [\![S]\!]\ Q$›
and *ev-trans-SyncR*:
‹$e \notin S \Longrightarrow Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P\ [\![S]\!]\ Q'$›
⟨*proof*⟩

lemma *ev-trans-SyncLR*:

$\langle [\![e \in S;\ P\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ P';\ Q\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ Q']\!] \Longrightarrow$
$P\ [\![S]\!]\ Q\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ P'\ [\![S]\!]\ Q' \rangle$
⟨*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*: $\langle P\ {}_{DT}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ SKIP\ [\![S]\!]\ Q \rangle$
and *tick-trans-SyncR*: $\langle Q\ {}_{DT}\!\rightsquigarrow\!\checkmark\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ [\![S]\!]\ SKIP \rangle$
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP\ [\![S]\!]\ SKIP\ {}_{DT}\!\rightsquigarrow\!\checkmark\ STOP \rangle$
⟨*proof*⟩

lemma *τ-trans-SKIP-Sync-SKIP*: $\langle SKIP\ [\![S]\!]\ SKIP\ {}_{DT}\!\rightsquigarrow_{\tau}\ SKIP \rangle$
⟨*proof*⟩

$P \rhd Q$ laws

lemma *Sliding-τ-trans-left*: $\langle P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \rhd Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \rhd Q \rangle$
⟨*proof*⟩

lemma $\langle P\ {}_{DT}\!\rightsquigarrow e\ P' \Longrightarrow P \rhd Q\ {}_{DT}\!\rightsquigarrow e\ P' \rangle$
⟨*proof*⟩

lemma $\langle P \rhd Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q \rangle$
⟨*proof*⟩

$P \triangle Q$ laws

lemma *Interrupt-τ-trans-left*: $\langle P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \triangle Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \triangle Q \rangle$
⟨*proof*⟩

lemma *Interrupt-τ-trans-right*: $\langle Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P \triangle Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P \triangle Q' \rangle$
⟨*proof*⟩

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

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

Throw P A Q laws

lemma *Throw-τ-trans-left*:
$\langle P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\rightsquigarrow_{\tau}\ P'\ \Theta\ a \in A.\ Q\ a \rangle$
⟨*proof*⟩

lemma *Throw-τ-trans-right*:
$\langle \forall\, a \in A.\ Q\ a\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q'\ a \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ \Theta\ a \in A.\ Q'\ a \rangle$
⟨*proof*⟩

lemma *Throw-event-trans-left*:
$\langle P\ {}_{DT}\!\rightsquigarrow e\ P' \Longrightarrow e \notin ev\ `\ A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\rightsquigarrow e\ (P'\ \Theta\ a \in A.\ Q\ a)\rangle$
⟨*proof*⟩

lemma *Throw-ev-trans-right*:
$\langle P\ {}_{DT}\!\rightsquigarrow (ev\ e)\ P' \Longrightarrow e \in A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\rightsquigarrow (ev\ e)\ (Q\ e)\rangle$
⟨*proof*⟩

lemma $\langle$*front-tickFree* $s \Longrightarrow \bot\ {}_{DT}\!\rightsquigarrow^* s\ P\rangle$
⟨*proof*⟩

9.2 Reality Checks

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

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

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

9.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
$\langle P\ {}_{DT}\!\rightsquigarrow^* s\ Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* $(P$ *afterTrace* $s)\rangle$
⟨*proof*⟩

lemma *trace-trans-imp-ready-set*:
$\langle P\ {}_{DT}\!\rightsquigarrow^* (s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in$ *ready-set* $(P$ *afterTrace* $s)\rangle$
⟨*proof*⟩

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
$\langle (P\ {}_{DT}\!\rightsquigarrow^* s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P$ *afterTrace* $s\ {}_{DT}\!\rightsquigarrow_\tau\ Q\rangle$
⟨*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

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P' \qquad ?P' \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P''}{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P''}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \qquad ?P' \ {}_{DT}\!\rightsquigarrow ?e \ ?P''}{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP \ {}_{DT}\!\rightsquigarrow \checkmark \ STOP}$$

$e \rightarrow P$ **rules**

$$\frac{?e \in ?A}{\Box a{\in}?A \rightarrow ?P \ a \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P \ ?e} \qquad \frac{?e \in ?A}{\sqcap a{\in}?A \rightarrow \ ?P \ a \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P \ ?e}$$

$$\frac{}{?e \rightarrow ?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P}$$

$(\sqcap)$ **rules**

$$\frac{}{?P \sqcap ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P} \qquad \frac{}{?P \sqcap ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q}$$

$$\frac{?e \in ?A}{\sqcap \ a{\in}?A. \ \ ?P \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ ?e}$$

$\mu \ x. \ f \ x$ **rule**

$$\frac{cont \ ?f \qquad ?P = (\mu \ x. \ ?f \ x)}{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?f \ ?P}$$

$(\Box)$ **rules (more powerful than in OpSemFD)**

$$?P \ \Box \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \qquad ?P \ \Box \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q$$

(;) **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ ; \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \ ; \ ?Q} \qquad \frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \ ; \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \ ; \ ?Q}$$

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

Hiding **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \setminus ?B \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \setminus ?B} \qquad \frac{?P \ {}_{DT}\!\rightsquigarrow \checkmark \ ?P'}{?P \setminus ?B \ {}_{DT}\!\rightsquigarrow \checkmark \ STOP}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \notin ?B}{?P \setminus ?B \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \setminus ?B} \qquad \frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \in ?B}{?P \setminus ?B \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \setminus ?B}$$

Renaming **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{Renaming \ ?P \ ?f \ {}_{DT}\!\rightsquigarrow_{\tau} \ Renaming \ ?P' \ ?f}$$

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

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

Sync **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \ [\![?S]\!] \ ?Q} \qquad \frac{?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \notin ?S \qquad ?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q}$$

$$\frac{?e \notin ?S \qquad ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \in ?S \qquad ?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q'}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow \checkmark \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ SKIP \ [\![?S]\!] \ ?Q}$$

$$\frac{?Q \ {}_{DT}\!\rightsquigarrow \checkmark \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ SKIP}$$

$$\frac{}{SKIP \ [\![?S]\!] \ SKIP \ {}_{DT}\!\rightsquigarrow_{\tau} \ SKIP}$$

($\rhd$) **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \rhd ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \rhd ?Q} \qquad \frac{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P'}{?P \rhd ?Q \ {}_{DT}\!\rightsquigarrow ?e \ ?P'}$$

$$\frac{}{?P \rhd ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q}$$

($\triangle$) **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \triangle ?Q'}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}$$

Throw **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{\forall \, a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q' \ a}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ \Theta \ a{\in}?A. \ ?Q' \ a}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P' \qquad ?e \notin ev \ `\ ?A}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow ?e \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \in ?A}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q \ ?e}$$

end

Chapter 10

Extension of the After Operator, bis

theory *AfterExtBis*
imports *After*
begin

10.1 The AfterExt Operator, bis

10.1.1 Definition

The refinements ($\sqsubseteq_F$) and ($\sqsubseteq_T$) are not verifying the locale axioms. In order to make the constructions available for these refinements, we will slightly modify the definition of AfterExt.

If the event is ✓ 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 if diverging, after accepting ✓ it has to stop.

definition *AfterExt* :: ‹['α *process*, 'α *event*] ⇒ 'α *process*› (**infixl** ‹*afterExt*› 77)
where ‹*P afterExt e* ≡ *case e of ev x* ⇒ *P after x* | ✓ ⇒ *STOP*›

lemma *not-ready-AfterExt*: ‹*e* ∉ *ready-set P* ⟹ *P afterExt e* = *STOP*›
⟨*proof*⟩

lemma *ready-set-AfterExt*:
‹*ready-set* (*P afterExt e*) = (*if e* = ✓ *then* {} *else* {*a*. *e* # [*a*] ∈ $\mathcal{T}$ *P*})›
⟨*proof*⟩

10.1.2 Projections

lemma *F-AfterExt*:
‹$\mathcal{F}$ (*P afterExt e*) =
(*if e* ∈ *ready-set P then* {(*tl s*, *X*)| *s X*. (*s*, *X*) ∈ $\mathcal{F}$ *P* ∧ *s* ≠ [] ∧ *hd s* = *e*}

else $\{(s,\ X).\ s = []\})$›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

lemma *D-AfterExt*: ‹$\mathcal{D}\ (P\ afterExt\ e) = (\ \ if\ e = \checkmark \wedge P = \bot\ then\ \{\}$
else $\{tl\ s |\ s\ .\ s \in \mathcal{D}\ P \wedge s \neq [] \wedge hd\ s = e\})$›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

lemma *T-AfterExt*: ‹$\mathcal{T}\ (P\ afterExt\ e) = (\ \ if\ e = \checkmark \wedge P = \bot\ then\ \{[]\}$
else insert $[]\ \{tl\ s |\ s\ .\ s \in \mathcal{T}\ P \wedge s \neq [] \wedge hd\ s = e\})$›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

10.1.3 Monotony

lemma *mono-AfterExt* : ‹$P \sqsubseteq Q \Longrightarrow P\ afterExt\ e \sqsubseteq Q\ afterExt\ e$›
⟨*proof*⟩

lemma *mono-AfterExt-T* : ‹$P \sqsubseteq_T Q \Longrightarrow P\ afterExt\ e \sqsubseteq_T Q\ afterExt\ e$›
⟨*proof*⟩

lemma *mono-AfterExt-F* :
‹$P \sqsubseteq_F Q \Longrightarrow ev\ e \notin ready\text{-}set\ P \vee ev\ e \in ready\text{-}set\ Q \Longrightarrow$
$P\ afterExt\ ev\ e \sqsubseteq_F Q\ afterExt\ ev\ e$›
⟨*proof*⟩

lemma *mono-AfterExt-D* : ‹$P \sqsubseteq_D Q \Longrightarrow P\ afterExt\ e \sqsubseteq_D Q\ afterExt\ e$›
⟨*proof*⟩

lemma *mono-AfterExt-FD* :
‹$P \sqsubseteq_{FD} Q \Longrightarrow e \notin ready\text{-}set\ P \vee e \in ready\text{-}set\ Q \Longrightarrow$
$P\ afterExt\ e \sqsubseteq_{FD} Q\ afterExt\ e$›
⟨*proof*⟩

lemma *mono-AfterExt-DT* : ‹$P \sqsubseteq_{DT} Q \Longrightarrow P\ afterExt\ e \sqsubseteq_{DT} Q\ afterExt\ e$›
⟨*proof*⟩

10.1.4 Behaviour of *AfterExt* with *STOP*, *SKIP* and ⊥

lemma *AfterExt-STOP*: ‹$STOP\ afterExt\ e = STOP$›
⟨*proof*⟩

lemma *AfterExt-is-STOP-iff*:
‹$P\ afterExt\ e = STOP \longleftrightarrow P = \bot \wedge e = \checkmark \vee (\forall s.\ e\ \#\ s \in \mathcal{T}\ P \longrightarrow s = [])$›
⟨*proof*⟩

lemma *AfterExt-SKIP*: ‹*SKIP afterExt e* = *STOP*›
⟨*proof*⟩

lemma *AfterExt-BOT* : ‹⊥ *afterExt e* = (*if e* = ✓ *then STOP else* ⊥)›
⟨*proof*⟩

lemma *AfterExt-is-BOT-iff*: ‹*P afterExt e* = ⊥ ⟷ *e* ≠ ✓ ∧ [*e*] ∈ $\mathcal{D}$ *P*›
⟨*proof*⟩

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

Here again, we lose determinism.

lemma *AfterExt-Mprefix-is-AfterExt-Mndetprefix*:
‹(□*a* ∈ *A* → *P a*) *afterExt e* = (⊓*a* ∈ *A* → *P a*) *afterExt e*›
⟨*proof*⟩

lemma *AfterExt-Det-is-AfterExt-Ndet*: ‹*P* □ *Q afterExt e* = *P* ⊓ *Q afterExt e*›
⟨*proof*⟩

lemma *AfterExt-Ndet*:
‹*P* ⊓ *Q afterExt e* = (*case e of ev x* ⇒ *if ev x* ∈ *ready-set P* ∩ *ready-set Q*
then (*P afterExt ev x*) ⊓ (*Q afterExt ev x*)
else if ev x ∈ *ready-set P*
then P afterExt ev x
else Q afterExt ev x
| ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-Mprefix*:
‹(□ *a* ∈ *A* → *P a*) *afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then P x else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

corollary *AfterExt-prefix*:
‹(*a* → *P*) *afterExt e* =
(*case e of ev x* ⇒ *if x* = *a then P else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

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*: ‹*Renaming P f* = ⊥ ⟷ *P* = ⊥›
⟨*proof*⟩

lemma *Renaming-is-STOP-iff*: ‹*Renaming P f* = *STOP* ⟷ *P* = *STOP*›
⟨*proof*⟩

lemma *AfterExt-Renaming*:
‹*Renaming P f afterExt e* =
(*case e of* ✓ ⇒ *STOP*
| *ev a* ⇒ *if P* = ⊥ *then* ⊥ *else*
⊓*a* ∈ {*a*. *ev a* ∈ *ready-set P* ∧ *ev* (*f a*) = *e*}. *Renaming* (*P afterExt ev a*) *f*)›
⟨*proof*⟩
lemma *Seq-is-BOT-iff*: ‹*P* ; *Q* = ⊥ ⟷ *P* = ⊥ ∨ ([✓] ∈ *T P* ∧ *Q* = ⊥)›
⟨*proof*⟩

lemma *AfterExt-Seq*:
‹(*P* ; *Q*) *afterExt e* =
(*if e* ∉ *ready-set P* ∧ *e* ∉ *ready-set Q then STOP*
else if e ∉ *ready-set Q then P afterExt e* ; *Q*
else if e ∉ *ready-set P then if* ✓ ∈ *ready-set P then Q afterExt e else STOP*
else if ✓ ∈ *ready-set P then* (*P afterExt e* ; *Q*) ⊓ (*Q afterExt e*)
else P afterExt e ; *Q*)›
⟨*proof*⟩

theorem *AfterExt-Sync*:
‹(*P* ⟦*S*⟧ *Q*) *afterExt e* =
(*case e of* ✓ ⇒ *STOP*
| *ev x* ⇒ *if P* = ⊥ ∨ *Q* = ⊥ *then* ⊥
else if e ∈ *ready-set P* ∧ *e* ∈ *ready-set Q*
then if x ∈ *S*
then P afterExt e ⟦*S*⟧ *Q afterExt e*
else (*P afterExt e* ⟦*S*⟧ *Q*) ⊓ (*P* ⟦*S*⟧ *Q afterExt e*)
else if e ∈ *ready-set P*
then if x ∈ *S then STOP else P afterExt e* ⟦*S*⟧ *Q*
else if e ∈ *ready-set Q*
then if x ∈ *S then STOP else P* ⟦*S*⟧ *Q afterExt e*
else STOP)›
⟨*proof*⟩

lemma *Hiding-FD-Hiding-AfterExt-if-ready-inside*:
‹*e* ∈ *B* ⟹ (*P* \ *B*) ⊑*FD* (*P afterExt ev e* \ *B*)›
and *AfterExt-Hiding-FD-Hiding-AfterExt-if-ready-notin*:
‹*e* ∉ *B* ⟹ (*P* \ *B*) *afterExt ev e* ⊑*FD* (*P afterExt ev e* \ *B*)›
if *ready*: ‹*ev e* ∈ *ready-set P*›

⟨*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*:
‹⟦*ev e* ∈ *ready-set P*; *e* ∈ *B*⟧ ⟹ (*P* \ *B*) ⊑$_{DT}$ (*P afterExt ev e* \ *B*)›
and *AfterExt-Hiding-DT-Hiding-AfterExt-if-ready-notin*:
‹⟦*ev e* ∈ *ready-set P*; *e* ∉ *B*⟧ ⟹ (*P* \ *B*) *afterExt ev e* ⊑$_{DT}$ (*P afterExt ev e* \
B)›
⟨*proof*⟩

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

lemma *AfterExt-MultiDet-is-AfterExt-MultiNdet*:
‹*finite A* ⟹ (□ *a* ∈ *A*. *P a*) *afterExt e* = (⨅ *a* ∈ *A*. *P a*) *afterExt e*›
⟨*proof*⟩

lemma *AfterExt-GlobalNdet*:
‹(⊓ *a* ∈ *A*. *P a*) *afterExt e* =
(*if e* = ✓ ∨ *e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⊓ *a* ∈ {*a* ∈ *A*. *e* ∈ *ready-set* (*P a*)}. *P a*) *afterExt e*)›
and *AfterExt-MultiNdet*:
‹*finite A* ⟹ (⨅ *a* ∈ *A*. *P a*) *afterExt e* =
(*if e* = ✓ ∨ *e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⨅ *a* ∈ {*a* ∈ *A*. *e* ∈ *ready-set* (*P a*)}. *P a*) *afterExt e*)›
and *AfterExt-MultiDet*:
‹*finite A* ⟹ (□ *a* ∈ *A*. *P a*) *afterExt e* =
(*if e* = ✓ ∨ *e* ∉ (⋃ *a* ∈ *A*. *ready-set* (*P a*)) *then STOP*
else (⨅ *a* ∈ {*a* ∈ *A*. *e* ∈ *ready-set* (*P a*)}. *P a*) *afterExt e*)›
⟨*proof*⟩

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

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

‹$P \rhd Q$ *afterExt* $e = P \sqcap Q$ *afterExt* e›
⟨*proof*⟩

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

lemma *AfterExt-Throw*:
‹($P \Theta\ a \in A.\ Q\ a$) *afterExt* $e =$
(*case* e *of* ✓ ⇒ *STOP*
| *ev* $x \Rightarrow$ *if* $P = \bot$ *then* $\bot$
else *if ev* $x \in$ *ready-set* P *then if* $x \in A$ *then* $Q\ x$
else (P *after* x) $\Theta\ a \in A.\ Q\ a$ *else STOP*)›
⟨*proof*⟩

lemma *AfterExt-Interrupt*:
‹($P \triangle Q$) *afterExt* $e =$
(*case* e *of* ✓ ⇒ *STOP*
| *ev* $x \Rightarrow$ *if* $P = \bot \vee Q = \bot$ *then* $\bot$
else *if ev* $x \in$ *ready-set* $P \wedge$ *ev* $x \in$ *ready-set* Q
then (Q *after* x) $\sqcap$ (P *after* $x \triangle Q$)
else *if ev* $x \in$ *ready-set* $P \wedge$ *ev* $x \notin$ *ready-set* Q
then P *after* $x \triangle Q$
else *if ev* $x \notin$ *ready-set* $P \wedge$ *ev* $x \in$ *ready-set* Q
then Q *after* x
else STOP)›
⟨*proof*⟩

10.1.8 Behaviour of *AfterExt* with Reference Processes

lemma *AfterExt-DF*:
‹*DF* A *afterExt* $e =$
(*case* e *of ev* $x \Rightarrow$ *if* $x \in A$ *then DF* A *else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-DF*$_{SKIP}$:
‹*DF*$_{SKIP}$ A *afterExt* $e =$
(*case* e *of ev* $x \Rightarrow$ *if* $x \in A$ *then DF*$_{SKIP}$ A *else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-RUN*:
‹*RUN* A *afterExt* $e =$
(*case* e *of ev* $x \Rightarrow$ *if* $x \in A$ *then RUN* A *else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-CHAOS*:
‹*CHAOS* A *afterExt* $e =$
(*case* e *of ev* $x \Rightarrow$ *if* $x \in A$ *then CHAOS* A *else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *AfterExt-CHAOS$_{SKIP}$*:
‹*CHAOS$_{SKIP}$ A afterExt e* =
(*case e of ev x* ⇒ *if x* ∈ *A then CHAOS$_{SKIP}$ A else STOP* | ✓ ⇒ *STOP*)›
⟨*proof*⟩

lemma *DF-FD-AfterExt*:
‹*DF A* ⊑$_{FD}$ *P* ⟹ *e* ∈ *ready-set P* ⟹ *DF A* ⊑$_{FD}$ *P afterExt e*›
⟨*proof*⟩

lemma *DF$_{SKIP}$-FD-AfterExt*:
‹*DF$_{SKIP}$ A* ⊑$_{FD}$ *P* ⟹ *ev e* ∈ *ready-set P* ⟹ *DF$_{SKIP}$ A* ⊑$_{FD}$ *P afterExt ev e*›
⟨*proof*⟩

lemma *deadlock-free-AfterExt*:
‹*deadlock-free P* ⟹ *deadlock-free* (*P afterExt e*) ⟷
(*if e* ∈ *ready-set P* ∧ *e* ≠ ✓ *then True else False*)›
⟨*proof*⟩

lemma *deadlock-free$_{SKIP}$-AfterExt*:
‹*deadlock-free$_{SKIP}$ P* ⟹ *deadlock-free$_{SKIP}$* (*P afterExt e*) ⟷
(*if e* ∈ *ready-set P* ∧ *e* ≠ ✓ *then True else False*)›
⟨*proof*⟩

end

10.2 The AfterTrace Operator, bis

theory *AfterTraceBis*
imports *AfterExtBis HOL−CSPM.DeadlockResults*
begin

10.2.1 Definition

We just defined *P afterExt e* for *P* an *'α process* and *e* of type *'α event*. Since traces of an *'α process* are *'α event list*, the following inductive definition is natural.

fun *AfterTrace* :: ‹*'α process* ⇒ *'α trace* ⇒ *'α process*› (**infixl** ‹*afterTrace*› *77*)
where ‹*P afterTrace* [] = *P*›
| ‹*P afterTrace* (*e* # *s*) = *P afterExt e afterTrace s*›

We can also induct backward.

lemma *AfterTrace-append*: ‹*P afterTrace* (*s* @ *t*) = *P afterTrace s afterTrace t*›
⟨*proof*⟩

lemma *AfterTrace-snoc* : ‹*P afterTrace* (*s* @ [*e*]) = *P afterTrace s afterExt e*›
⟨*proof*⟩

We have some immediate properties.

lemma *AfterTrace-STOP* : ‹*STOP afterTrace s* = *STOP*›
⟨*proof*⟩

lemma *AfterTrace-SKIP* : ‹*SKIP afterTrace s* = (*if s* = [] *then SKIP else STOP*)›
⟨*proof*⟩

lemma *AfterTrace-BOT* : ‹⊥ *afterTrace s* = (*if tickFree s then* ⊥ *else STOP*)›
⟨*proof*⟩

10.2.2 Projections

lemma *F-AfterTrace* : ‹(*s* @ *t*, *X*) ∈ ℱ *P* ⟹ (*t*, *X*) ∈ ℱ (*P afterTrace s*)›
⟨*proof*⟩

lemma *D-AfterTrace* :
‹*tickFree* (*s* @ *t*) ⟹ *s* @ *t* ∈ 𝒟 *P* ⟹ *t* ∈ 𝒟 (*P afterTrace s*)›
⟨*proof*⟩

lemma *T-AfterTrace* : ‹*s* @ *t* ∈ 𝒯 *P* ⟹ *t* ∈ 𝒯 (*P afterTrace s*)›
⟨*proof*⟩

corollary *ready-set-AfterTrace* :
‹*s* @ *e* # *t* ∈ 𝒯 *P* ⟹ *e* ∈ *ready-set* (*P afterTrace s*)›
⟨*proof*⟩

corollary *F-imp-refusal-AfterTrace*:
‹(*s*, *X*) ∈ ℱ *P* ⟹ ([], *X*) ∈ ℱ (*P afterTrace s*)›
⟨*proof*⟩

corollary *D-imp-AfterTrace-is-BOT*:
‹*tickFree s* ⟹ *s* ∈ 𝒟 *P* ⟹ *P afterTrace s* = ⊥›
⟨*proof*⟩

10.2.3 Monotony

lemma *mono-AfterTrace* : ‹*P* ⊑ *Q* ⟹ *P afterTrace s* ⊑ *Q afterTrace s*›
⟨*proof*⟩

lemma *mono-AfterTrace-T* :
‹*P* ⊑$_T$ *Q* ⟹ *s* ∈ 𝒯 *Q* ⟹ *tickFree s* ⟹ *P afterTrace s* ⊑$_T$ *Q afterTrace s*›

⟨*proof*⟩

lemma *mono-AfterTrace-F* :
‹$P \sqsubseteq_F Q \Longrightarrow s \in \mathcal{T}\ Q \Longrightarrow tickFree\ s \Longrightarrow P\ afterTrace\ s \sqsubseteq_F Q\ afterTrace\ s$›
⟨*proof*⟩

lemma *mono-AfterTrace-D* : ‹$P \sqsubseteq_D Q \Longrightarrow P\ afterTrace\ s \sqsubseteq_D Q\ afterTrace\ s$›
⟨*proof*⟩

lemma *mono-AfterTrace-FD* :
‹$P \sqsubseteq_{FD} Q \Longrightarrow s \in \mathcal{T}\ Q \Longrightarrow P\ afterTrace\ s \sqsubseteq_{FD} Q\ afterTrace\ s$›
⟨*proof*⟩

lemma *mono-AfterTrace-DT* :
‹$P \sqsubseteq_{DT} Q \Longrightarrow P\ afterTrace\ s \sqsubseteq_{DT} Q\ afterTrace\ s$›
⟨*proof*⟩

10.2.4 Another Definition of *events-of*

inductive *reachable-event* :: ‹$'\alpha\ process \Rightarrow '\alpha \Rightarrow bool$›
where ‹$ev\ e \in ready\text{-}set\ P \Longrightarrow reachable\text{-}event\ P\ e$›
| ‹$reachable\text{-}event\ (P\ after\ f)\ e \Longrightarrow reachable\text{-}event\ P\ e$›

lemma *events-of-iff-reachable-event*: ‹$e \in events\text{-}of\ P \longleftrightarrow reachable\text{-}event\ P\ e$›
⟨*proof*⟩

lemma *reachable-event-BOT*: ‹$reachable\text{-}event\ \bot\ e$›
⟨*proof*⟩

lemma *not-reachable-event-STOP*: ‹$\neg\ reachable\text{-}event\ STOP\ e$›
⟨*proof*⟩

lemma *reachable-tick-SKIP*: ‹$\neg\ reachable\text{-}event\ SKIP\ \checkmark$›
⟨*proof*⟩

lemma *reachable-event-iff-in-ready-set-AfterTrace*:
‹$reachable\text{-}event\ P\ e \longleftrightarrow e \in \{e|\ e\ s.\ ev\ e \in ready\text{-}set\ (P\ afterTrace\ s)\}$›
⟨*proof*⟩

10.2.5 Characterizations for Deadlock Freeness

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

lemma *deadlock-free$_{SKIP}$-AfterExt-characterization*:
‹*deadlock-free$_{SKIP}$ P* $\longleftrightarrow$
UNIV $\notin \mathcal{R}$ *P* $\wedge$ ($\forall$ *e* $\in$ *ready-set P* $-$ {✓}. *deadlock-free$_{SKIP}$* (*P afterExt e*))›
(**is** ‹*deadlock-free$_{SKIP}$ P* $\longleftrightarrow$ *?rhs*›)
⟨*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 τ-*trans* :: ‹$'\alpha$ *process* ⇒ $'\alpha$ *process* ⇒ *bool*› (**infixl** ‹$\rightsquigarrow_\tau$› *50*)
assumes τ-*trans-NdetL*: ‹$P \sqcap Q \rightsquigarrow_\tau P$›
and τ-*trans-transitivity*: ‹$P \rightsquigarrow_\tau Q \Longrightarrow Q \rightsquigarrow_\tau R \Longrightarrow P \rightsquigarrow_\tau R$›
and τ-*trans-anti-mono-ready-set*: ‹$P \rightsquigarrow_\tau Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* P›
and τ-*trans-mono-AfterExt*:
‹$e \in$ *ready-set* $Q \Longrightarrow P \rightsquigarrow_\tau Q \Longrightarrow P$ *afterExt* $e \rightsquigarrow_\tau Q$ *afterExt* e›
begin

This locale needs to be instantiated with a binary relation ($\rightsquigarrow_\tau$) 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* ⇒ $'\alpha$ *event* ⇒ $'\alpha$ *process* ⇒ *bool*› (‹-/ ↝-/ -› [*50*, *3*, *51*] *50*)
where ‹$P \rightsquigarrow e\ Q \equiv e \in$ *ready-set* $P \wedge P$ *afterExt* $e \rightsquigarrow_\tau Q$›

From idempotence, commutativity and $\perp$ absorbance of ($\sqcap$), we get the following for free.

lemma *τ-trans-eq*: ‹$P \rightsquigarrow_\tau P$›
and *τ-trans-NdetR*: ‹$P \sqcap Q \rightsquigarrow_\tau Q$›
and *BOT-τ-trans-anything*: ‹$\bot \rightsquigarrow_\tau P$›
and *BOT-ev-trans-anything*: ‹$\bot \rightsquigarrow (ev\ e)\ P$›
⟨*proof*⟩

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

lemma *event-trans-τ-trans*: ‹$P \rightsquigarrow e\ P' \Longrightarrow P' \rightsquigarrow_\tau P'' \Longrightarrow P \rightsquigarrow e\ P''$›
⟨*proof*⟩

lemma *τ-trans-event-trans*: ‹$P \rightsquigarrow_\tau P' \Longrightarrow P' \rightsquigarrow e\ P'' \Longrightarrow P \rightsquigarrow e\ P''$›
⟨*proof*⟩

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

inductive *trace-trans* :: ‹$'\alpha$ *process* ⇒ $'\alpha$ *trace* ⇒ $'\alpha$ *process* ⇒ *bool*› (‹-/ $\rightsquigarrow^*$-/ -› [*50*, *3*, *51*] *50*)
where *trace-τ-trans* : ‹$P \rightsquigarrow_\tau P' \Longrightarrow P \rightsquigarrow^* [] P'$›
| *trace-tick-trans* : ‹$P \rightsquigarrow ✓\ P' \Longrightarrow P \rightsquigarrow^* [✓] P'$›
| *trace-Cons-ev-trans* :
‹$P \rightsquigarrow (ev\ e)\ P' \Longrightarrow P' \rightsquigarrow^*\ s\ P'' \Longrightarrow P \rightsquigarrow^*\ (ev\ e) \#\ s\ P''$›

lemma *trace-trans-τ-trans*: ‹$P \rightsquigarrow^* s\ P' \Longrightarrow P' \rightsquigarrow_\tau P'' \Longrightarrow P \rightsquigarrow^* s\ P''$›
⟨*proof*⟩

lemma *τ-trans-trace-trans*: ‹$P \rightsquigarrow_\tau P' \Longrightarrow P' \rightsquigarrow^* s\ P'' \Longrightarrow P \rightsquigarrow^* s\ P''$›
⟨*proof*⟩

lemma *tickFree-if-trans-trans-not-STOP* :
‹$P \rightsquigarrow^* s\ Q \Longrightarrow Q \neq STOP \Longrightarrow tickFree\ s$›
⟨*proof*⟩

lemma *BOT-trace-trans-tickFree-anything* : ‹*tickFree* $s \Longrightarrow \bot \rightsquigarrow^* s\ P$›
⟨*proof*⟩

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*:
‹$P \rightsquigarrow^* s\ Q \Longrightarrow X \in \mathcal{R}\ Q \Longrightarrow (s, X) \in \mathcal{F}\ P$›
if ‹$\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q$›
⟨*proof*⟩

lemma *trace-trans-imp-T-if-τ-trans-imp-leT*: ‹$P \rightsquigarrow^* s\ Q \Longrightarrow s \in \mathcal{T}\ P$›
  if ‹$\forall\, P\ Q.\ P \rightsquigarrow_\tau\ Q \longrightarrow P \sqsubseteq_T\ Q$›
⟨*proof*⟩

lemma *trace-trans-BOT-imp-D-if-τ-trans-imp-leD*: ‹$P \rightsquigarrow^* s \perp \Longrightarrow s \in \mathcal{D}\ P$›
  if ‹$\forall\, P\ Q.\ P \rightsquigarrow_\tau\ Q \longrightarrow P \sqsubseteq_D\ Q$›
⟨*proof*⟩

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* :
  ‹$P \rightsquigarrow^* []\ Q \longleftrightarrow P \rightsquigarrow_\tau\ Q$›
  ‹$P \rightsquigarrow^* [✓]\ Q \longleftrightarrow P \rightsquigarrow✓\ Q$›
  ‹$P \rightsquigarrow^* (ev\ e) \#\ s\ Q' \longleftrightarrow (\exists\, Q.\ P \rightsquigarrow(ev\ e)\ Q \wedge Q \rightsquigarrow^*\ s\ Q')$›
  ‹*tickFree* $s \Longrightarrow (P \rightsquigarrow^*\ s\ @\ [f]\ Q') \longleftrightarrow (\exists\, Q.\ P \rightsquigarrow^* s\ Q \wedge Q \rightsquigarrow f\ Q')$›
  ‹*front-tickFree* $(s\ @\ t) \Longrightarrow (P \rightsquigarrow^* s\ @\ t\ Q') \longleftrightarrow (\exists\, Q.\ P \rightsquigarrow^* s\ Q \wedge Q \rightsquigarrow^* t\ Q')$›
⟨*proof*⟩

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

theorem *T-imp-trace-trans-iff-AfterTrace-τ-trans* :
  ‹$s \in \mathcal{T}\ P \Longrightarrow (P \rightsquigarrow^* s\ Q) \longleftrightarrow P$ *afterTrace* $s \rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

As corollaries we obtain the reciprocal results of
$[\![\forall\, P\ Q.\ P \rightsquigarrow_\tau\ Q \longrightarrow P \sqsubseteq_F\ Q;\ ?P \rightsquigarrow^* ?s\ ?Q;\ ?X \in \mathcal{R}\ ?Q]\!] \Longrightarrow (?s,\ ?X) \in \mathcal{F}\ ?P$
$[\![\forall\, P\ Q.\ P \rightsquigarrow_\tau\ Q \longrightarrow P \sqsubseteq_T\ Q;\ ?P \rightsquigarrow^* ?s\ ?Q]\!] \Longrightarrow ?s \in \mathcal{T}\ ?P$
$[\![\forall\, P\ Q.\ P \rightsquigarrow_\tau\ Q \longrightarrow P \sqsubseteq_D\ Q;\ ?P \rightsquigarrow^* ?s \perp]\!] \Longrightarrow ?s \in \mathcal{D}\ ?P$

lemma *F-imp-exists-trace-trans*: ‹$(s,\ X) \in \mathcal{F}\ P \Longrightarrow \exists\, Q.\ (P \rightsquigarrow^* s\ Q) \wedge X \in \mathcal{R}\ Q$›
  ⟨*proof*⟩

lemma *T-imp-exists-trace-trans*: ‹$s \in \mathcal{T}\ P \Longrightarrow \exists\, Q.\ P \rightsquigarrow^* s\ Q$›
  ⟨*proof*⟩

lemma *D-imp-trace-trans-BOT*: ‹*tickFree* $s \Longrightarrow s \in \mathcal{D}\ P \Longrightarrow P \rightsquigarrow^* s \perp$›
  ⟨*proof*⟩

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

lemma *τ-trans-imp-leT-imp-STOP-trace-trans-iff*:
  ‹$\forall\, P\ Q.\ P \rightsquigarrow_\tau\ Q \longrightarrow P \sqsubseteq_T\ Q \Longrightarrow STOP \rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = STOP$›
  ⟨*proof*⟩

lemma *τ-trans-imp-leF-imp-SKIP-trace-trans-iff*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_F$ *Q* ⟹
SKIP ⇝* *s P* ⟷ *s* = [] ∧ *P* = *SKIP* ∨ *s* = [✓] ∧ *P* = *STOP*›
⟨*proof*⟩

lemma *τ-trans-imp-leT-imp-trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_T$ *Q* ⟹ *P* ⇝* *s Q* ⟹
ready-set Q ⊆ *ready-set* (*P afterTrace s*)›
⟨*proof*⟩

lemma *τ-trans-imp-leT-imp-trace-trans-imp-ready-set*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_T$ *Q* ⟹ *P* ⇝*(*s* @ *e* # *t*) *Q* ⟹
e ∈ *ready-set* (*P afterTrace s*)›
⟨*proof*⟩

lemma *trace-trans-iff-T-and-AfterTrace-τ-trans-if-τ-trans-imp-leT*:
‹∀ *P Q*. *P* ⇝$_\tau$ *Q* ⟶ *P* ⊑$_T$ *Q* ⟹
(*P* ⇝* *s Q*) ⟷ *s* ∈ $\mathcal{T}$ *P* ∧ *P afterTrace s* ⇝$_\tau$ *Q*›
⟨*proof*⟩

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* ⇝$_\tau$ *Q* is.

lemma *SKIP-trans-tick*: ‹*SKIP* ⇝✓ *STOP*›
⟨*proof*⟩

lemma *tick-trans-imp-BOT-L-or-STOP-R*: ‹*P* ⇝✓ *Q* ⟹ *P* = ⊥ ∨ *Q* = *STOP*›
⟨*proof*⟩

lemma *STOP-trace-trans-iff* : ‹*STOP* ⇝* *s P* ⟷ *s* = [] ∧ *P* = *STOP*›
⟨*proof*⟩

lemma *ready-tick-imp-τ-trans-SKIP*: ‹*P* ⇝$_\tau$ *SKIP*› **if** ‹✓ ∈ *ready-set P*›
⟨*proof*⟩

lemma *exists-tick-trans-is-ready-tick*: ‹(∃ *P′*. *P* ⇝✓ *P′*) ⟷ ✓ ∈ *ready-set P*›
⟨*proof*⟩

lemma *tick-trans-iff* : ‹*P* ⇝✓ *P′* ⟷ *P* ⇝$_\tau$ *SKIP* ∧ *P′* = *STOP*›

⟨*proof*⟩

lemma *SKIP-cant-ev-trans*: ‹¬ *SKIP* ⇝(*ev e*) *STOP*›
⟨*proof*⟩

lemma *STOP-cant-event-trans*: ‹¬ *STOP* ⇝*e P*›
⟨*proof*⟩

lemma *ev-trans-Mprefix*: ‹*e* ∈ *A* ⟹ □*a* ∈ *A* → *P a* ⇝(*ev e*) (*P e*)›
⟨*proof*⟩

lemma *ev-trans-Mndetprefix*: ‹*e* ∈ *A* ⟹ ⊓*a* ∈ *A* → *P a* ⇝(*ev e*) (*P e*)›
⟨*proof*⟩

lemma *ev-trans-prefix*: ‹*e* → *P* ⇝ (*ev e*) *P*›
⟨*proof*⟩

lemma *τ-trans-MultiNdet*: ‹*finite A* ⟹ *x* ∈ *A* ⟹ (⨅ *a* ∈ *A*. *P a*) ⇝$_\tau$ *P x*›
⟨*proof*⟩

lemma *τ-trans-GlobalNdet*: ‹(⊓*a* ∈ *A*. *P a*) ⇝$_\tau$ *P e*› **if** ‹*e* ∈ *A*›
⟨*proof*⟩

lemma *fix-point-τ-trans*: ‹*cont f* ⟹ *P* = (*μ X*. *f X*) ⟹ *P* ⇝$_\tau$ *f P*›
⟨*proof*⟩

lemma *event-trans-DetL*: ‹*P* ⇝*e P′* ⟹ *P* □ *Q* ⇝*e P′*›
⟨*proof*⟩

lemma *event-trans-DetR*: ‹*Q* ⇝*e Q′* ⟹ *P* □ *Q* ⇝*e Q′*›
⟨*proof*⟩

lemma *event-trans-MultiDet*:
‹*finite A* ⟹ *a* ∈ *A* ⟹ *P a* ⇝*e Q* ⟹ (□*a* ∈ *A*. *P a*) ⇝*e Q*›
⟨*proof*⟩

lemma *Sliding-event-transL*: ‹*P* ⇝*e P′* ⟹ *P* ▷ *Q* ⇝*e P′*›
⟨*proof*⟩

lemma *Sliding-τ-transR*: ‹$P \rhd Q \rightsquigarrow_\tau Q$›
⟨*proof*⟩

lemma ‹$\exists\, P\ P'\ Q.\ P \rightsquigarrow\checkmark\ P' \wedge \neg\, P\ ;\ Q \rightsquigarrow\checkmark\ P'\ ;\ Q$›
⟨*proof*⟩

lemma *ev-trans-SeqR*:
‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q \rightsquigarrow(ev\ e)\ Q' \Longrightarrow P\ ;\ Q \rightsquigarrow(ev\ e)\ Q'$›
⟨*proof*⟩

lemma ‹$SKIP\ [\![S]\!]\ SKIP \rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma ‹$SKIP\ [\![S]\!]\ SKIP \rightsquigarrow_\tau SKIP$›
⟨*proof*⟩

lemma *tick-trans-Hiding*: ‹$P \setminus B \rightsquigarrow\checkmark\ STOP$› **if** ‹$P \rightsquigarrow\checkmark\ P'$›
⟨*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 ‹*Renaming* $P\ f \rightsquigarrow\checkmark\ STOP$› **if** ‹$P \rightsquigarrow\checkmark\ P'$›
⟨*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 *τ-trans* :: ‹'α *process* ⇒ 'α *process* ⇒ *bool*› (**infixl** ‹$_{FD}\!\rightsquigarrow_{\tau}$› *50*)
where ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q \equiv P \sqsubseteq_{FD} Q$›

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

interpretation *OpSemGeneric* ‹$({}_{FD}\!\rightsquigarrow_{\tau})$›
⟨*proof*⟩

notation *event-trans* (‹-/ $_{FD}\!\rightsquigarrow$-/ -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹-/ $_{FD}\!\rightsquigarrow^{*}$-/ -› [*50*, *3*, *51*] *50*)

lemma ‹$P\ {}_{FD}\!\rightsquigarrow\ e\ P' \Longrightarrow P'\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'' \Longrightarrow P\ {}_{FD}\!\rightsquigarrow\ e\ P''$›
‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P'\ {}_{FD}\!\rightsquigarrow\ e\ P'' \Longrightarrow P\ {}_{FD}\!\rightsquigarrow\ e\ P''$›
⟨*proof*⟩

12.1 Operational Semantics Laws

SKIP law

lemma ‹$SKIP\ {}_{FD}\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

$e \rightarrow P$ laws

lemma ‹$e \in A \Longrightarrow \Box a \in A \rightarrow P\ a\ {}_{FD}\!\rightsquigarrow(ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \in A \Longrightarrow \sqcap a \in A \rightarrow P\ a\ {}_{FD}\!\rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \rightarrow P\ {}_{FD}\!\rightsquigarrow\ (ev\ e)\ P$›
⟨*proof*⟩

$P \sqcap Q$ laws

lemma ‹$P \sqcap Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P$›
and ‹$P \sqcap Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$a \in A \Longrightarrow (\sqcap a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

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

lemma ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P\ {}_{FD}\!\rightsquigarrow_{\tau}\ f\ P$›
⟨*proof*⟩

$P \Box Q$ laws

lemma τ-*trans-DetL*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Box Q$ ›
and τ-*trans-DetR*: ‹$Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ \Box\ Q'$›
⟨*proof*⟩

lemma τ-*trans-MultiDet*:
‹*finite* $A \Longrightarrow \forall\, a \in A.\ P\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ a \Longrightarrow$
$(\Box a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow_{\tau}\ (\Box a \in A.\ P'\ a)$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow e\ P' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow e\ P'$›
and ‹$Q\ {}_{FD}\!\rightsquigarrow e\ Q' \Longrightarrow P \Box Q\ {}_{FD}\!\rightsquigarrow e\ Q'$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow P\ a\ {}_{FD}\!\rightsquigarrow e\ Q \Longrightarrow (\Box a \in A.\ P\ a)\ {}_{FD}\!\rightsquigarrow e\ Q$›
⟨*proof*⟩

$P\ ;\ Q$ laws

lemma τ-*trans-SeqL*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ ;\ Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ ;\ Q$›
⟨*proof*⟩

lemma *ev-trans-SeqL*: ‹$P\ {}_{FD}\!\rightsquigarrow (ev\ e)\ P' \Longrightarrow P\ ;\ Q\ {}_{FD}\!\rightsquigarrow (ev\ e)\ P'\ ;\ Q$›

⟨*proof*⟩

lemma *τ-trans-SeqR*: ‹$\exists\, P'.\ P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P\ ;;\ Q\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q\ {}_{FD}\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P\ ;;\ Q\ {}_{FD}\!\rightsquigarrow(ev\ e)\ Q'$›

⟨*proof*⟩

$P \setminus B$ laws

lemma *τ-trans-Hiding*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

lemma *ev-trans-Hiding-notin*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \notin B \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \setminus B$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow\!\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Hiding-inside*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in B \Longrightarrow P \setminus B\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

Renaming P f laws

lemma *τ-trans-Renaming*: ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow Renaming\ P\ f\ {}_{FD}\!\rightsquigarrow_{\tau}\ Renaming\ P'\ f$›
⟨*proof*⟩

lemma *tick-trans-Renaming*: ‹$P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow Renaming\ P\ f\ {}_{FD}\!\rightsquigarrow\!\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Renaming*:
‹$f\ a = b \Longrightarrow P\ {}_{FD}\!\rightsquigarrow(ev\ a)\ P' \Longrightarrow Renaming\ P\ f\ {}_{FD}\!\rightsquigarrow(ev\ b)\ (Renaming\ P'\ f)$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

lemma ‹$P\ {}_{FD}\!\rightsquigarrow\!\checkmark\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{FD}\!\rightsquigarrow\!\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-RenamingF*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ a)\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{FD}\!\rightsquigarrow(ev\ b)\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

$P\ [\![S]\!]\ Q$ laws

lemma τ-*trans-SyncL*: $\langle P \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \ [\![S]\!] \ Q\rangle$
and τ-*trans-SyncR*: $\langle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ Q' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P \ [\![S]\!] \ Q'\rangle$
⟨*proof*⟩

lemma *ev-trans-SyncL*:
$\langle e \notin S \Longrightarrow P \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \ [\![S]\!] \ Q \ \rangle$
and *ev-trans-SyncR*:
$\langle e \notin S \Longrightarrow Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ Q' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P \ \ [\![S]\!] \ Q'\rangle$
⟨*proof*⟩

lemma *ev-trans-SyncLR*:
$\langle [\![e \in S; \ P \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P'; \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ Q']\!] \Longrightarrow$
$P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \ [\![S]\!] \ Q'\rangle$
⟨*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*: $\langle P \ {}_{FD}\!\rightsquigarrow\checkmark \ P' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP \ [\![S]\!] \ Q\rangle$
and *tick-trans-SyncR*: $\langle Q \ {}_{FD}\!\rightsquigarrow\checkmark \ Q' \Longrightarrow P \ [\![S]\!] \ Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P \ [\![S]\!] \ SKIP\rangle$
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: $\langle SKIP \ [\![S]\!] \ SKIP \ {}_{FD}\!\rightsquigarrow\checkmark \ STOP\rangle$
⟨*proof*⟩

lemma τ-*trans-SKIP-Sync-SKIP*: $\langle SKIP \ [\![S]\!] \ SKIP \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP\rangle$
⟨*proof*⟩

$P \rhd Q$ laws

lemma *Sliding-*τ*-trans-left*: $\langle P \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \Longrightarrow P \rhd Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \rhd Q\rangle$
⟨*proof*⟩

lemma $\langle P \ {}_{FD}\!\rightsquigarrow e \ P' \Longrightarrow P \rhd Q \ {}_{FD}\!\rightsquigarrow e \ P'\rangle$
⟨*proof*⟩

lemma $\langle P \rhd Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ Q\rangle$
⟨*proof*⟩

$P \triangle Q$ laws

lemma *Interrupt-*τ*-trans-left*: $\langle P \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \Longrightarrow P \triangle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P' \triangle Q\rangle$
⟨*proof*⟩

lemma *Interrupt-*τ*-trans-right*: $\langle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ Q' \Longrightarrow P \triangle Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ P \triangle Q'\rangle$
⟨*proof*⟩

lemma *Interrupt-ev-trans-left*:
$\langle P \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \Longrightarrow P \triangle Q \ {}_{FD}\!\rightsquigarrow(ev \ e) \ P' \triangle Q\rangle$
⟨*proof*⟩

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

⟨*proof*⟩

Throw P A Q laws

lemma *Throw-τ-trans-left*:
‹$P\ {}_{FD}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ P'\ \Theta\ a \in A.\ Q\ a$›
⟨*proof*⟩

lemma *Throw-τ-trans-right*:
‹$\forall\, a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q'\ a \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow_{\tau}\ P\ \Theta\ a \in A.\ Q'\ a$›
⟨*proof*⟩

lemma *Throw-event-trans-left*:
‹$P\ {}_{FD}\!\rightsquigarrow e\ P' \Longrightarrow e \notin ev\ `\ A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow e\ (P'\ \Theta\ a \in A.\ Q\ a)$›
⟨*proof*⟩

lemma *Throw-ev-trans-right*:
‹$P\ {}_{FD}\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{FD}\!\rightsquigarrow(ev\ e)\ (Q\ e)$›
⟨*proof*⟩

lemma ‹*tickFree* $s \Longrightarrow \bot\ {}_{FD}\!\rightsquigarrow^{*} s\ P$›
⟨*proof*⟩

12.2 Reality Checks

lemma ‹$STOP\ {}_{FD}\!\rightsquigarrow^{*} s\ P \longleftrightarrow s = [] \wedge P = STOP$›
⟨*proof*⟩

lemma *SKIP-trace-trans-iff* :
‹$SKIP\ {}_{FD}\!\rightsquigarrow^{*} s\ P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP$›
⟨*proof*⟩

lemma *F-iff-exists-trans* :
‹$(s,\ X) \in \mathcal{F}\ P \longleftrightarrow (\exists\, P'.\ (P\ {}_{FD}\!\rightsquigarrow^{*} s\ P') \wedge X \in \mathcal{R}\ P')$›
⟨*proof*⟩

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

lemma *tickFree-imp-D-iff-trace-trans-BOT*:
‹*tickFree* $s \Longrightarrow s \in \mathcal{D}\ P \longleftrightarrow P\ {}_{FD}\!\rightsquigarrow^{*} s\ \bot$›
⟨*proof*⟩

lemma *D-iff-trace-trans-BOT*:
‹$s \in \mathcal{D}\ P \longleftrightarrow (\textit{if tickFree } s \textit{ then } P\ {}_{FD}\!\rightsquigarrow^{*} s\ \bot \textit{ else } P\ {}_{FD}\!\rightsquigarrow^{*}(\textit{butlast } s)\ \bot)$›

⟨*proof*⟩

12.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹$P\ {}_{FD}\!\rightsquigarrow^{*} s\ Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *trace-trans-imp-ready-set*:
‹$P\ {}_{FD}\!\rightsquigarrow^{*}(s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
‹$(P\ {}_{FD}\!\rightsquigarrow^{*} s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P$ *afterTrace* $s\ {}_{FD}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹*deadlock-free* $P \longleftrightarrow DF\ UNIV\ {}_{FD}\!\rightsquigarrow_{\tau}\ P$›
⟨*proof*⟩

lemma ‹*deadlock-free*$_{SKIP}$ $P \longleftrightarrow DF_{SKIP}\ UNIV\ {}_{FD}\!\rightsquigarrow_{\tau}\ P$›
⟨*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

$$\frac{?P\ {}_{FD}\!\rightsquigarrow ?e\ ?P' \qquad ?P'\ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P''}{?P\ {}_{FD}\!\rightsquigarrow ?e\ ?P''}$$

$$\frac{?P\ {}_{FD}\!\rightsquigarrow_{\tau}\ ?P' \qquad ?P'\ {}_{FD}\!\rightsquigarrow ?e\ ?P''}{?P\ {}_{FD}\!\rightsquigarrow ?e\ ?P''}$$

SKIP rule

$$\frac{}{SKIP\ {}_{FD}\!\rightsquigarrow\checkmark\ STOP}$$

$e \to P$ **rules**

$$\frac{?e \in ?A}{\Box a{\in}?A \to\ ?P\ a\ {}_{FD}{\rightsquigarrow} ev\ ?e\ ?P\ ?e} \qquad \frac{?e \in ?A}{\sqcap a{\in}?A \to\ \ ?P\ a\ {}_{FD}{\rightsquigarrow} ev\ ?e\ ?P\ ?e}$$

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

$(\sqcap)$ **rules**

$$\frac{}{?P \sqcap\ ?Q\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?P} \qquad \frac{}{?P \sqcap\ ?Q\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?Q}$$

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

$\mu\ x.\ f\ x$ **rule**

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

$(\Box)$ **rules**

$$\frac{?P\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?P'}{?P \Box\ ?Q\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?P' \Box\ ?Q} \qquad \frac{?Q\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?Q'}{?P \Box\ ?Q\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?P \Box\ ?Q'}$$

$$\frac{?P\ {}_{FD}{\rightsquigarrow} ?e\ ?P'}{?P \Box\ ?Q\ {}_{FD}{\rightsquigarrow} ?e\ ?P'} \qquad \frac{?Q\ {}_{FD}{\rightsquigarrow} ?e\ ?Q'}{?P \Box\ ?Q\ {}_{FD}{\rightsquigarrow} ?e\ ?Q'}$$

(;) **rules**

$$\frac{?P\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?P'}{?P\ ;\ ?Q\ {}_{FD}{\rightsquigarrow}_{\tau}\ ?P'\ ;\ ?Q} \qquad \frac{?P\ {}_{FD}{\rightsquigarrow} ev\ ?e\ ?P'}{?P\ ;\ ?Q\ {}_{FD}{\rightsquigarrow} ev\ ?e\ ?P'\ ;\ ?Q}$$

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

Hiding **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \setminus ?B} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow\checkmark \ ?P'}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow\checkmark \ STOP}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \notin ?B}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \setminus ?B} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \in ?B}{?P \setminus ?B \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \setminus ?B}$$

Renaming **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{Renaming \ ?P \ ?f \ {}_{FD}\!\rightsquigarrow_{\tau} \ Renaming \ ?P' \ ?f}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow\checkmark \ ?P'}{Renaming \ ?P \ ?f \ {}_{FD}\!\rightsquigarrow\checkmark \ STOP}$$

$$\frac{?f \ ?a = ?b \qquad ?P \ {}_{FD}\!\rightsquigarrow ev \ ?a \ ?P'}{Renaming \ ?P \ ?f \ {}_{FD}\!\rightsquigarrow ev \ ?b \ Renaming \ ?P' \ ?f}$$

Sync **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \ [\![?S]\!] \ ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \notin ?S \qquad ?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q}$$

$$\frac{?e \notin ?S \qquad ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \in ?S \qquad ?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q'}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow\checkmark \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP \ [\![?S]\!] \ ?Q}$$

$$\frac{?Q \ {}_{FD}\!\rightsquigarrow\checkmark \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ SKIP}$$

$$\frac{}{SKIP \ [\![?S]\!] \ SKIP \ {}_{FD}\!\rightsquigarrow_{\tau} \ SKIP}$$

($\triangleright$) **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangleright ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \triangleright ?Q} \qquad \frac{?P \ {}_{FD}\!\rightsquigarrow ?e \ ?P'}{?P \triangleright ?Q \ {}_{FD}\!\rightsquigarrow ?e \ ?P'}$$

$$\frac{}{?P \rhd ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q}$$

($\triangle$) **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangle ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \triangle ?Q \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \triangle ?Q'}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \triangle ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \triangle ?Q \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q'}$$

Throw **rules**

$$\frac{?P \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{\forall \, a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?Q' \ a}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow_{\tau} \ ?P \ \Theta \ a{\in}?A. \ ?Q' \ a}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ?e \ ?P' \qquad ?e \notin ev \ `\ ?A}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow ?e \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{?P \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \in ?A}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{FD}\!\rightsquigarrow ev \ ?e \ ?Q \ ?e}$$

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 *τ-trans* :: ‹'α *process* ⇒ 'α *process* ⇒ *bool*› (**infixl** ‹$_{DT}\!\rightsquigarrow_{\tau}$› *50*)
where ‹*P* $_{DT}\!\rightsquigarrow_{\tau}$ *Q* ≡ *P* $\sqsubseteq_{DT}$ *Q*›

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

interpretation *OpSemGeneric* ‹($_{DT}\!\rightsquigarrow_{\tau}$)›
⟨*proof*⟩

notation *event-trans* (‹-/ $_{DT}\!\rightsquigarrow$-/ -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹-/ $_{DT}\!\rightsquigarrow^{*}$-/ -› [*50*, *3*, *51*] *50*)

lemma ‹*P* $_{DT}\!\rightsquigarrow$ *e* *P'* ⟹ *P'* $_{DT}\!\rightsquigarrow_{\tau}$ *P''* ⟹ *P* $_{DT}\!\rightsquigarrow$ *e* *P''*›
‹*P* $_{DT}\!\rightsquigarrow_{\tau}$ *P'* ⟹ *P'* $_{DT}\!\rightsquigarrow$ *e* *P''* ⟹ *P* $_{DT}\!\rightsquigarrow$ *e* *P''*›
⟨*proof*⟩

13.1 Operational Semantics Laws

SKIP law

lemma ‹*SKIP* $_{DT}\!\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

$e \rightarrow P$ laws

lemma ‹*e* ∈ *A* ⟹ □*a* ∈ *A* → *P* *a* $_{DT}\!\rightsquigarrow$(*ev* *e*) (*P* *e*)›
⟨*proof*⟩

lemma ‹$e \in A \Longrightarrow \sqcap a \in A \rightarrow P\ a\ {}_{DT}\!\rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \rightarrow P\ {}_{DT}\!\rightsquigarrow\ (ev\ e)\ P$›
⟨*proof*⟩

$P \sqcap Q$ laws

lemma ‹$P \sqcap Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P$›
and ‹$P \sqcap Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

lemma ‹$a \in A \Longrightarrow (\sqcap a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ a$›
⟨*proof*⟩

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

lemma ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P\ {}_{DT}\!\rightsquigarrow_{\tau}\ f\ P$›
⟨*proof*⟩

$P \Box Q$ laws

lemma τ-*trans-DetL*: ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \Box Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Box Q$ ›
and τ-*trans-DetR*: ‹$Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P \Box Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P\ \Box\ Q'$›
⟨*proof*⟩

lemma τ-*trans-MultiDet*:
‹*finite* $A \Longrightarrow \forall\, a \in A.\ P\ a\ {}_{DT}\!\rightsquigarrow_{\tau}\ P'\ a \Longrightarrow$
$(\Box a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow_{\tau}\ (\Box a \in A.\ P'\ a)$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\rightsquigarrow e\ P' \Longrightarrow P \Box Q\ {}_{DT}\!\rightsquigarrow e\ P'$›
and ‹$Q\ {}_{DT}\!\rightsquigarrow e\ Q' \Longrightarrow P \Box Q\ {}_{DT}\!\rightsquigarrow e\ Q'$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow P\ a\ {}_{DT}\!\rightsquigarrow e\ Q \Longrightarrow (\Box a \in A.\ P\ a)\ {}_{DT}\!\rightsquigarrow e\ Q$›
⟨*proof*⟩

P **;** Q laws

lemma τ-*trans-SeqL*: ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ ;\ Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P'\ ;\ Q$›
⟨*proof*⟩

lemma *ev-trans-SeqL*: ‹$P\ {}_{DT}\!\rightsquigarrow (ev\ e)\ P' \Longrightarrow P\ ;\ Q\ {}_{DT}\!\rightsquigarrow (ev\ e)\ P'\ ;\ Q$›

⟨*proof*⟩

lemma τ-*trans-SeqR*: ‹$\exists\, P'.\ P\ {}_{DT}\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ ;\ Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q$›

⟨*proof*⟩

lemma ‹✓ ∈ *ready-set* $P \Longrightarrow Q\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ Q' \Longrightarrow P\ ;;\ Q\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ Q'$›

⟨*proof*⟩

$P \setminus B$ laws

lemma *τ-trans-Hiding*: ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \setminus B\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

lemma *ev-trans-Hiding-notin*:
‹$P\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ P' \Longrightarrow e \notin B \Longrightarrow P \setminus B\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ P' \setminus B$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\rightsquigarrow$✓ $P' \Longrightarrow P \setminus B\ {}_{DT}\!\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

lemma *ev-trans-Hiding-inside*:
‹$P\ {}_{DT}\!\rightsquigarrow\!(ev\ e)\ P' \Longrightarrow e \in B \Longrightarrow P \setminus B\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \setminus B$›
⟨*proof*⟩

Renaming P f laws

lemma *τ-trans-Renaming*:
‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow$ *Renaming P f* $\ {}_{DT}\!\rightsquigarrow_{\tau}$ *Renaming P' f*›
⟨*proof*⟩

lemma *tick-trans-Renaming*: ‹$P\ {}_{DT}\!\rightsquigarrow$✓ $P' \Longrightarrow$ *Renaming P f* $\ {}_{DT}\!\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

lemma *ev-trans-Renaming*:
‹$f\ a = b \Longrightarrow P\ {}_{DT}\!\rightsquigarrow\!(ev\ a)\ P' \Longrightarrow$ *Renaming P f* $\ {}_{DT}\!\rightsquigarrow\!(ev\ b)$ (*Renaming P' f*)›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{DT}\!\rightsquigarrow_{\tau}\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\rightsquigarrow$✓ $P' \Longrightarrow P\ [\![a := b]\!]\ {}_{DT}\!\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

lemma *ev-trans-RenamingF*:
‹$P\ {}_{DT}\!\rightsquigarrow\!(ev\ a)\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_{DT}\!\rightsquigarrow\!(ev\ b)\ P'\ [\![a := b]\!]$›
⟨*proof*⟩

$P\ [\![S]\!]\ Q$ laws

lemma *τ-trans-SyncL*: ‹$P\ {}_{DT}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ P'\ [\![S]\!]\ Q$›

and *τ-trans-SyncR*: ‹$Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P\ [\![S]\!]\ Q'$›
⟨*proof*⟩

lemma *ev-trans-SyncL*:
‹$e \notin S \Longrightarrow P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P'\ [\![S]\!]\ Q$ ›
and *ev-trans-SyncR*:
‹$e \notin S \Longrightarrow Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P\ \ [\![S]\!]\ Q'$›
⟨*proof*⟩

lemma *ev-trans-SyncLR*:
‹$[\![e \in S;\ P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P';\ Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ Q']\!] \Longrightarrow$
$P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P'\ [\![S]\!]\ Q'$›
⟨*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*: ‹$P\ {}_{DT}\!\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ SKIP\ [\![S]\!]\ Q$›
and *tick-trans-SyncR*: ‹$Q\ {}_{DT}\!\!\rightsquigarrow\checkmark\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P\ [\![S]\!]\ SKIP$›
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: ‹$SKIP\ [\![S]\!]\ SKIP\ {}_{DT}\!\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *τ-trans-SKIP-Sync-SKIP*: ‹$SKIP\ [\![S]\!]\ SKIP\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ SKIP$›
⟨*proof*⟩

$P \rhd Q$ laws

lemma *Sliding-τ-trans-left*: ‹$P\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \rhd Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \rhd Q$›
⟨*proof*⟩

lemma ‹$P\ {}_{DT}\!\!\rightsquigarrow e\ P' \Longrightarrow P \rhd Q\ {}_{DT}\!\!\rightsquigarrow e\ P'$›
⟨*proof*⟩

lemma ‹$P \rhd Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ Q$›
⟨*proof*⟩

$P \triangle Q$ laws

lemma *Interrupt-τ-trans-left*: ‹$P\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P \triangle Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \triangle Q$›
⟨*proof*⟩

lemma *Interrupt-τ-trans-right*: ‹$Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P \triangle Q\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P \triangle Q'$›
⟨*proof*⟩

lemma *Interrupt-ev-trans-left*:
‹$P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow P \triangle Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \triangle Q$›
⟨*proof*⟩

lemma *Interrupt-ev-trans-right*: ‹$Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P \triangle Q\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ Q'$›
⟨*proof*⟩

Throw P A Q laws

lemma *Throw-τ-trans-left*:
‹$P\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P'\ \Theta\ a \in A.\ Q\ a$›
⟨*proof*⟩

lemma *Throw-τ-trans-right*:
‹$\forall\, a \in A.\ Q\ a\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ Q'\ a \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\!\rightsquigarrow_{\tau}\ P\ \Theta\ a \in A.\ Q'\ a$›
⟨*proof*⟩

lemma *Throw-event-trans-left*:
‹$P\ {}_{DT}\!\!\rightsquigarrow e\ P' \Longrightarrow e \notin ev\ `\ A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\!\rightsquigarrow e\ (P'\ \Theta\ a \in A.\ Q\ a)$›
⟨*proof*⟩

lemma *Throw-ev-trans-right*:
‹$P\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_{DT}\!\!\rightsquigarrow(ev\ e)\ (Q\ e)$›
⟨*proof*⟩

lemma ‹*tickFree s* $\Longrightarrow \bot\ {}_{DT}\!\!\rightsquigarrow^{*} s\ P$›
⟨*proof*⟩

13.2 Reality Checks

lemma ‹$STOP\ {}_{DT}\!\!\rightsquigarrow^{*} s\ P \longleftrightarrow s = [] \wedge P = STOP$›
⟨*proof*⟩

lemma *T-iff-exists-trans* : ‹$s \in \mathcal{T}\ P \longleftrightarrow (\exists\, P'.\ P\ {}_{DT}\!\!\rightsquigarrow^{*} s\ P')$›
⟨*proof*⟩

lemma *tickFree-imp-D-iff-trace-trans-BOT*:
‹*tickFree s* $\Longrightarrow s \in \mathcal{D}\ P \longleftrightarrow P\ {}_{DT}\!\!\rightsquigarrow^{*} s\ \bot$›
⟨*proof*⟩

lemma *D-iff-trace-trans-BOT*:
‹$s \in \mathcal{D}\ P \longleftrightarrow$ (*if tickFree s then* $P\ {}_{DT}\!\!\rightsquigarrow^{*} s\ \bot$ *else* $P\ {}_{DT}\!\!\rightsquigarrow^{*}$(*butlast s*) $\bot$)›
⟨*proof*⟩

13.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹$P\ {}_{DT}\!\!\rightsquigarrow^{*} s\ Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* (P *afterTrace s*)›

⟨*proof*⟩

lemma *trace-trans-imp-ready-set*:
‹$P\ {}_{DT}\!\rightsquigarrow^{*}(s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
‹$(P\ {}_{DT}\!\rightsquigarrow^{*}s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P$ *afterTrace* $s\ {}_{DT}\!\rightsquigarrow_{\tau}\ Q$›
⟨*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

$$\frac{?P\ {}_{DT}\!\rightsquigarrow ?e\ ?P' \qquad ?P'\ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P''}{?P\ {}_{DT}\!\rightsquigarrow ?e\ ?P''}$$

$$\frac{?P\ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P' \qquad ?P'\ {}_{DT}\!\rightsquigarrow ?e\ ?P''}{?P\ {}_{DT}\!\rightsquigarrow ?e\ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP\ {}_{DT}\!\rightsquigarrow\checkmark\ STOP}$$

$e \rightarrow P$ **rules**

$$\frac{?e \in ?A}{\Box a{\in}?A \rightarrow ?P\ a\ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P\ ?e} \qquad \frac{?e \in ?A}{\sqcap a{\in}?A \rightarrow\ ?P\ a\ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P\ ?e}$$

$$\frac{}{?e \rightarrow ?P\ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P}$$

$(\sqcap)$ **rules**

$$\frac{}{?P \sqcap ?Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P} \qquad \frac{}{?P \sqcap ?Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ ?Q}$$

$$\frac{?e \in ?A}{\sqcap\ a{\in}?A.\ \ ?P\ a\ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P\ ?e}$$

$\mu\ x.\ f\ x$ **rule**

$$\frac{cont\ ?f \qquad ?P = (\mu\ x.\ ?f\ x)}{?P\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?f\ ?P}$$

($\Box$) **rules (more powerful than in OpSemFD)**

$$\frac{?P\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P'}{?P \Box ?Q\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P' \Box ?Q} \qquad \frac{?Q\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?Q'}{?P \Box ?Q\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P \Box ?Q'}$$

(;) **rules**

$$\frac{?P\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P'}{?P\ ;\ ?Q\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P'\ ;\ ?Q} \qquad \frac{?P\ \ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P'}{?P\ ;\ ?Q\ \ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P'\ ;\ ?Q}$$

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

Hiding **rules**

$$\frac{?P\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P'}{?P \setminus ?B\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P' \setminus ?B} \qquad \frac{?P\ \ {}_{DT}\!\rightsquigarrow\checkmark\ ?P'}{?P \setminus ?B\ \ {}_{DT}\!\rightsquigarrow\checkmark\ STOP}$$

$$\frac{?P\ \ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \notin ?B}{?P \setminus ?B\ \ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P' \setminus ?B} \qquad \frac{?P\ \ {}_{DT}\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \in ?B}{?P \setminus ?B\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P' \setminus ?B}$$

Renaming **rules**

$$\frac{?P\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ ?P'}{Renaming\ ?P\ ?f\ \ {}_{DT}\!\rightsquigarrow_{\tau}\ Renaming\ ?P'\ ?f}$$

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

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

Sync **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \ [\![?S]\!] \ ?Q} \qquad \frac{?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \notin ?S \qquad ?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q}$$

$$\frac{?e \notin ?S \qquad ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P \ [\![?S]\!] \ ?Q'}$$

$$\frac{?e \in ?S \qquad ?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \ [\![?S]\!] \ ?Q'}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow \checkmark \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ SKIP \ [\![?S]\!] \ ?Q}$$

$$\frac{?Q \ {}_{DT}\!\rightsquigarrow \checkmark \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ [\![?S]\!] \ SKIP}$$

$$\frac{}{SKIP \ [\![?S]\!] \ SKIP \ {}_{DT}\!\rightsquigarrow_{\tau} \ SKIP}$$

($\triangleright$) **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangleright ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \triangleright ?Q} \qquad \frac{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P'}{?P \triangleright ?Q \ {}_{DT}\!\rightsquigarrow ?e \ ?P'}$$

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

($\triangle$) **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \triangle ?Q'}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}{?P \triangle ?Q \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q'}$$

Throw **rules**

$$\frac{?P \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P'}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P' \ \Theta \ a{\in}?A. \ ?Q \ a}$$

$$\frac{\forall \, a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?Q' \ a}{?P \ \Theta \ a{\in}?A. \ ?Q \ a \ {}_{DT}\!\rightsquigarrow_{\tau} \ ?P \ \Theta \ a{\in}?A. \ ?Q' \ a}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ?e \ ?P' \qquad ?e \notin ev \ `\ ?A}{?P \ \Theta \ a{\in}?A.\ ?Q \ a \ {}_{DT}\!\rightsquigarrow ?e \ ?P' \ \Theta \ a{\in}?A.\ ?Q \ a}$$

$$\frac{?P \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?P' \qquad ?e \in ?A}{?P \ \Theta \ a{\in}?A.\ ?Q \ a \ {}_{DT}\!\rightsquigarrow ev \ ?e \ ?Q \ ?e}$$

end

Chapter 14

Failure Operational Semantics, bis

theory *OpSemFBis*
imports *OpSemGenericBis HOL−Library.LaTeXsugar*
begin

The definition with ($\sqsubseteq_{FD}$) motivates us to try with other refinements.

abbreviation *τ-trans* :: ‹'α process ⇒ 'α process ⇒ bool› (**infixl** ‹$_F\!\rightsquigarrow_\tau$› 50)
where ‹$P\ {}_F\!\rightsquigarrow_\tau\ Q \equiv P \sqsubseteq_F Q$›

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

interpretation *OpSemGeneric* ‹($_F\!\rightsquigarrow_\tau$) :: 'α process ⇒ 'α process ⇒ bool›
⟨*proof*⟩

notation *event-trans* (‹-/ $_F\!\rightsquigarrow$-/ -› [50, 3, 51] 50)
notation *trace-trans* (‹-/ $_F\!\rightsquigarrow^*$-/ -› [50, 3, 51] 50)

lemma ‹$P\ {}_F\!\rightsquigarrow\ e\ P' \Longrightarrow P'\ {}_F\!\rightsquigarrow_\tau\ P'' \Longrightarrow P\ {}_F\!\rightsquigarrow\ e\ P''$›
‹$P\ {}_F\!\rightsquigarrow_\tau\ P' \Longrightarrow P'\ {}_F\!\rightsquigarrow\ e\ P'' \Longrightarrow P\ {}_F\!\rightsquigarrow\ e\ P''$›
⟨*proof*⟩

14.1 Operational Semantics Laws

SKIP law

lemma ‹$SKIP\ {}_F\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

$e \rightarrow P$ laws

lemma ‹$e \in A \Longrightarrow \Box a \in A \rightarrow P\ a\ {}_F\!\rightsquigarrow(ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \in A \Longrightarrow \sqcap a \in A \to P\ a\ {}_F\!\rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \to P\ {}_F\!\rightsquigarrow\ (ev\ e)\ P$›
⟨*proof*⟩

$P \sqcap Q$ laws

lemma ‹$P \sqcap Q\ {}_F\!\rightsquigarrow_\tau\ P$›
and ‹$P \sqcap Q\ {}_F\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

lemma ‹$a \in A \Longrightarrow (\sqcap a \in A.\ P\ a)\ {}_F\!\rightsquigarrow_\tau\ P\ a$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a)\ {}_F\!\rightsquigarrow_\tau\ P\ a$›
⟨*proof*⟩

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

lemma ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P\ {}_F\!\rightsquigarrow_\tau\ f\ P$›
⟨*proof*⟩

$P \mathbin{\square} Q$ laws

lemma ‹$P\ {}_F\!\rightsquigarrow e\ P' \Longrightarrow P \mathbin{\square} Q\ {}_F\!\rightsquigarrow e\ P'$›
and ‹$Q\ {}_F\!\rightsquigarrow e\ Q' \Longrightarrow P \mathbin{\square} Q\ {}_F\!\rightsquigarrow e\ Q'$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow P\ a\ {}_F\!\rightsquigarrow e\ Q \Longrightarrow (\square a \in A.\ P\ a)\ {}_F\!\rightsquigarrow e\ Q$›
⟨*proof*⟩

$P \mathbin{;} Q$ laws

lemma *τ-trans-SeqR*: ‹$\exists P'.\ P\ {}_F\!\rightsquigarrow\checkmark\ P' \Longrightarrow P \mathbin{;} Q\ {}_F\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩
lemma ‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q\ {}_F\!\rightsquigarrow (ev\ e)\ Q' \Longrightarrow P \mathbin{;} Q\ {}_F\!\rightsquigarrow (ev\ e)\ Q'$›

⟨*proof*⟩

$P \setminus B$ laws

lemma *τ-trans-Hiding*: ‹$P\ {}_F\!\rightsquigarrow_\tau\ P' \Longrightarrow P \setminus B\ {}_F\!\rightsquigarrow_\tau\ P' \setminus B$›
⟨*proof*⟩

lemma *ev-trans-Hiding-notin*:
‹$P\ {}_F\!\rightsquigarrow (ev\ e)\ P' \Longrightarrow e \notin B \Longrightarrow P \setminus B\ {}_F\!\rightsquigarrow (ev\ e)\ P' \setminus B$›
⟨*proof*⟩

lemma ‹$P\ {}_F\!\rightsquigarrow\checkmark\ P' \Longrightarrow P \setminus B\ {}_F\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Hiding-inside*:
‹$P\ {}_F\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in B \Longrightarrow P \setminus B\ {}_F\!\rightsquigarrow_\tau\ P' \setminus B$›
⟨*proof*⟩

Renaming $P\ f$ laws

lemma *tick-trans-Renaming*: ‹$P\ {}_F\!\rightsquigarrow\checkmark\ P' \Longrightarrow Renaming\ P\ f\ {}_F\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma ‹$P\ {}_F\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_F\!\rightsquigarrow\checkmark\ STOP$›
⟨*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*: ‹$P\ {}_F\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_F\!\rightsquigarrow_\tau\ SKIP\ [\![S]\!]\ Q$›
and *tick-trans-SyncR*: ‹$Q\ {}_F\!\rightsquigarrow\checkmark\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_F\!\rightsquigarrow_\tau\ P\ [\![S]\!]\ SKIP$›
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: ‹$SKIP\ [\![S]\!]\ SKIP\ {}_F\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *τ-trans-SKIP-Sync-SKIP*: ‹$SKIP\ [\![S]\!]\ SKIP\ {}_F\!\rightsquigarrow_\tau\ SKIP$›
⟨*proof*⟩

$P \rhd Q$ laws

lemma ‹$P\ {}_F\!\rightsquigarrow e\ P' \Longrightarrow P \rhd Q\ {}_F\!\rightsquigarrow e\ P'$›
⟨*proof*⟩

lemma ‹$P \rhd Q\ {}_F\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

$P \triangle Q$ laws

lemma *Interrupt-ev-trans-right*: ‹$Q\ {}_F\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P \triangle Q\ {}_F\!\rightsquigarrow(ev\ e)\ Q'$›
⟨*proof*⟩

Throw $P\ A\ Q$ laws

lemma *Throw-τ-trans-right*:
‹$\forall a \in A.\ Q\ a\ {}_F\!\rightsquigarrow_\tau\ Q'\ a \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_F\!\rightsquigarrow_\tau\ P\ \Theta\ a \in A.\ Q'\ a$›
⟨*proof*⟩

lemma *Throw-ev-trans-right*:
‹$P\ {}_F\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_F\!\rightsquigarrow(ev\ e)\ (Q\ e)$›
⟨*proof*⟩

lemma ‹*tickFree* $s \Longrightarrow \bot\ {}_F\!\rightsquigarrow^* s\ P$›
⟨*proof*⟩

14.2 Reality Checks

lemma ‹$STOP\ {}_F\!\rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = STOP$›
⟨*proof*⟩

lemma *SKIP-trace-trans-iff* :
‹$SKIP\ {}_F\!\rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = SKIP \vee s = [\checkmark] \wedge P = STOP$›
⟨*proof*⟩

lemma *F-iff-exists-trans* :
‹$(s,\ X) \in \mathcal{F}\ P \longleftrightarrow (\exists\, P'.\ (P\ {}_F\!\rightsquigarrow^* s\ P') \wedge X \in \mathcal{R}\ P')$›
⟨*proof*⟩

lemma *T-iff-exists-trans* : ‹$s \in \mathcal{T}\ P \longleftrightarrow (\exists\, P'.\ P\ {}_F\!\rightsquigarrow^* s\ P')$›
⟨*proof*⟩

14.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹$P\ {}_F\!\rightsquigarrow^* s\ Q \Longrightarrow ready\text{-}set\ Q \subseteq ready\text{-}set\ (P\ afterTrace\ s)$›
⟨*proof*⟩

lemma *trace-trans-imp-ready-set*:
‹$P\ {}_F\!\rightsquigarrow^*(s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in ready\text{-}set\ (P\ afterTrace\ s)$›
⟨*proof*⟩

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
‹$(P\ {}_F\!\rightsquigarrow^* s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P\ afterTrace\ s\ {}_F\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

lemma ‹$deadlock\text{-}free\ P \longleftrightarrow DF\ UNIV\ {}_F\!\rightsquigarrow_\tau\ P$›
⟨*proof*⟩

lemma ‹$deadlock\text{-}free_{SKIP}\ P \longleftrightarrow DF_{SKIP}\ UNIV\ {}_F\!\rightsquigarrow_\tau\ P$›
⟨*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

$$\frac{?P \ {}_F\!\rightsquigarrow ?e \ ?P' \qquad ?P' \ {}_F\!\rightsquigarrow_\tau \ ?P''}{?P \ {}_F\!\rightsquigarrow ?e \ ?P''} \qquad \frac{?P \ {}_F\!\rightsquigarrow_\tau \ ?P' \qquad ?P' \ {}_F\!\rightsquigarrow ?e \ ?P''}{?P \ {}_F\!\rightsquigarrow ?e \ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP \ {}_F\!\rightsquigarrow \checkmark \ STOP}$$

$e \rightarrow P$ **rules**

$$\frac{?e \in ?A}{\Box a{\in}?A \rightarrow \ ?P \ a \ {}_F\!\rightsquigarrow ev \ ?e \ ?P \ ?e} \qquad \frac{?e \in ?A}{\sqcap a{\in}?A \rightarrow \ \ ?P \ a \ {}_F\!\rightsquigarrow ev \ ?e \ ?P \ ?e}$$

$$\frac{}{?e \rightarrow \ ?P \ {}_F\!\rightsquigarrow ev \ ?e \ ?P}$$

$(\sqcap)$ **rules**

$$\frac{}{?P \sqcap \ ?Q \ {}_F\!\rightsquigarrow_\tau \ ?P} \qquad \frac{}{?P \sqcap \ ?Q \ {}_F\!\rightsquigarrow_\tau \ ?Q}$$

$$\frac{?e \in \ ?A}{\sqcap \ a{\in}?A. \ \ ?P \ a \ {}_F\!\rightsquigarrow_\tau \ ?P \ ?e}$$

$\mu \ x. \ f \ x$ **rule**

$$\frac{cont \ ?f \qquad ?P = (\mu \ x. \ ?f \ x)}{?P \ {}_F\!\rightsquigarrow_\tau \ ?f \ ?P}$$

$(\Box)$ **rules**

$$\frac{?P \ {}_F\!\rightsquigarrow ?e \ ?P'}{?P \ \Box \ ?Q \ {}_F\!\rightsquigarrow ?e \ ?P'} \qquad \frac{?Q \ {}_F\!\rightsquigarrow ?e \ ?Q'}{?P \ \Box \ ?Q \ {}_F\!\rightsquigarrow ?e \ ?Q'}$$

(;) **rule**

$$\frac{\exists\, P'.\ ?P \ {}_F\!\rightsquigarrow\!\checkmark\ P'}{?P \ ; \ ?Q \ {}_F\!\rightsquigarrow_\tau\ ?Q}$$

Hiding **rules**

$$\frac{?P \ {}_F\!\rightsquigarrow_\tau\ ?P'}{?P \setminus ?B \ {}_F\!\rightsquigarrow_\tau\ ?P' \setminus ?B} \qquad \frac{?P \ {}_F\!\rightsquigarrow\!\checkmark\ ?P'}{?P \setminus ?B \ {}_F\!\rightsquigarrow\!\checkmark\ STOP}$$

$$\frac{?P \ {}_F\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \notin ?B}{?P \setminus ?B \ {}_F\!\rightsquigarrow ev\ ?e\ ?P' \setminus ?B} \qquad \frac{?P \ {}_F\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \in ?B}{?P \setminus ?B \ {}_F\!\rightsquigarrow_\tau\ ?P' \setminus ?B}$$

Renaming **rule**

$$\frac{?P \ {}_F\!\rightsquigarrow\!\checkmark\ ?P'}{Renaming\ ?P\ ?f \ {}_F\!\rightsquigarrow\!\checkmark\ STOP}$$

Sync **rules**

$$\frac{?P \ {}_F\!\rightsquigarrow\!\checkmark\ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_F\!\rightsquigarrow_\tau\ SKIP \ [\![?S]\!] \ ?Q} \qquad \frac{?Q \ {}_F\!\rightsquigarrow\!\checkmark\ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_F\!\rightsquigarrow_\tau\ ?P \ [\![?S]\!] \ SKIP}$$

$$\frac{}{SKIP \ [\![?S]\!] \ SKIP \ {}_F\!\rightsquigarrow_\tau\ SKIP}$$

($\triangleright$) **rules**

$$\frac{?P \ {}_F\!\rightsquigarrow ?e\ ?P'}{?P \triangleright ?Q \ {}_F\!\rightsquigarrow ?e\ ?P'}$$

$$\frac{}{?P \triangleright ?Q \ {}_F\!\rightsquigarrow_\tau\ ?Q}$$

($\triangle$) **rule**

$$\frac{?Q \ {}_F\!\rightsquigarrow ev\ ?e\ ?Q'}{?P \triangle ?Q \ {}_F\!\rightsquigarrow ev\ ?e\ ?Q'}$$

Throw **rules**

$$\frac{\forall\, a{\in}?A.\ ?Q\ a\ {}_F{\rightsquigarrow}_\tau\ ?Q'\ a}{?P\ \Theta\ a{\in}?A.\ ?Q\ a\ {}_F{\rightsquigarrow}_\tau\ ?P\ \Theta\ a{\in}?A.\ ?Q'\ a}$$

$$\frac{?P\ {}_F{\rightsquigarrow}ev\ ?e\ ?P' \qquad ?e \in ?A}{?P\ \Theta\ a{\in}?A.\ ?Q\ a\ {}_F{\rightsquigarrow}ev\ ?e\ ?Q\ ?e}$$

end

Chapter 15

Trace Operational Semantics, bis

theory *OpSemTBis*
imports *OpSemGenericBis HOL−Library.LaTeXsugar*
begin

The definition with ($\sqsubseteq_{FD}$) motivates us to try with other refinements.

abbreviation *τ-trans* :: ‹'α *process* ⇒ 'α *process* ⇒ *bool*› (**infixl** ‹$_T\rightsquigarrow_\tau$› *50*)
where ‹*P* $_T\rightsquigarrow_\tau$ *Q* ≡ *P* $\sqsubseteq_T$ *Q*›

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

interpretation *OpSemGeneric* ‹($_T\rightsquigarrow_\tau$)›
⟨*proof*⟩

notation *event-trans* (‹-/ $_T\rightsquigarrow$-/ -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹-/ $_T\rightsquigarrow^*$-/ -› [*50*, *3*, *51*] *50*)

lemma ‹*P* $_T\rightsquigarrow$ *e P'* ⟹ *P'* $_T\rightsquigarrow_\tau$ *P''* ⟹ *P* $_T\rightsquigarrow$ *e P''*›
‹*P* $_T\rightsquigarrow_\tau$ *P'* ⟹ *P'* $_T\rightsquigarrow$ *e P''* ⟹ *P* $_T\rightsquigarrow$ *e P''*›
⟨*proof*⟩

15.1 Operational Semantics Laws

SKIP law

lemma ‹*SKIP* $_T\rightsquigarrow$✓ *STOP*›
⟨*proof*⟩

$e \rightarrow P$ laws

lemma ‹*e* ∈ *A* ⟹ □*a* ∈ *A* → *P a* $_T\rightsquigarrow$(*ev e*) (*P e*)›
⟨*proof*⟩

lemma ‹$e \in A \Longrightarrow \sqcap a \in A \rightarrow P\ a\ {}_T\!\rightsquigarrow (ev\ e)\ (P\ e)$›
⟨*proof*⟩

lemma ‹$e \rightarrow P\ {}_T\!\rightsquigarrow\ (ev\ e)\ P$›
⟨*proof*⟩

$P \sqcap Q$ laws

lemma ‹$P \sqcap Q\ {}_T\!\rightsquigarrow_\tau\ P$›
and ‹$P \sqcap Q\ {}_T\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

lemma ‹$a \in A \Longrightarrow (\sqcap a \in A.\ P\ a)\ {}_T\!\rightsquigarrow_\tau\ P\ a$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow (\bigsqcap a \in A.\ P\ a)\ {}_T\!\rightsquigarrow_\tau\ P\ a$›
⟨*proof*⟩

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

lemma ‹*cont* $f \Longrightarrow P = (\mu\ X.\ f\ X) \Longrightarrow P\ {}_T\!\rightsquigarrow_\tau\ f\ P$›
⟨*proof*⟩

$P \Box Q$ laws

lemma τ-*trans-DetL*: ‹$P\ {}_T\!\rightsquigarrow_\tau\ P' \Longrightarrow P \Box Q\ {}_T\!\rightsquigarrow_\tau\ P' \Box Q$›
and τ-*trans-DetR*: ‹$Q\ {}_T\!\rightsquigarrow_\tau\ Q' \Longrightarrow P \Box Q\ {}_T\!\rightsquigarrow_\tau\ P \Box Q'$›
⟨*proof*⟩

lemma τ-*trans-MultiDet*:
‹*finite* $A \Longrightarrow \forall\, a \in A.\ P\ a\ {}_T\!\rightsquigarrow_\tau\ P'\ a \Longrightarrow (\Box a \in A.\ P\ a)\ {}_T\!\rightsquigarrow_\tau\ (\Box a \in A.\ P'\ a)$›
⟨*proof*⟩

lemma ‹$P\ {}_T\!\rightsquigarrow e\ P' \Longrightarrow P \Box Q\ {}_T\!\rightsquigarrow e\ P'$›
and ‹$Q\ {}_T\!\rightsquigarrow e\ Q' \Longrightarrow P \Box Q\ {}_T\!\rightsquigarrow e\ Q'$›
⟨*proof*⟩

lemma ‹*finite* $A \Longrightarrow a \in A \Longrightarrow P\ a\ {}_T\!\rightsquigarrow e\ Q \Longrightarrow (\Box a \in A.\ P\ a)\ {}_T\!\rightsquigarrow e\ Q$›
⟨*proof*⟩

P **;** Q laws

lemma τ-*trans-SeqR*: ‹$\exists\, P'.\ P\ {}_T\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ ;\ Q\ {}_T\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

lemma ‹$Q\ {}_T\!\rightsquigarrow_\tau\ Q' \Longrightarrow P\ ;\ Q\ {}_T\!\rightsquigarrow_\tau\ P\ ;\ Q'$›
⟨*proof*⟩

lemma ‹$\checkmark \in$ *ready-set* $P \Longrightarrow Q\ {}_T\!\rightsquigarrow (ev\ e)\ Q' \Longrightarrow P\ ;\ Q\ {}_T\!\rightsquigarrow (ev\ e)\ Q'$›

⟨*proof*⟩

$P \setminus B$ laws

lemma *τ-trans-Hiding*: ‹$P\ {}_T\!\rightsquigarrow_\tau\ P' \Longrightarrow P \setminus B\ {}_T\!\rightsquigarrow_\tau\ P' \setminus B$›
⟨*proof*⟩

lemma *ev-trans-Hiding-notin*:
‹$P\ {}_T\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \notin B \Longrightarrow P \setminus B\ {}_T\!\rightsquigarrow(ev\ e)\ P' \setminus B$›
⟨*proof*⟩

lemma ‹$P\ {}_T\!\rightsquigarrow\checkmark\ P' \Longrightarrow P \setminus B\ {}_T\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *ev-trans-Hiding-inside*:
‹$P\ {}_T\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in B \Longrightarrow P \setminus B\ {}_T\!\rightsquigarrow_\tau\ P' \setminus B$›
⟨*proof*⟩

Renaming P f laws

lemma *tick-trans-Renaming*: ‹$P\ {}_T\!\rightsquigarrow\checkmark\ P' \Longrightarrow Renaming\ P\ f\ {}_T\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma ‹$P\ {}_T\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ [\![a := b]\!]\ {}_T\!\rightsquigarrow\checkmark\ STOP$›
⟨*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*: ‹$P\ {}_T\!\rightsquigarrow\checkmark\ P' \Longrightarrow P\ [\![S]\!]\ Q\ {}_T\!\rightsquigarrow_\tau\ SKIP\ [\![S]\!]\ Q$›
and *tick-trans-SyncR*: ‹$Q\ {}_T\!\rightsquigarrow\checkmark\ Q' \Longrightarrow P\ [\![S]\!]\ Q\ {}_T\!\rightsquigarrow_\tau\ P\ [\![S]\!]\ SKIP$›
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: ‹$SKIP\ [\![S]\!]\ SKIP\ {}_T\!\rightsquigarrow\checkmark\ STOP$›
⟨*proof*⟩

lemma *τ-trans-SKIP-Sync-SKIP*: ‹$SKIP\ [\![S]\!]\ SKIP\ {}_T\!\rightsquigarrow_\tau\ SKIP$›
⟨*proof*⟩

$P \rhd Q$ laws

lemma *Sliding-τ-trans-left*: ‹$P\ {}_T\!\rightsquigarrow_\tau\ P' \Longrightarrow P \rhd Q\ {}_T\!\rightsquigarrow_\tau\ P' \rhd Q$›
⟨*proof*⟩

lemma ‹$P\ {}_T\!\rightsquigarrow e\ P' \Longrightarrow P \rhd Q\ {}_T\!\rightsquigarrow e\ P'$›
⟨*proof*⟩

lemma ‹$P \rhd Q\ {}_T\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

$P \triangle Q$ laws

lemma *Interrupt-τ-trans-left*: ‹$P\ {}_T\!\rightsquigarrow_\tau\ P' \Longrightarrow P \triangle Q\ {}_T\!\rightsquigarrow_\tau\ P' \triangle Q$›
⟨*proof*⟩

lemma *Interrupt-τ-trans-right*: ‹$Q\ {}_T\!\rightsquigarrow_\tau\ Q' \Longrightarrow P \triangle Q\ {}_T\!\rightsquigarrow_\tau\ P \triangle Q'$›
⟨*proof*⟩

lemma *Interrupt-ev-trans-left*: ‹$P\ {}_T\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow P \triangle Q\ {}_T\!\rightsquigarrow(ev\ e)\ P' \triangle Q$›
⟨*proof*⟩

lemma *Interrupt-ev-trans-right*: ‹$Q\ {}_T\!\rightsquigarrow(ev\ e)\ Q' \Longrightarrow P \triangle Q\ {}_T\!\rightsquigarrow(ev\ e)\ Q'$›
⟨*proof*⟩

Throw P A Q laws

lemma *Throw-τ-trans-right*:
‹$\forall a \in A.\ Q\ a\ {}_T\!\rightsquigarrow_\tau\ Q'\ a \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_T\!\rightsquigarrow_\tau\ P\ \Theta\ a \in A.\ Q'\ a$›
⟨*proof*⟩

lemma *Throw-ev-trans-right*:
‹$P\ {}_T\!\rightsquigarrow(ev\ e)\ P' \Longrightarrow e \in A \Longrightarrow P\ \Theta\ a \in A.\ Q\ a\ {}_T\!\rightsquigarrow(ev\ e)\ (Q\ e)$›
⟨*proof*⟩

lemma ‹*tickFree* $s \Longrightarrow \bot\ {}_T\!\rightsquigarrow^* s\ P$›
⟨*proof*⟩

15.2 Reality Checks

lemma ‹$STOP\ {}_T\!\rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = STOP$›
⟨*proof*⟩

lemma *T-iff-exists-trans* : ‹$s \in \mathcal{T}\ P \longleftrightarrow (\exists P'.\ P\ {}_T\!\rightsquigarrow^* s\ P')$›
⟨*proof*⟩

15.3 Other Results

lemma *trace-trans-ready-set-subset-ready-set-AfterTrace*:
‹$P\ {}_T\!\rightsquigarrow^*\! s\ Q \Longrightarrow$ *ready-set* $Q \subseteq$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *trace-trans-imp-ready-set*:
‹$P\ {}_T\!\rightsquigarrow^*\!(s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in$ *ready-set* $(P$ *afterTrace* $s)$›
⟨*proof*⟩

lemma *AfterTrace-τ-trans-if-τ-trans-imp-leT* :
‹$(P\ {}_T\!\rightsquigarrow^*\! s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P$ *afterTrace* $s\ {}_T\!\rightsquigarrow_\tau\ Q$›
⟨*proof*⟩

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

$$\frac{?P\ {}_T\!\rightsquigarrow ?e\ ?P' \qquad ?P'\ {}_T\!\rightsquigarrow_\tau\ ?P''}{?P\ {}_T\!\rightsquigarrow ?e\ ?P''} \qquad \frac{?P\ {}_T\!\rightsquigarrow_\tau\ ?P' \qquad ?P'\ {}_T\!\rightsquigarrow ?e\ ?P''}{?P\ {}_T\!\rightsquigarrow ?e\ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP\ {}_T\!\rightsquigarrow\checkmark\ STOP}$$

$e \rightarrow P$ **rules**

$$\frac{?e \in ?A}{\Box a{\in}?A \rightarrow\ ?P\ a\ {}_T\!\rightsquigarrow ev\ ?e\ ?P\ ?e} \qquad \frac{?e \in ?A}{\sqcap a{\in}?A \rightarrow\ \ ?P\ a\ {}_T\!\rightsquigarrow ev\ ?e\ ?P\ ?e}$$

$$\frac{}{?e \rightarrow\ ?P\ {}_T\!\rightsquigarrow ev\ ?e\ ?P}$$

$(\sqcap)$ **rules**

$$\frac{}{?P \sqcap ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P} \qquad \frac{}{?P \sqcap ?Q\ {}_T\!\rightsquigarrow_\tau\ ?Q}$$

$$\frac{?e \in ?A}{\sqcap\ a{\in}?A.\ \ ?P\ a\ {}_T\!\rightsquigarrow_\tau\ ?P\ ?e}$$

$\mu\ x.\ f\ x$ **rule**

$$\frac{cont\ ?f \qquad ?P = (\mu\ x.\ ?f\ x)}{?P\ {}_T\!\rightsquigarrow_\tau\ ?f\ ?P}$$

$(\Box)$ **rules**

$$\frac{?P\ {}_T\!\rightsquigarrow_\tau\ ?P'}{?P \Box ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P' \Box ?Q} \qquad \frac{?Q\ {}_T\!\rightsquigarrow_\tau\ ?Q'}{?P \Box ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P \Box ?Q'}$$

$$\frac{?P\ {}_T\!\rightsquigarrow ?e\ ?P'}{?P \Box ?Q\ {}_T\!\rightsquigarrow ?e\ ?P'} \qquad \frac{?Q\ {}_T\!\rightsquigarrow ?e\ ?Q'}{?P \Box ?Q\ {}_T\!\rightsquigarrow ?e\ ?Q'}$$

$(;)$ **rule**

$$\frac{\exists\, P'.\ ?P\ {}_T\!\rightsquigarrow\checkmark\ P'}{?P\ ;\ ?Q\ {}_T\!\rightsquigarrow_\tau\ ?Q}$$

Hiding **rules**

$$\frac{?P\ {}_T\!\rightsquigarrow_\tau\ ?P'}{?P \setminus ?B\ {}_T\!\rightsquigarrow_\tau\ ?P' \setminus ?B} \qquad \frac{?P\ {}_T\!\rightsquigarrow\checkmark\ ?P'}{?P \setminus ?B\ {}_T\!\rightsquigarrow\checkmark\ STOP}$$

$$\frac{?P\ {}_T\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \notin ?B}{?P \setminus ?B\ {}_T\!\rightsquigarrow ev\ ?e\ ?P' \setminus ?B} \qquad \frac{?P\ {}_T\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \in ?B}{?P \setminus ?B\ {}_T\!\rightsquigarrow_\tau\ ?P' \setminus ?B}$$

Renaming **rule**

$$\frac{?P\ {}_T\!\rightsquigarrow\checkmark\ ?P'}{Renaming\ ?P\ ?f\ {}_T\!\rightsquigarrow\checkmark\ STOP}$$

Sync **rules**

$$\frac{?P\ {}_T\!\rightsquigarrow\checkmark\ ?P'}{?P\ [\![?S]\!]\ ?Q\ {}_T\!\rightsquigarrow_\tau\ SKIP\ [\![?S]\!]\ ?Q} \qquad \frac{?Q\ {}_T\!\rightsquigarrow\checkmark\ ?Q'}{?P\ [\![?S]\!]\ ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P\ [\![?S]\!]\ SKIP}$$

$$\frac{}{SKIP\ [\![?S]\!]\ SKIP\ {}_T\!\rightsquigarrow_\tau\ SKIP}$$

($\rhd$) **rules**

$$\frac{?P\ {}_T\!\rightsquigarrow_\tau\ ?P'}{?P \rhd ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P' \rhd ?Q} \qquad \frac{?P\ {}_T\!\rightsquigarrow ?e\ ?P'}{?P \rhd ?Q\ {}_T\!\rightsquigarrow ?e\ ?P'}$$

$$\frac{}{?P \rhd ?Q\ {}_T\!\rightsquigarrow_\tau\ ?Q}$$

($\triangle$) **rules**

$$\frac{?P\ {}_T\!\rightsquigarrow_\tau\ ?P'}{?P \triangle ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P' \triangle ?Q} \qquad \frac{?Q\ {}_T\!\rightsquigarrow_\tau\ ?Q'}{?P \triangle ?Q\ {}_T\!\rightsquigarrow_\tau\ ?P \triangle ?Q'}$$

$$\frac{?P\ {}_T\!\rightsquigarrow ev\ ?e\ ?P'}{?P \triangle ?Q\ {}_T\!\rightsquigarrow ev\ ?e\ ?P' \triangle ?Q} \qquad \frac{?Q\ {}_T\!\rightsquigarrow ev\ ?e\ ?Q'}{?P \triangle ?Q\ {}_T\!\rightsquigarrow ev\ ?e\ ?Q'}$$

Throw **rules**

$$\frac{\forall\, a{\in}?A.\ ?Q\ a\ {}_T\!\rightsquigarrow_\tau\ ?Q'\ a}{?P\ \Theta\ a{\in}?A.\ ?Q\ a\ {}_T\!\rightsquigarrow_\tau\ ?P\ \Theta\ a{\in}?A.\ ?Q'\ a}$$

$$\frac{?P\ {}_T\!\rightsquigarrow ev\ ?e\ ?P' \qquad ?e \in ?A}{?P\ \Theta\ a{\in}?A.\ ?Q\ a\ {}_T\!\rightsquigarrow ev\ ?e\ ?Q\ ?e}$$

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:
Renaming $(\Box a{\in}A \rightarrow P\ (f\ a))\ f = \Box b{\in}f\ `\ A \rightarrow$ *Renaming* $(P\ b)\ f$
Renaming $(\sqcap a{\in}A \rightarrow\ P\ (f\ a))\ f = \sqcap b{\in}f\ `\ A \rightarrow\ $ *Renaming* $(P\ b)\ f$

lemma *Renaming-Mprefix*:
‹*Renaming* $(\Box a \in A \rightarrow P\ a)\ f =$
$\Box y \in f\ `\ A \rightarrow \sqcap a \in \{x \in A.\ y = f\ x\}.$ *Renaming* $(P\ a)\ f$› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *Renaming-Mprefix-Sliding*:
‹*Renaming* $((\Box a \in A \rightarrow P\ a) \rhd Q)\ f =$
$(\Box y \in f\ `\ A \rightarrow \sqcap a \in \{x \in A.\ y = f\ x\}.$ *Renaming* $(P\ a)\ f) \rhd$ *Renaming* $Q\ f$›
⟨*proof*⟩

lemma *Renaming-GlobalNdet*:
‹*Renaming* $(\sqcap a \in A.\ P\ a)\ f = \sqcap b \in f\ `\ A.\ \sqcap a \in \{a \in A.\ b = f\ a\}.$ *Renaming* $(P\ a)\ f$›
⟨*proof*⟩

lemma *Renaming-Mndetprefix*:
‹*Renaming* (⊓*a* ∈ *A* → *P a*) *f* =
⊓*y* ∈ *f* ‘ *A* → ⊓*a* ∈ {*x* ∈ *A*. *y* = *f x*}. *Renaming* (*P a*) *f*› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

16.1.2 *Renaming* and *Hiding*

When f is one to one, *Renaming* $(P \setminus S)$ f will behave like we expect it to do.

lemma *strict-mono-map*: ‹*strict-mono g* ⟹ *strict-mono* (λ*i*. *map f* (*g i*))›
⟨*proof*⟩

lemma *trace-hide-map-EvExt* :
‹*inj-on* (*EvExt f*) (*set s* ∪ *ev* ‘ *S*) ⟹
trace-hide (*map* (*EvExt f*) *s*) (*ev* ‘ *f* ‘ *S*) =
map (*EvExt f*) (*trace-hide s* (*ev* ‘ *S*))›
⟨*proof*⟩

lemma *inj-on-EvExt-iff*: ‹*inj-on* (*EvExt f*) (*insert tick* (*ev* ‘ *S*)) ⟷ *inj-on f S*›
⟨*proof*⟩

lemma *inj-on-EvExt-set-T*:
‹*inj-on f* (*events-of P*) ⟹ *s* ∈ 𝒯 *P* ⟹ *inj-on* (*EvExt f*) (*insert tick* (*set s*))›
⟨*proof*⟩

theorem *bij-Renaming-Hiding*: ‹*Renaming* (*P* \ *S*) *f* = *Renaming P f* \ *f* ‘ *S*›
(**is** ‹*?lhs* = *?rhs*›) **if** *bij-f*: ‹*bij f*›
⟨*proof*⟩

16.1.3 *Renaming* and *Sync*

Idem for the synchronization: when f is one to one, *Renaming* $(P \llbracket S \rrbracket Q)$ will behave as expected.

lemma *exists-map-antecedent-if-subset-range*:
‹*set u* ⊆ *range f* ⟹ ∃ *t*. *u* = *map f t*›
— In particular, when f is surjective or bijective.
⟨*proof*⟩

lemma *bij-map-setinterleaving-iff-setinterleaving* :
‹*map f r setinterleaves* ((*map f t*, *map f u*), *f* ‘ *S*) ⟷
r setinterleaves ((*t*, *u*), *S*)› **if** *bij-f* : ‹*bij f*›
⟨*proof*⟩

theorem *bij-Renaming-Sync*:
‹*Renaming* (*P* ⟦*S*⟧ *Q*) *f* = *Renaming P f* ⟦*f* ‘ *S*⟧ *Renaming Q f*›
(**is** ‹*?lhs P Q* = *?rhs P Q*›) **if** *bij-f*: ‹*bij f*›
⟨*proof*⟩

16.2 *Hiding* and *Mprefix*

We already have a way to distribute the *Hiding* operator on the *Mprefix* operator with $S \cap A = \{\} \Longrightarrow$ *Mprefix A ?P* \ *S* = □*x*∈*A* → (*?P x* \ *S*). But this is only usable when $A \cap S = \{\}$. With the (▷) operator, we can now handle the case $A \cap S \neq \{\}$.

16.2.1 Two intermediate Results

lemma *D-Hiding-Mprefix-dir2*:
assumes ‹*s* ∈ $\mathcal{D}$ (□*a* ∈ *A* − *S* → (*P a* \ *S*))›
shows ‹*s* ∈ $\mathcal{D}$ (□*a* ∈ *A* → *P a* \ *S*)›
⟨*proof*⟩

lemma *F-Hiding-Mprefix-dir2*:
assumes ‹*s* ≠ []› **and** ‹(*s*, *X*) ∈ $\mathcal{F}$ (□*a* ∈ *A* − *S* → (*P a* \ *S*))›
shows ‹(*s*, *X*) ∈ $\mathcal{F}$ (□*a* ∈ *A* → *P a* \ *S*)›
⟨*proof*⟩

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*:
‹□*a* ∈ *A* → *P a* \ *S* = □*a* ∈ *A* → (*P a* \ *S*)›
(**is** ‹*?lhs* = *?rhs*›) **if** *disjoint*: ‹*A* ∩ *S* = {}›
⟨*proof*⟩

And we obtain a similar result when adding a (▷) in the expression.

theorem *Hiding-Mprefix-Sliding-disjoint*:
‹((□*a* ∈ *A* → *P a*) ▷ *Q*) \ *S* = (□*a* ∈ *A* → (*P a* \ *S*)) ▷ (*Q* \ *S*)›
if *disjoint*: ‹*A* ∩ *S* = {}›
⟨*proof*⟩

16.2.3 *Hiding* and *Mprefix* for non-disjoint Sets

Finally the new version, when $A \cap S \neq \{\}$.

theorem *Hiding-Mprefix-non-disjoint*:
— Rework this proof

‹$\Box a \in A \rightarrow P\ a \setminus S = (\Box a \in A - S \rightarrow (P\ a \setminus S)) \rhd (\sqcap a \in A \cap S.\ (P\ a \setminus S))$›
(**is** ‹*?lhs* = *?rhs*›) **if** *non-disjoint*: ‹$A \cap S \neq \{\}$›
⟨*proof*⟩
lemma ‹$\exists A\ P\ S.\ A \cap S = \{\} \wedge$
$\Box a \in A{::}nat\ set \rightarrow P\ a \setminus S \neq$
$(\Box a \in A - S \rightarrow (P\ a \setminus S)) \rhd (\sqcap a \in A \cap S.\ (P\ a \setminus S))$›
⟨*proof*⟩

And we obtain a similar result when adding a ($\rhd$) in the expression.

lemma *Hiding-Mprefix-Sliding-non-disjoint*:
‹$((\Box a \in A \rightarrow P\ a) \rhd Q) \setminus S = (\Box a \in A - S \rightarrow (P\ a \setminus S)) \rhd$
$(Q \setminus S) \sqcap (\sqcap a \in A \cap S.\ (P\ a \setminus S))$›
if *non-disjoint*: ‹$A \cap S \neq \{\}$›
⟨*proof*⟩

16.3 ($\rhd$) behaviour

We already proved several laws for the ($\rhd$) operator. Here we give other results in the same spirit as *Hiding-Mprefix-Sliding-disjoint* and *Hiding-Mprefix-Sliding-non-disjoin*

lemma *Mprefix-Sliding-Mprefix-Sliding*:
‹$(\Box a \in A \rightarrow P\ a) \rhd (\Box b \in B \rightarrow Q\ b) \rhd R =$
$(\Box\ x \in A \cup B \rightarrow (if\ x \in A \cap B\ then\ P\ x \sqcap Q\ x\ else\ if\ x \in A\ then\ P\ x\ else\ Q\ x)) \rhd R$›
(**is** ‹$(\Box a \in A \rightarrow P\ a) \rhd (\Box b \in B \rightarrow Q\ b) \rhd R$ = *?term* $\rhd R$›)
⟨*proof*⟩

lemma *Mprefix-Sliding-Seq*:
‹$((\Box a \in A \rightarrow P\ a) \rhd P')\ ;\ Q = (\Box a \in A \rightarrow P\ a\ ;\ Q) \rhd (P'\ ;\ Q)$›
⟨*proof*⟩

lemma *Throw-Sliding* :
‹$(\Box a \in A \rightarrow P\ a) \rhd P'\ \Theta\ b \in B.\ Q\ b =$
$(\Box a \in A \rightarrow (if\ a \in B\ then\ Q\ a\ else\ (P\ a\ \Theta\ b \in B.\ Q\ b))) \rhd (P'\ \Theta\ b \in B.\ Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

end

Chapter 17

Conclusion

theory *Conclusion*
imports *OpSemFD OpSemDT OpSemFDBis OpSemDTBis OpSemFBis OpSemT-Bis 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 $\Box a \in A \rightarrow P\ a$) 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 $\Longrightarrow$ *Renaming* (*?P* $\setminus$ *?S*) *?f* = *Renaming ?P ?f* $\setminus$ *?f* ‘ *?S*

bij ?f $\Longrightarrow$ *Renaming* (*?P* $[\![$*?S*$]\!]$ *?Q*) *?f* = *Renaming ?P ?f* $[\![$*?f* ‘ *?S*$]\!]$ *Renaming ?Q ?f*

?A $\cap$ *?S* $\neq \emptyset \Longrightarrow \Box a{\in}?A \rightarrow ?P\ a \setminus ?S = (\Box a{\in}?A - ?S \rightarrow (?P\ a \setminus ?S)) \rhd (\sqcap\ a{\in}?A \cap ?S.\ \ ?P\ a \setminus ?S)$

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.