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

Benoît Ballenghien and Burkhart Wolff

March 17, 2025

Abstract

Recently, a modern version of Roscoe and Brookes [3] Failure-Divergence Semantics for CSP has been formalized in Isabelle [7] 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 [3] Failure-Divergence Semantics for CSP has been formalized in Isabelle [7] 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, the construction is done via locale contexts to be as general as possible, and several possibilities are discussed.

As a bonus, we have proven new "'laws" for HOL-CSP.

Contents

Chapter 1

Introduction

1.1 Motivations

HOL-CSP [7] 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" [4] and the semantic details in a joint paper of Roscoe and Brooks "An improved failures model for communicating processes" [3].

Basically, the session HOL-CSP introduces the type $('a, 'r)$ $process_{ptick}$, 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 $P \leadsto_\tau Q$ ("P evolves internally to Q) and $P \leadsto_e Q$ ("P evolves to Q by emitting e") in terms of the denotational semantics and derive the expected laws for operational semantics from these. It has been published in ITP24 [2]

The overall objective of this work is to provide a formal, machine checked foundation for the laws provided by Roscoe in [4, 6].

1.2 The Global Architecture of HOL-CSP_OpSem

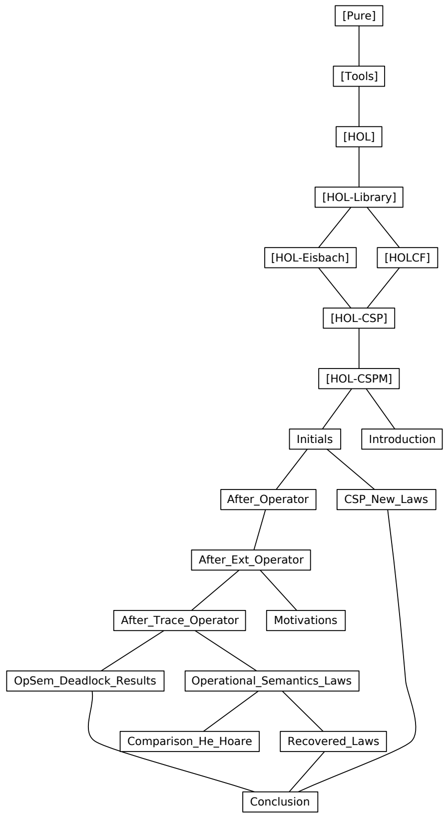

Figure 1.1: The overall architecture

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

The Initials Notion

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 $e \to P$.

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

Therefore we must define a notion that captures this idea.

2.1 Definition

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

definition *initials* :: ‹('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $event_{ptick}$ *set*› (‹(-^{0})› [*1000*] *999*)
  where ‹$P^0 \equiv \{e.\ [e] \in \mathcal{T}\ P\}$›

lemma *initials-memI'* : ‹$[e] \in \mathcal{T}\ P \Longrightarrow e \in P^0$›
  and *initials-memD* : ‹$e \in P^0 \Longrightarrow [e] \in \mathcal{T}\ P$›
  ⟨*proof*⟩

lemma *initials-def-bis*: ‹$P^0 = \{e.\ \exists s.\ e \# s \in \mathcal{T}\ P\}$›
  ⟨*proof*⟩

lemma *initials-memI* : ‹$e \# s \in \mathcal{T}\ P \Longrightarrow e \in P^0$›
  ⟨*proof*⟩

We say here that the *initials* 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 P^0 as the set of events that P can not refuse at first: $\{e.\ \{e\} \notin \mathcal{R}\ P\}$. These two definitions are not equivalent (and the second one is more restrictive than the first one). Moreover, the second does not behave well with the non-deterministic choice ($\sqcap$) (see the **notepad** below).

Therefore, we will keep the first one.

We also have a strong argument of authority: this is the definition given by Roscoe [5, p.40].

notepad
begin
⟨*proof*⟩
end

2.2 Anti-Mono Rules

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

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

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

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

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

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

2.3 Behaviour of *initials* with *STOP*, *SKIP* and $\bot$

lemma *initials-STOP* [*simp*] : ‹$STOP^0 = \{\}$›
⟨*proof*⟩

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

lemma *initials-empty-iff-STOP*: ‹$P^0 = \{\} \longleftrightarrow P = STOP$›
⟨*proof*⟩

lemma *initials-SKIP* [*simp*] : ‹$(SKIP\ r)^0 = \{\checkmark(r)\}$›
⟨*proof*⟩

lemma *initials-SKIPS* [*simp*] : ‹$(SKIPS\ R)^0 = tick$ ‘ R›
⟨*proof*⟩

lemma *initials-BOT* [*simp*] : ‹$\bot^0 = UNIV$›

$\langle proof \rangle$

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

lemma ‹$\exists P.\ P^0 = \{✓(r)\} \wedge P \neq (SKIP\ r)$›
$\langle proof \rangle$

lemma ‹$\exists P.\ P^0 = UNIV \wedge P \neq \bot$›
$\langle proof \rangle$

But when $✓(r) \in P^0$, we can still have this refinement:

lemma *initial-tick-iff-FD-SKIP* : ‹$✓(r) \in P^0 \longleftrightarrow P \sqsubseteq_{FD} SKIP\ r$›
$\langle proof \rangle$

lemma *initial-ticks-iff-FD-SKIPS* : ‹$R \neq \{\} \Longrightarrow tick\ `\ R \subseteq P^0 \longleftrightarrow P \sqsubseteq_{FD} SKIPS\ R$›
$\langle proof \rangle$

We also obtain characterizations for $P\ ;\ Q = \bot$.

lemma *Seq-is-BOT-iff* : ‹$P\ ;\ Q = \bot \longleftrightarrow P = \bot \vee (\exists r.\ ✓(r) \in P^0 \wedge Q = \bot)$›
$\langle proof \rangle$

2.4 Behaviour of *initials* with Operators of `HOL-CSP`

lemma *initials-Mprefix* : ‹$(\Box a \in A \rightarrow P\ a)^0 = ev\ `\ A$›
and *initials-Mndetprefix* : ‹$(\sqcap a \in A \rightarrow P\ a)^0 = ev\ `\ A$›
and *initials-write0* : ‹$(a \rightarrow Q)^0 = \{ev\ a\}$›
and *initials-write* : ‹$(c!a \rightarrow Q)^0 = \{ev\ (c\ a)\}$›
and *initials-read* : ‹$(c?a{\in}A \rightarrow P\ a)^0 = ev\ `\ c\ `\ A$›
and *initials-ndet-write* : ‹$(c!!a{\in}A \rightarrow P\ a)^0 = ev\ `\ c\ `\ A$›
$\langle proof \rangle$

As discussed earlier, *initials* behaves very well with $(\Box)$, $(\sqcap)$ and $(\rhd)$.

lemma *initials-Det* : ‹$(P \Box Q)^0 = P^0 \cup Q^0$›
and *initials-Ndet* : ‹$(P \sqcap Q)^0 = P^0 \cup Q^0$›
and *initials-Sliding* : ‹$(P \rhd Q)^0 = P^0 \cup Q^0$›
$\langle proof \rangle$

lemma *initials-Seq*:
‹$(P\ ;\ Q)^0 = (\ if\ P = \bot\ then\ UNIV$
$else\ P^0 - range\ tick \cup (\bigcup r{\in}\{r.\ ✓(r) \in P^0\}.\ Q^0))$›
(**is** ‹- = (*if* - *then* - *else* *?rhs*)›)
$\langle proof \rangle$

lemma *initials-Sync*:
$\langle (P\ [\![S]\!]\ Q)^0 = (\textit{if } P = \bot \lor Q = \bot \textit{ then UNIV else}$
$P^0 \cup Q^0 - (\textit{range tick} \cup \textit{ev} ` S) \cup P^0 \cap Q^0 \cap (\textit{range tick} \cup \textit{ev} ` S))\rangle$
(**is** $\langle (P\ [\![S]\!]\ Q)^0 = (\textit{if } P = \bot \lor Q = \bot \textit{ then UNIV else ?rhs})\rangle$)
$\langle proof \rangle$

lemma *initials-Renaming*:
$\langle (\textit{Renaming } P\ f\ g)^0 = (\textit{if } P = \bot \textit{ then UNIV else map-event}_{ptick}\ f\ g\ `\ P^0)\rangle$
$\langle proof \rangle$

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

We start with two characterizations:

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

lemma *Hiding-is-BOT-iff* :
$\langle P \setminus S = \bot \longleftrightarrow (\exists t.\ \textit{set } t \subseteq \textit{ev} ` S \land$
$(t \in \mathcal{D}\ P \lor (\exists\ f.\ \textit{isInfHiddenRun } f\ P\ S \land t \in \textit{range } f)))\rangle$
(**is** $\langle P \setminus S = \bot \longleftrightarrow \textit{?rhs}\rangle$)
$\langle proof \rangle$

lemma *event-in-D-Hiding-iff* :
$\langle [e] \in \mathcal{D}\ (P \setminus S) \longleftrightarrow$
$P \setminus S = \bot \lor (\exists x\ t.\ e = \textit{ev } x \land x \notin S \land [\textit{ev } x] = \textit{trace-hide } t\ (\textit{ev} ` S) \land$
$(t \in \mathcal{D}\ P \lor (\exists f.\ \textit{isInfHiddenRun } f\ P\ S \land t \in \textit{range } f)))\rangle$
(**is** $\langle [e] \in \mathcal{D}\ (P \setminus S) \longleftrightarrow P \setminus S = \bot \lor \textit{?ugly-assertion}\rangle$)
$\langle proof \rangle$

Now we can express $(P \setminus S)^0$. This result contains the term $P \setminus S = \bot$ that can be unfolded with *Hiding-is-BOT-iff* and the term $[\textit{ev } x] \in \mathcal{D}\ (P \setminus S)$ that can be unfolded with *event-in-D-Hiding-iff*.

lemma *initials-Hiding*:
$\langle (P \setminus S)^0 = (\textit{if } P \setminus S = \bot \textit{ then UNIV else}$
$\{e.\ \textit{case } e \textit{ of ev } x \Rightarrow x \notin S \land ([\textit{ev } x] \in \mathcal{D}\ (P \setminus S) \lor (\exists t.\ [\textit{ev } x] = \textit{trace-hide } t\ (\textit{ev} ` S) \land (t, \textit{ev} ` S) \in \mathcal{F}\ P))$
$\mid ✓(r) \Rightarrow \exists t.\ \textit{set } t \subseteq \textit{ev} ` S \land t\ @\ [✓(r)] \in \mathcal{T}\ P\})\rangle$
(**is** $\langle \textit{initials } (P \setminus S) = (\textit{if } P \setminus S = \bot \textit{ then UNIV else ?set})\rangle$)
$\langle proof \rangle$

In the end the result would look something like this:

$(P \setminus S)^0 = (\textit{if } \exists t.\ \textit{set } t \subseteq \textit{ev} ` S \land (t \in \mathcal{D}\ P \lor (\exists f.\ \textit{isInfHiddenRun } f\ P\ S \land t \in \textit{range } f)) \textit{ then UNIV else } \{e.\ \textit{case } e \textit{ of ev } x \Rightarrow x \notin S \land (((\exists t.\ \textit{set } t \subseteq$

ev ‘ $S \wedge (t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f))) \vee (\exists xa\ t.\ ev\ x = ev\ xa \wedge xa \notin S \wedge [ev\ xa] = trace\text{-}hide\ t\ (ev$ ‘ $S) \wedge (t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f)))) \vee (\exists t.\ [ev\ x] = trace\text{-}hide\ t\ (ev$ ‘ $S) \wedge (t,\ ev$ ‘ $S) \in \mathcal{F}\ P)) \mid ✓(r) \Rightarrow \exists t.\ set\ t \subseteq ev$ ‘ $S \wedge t\ @\ [✓(r)] \in \mathcal{T}\ P\})$

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

corollary *initial-tick-Hiding-iff* :
‹$✓(r) \in (P \setminus B)^0 \longleftrightarrow P \setminus B = \bot \vee (\exists t.\ set\ t \subseteq ev$ ‘ $B \wedge t\ @\ [✓(r)] \in \mathcal{T}\ P)$›
⟨*proof*⟩

corollary *initial-tick-imp-initial-tick-Hiding*:
‹$✓(r) \in P^0 \Longrightarrow ✓(r) \in (P \setminus B)^0$›
⟨*proof*⟩

corollary *initial-inside-Hiding-iff* :
‹$e \in S \Longrightarrow ev\ e \in (P \setminus S)^0 \longleftrightarrow P \setminus S = \bot$›
⟨*proof*⟩

corollary *initial-notin-Hiding-iff* :
‹$e \notin S \Longrightarrow ev\ e \in (P \setminus S)^0 \longleftrightarrow$
$P \setminus S = \bot \vee$
$(\exists t.\ [ev\ e] = trace\text{-}hide\ t\ (ev$ ‘ $S) \wedge$
$(t \in \mathcal{D}\ P \vee (\exists f.\ isInfHiddenRun\ f\ P\ S \wedge t \in range\ f) \vee (t,\ ev$ ‘ $S) \in \mathcal{F}\ P))$›
⟨*proof*⟩

corollary *initial-notin-imp-initial-Hiding*:
‹$ev\ e \in (P \setminus S)^0$› **if** *initial* : ‹$ev\ e \in P^0$› **and** *notin* : ‹$e \notin S$›
⟨*proof*⟩

2.5 Behaviour of *initials* with Operators of `HOL-CSPM`

lemma *initials-GlobalDet*:
‹$(\Box a \in A.\ P\ a)^0 = (\bigcup a \in A.\ initials\ (P\ a))$›
⟨*proof*⟩

lemma *initials-GlobalNdet*:
‹$(\sqcap a \in A.\ P\ a)^0 = (\bigcup a \in A.\ initials\ (P\ a))$›
⟨*proof*⟩

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

lemma *initials-Throw* : ‹$(P\ \Theta\ a \in A.\ Q\ a)^0 = P^0$›
⟨*proof*⟩

lemma *initials-Interrupt*: ‹$(P \triangle Q)^0 = P^0 \cup Q^0$›
⟨*proof*⟩

2.6 Behaviour of *initials* with Reference Processes

lemma *initials-DF*: ‹$(DF\ A)^0 = ev\ `\ A$›
⟨*proof*⟩

lemma *initials-DF_{SKIPS}*: ‹$(DF_{SKIPS}\ A\ R)^0 = ev\ `\ A \cup tick\ `\ R$›
⟨*proof*⟩

lemma *initials-RUN*: ‹$(RUN\ A)^0 = ev\ `\ A$›
⟨*proof*⟩

lemma *initials-CHAOS*: ‹$(CHAOS\ A)^0 = ev\ `\ A$›
⟨*proof*⟩

lemma *initials-$CHAOS_{SKIPS}$*: ‹$(CHAOS_{SKIPS}\ A\ R)^0 = ev\ `\ A \cup tick\ `\ R$›
⟨*proof*⟩

lemma *empty-ev-initials-iff-empty-events-of* :
‹$\{a.\ ev\ a \in P^0\} = \{\} \longleftrightarrow \alpha(P) = \{\}$›
⟨*proof*⟩

2.7 Properties of *initials* related to continuity

We prove here some properties that we will need later in continuity or admissibility proofs.

lemma *initials-LUB*:
‹$chain\ Y \Longrightarrow (\bigsqcup i.\ Y\ i)^0 = (\bigcap P \in (range\ Y).\ P^0)$›

⟨*proof*⟩

lemma *adm-in-F*: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ (s,\ X) \in \mathcal{F}\ (u\ x))$›
⟨*proof*⟩

lemma *adm-in-D*: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ s \in \mathcal{D}\ (u\ x))$›
⟨*proof*⟩

lemma *adm-in-T*: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ s \in \mathcal{T}\ (u\ x))$›
⟨*proof*⟩

lemma *initial-adm*[*simp*] : ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ e \in (u\ x)^0)$›
⟨*proof*⟩

Chapter 3

Construction of the After Operator

Now that we have defined P^0, we can talk about what happens to P after an event belonging to this set.

3.1 Definition

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

The intuitive way of doing so is to only keep the tails of the traces beginning by *ev a* (and similar for failures and divergences). However we have an issue if *ev a* $\notin P^0$ i.e. if no trace of P begins with *ev a*: the result would no longer verify the invariant *is-process* because its trace set would be empty. We must therefore distinguish this case.

In the previous version, we agreed to get *STOP* after an event *ev a* that was not in the *initials* of P. But even if its repercussions were minimal, this choice seemed a little artificial and arbitrary. In this new version we use a placeholder instead: Ψ. When *ev a* $\in P^0$ we use our intuitive definition, and *ev a* $\notin P^0$ we define P after a being equal to $\Psi\ P\ a$.

For the moment we have no additional assumption on Ψ.

locale *After* =
  fixes Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
begin

lift-definition *After* :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$› (**infixl** ‹*after*› *86*)
  is ‹$\lambda P\ a.$ *if ev a* $\in P^0$

then $(\{(t, X). (ev\ a \# t, X) \in \mathcal{F}\ P\},$
$\{\ t\ \ .\ ev\ a \# t\ \ \in \mathcal{D}\ P\})$
else $(\mathcal{F}\ (\Psi\ P\ a), \mathcal{D}\ (\Psi\ P\ a))$›
⟨*proof*⟩

3.2 Projections

lemma *F-After* :
‹$\mathcal{F}$ (*P after a*) = (*if ev a* $\in P^0$ *then* $\{(t, X). (ev\ a \# t, X) \in \mathcal{F}\ P\}$ *else* $\mathcal{F}$ (Ψ *P a*))›
⟨*proof*⟩

lemma *D-After* :
‹$\mathcal{D}$ (*P after a*) = (*if ev a* $\in P^0$ *then* $\{s.\ ev\ a \# s \in \mathcal{D}\ P\}$ *else* $\mathcal{D}$ (Ψ *P a*))›
⟨*proof*⟩

lemma *T-After* :
‹$\mathcal{T}$ (*P after a*) = (*if ev a* $\in P^0$ *then* $\{s.\ ev\ a \# s \in \mathcal{T}\ P\}$ *else* $\mathcal{T}$ (Ψ *P a*))›
⟨*proof*⟩

lemmas *After-projs* = *F-After D-After T-After*

lemma *not-initial-After* : ‹*ev a* $\notin P^0 \Longrightarrow$ *P after a* = Ψ *P a*›
⟨*proof*⟩

lemma *initials-After* :
‹$(P\ after\ a)^0$ = (*if ev a* $\in P^0$ *then* $\{e.\ ev\ a \# [e] \in \mathcal{T}\ P\}$ *else* $(\Psi\ P\ a)^0$)›
⟨*proof*⟩

3.3 Monotony

lemma *mono-After* : ‹*P after a* $\sqsubseteq$ *Q after a*›
if ‹$P \sqsubseteq Q$› **and** ‹*ev a* $\notin Q^0 \Longrightarrow \Psi\ P\ a \sqsubseteq \Psi\ Q\ a$›
⟨*proof*⟩

lemma *mono-After-T* : ‹$P \sqsubseteq_T Q \Longrightarrow$ *P after a* $\sqsubseteq_T$ *Q after a*›
and *mono-After-F* : ‹$P \sqsubseteq_F Q \Longrightarrow$ *P after a* $\sqsubseteq_F$ *Q after a*›
and *mono-After-D* : ‹$P \sqsubseteq_D Q \Longrightarrow$ *P after a* $\sqsubseteq_D$ *Q after a*›
and *mono-After-FD* : ‹$P \sqsubseteq_{FD} Q \Longrightarrow$ *P after a* $\sqsubseteq_{FD}$ *Q after a*›
and *mono-After-DT* : ‹$P \sqsubseteq_{DT} Q \Longrightarrow$ *P after a* $\sqsubseteq_{DT}$ *Q after a*›
if ‹*ev a* $\in Q^0$›
⟨*proof*⟩

lemmas *monos-After* = *mono-After mono-After-FD mono-After-DT*
mono-After-F mono-After-D mono-After-T

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

lemma *After-STOP* : ‹$STOP\ after\ a = \Psi\ STOP\ a$›
⟨*proof*⟩

lemma *After-SKIP* : ‹$SKIP\ r\ after\ a = \Psi\ (SKIP\ r)\ a$›
⟨*proof*⟩

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

lemma *After-is-BOT-iff* :
‹$P\ after\ a = \bot \longleftrightarrow (if\ ev\ a \in P^0\ then\ [ev\ a] \in \mathcal{D}\ P\ else\ \Psi\ P\ a = \bot)$›
⟨*proof*⟩

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

In future theories, we will need to know how *After* behaves with other operators of CSP. More specifically, we want to know how *After* can be "distributed" over a sequential composition, a synchronization, etc.

In some way, we are looking for reversing the "step-laws" (laws of distributivity of *Mprefix* over other operators). Given the difficulty in establishing these results in HOL-CSP and HOL-CSPM, one can easily imagine that proving *After* versions will require a lot of work.

3.5.1 Loss of Determinism

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

lemma *After-Mprefix-is-After-Mndetprefix*:
‹$a \in A \Longrightarrow (\Box a \in A \to P\ a)\ after\ a = (\sqcap a \in A \to P\ a)\ after\ a$›
⟨*proof*⟩

lemma *After-Det-is-After-Ndet* :
‹$ev\ a \in P^0 \cup Q^0 \Longrightarrow (P \Box Q)\ after\ a = (P \sqcap Q)\ after\ a$›
⟨*proof*⟩

lemma *After-Sliding-is-After-Ndet* :
‹$ev\ a \in P^0 \cup Q^0 \Longrightarrow (P \rhd Q)\ after\ a = (P \sqcap Q)\ after\ a$›
⟨*proof*⟩

lemma *After-Ndet*:
‹$(P \sqcap Q)\ after\ a =$
$(\ if\ ev\ a \in P^0 \cap Q^0\ then\ P\ after\ a \sqcap Q\ after\ a$
$else\ \ if\ ev\ a \in P^0\ then\ P\ after\ a$

else *if ev a* ∈ Q^0 *then Q after a*
else Ψ (*P* ⊓ *Q*) *a*)› **for** *P Q* :: ‹('*a*, '*r*) $process_{ptick}$›
⟨*proof*⟩

lemma *After-Det*:
‹(*P* □ *Q*) *after a* =
(*if ev a* ∈ P^0 ∩ Q^0 *then P after a* ⊓ *Q after a*
else *if ev a* ∈ P^0 *then P after a*
else *if ev a* ∈ Q^0 *then Q after a*
else Ψ (*P* □ *Q*) *a*)›
⟨*proof*⟩

lemma *After-Sliding*:
‹(*P* ▷ *Q*) *after a* =
(*if ev a* ∈ P^0 ∩ Q^0 *then P after a* ⊓ *Q after a*
else *if ev a* ∈ P^0 *then P after a*
else *if ev a* ∈ Q^0 *then Q after a*
else Ψ (*P* ▷ *Q*) *a*)›
⟨*proof*⟩

lemma *After-Mprefix*:
‹(□ *a* ∈ *A* → *P a*) *after a* = (*if a* ∈ *A then P a else* Ψ (□ *a* ∈ *A* → *P a*) *a*)›
⟨*proof*⟩

lemma *After-Mndetprefix*:
‹(⊓ *a* ∈ *A* → *P a*) *after a* = (*if a* ∈ *A then P a else* Ψ (⊓ *a* ∈ *A* → *P a*) *a*)›
⟨*proof*⟩

corollary *After-write0* : ‹(*a* → *P*) *after b* = (*if b* = *a then P else* Ψ (*a* → *P*) *b*)›
⟨*proof*⟩

lemma ‹(*a* → *P*) *after a* = *P*› ⟨*proof*⟩

This result justifies seeing *P after a* as the reciprocal operator of the prefix *a* → *P*.

However, we lose information with *After*: in general, *a* → *P after a* ≠ *P* (even when *ev a* ∈ P^0 and *P* ≠ ⊥).

lemma ‹∃ *P*. *a* → *P after a* ≠ *P*›
⟨*proof*⟩

lemma ‹∃ *P*. *ev a* ∈ P^0 ∧ *a* → *P after a* ≠ *P*›
⟨*proof*⟩

lemma ‹∃ *P*. *ev a* ∈ P^0 ∧ *P* ≠ ⊥ ∧ *a* → *P after a* ≠ *P*›
⟨*proof*⟩

corollary *After-write* : ‹(*c*!*a* → *P*) *after b* = (*if b* = *c a then P else* Ψ (*c*!*a* → *P*)

$b)$›
⟨*proof*⟩

corollary *After-read* :
‹$(c?a{\in}A \to P\ a)$ *after* $b = ($*if* $b \in c$ ‘ A *then* P (*inv-into* A c b) *else* Ψ $(c?a{\in}A \to P\ a)\ b)$›
⟨*proof*⟩

corollary *After-ndet-write* :
‹$(c!!a{\in}A \to P\ a)$ *after* $b = ($*if* $b \in c$ ‘ A *then* P (*inv-into* A c b) *else* Ψ $(c!!a{\in}A \to P\ a)\ b)$›
⟨*proof*⟩

3.5.2 *After* Sequential Composition

The first goal is to obtain an equivalent of $a \to P\ ;\ Q = a \to (P\ ;\ Q)$. But in order to be exhaustive we also have to consider the possibility of Q taking the lead when ✓$(r) \in P^0$ in the sequential composition $P\ ;\ Q$.

lemma *not-skippable-or-not-initialR-After-Seq*:
‹$(P\ ;\ Q)$ *after* $a = ($*if ev* $a \in P^0$ *then* P *after* $a\ ;\ Q$ *else* Ψ $(P\ ;\ Q)\ a)$›
if ‹*range tick* $\cap P^0 = \{\} \vee (\forall r.$ ✓$(r) \in P^0 \longrightarrow$ *ev* $a \notin Q^0)$›
⟨*proof*⟩

lemma *skippable-not-initialL-After-Seq*:
‹$(P\ ;\ Q)$ *after* $a = ($ *if* $(\exists r.$ ✓$(r) \in P^0) \wedge$ *ev* $a \in Q^0$
then Q *after* a *else* Ψ $(P\ ;\ Q)\ a)$›
(**is** ‹$(P\ ;\ Q)$ *after* $a = ($*if ?prem then ?rhs else* -$)$›) **if** ‹*ev* $a \notin P^0$›
⟨*proof*⟩

lemma *skippable-initialL-initialR-After-Seq*:
‹$(P\ ;\ Q)$ *after* $a = (P$ *after* $a\ ;\ Q) \sqcap Q$ *after* a›
(**is** ‹$(P\ ;\ Q)$ *after* $a = (P$ *after* $a\ ;\ Q) \sqcap$ *?rhs*›)
if *assms* : ‹$(\exists r.$ ✓$(r) \in P^0) \wedge$ *ev* $a \in Q^0$› ‹*ev* $a \in P^0$›
⟨*proof*⟩

lemma *not-initialL-not-skippable-or-not-initialR-After-Seq*:
‹*ev* $a \notin P^0 \Longrightarrow$ *range tick* $\cap P^0 = \{\} \vee (\forall r.$ *tick* $r \in P^0 \longrightarrow$ *ev* $a \notin Q^0) \Longrightarrow$
$(P\ ;\ Q)$ *after* $a = \Psi$ $(P\ ;\ Q)\ a$›
⟨*proof*⟩

lemma *After-Seq*:
‹$(P\ ;\ Q)$ *after* $a =$
(*if range tick* $\cap P^0 = \{\} \vee (\forall r.$ ✓$(r) \in P^0 \longrightarrow$ *ev* $a \notin Q^0)$
then *if ev* $a \in P^0$ *then* P *after* $a\ ;\ Q$ *else* Ψ $(P\ ;\ Q)\ a$
else *if ev* $a \in P^0$

then $(P$ *after* a **;** $Q) \sqcap Q$ *after* a
else Q *after* $a)$›
⟨*proof*⟩

3.5.3 *After* Synchronization

Now let's focus on *Sync*. We want to obtain an equivalent of
Mprefix ?A ?P ⟦*?S*⟧ *Mprefix ?B ?Q* = (□a∈(*?A* − *?S*) → (*?P a* ⟦*?S*⟧ *Mprefix ?B ?Q*)) □ (□b∈(*?B* − *?S*) → (*Mprefix ?A ?P* ⟦*?S*⟧ *?Q b*)) □ (□x∈(*?A* ∩ *?B* ∩ *?S*) → (*?P x* ⟦*?S*⟧ *?Q x*))
We will also divide the task.

After version of
⟦$a \notin$ *?S*; *?B* ⊆ *?S*⟧ ⟹ $a \to P$ ⟦*?S*⟧ *Mprefix ?B ?Q* = $a \to (P$ ⟦*?S*⟧ *Mprefix ?B ?Q*).

lemma *initialL-not-initialR-not-in-After-Sync*:
‹$(P$ ⟦S⟧ $Q)$ *after* $a = P$ *after* a ⟦S⟧ Q›
if *initial-hyps*: ‹*ev* $a \in P^0$› ‹*ev* $a \notin Q^0$› **and** *notin*: ‹$a \notin S$›
⟨*proof*⟩

lemma *not-initialL-in-After-Sync*:
‹*ev* $a \notin P^0 \Longrightarrow a \in S \Longrightarrow$
$(P$ ⟦S⟧ $Q)$ *after* a = (*if* $Q = \bot$ *then* $\bot$ *else* Ψ $(P$ ⟦S⟧ $Q)$ $a)$›
⟨*proof*⟩

After version of ⟦$a \in$ *?S*; $a \in$ *?S*⟧ ⟹ $a \to P$ ⟦*?S*⟧ $a \to Q = a \to (P$ ⟦*?S*⟧ $Q)$.

lemma *initialL-initialR-in-After-Sync*:
‹$(P$ ⟦S⟧ $Q)$ *after* $a = P$ *after* a ⟦S⟧ Q *after* a›
if *initial-hyps*: ‹*ev* $a \in P^0$› ‹*ev* $a \in Q^0$› **and** *inside*: ‹$a \in S$›
⟨*proof*⟩

After version of
⟦$e \notin$ *?S*; $e \notin$ *?S*⟧ ⟹ $e \to P$ ⟦*?S*⟧ $e \to Q = (e \to (P$ ⟦*?S*⟧ $e \to Q)) \;\square\; (e \to (e \to P$ ⟦*?S*⟧ $Q))$.

lemma *initialL-initialR-not-in-After-Sync*:
‹$(P$ ⟦S⟧ $Q)$ *after* $a = (P$ *after* a ⟦S⟧ $Q) \sqcap (P$ ⟦S⟧ Q *after* $a)$›
if *initial-hyps*: ‹*ev* $a \in P^0$› ‹*ev* $a \in Q^0$› **and** *notin*: ‹$a \notin S$› **for** P Q :: ‹$('a, 'r)$ $process_{ptick}$›
⟨*proof*⟩

lemma *not-initialL-not-initialR-After-Sync*: ‹$(P$ ⟦S⟧ $Q)$ *after* $a = \Psi$ $(P$ ⟦S⟧ $Q)$ a›

if *initial-hyps*: ‹$ev\ a \notin P^0$› ‹$ev\ a \notin Q^0$›
⟨*proof*⟩

Finally, the monster theorem !

theorem *After-Sync*:
‹$(P\ [\![S]\!]\ Q)\ after\ a =$
$(\ \ if\ P = \bot \lor Q = \bot\ then\ \bot$
$\ \ else\ \ \ if\ ev\ a \in P^0 \cap Q^0$
$\ \ \ \ \ \ then\ \ \ if\ a \in S\ then\ P\ after\ a\ [\![S]\!]\ Q\ after\ a$
$\ \ \ \ \ \ \ \ \ \ else\ (P\ after\ a\ [\![S]\!]\ Q) \sqcap (P\ [\![S]\!]\ Q\ after\ a)$
$\ \ \ \ \ \ else\ \ \ if\ ev\ a \in P^0 \land a \notin S\ then\ P\ after\ a\ [\![S]\!]\ Q$
$\ \ \ \ \ \ \ \ \ \ else\ \ \ if\ ev\ a \in Q^0 \land a \notin S\ then\ P\ [\![S]\!]\ Q\ after\ a$
$\ \ \ \ \ \ \ \ \ \ \ \ \ \ else\ \Psi\ (P\ [\![S]\!]\ Q)\ a)$›
⟨*proof*⟩

3.5.4 *After* Hiding Operator

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

lemma *Hiding-FD-Hiding-After-if-initial-inside*:
‹$a \in A \Longrightarrow P \setminus A \sqsubseteq_{FD} P\ after\ a \setminus A$›
and *After-Hiding-FD-Hiding-After-if-initial-notin*:
‹$a \notin A \Longrightarrow (P \setminus A)\ after\ a \sqsubseteq_{FD} P\ after\ a \setminus A$›
if *initial*: ‹$ev\ a \in P^0$›
⟨*proof*⟩

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

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

corollary *Hiding-DT-Hiding-After-if-initial-inside*:
‹$ev\ a \in P^0 \Longrightarrow a \in A \Longrightarrow P \setminus A \sqsubseteq_{DT} P\ after\ a \setminus A$›
and *After-Hiding-DT-Hiding-After-if-initial-notin*:
‹$ev\ a \in P^0 \Longrightarrow a \notin A \Longrightarrow (P \setminus A)\ after\ a \sqsubseteq_{DT} P\ after\ a \setminus A$›
⟨*proof*⟩

end

3.5.5 Renaming is tricky

In all generality, *Renaming* takes a process P :: ('a, 'r) $process_{ptick}$, a function f :: 'a ⇒ 'b, a function g :: 'r ⇒ 's, and returns *Renaming P f g* :: ('b, 's) $process_{ptick}$. But if we try to write and prove a lemma *After-Renaming* like we did for the other operators, the mechanism of the locale *After* would constrain f :: 'a ⇒ 'a and g :: 'r ⇒ 'r.

We solve this issue with a trick: we duplicate the locale, instantiating each one with a different free type.

locale *AfterDuplicated* = $After_\alpha$: *After* Ψ_α + $After_\beta$: *After* Ψ_β
  for Ψ_α :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
    and Ψ_β :: ‹[('b, 's) $process_{ptick}$, 'b] ⇒ ('b, 's) $process_{ptick}$›
begin

notation $After_\alpha$.*After* (**infixl** ‹$after_\alpha$› *86*)
notation $After_\beta$.*After* (**infixl** ‹$after_\beta$› *86*)

lemma *After-Renaming*:
  ‹*Renaming P f g* $after_\beta$ *b* =
  — We highlight the fact that f :: 'a ⇒ 'b
   (*if P* = ⊥ *then* ⊥
    else   *if* ∃ *a. ev a* ∈ P^0 ∧ *f a* = *b*
        then ⊓ *a*∈{*a. ev a* ∈ P^0 ∧ *f a* = *b*}. *Renaming* (*P* $after_\alpha$ *a*) *f g*
        else Ψ_β (*Renaming P f g*) *b*)›
  (**is** ‹*?lhs* = (*if P* = ⊥ *then* ⊥
        else if ∃ *a. ev a* ∈ P^0 ∧ *f a* = *b then ?rhs else* -)›)

⟨*proof*⟩

no-notation $After_\alpha$.*After* (**infixl** ‹$after_\alpha$› *86*)
no-notation $After_\beta$.*After* (**infixl** ‹$after_\beta$› *86*)

end

Now we can get back to *After*.

context *After*
begin

3.6 Behaviour of *After* with Operators of HOL-CSPM

lemma *After-GlobalDet-is-After-GlobalNdet*:
  ‹*ev a* ∈ (⋃ *a* ∈ *A*. (*P a*)^0) ⟹ (□ *a* ∈ *A*. *P a*) *after a* = (⊓ *a* ∈ *A*. *P a*) *after a*›
  ⟨*proof*⟩

lemma *After-GlobalNdet*:
  ‹(⊓ *a* ∈ *A*. *P a*) *after a* = (*if ev a* ∈ (⋃ *a* ∈ *A*. (*P a*)^0)

$\qquad\qquad\qquad\qquad$ *then* $\sqcap$ $x \in \{x \in A.\ ev\ a \in (P\ x)^0\}.\ P\ x$ *after* a
$\qquad\qquad\qquad\qquad$ *else* Ψ ($\sqcap$ $a \in A.\ P\ a$) a)›
(**is** ‹*?lhs* = (*if ?prem then ?rhs else* -)›)
⟨*proof*⟩

lemma *After-GlobalDet*:
‹($\square$ $a \in A.\ P\ a$) *after* a = (*if ev* $a \in (\bigcup a \in A.\ (P\ a)^0)$
then $\sqcap$ $x \in \{x \in A.\ ev\ a \in (P\ x)^0\}.\ P\ x$ *after* a
else Ψ ($\square$ $a \in A.\ P\ a$) a)›
⟨*proof*⟩

3.6.1 *After* Throwing

lemma *After-Throw*:
‹($P\ \Theta\ a \in A.\ Q\ a$) *after* a =
(*if* $P = \bot$ *then* $\bot$
else *if ev* $a \in P^0$ *then* *if* $a \in A$ *then* $Q\ a$
else P *after* $a\ \Theta\ a \in A.\ Q\ a$
else Ψ ($P\ \Theta\ a \in A.\ Q\ a$) a)›
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

3.6.2 *After* Interrupting

theorem *After-Interrupt*:
‹($P \triangle Q$) *after* a =
(*if ev* $a \in P^0 \cap Q^0$
then Q *after* $a \sqcap$ (P *after* $a \triangle Q$)
else *if ev* $a \in P^0 \wedge$ *ev* $a \notin Q^0$
then P *after* $a \triangle Q$
else *if ev* $a \notin P^0 \wedge$ *ev* $a \in Q^0$
then Q *after* a
else Ψ ($P \triangle Q$) a)›
⟨*proof*⟩

3.7 Behaviour of *After* with Reference Processes

lemma *After-DF*:
‹*DF A after a* = (*if* $a \in A$ *then DF A else* Ψ (*DF A*) a)›
⟨*proof*⟩

lemma *After-DF*$_{SKIPS}$:
‹DF_{SKIPS} *A R after a* = (*if* $a \in A$ *then* DF_{SKIPS} *A R else* Ψ (DF_{SKIPS} *A R*) a)›
⟨*proof*⟩

lemma *After-RUN*:
‹*RUN A after a* = (*if* $a \in A$ *then RUN A else* Ψ (*RUN A*) a)›

⟨*proof*⟩

lemma *After-CHAOS*:
‹$CHAOS\ A\ after\ a = (if\ a \in A\ then\ CHAOS\ A\ else\ \Psi\ (CHAOS\ A)\ a)$›
⟨*proof*⟩

lemma *After-CHAOS*$_{SKIPS}$:
‹$CHAOS_{SKIPS}\ A\ R\ after\ a = (if\ a \in A\ then\ CHAOS_{SKIPS}\ A\ R\ else\ \Psi\ (CHAOS_{SKIPS}\ A\ R)\ a)$›
⟨*proof*⟩

lemma *DF-FD-After*: ‹$DF\ A \sqsubseteq_{FD} P\ after\ a$› **if** ‹$ev\ a \in P^0$› **and** ‹$DF\ A \sqsubseteq_{FD} P$›
⟨*proof*⟩

lemma *DF*$_{SKIPS}$*-FD-After*: ‹$DF_{SKIPS}\ A\ R \sqsubseteq_{FD} P\ after\ a$› **if** ‹$ev\ a \in P^0$› **and** ‹$DF_{SKIPS}\ A\ R \sqsubseteq_{FD} P$›
⟨*proof*⟩

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

corollary *deadlock-free-After*:
‹$deadlock\text{-}free\ P \Longrightarrow$
$deadlock\text{-}free\ (P\ after\ a) \longleftrightarrow$
$(if\ ev\ a \in P^0\ then\ True\ else\ deadlock\text{-}free\ (\Psi\ P\ a))$›
⟨*proof*⟩

corollary *deadlock-free*$_{SKIPS}$*-After*:
‹$deadlock\text{-}free_{SKIPS}\ P \Longrightarrow$
$deadlock\text{-}free_{SKIPS}\ (P\ after\ a) \longleftrightarrow$
$(if\ ev\ a \in P^0\ then\ True\ else\ deadlock\text{-}free_{SKIPS}\ (\Psi\ P\ a))$›
⟨*proof*⟩

3.8 Continuity

This is a new result whose main consequence will be the admissibility of the event transition that is defined later (property that paves the way for point-fixed induction)...

Of course this result will require an additional assumption of continuity on the placeholder Ψ.

context begin

private lemma *mono-Ψ-imp-chain-After*:
‹$(\bigwedge P\ Q.\ P \sqsubseteq Q \Longrightarrow \Psi\ P\ a \sqsubseteq \Psi\ Q\ a) \Longrightarrow chain\ Y \Longrightarrow chain\ (\lambda i.\ Y\ i\ after\ a)$›
⟨*proof*⟩ **lemma** *cont-prem-After* :

‹($\bigsqcup$ *i*. *Y i*) *after a* = ($\bigsqcup$ *i*. *Y i after a*)› (**is** ‹*?lhs* = *?rhs*›)
if *cont-*Ψ: ‹*cont* (λ*P*. Ψ *P a*)› **and** *chain-Y* : ‹*chain Y*›
⟨*proof*⟩

lemma *After-cont* [*simp*] :
‹*cont* (λ*x*. *f x after a*)› **if** *cont-*Ψ : ‹*cont* (λ*P*. Ψ *P a*)› **and** *cont-f* : ‹*cont f*›
⟨*proof*⟩

end

end

Chapter 4

Extension of the After Operator

4.1 The After$_{\checkmark}$ Operator

locale *AfterExt* = *After* Ψ
  for Ψ :: ‹[($'a$, $'r$) $process_{ptick}$, $'a$] ⇒ ($'a$, $'r$) $process_{ptick}$ ›
    — Just declaring the types $'a$ and $'r$. +
  fixes Ω :: ‹[($'a$, $'r$) $process_{ptick}$, $'r$] ⇒ ($'a$, $'r$) $process_{ptick}$›
begin

4.1.1 Definition

We just defined *P after e* for *P*::($'a$, $'r$) $process_{ptick}$ and *e*::$'a$; in other words we cannot handle ✓(r). We now introduce a generalisation for *e*::($'a$, $'r$) $event_{ptick}$.

In the previous version, we agreed to get *STOP* after a termination, but only if *P* was not ⊥ since otherwise we kept ⊥. We were not really sure about this choice, and we even introduced a variation where the result after a termination was always *STOP*. In this new version we use a placeholder instead: Ω. We define *P* after ✓(r) being equal to Ω *P r*.

For the moment we have no additional assumption on Ω. This will be discussed later.

definition $After_{tick}$ :: ‹[($'a$, $'r$) $process_{ptick}$, ($'a$, $'r$) $event_{ptick}$] ⇒ ($'a$, $'r$) $process_{ptick}$› (**infixl** ‹$after_{\checkmark}$› *86*)
  where ‹*P* $after_{\checkmark}$ *e* ≡ *case e of ev x* ⇒ *P after x* | ✓(r) ⇒ Ω *P r*›

lemma *not-initial-*$After_{tick}$:
  ‹*e* ∉ *initials P* ⟹ *P* $after_{\checkmark}$ *e* = (*case e of ev x* ⇒ Ψ *P x* | ✓(r) ⇒ Ω *P r*)›
  ⟨*proof*⟩

lemma *initials-After$_{tick}$*:
‹$(P\ after_{✓}\ e)^0 =$
$(case\ e\ of\ ✓(r) \Rightarrow (\Omega\ P\ r)^0$
$\quad | \ ev\ a \Rightarrow if\ ev\ a \in P^0\ then\ \{e.\ [ev\ a,\ e] \in \mathcal{T}\ P\}\ else\ (\Psi\ P\ a)^0)$›
⟨*proof*⟩

4.1.2 Projections

lemma *F-After$_{tick}$*:
‹$\mathcal{F}\ (P\ after_{✓}\ e) =$
$(case\ e\ of\ ✓(r) \Rightarrow \mathcal{F}\ (\Omega\ P\ r)$
$\quad | \ ev\ a \Rightarrow if\ ev\ a \in P^0\ then\ \{(s,\ X).\ (ev\ a\ \#\ s,\ X) \in \mathcal{F}\ P\}\ else\ \mathcal{F}\ (\Psi\ P\ a))$›
⟨*proof*⟩

lemma *D-After$_{tick}$*:
‹$\mathcal{D}\ (P\ after_{✓}\ e) =$
$(case\ e\ of\ ✓(r) \Rightarrow \mathcal{D}\ (\Omega\ P\ r)$
$\quad | \ ev\ a \Rightarrow if\ ev\ a \in P^0\ then\ \{s.\ ev\ a\ \#\ s \in \mathcal{D}\ P\}\ else\ \mathcal{D}\ (\Psi\ P\ a))$›
⟨*proof*⟩

lemma *T-After$_{tick}$*:
‹$\mathcal{T}\ (P\ after_{✓}\ e) =$
$(case\ e\ of\ ✓(r) \Rightarrow \mathcal{T}\ (\Omega\ P\ r)$
$\quad | \ ev\ a \Rightarrow if\ ev\ a \in P^0\ then\ \{s.\ ev\ a\ \#\ s \in \mathcal{T}\ P\}\ else\ \mathcal{T}\ (\Psi\ P\ a))$›
⟨*proof*⟩

4.1.3 Monotony

lemma *mono-After$_{tick}$-T* :
‹$e \in Q^0 \Longrightarrow (case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r \sqsubseteq_T \Omega\ Q\ r) \Longrightarrow P \sqsubseteq_T Q \Longrightarrow P\ after_{✓}\ e \sqsubseteq_T Q\ after_{✓}\ e$›
and *mono-After$_{tick}$-F* :
‹$e \in Q^0 \Longrightarrow (case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r \sqsubseteq_F \Omega\ Q\ r) \Longrightarrow P \sqsubseteq_F Q \Longrightarrow P\ after_{✓}\ e \sqsubseteq_F Q\ after_{✓}\ e$›
and *mono-After$_{tick}$-D* :
‹$e \in Q^0 \Longrightarrow (case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r \sqsubseteq_D \Omega\ Q\ r) \Longrightarrow P \sqsubseteq_D Q \Longrightarrow P\ after_{✓}\ e \sqsubseteq_D Q\ after_{✓}\ e$›
and *mono-After$_{tick}$-FD* :
‹$e \in Q^0 \Longrightarrow (case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r \sqsubseteq_{FD} \Omega\ Q\ r) \Longrightarrow P \sqsubseteq_{FD} Q \Longrightarrow P\ after_{✓}\ e \sqsubseteq_{FD} Q\ after_{✓}\ e$›
and *mono-After$_{tick}$-DT* :
‹$e \in Q^0 \Longrightarrow (case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r \sqsubseteq_{DT} \Omega\ Q\ r) \Longrightarrow P \sqsubseteq_{DT} Q \Longrightarrow P\ after_{✓}\ e \sqsubseteq_{DT} Q\ after_{✓}\ e$›
⟨*proof*⟩

lemma *mono-After$_{tick}$* :
‹$[\![P \sqsubseteq Q;$
$(case\ e\ of\ ev\ a \Rightarrow (ev\ a \in Q^0 \lor \Psi\ P\ a \sqsubseteq \Psi\ Q\ a));$

$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r \sqsubseteq \Omega\ Q\ r)]\!] \Longrightarrow$
$P\ after_{✓}\ e \sqsubseteq Q\ after_{✓}\ e$›
⟨*proof*⟩

4.1.4 Behaviour of $After_{tick}$ with *STOP*, *SKIP* and ⊥

lemma $After_{tick}$*-STOP*: ‹$STOP\ after_{✓}\ e = (case\ e\ of\ ev\ a \Rightarrow \Psi\ STOP\ a \mid ✓(r) \Rightarrow \Omega\ STOP\ r)$›
and $After_{tick}$*-SKIP*: ‹$SKIP\ r\ after_{✓}\ e = (case\ e\ of\ ev\ a \Rightarrow \Psi\ (SKIP\ r)\ a \mid ✓(s) \Rightarrow \Omega\ (SKIP\ r)\ s)$›
and $After_{tick}$*-BOT* : ‹$\bot\ after_{✓}\ e = (case\ e\ of\ ev\ x \Rightarrow \bot \mid ✓(r) \Rightarrow \Omega\ \bot\ r)$›
⟨*proof*⟩

lemma $After_{tick}$*-is-BOT-iff*:
‹$P\ after_{✓}\ e = \bot \longleftrightarrow$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ P\ r = \bot$
$\mid ev\ a \Rightarrow if\ ev\ a \in P^0\ then\ [ev\ a] \in \mathcal{D}\ P\ else\ \Psi\ P\ a = \bot)$›
⟨*proof*⟩

4.1.5 Behaviour of $After_{tick}$ with Operators of `HOL-CSP`

Here again, we lose determinism.

lemma $After_{tick}$*-Mprefix-is-*$After_{tick}$*-Mndetprefix*:
‹$a \in A \Longrightarrow (\square a \in A \to P\ a)\ after_{✓}\ ev\ a = (\sqcap a \in A \to P\ a)\ after_{✓}\ ev\ a$›
⟨*proof*⟩

lemma $After_{tick}$*-Det-is-*$After_{tick}$*-Ndet*:
‹$ev\ a \in P^0 \cup Q^0 \Longrightarrow (P \square Q)\ after_{✓}\ ev\ a = (P \sqcap Q)\ after_{✓}\ ev\ a$›
⟨*proof*⟩

lemma $After_{tick}$*-Sliding-is-*$After_{tick}$*-Ndet*:
‹$ev\ a \in P^0 \cup Q^0 \Longrightarrow (P \rhd Q)\ after_{✓}\ ev\ a = (P \sqcap Q)\ after_{✓}\ ev\ a$›
⟨*proof*⟩

lemma $After_{tick}$*-Ndet*:
‹$(P \sqcap Q)\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (P \sqcap Q)\ r$
$\mid ev\ a \Rightarrow\ if\ ev\ a \in P^0 \cap Q^0$
$then\ P\ after_{✓}\ ev\ a \sqcap Q\ after_{✓}\ ev\ a$
$else\ \ if\ ev\ a \in P^0$
$then\ P\ after_{✓}\ ev\ a$
$else\ \ if\ ev\ a \in Q^0$
$then\ Q\ after_{✓}\ ev\ a$
$else\ \Psi\ (P \sqcap Q)\ a)$›
⟨*proof*⟩

lemma $After_{tick}$*-Det*:

‹$(P \square Q)$ $after_{✓}$ e =
($case$ e of ✓(r) ⇒ Ω ($P \square Q$) r
| ev a ⇒ if ev $a \in P^0 \cap Q^0$
$then$ P $after_{✓}$ ev a ⊓ Q $after_{✓}$ ev a
$else$ if ev $a \in P^0$
$then$ P $after_{✓}$ ev a
$else$ if ev $a \in Q^0$
$then$ Q $after_{✓}$ ev a
$else$ Ψ ($P \square Q$) a)›
⟨$proof$⟩

lemma $After_{tick}$-$Sliding$:
‹$(P \rhd Q)$ $after_{✓}$ e =
($case$ e of ✓(r) ⇒ Ω ($P \rhd Q$) r
| ev a ⇒ if ev $a \in P^0 \cap Q^0$
$then$ P $after_{✓}$ ev a ⊓ Q $after_{✓}$ ev a
$else$ if ev $a \in P^0$
$then$ P $after_{✓}$ ev a
$else$ if ev $a \in Q^0$
$then$ Q $after_{✓}$ ev a
$else$ Ψ ($P \rhd Q$) a)›
⟨$proof$⟩

lemma $After_{tick}$-$Mprefix$:
‹($\square$ $a \in A \to P$ a) $after_{✓}$ e =
($case$ e of ✓(r) ⇒ Ω ($\square$ $a \in A \to P$ a) r
| ev a ⇒ if $a \in A$ $then$ P a $else$ Ψ ($\square$ $a \in A \to P$ a) a)›
⟨$proof$⟩

lemma $After_{tick}$-$Mndetprefix$:
‹($\sqcap$ $a \in A \to P$ a) $after_{✓}$ e =
($case$ e of ✓(r) ⇒ Ω ($\sqcap$ $a \in A \to P$ a) r
| ev a ⇒ if $a \in A$ $then$ P a $else$ Ψ ($\sqcap$ $a \in A \to P$ a) a)›
⟨$proof$⟩

corollary $After_{tick}$-$write0$:
‹($a \to P$) $after_{✓}$ e =
($case$ e of ✓(r) ⇒ Ω ($a \to P$) r
| ev b ⇒ if $b = a$ $then$ P $else$ Ψ ($a \to P$) b)›
⟨$proof$⟩

corollary ‹($a \to P$) $after_{✓}$ ev a = P› ⟨$proof$⟩

corollary $After_{tick}$-$read$:
‹(c?$a \in A \to P$ a) $after_{✓}$ e =
($case$ e of ✓(r) ⇒ Ω (c?$a \in A \to P$ a) r
| ev b ⇒ if $b \in c$ ‘ A $then$ P (inv-$into$ A c b) $else$ Ψ (c?$a \in A \to P$ a) b)›
⟨$proof$⟩

corollary After_{tick}*-ndet-write*:
‹$(c!!a{\in}A \rightarrow P\ a)$ *after*$_{✓}$ e =
$($*case* e *of* $✓(r) \Rightarrow \Omega\ (c!!a{\in}A \rightarrow P\ a)\ r$
| *ev* $b \Rightarrow$ *if* $b \in c$ ‘ A *then* P (*inv-into* $A\ c\ b$) *else* $\Psi\ (c!!a{\in}A \rightarrow P\ a)\ b)$›
⟨*proof*⟩

lemma After_{tick}*-Seq*:
‹$(P\ ;\ Q)$ *after*$_{✓}$ e =
$($*case* e *of* $✓(r) \Rightarrow \Omega\ (P\ ;\ Q)\ r$
| *ev* $a \Rightarrow$ *if range tick* $\cap P^0 = \{\} \vee (\forall r.\ ✓(r) \in P^0 \longrightarrow ev\ a \notin Q^0)$
then if ev $a \in P^0$ *then* P *after*$_{✓}$ *ev* a ; Q *else* $\Psi\ (P\ ;\ Q)\ a$
else *if ev* $a \in P^0$ *then* $(P$ *after*$_{✓}$ *ev* a ; $Q) \sqcap Q$ *after*$_{✓}$ *ev* a
else Q *after*$_{✓}$ *ev* $a)$›
⟨*proof*⟩

lemma After_{tick}*-Sync*:
‹$(P\ [\![S]\!]\ Q)$ *after*$_{✓}$ e =
$($*case* e *of* $✓(r) \Rightarrow \Omega\ (P\ [\![S]\!]\ Q)\ r$
| *ev* $a \Rightarrow$ *if* $P = \bot \vee Q = \bot$ *then* $\bot$
else *if ev* $a \in P^0 \cap Q^0$
then *if* $a \in S$ *then* P *after*$_{✓}$ *ev* $a\ [\![S]\!]\ Q$ *after*$_{✓}$ *ev* a
else $(P$ *after*$_{✓}$ *ev* $a\ [\![S]\!]\ Q) \sqcap (P\ [\![S]\!]\ Q$ *after*$_{✓}$ *ev* $a)$
else *if ev* $a \in P^0 \wedge a \notin S$ *then* P *after*$_{✓}$ *ev* $a\ [\![S]\!]\ Q$
else *if ev* $a \in Q^0 \wedge a \notin S$ *then* $P\ [\![S]\!]\ Q$ *after*$_{✓}$ *ev* a
else $\Psi\ (P\ [\![S]\!]\ Q)\ a)$›
⟨*proof*⟩

lemma *Hiding-FD-Hiding-*After_{tick}*-if-initial-inside*:
‹*ev* $a \in P^0 \Longrightarrow a \in A \Longrightarrow P \setminus A \sqsubseteq_{FD} P$ *after*$_{✓}$ *ev* $a \setminus A$›
and After_{tick}*-Hiding-FD-Hiding-*After_{tick}*-if-initial-notin*:
‹*ev* $a \in P^0 \Longrightarrow a \notin A \Longrightarrow (P \setminus A)$ *after*$_{✓}$ *ev* $a \sqsubseteq_{FD} P$ *after*$_{✓}$ *ev* $a \setminus A$›
⟨*proof*⟩

lemmas *Hiding-F-Hiding-*After_{tick}*-if-initial-inside* =
Hiding-FD-Hiding-After_{tick}*-if-initial-inside*[*THEN leFD-imp-leF*]
and After_{tick}*-Hiding-F-Hiding-*After_{tick}*-if-initial-notin* =
After_{tick}*-Hiding-FD-Hiding-*After_{tick}*-if-initial-notin*[*THEN leFD-imp-leF*]
and *Hiding-D-Hiding-*After_{tick}*-if-initial-inside* =
Hiding-FD-Hiding-After_{tick}*-if-initial-inside*[*THEN leFD-imp-leD*]
and After_{tick}*-Hiding-D-Hiding-*After_{tick}*-if-initial-notin* =
After_{tick}*-Hiding-FD-Hiding-*After_{tick}*-if-initial-notin*[*THEN leFD-imp-leD*]
and *Hiding-T-Hiding-*After_{tick}*-if-initial-inside* =
Hiding-FD-Hiding-After_{tick}*-if-initial-inside*[*THEN leFD-imp-leF*, *THEN leF-imp-leT*]

and $After_{tick}$-*Hiding-T-Hiding-*$After_{tick}$-*if-initial-notin* =
$After_{tick}$-*Hiding-FD-Hiding-*$After_{tick}$-*if-initial-notin*[*THEN leFD-imp-leF*, *THEN leF-imp-leT*]

corollary *Hiding-DT-Hiding-*$After_{tick}$-*if-initial-inside*:
‹$ev\ a \in P^0 \Longrightarrow a \in A \Longrightarrow P \setminus A \sqsubseteq_{DT} P\ after_{✓}\ ev\ a \setminus A$›
and $After_{tick}$-*Hiding-DT-Hiding-*$After_{tick}$-*if-initial-notin*:
‹$ev\ a \in P^0 \Longrightarrow a \notin A \Longrightarrow (P \setminus A)\ after_{✓}\ ev\ a \sqsubseteq_{DT} P\ after_{✓}\ ev\ a \setminus A$›
⟨*proof*⟩

end

As with *After*, we need to "duplicate" the locale to formalize the result for the *Renaming* operator.

locale *AfterExtDuplicated* = $After_{tick\alpha}$: *AfterExt* Ψ_α Ω_α + $After_{tick\beta}$: *AfterExt* Ψ_β Ω_β
for Ψ_α :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω_α :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ψ_β :: ‹$[('b, 's)\ process_{ptick}, 'b] \Rightarrow ('b, 's)\ process_{ptick}$›
and Ω_β :: ‹$[('b, 's)\ process_{ptick}, 's] \Rightarrow ('b, 's)\ process_{ptick}$›

sublocale *AfterExtDuplicated* ⊆ *AfterDuplicated* ⟨*proof*⟩

context *AfterExtDuplicated*
begin

notation $After_{tick\alpha}$.*After* (**infixl** ‹$after_\alpha$› *86*)
notation $After_{tick\beta}$.*After* (**infixl** ‹$after_\beta$› *86*)
notation $After_{tick\alpha}$.$After_{tick}$ (**infixl** ‹$after_{✓\alpha}$› *86*)
notation $After_{tick\beta}$.$After_{tick}$ (**infixl** ‹$after_{✓\beta}$› *86*)

lemma $After_{tick}$-*Renaming*:
‹*Renaming P f g* $after_{✓\beta}$ e =
(*case e of* ✓(*s*) ⇒ Ω_β (*Renaming P f g*) *s*
| *ev b* ⇒ *if* $P = \bot$ *then* $\bot$
else *if* $\exists\, a.\ ev\ a \in P^0 \wedge f\ a = b$
then $\sqcap\ a \in \{a.\ ev\ a \in P^0 \wedge f\ a = b\}$. *Renaming* (*P* $after_\alpha$ *a*) *f g*
else Ψ_β (*Renaming P f g*) *b*)›
⟨*proof*⟩

end

context *AfterExt* — Back to *AfterExt*.
begin

4.1.6 Behaviour of $After_{tick}$ with Operators of HOL-CSPM

lemma $After_{tick}$-*GlobalDet-is-*$After_{tick}$-*GlobalNdet*:
$\langle ev\ a \in (\bigcup a \in A.\ (P\ a)^0) \Longrightarrow$
$(\square\ a \in A.\ P\ a)\ after_{✓}\ ev\ a = (\sqcap\ a \in A.\ P\ a)\ after_{✓}\ ev\ a\rangle$
⟨*proof*⟩

lemma $After_{tick}$-*GlobalNdet*:
$\langle(\sqcap\ a \in A.\ P\ a)\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (\sqcap\ a \in A.\ P\ a)\ r$
$\mid ev\ a \Rightarrow\ \ if\ ev\ a \in (\bigcup a \in A.\ (P\ a)^0)$
$then\ \sqcap\ x \in \{x \in A.\ ev\ a \in (P\ x)^0\}.\ P\ x\ after_{✓}\ ev\ a$
$else\ \Psi\ (\sqcap\ a \in A.\ P\ a)\ a)\rangle$
and $After_{tick}$-*GlobalDet*:
$\langle(\square\ a \in A.\ P\ a)\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (\square\ a \in A.\ P\ a)\ r$
$\mid ev\ a \Rightarrow\ \ if\ ev\ a \in (\bigcup a \in A.\ (P\ a)^0)$
$then\ \sqcap\ x \in \{x \in A.\ ev\ a \in (P\ x)^0\}.\ P\ x\ after_{✓}\ ev\ a$
$else\ \Psi\ (\square\ a \in A.\ P\ a)\ a)\rangle$
⟨*proof*⟩

lemma $After_{tick}$-*Throw*:
$\langle(P\ \Theta\ a \in A.\ Q\ a)\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (P\ \Theta\ a \in A.\ Q\ a)\ r$
$\mid ev\ a \Rightarrow\ \ if\ P = \bot\ then\ \bot$
$else\ \ if\ ev\ a \in P^0$
$then\ \ if\ a \in A$
$then\ Q\ a$
$else\ P\ after_{✓}\ ev\ a\ \Theta\ a \in A.\ Q\ a$
$else\ \Psi\ (P\ \Theta\ a \in A.\ Q\ a)\ a)\rangle$
⟨*proof*⟩

lemma $After_{tick}$-*Interrupt*:
$\langle(P \triangle Q)\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (P \triangle Q)\ r$
$\mid ev\ a \Rightarrow\ \ if\ ev\ a \in P^0 \cap Q^0$
$then\ Q\ after_{✓}\ ev\ a \sqcap (P\ after_{✓}\ ev\ a \triangle Q)$
$else\ \ if\ ev\ a \in P^0 \wedge ev\ a \notin Q^0$
$then\ P\ after_{✓}\ ev\ a \triangle Q$
$else\ \ if\ ev\ a \notin P^0 \wedge ev\ a \in Q^0$
$then\ Q\ after_{✓}\ ev\ a$
$else\ \Psi\ (P \triangle Q)\ a)\rangle$
⟨*proof*⟩

4.1.7 Behaviour of $After_{tick}$ with Reference Processes

lemma $After_{tick}$-DF:
‹$DF\ A\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (DF\ A)\ r$
$\mid ev\ a \Rightarrow if\ a \in A\ then\ DF\ A\ else\ \Psi\ (DF\ A)\ a)$›
⟨proof⟩

lemma $After_{tick}$-DF_{SKIPS}:
‹$DF_{SKIPS}\ A\ R\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (DF_{SKIPS}\ A\ R)\ r$
$\mid ev\ a \Rightarrow if\ a \in A\ then\ DF_{SKIPS}\ A\ R\ else\ \Psi\ (DF_{SKIPS}\ A\ R)\ a)$›
⟨proof⟩

lemma $After_{tick}$-RUN:
‹$RUN\ A\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (RUN\ A)\ r$
$\mid ev\ a \Rightarrow if\ a \in A\ then\ RUN\ A\ else\ \Psi\ (RUN\ A)\ a)$›
⟨proof⟩

lemma $After_{tick}$-$CHAOS$:
‹$CHAOS\ A\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (CHAOS\ A)\ r$
$\mid ev\ a \Rightarrow if\ a \in A\ then\ CHAOS\ A\ else\ \Psi\ (CHAOS\ A)\ a)$›
⟨proof⟩

lemma $After_{tick}$-$CHAOS_{SKIPS}$:
‹$CHAOS_{SKIPS}\ A\ R\ after_{✓}\ e =$
$(case\ e\ of\ ✓(r) \Rightarrow \Omega\ (CHAOS_{SKIPS}\ A\ R)\ r$
$\mid ev\ a \Rightarrow if\ a \in A\ then\ CHAOS_{SKIPS}\ A\ R\ else\ \Psi\ (CHAOS_{SKIPS}\ A\ R)$
$a)$›
⟨proof⟩

lemma DF-FD-$After_{tick}$:
‹$DF\ A \sqsubseteq_{FD} P \Longrightarrow e \in P^0 \Longrightarrow DF\ A \sqsubseteq_{FD} P\ after_{✓}\ e$›
⟨proof⟩

lemma DF_{SKIPS}-FD-$After_{tick}$:
‹$DF_{SKIPS}\ A\ R \sqsubseteq_{FD} P \Longrightarrow ev\ a \in P^0 \Longrightarrow DF_{SKIPS}\ A\ R \sqsubseteq_{FD} P\ after_{✓}\ ev\ a$›
⟨proof⟩

lemma $deadlock$-$free$-$After_{tick}$:
‹$e \in P^0 \Longrightarrow deadlock\text{-}free\ P \Longrightarrow$
$deadlock\text{-}free\ (P\ after_{✓}\ e) \longleftrightarrow$
$(case\ e\ of\ ev\ a \Rightarrow True \mid ✓(r) \Rightarrow deadlock\text{-}free\ (\Omega\ P\ r))$›
⟨proof⟩

lemma *deadlock-free*$_{SKIPS}$*-After*$_{tick}$:
‹$e \in P^0 \Longrightarrow$ *deadlock-free*$_{SKIPS}$ $P \Longrightarrow$
deadlock-free$_{SKIPS}$ (P *after*$_✓$ e) $\longleftrightarrow$
(*case e of ev a* $\Rightarrow$ *True* | ✓(r) $\Rightarrow$ *deadlock-free*$_{SKIPS}$ (Ω P r))›
⟨*proof*⟩

4.1.8 Characterizations for Deadlock Freeness

lemma *deadlock-free-imp-not-initial-tick*: ‹*deadlock-free* $P \Longrightarrow$ *range tick* $\cap P^0 = \{\}$›
⟨*proof*⟩

lemma *initial-tick-imp-range-ev-in-refusals*: ‹✓(r) $\in P^0 \Longrightarrow$ *range ev* $\in \mathcal{R}$ P›
⟨*proof*⟩

lemma *deadlock-free-After*$_{tick}$*-characterization*:
‹*deadlock-free* $P \longleftrightarrow$ *range ev* $\notin \mathcal{R}$ P $\wedge$ ($\forall$ *e*. *ev e* $\in P^0 \longrightarrow$ *deadlock-free* (P *after*$_✓$ *ev e*))›
(**is** ‹*deadlock-free* $P \longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

lemma *deadlock-free*$_{SKIPS}$*-After*$_{tick}$*-characterization*:
‹*deadlock-free*$_{SKIPS}$ $P \longleftrightarrow$
UNIV $\notin \mathcal{R}$ P $\wedge$ ($\forall e \in P^0 -$ *range tick*. *deadlock-free*$_{SKIPS}$ (P *after*$_✓$ e))›
(**is** ‹*deadlock-free*$_{SKIPS}$ $P \longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

4.1.9 Continuity

lemma *After*$_{tick}$*-cont* [*simp*] :
assumes *cont-*$\Psi\Omega$: ‹*case e of ev a* $\Rightarrow$ *cont* (λP. Ψ P a) | ✓(r) $\Rightarrow$ *cont* (λP. Ω P r)›
and *cont-f* : ‹*cont f*›
shows ‹*cont* (λx. *f x after*$_✓$ e)›
⟨*proof*⟩

end

4.2 The After trace Operator

context *AfterExt*
begin

4.2.1 Definition

We just defined P *after*$_✓$ *e* for P::('*a*, '*r*) *process*$_{ptick}$ and *e*::('*a*, '*r*) *event*$_{ptick}$. Since a trace of a P is just an ('*a*, '*r*) *event*$_{ptick}$ *list*, the following inductive definition is natural.

fun $After_{trace}$ :: ‹('*a*, '*r*) *process*$_{ptick}$ ⇒ ('*a*, '*r*) *trace*$_{ptick}$ ⇒ ('*a*, '*r*) *process*$_{ptick}$›
(**infixl** ‹*after*$_{\mathcal{T}}$› *86*)
  where ‹P *after*$_{\mathcal{T}}$ [] = P›
  | ‹P *after*$_{\mathcal{T}}$ (e # t) = P *after*$_✓$ e *after*$_{\mathcal{T}}$ t›

We can also induct backward.

lemma *After*$_{trace}$*-append*: ‹P *after*$_{\mathcal{T}}$ (t @ u) = P *after*$_{\mathcal{T}}$ t *after*$_{\mathcal{T}}$ u›
  ⟨*proof*⟩

lemma *After*$_{trace}$*-snoc* : ‹P *after*$_{\mathcal{T}}$ (t @ [e]) = P *after*$_{\mathcal{T}}$ t *after*$_✓$ e›
  ⟨*proof*⟩

4.2.2 Projections

lemma *F-After*$_{trace}$:
  ‹*tF* $t \Longrightarrow (t\ @\ u,\ X) \in \mathcal{F}\ P \Longrightarrow (u,\ X) \in \mathcal{F}$ (P *after*$_{\mathcal{T}}$ t)›
⟨*proof*⟩

lemma *D-After*$_{trace}$:
  ‹*tF* $t \Longrightarrow t\ @\ u \in \mathcal{D}\ P \Longrightarrow u \in \mathcal{D}$ (P *after*$_{\mathcal{T}}$ t)›
⟨*proof*⟩

lemma *T-After*$_{trace}$: ‹$t\ @\ u \in \mathcal{T}\ P \Longrightarrow u \in \mathcal{T}$ (P *after*$_{\mathcal{T}}$ t)›
⟨*proof*⟩

corollary *initials-After*$_{trace}$:
  ‹$t\ @\ e\ \#\ u \in \mathcal{T}\ P \Longrightarrow e \in$ (P *after*$_{\mathcal{T}}$ t)^0›
  ⟨*proof*⟩

corollary *F-imp-R-After*$_{trace}$: ‹*tF* $t \Longrightarrow (t,\ X) \in \mathcal{F}\ P \Longrightarrow X \in \mathcal{R}$ (P *after*$_{\mathcal{T}}$ t)›
  ⟨*proof*⟩

corollary *D-imp-After*$_{trace}$*-is-BOT*: ‹*tF* $t \Longrightarrow t \in \mathcal{D}\ P \Longrightarrow P$ *after*$_{\mathcal{T}}$ $t = \bot$›
  ⟨*proof*⟩

lemma *F-After*$_{trace}$*-eq*:

‹$t \in \mathcal{T}\ P \Longrightarrow tF\ t \Longrightarrow \mathcal{F}\ (P\ after_{\mathcal{T}}\ t) = \{(u, X).\ (t\ @\ u, X) \in \mathcal{F}\ P\}$›
⟨*proof*⟩

lemma *D-After$_{trace}$-eq*:
‹$t \in \mathcal{T}\ P \Longrightarrow tF\ t \Longrightarrow \mathcal{D}\ (P\ after_{\mathcal{T}}\ t) = \{u.\ t\ @\ u \in \mathcal{D}\ P\}$›
⟨*proof*⟩

lemma *T-After$_{trace}$-eq*:
‹$t \in \mathcal{T}\ P \Longrightarrow tF\ t \Longrightarrow \mathcal{T}\ (P\ after_{\mathcal{T}}\ t) = \{u.\ t\ @\ u \in \mathcal{T}\ P\}$›
⟨*proof*⟩

4.2.3 Monotony

4.2.4 Four inductive Constructions with *After$_{trace}$*

Reachable Processes

inductive-set *reachable-processes* :: ‹$('a, 'r)\ process_{ptick} \Rightarrow ('a, 'r)\ process_{ptick}$ *set*› (‹$\mathcal{R}_{proc}$›)
for P :: ‹$('a, 'r)\ process_{ptick}$›
where *reachable-self* : ‹$P \in \mathcal{R}_{proc}\ P$›
| *reachable-after*: ‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow ev\ e \in Q^0 \Longrightarrow Q\ after\ e \in \mathcal{R}_{proc}\ P$›

lemma *reachable-processes-is*: ‹$\mathcal{R}_{proc}\ P = \{Q.\ \exists\, t \in \mathcal{T}\ P.\ tF\ t \wedge Q = P\ after_{\mathcal{T}}\ t\}$›
⟨*proof*⟩

lemma *reachable-processes-trans*: ‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow R \in \mathcal{R}_{proc}\ Q \Longrightarrow R \in \mathcal{R}_{proc}\ P$›
⟨*proof*⟩

Antecedent Processes

inductive-set *antecedent-processes* :: ‹$('a, 'r)\ process_{ptick} \Rightarrow ('a, 'r)\ process_{ptick}$ *set*› (‹$\mathcal{A}_{proc}$›)
for P :: ‹$('a, 'r)\ process_{ptick}$›
where *antecedent-self* : ‹$P \in \mathcal{A}_{proc}\ P$›
| *antecedent-after*: ‹$Q\ after\ e \in \mathcal{A}_{proc}\ P \Longrightarrow ev\ e \in Q^0 \Longrightarrow Q \in \mathcal{A}_{proc}\ P$›

lemma *antecedent-processes-is*: ‹$\mathcal{A}_{proc}\ P = \{Q.\ \exists\, t \in \mathcal{T}\ Q.\ tF\ t \wedge P = Q\ after_{\mathcal{T}}\ t\}$›
⟨*proof*⟩

lemma *antecedent-processes-trans*: ‹$Q \in \mathcal{A}_{proc}\ P \Longrightarrow R \in \mathcal{A}_{proc}\ Q \Longrightarrow R \in \mathcal{A}_{proc}\ P$›
⟨*proof*⟩

corollary *antecedent-processes-iff-rev-reachable-processes*: ‹$P \in \mathcal{A}_{proc}\ Q \longleftrightarrow Q \in \mathcal{R}_{proc}\ P$›
⟨*proof*⟩

4.2.5 Nth initials Events

primrec *nth-initials* :: ‹$('a, 'r)\ process_{ptick} \Rightarrow nat \Rightarrow ('a, 'r)\ event_{ptick}\ set$› (‹-⁻› [*1000*, *3*] *999*)
where ‹$P^{0} = P^0$›
| ‹$P^{Suc\ n} = \bigcup\ \{(P\ after\ e)^n\ | e.\ ev\ e \in P^0\}$›

lemma ‹$P^{0} = P^0$› ⟨*proof*⟩

lemma *first-initials-is* : ‹$P^{1} = \bigcup\ \{(P\ after\ e)^0\ | e.\ ev\ e \in P^0\}$› ⟨*proof*⟩

lemma *second-initials-is* :
‹$P^{2} = \bigcup\ \{(P\ after\ e\ after\ f)^0\ | e\ f.\ ev\ e \in P^0 \wedge ev\ f \in (P\ after\ e)^0\}$›
⟨*proof*⟩

lemma *third-initials-is* :
‹$P^{3} = \bigcup\ \{(P\ after\ e\ after\ f\ after\ g)^0\ | e\ f\ g.$
$ev\ e \in P^0 \wedge ev\ f \in (P\ after\ e)^0 \wedge ev\ g \in (P\ after\ e\ after\ f)^0\}$›
⟨*proof*⟩

More generally, we have the following result.

lemma *nth-initials-is*: ‹$P^n = \bigcup\ \{(P\ after_{\mathcal{T}}\ t)^0\ |\ t.\ t \in \mathcal{T}\ P \wedge tF\ t \wedge length\ t = n\}$›
⟨*proof*⟩

lemma *nth-initials-DF*: ‹$(DF\ A)^n = ev\ `\ A$›
⟨*proof*⟩

lemma *nth-initials-DF*$_{SKIPS}$:
‹$(DF_{SKIPS}\ A\ R)^n = (if\ A = \{\}\ then\ if\ n = 0\ then\ tick\ `\ R\ else\ \{\}\ else\ ev\ `\ A \cup tick\ `\ R)$›
⟨*proof*⟩

lemma *nth-initials-RUN*: ‹$(RUN\ A)^n = ev\ `\ A$›
⟨*proof*⟩

lemma *nth-initials-CHAOS*: ‹$(CHAOS\ A)^n = ev\ `\ A$›
⟨*proof*⟩

lemma *nth-initials-CHAOS*$_{SKIPS}$:
‹$(CHAOS_{SKIPS}\ A\ R)^n = (if\ A = \{\}\ then\ if\ n = 0\ then\ tick\ `\ R\ else\ \{\}\ else\ ev$

‘ $A \cup$ *tick* ‘ R)›
⟨*proof*⟩

Reachable Ev

inductive *reachable-ev* :: ‹(′a, ′r) $process_{ptick}$ ⇒ ′a ⇒ bool›
where *initial-ev-reachable*:
‹$ev\ a \in P^0 \Longrightarrow$ *reachable-ev* $P\ a$›
| *reachable-ev-after-reachable*:
‹$ev\ b \in P^0 \Longrightarrow$ *reachable-ev* (P *after* b) $a \Longrightarrow$ *reachable-ev* $P\ a$›

definition *reachable-ev-set* :: ‹(′a, ′r) $process_{ptick}$ ⇒ ′a set› (‹$\mathcal{R}_{ev}$›)
where ‹$\mathcal{R}_{ev}\ P \equiv \bigcup Q \in \mathcal{R}_{proc}\ P.\ \{a.\ ev\ a \in Q^0\}$›

lemma *reachable-ev-BOT* : ‹*reachable-ev* ⊥ a›
and *not-reachable-ev-STOP* : ‹¬ *reachable-ev* *STOP* a›
and *not-reachable-ev-SKIP* : ‹¬ *reachable-ev* (*SKIP* r) a›
⟨*proof*⟩

lemma *events-of-iff-reachable-ev*: ‹$a \in \alpha(P) \longleftrightarrow$ *reachable-ev* $P\ a$›
⟨*proof*⟩

lemma *reachable-ev-iff-in-initials-$After_{trace}$-for-some-tickFree-T*:
‹*reachable-ev* $P\ a \longleftrightarrow (\exists\, t \in \mathcal{T}\ P.\ tF\ t \wedge ev\ a \in (P\ after_{\mathcal{T}}\ t)^0)$›
⟨*proof*⟩

Properties

corollary *reachable-ev-set-is-mem-Collect-reachable-ev*:
‹$\mathcal{R}_{ev}\ P = \{a.$ *reachable-ev* $P\ a\}$›
⟨*proof*⟩

corollary *events-of-is-reachable-ev-set*: ‹$\alpha(P) = \mathcal{R}_{ev}\ P$›
⟨*proof*⟩

lemma *events-of-reachable-processes-subset*: ‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow \alpha(Q) \subseteq \alpha(P)$›
⟨*proof*⟩

corollary *events-of-antecedent-processes-superset*: ‹$Q \in \mathcal{A}_{proc}\ P \Longrightarrow \alpha(P) \subseteq \alpha(Q)$›
⟨*proof*⟩

lemma *events-of-is-Union-nth-initials*: ‹$\alpha(P) = (\bigcup n.\ \{a.\ ev\ a \in P^n\})$›
⟨*proof*⟩

Reachable Tick

inductive $reachable\text{-}tick$ :: ‹('a, 'r) $process_{ptick}$ ⇒ 'r ⇒ bool›
  where $initial\text{-}tick\text{-}reachable$:
  ‹$\checkmark(r) \in P^0 \Longrightarrow reachable\text{-}tick\ P\ r$›
  | $reachable\text{-}tick\text{-}after\text{-}reachable$:
  ‹$ev\ a \in P^0 \Longrightarrow reachable\text{-}tick\ (P\ after\ a)\ r \Longrightarrow reachable\text{-}tick\ P\ r$›

definition $reachable\text{-}tick\text{-}set$ :: ‹('a, 'r) $process_{ptick}$ ⇒ 'r set› (‹$\mathcal{R}_\checkmark$›)
  where ‹$\mathcal{R}_\checkmark\ P \equiv \bigcup Q \in \mathcal{R}_{proc}\ P.\ \{r.\ \checkmark(r) \in Q^0\}$›

lemma $reachable\text{-}tick\text{-}BOT$: ‹$reachable\text{-}tick\ \bot\ r$›
  and $not\text{-}reachable\text{-}tick\text{-}STOP$: ‹$\neg\ reachable\text{-}tick\ STOP\ s$›
  and $reachable\text{-}tick\text{-}SKIP\text{-}iff$: ‹$reachable\text{-}tick\ (SKIP\ r)\ s \longleftrightarrow s = r$›
  ⟨*proof*⟩

lemma $ticks\text{-}of\text{-}iff\text{-}reachable\text{-}tick$: ‹$r \in \checkmark s(P) \longleftrightarrow reachable\text{-}tick\ P\ r$›
⟨*proof*⟩

lemma $reachable\text{-}tick\text{-}iff\text{-}in\text{-}initials\text{-}After_{trace}\text{-}for\text{-}some\text{-}tickFree\text{-}T$:
  ‹$reachable\text{-}tick\ P\ r \longleftrightarrow (\exists\, t \in \mathcal{T}\ P.\ tF\ t \wedge \checkmark(r) \in (P\ after_{\mathcal{T}}\ t)^0)$›
⟨*proof*⟩

Properties

corollary $reachable\text{-}tick\text{-}set\text{-}is\text{-}mem\text{-}Collect\text{-}reachable\text{-}tick$:
  ‹$\mathcal{R}_\checkmark\ P = \{a.\ reachable\text{-}tick\ P\ a\}$›
  ⟨*proof*⟩

corollary $ticks\text{-}of\text{-}is\text{-}reachable\text{-}tick\text{-}set$: ‹$\checkmark s(P) = \mathcal{R}_\checkmark\ P$›
  ⟨*proof*⟩

lemma $ticks\text{-}of\text{-}reachable\text{-}processes\text{-}subset$: ‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow \checkmark s(Q) \subseteq \checkmark s(P)$›
  ⟨*proof*⟩

corollary $ticks\text{-}of\text{-}antecedent\text{-}processes\text{-}superset$: ‹$Q \in \mathcal{A}_{proc}\ P \Longrightarrow \checkmark s(P) \subseteq \checkmark s(Q)$›
  ⟨*proof*⟩

lemma $ticks\text{-}of\text{-}is\text{-}Union\text{-}nth\text{-}initials$: ‹$\checkmark s(P) = (\bigcup n.\ \{r.\ \checkmark(r) \in P^n\})$›
  ⟨*proof*⟩

4.2.6 Characterizations for Deadlock Freeness

Remember that we have characterized *deadlock-free P* with an equality involving ($after_\checkmark$): $deadlock\text{-}free\ P = (range\ ev \notin \mathcal{R}\ P \wedge (\forall\, e.\ ev\ e \in P^0$

$\longrightarrow$ *deadlock-free* $(P$ *after*$_{\checkmark}$ *ev* $e)))$. This can of course be derived in a characterization involving $(after_{\mathcal{T}})$.

lemma *deadlock-free-After$_{trace}$-characterization*:
‹*deadlock-free* $P \longleftrightarrow (\forall t \in \mathcal{T}\ P.\ range\ ev \notin \mathcal{R}_a\ P\ t \wedge (t \neq [] \longrightarrow$ *deadlock-free* $(P\ after_{\mathcal{T}}\ t)))$›
⟨*proof*⟩

lemma *deadlock-free$_{SKIPS}$-After$_{trace}$-characterization*:
‹*deadlock-free*$_{SKIPS}$ $P \longleftrightarrow$
$(\forall t \in \mathcal{T}\ P.\ tF\ t \longrightarrow UNIV \notin \mathcal{R}_a\ P\ t \wedge (t \neq [] \longrightarrow$ *deadlock-free*$_{SKIPS}$ $(P$ $after_{\mathcal{T}}\ t)))$›
⟨*proof*⟩

Actually, with *After$_{trace}$*, we can obtain much more powerful results. This will be developed later.

4.2.7 Continuity

lemma *After$_{trace}$-cont* :
‹$[\![\forall a.\ ev\ a \in set\ t \longrightarrow cont\ (\lambda P.\ \Psi\ P\ a);$
$\forall r.\ \checkmark(r) \in set\ t \longrightarrow cont\ (\lambda P.\ \Omega\ P\ r);\ cont\ f]\!] \Longrightarrow$
$cont\ (\lambda x.\ f\ x\ after_{\mathcal{T}}\ t)$›
⟨*proof*⟩

end

Chapter 5

Motivations for our Definitions

To construct our bridge between denotational and operational semantics, we want to define two kind of transitions:

- without external event: $P \leadsto_\tau P'$
- with the terminating event $\checkmark(r)$: $P \leadsto_{\checkmark r} P'$
- with a non terminating external event *ev e*: $P \leadsto_e P'$.

  We will discuss in this theory some fundamental properties that we want

  $P \leadsto_\tau Q$, $P \leadsto_e P'$ and $P \leadsto_{\checkmark r} P'$ to verify, and 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: $P \leadsto_\tau P$
- the non-deterministic choice ($\sqcap$) can make a τ transition to the left side $P \sqcap Q \leadsto_\tau P$
- the non-deterministic choice ($\sqcap$) can make a τ transition to the right side $P \sqcap Q \leadsto_\tau Q$.

inductive τ*-trans* :: ‹(′*a*, ′*r*) $process_{ptick}$ ⇒ (′*a*, ′*r*) $process_{ptick}$ ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*)
  where τ*-trans-eq* : ‹$P \leadsto_\tau P$›
  | τ*-trans-NdetL* : ‹$P \sqcap Q \leadsto_\tau P$›

| τ*-trans-NdetR* : ‹$P \sqcap Q \rightsquigarrow_\tau Q$›

— 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*: ‹$(\rightsquigarrow_\tau) = (\sqsubseteq_{FD})$›
⟨*proof*⟩

The definition of the event transition will be a little bit more complex.
First of all we want to prevent a process P::('a, 'r) *process*$_{ptick}$ to make a transition with *ev* (e::'a) (resp. ✓(r::'r)) if P can not begin with *ev e* (resp. ✓(r)).
More formally, we want $P \rightsquigarrow_e P' \Longrightarrow$ *ev e* $\in P^0$ (resp. $P \rightsquigarrow_{✓r} P' \Longrightarrow$ ✓(r) $\in P^0$).
Moreover, we want the event transitions to absorb the τ transitions.
Finally, when $e \in P^0$ (resp. ✓(r) $\in P^0$), we want to have $P \rightsquigarrow_e P$ *after*$_✓$ *ev e* (resp. $P \rightsquigarrow_{✓r} P$ *after*$_✓$ ✓(r)).
This brings us to the following inductive definition:

inductive *event-trans-prem* :: ‹('a, 'r) *process*$_{ptick}$ ⇒ ('a, 'r) *event*$_{ptick}$ ⇒ ('a, 'r) *process*$_{ptick}$ ⇒ *bool*›
where
τ*-left-absorb* : ‹⟦$e \in$ *initials* P'; $P \rightsquigarrow_\tau P'$; *event-trans-prem* P' e P''⟧
$\Longrightarrow$ *event-trans-prem* P e P''›
| τ*-right-absorb* : ‹⟦$e \in$ *initials* P; *event-trans-prem* P e P'; $P' \rightsquigarrow_\tau P''$⟧
$\Longrightarrow$ *event-trans-prem* P e P''›
| *initial-trans-to-After*$_{tick}$: ‹$e \in$ *initials* $P \Longrightarrow$ *event-trans-prem* P e (P *after*$_✓$ e)›

abbreviation *event-trans* :: ‹('a, 'r) *process*$_{ptick}$ ⇒ 'a ⇒ ('a, 'r) *process*$_{ptick}$ ⇒ *bool*›
(‹- $\rightsquigarrow_{-}$ -› [*50*, *3*, *51*] *50*)
where ‹$P \rightsquigarrow_e P' \equiv$ *ev e* $\in$ *initials* $P \wedge$ *event-trans-prem* P (*ev e*) P'›

abbreviation *tick-trans* :: ‹('a, 'r) *process*$_{ptick}$ ⇒ 'r ⇒ ('a, 'r) *process*$_{ptick}$ ⇒ *bool*› (‹- $\rightsquigarrow_{✓-}$ -› [*50*, *3*, *51*] *50*)
where ‹$P \rightsquigarrow_{✓r} P' \equiv$ ✓(r) $\in P^0 \wedge$ *event-trans-prem* P ✓(r) P'›

We immediately show that, under the assumption of monotony of Ω, this event transition definition is equivalent to the following:

lemma *startable-imp-ev-trans-is-startable-and-FD-After*:
‹(*case e of ev x* ⇒ $P \rightsquigarrow_x P'$ | ✓(r) ⇒ $P \rightsquigarrow_{✓r} P'$) $\longleftrightarrow e \in P^0 \wedge P$ *after*$_✓$ $e \rightsquigarrow_\tau P'$›
if ‹$\bigwedge P\ Q$. *case e of* ✓(r) ⇒ $\Omega\ P\ r \rightsquigarrow_\tau \Omega\ Q\ r$›

$\langle proof \rangle$

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

- $P \leadsto_{\tau} Q \equiv P \sqsubseteq_{FD} Q$
- $P \leadsto_{e} P' \equiv ev\ e \in P^0 \wedge P\ after_{\checkmark}\ ev\ e \leadsto_{\tau} Q$
- $P \leadsto_{\checkmark r} P' \equiv \checkmark(r) \in P^0 \wedge P\ after_{\checkmark}\ \checkmark(r) \leadsto_{\tau} Q$

and possible variations with other refinements.

But we want to make the construction as general as possible. Therefore we will continue with the locale mechanism, eventually adding additional required assumptions for each operator, and we will instantiate with refinements at the end.

Chapter 6

Generic Operational Semantics as a Locale

Some Properties of Monotony **lemma** *FD-iff-eq-Ndet*: ‹$P \sqsubseteq_{FD} Q \longleftrightarrow P = P \sqcap Q$›
⟨*proof*⟩

lemma *non-BOT-mono-Det-F* :
‹$P = \bot \lor P' \neq \bot \Longrightarrow Q = \bot \lor Q' \neq \bot \Longrightarrow P \sqsubseteq_F P' \Longrightarrow Q \sqsubseteq_F Q' \Longrightarrow P \Box Q \sqsubseteq_F P' \Box Q'$›
⟨*proof*⟩

lemma *non-BOT-mono-Det-left-F* : ‹$P = \bot \lor P' \neq \bot \lor Q = \bot \Longrightarrow P \sqsubseteq_F P' \Longrightarrow P \Box Q \sqsubseteq_F P' \Box Q$ ›
and *non-BOT-mono-Det-right-F* : ‹$Q = \bot \lor Q' \neq \bot \lor P = \bot \Longrightarrow Q \sqsubseteq_F Q' \Longrightarrow P \Box Q \sqsubseteq_F P \Box Q'$›
⟨*proof*⟩

lemma *non-BOT-mono-Sliding-F* :
‹$P = \bot \lor P' \neq \bot \lor Q = \bot \Longrightarrow P \sqsubseteq_F P' \Longrightarrow Q \sqsubseteq_F Q' \Longrightarrow P \rhd Q \sqsubseteq_F P' \rhd Q'$›
⟨*proof*⟩

6.1 Definition

locale *OpSemTransitions* = *AfterExt* Ψ Ω
for Ψ :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
— Just declaring the types $'a$ and $'r$. +
fixes τ*-trans* :: ‹$[('a, 'r)\ process_{ptick}, ('a, 'r)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹$\leadsto_\tau$› *50*)
assumes τ*-trans-NdetL*: ‹$P \sqcap Q \leadsto_\tau P$›
and τ*-trans-transitivity*: ‹$P \leadsto_\tau Q \Longrightarrow Q \leadsto_\tau R \Longrightarrow P \leadsto_\tau R$›
and τ*-trans-anti-mono-initials*: ‹$P \leadsto_\tau Q \Longrightarrow initials\ Q \subseteq initials\ P$›

and τ-*trans-mono-After*$_{tick}$: ‹$e \in$ *initials* $Q \Longrightarrow P \rightsquigarrow_\tau Q \Longrightarrow P$ *after*$_\checkmark$ $e \rightsquigarrow_\tau$ Q *after*$_\checkmark$ e›

begin

This locale needs to be instantiated with:

- a function Ψ::$('a, 'r)$ *process*$_{ptick}$ $\Rightarrow$ $'a$ $\Rightarrow$ $('a, 'r)$ *process*$_{ptick}$ that is a placeholder for the value of P *after* e when $ev\ e \notin P^0$
- a function Ω::$('a, 'r)$ *process*$_{ptick}$ $\Rightarrow$ $'r$ $\Rightarrow$ $('a, 'r)$ *process*$_{ptick}$ that is a placeholder for the value of P *after*$_\checkmark$ $\checkmark(r)$
- a binary relation ($\rightsquigarrow_\tau$) which:
  - is compatible with ($\sqcap$)
  - is transitive
  - makes *initials* anti-monotonic
  - makes *After*$_{tick}$ monotonic.

From the τ transition $P \rightsquigarrow_\tau Q$ we derive the event transition as follows:

abbreviation *ev-trans* :: ‹[$('a, 'r)$ *process*$_{ptick}$, $'a$, $('a, 'r)$ *process*$_{ptick}$] $\Rightarrow$ *bool*›
(‹- $\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
where ‹$P \rightsquigarrow_e Q \equiv ev\ e \in P^0 \wedge P$ *after*$_\checkmark$ $ev\ e \rightsquigarrow_\tau Q$›

abbreviation *tick-trans* :: ‹[$('a, 'r)$ *process*$_{ptick}$, $'r$, $('a, 'r)$ *process*$_{ptick}$] $\Rightarrow$ *bool*›
(‹- $\rightsquigarrow_\checkmark$- -› [*50*, *3*, *51*] *50*)
where ‹$P \rightsquigarrow_{\checkmark r} Q \equiv \checkmark(r) \in P^0 \wedge P$ *after*$_\checkmark$ $\checkmark(r) \rightsquigarrow_\tau Q$›

lemma *ev-trans-is*: ‹$P \rightsquigarrow_e Q \longleftrightarrow ev\ e \in$ *initials* $P \wedge P$ *after* $e \rightsquigarrow_\tau Q$›
⟨*proof*⟩

lemma *tick-trans-is*: ‹$P \rightsquigarrow_{\checkmark r} Q \longleftrightarrow \checkmark(r) \in P^0 \wedge \Omega\ P\ r \rightsquigarrow_\tau Q$›
⟨*proof*⟩

lemma *reverse-event-trans-is*:
‹$e \in P^0 \wedge P$ *after*$_\checkmark$ $e \rightsquigarrow_\tau Q \longleftrightarrow$ (*case* e *of* $\checkmark(r) \Rightarrow P \rightsquigarrow_{\checkmark r} Q \mid ev\ x \Rightarrow P \rightsquigarrow_x Q$)›
⟨*proof*⟩

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

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

lemma *BOT-ev-trans-anything*: ‹$\bot \leadsto_e P$›
and *BOT-tick-trans*: ‹$\bot \leadsto_{\checkmark r} \Omega \bot r$›
⟨*proof*⟩

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

lemma *ev-trans-τ-trans*: ‹$P \leadsto_e P' \Longrightarrow P' \leadsto_\tau P'' \Longrightarrow P \leadsto_e P''$›
and *tick-trans-τ-trans*: ‹$P \leadsto_{\checkmark r} P' \Longrightarrow P' \leadsto_\tau P'' \Longrightarrow P \leadsto_{\checkmark r} P''$›
⟨*proof*⟩

lemma *τ-trans-ev-trans*: ‹$P \leadsto_\tau P' \Longrightarrow P' \leadsto_e P'' \Longrightarrow P \leadsto_e P''$›
and *τ-trans-tick-trans*: ‹$P \leadsto_\tau P' \Longrightarrow P' \leadsto_{\checkmark r} P'' \Longrightarrow P \leadsto_{\checkmark r} P''$›
⟨*proof*⟩

We can also add these result which will be useful later.

lemma *initial-tick-imp-τ-trans-SKIP*: ‹$P \leadsto_\tau SKIP\ r$› **if** ‹$\checkmark(r) \in P^0$›
⟨*proof*⟩

lemma *exists-tick-trans-is-initial-tick*: ‹$(\exists P'.\ P \leadsto_{\checkmark r} P') \longleftrightarrow \checkmark(r) \in P^0$›
⟨*proof*⟩

There is also a major property we can already prove.

lemma *τ-trans-imp-leT* : ‹$P \leadsto_\tau Q \Longrightarrow P \sqsubseteq_T Q$›
⟨*proof*⟩

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

inductive *trace-trans* :: ‹$('a, 'r)\ process_{ptick} \Rightarrow ('a, 'r)\ trace_{ptick} \Rightarrow ('a, 'r)\ process_{ptick} \Rightarrow bool$›
(‹- $\leadsto^*$- -› [*50*, *3*, *51*] *50*)
where *trace-τ-trans* : ‹$P \leadsto_\tau P' \Longrightarrow P \leadsto^* []\ P'$›
| *trace-tick-trans* : ‹$P \leadsto_{\checkmark r} P' \Longrightarrow P \leadsto^* [\checkmark(r)]\ P'$›
| *trace-Cons-ev-trans* : ‹$P \leadsto_e P' \Longrightarrow P' \leadsto^* s\ P'' \Longrightarrow P \leadsto^* (ev\ e) \# s\ P''$›

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

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

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

6.2 Consequences of $P \leadsto^* s\ Q$ on $\mathcal{F}$, $\mathcal{T}$ and $\mathcal{D}$

lemma *trace-trans-imp-F-if-τ-trans-imp-leF*:

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

lemma *trace-trans-imp-T*: $\langle P \leadsto^* s\ Q \Longrightarrow s \in \mathcal{T}\ P\rangle$
⟨*proof*⟩

lemma *tickFree-trace-trans-BOT-imp-D-if-τ-trans-BOT-imp-eq-BOT-weak*:
$\langle$*tickFree* $s \Longrightarrow P \leadsto^* s\ \bot \Longrightarrow s \in \mathcal{D}\ P\rangle$
if $\langle\forall\, P.\ P \leadsto_\tau\ \bot \longrightarrow P = \bot\rangle$
⟨*proof*⟩

6.3 Characterizations for $P \leadsto^* s\ Q$

lemma *trace-trans-iff* :
$\langle P \leadsto^*\ []\ Q \longleftrightarrow P \leadsto_\tau\ Q\rangle$
$\langle P \leadsto^*\ [\checkmark(r)]\ Q \longleftrightarrow P \leadsto_{\checkmark r}\ Q\rangle$
$\langle P \leadsto^*\ (ev\ e)\ \#\ s\ Q' \longleftrightarrow (\exists\, Q.\ P \leadsto_e\ Q \wedge Q \leadsto^*\ s\ Q')\rangle$
$\langle(P \leadsto^*\ s\ @\ [f]\ Q') \longleftrightarrow$
tickFree $s \wedge (\exists\, Q.\ P \leadsto^* s\ Q \wedge (case\ f\ of\ \checkmark(r) \Rightarrow Q \leadsto_{\checkmark r}\ Q'\ |\ ev\ x \Rightarrow Q \leadsto_x\ Q'))\rangle$
$\langle$*front-tickFree* $(s\ @\ t) \Longrightarrow (P \leadsto^* s\ @\ t\ Q') \longleftrightarrow (\exists\, Q.\ P \leadsto^* s\ Q \wedge Q \leadsto^* t\ Q')\rangle$
⟨*proof*⟩

6.4 Finally: $P \leadsto^* s\ Q$ is $P\ after_\mathcal{T}\ s \leadsto_\tau\ Q$

theorem *trace-trans-iff-T-and-After*$_{trace}$*-τ-trans* :
$\langle(P \leadsto^* s\ Q) \longleftrightarrow s \in \mathcal{T}\ P \wedge P\ after_\mathcal{T}\ s \leadsto_\tau\ Q\rangle$
⟨*proof*⟩

As corollaries we obtain the reciprocal results of

$[\![\forall\, P\ Q.\ P \leadsto_\tau\ Q \longrightarrow P \sqsubseteq_F\ Q;\ ?P \leadsto^* ?s\ ?Q;\ ?X \in \mathcal{R}\ ?Q]\!] \Longrightarrow (?s,\ ?X) \in \mathcal{F}\ ?P$

$?P \leadsto^* ?s\ ?Q \Longrightarrow\ ?s \in \mathcal{T}\ ?P$

$[\![\forall\, P.\ P \leadsto_\tau\ \bot \longrightarrow P = \bot;\ tF\ ?s;\ ?P \leadsto^* ?s\ \bot]\!] \Longrightarrow\ ?s \in \mathcal{D}\ ?P$

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

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

lemma *tickFree-D-imp-trace-trans-BOT*: $\langle$*tickFree* $s \Longrightarrow s \in \mathcal{D}\ P \Longrightarrow P \leadsto^* s\ \bot\rangle$

⟨*proof*⟩

And therefore, we obtain equivalences.

lemma *F-trace-trans-reality-check-weak*:
‹$\forall P\ Q.\ P \rightsquigarrow_\tau Q \longrightarrow P \sqsubseteq_F Q \Longrightarrow \mathit{tickFree}\ s \Longrightarrow$
$(s,\ X) \in \mathcal{F}\ P \longleftrightarrow (\exists\, Q.\ (P \rightsquigarrow^* s\ Q) \wedge X \in \mathcal{R}\ Q)$›
⟨*proof*⟩

lemma *T-trace-trans-reality-check*: ‹$s \in \mathcal{T}\ P \longleftrightarrow (\exists\, Q.\ P \rightsquigarrow^* s\ Q)$›
⟨*proof*⟩

lemma *D-trace-trans-reality-check-weak*:
‹$\forall P.\ P \rightsquigarrow_\tau \bot \longrightarrow P = \bot \Longrightarrow \mathit{tickFree}\ s \Longrightarrow s \in \mathcal{D}\ P \longleftrightarrow P \rightsquigarrow^* s\ \bot$›
⟨*proof*⟩

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

lemma *STOP-trace-trans-iff*: ‹$\mathit{STOP} \rightsquigarrow^* s\ P \longleftrightarrow s = [] \wedge P = \mathit{STOP}$›
⟨*proof*⟩

lemma Ω-*SKIP-is-STOP-imp-τ-trans-imp-leF-imp-SKIP-trace-trans-iff*:
‹$\Omega\ (\mathit{SKIP}\ r)\ r = \mathit{STOP} \Longrightarrow (\bigwedge P\ Q.\ P \rightsquigarrow_\tau Q \Longrightarrow P \sqsubseteq_F Q) \Longrightarrow$
$(\mathit{SKIP}\ r \rightsquigarrow^* s\ P) \longleftrightarrow s = [] \wedge P = \mathit{SKIP}\ r \vee s = [✓(r)] \wedge P = \mathit{STOP}$›
⟨*proof*⟩

lemma *trace-trans-imp-initials-subset-initials-After$_{trace}$*:
‹$P \rightsquigarrow^* s\ Q \Longrightarrow \mathit{initials}\ Q \subseteq \mathit{initials}\ (P\ \mathit{after}_\mathcal{T}\ s)$›
⟨*proof*⟩

lemma *imp-trace-trans-imp-initial*:
‹$P \rightsquigarrow^*(s\ @\ e\ \#\ t)\ Q \Longrightarrow e \in \mathit{initials}\ (P\ \mathit{after}_\mathcal{T}\ s)$›
⟨*proof*⟩

Under additional assumptions, we can show that the event transition and the trace transition are admissible.

lemma *ev-trans-adm-weak*[*simp*]:
assumes *τ-trans-adm*:
‹$\bigwedge u\ v.\ \mathit{cont}\ (u :: 'b :: \mathit{cpo} \Rightarrow ('a,\ 'r)\ \mathit{process}_{ptick}) \Longrightarrow \mathit{monofun}\ v \Longrightarrow \mathit{adm}(\lambda x.\ u\ x \rightsquigarrow_\tau v\ x)$›
and Ψ-*cont-hyp* : ‹$\mathit{cont}\ (\lambda P.\ \Psi\ P\ e)$›
and *cont-u*: ‹$\mathit{cont}\ (u :: 'b \Rightarrow ('a,\ 'r)\ \mathit{process}_{ptick})$› **and** *monofun-v* : ‹$\mathit{monofun}\ v$›
shows ‹$\mathit{adm}(\lambda x.\ u\ x \rightsquigarrow_e (v\ x))$›
⟨*proof*⟩

lemma *tick-trans-adm-weak*[*simp*]:
assumes *τ-trans-adm*:

⟨$\bigwedge u\ v.\ cont\ (u :: 'b :: cpo \Rightarrow ('a, 'r)\ process_{ptick}) \Longrightarrow monofun\ v \Longrightarrow adm(\lambda x.\ u\ x \leadsto_\tau v\ x)$⟩
and Ω-*cont-hyp* : ⟨$cont\ (\lambda P.\ \Omega\ P\ r)$⟩
and *cont-u*: ⟨$cont\ (u :: 'b \Rightarrow ('a, 'r)\ process_{ptick})$⟩ **and** *monofun-v* : ⟨$monofun\ v$⟩
shows ⟨$adm(\lambda x.\ u\ x \leadsto_{\checkmark r} (v\ x))$⟩
⟨*proof*⟩

lemma *trace-trans-adm-weak*[*simp*]:
assumes τ-*trans-adm*: ⟨$\bigwedge u\ v.\ cont\ (u :: 'b :: cpo \Rightarrow ('a, 'r)\ process_{ptick}) \Longrightarrow monofun\ v \Longrightarrow adm\ (\lambda x.\ u\ x \leadsto_\tau v\ x)$⟩
and $After_{trace}$-*cont-hyps*: ⟨$\forall x.\ ev\ x \in set\ s \longrightarrow cont\ (\lambda P.\ \Psi\ P\ x)$⟩ ⟨$\forall r.\ \checkmark(r) \in set\ s \longrightarrow cont\ (\lambda P.\ \Omega\ P\ r)$⟩
and *cont-u*: ⟨$cont\ (u :: 'b :: cpo \Rightarrow ('a, 'r)\ process_{ptick})$⟩ **and** *monofun-v*: ⟨$monofun\ v$⟩
shows ⟨$adm\ (\lambda x.\ u\ x \leadsto^* s\ (v\ x))$⟩
⟨*proof*⟩

6.5 General Rules of Operational Semantics

We can now derive some rules or the operational semantics that we are defining.

lemma *SKIP-trans-tick-*Ω*-SKIP*: ⟨$SKIP\ r \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$⟩
⟨*proof*⟩

lemmas *SKIP-OpSem-rule* = *SKIP-trans-tick-*Ω*-SKIP*

This is quite obvious, but we can get better.

lemma *initial-tick-imp-tick-trans-*Ω*-SKIP*: ⟨$\checkmark(r) \in P^0 \Longrightarrow P \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$⟩
⟨*proof*⟩

lemma *tick-trans-imp-tick-trans-*Ω*-SKIP* : ⟨$\exists P'.\ P \leadsto_{\checkmark r} P' \Longrightarrow P \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$⟩
⟨*proof*⟩

lemma *SKIP-cant-ev-trans*: ⟨$\neg\ SKIP\ r \leadsto_e P$⟩
and *STOP-cant-ev-trans*: ⟨$\neg\ STOP \leadsto_e P$⟩
and *STOP-cant-tick-trans*: ⟨$\neg\ STOP \leadsto_{\checkmark r} P$⟩ ⟨*proof*⟩

lemma *ev-trans-Mprefix*: ‹$e \in A \Longrightarrow \square a \in A \to P\ a \rightsquigarrow_e (P\ e)$›
⟨*proof*⟩

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

lemma *ev-trans-prefix*: ‹$e \to P \rightsquigarrow_e P$›
⟨*proof*⟩

lemmas *prefix-OpSem-rules* = *ev-trans-prefix ev-trans-Mprefix ev-trans-Mndetprefix*

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

lemmas *Ndet-OpSem-rules* = *τ-trans-NdetL τ-trans-NdetR τ-trans-GlobalNdet*

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

lemmas *fix-point-OpSem-rule* = *τ-trans-fix-point*

lemma *ev-trans-DetL*: ‹$P \rightsquigarrow_e P' \Longrightarrow P \square Q \rightsquigarrow_e P'$›
⟨*proof*⟩

lemma *ev-trans-DetR*: ‹$Q \rightsquigarrow_e Q' \Longrightarrow P \square Q \rightsquigarrow_e Q'$›
⟨*proof*⟩

lemma *tick-trans-DetL*: ‹$P \rightsquigarrow_{\checkmark r} P' \Longrightarrow P \square Q \rightsquigarrow_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
⟨*proof*⟩

lemma *tick-trans-DetR*: ‹$Q \rightsquigarrow_{\checkmark r} Q' \Longrightarrow P \square Q \rightsquigarrow_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
⟨*proof*⟩

lemma *ev-trans-GlobalDet*:
‹$\square a \in A.\ P\ a \rightsquigarrow_e Q$› **if** ‹$a \in A$› **and** ‹$P\ a \rightsquigarrow_e Q$›
⟨*proof*⟩

lemma *tick-trans-GlobalDet*:
‹$\square a \in A.\ P\ a \rightsquigarrow_{\checkmark r} \Omega\ (SKIP\ r)\ r$› **if** ‹$a \in A$› **and** ‹$P\ a \rightsquigarrow_{\checkmark r} Q$›
⟨*proof*⟩

lemma *ev-trans-SlidingL*: ‹$P \leadsto_e P' \Longrightarrow P \rhd Q \leadsto_e P'$›
⟨*proof*⟩

lemma *tick-trans-SlidingL*: ‹$P \leadsto_{\checkmark r} P' \Longrightarrow P \rhd Q \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
⟨*proof*⟩

lemma *τ-trans-SlidingR*: ‹$P \rhd Q \leadsto_\tau Q$›
⟨*proof*⟩

lemma *τ-trans-SeqR*: ‹$P \mathbin{;} Q \leadsto_\tau Q'$› **if** ‹$P \leadsto_{\checkmark r} P'$› **and** ‹$Q \leadsto_\tau Q'$›
⟨*proof*⟩

lemma ‹$\checkmark(r) \in P^0 \Longrightarrow Q \leadsto_e Q' \Longrightarrow P \mathbin{;} Q \leadsto_e Q'$›
⟨*proof*⟩

lemma *tick-trans-Hiding*: ‹$P \leadsto_{\checkmark r} P' \Longrightarrow P \setminus B \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
⟨*proof*⟩

lemma *τ-trans-SKIP-SyncL*: ‹$P\ [\![S]\!]\ Q \leadsto_\tau SKIP\ r\ [\![S]\!]\ Q$› **if** ‹$P \leadsto_{\checkmark r} P'$›
⟨*proof*⟩

lemma *τ-trans-SKIP-SyncR*: ‹$Q \leadsto_{\checkmark r} Q' \Longrightarrow P\ [\![S]\!]\ Q \leadsto_\tau P\ [\![S]\!]\ SKIP\ r$›
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync-SKIP*: ‹$SKIP\ r\ [\![S]\!]\ SKIP\ r \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
⟨*proof*⟩

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

lemma *tick-trans-InterruptL* : ‹$P \leadsto_{\checkmark r} P' \Longrightarrow P \triangle Q \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
and *tick-trans-InterruptR* : ‹$Q \leadsto_{\checkmark r} Q' \Longrightarrow P \triangle Q \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$›

⟨*proof*⟩

lemma *tick-trans-ThrowL* : ‹$P \leadsto_{\checkmark r} P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a \leadsto_{\checkmark r} \Omega\ (SKIP\ r)\ r$›
⟨*proof*⟩

lemma *ev-trans-ThrowR-inside*:
‹$e \in A \Longrightarrow P \leadsto_e P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a \leadsto_e (Q\ e)$›
⟨*proof*⟩

end

6.6 Recovering other operational rules

By adding a τ-transition hypothesis on each operator, we can recover the remaining operational rules.

6.6.1 *Det* Laws

locale *OpSemTransitionsDet* = *OpSemTransitions* Ψ Ω ‹$(\leadsto_\tau)$›
for Ψ :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
and *τ-trans* :: ‹$[('a, 'r)\ process_{ptick}, ('a, 'r)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes *τ-trans-DetL* : ‹$P \leadsto_\tau P' \Longrightarrow P \square Q \leadsto_\tau P' \square Q$›
begin

lemma *τ-trans-DetR* : ‹$Q \leadsto_\tau Q' \Longrightarrow P \square Q \leadsto_\tau P \square Q'$›
⟨*proof*⟩

lemmas *Det-OpSem-rules* = *τ-trans-DetL τ-trans-DetR*
ev-trans-DetL ev-trans-DetR
tick-trans-DetL tick-trans-DetR

end

6.6.2 *Det* relaxed Laws

locale *OpSemTransitionsDetRelaxed* = *OpSemTransitions* Ψ Ω ‹$(\leadsto_\tau)$›
for Ψ :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
and *τ-trans* :: ‹$[('a, 'r)\ process_{ptick}, ('a, 'r)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes *τ-trans-DetL* : ‹$P = \bot \lor P' \neq \bot \lor Q = \bot \Longrightarrow P \leadsto_\tau P' \Longrightarrow P \square Q \leadsto_\tau P' \square Q$›
begin

lemma τ*-trans-DetR* : ‹$Q = \bot \vee Q' \neq \bot \vee Q = \bot \Longrightarrow Q \rightsquigarrow_\tau Q' \Longrightarrow P \,\Box\, Q \rightsquigarrow_\tau P \,\Box\, Q'$›
⟨*proof*⟩

lemmas *Det-OpSem-rules* = τ*-trans-DetL* τ*-trans-DetR*
ev-trans-DetL ev-trans-DetR
tick-trans-DetL tick-trans-DetR

end

6.6.3 *Seq* Laws

locale *OpSemTransitionsSeq* = *OpSemTransitions* Ψ Ω ‹($\rightsquigarrow_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ*-trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\rightsquigarrow_\tau$› *50*) +
assumes τ*-trans-SeqL* : ‹$P \rightsquigarrow_\tau P' \Longrightarrow P \,\mathbf{;}\, Q \rightsquigarrow_\tau P' \,\mathbf{;}\, Q$›
begin

lemma *ev-trans-SeqL*: ‹$P \rightsquigarrow_e P' \Longrightarrow P \,\mathbf{;}\, Q \rightsquigarrow_e P' \,\mathbf{;}\, Q$›
⟨*proof*⟩

lemmas *Seq-OpSem-rules* = τ*-trans-SeqL ev-trans-SeqL* τ*-trans-SeqR*

end

6.6.4 *Renaming* Laws

We are used to it now: we need to duplicate the locale in order to obtain the rules for the *Renaming* operator.

locale *OpSemTransitionsDuplicated* =
$OpSemTransitions_\alpha$: *OpSemTransitions* Ψ_α Ω_α ‹($_\alpha\rightsquigarrow_\tau$)› +
$OpSemTransitions_\beta$: *OpSemTransitions* Ψ_β Ω_β ‹($_\beta\rightsquigarrow_\tau$)›
for Ψ_α :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω_α :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ-$trans_\alpha$ :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\alpha\rightsquigarrow_\tau$› *50*)
and Ψ_β :: ‹[('b, 's) $process_{ptick}$, 'b] ⇒ ('b, 's) $process_{ptick}$›
and Ω_β :: ‹[('b, 's) $process_{ptick}$, 's] ⇒ ('b, 's) $process_{ptick}$›
and τ-$trans_\beta$ :: ‹[('b, 's) $process_{ptick}$, ('b, 's) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\beta\rightsquigarrow_\tau$› *50*)
begin

notation $OpSemTransitions_\alpha$*.ev-trans* (‹- $_\alpha\rightsquigarrow_{-}$ -› [*50*, *3*, *51*] *50*)
notation $OpSemTransitions_\alpha$*.tick-trans* (‹- $_\alpha\rightsquigarrow_{\checkmark -}$ -› [*50*, *3*, *51*] *50*)
notation $OpSemTransitions_\beta$*.ev-trans* (‹- $_\beta\rightsquigarrow_{-}$ -› [*50*, *3*, *51*] *50*)
notation $OpSemTransitions_\beta$*.tick-trans* (‹- $_\beta\rightsquigarrow_{\checkmark -}$ -› [*50*, *3*, *51*] *50*)

lemma *tick-trans-Renaming*: ‹$P \;{}_{\alpha}{\leadsto}_{\checkmark r}\; P' \Longrightarrow Renaming\ P\ f\ g \;{}_{\beta}{\leadsto}_{\checkmark g\ r}\; (\Omega_\beta\ (SKIP\ (g\ r))\ (g\ r))$›
⟨*proof*⟩

end

sublocale *OpSemTransitionsDuplicated* ⊆ *AfterExtDuplicated* ⟨*proof*⟩

locale *OpSemTransitionsRenaming* =
OpSemTransitionsDuplicated $\Psi_\alpha\ \Omega_\alpha\ \tau\text{-}trans_\alpha\ \Psi_\beta\ \Omega_\beta\ \tau\text{-}trans_\beta$
for Ψ_α :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω_α :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
and $\tau\text{-}trans_\alpha$ :: ‹$[('a, 'r)\ process_{ptick}, ('a, 'r)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹${}_{\alpha}{\leadsto}_{\tau}$› *50*)
and Ψ_β :: ‹$[('b, 's)\ process_{ptick}, 'b] \Rightarrow ('b, 's)\ process_{ptick}$›
and Ω_β :: ‹$[('b, 's)\ process_{ptick}, 's] \Rightarrow ('b, 's)\ process_{ptick}$›
and $\tau\text{-}trans_\beta$ :: ‹$[('b, 's)\ process_{ptick}, ('b, 's)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹${}_{\beta}{\leadsto}_{\tau}$› *50*) +
assumes *τ-trans-Renaming* : ‹$P \;{}_{\alpha}{\leadsto}_{\tau}\; P' \Longrightarrow Renaming\ P\ f\ g \;{}_{\beta}{\leadsto}_{\tau}\; Renaming\ P'\ f\ g$›
begin

lemma *ev-trans-Renaming*: ‹$Renaming\ P\ f\ g \;{}_{\beta}{\leadsto}_{b}\; (Renaming\ P'\ f\ g)$›
if ‹$f\ a = b$› **and** ‹$P \;{}_{\alpha}{\leadsto}_{a}\; P'$›
⟨*proof*⟩

lemmas *Renaming-OpSem-rules* = *τ-trans-Renaming tick-trans-Renaming ev-trans-Renaming*

end

6.6.5 *Hiding* Laws

locale *OpSemTransitionsHiding* = *OpSemTransitions* Ψ Ω ‹$(\leadsto_\tau)$›
for Ψ :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
and *τ-trans* :: ‹$[('a, 'r)\ process_{ptick}, ('a, 'r)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes *τ-trans-Hiding* : ‹$P \leadsto_\tau P' \Longrightarrow P \setminus A \leadsto_\tau P' \setminus A$›
begin

lemma *τ-trans-Hiding-inside*: ‹$P \setminus A \leadsto_\tau P' \setminus A$› **if** ‹$e \in A$› **and** ‹$P \leadsto_e P'$›
⟨*proof*⟩

lemma *ev-trans-Hiding-notin*: ‹$P \setminus A \leadsto_e P' \setminus A$› **if** ‹$e \notin A$› **and** ‹$P \leadsto_e P'$›
⟨*proof*⟩

lemmas *Hiding-OpSem-rules* = τ*-trans-Hiding tick-trans-Hiding*
ev-trans-Hiding-notin τ*-trans-Hiding-inside*

end

6.6.6 *Sync* Laws

locale *OpSemTransitionsSync* = *OpSemTransitions* Ψ Ω ‹($\rightsquigarrow_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ*-trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\rightsquigarrow_\tau$› *50*) +
assumes τ*-trans-SyncL* : ‹$P \rightsquigarrow_\tau P' \Longrightarrow P [\![S]\!] Q \rightsquigarrow_\tau P' [\![S]\!] Q$›
begin

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

lemma *ev-trans-SyncL* : ‹$e \notin S \Longrightarrow P \rightsquigarrow_e P' \Longrightarrow P [\![S]\!] Q \rightsquigarrow_e P' [\![S]\!] Q$›
⟨*proof*⟩

lemma *ev-trans-SyncR* : ‹$e \notin S \Longrightarrow Q \rightsquigarrow_e Q' \Longrightarrow P [\![S]\!] Q \rightsquigarrow_e P [\![S]\!] Q'$›
⟨*proof*⟩

lemma *ev-trans-SyncLR* :
‹$e \in S \Longrightarrow P \rightsquigarrow_e P' \Longrightarrow Q \rightsquigarrow_e Q' \Longrightarrow P [\![S]\!] Q \rightsquigarrow_e P' [\![S]\!] Q'$›
⟨*proof*⟩

lemmas *Sync-OpSem-rules* = τ*-trans-SyncL* τ*-trans-SyncR*
ev-trans-SyncL ev-trans-SyncR
ev-trans-SyncLR
τ*-trans-SKIP-SyncL* τ*-trans-SKIP-SyncR*
tick-trans-SKIP-Sync-SKIP

end

6.6.7 *Sliding* Laws

locale *OpSemTransitionsSliding* = *OpSemTransitions* Ψ Ω ‹($\rightsquigarrow_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ*-trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\rightsquigarrow_\tau$› *50*) +
assumes τ*-trans-SlidingL* : ‹$P \rightsquigarrow_\tau P' \Longrightarrow P \rhd Q \rightsquigarrow_\tau P' \rhd Q$›
— We just add the τ*-trans-SlidingL* property.
begin

lemmas *Sliding-OpSem-rules* = τ*-trans-SlidingR* τ*-trans-SlidingL* *ev-trans-SlidingL tick-trans-SlidingL*

end

6.6.8 *Sliding* relaxed Laws

locale *OpSemTransitionsSlidingRelaxed* = *OpSemTransitions* Ψ Ω ‹($\leadsto_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ*-trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes τ*-trans-SlidingL* : ‹$P = \bot \vee P' \neq \bot \vee Q = \bot \Longrightarrow P \leadsto_\tau P' \Longrightarrow P \rhd Q \leadsto_\tau P' \rhd Q$›
— We just add the τ*-trans-SlidingL* property.
begin

lemmas *Sliding-OpSem-rules* = τ*-trans-SlidingR* τ*-trans-SlidingL*
ev-trans-SlidingL tick-trans-SlidingL

end

6.6.9 *Interrupt* Laws

locale *OpSemTransitionsInterruptL* = *OpSemTransitions* Ψ Ω ‹($\leadsto_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ*-trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes τ*-trans-InterruptL* : ‹$P \leadsto_\tau P' \Longrightarrow P \triangle Q \leadsto_\tau P' \triangle Q$ ›
begin

lemma *ev-trans-InterruptL*: ‹$P \leadsto_e P' \Longrightarrow P \triangle Q \leadsto_e P' \triangle Q$›
⟨*proof*⟩

lemma *ev-trans-InterruptR*: ‹$Q \leadsto_e Q' \Longrightarrow P \triangle Q \leadsto_e Q'$›
⟨*proof*⟩

end

locale *OpSemTransitionsInterrupt* = *OpSemTransitionsInterruptL* Ψ Ω ‹($\leadsto_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ*-trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes τ*-trans-InterruptR* : ‹$Q \leadsto_\tau Q' \Longrightarrow P \triangle Q \leadsto_\tau P \triangle Q'$›

— We just add the τ-*trans-InterruptR* property.
begin

lemmas *Interrupt-OpSem-rules* = τ-*trans-InterruptL* τ-*trans-InterruptR*
ev-trans-InterruptL ev-trans-InterruptR
tick-trans-InterruptL tick-trans-InterruptR

end

6.6.10 *Throw* Laws

locale *OpSemTransitionsThrow* = *OpSemTransitions* Ψ Ω ‹($\leadsto_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ-*trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*) +
assumes τ-*trans-ThrowL* : ‹$P \leadsto_\tau P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a \leadsto_\tau P'\ \Theta\ a \in A.\ Q\ a$›
begin

lemma *ev-trans-ThrowL-notin*:
‹$e \notin A \Longrightarrow P \leadsto_e P' \Longrightarrow P\ \Theta\ a \in A.\ Q\ a \leadsto_e (P'\ \Theta\ a \in A.\ Q\ a)$›
⟨*proof*⟩

lemmas *Throw-OpSem-rules* = τ-*trans-ThrowL tick-trans-ThrowL*
ev-trans-ThrowL-notin ev-trans-ThrowR-inside

end

6.7 Locales, Assemble !

It is now time to assemble our locales.

locale *OpSemTransitionsAll* =
OpSemTransitionsDet Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsSeq Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsHiding Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsSync Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsSliding Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsInterrupt Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsThrow Ψ Ω ‹($\leadsto_\tau$)›
for Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and τ-*trans* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*)

Of course we need to duplicate the locale for obtaining *Renaming* rules.

locale *OpSemTransitionsAllDuplicated* =
OpSemTransitionsAll$_\alpha$: *OpSemTransitionsAll* Ψ_α Ω_α ‹$({}_\alpha\rightsquigarrow_\tau)$› +
OpSemTransitionsAll$_\beta$: *OpSemTransitionsAll* Ψ_β Ω_β ‹$({}_\beta\rightsquigarrow_\tau)$› +
OpSemTransitionsRenaming Ψ_α Ω_α τ-*trans*$_\alpha$ Ψ_β Ω_β τ-*trans*$_\beta$
for Ψ_α :: ‹[('a, 'r) *process*$_{ptick}$, 'a] ⇒ ('a, 'r) *process*$_{ptick}$›
and Ω_α :: ‹[('a, 'r) *process*$_{ptick}$, 'r] ⇒ ('a, 'r) *process*$_{ptick}$›
and τ-*trans*$_\alpha$ :: ‹[('a, 'r) *process*$_{ptick}$, ('a, 'r) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹${}_\alpha\rightsquigarrow_\tau$› *50*)
and Ψ_β :: ‹[('b, 's) *process*$_{ptick}$, 'b] ⇒ ('b, 's) *process*$_{ptick}$›
and Ω_β :: ‹[('b, 's) *process*$_{ptick}$, 's] ⇒ ('b, 's) *process*$_{ptick}$›
and τ-*trans*$_\beta$ :: ‹[('b, 's) *process*$_{ptick}$, ('b, 's) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹${}_\beta\rightsquigarrow_\tau$› *50*)
begin

notation *OpSemTransitionsAll*$_\alpha$.*ev-trans* (‹- ${}_\alpha\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitionsAll*$_\alpha$.*tick-trans* (‹- ${}_\alpha\rightsquigarrow_✓$- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitionsAll*$_\beta$.*ev-trans* (‹- ${}_\beta\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitionsAll*$_\beta$.*tick-trans* (‹- ${}_\beta\rightsquigarrow_✓$- -› [*50*, *3*, *51*] *50*)

end

6.8 $(\rightsquigarrow_\tau)$ instantiated with $(\sqsubseteq_{FD})$ or $(\sqsubseteq_{DT})$

6.8.1 $(\rightsquigarrow_\tau)$ instantiated with $(\sqsubseteq_{FD})$

locale *OpSemFD* =
fixes Ψ :: ‹[('a, 'r) *process*$_{ptick}$, 'a] ⇒ ('a, 'r) *process*$_{ptick}$›
and Ω :: ‹[('a, 'r) *process*$_{ptick}$, 'r] ⇒ ('a, 'r) *process*$_{ptick}$›
assumes *mono*-Ω-*FD*: ‹$✓(r) \in Q^0 \Longrightarrow P \sqsubseteq_{FD} Q \Longrightarrow \Omega\ P\ r \sqsubseteq_{FD} \Omega\ Q\ r$›

sublocale *OpSemFD* ⊆ *OpSemTransitionsAll* - - ‹$(\sqsubseteq_{FD})$ :: ('a, 'r) *process*$_{ptick}$ ⇒ ('a, 'r) *process*$_{ptick}$ ⇒ *bool*›
⟨*proof*⟩

context *OpSemFD*
begin

Finally, the only remaining hypothesis is $[\![✓(?r) \in ?Q^0;\ ?P \sqsubseteq_{FD} ?Q]\!] \Longrightarrow \Omega\ ?P\ ?r \sqsubseteq_{FD} \Omega\ ?Q\ ?r$ when we instantiate our locale with the failure-divergence refinement $(\sqsubseteq_{FD})$.

Of course, we can strengthen some previous results.

notation *failure-divergence-refine* (**infixl** ‹${}_{FD}\rightsquigarrow_\tau$› *50*)
notation *ev-trans* (‹- ${}_{FD}\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
notation *tick-trans* (‹- ${}_{FD}\rightsquigarrow_✓$- -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹- ${}_{FD}\rightsquigarrow^*$- -› [*50*, *3*, *51*] *50*)

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

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

lemma *F-trace-trans-reality-check*: ‹$\mathit{tickFree}\ s \Longrightarrow (s,\ X) \in \mathcal{F}\ P \longleftrightarrow (\exists\, Q.\ (P\ {}_{FD}{\leadsto}^{*}s\ Q) \wedge X \in \mathcal{R}\ Q)$›
⟨*proof*⟩

lemma *D-trace-trans-reality-check*: ‹$\mathit{tickFree}\ s \Longrightarrow s \in \mathcal{D}\ P \longleftrightarrow P\ {}_{FD}{\leadsto}^{*}s\ \bot$›
⟨*proof*⟩

lemma *Ω-SKIP-is-STOP-imp-SKIP-trace-trans-iff*:
‹$\Omega\ (\mathit{SKIP}\ r)\ r = \mathit{STOP} \Longrightarrow (\mathit{SKIP}\ r\ {}_{FD}{\leadsto}^{*}s\ P) \longleftrightarrow s = [] \wedge P = \mathit{SKIP}\ r \vee s = [\checkmark(r)] \wedge P = \mathit{STOP}$›
⟨*proof*⟩

lemmas *τ-trans-adm* = *le-FD-adm*

lemma *ev-trans-adm*[*simp*]:
‹$[\![\mathit{cont}\ (\lambda P.\ \Psi\ P\ e);\ \mathit{cont}\ u;\ \mathit{monofun}\ v]\!] \Longrightarrow \mathit{adm}\ (\lambda x.\ u\ x\ {}_{FD}{\leadsto}_{e}\ v\ x)$›
⟨*proof*⟩

lemma *tick-trans-adm*[*simp*]:
‹$[\![\mathit{cont}\ (\lambda P.\ \Omega\ P\ r);\ \mathit{cont}\ u;\ \mathit{monofun}\ v]\!] \Longrightarrow \mathit{adm}\ (\lambda x.\ u\ x\ {}_{FD}{\leadsto}_{\checkmark r}\ v\ x)$›
⟨*proof*⟩

lemma *trace-trans-adm*[*simp*]:
‹$[\![\forall\, x.\ \mathit{ev}\ x \in \mathit{set}\ s \longrightarrow \mathit{cont}\ (\lambda P.\ \Psi\ P\ x);$
$\forall\, r.\ \checkmark(r) \in \mathit{set}\ s \longrightarrow \mathit{cont}\ (\lambda P.\ \Omega\ P\ r);\ \mathit{cont}\ u;\ \mathit{monofun}\ v]\!]$
$\Longrightarrow \mathit{adm}\ (\lambda x.\ u\ x\ {}_{FD}{\leadsto}^{*}s\ (v\ x))$›
⟨*proof*⟩

end

locale *OpSemFDDuplicated* =
$\mathit{OpSemFD}_{\alpha}$: *OpSemFD* $\Psi_{\alpha}\ \Omega_{\alpha}$ + $\mathit{OpSemFD}_{\beta}$: *OpSemFD* $\Psi_{\beta}\ \Omega_{\beta}$
for Ψ_{α} :: ‹$[('a,\ 'r)\ \mathit{process}_{ptick},\ 'a] \Rightarrow ('a,\ 'r)\ \mathit{process}_{ptick}$›
and Ω_{α} :: ‹$[('a,\ 'r)\ \mathit{process}_{ptick},\ 'r] \Rightarrow ('a,\ 'r)\ \mathit{process}_{ptick}$›
and Ψ_{β} :: ‹$[('b,\ 's)\ \mathit{process}_{ptick},\ 'b] \Rightarrow ('b,\ 's)\ \mathit{process}_{ptick}$›
and Ω_{β} :: ‹$[('b,\ 's)\ \mathit{process}_{ptick},\ 's] \Rightarrow ('b,\ 's)\ \mathit{process}_{ptick}$›

sublocale *OpSemFDDuplicated* ⊆ *OpSemTransitionsAllDuplicated* - - ‹$(\sqsubseteq_{FD})$› - - ‹$(\sqsubseteq_{FD})$›
⟨*proof*⟩

6.8.2 $(\leadsto_\tau)$ instantiated with $(\sqsubseteq_{DT})$

locale *OpSemDT* =
fixes Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
assumes *mono-Ω-DT*: ‹$\checkmark(r) \in Q^0 \Longrightarrow P \sqsubseteq_{DT} Q \Longrightarrow \Omega\ P\ r \sqsubseteq_{DT} \Omega\ Q\ r$›

sublocale *OpSemDT* ⊆ *OpSemTransitionsAll* - - ‹$(\sqsubseteq_{DT})$ :: ('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $process_{ptick}$ ⇒ *bool*›
⟨*proof*⟩

context *OpSemDT*
begin

Finally, the only remaining hypothesis is $[\![\checkmark(?r) \in ?Q^0;\ ?P \sqsubseteq_{DT} ?Q]\!] \Longrightarrow \Omega\ ?P\ ?r \sqsubseteq_{DT} \Omega\ ?Q\ ?r$ when we instantiate our locale with the failure-divergence refinement $(\sqsubseteq_{DT})$.

Of course, we can strengthen some previous results.

notation *trace-divergence-refine* (**infixl** ‹${}_{DT}\leadsto_\tau$› *50*)
notation *ev-trans* (‹- ${}_{DT}\leadsto$- -› [*50*, *3*, *51*] *50*)
notation *tick-trans* (‹- ${}_{DT}\leadsto_\checkmark$- -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹- ${}_{DT}\leadsto^*$- -› [*50*, *3*, *51*] *50*)

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

lemma *D-trace-trans-reality-check*: ‹$tickFree\ s \Longrightarrow s \in \mathcal{D}\ P \longleftrightarrow P\ {}_{DT}\leadsto^* s\ \bot$›
⟨*proof*⟩

lemmas *τ-trans-adm* = *le-DT-adm*

lemma *ev-trans-adm*[*simp*]:
‹$[\![cont\ (\lambda P.\ \Psi\ P\ e);\ cont\ u;\ monofun\ v]\!] \Longrightarrow adm\ (\lambda x.\ u\ x\ {}_{DT}\leadsto_e\ v\ x)$›
⟨*proof*⟩

lemma *tick-trans-adm*[*simp*]:
‹$[\![cont\ (\lambda P.\ \Omega\ P\ r);\ cont\ u;\ monofun\ v]\!] \Longrightarrow adm\ (\lambda x.\ u\ x\ {}_{DT}\leadsto_{\checkmark r}\ v\ x)$›
⟨*proof*⟩

lemma *trace-trans-adm*[*simp*]:
‹$[\![\forall x.\ ev\ x \in set\ s \longrightarrow cont\ (\lambda P.\ \Psi\ P\ x);$
$\forall r.\ ✓(r) \in set\ s \longrightarrow cont\ (\lambda P.\ \Omega\ P\ r);\ cont\ u;\ monofun\ v]\!]$
$\Longrightarrow adm\ (\lambda x.\ u\ x\ {}_{DT}\!\rightsquigarrow^{*} s\ (v\ x))$›
⟨*proof*⟩

If we only look at the traces and the divergences, non-deterministic and deterministic choices are the same. Therefore we can obtain even stronger results for the operational rules.

lemma *τ-trans-Det-is-τ-trans-Ndet*: ‹$P \square Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ R \longleftrightarrow P \sqcap Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ R$›
⟨*proof*⟩

lemma *τ-trans-Sliding-is-τ-trans-Ndet*: ‹$P \rhd Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ R \longleftrightarrow P \sqcap Q\ {}_{DT}\!\rightsquigarrow_{\tau}\ R$›
⟨*proof*⟩

end

locale *OpSemDTDuplicated* =
$OpSemDT_{\alpha}$: *OpSemDT* $\Psi_{\alpha}\ \Omega_{\alpha}$ + $OpSemDT_{\beta}$: *OpSemDT* $\Psi_{\beta}\ \Omega_{\beta}$
for Ψ_{α} :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω_{α} :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
and Ψ_{β} :: ‹[('b, 's) $process_{ptick}$, 'b] ⇒ ('b, 's) $process_{ptick}$›
and Ω_{β} :: ‹[('b, 's) $process_{ptick}$, 's] ⇒ ('b, 's) $process_{ptick}$›

sublocale *OpSemDTDuplicated* ⊆ *OpSemTransitionsAllDuplicated* - - ‹($\sqsubseteq_{DT}$)› - - ‹($\sqsubseteq_{DT}$)›
⟨*proof*⟩

6.9 ($\rightsquigarrow_{\tau}$) instantiated with ($\sqsubseteq_F$) or ($\sqsubseteq_T$)

We will only recover the rules for some operators.

6.9.1 ($\rightsquigarrow_{\tau}$) instantiated with ($\sqsubseteq_F$)

locale *OpSemF* =
fixes Ψ :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
and Ω :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
assumes *mono-Ω-F*: ‹$✓(r) \in Q^0 \Longrightarrow P \sqsubseteq_F Q \Longrightarrow \Omega\ P\ r \sqsubseteq_F \Omega\ Q\ r$›

sublocale *OpSemF* ⊆ *OpSemTransitionsHiding* - - ‹($\sqsubseteq_F$) :: ('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $process_{ptick}$ ⇒ *bool*› +
OpSemTransitionsDetRelaxed - - ‹($\sqsubseteq_F$) :: ('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $process_{ptick}$ ⇒ *bool*› +
OpSemTransitionsSlidingRelaxed - - ‹($\sqsubseteq_F$) :: ('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $process_{ptick}$ ⇒ *bool*›
⟨*proof*⟩

context *OpSemF*
begin

notation *failure-refine* (**infixl** ‹$_F\rightsquigarrow_\tau$› *50*)
notation *ev-trans* (‹- $_F\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
notation *tick-trans* (‹- $_F\rightsquigarrow$✓- -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹- $_F\rightsquigarrow^*$- -› [*50*, *3*, *51*] *50*)

For *Det* and *Sliding*, we have relaxed versions on τ transitions.

end

By duplicating the locale, we can recover a rules for *Renaming*.

locale *OpSemFDuplicated* =
  $OpSemF_\alpha$: *OpSemF* Ψ_α Ω_α + $OpSemF_\beta$: *OpSemF* Ψ_β Ω_β
  for Ψ_α :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
    and Ω_α :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
    and Ψ_β :: ‹[('b, 's) $process_{ptick}$, 'b] ⇒ ('b, 's) $process_{ptick}$›
    and Ω_β :: ‹[('b, 's) $process_{ptick}$, 's] ⇒ ('b, 's) $process_{ptick}$›

sublocale *OpSemFDuplicated* ⊆ *OpSemTransitionsDuplicated* - - ‹($\sqsubseteq_F$)› - - ‹($\sqsubseteq_F$)›
  ⟨*proof*⟩

context *OpSemFDuplicated*
begin

notation $OpSemF_\alpha$.*ev-trans* (‹- $_{\alpha F}\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
notation $OpSemF_\alpha$.*tick-trans* (‹- $_{\alpha F}\rightsquigarrow$✓- -› [*50*, *3*, *51*] *50*)
notation $OpSemF_\beta$.*ev-trans* (‹- $_{\beta F}\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
notation $OpSemF_\beta$.*tick-trans* (‹- $_{\beta F}\rightsquigarrow$✓- -› [*50*, *3*, *51*] *50*)

end

context *OpSemF*
begin

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

lemma Ω*-SKIP-is-STOP-imp-SKIP-trace-trans-iff*:
  ‹Ω (*SKIP r*) *r* = *STOP* ⟹ (*SKIP r* $_F\rightsquigarrow^*$*s P*) ⟷ *s* = [] ∧ *P* = *SKIP r* ∨ *s* = [✓(*r*)] ∧ *P* = *STOP*›
  ⟨*proof*⟩

lemmas τ*-trans-adm* = *le-F-adm*

lemma *ev-trans-adm*[*simp*]:
‹⟦*cont* (λ*P*. Ψ *P e*); *cont u*; *monofun v*⟧ ⟹ *adm* (λ*x*. *u x* $_F{\leadsto}_e$ *v x*)›
⟨*proof*⟩

lemma *tick-trans-adm*[*simp*]:
‹⟦*cont* (λ*P*. Ω *P r*); *cont u*; *monofun v*⟧ ⟹ *adm* (λ*x*. *u x* $_F{\leadsto}_{\checkmark r}$ *v x*)›
⟨*proof*⟩

lemma *trace-trans-adm*[*simp*]:
‹⟦∀ *x*. *ev x* ∈ *set s* ⟶ *cont* (λ*P*. Ψ *P x*);
∀ *r*. ✓(*r*) ∈ *set s* ⟶ *cont* (λ*P*. Ω *P r*);
cont u; *monofun v*⟧ ⟹ *adm* (λ*x*. *u x* $_F{\leadsto}^*s$ (*v x*))›
⟨*proof*⟩

end

6.9.2 ($\leadsto_\tau$) instantiated with ($\sqsubseteq_T$)

locale *OpSemTransitionsForT* =
OpSemTransitionsDet Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsHiding Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsSliding Ψ Ω ‹($\leadsto_\tau$)› +
OpSemTransitionsInterrupt Ψ Ω ‹($\leadsto_\tau$)›
for Ψ :: ‹[('*a*, '*r*) $process_{ptick}$, '*a*] ⇒ ('*a*, '*r*) $process_{ptick}$›
and Ω :: ‹[('*a*, '*r*) $process_{ptick}$, '*r*] ⇒ ('*a*, '*r*) $process_{ptick}$›
and τ*-trans* :: ‹[('*a*, '*r*) $process_{ptick}$, ('*a*, '*r*) $process_{ptick}$] ⇒ *bool*› (**infixl** ‹$\leadsto_\tau$› *50*)

locale *OpSemT* =
fixes Ψ :: ‹[('*a*, '*r*) $process_{ptick}$, '*a*] ⇒ ('*a*, '*r*) $process_{ptick}$›
and Ω :: ‹[('*a*, '*r*) $process_{ptick}$, '*r*] ⇒ ('*a*, '*r*) $process_{ptick}$›
assumes *mono-Ω-T*: ‹✓(*r*) ∈ Q^0 ⟹ *P* $\sqsubseteq_T$ *Q* ⟹ Ω *P r* $\sqsubseteq_T$ Ω *Q r*›

sublocale *OpSemT* ⊆ *OpSemTransitionsForT* - - ‹($\sqsubseteq_T$) :: ('*a*, '*r*) $process_{ptick}$ ⇒ ('*a*, '*r*) $process_{ptick}$ ⇒ *bool*›
⟨*proof*⟩

context *OpSemT*
begin

notation *trace-refine* (**infixl** ‹$_T{\leadsto}_\tau$› *50*)
notation *ev-trans* (‹- $_T{\leadsto}_-$ -› [*50*, *3*, *51*] *50*)
notation *tick-trans* (‹- $_T{\leadsto}_{\checkmark -}$ -› [*50*, *3*, *51*] *50*)
notation *trace-trans* (‹- $_T{\leadsto}^*$- -› [*50*, *3*, *51*] *50*)

end

By duplicating the locale, we can recover a rules for *Renaming*.

locale *OpSemTDuplicated* =
  $OpSemT_\alpha$: *OpSemT* Ψ_α Ω_α + $OpSemT_\beta$: *OpSemT* Ψ_β Ω_β
  for Ψ_α :: ‹[('a, 'r) $process_{ptick}$, 'a] ⇒ ('a, 'r) $process_{ptick}$›
    and Ω_α :: ‹[('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
    and Ψ_β :: ‹[('b, 's) $process_{ptick}$, 'b] ⇒ ('b, 's) $process_{ptick}$›
    and Ω_β :: ‹[('b, 's) $process_{ptick}$, 's] ⇒ ('b, 's) $process_{ptick}$›

sublocale *OpSemTDuplicated* ⊆ *OpSemTransitionsDuplicated* - - ‹($\sqsubseteq_T$)› - - ‹($\sqsubseteq_T$)›
  ⟨*proof*⟩

context *OpSemTDuplicated*
begin

notation $OpSemT_\alpha$.*ev-trans* (‹- ${}_{\alpha T}$⇝- -› [*50*, *3*, *51*] *50*)
notation $OpSemT_\alpha$.*tick-trans* (‹- ${}_{\alpha T}$⇝✓- -› [*50*, *3*, *51*] *50*)
notation $OpSemT_\beta$.*ev-trans* (‹- ${}_{\beta T}$⇝- -› [*50*, *3*, *51*] *50*)
notation $OpSemT_\beta$.*tick-trans* (‹- ${}_{\beta T}$⇝✓- -› [*50*, *3*, *51*] *50*)

end

context *OpSemT*
begin

lemmas τ-*trans-adm* = *le-T-adm*

lemma *ev-trans-adm*[*simp*]:
  ‹⟦*cont* (λ*P*. Ψ *P e*); *cont u*; *monofun v*⟧ ⟹ *adm* (λ*x*. *u x* ${}_T$⇝$_e$ *v x*)›
  ⟨*proof*⟩

lemma *tick-trans-adm*[*simp*]:
  ‹⟦*cont* (λ*P*. Ω *P r*); *cont u*; *monofun v*⟧ ⟹ *adm* (λ*x*. *u x* ${}_T$⇝✓$_r$ *v x*)›
  ⟨*proof*⟩

lemma *trace-trans-adm*[*simp*]:
  ‹⟦∀ *x*. *ev x* ∈ *set s* ⟶ *cont* (λ*P*. Ψ *P x*);
    ∀ *r*. ✓(*r*) ∈ *set s* ⟶ *cont* (λ*P*. Ω *P r*); *cont u*; *monofun v*⟧
  ⟹ *adm* (λ*x*. *u x* ${}_T$⇝**s* (*v x*))›
  ⟨*proof*⟩

If we only look at the traces, non-deterministic and deterministic choices are the same. Therefore we can obtain even stronger results for the operational rules.

lemma τ-*trans-Det-is*-τ-*trans-Ndet*: ‹*P* □ *Q* ${}_T$⇝$_\tau$ *R* ⟷ *P* ⊓ *Q* ${}_T$⇝$_\tau$ *R*›
  ⟨*proof*⟩

lemma τ-*trans-Sliding-is*-τ-*trans-Ndet*: ‹*P* ▷ *Q* ${}_T$⇝$_\tau$ *R* ⟷ *P* ⊓ *Q* ${}_T$⇝$_\tau$ *R*›

⟨*proof*⟩

end

Chapter 7

Recovered Laws pretty printed

7.1 General Case

This is the general case, working for $(\sqsubseteq_{FD})$ and $(\sqsubseteq_{DT})$.

context *OpSemTransitionsAll* **begin**

Absorbency rules

$$\frac{?P \leadsto_{?e} ?P' \qquad ?P' \leadsto_{\tau} ?P''}{?P \leadsto_{?e} ?P''} \qquad \frac{?P \leadsto_{\tau} ?P' \qquad ?P' \leadsto_{?e} ?P''}{?P \leadsto_{?e} ?P''}$$

$$\frac{?P \leadsto_{\checkmark ?r} ?P' \qquad ?P' \leadsto_{\tau} ?P''}{?P \leadsto_{\checkmark ?r} ?P''} \qquad \frac{?P \leadsto_{\tau} ?P' \qquad ?P' \leadsto_{\checkmark ?r} ?P''}{?P \leadsto_{\checkmark ?r} ?P''}$$

SKIP rule

$$\overline{SKIP\ ?r \leadsto_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

$e \to P$ rules

$$\overline{?e \to ?P \leadsto_{?e} ?P}$$

$$\frac{?e \in ?A}{\square a{\in}?A \to ?P\ a \leadsto_{?e} ?P\ ?e} \qquad \frac{?e \in ?A}{\sqcap a{\in}?A \to ?P\ a \leadsto_{?e} ?P\ ?e}$$

$(\sqcap)$ **rules**

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

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

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

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

$(\Box)$ **rules**

$$\frac{?P \rightsquigarrow_{\tau}\ ?P'}{?P \Box\ ?Q \rightsquigarrow_{\tau}\ ?P' \Box\ ?Q} \qquad \frac{?Q \rightsquigarrow_{\tau}\ ?Q'}{?P \Box\ ?Q \rightsquigarrow_{\tau}\ ?P \Box\ ?Q'}$$

$$\frac{?P \rightsquigarrow_{?e}\ ?P'}{?P \Box\ ?Q \rightsquigarrow_{?e}\ ?P'} \qquad \frac{?Q \rightsquigarrow_{?e}\ ?Q'}{?P \Box\ ?Q \rightsquigarrow_{?e}\ ?Q'}$$

$$\frac{?P \rightsquigarrow_{\checkmark ?r}\ ?P'}{?P \Box\ ?Q \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r} \qquad \frac{?Q \rightsquigarrow_{\checkmark ?r}\ ?Q'}{?P \Box\ ?Q \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

$(;)$ **rules**

$$\frac{?P \rightsquigarrow_{\tau}\ ?P'}{?P\ ;\ ?Q \rightsquigarrow_{\tau}\ ?P'\ ;\ ?Q}$$

$$\frac{?P \rightsquigarrow_{?e}\ ?P'}{?P\ ;\ ?Q \rightsquigarrow_{?e}\ ?P'\ ;\ ?Q} \qquad \frac{?P \rightsquigarrow_{\checkmark ?r}\ ?P' \qquad ?Q \rightsquigarrow_{\tau}\ ?Q'}{?P\ ;\ ?Q \rightsquigarrow_{\tau}\ ?Q'}$$

$(\backslash)$ **rules**

$$\frac{?P \rightsquigarrow_{\tau}\ ?P'}{?P \setminus\ ?A \rightsquigarrow_{\tau}\ ?P' \setminus\ ?A} \qquad \frac{?P \rightsquigarrow_{\checkmark ?r}\ ?P'}{?P \setminus\ ?B \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

$$\frac{?e \notin\ ?A \qquad ?P \rightsquigarrow_{?e}\ ?P'}{?P \setminus\ ?A \rightsquigarrow_{?e}\ ?P' \setminus\ ?A} \qquad \frac{?e \in\ ?A \qquad ?P \rightsquigarrow_{?e}\ ?P'}{?P \setminus\ ?A \rightsquigarrow_{\tau}\ ?P' \setminus\ ?A}$$

Sync **rules**

$$\frac{?P \rightsquigarrow_{\tau} ?P'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{\tau} ?P' \llbracket ?S \rrbracket\ ?Q} \qquad \frac{?Q \rightsquigarrow_{\tau} ?Q'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{\tau} ?P \llbracket ?S \rrbracket\ ?Q'}$$

$$\frac{?e \notin ?S \qquad ?P \rightsquigarrow_{?e} ?P'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{?e} ?P' \llbracket ?S \rrbracket\ ?Q} \qquad \frac{?e \notin ?S \qquad ?Q \rightsquigarrow_{?e} ?Q'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{?e} ?P \llbracket ?S \rrbracket\ ?Q'}$$

$$\frac{?e \in ?S \qquad ?P \rightsquigarrow_{?e} ?P' \qquad ?Q \rightsquigarrow_{?e} ?Q'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{?e} ?P' \llbracket ?S \rrbracket\ ?Q'}$$

$$\frac{?P \rightsquigarrow_{\checkmark ?r} ?P'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{\tau} SKIP\ ?r \llbracket ?S \rrbracket\ ?Q}$$

$$\frac{?Q \rightsquigarrow_{\checkmark ?r} ?Q'}{?P \llbracket ?S \rrbracket\ ?Q \rightsquigarrow_{\tau} ?P \llbracket ?S \rrbracket\ SKIP\ ?r}$$

$$\frac{}{SKIP\ ?r \llbracket ?S \rrbracket\ SKIP\ ?r \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

($\triangleright$) **rules**

$$\frac{}{?P \triangleright ?Q \rightsquigarrow_{\tau} ?Q} \qquad \frac{?P \rightsquigarrow_{\tau} ?P'}{?P \triangleright ?Q \rightsquigarrow_{\tau} ?P' \triangleright ?Q}$$

$$\frac{?P \rightsquigarrow_{?e} ?P'}{?P \triangleright ?Q \rightsquigarrow_{?e} ?P'} \qquad \frac{?P \rightsquigarrow_{\checkmark ?r} ?P'}{?P \triangleright ?Q \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

($\triangle$) **rules**

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

$$\frac{?P \rightsquigarrow_{?e} ?P'}{?P \triangle ?Q \rightsquigarrow_{?e} ?P' \triangle ?Q} \qquad \frac{?Q \rightsquigarrow_{?e} ?Q'}{?P \triangle ?Q \rightsquigarrow_{?e} ?Q'}$$

$$\frac{?P \rightsquigarrow_{\checkmark ?r} ?P'}{?P \triangle ?Q \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r} \qquad \frac{?Q \rightsquigarrow_{\checkmark ?r} ?Q'}{?P \triangle ?Q \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

Throw **rules**

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

$$\frac{?P \rightsquigarrow_{\checkmark ?r} ?P'}{?P\ \Theta\ a{\in}?A.\ ?Q\ a \rightsquigarrow_{\checkmark ?r} \Omega\ (SKIP\ ?r)\ ?r}$$

$$\frac{?e \notin ?A \qquad ?P \leadsto_{?e} ?P'}{?P \ \Theta \ a \in ?A. \ ?Q \ a \leadsto_{?e} ?P' \ \Theta \ a \in ?A. \ ?Q \ a}$$

$$\frac{?e \in ?A \qquad ?P \leadsto_{?e} ?P'}{?P \ \Theta \ a \in ?A. \ ?Q \ a \leadsto_{?e} ?Q \ ?e}$$

end

context *OpSemTransitionsAllDuplicated* **begin**

Renaming **rules**

$$\frac{?P \ _{\alpha}\!\leadsto_{\tau} \ ?P'}{Renaming \ ?P \ ?f \ ?g \ _{\beta}\!\leadsto_{\tau} \ Renaming \ ?P' \ ?f \ ?g}$$

$$\frac{?P \ _{\alpha}\!\leadsto_{\checkmark} {?r} \ ?P'}{Renaming \ ?P \ ?f \ ?g \ _{\beta}\!\leadsto_{\checkmark} {?g} \ {?r} \ \Omega_{\beta} \ (SKIP \ (?g \ ?r)) \ (?g \ ?r)}$$

$$\frac{?f \ ?a = ?b \qquad ?P \ _{\alpha}\!\leadsto_{?a} \ ?P'}{Renaming \ ?P \ ?f \ ?g \ _{\beta}\!\leadsto_{?b} \ Renaming \ ?P' \ ?f \ ?g}$$

end

7.2 Special Cases

7.2.1 With the Refinement ($\sqsubseteq_{DT}$)

context *OpSemDT* **begin**

($\square$) **rules**

$$\frac{}{P \ \square \ Q \ _{DT}\!\leadsto_{\tau} \ P} \qquad \frac{}{P \ \square \ Q \ _{DT}\!\leadsto_{\tau} \ Q}$$

($\rhd$) **rules**

$$\frac{}{P \rhd Q \ _{DT}\!\leadsto_{\tau} \ P} \qquad \frac{}{P \rhd Q \ _{DT}\!\leadsto_{\tau} \ Q}$$

end

7.2.2 With the Refinement ($\sqsubseteq_{F}$)

context *OpSemF* **begin**

Absorbency rules

$$\frac{?P\ {}_F{\leadsto}_{?e}\ ?P' \qquad ?P'\ {}_F{\leadsto}_{\tau}\ ?P''}{?P\ {}_F{\leadsto}_{?e}\ ?P''} \qquad \frac{?P\ {}_F{\leadsto}_{\tau}\ ?P' \qquad ?P'\ {}_F{\leadsto}_{?e}\ ?P''}{?P\ {}_F{\leadsto}_{?e}\ ?P''}$$

$$\frac{?P\ {}_F{\leadsto}_{\checkmark ?r}\ ?P' \qquad ?P'\ {}_F{\leadsto}_{\tau}\ ?P''}{?P\ {}_F{\leadsto}_{\checkmark ?r}\ ?P''}$$

$$\frac{?P\ {}_F{\leadsto}_{\tau}\ ?P' \qquad ?P'\ {}_F{\leadsto}_{\checkmark ?r}\ ?P''}{?P\ {}_F{\leadsto}_{\checkmark ?r}\ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP\ ?r\ {}_F{\leadsto}_{\checkmark ?r}\ \Omega\ (SKIP\ ?r)\ ?r}$$

$e \to P$ **rules**

$$\frac{}{?e \to\ ?Pa\ {}_F{\leadsto}_{?e}\ ?Pa}$$

$$\frac{?e \in\ ?A}{\Box a{\in}?A \to\ ?P\ a\ {}_F{\leadsto}_{?e}\ ?P\ ?e} \qquad \frac{?e \in\ ?A}{\sqcap a{\in}?A \to\ ?P\ a\ {}_F{\leadsto}_{?e}\ ?P\ ?e}$$

$(\sqcap)$ **rules**

$$\frac{}{?Pa \sqcap\ ?Q\ {}_F{\leadsto}_{\tau}\ ?Pa} \qquad \frac{}{?Pa \sqcap\ ?Q\ {}_F{\leadsto}_{\tau}\ ?Q}$$

$$\frac{?e \in\ ?A}{\sqcap a{\in}?A.\ ?P\ a\ {}_F{\leadsto}_{\tau}\ ?P\ ?e}$$

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

$$\frac{cont\ ?f \qquad ?P = (\mu\ x.\ ?f\,x)}{?P\ {}_F{\leadsto}_{\tau}\ ?f\ ?P}$$

($\Box$) **rules**

$$\frac{?P = \bot \lor ?P' \neq \bot \lor ?Q = \bot \qquad ?P \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P'}{?P \;\Box\; ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P' \;\Box\; ?Q}$$

$$\frac{?Q = \bot \lor ?Q' \neq \bot \lor ?Q = \bot \qquad ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?Q'}{?P \;\Box\; ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P \;\Box\; ?Q'}$$

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

$$\frac{?P \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?P'}{?P \;\Box\; ?Q \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; \Omega\; (SKIP\ ?r)\ ?r} \qquad \frac{?Q \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?Q'}{?P \;\Box\; ?Q \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; \Omega\; (SKIP\ ?r)\ ?r}$$

(;) **rules**

$$\frac{?P \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?P' \qquad ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?Q'}{?P \;\mathbf{;}\; ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?Q'}$$

($\setminus$) **rules**

$$\frac{?P \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P'}{?P \setminus ?A \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P' \setminus ?A} \qquad \frac{?P \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?P'}{?P \setminus ?B \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; \Omega\; (SKIP\ ?r)\ ?r}$$

$$\frac{?e \notin ?A \qquad ?P \;{}_{F}{\rightsquigarrow}_{?e}\; ?P'}{?P \setminus ?A \;{}_{F}{\rightsquigarrow}_{?e}\; ?P' \setminus ?A} \qquad \frac{?e \in ?A \qquad ?P \;{}_{F}{\rightsquigarrow}_{?e}\; ?P'}{?P \setminus ?A \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P' \setminus ?A}$$

Sync **rules**

$$\frac{?P \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?P'}{?P \;[\![?S]\!]\; ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; SKIP\ ?r \;[\![?S]\!]\; ?Q}$$

$$\frac{?Q \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?Q'}{?P \;[\![?S]\!]\; ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P \;[\![?S]\!]\; SKIP\ ?r}$$

$$\frac{}{SKIP\ ?r \;[\![?S]\!]\; SKIP\ ?r \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; \Omega\; (SKIP\ ?r)\ ?r}$$

($\rhd$) **rules**

$$\frac{}{?P \rhd ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?Q} \qquad \frac{?P = \bot \lor ?P' \neq \bot \lor ?Q = \bot \qquad ?P \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P'}{?P \rhd ?Q \;{}_{F}{\rightsquigarrow}_{\tau}\; ?P' \rhd ?Q}$$

$$\frac{?P \;{}_{F}{\rightsquigarrow}_{?e}\; ?P'}{?P \rhd ?Q \;{}_{F}{\rightsquigarrow}_{?e}\; ?P'} \qquad \frac{?P \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; ?P'}{?P \rhd ?Q \;{}_{F}{\rightsquigarrow}_{\checkmark ?r}\; \Omega\; (SKIP\ ?r)\ ?r}$$

($\triangle$) rules

$$\frac{?P\ {}_{F}{\leadsto}_{\checkmark\,?r}\ ?P'}{?P \triangle ?Q\ {}_{F}{\leadsto}_{\checkmark\,?r}\ \Omega\ (SKIP\ ?r)\ ?r} \qquad \frac{?Q\ {}_{F}{\leadsto}_{\checkmark\,?r}\ ?Q'}{?P \triangle ?Q\ {}_{F}{\leadsto}_{\checkmark\,?r}\ \Omega\ (SKIP\ ?r)\ ?r}$$

Throw **rules**

$$\frac{?e \in ?A \qquad ?P\ {}_{F}{\leadsto}_{?e}\ ?P'}{?P\ \Theta\ a{\in}?A.\ ?Q\ a\ {}_{F}{\leadsto}_{?e}\ ?Q\ ?e}$$

$$\frac{?P\ {}_{F}{\leadsto}_{\checkmark\,?r}\ ?P'}{?P\ \Theta\ a{\in}?A.\ ?Q\ a\ {}_{F}{\leadsto}_{\checkmark\,?r}\ \Omega\ (SKIP\ ?r)\ ?r}$$

end

context *OpSemFDuplicated* **begin**

Renaming **rules**

$$\frac{?P\ {}_{\alpha F}{\leadsto}_{\checkmark\,?r}\ ?P'}{Renaming\ ?P\ ?f\ ?g\ {}_{\beta F}{\leadsto}_{\checkmark\,?g\ ?r}\ \Omega_{\beta}\ (SKIP\ (?g\ ?r))\ (?g\ ?r)}$$

end

7.2.3 With the Refinement ($\sqsubseteq_T$)

context *OpSemT* **begin**

Absorbency rules

$$\frac{?P\ {}_{T}{\leadsto}_{?e}\ ?P' \qquad ?P'\ {}_{T}{\leadsto}_{\tau}\ ?P''}{?P\ {}_{T}{\leadsto}_{?e}\ ?P''} \qquad \frac{?P\ {}_{T}{\leadsto}_{\tau}\ ?P' \qquad ?P'\ {}_{T}{\leadsto}_{?e}\ ?P''}{?P\ {}_{T}{\leadsto}_{?e}\ ?P''}$$

$$\frac{?P\ {}_{T}{\leadsto}_{\checkmark\,?r}\ ?P' \qquad ?P'\ {}_{T}{\leadsto}_{\tau}\ ?P''}{?P\ {}_{T}{\leadsto}_{\checkmark\,?r}\ ?P''}$$

$$\frac{?P\ {}_{T}{\leadsto}_{\tau}\ ?P' \qquad ?P'\ {}_{T}{\leadsto}_{\checkmark\,?r}\ ?P''}{?P\ {}_{T}{\leadsto}_{\checkmark\,?r}\ ?P''}$$

SKIP **rule**

$$\frac{}{SKIP\ ?r\ {}_{T}{\leadsto}_{\checkmark\,?r}\ \Omega\ (SKIP\ ?r)\ ?r}$$

$e \to P$ **rules**

$$\frac{}{?e \to ?Pa \ {}_{T}{\leadsto}_{?e} \ ?Pa}$$

$$\frac{?e \in ?A}{\Box a {\in} ?A \to \ ?P \ a \ {}_{T}{\leadsto}_{?e} \ ?P \ ?e} \qquad \frac{?e \in ?A}{\sqcap a {\in} ?A \to \ ?P \ a \ {}_{T}{\leadsto}_{?e} \ ?P \ ?e}$$

$(\sqcap)$ **rules**

$$\frac{}{?Pa \sqcap \ ?Q \ {}_{T}{\leadsto}_{\tau} \ ?Pa} \qquad \frac{}{?Pa \sqcap \ ?Q \ {}_{T}{\leadsto}_{\tau} \ ?Q}$$

$$\frac{?e \in ?A}{\sqcap a {\in} ?A. \ ?P \ a \ {}_{T}{\leadsto}_{\tau} \ ?P \ ?e}$$

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

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

$(\Box)$ **rules**

$$\frac{?P \ {}_{T}{\leadsto}_{\tau} \ ?P'}{?P \ \Box \ ?Q \ {}_{T}{\leadsto}_{\tau} \ ?P' \ \Box \ ?Q} \qquad \frac{?Q \ {}_{T}{\leadsto}_{\tau} \ ?Q'}{?P \ \Box \ ?Q \ {}_{T}{\leadsto}_{\tau} \ ?P \ \Box \ ?Q'}$$

$$\frac{?P \ {}_{T}{\leadsto}_{?e} \ ?P'}{?P \ \Box \ ?Q \ {}_{T}{\leadsto}_{?e} \ ?P'} \qquad \frac{?Q \ {}_{T}{\leadsto}_{?e} \ ?Q'}{?P \ \Box \ ?Q \ {}_{T}{\leadsto}_{?e} \ ?Q'}$$

$$\frac{?P \ {}_{T}{\leadsto}_{\checkmark ?r} \ ?P'}{?P \ \Box \ ?Q \ {}_{T}{\leadsto}_{\checkmark ?r} \ \Omega \ (SKIP \ ?r) \ ?r} \qquad \frac{?Q \ {}_{T}{\leadsto}_{\checkmark ?r} \ ?Q'}{?P \ \Box \ ?Q \ {}_{T}{\leadsto}_{\checkmark ?r} \ \Omega \ (SKIP \ ?r) \ ?r}$$

(**;**) **rules**

$$\frac{?P \ {}_{T}{\leadsto}_{\checkmark ?r} \ ?P' \qquad ?Q \ {}_{T}{\leadsto}_{\tau} \ ?Q'}{?P \ \mathbf{;} \ ?Q \ {}_{T}{\leadsto}_{\tau} \ ?Q'}$$

$(\backslash)$ **rules**

$$\frac{?P \ {}_T{\leadsto}_\tau \ ?P'}{?P \setminus ?A \ {}_T{\leadsto}_\tau \ ?P' \setminus ?A} \qquad \frac{?P \ {}_T{\leadsto}_{\checkmark\,?r} \ ?P'}{?P \setminus ?B \ {}_T{\leadsto}_{\checkmark\,?r} \ \Omega\ (SKIP\ ?r)\ ?r}$$

$$\frac{?e \notin ?A \qquad ?P \ {}_T{\leadsto}_{?e} \ ?P'}{?P \setminus ?A \ {}_T{\leadsto}_{?e} \ ?P' \setminus ?A} \qquad \frac{?e \in ?A \qquad ?P \ {}_T{\leadsto}_{?e} \ ?P'}{?P \setminus ?A \ {}_T{\leadsto}_\tau \ ?P' \setminus ?A}$$

Sync **rules**

$$\frac{?P \ {}_T{\leadsto}_{\checkmark\,?r} \ ?P'}{?P \ [\![?S]\!] \ ?Q \ {}_T{\leadsto}_\tau \ SKIP\ ?r \ [\![?S]\!] \ ?Q}$$

$$\frac{?Q \ {}_T{\leadsto}_{\checkmark\,?r} \ ?Q'}{?P \ [\![?S]\!] \ ?Q \ {}_T{\leadsto}_\tau \ ?P \ [\![?S]\!] \ SKIP\ ?r}$$

$$\frac{}{SKIP\ ?r \ [\![?S]\!] \ SKIP\ ?r \ {}_T{\leadsto}_{\checkmark\,?r} \ \Omega\ (SKIP\ ?r)\ ?r}$$

$(\triangleright)$ **rules**

$$\frac{}{?P \triangleright ?Q \ {}_T{\leadsto}_\tau \ ?Q} \qquad \frac{?P \ {}_T{\leadsto}_\tau \ ?P'}{?P \triangleright ?Q \ {}_T{\leadsto}_\tau \ ?P' \triangleright ?Q}$$

$$\frac{?P \ {}_T{\leadsto}_{?e} \ ?P'}{?P \triangleright ?Q \ {}_T{\leadsto}_{?e} \ ?P'} \qquad \frac{?P \ {}_T{\leadsto}_{\checkmark\,?r} \ ?P'}{?P \triangleright ?Q \ {}_T{\leadsto}_{\checkmark\,?r} \ \Omega\ (SKIP\ ?r)\ ?r}$$

$(\triangle)$ **rules**

$$\frac{?P \ {}_T{\leadsto}_\tau \ ?P'}{?P \triangle ?Q \ {}_T{\leadsto}_\tau \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_T{\leadsto}_\tau \ ?Q'}{?P \triangle ?Q \ {}_T{\leadsto}_\tau \ ?P \triangle ?Q'}$$

$$\frac{?P \ {}_T{\leadsto}_{?e} \ ?P'}{?P \triangle ?Q \ {}_T{\leadsto}_{?e} \ ?P' \triangle ?Q} \qquad \frac{?Q \ {}_T{\leadsto}_{?e} \ ?Q'}{?P \triangle ?Q \ {}_T{\leadsto}_{?e} \ ?Q'}$$

$$\frac{?P \ {}_T{\leadsto}_{\checkmark\,?r} \ ?P'}{?P \triangle ?Q \ {}_T{\leadsto}_{\checkmark\,?r} \ \Omega\ (SKIP\ ?r)\ ?r} \qquad \frac{?Q \ {}_T{\leadsto}_{\checkmark\,?r} \ ?Q'}{?P \triangle ?Q \ {}_T{\leadsto}_{\checkmark\,?r} \ \Omega\ (SKIP\ ?r)\ ?r}$$

Throw **rules**

$$\frac{?e \in ?A \qquad ?P \ {}_T{\leadsto}_{?e} \ ?P'}{?P \ \Theta \ a{\in}?A.\ ?Q\ a \ {}_T{\leadsto}_{?e} \ ?Q\ ?e}$$

$$\frac{?P \ {}_T{\leadsto}_{\checkmark\,?r} \ ?P'}{?P \ \Theta \ a{\in}?A.\ ?Q\ a \ {}_T{\leadsto}_{\checkmark\,?r} \ \Omega\ (SKIP\ ?r)\ ?r}$$

Because we only look at the traces, we actully have the following results.

($\square$) **rules**

$$\frac{}{P \,\square\, Q \;{}_T{\leadsto}_\tau\; P} \qquad \frac{}{P \,\square\, Q \;{}_T{\leadsto}_\tau\; Q}$$

($\rhd$) **rules**

$$\frac{}{P \rhd Q \;{}_T{\leadsto}_\tau\; P} \qquad \frac{}{P \rhd Q \;{}_T{\leadsto}_\tau\; Q}$$

end

context *OpSemTDuplicated* **begin**

Renaming **rules**

$$\frac{?P \;{}_{\alpha T}{\leadsto}_{\checkmark\, ?r}\; ?P'}{\mathit{Renaming}\ ?P\ ?f\ ?g \;{}_{\beta T}{\leadsto}_{\checkmark\, ?g\ ?r}\; \Omega_\beta\ (\mathit{SKIP}\ (?g\ ?r))\ (?g\ ?r)}$$

end

Chapter 8

Comparison with He and Hoare

lemma (**in** *After*) *initial-ev-imp-eq-prefix-After-Sliding* :
 ‹$P = (e \to (P\ after\ e)) \rhd P$› **if** ‹$ev\ e \in P^0$›
⟨*proof*⟩

context *OpSemTransitions*
begin

abbreviation τ*-eq* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infix** ‹$=_\tau$› *50*)
 where ‹$P =_\tau Q \equiv P \leadsto_\tau Q \wedge Q \leadsto_\tau P$›

lemma τ*-eqI* : ‹$P \leadsto_\tau Q \Longrightarrow Q \leadsto_\tau P \Longrightarrow P =_\tau Q$›
 and τ*-eqD1* : ‹$P =_\tau Q \Longrightarrow P \leadsto_\tau Q$›
 and τ*-eqD2* : ‹$P =_\tau Q \Longrightarrow Q \leadsto_\tau P$›
 ⟨*proof*⟩

lemma τ*-trans-iff-*τ*-eq-Ndet*:
 ‹$\forall P\ Q\ P'\ Q'.\ P \leadsto_\tau P' \longrightarrow Q \leadsto_\tau Q' \longrightarrow P \sqcap Q \leadsto_\tau P' \sqcap Q' \Longrightarrow P \leadsto_\tau Q \longleftrightarrow P =_\tau P \sqcap Q$›
 ⟨*proof*⟩

lemma *eq-imp-*τ*-eq*: ‹$P = Q \Longrightarrow P =_\tau Q$› ⟨*proof*⟩

definition *ev-trans*$_{HOARE}$ :: ‹('a, 'r) *process*$_{ptick}$ ⇒ 'a ⇒ ('a, 'r) *process*$_{ptick}$ ⇒ *bool*› (‹- $_{HOARE}$⇝- -› [50, 3, 51] 50)
where ‹P $_{HOARE}$⇝$_e$ $Q \equiv P \leadsto_\tau (e \to Q) \square P$›

lemma *ev-trans*$_{HOARE}$*-imp-in-initials*:
‹P $_{HOARE}$⇝$_e$ $Q \Longrightarrow ev\ e \in P^0$›
⟨*proof*⟩

lemma *ev-trans*$_{HOARE}$*-imp-ev-trans*: ‹$P \leadsto_e Q$› **if** ‹P $_{HOARE}$⇝$_e$ Q›
⟨*proof*⟩

Two assumptions on τ transitions are necessary in the following proof, but are automatic when we instantiate ($\leadsto_\tau$) with ($\sqsubseteq_{FD}$), ($\sqsubseteq_{DT}$), ($\sqsubseteq_F$) or ($\sqsubseteq_T$).

lemma *hyps-on-τ-trans-imp-ev-trans-imp-ev-trans*$_{HOARE}$: ‹P $_{HOARE}$⇝$_e$ Q›
if *non-BOT-τ-trans-DetL* : ‹$\forall P\ P'\ Q.\ P = \bot \lor P' \neq \bot \longrightarrow P \leadsto_\tau P' \longrightarrow P \square Q \leadsto_\tau P' \square Q$›
and *τ-trans-prefix* : ‹$\forall P\ P'\ e.\ P \leadsto_\tau P' \longrightarrow (e \to P) \leadsto_\tau (e \to P')$›
and ‹$P \leadsto_e Q$›
⟨*proof*⟩
lemma *hyps-on-τ-trans-imp-ev-trans*$_{HOARE}$*-iff-ev-trans*:
‹$\forall P\ P'\ Q.\ P = \bot \lor P' \neq \bot \longrightarrow P \leadsto_\tau P' \longrightarrow P \square Q \leadsto_\tau P' \square Q \Longrightarrow$
$\forall P\ P'\ e.\ P \leadsto_\tau P' \longrightarrow (e \to P) \leadsto_\tau (e \to P') \Longrightarrow P$ $_{HOARE}$⇝$_e$ $Q \longleftrightarrow P \leadsto_e Q$›
⟨*proof*⟩
lemma *BOT-ev-trans*$_{HOARE}$*-anything*: ‹$\bot$ $_{HOARE}$⇝$_e$ P›
⟨*proof*⟩
lemma *hyps-on-τ-trans-imp-ev-trans*$_{HOARE}$*-def-bis*: ‹P $_{HOARE}$⇝$_e$ $Q \longleftrightarrow P =_\tau (e \to Q) \rhd P$›
if *non-BOT-τ-trans-SlidingL* : ‹$\forall P\ P'\ Q.\ P = \bot \lor P' \neq \bot \longrightarrow P \leadsto_\tau P' \longrightarrow P \rhd Q \leadsto_\tau P' \rhd Q$›
and *τ-trans-prefix* : ‹$\forall P\ P'\ e.\ P \leadsto_\tau P' \longrightarrow (e \to P) \leadsto_\tau (e \to P')$›
⟨*proof*⟩

end

context *OpSemFD*
begin

notation *ev-trans*$_{HOARE}$ (‹- $_{FD-HOARE}$⇝- -› [50, 3, 51] 50)
notation *τ-eq* (**infix** ‹$_{FD}=_\tau$› 50)

theorem *ev-trans*$_{HOARE}$*-iff-ev-trans* : ‹P $_{FD-HOARE}$⇝$_e$ $Q \longleftrightarrow P$ $_{FD}$⇝$_e$ Q›
⟨*proof*⟩

theorem *ev-trans*$_{HOARE}$*-def-bis*: ‹P $_{FD-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{FD}{=}_{\tau}$ $(e \rightarrow Q)$ $\rhd$ P›
⟨*proof*⟩

end

context *OpSemDT*
begin

notation *ev-trans*$_{HOARE}$ (‹- $_{DT-HOARE}{\leadsto}$- -› [*50*, *3*, *51*] *50*)
notation *τ-eq* (**infix** ‹$_{DT}{=}_{\tau}$› *50*)

theorem *ev-trans*$_{HOARE}$*-iff-ev-trans* : ‹P $_{DT-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{DT}{\leadsto}_e$ Q›
⟨*proof*⟩

theorem *ev-trans*$_{HOARE}$*-def-bis*: ‹P $_{DT-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{DT}{=}_{\tau}$ $(e \rightarrow Q)$ $\rhd$ P›
⟨*proof*⟩

end

context *OpSemF*
begin

notation *ev-trans*$_{HOARE}$ (‹- $_{F-HOARE}{\leadsto}$- -› [*50*, *3*, *51*] *50*)
notation *τ-eq* (**infix** ‹$_{F}{=}_{\tau}$› *50*)

theorem *ev-trans*$_{HOARE}$*-iff-ev-trans* : ‹P $_{F-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{F}{\leadsto}_e$ Q›
⟨*proof*⟩

theorem *ev-trans*$_{HOARE}$*-def-bis*: ‹P $_{F-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{F}{=}_{\tau}$ $(e \rightarrow Q)$ $\rhd$ P›
⟨*proof*⟩

end

context *OpSemT*
begin

notation *ev-trans*$_{HOARE}$ (‹- $_{T-HOARE}{\leadsto}$- -› [*50*, *3*, *51*] *50*)
notation *τ-eq* (**infix** ‹$_{T}{=}_{\tau}$› *50*)

theorem *ev-trans*$_{HOARE}$*-iff-ev-trans* : ‹P $_{T-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{T}{\leadsto}_e$ Q›
⟨*proof*⟩

theorem *ev-trans*$_{HOARE}$*-def-bis*: ‹P $_{T-HOARE}{\leadsto}_e$ $Q \longleftrightarrow P$ $_{T}{=}_{\tau}$ $(e \rightarrow Q)$ $\rhd$

P›
⟨*proof*⟩

end

8.1 Deadlock Results

lemma *initial-ev-imp-in-events-of*: ‹$ev\ a \in P^0 \Longrightarrow a \in \alpha(P)$›
⟨*proof*⟩

lemma *initial-tick-imp-in-ticks-of*: ‹$✓(r) \in P^0 \Longrightarrow r \in ✓s(P)$›
⟨*proof*⟩

lemma ‹$UNIV \in \mathcal{R}\ P \longleftrightarrow P \sqsubseteq_F STOP$›
⟨*proof*⟩

lemma *no-events-of-if-at-most-initial-tick*: ‹$P^0 \subseteq range\ tick \Longrightarrow \alpha(P) = \{\}$›
⟨*proof*⟩

lemma *deadlock-free-initial-evE*:
‹$deadlock\text{-}free\ P \Longrightarrow (\bigwedge a.\ ev\ a \in P^0 \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

context *AfterExt*
begin

As we said earlier, $After_{trace}$ allows us to obtain some very powerful results about *deadlock-free* and $deadlock\text{-}free_{SKIPS}$.

8.1.1 Preliminaries and induction Rules

context fixes P :: ‹$('a,\ 'r)\ process_{ptick}$› **begin**

lemma *initials-*$After_{trace}$*-subset-events-of*:
‹$(P\ after_{\mathcal{T}}\ t)^0 \subseteq ev$ ‘ $\alpha(P)$› **if** ‹*non-terminating P*› ‹$t \in \mathcal{T}\ P$›
⟨*proof*⟩

end

With the next result, the general idea appears: instead of doing an induction only on the process *P* we are interested in, we include a quantification over all the processes than can be reached from *P* after some trace of *P*.

theorem $After_{trace}$*-fix-ind* [*consumes 2*, *case-names cont step*]:

fixes *ref* :: ‹[('a, 'r) $process_{ptick}$, ('a, 'r) $process_{ptick}$] ⇒ *bool*› (**infix** ‹$\sqsubseteq_{ref}$› *60*)
assumes *adm-ref* : ‹$\bigwedge$*u v. cont* (*u* :: ('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $process_{ptick}$) ⟹ *monofun v*
⟹ *adm* (λ*x. u x* $\sqsubseteq_{ref}$ *v x*)›
and *BOT-le-ref* : ‹$\bigwedge$*Q*. ⊥ $\sqsubseteq_{ref}$ *Q*›
and *cont-f* : ‹*cont f*›
and *hyp* : ‹$\bigwedge$*s x*. ∀ *Q* ∈ {*Q*. ∃ *s* ∈ $\mathcal{T}$ *P*. *g s Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *s*}. *x* $\sqsubseteq_{ref}$ *Q* ⟹
s ∈ $\mathcal{T}$ *P* ⟹ *g s* (*P after*$_{\mathcal{T}}$ *s*) ⟹ *f x* $\sqsubseteq_{ref}$ *P after*$_{\mathcal{T}}$ *s*›
shows ‹∀ *Q* ∈ {*Q*. ∃ *s* ∈ $\mathcal{T}$ *P*. *g s Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *s*}. (μ *X*. *f X*) $\sqsubseteq_{ref}$ *Q*›
⟨*proof*⟩

lemma *After*$_{trace}$*-fix-ind-F* [*consumes 1*, *case-names cont step*]:
‹⟦*Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}; *cont f*;
$\bigwedge$*t x*. ⟦∀ *Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}. *x* $\sqsubseteq_F$ *Q*;
t ∈ $\mathcal{T}$ *P*; *g t* (*P after*$_{\mathcal{T}}$ *t*)⟧ ⟹ *f x* $\sqsubseteq_F$ *P after*$_{\mathcal{T}}$ *t*⟧ ⟹
(μ *X*. *f X*) $\sqsubseteq_F$ *Q*›
and *After*$_{trace}$*-fix-ind-D* [*consumes 1*, *case-names cont step*]:
‹⟦*Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}; *cont f*;
$\bigwedge$*t x*. ⟦∀ *Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}. *x* $\sqsubseteq_D$ *Q*;
t ∈ $\mathcal{T}$ *P*; *g t* (*P after*$_{\mathcal{T}}$ *t*)⟧ ⟹ *f x* $\sqsubseteq_D$ *P after*$_{\mathcal{T}}$ *t*⟧ ⟹
(μ *X*. *f X*) $\sqsubseteq_D$ *Q*›
and *After*$_{trace}$*-fix-ind-T* [*consumes 1*, *case-names cont step*]:
‹⟦*Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}; *cont f*;
$\bigwedge$*t x*. ⟦∀ *Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}. *x* $\sqsubseteq_T$ *Q*;
t ∈ $\mathcal{T}$ *P*; *g t* (*P after*$_{\mathcal{T}}$ *t*)⟧ ⟹ *f x* $\sqsubseteq_T$ *P after*$_{\mathcal{T}}$ *t*⟧ ⟹
(μ *X*. *f X*) $\sqsubseteq_T$ *Q*›
and *After*$_{trace}$*-fix-ind-FD* [*consumes 1*, *case-names cont step*]:
‹⟦*Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}; *cont f*;
$\bigwedge$*t x*. ⟦∀ *Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}. *x* $\sqsubseteq_{FD}$ *Q*;
t ∈ $\mathcal{T}$ *P*; *g t* (*P after*$_{\mathcal{T}}$ *t*)⟧ ⟹ *f x* $\sqsubseteq_{FD}$ *P after*$_{\mathcal{T}}$ *t*⟧ ⟹
(μ *X*. *f X*) $\sqsubseteq_{FD}$ *Q*›
and *After*$_{trace}$*-fix-ind-DT* [*consumes 1*, *case-names cont step*]:
‹⟦*Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}; *cont f*;
$\bigwedge$*t x*. ⟦∀ *Q* ∈ {*Q*. ∃ *t* ∈ $\mathcal{T}$ *P*. *g t Q* ∧ *Q* = *P after*$_{\mathcal{T}}$ *t*}. *x* $\sqsubseteq_{DT}$ *Q*;
t ∈ $\mathcal{T}$ *P*; *g t* (*P after*$_{\mathcal{T}}$ *t*)⟧ ⟹ *f x* $\sqsubseteq_{DT}$ *P after*$_{\mathcal{T}}$ *t*⟧ ⟹
(μ *X*. *f X*) $\sqsubseteq_{DT}$ *Q*›
⟨*proof*⟩

corollary *reachable-processes-fix-ind* [*consumes 3*, *case-names cont step*]:
‹⟦*Q* ∈ $\mathcal{R}_{proc}$ *P*;
$\bigwedge$*u v*. ⟦*cont* (*u* :: ('a, 'r) $process_{ptick}$ ⇒ ('a, 'r) $process_{ptick}$); *monofun v*⟧ ⟹ *adm* (λ*x*. *ref* (*u x*) (*v x*));
$\bigwedge$*Q*. *ref* ⊥ *Q*;
cont f;
$\bigwedge$*t x*. ⟦∀ *Q* ∈ $\mathcal{R}_{proc}$ *P*. *ref x Q*; *t* ∈ $\mathcal{T}$ *P*; *tickFree t*⟧ ⟹ *ref* (*f x*) (*P after*$_{\mathcal{T}}$ *t*)⟧

$\Longrightarrow$
ref (μ *x*. *f x*) *Q*›
⟨*proof*⟩

corollary *reachable-processes-fix-ind-F* [*consumes 1*, *case-names cont step*]:
‹⟦$Q \in \mathcal{R}_{proc}$ *P*; *cont f*;
$\bigwedge t\ x.\ \forall\ Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_F Q \Longrightarrow t \in \mathcal{T}\ P \Longrightarrow$ *tickFree t* $\Longrightarrow f\ x \sqsubseteq_F P$ *after*$_{\mathcal{T}}$ *t*⟧ $\Longrightarrow$
$(\mu\ X.\ f\ X) \sqsubseteq_F Q$›
and *reachable-processes-fix-ind-D* [*consumes 1*, *case-names cont step*]:
‹⟦$Q \in \mathcal{R}_{proc}$ *P*; *cont f*;
$\bigwedge t\ x.\ \forall\ Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_D Q \Longrightarrow t \in \mathcal{T}\ P \Longrightarrow$ *tickFree t* $\Longrightarrow f\ x \sqsubseteq_D P$ *after*$_{\mathcal{T}}$ *t*⟧ $\Longrightarrow$
$(\mu\ X.\ f\ X) \sqsubseteq_D Q$›
and *reachable-processes-fix-ind-T* [*consumes 1*, *case-names cont step*]:
‹⟦$Q \in \mathcal{R}_{proc}$ *P*; *cont f*;
$\bigwedge t\ x.\ \forall\ Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_T Q \Longrightarrow t \in \mathcal{T}\ P \Longrightarrow$ *tickFree t* $\Longrightarrow f\ x \sqsubseteq_T P$ *after*$_{\mathcal{T}}$ *t*⟧ $\Longrightarrow$
$(\mu\ X.\ f\ X) \sqsubseteq_T Q$›
and *reachable-processes-fix-ind-FD* [*consumes 1*, *case-names cont step*]:
‹⟦$Q \in \mathcal{R}_{proc}$ *P*; *cont f*;
$\bigwedge t\ x.\ \forall\ Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_{FD} Q \Longrightarrow t \in \mathcal{T}\ P \Longrightarrow$ *tickFree t* $\Longrightarrow f\ x \sqsubseteq_{FD} P$ *after*$_{\mathcal{T}}$ *t*⟧ $\Longrightarrow$
$(\mu\ X.\ f\ X) \sqsubseteq_{FD} Q$›
and *reachable-processes-fix-ind-DT* [*consumes 1*, *case-names cont step*]:
‹⟦$Q \in \mathcal{R}_{proc}$ *P*; *cont f*;
$\bigwedge t\ x.\ \forall\ Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_{DT} Q \Longrightarrow t \in \mathcal{T}\ P \Longrightarrow$ *tickFree t* $\Longrightarrow f\ x \sqsubseteq_{DT} P$ *after*$_{\mathcal{T}}$ *t*⟧ $\Longrightarrow$
$(\mu\ X.\ f\ X) \sqsubseteq_{DT} Q$›
⟨*proof*⟩

8.1.2 New idea: (*after*) induct instead of (*after*$_{\mathcal{T}}$)

8.1.3 New results on $\mathcal{R}_{proc}$

lemma *reachable-processes-FD-refinement-propagation-induct* [*consumes 1*, *case-names cont base step*]:
— May be generalized or duplicated to other refinements.
assumes *reachable* : ‹(Q :: ('*a*, '*r*) *process*$_{ptick}$) $\in \mathcal{R}_{proc}$ *P*›
and *cont-f* : ‹*cont f*›
and *base* : ‹$(\mu\ x.\ f\ x) \sqsubseteq_{FD} P$›
and *step* : ‹$\bigwedge a.\ a \in \alpha(P) \Longrightarrow f\ (\mu\ x.\ f\ x)$ *after* $a = (\mu\ x.\ f\ x)$›
shows ‹$(\mu\ x.\ f\ x) \sqsubseteq_{FD} Q$›
⟨*proof*⟩

theorem $\mathcal{R}_{proc}$*-fix-ind* [*consumes 3*, *case-names cont step*]:
fixes *ref* :: ‹[('*a*, '*r*) *process*$_{ptick}$, ('*a*, '*r*) *process*$_{ptick}$] ⇒ *bool*› (**infix** ‹$\sqsubseteq_{ref}$› *60*)
assumes *reachable* : ‹$Q \in \mathcal{R}_{proc}$ *P*›

and *adm-ref* : ‹$\bigwedge u\ v.\ cont\ (u :: ('a, 'r)\ process_{ptick} \Rightarrow ('a, 'r)\ process_{ptick})$ $\Longrightarrow monofun\ v$
$\Longrightarrow adm\ (\lambda x.\ u\ x \sqsubseteq_{ref}\ v\ x)$›
and *BOT-le-ref* : ‹$\bigwedge Q.\ \bot \sqsubseteq_{ref}\ Q$›
and *cont-f* : ‹*cont f*›
and *hyp* : ‹$\bigwedge x.\ \forall\, Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_{ref}\ Q \Longrightarrow \forall\, Q \in \mathcal{R}_{proc}\ P.\ f\ x \sqsubseteq_{ref}\ Q$›
shows ‹$(\mu\ X.\ f\ X) \sqsubseteq_{ref}\ Q$›
⟨*proof*⟩

lemma $\mathcal{R}_{proc}$*-fix-ind-FD* [*consumes 1*, *case-names cont step*]:
‹$[\![Q \in \mathcal{R}_{proc}\ P;\ cont\ f;\ \bigwedge x\ Q.\ \forall\, Q \in \mathcal{R}_{proc}\ P.\ x \sqsubseteq_{FD}\ Q \Longrightarrow Q \in \mathcal{R}_{proc}\ P \Longrightarrow f\ x \sqsubseteq_{FD}\ Q]\!]$
$\Longrightarrow (\mu\ X.\ f\ X) \sqsubseteq_{FD}\ Q$›
⟨*proof*⟩

8.1.4 Induction Proofs

Generalizations

lemma ‹*Mprefix A P* = *Mprefix B Q* ⟹ *A* = *B*›
⟨*proof*⟩

lemma ‹*Mndetprefix A P* = *Mprefix B Q* ⟹ *A* = *B*›
⟨*proof*⟩

lemma ‹*Mndetprefix A P* = *Mndetprefix B Q* ⟹ *A* = *B*›
⟨*proof*⟩

print-context

lemma *superset-initials-restriction-Mndetprefix-FD*:
‹$\sqcap a \in B \rightarrow P\ a \sqsubseteq_{FD}\ Q$›
if ‹$\sqcap a \in A \rightarrow P\ a \sqsubseteq_{FD}\ Q$› **and** ‹$\{e.\ ev\ e \in Q^0\} \subseteq B$› **and** ‹$A \neq \{\} \vee B = \{\}$›
⟨*proof*⟩

corollary *initials-restriction-Mndetprefix-FD*:
‹$\sqcap a \in A \rightarrow P\ a \sqsubseteq_{FD}\ Q \Longrightarrow \sqcap a \in \{e.\ ev\ e \in Q^0\} \rightarrow P\ a \sqsubseteq_{FD}\ Q$›
⟨*proof*⟩

corollary *events-of-restriction-Mndetprefix-FD*:
‹$\sqcap a \in A \rightarrow P\ a \sqsubseteq_{FD}\ (Q :: ('a, 'r)\ process_{ptick}) \Longrightarrow \sqcap a \in \alpha(Q) \rightarrow P\ a \sqsubseteq_{FD}\ Q$›
⟨*proof*⟩

lemma *superset-initials-restriction-Mprefix-FD*:
‹$\square a \in B \rightarrow P\ a \sqsubseteq_{FD}\ Q$›

if ‹$\square a \in A \rightarrow P\ a \sqsubseteq_{FD}\ Q$› **and** ‹$\{e.\ ev\ e \in Q^0\} \subseteq B$›
and ‹$B \subseteq A$› — Stronger assumption than with *Mndetprefix*.
⟨*proof*⟩

corollary *initials-restriction-Mprefix-FD*:
‹$\{e.\ ev\ e \in Q^0\} \subseteq A \Longrightarrow \square a \in A \rightarrow P\ a \sqsubseteq_{FD}\ Q \Longrightarrow$
$\square a \in \{e.\ ev\ e \in Q^0\} \rightarrow P\ a \sqsubseteq_{FD}\ Q$›
⟨*proof*⟩

corollary *events-of-restriction-Mprefix-FD*:
‹$\alpha(Q) \subseteq A \Longrightarrow \square a \in A \rightarrow P\ a \sqsubseteq_{FD} (Q :: ('a, 'r)\ process_{ptick}) \Longrightarrow$
$\square a \in \alpha(Q) \rightarrow P\ a \sqsubseteq_{FD}\ Q$›
⟨*proof*⟩
lemma *superset-initials-restriction-Mprefix-DT*:
‹$\square a \in B \rightarrow P\ a \sqsubseteq_{DT}\ Q$› **if** ‹$\square a \in A \rightarrow P\ a \sqsubseteq_{DT}\ Q$› **and** ‹$\{e.\ ev\ e \in Q^0\} \subseteq B$›
⟨*proof*⟩

corollary *initials-restriction-Mprefix-DT*:
‹$\square a \in A \rightarrow P\ a \sqsubseteq_{DT}\ Q \Longrightarrow \square a \in \{e.\ ev\ e \in Q^0\} \rightarrow P\ a \sqsubseteq_{DT}\ Q$›
⟨*proof*⟩

corollary *events-restriction-Mprefix-DT*:
‹$\square a \in A \rightarrow P\ a \sqsubseteq_{DT} (Q :: ('a, 'r)\ process_{ptick}) \Longrightarrow \square a \in \alpha(Q) \rightarrow P\ a \sqsubseteq_{DT}\ Q$›
⟨*proof*⟩

Admissibility **lemma** *not-le-F-adm*[*simp*]: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ \neg\ u\ x \sqsubseteq_F P)$›
⟨*proof*⟩

lemma *not-le-T-adm*[*simp*]: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ \neg\ u\ x \sqsubseteq_T P)$›
⟨*proof*⟩

lemma *not-le-D-adm*[*simp*]: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ \neg\ u\ x \sqsubseteq_D P)$›
⟨*proof*⟩

lemma *not-le-FD-adm*[*simp*]: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ \neg\ u\ x \sqsubseteq_{FD} P)$›
⟨*proof*⟩

lemma *not-le-DT-adm*[*simp*]: ‹$cont\ u \Longrightarrow adm\ (\lambda x.\ \neg\ u\ x \sqsubseteq_{DT} P)$›
⟨*proof*⟩

lemma *initials-refusal*:
‹$(t,\ UNIV) \in \mathcal{F}\ P$› **if** *assms*: ‹$t \in \mathcal{T}\ P$› ‹$tF\ t$› ‹$(t,\ (P\ after_{\mathcal{T}}\ t)^0) \in \mathcal{F}\ P$›
⟨*proof*⟩

lemma *leF-ev-initialE'* :
assumes ‹$STOP \sqcap (\sqcap a \in UNIV \rightarrow P\ a) \sqsubseteq_F Q$› ‹$Q \neq STOP$› **obtains** a **where** ‹$ev\ a \in Q^0$›
⟨*proof*⟩

corollary *leF-ev-initialE* :
assumes ‹$\sqcap a \in UNIV \rightarrow P\ a \sqsubseteq_F Q$› **obtains** a **where** ‹$ev\ a \in Q^0$›
⟨*proof*⟩

lemma *leFD-ev-initialE'* :
‹$STOP \sqcap (\sqcap a \in UNIV \rightarrow P\ a) \sqsubseteq_{FD} Q \Longrightarrow Q \neq STOP \Longrightarrow (\bigwedge a.\ ev\ a \in Q^0 \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

lemma *leFD-ev-initialE* :
‹$\sqcap a \in UNIV \rightarrow P\ a \sqsubseteq_{FD} Q \Longrightarrow (\bigwedge a.\ ev\ a \in Q^0 \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

method *prove-propagation* **uses** *simp base* =
induct rule: *reachable-processes-FD-refinement-propagation-induct*,
solves simp, *solves* ‹*use base in* ‹*simp add*: *simp*››, *solves* ‹*simp add*: *simp*›

The three following results illustrate how powerful are our new rules of induction.

Really ? The second version with ⟦$cont\ ?F$; $cont\ ?G$; $adm\ (\lambda x.\ ?P\ (fst\ x)\ (snd\ x))$; $?P \perp \perp$; $\bigwedge x\ y.\ ?P\ x\ y \Longrightarrow ?P\ (?F\ x)\ (?G\ y)$⟧ $\Longrightarrow ?P\ (fix\text{-}syn\ ?F)\ (fix\text{-}syn\ ?G)$ seems easier...

lemma
‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow DF\ \alpha(P) \sqsubseteq_F Q$› **if** *df-P*: ‹*deadlock-free P*›
⟨*proof*⟩

lemma
‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow DF_{SKIPS}\ \alpha(P)\ UNIV \sqsubseteq_F Q$› **if** df_{SKIP}*-P*: ‹*deadlock-free*$_{SKIPS}$ *P*›
⟨*proof*⟩

context fixes P :: ‹$('a, 'r)\ process_{ptick}$› **begin**

theorem *deadlock-free-iff-empty-ticks-of-and-deadlock-free*$_{SKIPS}$:
‹$deadlock\text{-}free\ P \longleftrightarrow ✓s(P) = \{\} \wedge deadlock\text{-}free_{SKIPS}\ P$›

⟨*proof*⟩

lemma *reachable-processes-DF-UNIV-leF-imp-DF-events-of-leF* :
‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow DF\ UNIV \sqsubseteq_F Q \Longrightarrow DF\ \alpha(P) \sqsubseteq_F Q$› **for** Q
⟨*proof*⟩

lemma *reachable-processes-CHAOS-UNIV-leF-imp-CHAOS-events-of-leF* :
‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow CHAOS\ UNIV \sqsubseteq_F Q \Longrightarrow CHAOS\ \alpha(P) \sqsubseteq_F Q$› **for** Q
⟨*proof*⟩

lemma *reachable-processes-$CHAOS_{SKIPS}$-UNIV-UNIV-leF-imp-CHAOS-events-of-ticks-of-leFD* :
‹$Q \in \mathcal{R}_{proc}\ P \Longrightarrow CHAOS_{SKIPS}\ UNIV\ UNIV \sqsubseteq_{FD} Q \Longrightarrow CHAOS_{SKIPS}\ \alpha(P)\ \checkmark s(P) \sqsubseteq_{FD} Q$› **for** Q
⟨*proof*⟩

theorem *deadlock-free-iff-DF-events-of-leF* :
‹$deadlock\text{-}free\ P \longleftrightarrow \alpha(P) \neq \{\} \land DF\ \alpha(P) \sqsubseteq_F P$›
⟨*proof*⟩

corollary *deadlock-free-iff-DF-events-of-leFD* :
‹$deadlock\text{-}free\ P \longleftrightarrow \alpha(P) \neq \{\} \land DF\ \alpha(P) \sqsubseteq_{FD} P$›
⟨*proof*⟩

corollary *deadlock-free-iff-DF-strict-events-of-leF* :
‹$deadlock\text{-}free\ P \longleftrightarrow \boldsymbol{\alpha}(P) \neq \{\} \land DF\ \boldsymbol{\alpha}(P) \sqsubseteq_F P$›
⟨*proof*⟩

corollary *deadlock-free-iff-DF-strict-events-of-leFD* :
‹$deadlock\text{-}free\ P \longleftrightarrow \boldsymbol{\alpha}(P) \neq \{\} \land DF\ \boldsymbol{\alpha}(P) \sqsubseteq_{FD} P$›
⟨*proof*⟩

theorem *lifelock-free-iff-CHAOS-events-of-leF* :
‹$lifelock\text{-}free\ P \longleftrightarrow CHAOS\ \alpha(P) \sqsubseteq_F P$›
⟨*proof*⟩

corollary *lifelock-free-iff-CHAOS-strict-events-of-leF* :
‹$lifelock\text{-}free\ P \longleftrightarrow CHAOS\ \boldsymbol{\alpha}(P) \sqsubseteq_F P$›
⟨*proof*⟩

corollary *lifelock-free-iff-CHAOS-events-of-leFD* :

‹*lifelock-free* $P \longleftrightarrow$ *CHAOS* $\alpha(P) \sqsubseteq_{FD} P$›
⟨*proof*⟩

corollary *lifelock-free-iff-CHAOS-strict-events-of-leFD* :
‹*lifelock-free* $P \longleftrightarrow$ *CHAOS* $\boldsymbol{\alpha}(P) \sqsubseteq_{FD} P$›
⟨*proof*⟩

theorem *lifelock-free$_{SKIPS}$-iff-CHAOS$_{SKIPS}$-events-of-ticks-of-FD* :
‹*lifelock-free$_{SKIPS}$* $P \longleftrightarrow$ *CHAOS$_{SKIPS}$* $\alpha(P)$ ✓$s(P) \sqsubseteq_{FD} P$›
⟨*proof*⟩

corollary *lifelock-free$_{SKIPS}$-iff-CHAOS$_{SKIPS}$-strict-events-of-strict-ticks-of-FD* :
‹*lifelock-free$_{SKIPS}$* $P \longleftrightarrow$ *CHAOS$_{SKIPS}$* $\boldsymbol{\alpha}(P)$ ✓$\mathbf{s}(P) \sqsubseteq_{FD} P$›
⟨*proof*⟩

theorem *deadlock-free$_{SKIPS}$-iff-DF$_{SKIPS}$-events-of-ticks-of-leF* :
‹*deadlock-free$_{SKIPS}$* $P \longleftrightarrow$ (*if* $\alpha(P) = \{\} \wedge$ ✓$s(P) = \{\}$ *then False*
else *if* ✓$s(P) = \{\}$ *then DF* $\alpha(P) \sqsubseteq_F P$
else *if* $\alpha(P) = \{\}$ *then SKIPS* ✓$s(P) \sqsubseteq_F P$
else DF$_{SKIPS}$ $\alpha(P)$ ✓$s(P) \sqsubseteq_F P$)›
(**is** ‹- $\longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

corollary *deadlock-free$_{SKIPS}$-iff-DF$_{SKIPS}$-events-of-ticks-of-leFD* :
‹*deadlock-free$_{SKIPS}$* $P \longleftrightarrow$ (*if* $\alpha(P) = \{\} \wedge$ ✓$s(P) = \{\}$ *then False*
else *if* ✓$s(P) = \{\}$ *then DF* $\alpha(P) \sqsubseteq_{FD} P$
else *if* $\alpha(P) = \{\}$ *then SKIPS* ✓$s(P) \sqsubseteq_{FD} P$
else DF$_{SKIPS}$ $\alpha(P)$ ✓$s(P) \sqsubseteq_{FD} P$)›
⟨*proof*⟩

corollary *deadlock-free$_{SKIPS}$-iff-DF$_{SKIPS}$-strict-events-of-strict-ticks-of-leF* :
‹*deadlock-free$_{SKIPS}$* $P \longleftrightarrow$ (*if* $\boldsymbol{\alpha}(P) = \{\} \wedge$ ✓$\mathbf{s}(P) = \{\}$ *then False*
else *if* ✓$\mathbf{s}(P) = \{\}$ *then DF* $\boldsymbol{\alpha}(P) \sqsubseteq_F P$
else *if* $\boldsymbol{\alpha}(P) = \{\}$ *then SKIPS* ✓$\mathbf{s}(P) \sqsubseteq_F P$
else DF$_{SKIPS}$ $\boldsymbol{\alpha}(P)$ ✓$\mathbf{s}(P) \sqsubseteq_F P$)›
⟨*proof*⟩

corollary *deadlock-free$_{SKIPS}$-iff-DF$_{SKIPS}$-strict-events-of-strict-ticks-of-leFD* :
‹*deadlock-free$_{SKIPS}$* $P \longleftrightarrow$ (*if* $\boldsymbol{\alpha}(P) = \{\} \wedge$ ✓$\mathbf{s}(P) = \{\}$ *then False*
else *if* ✓$\mathbf{s}(P) = \{\}$ *then DF* $\boldsymbol{\alpha}(P) \sqsubseteq_{FD} P$
else *if* $\boldsymbol{\alpha}(P) = \{\}$ *then SKIPS* ✓$\mathbf{s}(P) \sqsubseteq_{FD} P$
else DF$_{SKIPS}$ $\boldsymbol{\alpha}(P)$ ✓$\mathbf{s}(P) \sqsubseteq_{FD} P$)›
⟨*proof*⟩

8.1.5 Big results

As consequences, we have very powerful results, and especially a "data independence" deadlock freeness theorem.

lemma *deadlock-free-is-right*:
‹*deadlock-free* $P \longleftrightarrow (\forall t \in \mathcal{T}\ P.\ tF\ t \wedge (t, \quad UNIV) \notin \mathcal{F}\ P)$›
‹*deadlock-free* $P \longleftrightarrow (\forall t \in \mathcal{T}\ P.\ tF\ t \wedge (t,\ ev\ `\ UNIV) \notin \mathcal{F}\ P)$›
⟨*proof*⟩

end

— We may probably prove *deadlock-free* $P = (\forall t{\in}\mathcal{T}\ P.\ tF\ t \wedge (t,\ ev\ `\ \alpha(P)) \notin \mathcal{F}\ P)$

theorem *data-independence-deadlock-free-Sync*:
fixes $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$›
assumes *df-P* : ‹*deadlock-free* P› **and** *df-Q* : ‹*deadlock-free* Q›
and *hyp* : ‹*events-of* $Q \cap S = \{\} \vee (\exists y.$ *events-of* $Q \cap S = \{y\} \wedge$ *events-of* $P \cap S \subseteq \{y\})$›
shows ‹*deadlock-free* $(P\ [\![S]\!]\ Q)$›
⟨*proof*⟩

lemma *data-independence-deadlock-free-Sync-bis*:
‹$[\![$*deadlock-free* P; *deadlock-free* Q; $\alpha(Q) \cap S = \{\}]\!] \Longrightarrow$
deadlock-free $(P\ [\![S]\!]\ Q)$› **for** P :: ‹$('a, 'r)\ process_{ptick}$›
⟨*proof*⟩

We can't expect much better without hypothesis on the processes P and Q. We can easily build the following counter example.

lemma ‹$\exists P\ Q\ S.$ *deadlock-free* $P \wedge$ *deadlock-free* $Q \wedge$
$(\exists y\ z.$ *events-of* $Q \cap S \subseteq \{y, z\} \wedge$ *events-of* $P \cap S \subseteq \{y, z\}) \wedge$
$\neg$ *deadlock-free* $(P\ [\![S :: nat\ set]\!]\ Q)$›
⟨*proof*⟩

end

find-theorems *name*: *data-independence-deadlock-free-Sync-bis*

— Think about a $deadlock\text{-}free_{SKIPS}$ version.

8.1.6 Results with other references Processes

RUN **and** *non-terminating*

lemma *non-terminating-STOP* [*simp*] : ‹*non-terminating STOP*›
⟨*proof*⟩

lemma *not-non-terminating-SKIP* [*simp*]: ‹¬ *non-terminating* (*SKIP r*)›
⟨*proof*⟩

lemma *not-non-terminating-BOT* [*simp*] : ‹¬ *non-terminating* ⊥›
⟨*proof*⟩

context *AfterExt* **begin**

lemma *non-terminating-iff-RUN-events-T*:
‹*non-terminating P* ⟷ *RUN* $\alpha(P)$ $\sqsubseteq_T$ *P*› **for** *P* :: ‹('*a*, '*r*) $process_{ptick}$›
⟨*proof*⟩

lemma *lifelock-free-F*: ‹*lifelock-free P* ⟷ *CHAOS UNIV* $\sqsubseteq_F$ *P*›
⟨*proof*⟩

end

Chapter 9

Bonus: powerful new Laws

9.1 Powerful Results about *Sync*

lemma *add-complementary-events-of-in-failure*:
‹$(t, X) \in \mathcal{F}\ P \Longrightarrow (t, X \cup ev\ `\ (-\ \alpha(P))) \in \mathcal{F}\ P$›
⟨*proof*⟩

lemma *add-complementary-initials-in-refusal*: ‹$X \in \mathcal{R}\ P \Longrightarrow X \cup -\ P^0 \in \mathcal{R}\ P$›
⟨*proof*⟩

lemma *TickRightSync*:
‹$\checkmark(r) \in S \Longrightarrow ftF\ u \Longrightarrow t\ setinterleaves\ ((u,\ [\checkmark(r)]),\ S) \Longrightarrow t = u \wedge last\ u = \checkmark(r)$›
⟨*proof*⟩

theorem *Sync-is-Sync-restricted-superset-events*:
fixes $S\ A$ :: ‹$'a\ set$› **and** $P\ Q$ :: ‹$('a,\ 'r)\ process_{ptick}$›
assumes *superset* : ‹$\alpha(P) \cup \alpha(Q) \subseteq A$›
defines ‹$S' \equiv S \cap A$›
shows ‹$P\ [\![S]\!]\ Q = P\ [\![S']\!]\ Q$›
⟨*proof*⟩

corollary *Sync-is-Sync-restricted-events* : ‹$P\ [\![S]\!]\ Q = P\ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]\ Q$›
⟨*proof*⟩

This version is closer to the intuition that we may have, but the first one would be more useful if we don't want to compute the events of a process but know a superset approximation.

corollary ‹$deadlock\text{-}free\ P \Longrightarrow deadlock\text{-}free\ Q \Longrightarrow$
$S \cap (\alpha(P) \cup \alpha(Q)) = \{\} \Longrightarrow deadlock\text{-}free\ (P\ [\![S]\!]\ Q)$›
⟨*proof*⟩

9.2 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.

9.2.1 Some Generalizations

For *Renaming*, we can obtain generalizations of the following results:
Renaming (*Mprefix A P*) *f g* = □*y*∈*f* ‘ *A* → ⊓*a*∈{*x* ∈ *A*. *y* = *f x*}. *Renaming* (*P a*) *f g*
Renaming (*Mndetprefix A P*) *f g* = ⊓*b*∈*f* ‘ *A* → ⊓*a*∈{*a* ∈ *A*. *b* = *f a*}. *Renaming* (*P a*) *f g*

lemma *Renaming-Mprefix-Sliding*:
‹*Renaming* ((□*a* ∈ *A* → *P a*) ▷ *Q*) *f g* =
(□*y* ∈ *f* ‘ *A* → ⊓*a* ∈ {*x* ∈ *A*. *y* = *f x*}. *Renaming* (*P a*) *f g*) ▷ *Renaming Q f g*›
⟨*proof*⟩

9.2.2 *Renaming* and (\)

When *f* is one to one, *Renaming* (*P* \ *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-map-event*$_{ptick}$:
‹*inj-on* (*map-event*$_{ptick}$ *f g*) (*set s* ∪ *ev* ‘ *S*) ⟹
trace-hide (*map* (*map-event*$_{ptick}$ *f g*) *s*) (*ev* ‘ *f* ‘ *S*) =
map (*map-event*$_{ptick}$ *f g*) (*trace-hide s* (*ev* ‘ *S*))›
⟨*proof*⟩

theorem *bij-Renaming-Hiding*: ‹*Renaming* (*P* \ *S*) *f g* = *Renaming P f g* \ *f* ‘ *S*›
(**is** ‹*?lhs* = *?rhs*›) **if** *bij-f*: ‹*bij f*› **and** *bij-g* : ‹*bij g*›
⟨*proof*⟩

9.2.3 *Renaming* and *Sync*

Idem for the synchronization: when *f* is one to one, *Renaming* (*P* ⟦*S*⟧ *Q*) will behave as expected.

lemma *map-antecedent-if-subset-rangeE* :
assumes ‹*set u* ⊆ *range f*›

obtains t **where** ‹$u = map\ f\ t$›
— In particular, when f is surjective or bijective.
⟨*proof*⟩

lemma *bij-map-setinterleaving-iff-setinterleaving* :
‹$map\ f\ r$ *setinterleaves* $((map\ f\ t,\ map\ f\ u),\ f\ `\ S) \longleftrightarrow$
r *setinterleaves* $((t,\ u),\ S)$› **if** *bij-f* : ‹*bij f*›
⟨*proof*⟩

theorem *bij-Renaming-Sync*:
‹$Renaming\ (P\ [\![S]\!]\ Q)\ f\ g = Renaming\ P\ f\ g\ [\![f\ `\ S]\!]\ Renaming\ Q\ f\ g$›
(**is** ‹$?lhs\ P\ Q = ?rhs\ P\ Q$›) **if** *bij-f*: ‹*bij f*› **and** *bij-g* : ‹*bij g*›
⟨*proof*⟩

9.3 (\) and *Mprefix*

We already have a way to distribute the (\) operator on the *Mprefix* operator with $S \cap A = \{\} \Longrightarrow Mprefix\ S\ ?P \setminus A = \square a{\in}S \to (?P\ a \setminus A)$. But this is only usable when $A \cap S = \{\}$. With the ($\rhd$) operator, we can now handle the case $A \cap S \neq \{\}$.

9.3.1 (\) and *Mprefix* for disjoint Sets

This is a result similar to $?A \cap ?S = \{\} \Longrightarrow Mprefix\ ?A\ ?P \setminus ?S = \square a{\in}?A \to (?P\ a \setminus ?S)$ when we add a ($\rhd$) in the expression.

theorem *Hiding-Mprefix-Sliding-disjoint*:
‹$((\square a \in A \to P\ a) \rhd Q) \setminus S = (\square a \in A \to (P\ a \setminus S)) \rhd (Q \setminus S)$›
if *disjoint*: ‹$A \cap S = \{\}$›
⟨*proof*⟩

9.3.2 (\) and *Mprefix* for non-disjoint Sets

Finally the new version, when $A \cap S \neq \{\}$.

lemma ‹$\exists\, A$ :: *nat set*. $\exists\, P\ S$.
$A \cap S = \{\} \land \square a \in A \to P\ a \setminus S \neq$
$(\square a \in (A - S) \to (P\ a \setminus S)) \rhd (\sqcap a \in (A \cap S).\ (P\ a \setminus S))$›
⟨*proof*⟩

This is a result similar to $?A \cap ?S \neq \{\} \Longrightarrow Mprefix\ ?A\ ?P \setminus ?S = (\square a{\in}(?A - ?S) \to (?P\ a \setminus ?S)) \rhd (\sqcap a{\in}(?A \cap ?S).\ (?P\ a \setminus ?S))$ when we add a ($\rhd$) in the expression.

lemma *Hiding-Mprefix-Sliding-non-disjoint*:
‹$(\square a \in A \to P\ a) \rhd Q \setminus S = (\square a \in (A - S) \to (P\ a \setminus S)) \rhd$
$(Q \setminus S) \sqcap (\sqcap a \in (A \cap S).\ (P\ a \setminus S))$›

if *non-disjoint*: ‹$A \cap S \neq \{\}$›
⟨*proof*⟩

9.4 (▷) behaviour

We already proved several laws for the (▷) 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*:
‹$(\square a \in A \to P\ a) \rhd (\square b \in B \to Q\ b) \rhd R =$
$(\square\ x \in (A \cup B) \to (\mathit{if}\ x \in A \cap B\ \mathit{then}\ P\ x \sqcap Q\ x\ \mathit{else}\ \mathit{if}\ x \in A\ \mathit{then}\ P\ x\ \mathit{else}\ Q\ x)) \rhd R$›
(**is** ‹$(\square a \in A \to P\ a) \rhd (\square b \in B \to Q\ b) \rhd R = ?term \rhd R$›)
⟨*proof*⟩

lemma *Mprefix-Sliding-Seq*:
‹$(\square a \in A \to P\ a) \rhd P' \mathbin{;} Q = (\square a \in A \to (P\ a \mathbin{;} Q)) \rhd (P' \mathbin{;} Q)$›
⟨*proof*⟩

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

9.5 Dealing with *SKIP*

lemma *Renaming-Mprefix-Det-SKIP*:
‹$\mathit{Renaming}\ ((\square\ a \in A \to P\ a) \square \mathit{SKIP}\ r)\ f\ g =$
$(\square y \in f\ `\ A \to \sqcap\ a \in \{x \in A.\ y = f\ x\}.\ \mathit{Renaming}\ (P\ a)\ f\ g) \square \mathit{SKIP}\ (g\ r)$›

⟨*proof*⟩

lemma *Mprefix-Sliding-SKIP-Seq*: ‹$((\square\ a \in A \to P\ a) \rhd \mathit{SKIP}\ r) \mathbin{;} Q = (\square\ a \in A \to (P\ a \mathbin{;} Q)) \rhd Q$›

⟨*proof*⟩

lemma *Mprefix-Det-SKIP-Seq*: ‹$((\square\ a \in A \to P\ a) \square \mathit{SKIP}\ r) \mathbin{;} Q = (\square\ a \in A \to (P\ a \mathbin{;} Q)) \rhd Q$›
⟨*proof*⟩

lemma *Sliding-Ndet-pseudo-assoc* : ‹$(P \rhd Q) \sqcap R = P \rhd Q \sqcap R$›

$\langle proof \rangle$

lemma *Hiding-Mprefix-Det-SKIP*:
‹(□ a ∈ A → P a) □ SKIP r \ S =
(if A ∩ S = {} then (□ a ∈ A → (P a \ S)) □ SKIP r
else ((□ a ∈ (A − S) → (P a \ S)) □ SKIP r) ⊓ (⊓ a ∈ (A ∩ S). (P a \ S)))›
$\langle proof \rangle$

lemma ‹s ≠ [] ⟹ (s, X) ∈ $\mathcal{F}$ (P □ Q) ⟷ (s, X) ∈ $\mathcal{F}$ (P ⊓ Q)›
$\langle proof \rangle$

lemma *Mprefix-Det-SKIP-Sync-SKIP* :
‹((□ a ∈ A → P a) □ SKIP res) ⟦S⟧ SKIP res' =
(if res = res' then (□ a ∈ (A − S) → (P a ⟦S⟧ SKIP res')) □ SKIP res'
else (□ a ∈ (A − S) → (P a ⟦S⟧ SKIP res')) ⊓ STOP)›
(**is** ‹?lhs = (if res = res' then ?rhs1 else ?rhs2)›)
$\langle proof \rangle$

lemma *Sliding-def-bis* : ‹P ▷ Q = (P ⊓ Q) □ Q›
$\langle proof \rangle$

Chapter 10

Conclusion

We started by defining the operators ($\triangleright$), *Throw* and ($\triangle$) and provided on them several new laws, especially monotony, "step-law" (behaviour with $\square a \in A \rightarrow P\ a$) and continuity.

We defined the *initials* notion, and described its behaviour with the reference processes and the operators of `HOL-CSP` and `HOL-CSPM` (which is already a minor contribution).

As main contribution, we defined the *After.After* operator which represents a bridge between the denotational and the versions of operational semantics for CSP. We made the construction as generic as possible, by exploiting the locale mechanism. 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 [4, 6].
We have slight variations that can open up for discussion.
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 ($\sqsubseteq_{DT}$) 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 with ($\sqsubseteq_F$) and ($\sqsubseteq_T$) orders but this remains a bit disappointing because their monotony w.r.t. to some operators does not allow to recover all the laws of [4, 6].

As a bonus we provided in *HOL−CSP-OpSem.CSP-New-Laws* some powerful laws for CSP. Here, we recall only the most important ones:

$$\frac{bij\ ?f \qquad bij\ ?g}{Renaming\ (?P \setminus ?S)\ ?f\ ?g = Renaming\ ?P\ ?f\ ?g \setminus ?f\ `\ ?S}$$

$$\frac{bij\ ?f \qquad bij\ ?g}{Renaming\ (?P\ [\![?S]\!]\ ?Q)\ ?f\ ?g = Renaming\ ?P\ ?f\ ?g\ [\![?f\ `\ ?S]\!]\ Renaming\ ?Q\ ?f\ ?g}$$

$$\frac{?A \cap ?S \neq \emptyset}{\Box a{\in}?A \to ?P\ a \setminus ?S = (\Box a{\in}(?A - ?S) \to (?P\ a \setminus ?S)) \rhd (\sqcap a{\in}(?A \cap ?S).\ (?P\ a \setminus ?S))}$$

Finally, we discovered that the *After.After* operator and its extensions *AfterExt.After*$_{tick}$ and *AfterExt.After*$_{trace}$ have a real interest even without the construction of operational semantics.

With induction rules based on *AfterExt.After*$_{trace}$, we could for example prove the following theorem:

$$\frac{deadlock\text{-}free\ ?P \qquad deadlock\text{-}free\ ?Q \qquad \alpha(?Q) \cap ?S = \emptyset}{deadlock\text{-}free\ (?P\ [\![?S]\!]\ ?Q)}$$

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] B. Ballenghien and B. Wolff. An operational semantics in isabelle/hol-csp. In Y. Bertot, T. Kutsia, and M. Norrish, editors, *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[3] 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.

[4] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.

[5] A. W. Roscoe. Understanding concurrent systems. In *Texts in Computer Science*, 2010.

[6] 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.

[7] 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.