HOL-CSP_RS: CSP Semantics over Restriction Spaces

Benoît Ballenghien Burkhart Wolff

June 3, 2025

Abstract

We use the `Restriction_Spaces` library as a semantic foundation for the process algebra framework `HOL-CSP`, offering a complementary backend to the existing `HOLCF` infrastructure. The type of processes is instantiated as a restriction space, and we prove that it is complete in this setting. This enables the construction of fixed points for recursive process definitions without having to rely exclusively on a pointed complete partial order. Notably, some operators are constructive without being Scott-continuous, and vice versa, illustrating the genuine complementarity between the two approaches. We also show that key CSP operators are either constructive or non-destructive, and verify the admissibility of several predicates, thereby supporting automated reasoning over recursive specifications.

Contents

1 Depth Operator

1.1 Definition

instantiation $process_{ptick}$ :: $(type,\ type)$ *order-restriction-space*
begin

lift-definition $restriction\text{-}process_{ptick}$ ::
‹$[('a,\ 'r)\ process_{ptick},\ nat] \Rightarrow ('a,\ 'r)\ process_{ptick}$›
is ‹$\lambda P\ n.\ (\mathcal{F}\ P \cup \{(t\ @\ u,\ X)\ |t\ u\ X.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\},$
$\mathcal{D}\ P \cup \{\ t\ @\ u\ \ \ \ |t\ u.\ \ \ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\})$›
⟨*proof*⟩

1.2 Projections

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

lemma $F\text{-}restriction\text{-}process_{ptick}$:
‹$\mathcal{F}\ (P \downarrow n) = \mathcal{F}\ P \cup \{(t\ @\ u,\ X)\ |t\ u\ X.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\}$›
⟨*proof*⟩

lemma $D\text{-}restriction\text{-}process_{ptick}$:
‹$\mathcal{D}\ (P \downarrow n) = \mathcal{D}\ P \cup \{t\ @\ u\ |t\ u.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\}$›
⟨*proof*⟩

lemma $T\text{-}restriction\text{-}process_{ptick}$:
‹$\mathcal{T}\ (P \downarrow n) = \mathcal{T}\ P \cup \{t\ @\ u\ |t\ u.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\}$›
⟨*proof*⟩

lemmas $restriction\text{-}process_{ptick}\text{-}projs = F\text{-}restriction\text{-}process_{ptick}\ D\text{-}restriction\text{-}process_{ptick}\ T\text{-}restriction\text{-}process_{ptick}$

lemma $D\text{-}restriction\text{-}process_{ptick}E$:
assumes ‹$t \in \mathcal{D}\ (P \downarrow n)$›
obtains ‹$t \in \mathcal{D}\ P$› **and** ‹$length\ t \leq n$›
| $u\ v$ **where** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ P$› ‹$length\ u = n$› ‹$tF\ u$› ‹$ftF\ v$›
⟨*proof*⟩

lemma $F\text{-}restriction\text{-}process_{ptick}E$:
assumes ‹$(t,\ X) \in \mathcal{F}\ (P \downarrow n)$›
obtains ‹$(t,\ X) \in \mathcal{F}\ P$› **and** ‹$length\ t \leq n$›
| $u\ v$ **where** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ P$› ‹$length\ u = n$› ‹$tF\ u$› ‹$ftF\ v$›
⟨*proof*⟩

lemma *T-restriction-process*$_{ptick}$*E* :
‹$[\![t \in \mathcal{T}\ (P \downarrow n);\ t \in \mathcal{T}\ P \Longrightarrow \mathit{length}\ t \le n \Longrightarrow \mathit{thesis};$
$\bigwedge u\ v.\ t = u\ @\ v \Longrightarrow u \in \mathcal{T}\ P \Longrightarrow \mathit{length}\ u = n \Longrightarrow \mathit{tF}\ u \Longrightarrow \mathit{ftF}\ v \Longrightarrow \mathit{thesis}]\!] \Longrightarrow \mathit{thesis}$›
⟨*proof*⟩

lemmas *restriction-process*$_{ptick}$*-elims* =
F-restriction-process$_{ptick}$*E* *D-restriction-process*$_{ptick}$*E* *T-restriction-process*$_{ptick}$*E*

lemma *D-restriction-process*$_{ptick}$*I* :
‹$t \in \mathcal{D}\ P \lor t \in \mathcal{T}\ P \land (\mathit{length}\ t = n \land \mathit{tF}\ t \lor n < \mathit{length}\ t) \Longrightarrow t \in \mathcal{D}\ (P \downarrow n)$›
⟨*proof*⟩

lemma *F-restriction-process*$_{ptick}$*I* :
‹$(t,\ X) \in \mathcal{F}\ P \lor t \in \mathcal{T}\ P \land (\mathit{length}\ t = n \land \mathit{tF}\ t \lor n < \mathit{length}\ t) \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n)$›
⟨*proof*⟩

lemma *T-restriction-process*$_{ptick}$*I* :
‹$t \in \mathcal{T}\ P \lor t \in \mathcal{T}\ P \land (\mathit{length}\ t = n \land \mathit{tF}\ t \lor n < \mathit{length}\ t) \Longrightarrow t \in \mathcal{T}\ (P \downarrow n)$›
⟨*proof*⟩

lemma *F-restriction-process*$_{ptick}$*-Suc-length-iff-F* :
‹$(t,\ X) \in \mathcal{F}\ (P \downarrow \mathit{Suc}\ (\mathit{length}\ t)) \longleftrightarrow (t,\ X) \in \mathcal{F}\ P$›
and *D-restriction-process*$_{ptick}$*-Suc-length-iff-D* :
‹$t \in \mathcal{D}\ (P \downarrow \mathit{Suc}\ (\mathit{length}\ t)) \longleftrightarrow t \in \mathcal{D}\ P$›
and *T-restriction-process*$_{ptick}$*-Suc-length-iff-T* :
‹$t \in \mathcal{T}\ (P \downarrow \mathit{Suc}\ (\mathit{length}\ t)) \longleftrightarrow t \in \mathcal{T}\ P$›
⟨*proof*⟩

lemma *length-less-in-F-restriction-process*$_{ptick}$:
‹$\mathit{length}\ t < n \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n) \Longrightarrow (t,\ X) \in \mathcal{F}\ P$›
⟨*proof*⟩

lemma *length-le-in-T-restriction-process*$_{ptick}$:
‹$\mathit{length}\ t \le n \Longrightarrow t \in \mathcal{T}\ (P \downarrow n) \Longrightarrow t \in \mathcal{T}\ P$›
⟨*proof*⟩

lemma *length-less-in-D-restriction-process*$_{ptick}$:
‹$\mathit{length}\ t < n \Longrightarrow t \in \mathcal{D}\ (P \downarrow n) \Longrightarrow t \in \mathcal{D}\ P$›

⟨*proof*⟩

lemma *not-tickFree-in-F-restriction-process*$_{ptick}$*-iff* :
‹*length* $t \leq n \Longrightarrow \neg$ *tF* $t \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n) \longleftrightarrow (t,\ X) \in \mathcal{F}\ P$›
⟨*proof*⟩

lemma *not-tickFree-in-D-restriction-process*$_{ptick}$*-iff* :
‹*length* $t \leq n \Longrightarrow \neg$ *tF* $t \Longrightarrow t \in \mathcal{D}\ (P \downarrow n) \longleftrightarrow t \in \mathcal{D}\ P$›
⟨*proof*⟩

end

lemma *front-tickFreeE* :
‹$[\![$*ftF* t; *tF* $t \Longrightarrow$ *thesis*; $\bigwedge t'\ r.\ t = t'\ @\ [✓(r)] \Longrightarrow$ *tF* $t' \Longrightarrow$ *thesis*$]\!]$ $\Longrightarrow$ *thesis*›
⟨*proof*⟩

1.3 Proof obligation

instance
⟨*proof*⟩

corollary ‹*OFCLASS*$(('a,\ 'r)$ *process*$_{ptick}$, *restriction-space-class*$)$›
⟨*proof*⟩

end

instance *process*$_{ptick}$ :: (*type*, *type*) *pcpo-restriction-space*
⟨*proof*⟩

⟨*ML*⟩

1.4 Compatibility with Refinements

lemma *leF-restriction-process*$_{ptick}$*I*: ‹$P \downarrow n \sqsubseteq_F Q \downarrow n$›
if ‹$\bigwedge s\ X.\ (s,\ X) \in \mathcal{F}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow (s,\ X) \in \mathcal{F}\ (P \downarrow n)$›
⟨*proof*⟩

lemma *leT-restriction-process*$_{ptick}$*I*: ‹$P \downarrow n \sqsubseteq_T Q \downarrow n$›
if ‹$\bigwedge s.\ s \in \mathcal{T}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{T}\ (P \downarrow n)$›
⟨*proof*⟩

lemma *leDT-restriction-process*$_{ptick}$*I*: ‹$P \downarrow n \sqsubseteq_{DT} Q \downarrow n$›
if ‹$\bigwedge s.\ s \in \mathcal{T}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{T}\ (P \downarrow n)$›

and ‹$\bigwedge s$. *length* $s \leq n \Longrightarrow s \in \mathcal{D}\ Q \Longrightarrow s \in \mathcal{D}\ (P \downarrow n)$›
⟨*proof*⟩

lemma *leFD-restriction-process*$_{ptick}$*I*: ‹$P \downarrow n \sqsubseteq_{FD} Q \downarrow n$›
if ‹$\bigwedge s\ X.\ (s, X) \in \mathcal{F}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow (s, X) \in \mathcal{F}\ (P \downarrow n)$›
and ‹$\bigwedge s.\ s \in \mathcal{D}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{D}\ (P \downarrow n)$›
⟨*proof*⟩

1.5 First Laws

lemma *restriction-process*$_{ptick}$*-0* [*simp*] : ‹$P \downarrow 0 = \bot$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-BOT* [*simp*] : ‹$(\bot :: ('a, 'r)\ process_{ptick}) \downarrow n = \bot$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-is-BOT-iff* :
‹$P \downarrow n = \bot \longleftrightarrow n = 0 \lor P = \bot$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-STOP* [*simp*] : ‹$STOP \downarrow n =$ (*if* $n = 0$ *then* $\bot$ *else* $STOP$)›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-is-STOP-iff* : ‹$P \downarrow n = STOP \longleftrightarrow n \neq 0 \land P = STOP$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-SKIP* [*simp*] : ‹$SKIP\ r \downarrow n =$ (*if* $n = 0$ *then* $\bot$ *else* $SKIP\ r$)›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-is-SKIP-iff* : ‹$P \downarrow n = SKIP\ r \longleftrightarrow n \neq 0 \land P = SKIP\ r$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-SKIPS* [*simp*] : ‹$SKIPS\ R \downarrow n =$ (*if* $n = 0$ *then* $\bot$ *else* $SKIPS\ R$)›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-is-SKIPS-iff* : ‹$P \downarrow n = SKIPS\ R \longleftrightarrow n \neq 0 \land P = SKIPS\ R$›
⟨*proof*⟩

1.6 Monotony

1.6.1 $P \downarrow n$ is an Approximation of the P

lemma *restriction-process$_{ptick}$-approx-self* : ‹$P \downarrow n \sqsubseteq P$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-FD-self* : ‹$P \downarrow n \sqsubseteq_{FD} P$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-F-self* : ‹$P \downarrow n \sqsubseteq_{F} P$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-D-self* : ‹$P \downarrow n \sqsubseteq_{D} P$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-T-self* : ‹$P \downarrow n \sqsubseteq_{T} P$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-DT-self* : ‹$P \downarrow n \sqsubseteq_{DT} P$›
⟨*proof*⟩

1.6.2 Monotony of $(\downarrow)$

lemma *Suc-right-mono-restriction-process$_{ptick}$* : ‹$P \downarrow n \sqsubseteq P \downarrow Suc\ n$›
⟨*proof*⟩

lemma *Suc-right-mono-restriction-process$_{ptick}$-FD* : ‹$P \downarrow n \sqsubseteq_{FD} P \downarrow Suc\ n$›
⟨*proof*⟩

lemma *Suc-right-mono-restriction-process$_{ptick}$-F* : ‹$P \downarrow n \sqsubseteq_{F} P \downarrow Suc\ n$›
⟨*proof*⟩

lemma *Suc-right-mono-restriction-process$_{ptick}$-D* : ‹$P \downarrow n \sqsubseteq_{D} P \downarrow Suc\ n$›
⟨*proof*⟩

lemma *Suc-right-mono-restriction-process$_{ptick}$-T* : ‹$P \downarrow n \sqsubseteq_{T} P \downarrow Suc\ n$›
⟨*proof*⟩

lemma *Suc-right-mono-restriction-process$_{ptick}$-DT* : ‹$P \downarrow n \sqsubseteq_{DT} P \downarrow Suc\ n$›
⟨*proof*⟩

lemma *le-right-mono-restriction-process$_{ptick}$* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq P \downarrow m$›

⟨*proof*⟩

lemma *le-right-mono-restriction-process*$_{ptick}$*-FD* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{FD} P \downarrow m$›
⟨*proof*⟩

lemma *le-right-mono-restriction-process*$_{ptick}$*-F* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{F} P \downarrow m$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-le-right-mono-D* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{D} P \downarrow m$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-le-right-mono-T* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{T} P \downarrow m$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-le-right-mono-DT* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{DT} P \downarrow m$›
⟨*proof*⟩

1.6.3 Interpretations of Refinements

lemma *ex-not-restriction-leD* : ‹$\exists\, n.\ \neg\ P \downarrow n \sqsubseteq_{D} Q \downarrow n$› **if** ‹$\neg\ P \sqsubseteq_{D} Q$›
⟨*proof*⟩

interpretation *PRS-leF* : *PreorderRestrictionSpace* ‹($\downarrow$)› ‹($\sqsubseteq_F$)›
⟨*proof*⟩

interpretation *PRS-leT* : *PreorderRestrictionSpace* ‹($\downarrow$)› ‹($\sqsubseteq_T$)›
⟨*proof*⟩

interpretation *PRS-leDT* : *PreorderRestrictionSpace* ‹($\downarrow$)› ‹($\sqsubseteq_{DT}$)›
⟨*proof*⟩

1.7 Continuity

context begin

private lemma *chain-restriction-process*$_{ptick}$: ‹*chain* $Y \Longrightarrow$ *chain* $(\lambda i.\ Y\ i \downarrow n)$›
⟨*proof*⟩ **lemma** *cont-prem-restriction-process*$_{ptick}$:
‹$(\bigsqcup\ i.\ Y\ i) \downarrow n = (\bigsqcup\ i.\ Y\ i \downarrow n)$› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*chain* Y›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-cont* [*simp*] : ‹*cont* ($\lambda x.$ *f x* ↓ *n*)› **if** ‹*cont f*›
⟨*proof*⟩

end

1.8 Completeness

Processes are actually an instance of *complete-restriction-space*.

lemma *chain-restriction-chain* :
‹*restriction-chain* $\sigma \Longrightarrow$ *chain* σ› **for** σ :: ‹*nat* ⇒ ('*a*, '*r*) *process*$_{ptick}$›
⟨*proof*⟩

lemma *restricted-LUB-restriction-chain-is* :
‹($\lambda n.$ ($\bigsqcup n.$ σ *n*) ↓ *n*) = σ› **if** ‹*restriction-chain* σ›
⟨*proof*⟩

instance *process*$_{ptick}$ :: (*type*, *type*) *complete-restriction-space*
⟨*proof*⟩

This is a very powerful result. Now we can write fixed-point equations for processes like υ *X*. *f X*, providing the fact that *f* is *constructive*.

⟨*ML*⟩

2 Constructiveness of Prefixes

2.1 Equality

lemma *restriction-process*$_{ptick}$*-Mprefix* :
‹□*a* ∈ *A* → *P a* ↓ *n* = (*case n of 0* ⇒ ⊥ | *Suc m* ⇒ □*a*∈*A* → (*P a* ↓ *m*))› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-Mndetprefix* :
‹⊓*a* ∈ *A* → *P a* ↓ *n* = (*case n of 0* ⇒ ⊥ | *Suc m* ⇒ ⊓*a*∈*A* → (*P a* ↓ *m*))› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

corollary *restriction-process*$_{ptick}$*-write0* :
‹*a* → *P* ↓ *n* = (*case n of 0* ⇒ ⊥ | *Suc m* ⇒ *a* → (*P* ↓ *m*))›
⟨*proof*⟩

corollary *restriction-process$_{ptick}$-write* :
‹$c!a \to P \downarrow n = (case\ n\ of\ 0 \Rightarrow \bot \mid Suc\ m \Rightarrow c!a \to (P \downarrow m))$›
⟨*proof*⟩

corollary *restriction-process$_{ptick}$-read* :
‹$c?a{\in}A \to P\ a \downarrow n = (case\ n\ of\ 0 \Rightarrow \bot \mid Suc\ m \Rightarrow c?a{\in}A \to (P\ a \downarrow m))$›
⟨*proof*⟩

corollary *restriction-process$_{ptick}$-ndet-write* :
‹$c!!a{\in}A \to P\ a \downarrow n = (case\ n\ of\ 0 \Rightarrow \bot \mid Suc\ m \Rightarrow c!!a{\in}A \to (P\ a \downarrow m))$›
⟨*proof*⟩

2.2 Constructiveness

lemma *Mprefix-constructive* : ‹$constructive\ (\lambda P.\ \Box a \in A \to P\ a)$›
⟨*proof*⟩

lemma *Mndetprefix-constructive* : ‹$constructive\ (\lambda P.\ \sqcap a \in A \to P\ a)$›
⟨*proof*⟩

lemma *write0-constructive* : ‹$constructive\ (\lambda P.\ a \to P)$›
⟨*proof*⟩

lemma *write-constructive* : ‹$constructive\ (\lambda P.\ c!a \to P)$›
⟨*proof*⟩

lemma *read-constructive* : ‹$constructive\ (\lambda P.\ c?a \in A \to P\ a)$›
⟨*proof*⟩

lemma *ndet-write-constructive* : ‹$constructive\ (\lambda P.\ c!!a \in A \to P\ a)$›
⟨*proof*⟩

3 Non Destructiveness of Choices

3.1 Equality

lemma *restriction-process$_{ptick}$-Ndet* : ‹$P \sqcap Q \downarrow n = (P \downarrow n) \sqcap (Q \downarrow n)$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-GlobalNdet* :
‹$(\sqcap a \in A.\ P\ a) \downarrow n = (if\ n = 0\ then\ \bot\ else\ \sqcap a \in A.\ (P\ a \downarrow n))$›
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-GlobalDet* :
‹$(\Box a \in A.\ P\ a) \downarrow n = (\mathit{if}\ n = 0\ \mathit{then}\ \bot\ \mathit{else}\ \Box a \in A.\ (P\ a \downarrow n))$›
(**is** ‹$?lhs = (\mathit{if}\ n = 0\ \mathit{then}\ \bot\ \mathit{else}\ ?rhs)$›)
⟨*proof*⟩

lemma *restriction-process*$_{ptick}$*-Det*: ‹$P \Box Q \downarrow n = (P \downarrow n) \Box (Q \downarrow n)$› (**is** ‹$?lhs = ?rhs$›)
⟨*proof*⟩

corollary *restriction-process*$_{ptick}$*-Sliding*: ‹$P \rhd Q \downarrow n = (P \downarrow n) \rhd (Q \downarrow n)$›
⟨*proof*⟩

3.2 Non Destructiveness

lemma *GlobalNdet-non-destructive* : ‹*non-destructive* $(\lambda P.\ \sqcap a \in A.\ P\ a)$›
⟨*proof*⟩

lemma *Ndet-non-destructive* : ‹*non-destructive* $(\lambda(P, Q).\ P \sqcap Q)$›
⟨*proof*⟩

lemma *GlobalDet-non-destructive* : ‹*non-destructive* $(\lambda P.\ \Box a \in A.\ P\ a)$›
⟨*proof*⟩

lemma *Det-non-destructive* : ‹*non-destructive* $(\lambda(P, Q).\ P \Box Q)$›
⟨*proof*⟩

corollary *Sliding-non-destructive* : ‹*non-destructive* $(\lambda(P, Q).\ P \rhd Q)$›
⟨*proof*⟩

4 Non Destructiveness of Renaming

4.1 Equality

lemma *restriction-process*$_{ptick}$*-Renaming*:
‹*Renaming* $P\ f\ g \downarrow n =$ *Renaming* $(P \downarrow n)\ f\ g$› (**is** ‹$?lhs = ?rhs$›)
⟨*proof*⟩

4.2 Non Destructiveness

lemma *Renaming-non-destructive* [*simp*] :
‹*non-destructive* $(\lambda P.$ *Renaming* $P\ f\ g)$›
⟨*proof*⟩

5 Non Destructiveness of Sequential Composition

5.1 Refinement

lemma *restriction-process$_{ptick}$-Seq-FD* :
‹P **;** $Q \downarrow n \sqsubseteq_{FD} (P \downarrow n)$ **;** $(Q \downarrow n)$› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
⟨*proof*⟩

corollary *restriction-process$_{ptick}$-MultiSeq-FD* :
‹$(SEQ\ l \in@ L.\ P\ l) \downarrow n \sqsubseteq_{FD} SEQ\ l \in@ L.\ (P\ l \downarrow n)$›
⟨*proof*⟩

5.2 Non Destructiveness

lemma *Seq-non-destructive* :
‹*non-destructive* $(\lambda(P :: ('a, 'r)\ process_{ptick}, Q).\ P$ **;** $Q)$›
⟨*proof*⟩

6 Non Destructiveness of Synchronization Product

6.1 Preliminaries

lemma *D-Sync-optimized* :
‹$\mathcal{D}\ (P\ [\![A]\!]\ Q) =$
$\{v\ @\ w\ |t\ u\ v\ w.\ tF\ v \land ftF\ w \land$
$v\ setinterleaves\ ((t,\ u),\ range\ tick \cup ev\ `\ A) \land$
$(t \in \mathcal{D}\ P \land u \in \mathcal{T}\ Q \lor t \in \mathcal{D}\ Q \land u \in \mathcal{T}\ P)\}$›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

lemma *tickFree-interleave-iff* :
‹$t\ setinterleaves\ ((u,\ v),\ S) \Longrightarrow tF\ t \longleftrightarrow tF\ u \land tF\ v$›
⟨*proof*⟩

lemma *interleave-subsetL* :
‹$tF\ t \Longrightarrow \{a.\ ev\ a \in set\ u\} \subseteq A \Longrightarrow$
$t\ setinterleaves\ ((u,\ v),\ range\ tick \cup ev\ `\ A) \Longrightarrow t = v$›
for $t\ u\ v$ :: ‹$('a,\ 'r)\ trace_{ptick}$›
⟨*proof*⟩

lemma *interleave-subsetR* :
‹$tF\ t \Longrightarrow \{a.\ ev\ a \in set\ v\} \subseteq A \Longrightarrow$
$t\ setinterleaves\ ((u,\ v),\ range\ tick \cup ev\ `\ A) \Longrightarrow t = u$›
⟨*proof*⟩

lemma *interleave-imp-lengthLR-le* :
‹t *setinterleaves* $((u,\ v),\ S) \Longrightarrow$
$\ $ *length* $u \leq$ *length* $t \wedge$ *length* $v \leq$ *length* t›
⟨*proof*⟩

lemma *interleave-le-prefixLR* :
‹t *setinterleaves* $((u,\ v),\ S) \Longrightarrow u' \leq u \Longrightarrow v' \leq v \Longrightarrow$
$(\exists\, t' \leq t.\ \exists\, v'' \leq v'.\ t'$ *setinterleaves* $((u',\ v''),\ S)) \vee$
$(\exists\, t' \leq t.\ \exists\, u'' \leq u'.\ t'$ *setinterleaves* $((u'',\ v'),\ S))$›
⟨*proof*⟩

lemma *restriction-process$_{ptick}$-Sync-FD-div-oneside* :
assumes ‹*tF* u› ‹*ftF* v› ‹$t\text{-}P \in \mathcal{D}\ (P \downarrow n)$› ‹$t\text{-}Q \in \mathcal{T}\ (Q \downarrow n)$›
‹u *setinterleaves* $((t\text{-}P,\ t\text{-}Q),$ *range tick* $\cup$ *ev* ‘ $A)$›
shows ‹$u\ @\ v \in \mathcal{D}\ (P\ [\![A]\!]\ Q \downarrow n)$›
⟨*proof*⟩

6.2 Refinement

lemma *restriction-process$_{ptick}$-Sync-FD* :
‹$P\ [\![A]\!]\ Q \downarrow n \sqsubseteq_{FD} (P \downarrow n)\ [\![A]\!]\ (Q \downarrow n)$› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
⟨*proof*⟩

The equality does not hold in general, but we can establish it by adding an assumption over the strict alphabets of the processes.

lemma *strict-events-of-subset-restriction-process$_{ptick}$-Sync* :
‹$P\ [\![A]\!]\ Q \downarrow n = (P \downarrow n)\ [\![A]\!]\ (Q \downarrow n)$› (**is** ‹*?lhs* = *?rhs*›)
if ‹$\boldsymbol{\alpha}(P) \subseteq A \vee \boldsymbol{\alpha}(Q) \subseteq A$›
⟨*proof*⟩

corollary *restriction-process$_{ptick}$-MultiSync-FD* :
‹$[\![A]\!]\ m \in\#\ M.\ P\ l \downarrow n \sqsubseteq_{FD} [\![A]\!]\ m \in\#\ M.\ (P\ l \downarrow n)$›
⟨*proof*⟩

In the following corollary, we could be more precise by having the condition on at least *size* $M - 1$ processes.

corollary *strict-events-of-subset-restriction-process$_{ptick}$-MultiSync* :
‹$[\![A]\!]\ m \in\#\ M.\ P\ m \downarrow n =$ (*if* $n = 0$ *then* $\bot$ *else* $[\![A]\!]\ m \in\#\ M.\ (P\ m \downarrow n))$›
— *if* $n = 0$ *then* $\bot$ *else* - is necessary because we can have $M = \{\#\}$.
if ‹$\bigwedge m.\ m \in\#\ M \Longrightarrow \boldsymbol{\alpha}(P\ m) \subseteq A$›
⟨*proof*⟩

corollary *restriction-process$_{ptick}$-Par* :
‹$P\ ||\ Q \downarrow n = (P \downarrow n)\ ||\ (Q \downarrow n)$›

⟨*proof*⟩

corollary *restriction-process$_{ptick}$-MultiPar* :
‹$\|$ $m \in\# M$. $P\ l \downarrow n$ = (*if* $n = 0$ *then* $\bot$ *else* $\|$ $m \in\# M$. ($P\ l \downarrow n$))›
⟨*proof*⟩

6.3 Non Destructiveness

lemma *Sync-non-destructive* :
‹*non-destructive* ($\lambda(P, Q)$. P $[\![A]\!]$ Q)›
⟨*proof*⟩

end

7 Non Destructiveness of Throw

7.1 Equality

lemma *Depth-Throw-1-is-constant*: ‹$P\ \Theta\ a \in A$. $Q1\ a \downarrow 1 = P\ \Theta\ a \in A$. $Q2\ a \downarrow 1$›
⟨*proof*⟩

7.2 Refinement

lemma *restriction-process$_{ptick}$-Throw-FD* :
‹($P\ \Theta\ a \in A$. $Q\ a$) $\downarrow n \sqsubseteq_{FD}$ ($P \downarrow n$) $\Theta\ a \in A$. ($Q\ a \downarrow n$)› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
⟨*proof*⟩

7.3 Non Destructiveness

lemma *Throw-non-destructive* :
‹*non-destructive* (λ(P :: ('a, 'r) *process$_{ptick}$*, Q). $P\ \Theta\ a \in A$. $Q\ a$)›
⟨*proof*⟩

lemma *ThrowR-constructive-if-disjoint-initials* :
‹*constructive* (λQ :: 'a $\Rightarrow$ ('a, 'r) *process$_{ptick}$*. $P\ \Theta\ a \in A$. $Q\ a$)›
if ‹$A \cap \{e.\ ev\ e \in P^0\} = \{\}$›
⟨*proof*⟩

8 Non Destructiveness of Interrupt

8.1 Refinement

lemma *restriction-process$_{ptick}$-Interrupt-FD* :
‹$P \triangle Q \downarrow n \sqsubseteq_{FD} (P \downarrow n) \triangle (Q \downarrow n)$› (**is** ‹?lhs $\sqsubseteq_{FD}$?rhs›)
⟨*proof*⟩

8.2 Non Destructiveness

lemma *Interrupt-non-destructive* :
‹*non-destructive* $(\lambda(P :: ('a, 'r)\ process_{ptick}, Q).\ P \triangle Q)$›
⟨*proof*⟩

9 Non too Destructiveness of After

9.1 Equality

lemma *initials-restriction-process$_{ptick}$*: ‹$(P \downarrow n)^0 = (if\ n = 0\ then\ UNIV\ else\ P^0)$›
⟨*proof*⟩

lemma (**in** *After*) *restriction-process$_{ptick}$-After*:
‹$P\ after\ e \downarrow n = (if\ ev\ e \in P^0\ then\ (P \downarrow Suc\ n)\ after\ e\ else\ \Psi\ P\ e \downarrow n)$›
⟨*proof*⟩

lemma (**in** *AfterExt*) *restriction-process$_{ptick}$-After$_{tick}$*:
‹$P\ after_{\checkmark}\ e \downarrow n =$
$(case\ e\ of\ \checkmark(r) \Rightarrow \Omega\ P\ r \downarrow n \mid ev\ x \Rightarrow if\ e \in P^0\ then\ (P \downarrow Suc\ n)\ after_{\checkmark}\ e\ else\ \Psi\ P\ x \downarrow n)$›
⟨*proof*⟩

lemma (**in** *AfterExt*) *restriction-process$_{ptick}$-After$_{trace}$*:
‹$t \in \mathcal{T}\ P \Longrightarrow tF\ t \Longrightarrow P\ after_{\mathcal{T}}\ t \downarrow n = (P \downarrow (n + length\ t))\ after_{\mathcal{T}}\ t$›
⟨*proof*⟩

9.2 Non too Destructiveness

lemma (**in** *After*) *non-too-destructive-on-After* :
‹*non-too-destructive-on* $(\lambda P.\ P\ after\ e)\ \{P.\ ev\ e \in P^0\}$›
⟨*proof*⟩

lemma (**in** *AfterExt*) *non-too-destructive-on-After$_{tick}$* :
‹*non-too-destructive-on* $(\lambda P.\ P\ after_{\checkmark}\ e)\ \{P.\ e \in P^0\}$›
if ‹$\bigwedge r.\ e = \checkmark(r) \Longrightarrow$ *non-too-destructive-on* $\Omega\ \{P.\ \checkmark(r) \in P^0\}$›
⟨*proof*⟩

lemma (**in** *After*) *non-too-destructive-After* :
‹*non-too-destructive* ($\lambda P.\ P$ *after* e)› **if** $*$: ‹*non-too-destructive-on* Ψ $\{P.\ ev\ e \notin P^0\}$›
⟨*proof*⟩

lemma (**in** *AfterExt*) *non-too-destructive-After*$_{tick}$:
‹*non-too-destructive* ($\lambda P.\ P$ *after*$_{\checkmark}$ e)›
if $*$: ‹$\bigwedge a.\ e = ev\ a \Longrightarrow$ *non-too-destructive-on* Ψ $\{P.\ ev\ a \notin P^0\}$›
‹$\bigwedge r.\ e = \checkmark(r) \Longrightarrow$ *non-too-destructive* ($\lambda P.\ \Omega\ P\ r$)›
⟨*proof*⟩

lemma (**in** *AfterExt*) *restriction-shift-After*$_{trace}$:
‹*restriction-shift* ($\lambda P.\ P$ *after*$_{\mathcal{T}}$ t) ($-$ *int* (*length* t))›
if ‹*non-too-destructive* Ψ› ‹*non-too-destructive* Ω›
— We could imagine more precise assumptions, but is it useful?
⟨*proof*⟩

10 Destructiveness of Hiding

theory *Hiding-Destructive*
imports *HOL−CSPM.CSPM-Laws Prefixes-Constructive*
begin

10.1 Refinement

lemma *Hiding-restriction-process*$_{ptick}$*-FD* : ‹$(P \downarrow n) \setminus S \sqsubseteq_{FD} P \setminus S \downarrow n$›
⟨*proof*⟩

10.2 Destructiveness

lemma *Hiding-destructive* :
‹$\exists\, P\ Q$ :: ($'a$, $'r$) *process*$_{ptick}$. $P \downarrow n = Q \downarrow n \wedge (P \setminus S) \downarrow Suc\ 0 \neq (Q \setminus S) \downarrow Suc\ 0$› **if** ‹$S \neq \{\}$›
⟨*proof*⟩

11 Admissibility

named-theorems *restriction-adm-process*$_{ptick}$*-simpset*

11.1 Belonging

lemma *restriction-adm-in-D* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.$ $t \in \mathcal{D}$ (f x))›
and *restriction-adm-notin-D* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.$ $t \notin \mathcal{D}$ (f x))›
and *restriction-adm-in-F* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.$ (t, X) $\in \mathcal{F}$ (f x))›
and *restriction-adm-notin-F* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.$ (t, X) $\notin \mathcal{F}$ (f x))› **if** ‹$cont_{\downarrow}$ f›
for f :: ‹$'b$ :: *restriction* $\Rightarrow$ ($'a$, $'r$) *process*$_{ptick}$›
⟨*proof*⟩

corollary *restriction-adm-in-T* [*restriction-adm-process*$_{ptick}$*-simpset*]
:
‹$cont_{\downarrow}$ $f \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ $t \in \mathcal{T}$ (f x))›
and *restriction-adm-notin-T* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$cont_{\downarrow}$ $f \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ $t \notin \mathcal{T}$ (f x))›
⟨*proof*⟩

corollary *restriction-adm-in-initials* [*restriction-adm-process*$_{ptick}$*-simpset*]
:
‹$cont_{\downarrow}$ $f \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ $e \in (f\ x)^0$)›
and *restriction-adm-notin-initials* [*restriction-adm-process*$_{ptick}$*-simpset*]
:
‹$cont_{\downarrow}$ $f \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ $e \notin (f\ x)^0$)›
⟨*proof*⟩

11.2 Refining

corollary *restriction-adm-leF* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$cont_{\downarrow}$ $f \Longrightarrow cont_{\downarrow}$ $g \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ f $x \sqsubseteq_F g$ x)›
⟨*proof*⟩

corollary *restriction-adm-leD* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$cont_{\downarrow}$ $f \Longrightarrow cont_{\downarrow}$ $g \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ f $x \sqsubseteq_D g$ x)›
⟨*proof*⟩

corollary *restriction-adm-leT* [*restriction-adm-process*$_{ptick}$*-simpset*] :
‹$cont_{\downarrow}$ $f \Longrightarrow cont_{\downarrow}$ $g \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ f $x \sqsubseteq_T g$ x)›
⟨*proof*⟩

corollary *restriction-adm-leFD* [*restriction-adm-process*$_{ptick}$*-simpset*]
:
‹$cont_{\downarrow}$ $f \Longrightarrow cont_{\downarrow}$ $g \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ f $x \sqsubseteq_{FD} g$ x)›
⟨*proof*⟩

corollary *restriction-adm-leDT* [*restriction-adm-process*$_{ptick}$*-simpset*]

:
$\langle cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}\ (\lambda x.\ f\ x \sqsubseteq_{DT}\ g\ x)\rangle$
$\langle proof\rangle$

11.2.1 Transitions

lemma (**in** *After*) *restriction-cont-After* [*restriction-adm-simpset*] :
$\langle cont_{\downarrow}\ (\lambda x.\ f\ x\ after\ a)\rangle$ **if** $\langle cont_{\downarrow}\ f\rangle$ **and** $\langle cont_{\downarrow}\ \Psi\rangle$
— We could imagine more precise assumptions, but is it useful?
$\langle proof\rangle$

lemma (**in** *AfterExt*) *restriction-cont-After$_{tick}$* [*restriction-adm-simpset*]
:
$\langle cont_{\downarrow}\ (\lambda x.\ f\ x\ after_{\checkmark}\ e)\rangle$ **if** $\langle cont_{\downarrow}\ f\rangle$ **and** $\langle cont_{\downarrow}\ \Psi\rangle$ **and** $\langle cont_{\downarrow}\ \Omega\rangle$
— We could imagine more precise assumptions, but is it useful?
$\langle proof\rangle$

lemma (**in** *AfterExt*) *restriction-cont-After$_{trace}$* [*restriction-adm-simpset*]
:
$\langle cont_{\downarrow}\ (\lambda x.\ f\ x\ after_{\mathcal{T}}\ t)\rangle$ **if** $\langle cont_{\downarrow}\ f\rangle$ **and** $\langle cont_{\downarrow}\ \Psi\rangle$ **and** $\langle cont_{\downarrow}\ \Omega\rangle$
— We could imagine more precise assumptions, but is it useful?
$\langle proof\rangle$

lemma (**in** *OpSemTransitions*) *restriction-adm-weak-ev-trans* [*restriction-adm-process$_{ptick}$-simpset*]:
— Could be weakened to a continuity assumption on Ψ.
fixes $f\ g :: \langle 'b :: restriction \Rightarrow ('a, 'r)\ process_{ptick}\rangle$
assumes *τ-trans-restriction-adm*:
$\langle \bigwedge f\ g :: 'b \Rightarrow ('a, 'r)\ process_{ptick}.\ cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$
$(\lambda x.\ f\ x \leadsto_{\tau}\ g\ x)\rangle$
and $\langle cont_{\downarrow}\ f\rangle$ **and** $\langle cont_{\downarrow}\ g\rangle$ **and** $\langle cont_{\downarrow}\ \Psi\rangle$ **and** $\langle cont_{\downarrow}\ \Omega\rangle$
shows $\langle adm_{\downarrow}\ (\lambda x.\ f\ x \leadsto_{e}\ g\ x)\rangle$
$\langle proof\rangle$

lemma (**in** *OpSemTransitions*) *restriction-adm-weak-tick-trans* [*restriction-adm-process$_{ptick}$-simpset*]:
fixes $f\ g :: \langle 'b :: restriction \Rightarrow ('a, 'r)\ process_{ptick}\rangle$
assumes *τ-trans-restriction-adm*:
$\langle \bigwedge f\ g :: 'b \Rightarrow ('a, 'r)\ process_{ptick}.\ cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$
$(\lambda x.\ f\ x \leadsto_{\tau}\ g\ x)\rangle$
and $\langle cont_{\downarrow}\ f\rangle$ **and** $\langle cont_{\downarrow}\ g\rangle$ **and** $\langle cont_{\downarrow}\ \Psi\rangle$ **and** $\langle cont_{\downarrow}\ \Omega\rangle$
shows $\langle adm_{\downarrow}\ (\lambda x.\ f\ x \leadsto_{\checkmark r}\ (g\ x))\rangle$
$\langle proof\rangle$

lemma (**in** *OpSemTransitions*) *restriction-adm-weak-trace-trans* [*restriction-adm-process$_{ptick}$-simpset*]:
fixes $f\ g :: \langle 'b :: restriction \Rightarrow ('a, 'r)\ process_{ptick}\rangle$
assumes *τ-trans-restriction-adm*:
$\langle \bigwedge f\ g :: 'b \Rightarrow ('a, 'r)\ process_{ptick}.\ cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$
$(\lambda x.\ f\ x \leadsto_{\tau}\ g\ x)\rangle$
and $\langle cont_{\downarrow}\ f\rangle$ **and** $\langle cont_{\downarrow}\ g\rangle$ **and** $\langle cont_{\downarrow}\ \Psi\rangle$ **and** $\langle cont_{\downarrow}\ \Omega\rangle$

shows ‹$adm_{\downarrow}$ ($\lambda x.\ f\ x \rightsquigarrow^{*}\ t\ (g\ x)$)›
⟨*proof*⟩

declare *restriction-adm-process*$_{ptick}$*-simpset* [*simp*]

12 Higher-Order Rules

This is the main entry point. We configure the simplifier below.

named-theorems *restriction-shift-process*$_{ptick}$*-simpset*

12.1 Prefixes

lemma *Mprefix-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*constructive* ($\lambda x.\ \square a \in A \to f\ a\ x$)› **if** ‹($\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive*
($f\ a$))›
⟨*proof*⟩

lemma *Mndetprefix-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*constructive* ($\lambda x.\ \sqcap a \in A \to f\ a\ x$)› **if** ‹($\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive*
($f\ a$))›
⟨*proof*⟩

corollary *write0-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *constructive* ($\lambda x.\ a \to f\ x$)›
⟨*proof*⟩

corollary *write-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *constructive* ($\lambda x.\ c\mathbf{!}a \to f\ x$)›
⟨*proof*⟩

corollary *read-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹($\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* ($f\ a$)) $\Longrightarrow$ *constructive* ($\lambda x.\ c\mathbf{?}a \in$
$A \to f\ a\ x$)›
⟨*proof*⟩

corollary *ndet-write-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹($\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* ($f\ a$)) $\Longrightarrow$ *constructive* ($\lambda x.\ c\mathbf{!!}a \in$
$A \to f\ a\ x$)›

$\langle proof \rangle$

12.2 Choices

lemma *GlobalNdet-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* $(f\ a)) \Longrightarrow$ *non-destructive* $(\lambda x.\ \sqcap a$
$\in A.\ f\ a\ x)$›
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *constructive* $(f\ a)) \Longrightarrow$ *constructive* $(\lambda x.\ \sqcap a \in A.$
$f\ a\ x)$›
$\langle proof \rangle$

lemma *GlobalDet-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* $(f\ a)) \Longrightarrow$ *non-destructive* $(\lambda x.\ \square a$
$\in A.\ f\ a\ x)$›
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *constructive* $(f\ a)) \Longrightarrow$ *constructive* $(\lambda x.\ \square a \in A.$
$f\ a\ x)$›
$\langle proof \rangle$

lemma *Ndet-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *non-destructive* $g \Longrightarrow$ *non-destructive* $(\lambda x.\ f$
$x \sqcap g\ x)$›
‹*constructive* $f \Longrightarrow$ *constructive* $g \Longrightarrow$ *constructive* $(\lambda x.\ f\ x \sqcap g\ x)$›
$\langle proof \rangle$

lemma *Det-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *non-destructive* $g \Longrightarrow$ *non-destructive* $(\lambda x.\ f$
$x \square g\ x)$›
‹*constructive* $f \Longrightarrow$ *constructive* $g \Longrightarrow$ *constructive* $(\lambda x.\ f\ x \square g\ x)$›
$\langle proof \rangle$

lemma *Sliding-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *non-destructive* $g \Longrightarrow$ *non-destructive* $(\lambda x.\ f$
$x \rhd g\ x)$›
‹*constructive* $f \Longrightarrow$ *constructive* $g \Longrightarrow$ *constructive* $(\lambda x.\ f\ x \rhd g\ x)$›
$\langle proof \rangle$

12.3 Renaming

lemma *Renaming-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $P \Longrightarrow$ *non-destructive* $(\lambda x.$ *Renaming* $(P\ x)\ f\ g)$›
‹*constructive* $P \Longrightarrow$ *constructive* $(\lambda x.$ *Renaming* $(P\ x)\ f\ g)$›
$\langle proof \rangle$

12.4 Sequential Composition

lemma *Seq-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
  ‹*non-destructive f* $\Longrightarrow$ *non-destructive g* $\Longrightarrow$ *non-destructive* (λx. *f*
x **;** *g x*)›
  ‹*constructive f* $\Longrightarrow$ *constructive g* $\Longrightarrow$ *constructive* (λx. *f x* **;** *g x*)›
  ⟨*proof*⟩

lemma *MultiSeq-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
  ‹($\bigwedge l$. $l \in$ *set L* $\Longrightarrow$ *non-destructive* (*f l*)) $\Longrightarrow$ *non-destructive* (λx.
SEQ l ∈@ *L*. *f l x*)›
  ‹($\bigwedge l$. $l \in$ *set L* $\Longrightarrow$ *constructive* (*f l*)) $\Longrightarrow$ *constructive* (λx. *SEQ l*
∈@ *L*. *f l x*)›
  ⟨*proof*⟩

corollary *MultiSeq-non-destructive* : ‹*non-destructive* (λP. *SEQ l* ∈@
L. *P l*)›
  ⟨*proof*⟩

12.5 Synchronization Product

lemma *Sync-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
  ‹*non-destructive f* $\Longrightarrow$ *non-destructive g* $\Longrightarrow$ *non-destructive* (λx. *f*
x ⟦*S*⟧ *g x*)›
  ‹*constructive f* $\Longrightarrow$ *constructive g* $\Longrightarrow$ *constructive* (λx. *f x* ⟦*S*⟧ *g x*)›
  ⟨*proof*⟩

lemma *MultiSync-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
  ‹($\bigwedge m$. $m \in\#$ *M* $\Longrightarrow$ *non-destructive* (*f m*)) $\Longrightarrow$ *non-destructive* (λx.
⟦*S*⟧ *m* ∈# *M*. *f m x*)›
  ‹($\bigwedge m$. $m \in\#$ *M* $\Longrightarrow$ *constructive* (*f m*)) $\Longrightarrow$ *constructive* (λx. ⟦*S*⟧
m ∈# *M*. *f m x*)›
  ⟨*proof*⟩

corollary *MultiSync-non-destructive* : ‹*non-destructive* (λP. ⟦*S*⟧ *m*
∈# *M*. *P m*)›
  ⟨*proof*⟩

12.6 Throw

lemma *Throw-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:

‹$non\text{-}destructive\ f \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow non\text{-}destructive\ (g\ a)) \Longrightarrow$
$non\text{-}destructive\ (\lambda x.\ f\ x\ \Theta\ a \in A.\ g\ a\ x)$›
‹$constructive\ f \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow constructive\ (g\ a)) \Longrightarrow construc$-
$tive\ (\lambda x.\ f\ x\ \Theta\ a \in A.\ g\ a\ x)$›
⟨*proof*⟩

12.7 Interrupt

lemma *Interrupt-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹$non\text{-}destructive\ f \Longrightarrow non\text{-}destructive\ g \Longrightarrow non\text{-}destructive\ (\lambda x.\ f$
$x \bigtriangleup g\ x)$›
‹$constructive\ f \Longrightarrow constructive\ g \Longrightarrow constructive\ (\lambda x.\ f\ x \bigtriangleup g\ x)$›
⟨*proof*⟩

12.8 After

lemma (**in** *After*) *After-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹$non\text{-}too\text{-}destructive\ \Psi \Longrightarrow constructive\ f \Longrightarrow non\text{-}destructive\ (\lambda x.$
$f\ x\ \ after\ a)$›
‹$non\text{-}too\text{-}destructive\ \Psi \Longrightarrow non\text{-}destructive\ f \Longrightarrow non\text{-}too\text{-}destructive$
$(\lambda x.\ f\ x\ \ after\ a)$›
⟨*proof*⟩

lemma (**in** *AfterExt*) *After*$_{tick}$*-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹$[\![non\text{-}too\text{-}destructive\ \Psi;\ non\text{-}too\text{-}destructive\ \Omega;\ constructive\ f]\!]$
$\Longrightarrow non\text{-}destructive\ (\lambda x.\ f\ x\ \ after_{\checkmark}\ e)$›
‹$[\![non\text{-}too\text{-}destructive\ \Psi;\ non\text{-}too\text{-}destructive\ \Omega;\ non\text{-}destructive\ f]\!]$
$\Longrightarrow non\text{-}too\text{-}destructive\ (\lambda x.\ f\ x\ \ after_{\checkmark}\ e)$›
⟨*proof*⟩

12.9 Illustration

declare *restriction-shift-process*$_{ptick}$*-simpset* [*simp*]

notepad begin
⟨*proof*⟩

end

end