

HOL-CSP_RS: CSP Semantics over Restriction Spaces

Benoît Ballenghien Burkhart Wolff

June 15, 2026

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
1.1	Definition	1
1.2	Projections	1
1.3	Proof obligation	4
1.4	Compatibility with Refinements	4
1.5	First Laws	4
1.6	Monotony	5
1.6.1	$P \downarrow n$ is an Approximation of the P	5
1.6.2	Monotony of (\downarrow)	6
1.6.3	Interpretations of Refinements	7
1.7	Continuity	7
1.8	Completeness	7
2	Constructiveness of Prefixes	8
2.1	Equality	8
2.2	Constructiveness	9
3	Non Destructiveness of Choices	9
3.1	Equality	9
3.2	Non Destructiveness	10

4	Non Destructiveness of Renaming	10
4.1	Equality	10
4.2	Non Destructiveness	10
5	Non Destructiveness of Sequential Composition	10
5.1	Refinement	10
5.2	Non Destructiveness	11
6	Non Destructiveness of Synchronization Product	11
6.1	Preliminaries	11
6.2	Refinement	12
6.3	Non Destructiveness	12
7	Non Destructiveness of Throw	13
7.1	Equality	13
7.2	Refinement	13
7.3	Non Destructiveness	13
8	Non Destructiveness of Interrupt	13
8.1	Refinement	13
8.2	Non Destructiveness	13
9	Non too Destructiveness of After	14
9.1	Equality	14
9.2	Non too Destructiveness	14
10	Destructiveness of Hiding	15
10.1	Refinement	15
10.2	Destructiveness	15
11	Admissibility	15
11.1	Belonging	15
11.2	Refining	16
11.2.1	Transitions	16
12	Higher-Order Rules	17
12.1	Prefixes	18
12.2	Choices	18
12.3	Renaming	19
12.4	Sequential Composition	19
12.5	Synchronization Product	20
12.6	Throw	20
12.7	Interrupt	20
12.8	After	21
12.9	Illustration	21

1 Depth Operator

1.1 Definition

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

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

1.2 Projections

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

lemma $F\text{-restriction-process}_{ptick} :$
 $\langle \mathcal{F} (P \downarrow n) = \mathcal{F} P \cup \{(t @ u, X) \mid t u X. t \in \mathcal{T} P \wedge \text{length } t = n \wedge$
 $tF t \wedge ftF u\} \rangle$
 $\langle proof \rangle$

lemma $D\text{-restriction-process}_{ptick} :$
 $\langle \mathcal{D} (P \downarrow n) = \mathcal{D} P \cup \{t @ u \mid t u. t \in \mathcal{T} P \wedge \text{length } t = n \wedge tF t \wedge$
 $ftF u\} \rangle$
 $\langle proof \rangle$

lemma $T\text{-restriction-process}_{ptick} :$
 $\langle \mathcal{T} (P \downarrow n) = \mathcal{T} P \cup \{t @ u \mid t u. t \in \mathcal{T} P \wedge \text{length } t = n \wedge tF t \wedge$
 $ftF u\} \rangle$
 $\langle proof \rangle$

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

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

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

lemma *T-restriction-process_{ptick}E* :
 $\langle \llbracket t \in \mathcal{T} (P \downarrow n); t \in \mathcal{T} P \implies \text{length } t \leq n \implies \text{thesis};$
 $\bigwedge u v. t = u @ v \implies u \in \mathcal{T} P \implies \text{length } u = n \implies tF u \implies ftF$
 $v \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
 $\langle \text{proof} \rangle$

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* :
 $\langle t \in \mathcal{D} P \vee t \in \mathcal{T} P \wedge (\text{length } t = n \wedge tF t \vee n < \text{length } t) \implies t$
 $\in \mathcal{D} (P \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

lemma *F-restriction-process_{ptick}I* :
 $\langle (t, X) \in \mathcal{F} P \vee t \in \mathcal{T} P \wedge (\text{length } t = n \wedge tF t \vee n < \text{length } t)$
 $\implies (t, X) \in \mathcal{F} (P \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

lemma *T-restriction-process_{ptick}I* :
 $\langle t \in \mathcal{T} P \vee t \in \mathcal{T} P \wedge (\text{length } t = n \wedge tF t \vee n < \text{length } t) \implies t$
 $\in \mathcal{T} (P \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

lemma *F-restriction-process_{ptick}-Suc-length-iff-F* :
 $\langle (t, X) \in \mathcal{F} (P \downarrow \text{Suc } (\text{length } t)) \iff (t, X) \in \mathcal{F} P \rangle$
and *D-restriction-process_{ptick}-Suc-length-iff-D* :
 $\langle t \in \mathcal{D} (P \downarrow \text{Suc } (\text{length } t)) \iff t \in \mathcal{D} P \rangle$
and *T-restriction-process_{ptick}-Suc-length-iff-T* :
 $\langle t \in \mathcal{T} (P \downarrow \text{Suc } (\text{length } t)) \iff t \in \mathcal{T} P \rangle$
 $\langle \text{proof} \rangle$

lemma *length-less-in-F-restriction-process_{ptick}* :
 $\langle \text{length } t < n \implies (t, X) \in \mathcal{F} (P \downarrow n) \implies (t, X) \in \mathcal{F} P \rangle$
 $\langle \text{proof} \rangle$

lemma *length-le-in-T-restriction-process_{ptick}* :
 $\langle \text{length } t \leq n \implies t \in \mathcal{T} (P \downarrow n) \implies t \in \mathcal{T} P \rangle$
 $\langle \text{proof} \rangle$

lemma *length-less-in-D-restriction-process_{ptick}* :
 $\langle \text{length } t < n \implies t \in \mathcal{D} (P \downarrow n) \implies t \in \mathcal{D} P \rangle$

⟨proof⟩

lemma *not-tickFree-in-F-restriction-process_{ptick}-iff* :

⟨length $t \leq n \implies \neg tF t \implies (t, X) \in \mathcal{F} (P \downarrow n) \iff (t, X) \in \mathcal{F} P$ ⟩

⟨proof⟩

lemma *not-tickFree-in-D-restriction-process_{ptick}-iff* :

⟨length $t \leq n \implies \neg tF t \implies t \in \mathcal{D} (P \downarrow n) \iff t \in \mathcal{D} P$ ⟩

⟨proof⟩

end

lemma *front-tickFreeE* :

⟨ $\llbracket tF t; tF t \implies thesis; \bigwedge t' r. t = t' @ [\checkmark(r)] \implies tF t' \implies thesis \rrbracket \implies thesis$ ⟩

⟨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*: $\langle P \downarrow n \sqsubseteq_F Q \downarrow n \rangle$

if $\langle \bigwedge s X. (s, X) \in \mathcal{F} Q \implies length s \leq n \implies (s, X) \in \mathcal{F} (P \downarrow n) \rangle$

⟨proof⟩

lemma *leT-restriction-process_{ptick}I*: $\langle P \downarrow n \sqsubseteq_T Q \downarrow n \rangle$

if $\langle \bigwedge s. s \in \mathcal{T} Q \implies length s \leq n \implies s \in \mathcal{T} (P \downarrow n) \rangle$

⟨proof⟩

lemma *leDT-restriction-process_{ptick}I*: $\langle P \downarrow n \sqsubseteq_{DT} Q \downarrow n \rangle$

if $\langle \bigwedge s. s \in \mathcal{T} Q \implies length s \leq n \implies s \in \mathcal{T} (P \downarrow n) \rangle$

and $\langle \bigwedge s. \text{length } s \leq n \implies s \in \mathcal{D} \ Q \implies s \in \mathcal{D} \ (P \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

lemma *leFD-restriction-process_{ptick}I*: $\langle P \downarrow n \sqsubseteq_{FD} \ Q \downarrow n \rangle$
if $\langle \bigwedge s \ X. (s, X) \in \mathcal{F} \ Q \implies \text{length } s \leq n \implies (s, X) \in \mathcal{F} \ (P \downarrow n) \rangle$
and $\langle \bigwedge s. s \in \mathcal{D} \ Q \implies \text{length } s \leq n \implies s \in \mathcal{D} \ (P \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

1.5 First Laws

lemma *restriction-process_{ptick}-0* [*simp*]: $\langle P \downarrow 0 = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-BOT* [*simp*]: $\langle (\perp :: ('a, 'r) \text{process}_{ptick}) \downarrow n = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-is-BOT-iff* :
 $\langle P \downarrow n = \perp \longleftrightarrow n = 0 \vee P = \perp \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-STOP* [*simp*]: $\langle \text{STOP} \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \text{STOP}) \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-is-STOP-iff* : $\langle P \downarrow n = \text{STOP} \longleftrightarrow n \neq 0 \wedge P = \text{STOP} \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-SKIP* [*simp*]: $\langle \text{SKIP } r \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \text{SKIP } r) \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-is-SKIP-iff* : $\langle P \downarrow n = \text{SKIP } r \longleftrightarrow n \neq 0 \wedge P = \text{SKIP } r \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-SKIPS* [*simp*]: $\langle \text{SKIPS } R \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \text{SKIPS } R) \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-is-SKIPS-iff* : $\langle P \downarrow n = \text{SKIPS } R \longleftrightarrow n \neq 0 \wedge P = \text{SKIPS } R \rangle$
 $\langle \text{proof} \rangle$

1.6 Monotony

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

lemma *restriction-process_{ptick}-approx-self* : $\langle P \downarrow n \sqsubseteq P \rangle$
\langle proof \rangle

lemma *restriction-process_{ptick}-FD-self* : $\langle P \downarrow n \sqsubseteq_{FD} P \rangle$
\langle proof \rangle

lemma *restriction-process_{ptick}-F-self* : $\langle P \downarrow n \sqsubseteq_F P \rangle$
\langle proof \rangle

lemma *restriction-process_{ptick}-D-self* : $\langle P \downarrow n \sqsubseteq_D P \rangle$
\langle proof \rangle

lemma *restriction-process_{ptick}-T-self* : $\langle P \downarrow n \sqsubseteq_T P \rangle$
\langle proof \rangle

lemma *restriction-process_{ptick}-DT-self* : $\langle P \downarrow n \sqsubseteq_{DT} P \rangle$
\langle proof \rangle

1.6.2 Monotony of (\downarrow)

lemma *Suc-right-mono-restriction-process_{ptick}* : $\langle P \downarrow n \sqsubseteq P \downarrow \text{Suc } n \rangle$
\langle proof \rangle

lemma *Suc-right-mono-restriction-process_{ptick}-FD* : $\langle P \downarrow n \sqsubseteq_{FD} P \downarrow \text{Suc } n \rangle$
\langle proof \rangle

lemma *Suc-right-mono-restriction-process_{ptick}-F* : $\langle P \downarrow n \sqsubseteq_F P \downarrow \text{Suc } n \rangle$
\langle proof \rangle

lemma *Suc-right-mono-restriction-process_{ptick}-D* : $\langle P \downarrow n \sqsubseteq_D P \downarrow \text{Suc } n \rangle$
\langle proof \rangle

lemma *Suc-right-mono-restriction-process_{ptick}-T* : $\langle P \downarrow n \sqsubseteq_T P \downarrow \text{Suc } n \rangle$
\langle proof \rangle

lemma *Suc-right-mono-restriction-process_{ptick}-DT* : $\langle P \downarrow n \sqsubseteq_{DT} P \downarrow \text{Suc } n \rangle$
\langle proof \rangle

lemma *le-right-mono-restriction-process_{ptick}* : $\langle n \leq m \implies P \downarrow n \sqsubseteq P \downarrow m \rangle$

$\langle proof \rangle$

lemma *le-right-mono-restriction-process_{ptick}-FD* : $\langle n \leq m \implies P \downarrow n \sqsubseteq_{FD} P \downarrow m \rangle$
 $\langle proof \rangle$

lemma *le-right-mono-restriction-process_{ptick}-F* : $\langle n \leq m \implies P \downarrow n \sqsubseteq_F P \downarrow m \rangle$
 $\langle proof \rangle$

lemma *restriction-process_{ptick}-le-right-mono-D* : $\langle n \leq m \implies P \downarrow n \sqsubseteq_D P \downarrow m \rangle$
 $\langle proof \rangle$

lemma *restriction-process_{ptick}-le-right-mono-T* : $\langle n \leq m \implies P \downarrow n \sqsubseteq_T P \downarrow m \rangle$
 $\langle proof \rangle$

lemma *restriction-process_{ptick}-le-right-mono-DT* : $\langle n \leq m \implies P \downarrow n \sqsubseteq_{DT} P \downarrow m \rangle$
 $\langle proof \rangle$

1.6.3 Interpretations of Refinements

lemma *ex-not-restriction-leD* : $\langle \exists n. \neg P \downarrow n \sqsubseteq_D Q \downarrow n \text{ if } \langle \neg P \sqsubseteq_D Q \rangle \rangle$
 $\langle proof \rangle$

interpretation *PRS-leF* : *PreorderRestrictionSpace* $\langle (\downarrow) \rangle \langle (\sqsubseteq_F) \rangle$
 $\langle proof \rangle$

interpretation *PRS-leT* : *PreorderRestrictionSpace* $\langle (\downarrow) \rangle \langle (\sqsubseteq_T) \rangle$
 $\langle proof \rangle$

interpretation *PRS-leDT* : *PreorderRestrictionSpace* $\langle (\downarrow) \rangle \langle (\sqsubseteq_{DT}) \rangle$
 $\langle proof \rangle$

1.7 Continuity

context begin

private lemma *chain-restriction-process_{ptick}* : $\langle chain Y \implies chain (\lambda i. Y i \downarrow n) \rangle$
 $\langle proof \rangle$ **lemma** *cont-prem-restriction-process_{ptick}* :
 $\langle (\bigsqcup i. Y i) \downarrow n = (\bigsqcup i. Y i \downarrow n) \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$) **if** $\langle chain Y \rangle$
 $\langle proof \rangle$

lemma *restriction-process_{ptick}-cont [simp]* : $\langle \text{cont } (\lambda x. f x \downarrow n) \rangle$ **if**
 $\langle \text{cont } f \rangle$
 $\langle \text{proof} \rangle$

end

1.8 Completeness

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

lemma *chain-restriction-chain* :
 $\langle \text{restriction-chain } \sigma \implies \text{chain } \sigma \rangle$ **for** $\sigma :: \langle \text{nat} \Rightarrow ('a, 'r) \text{ process}_{\text{ptick}} \rangle$
 $\langle \text{proof} \rangle$

lemma *restricted-LUB-restriction-chain-is* :
 $\langle (\lambda n. (\bigsqcup n. \sigma n) \downarrow n) = \sigma \rangle$ **if** $\langle \text{restriction-chain } \sigma \rangle$
 $\langle \text{proof} \rangle$

instance *process_{ptick}* :: *(type, type) complete-restriction-space*
 $\langle \text{proof} \rangle$

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

$\langle \text{ML} \rangle$

2 Constructiveness of Prefixes

2.1 Equality

lemma *restriction-process_{ptick}-Mprefix* :
 $\langle \square a \in A \rightarrow P a \downarrow n = (\text{case } n \text{ of } 0 \Rightarrow \perp \mid \text{Suc } m \Rightarrow \square a \in A \rightarrow (P a \downarrow m)) \rangle$ **(is** $\langle ?lhs = ?rhs \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-Mndetprefix* :
 $\langle \square a \in A \rightarrow P a \downarrow n = (\text{case } n \text{ of } 0 \Rightarrow \perp \mid \text{Suc } m \Rightarrow \square a \in A \rightarrow (P a \downarrow m)) \rangle$ **(is** $\langle ?lhs = ?rhs \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-write0* :
 $\langle a \rightarrow P \downarrow n = (\text{case } n \text{ of } 0 \Rightarrow \perp \mid \text{Suc } m \Rightarrow a \rightarrow (P \downarrow m)) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-write* :

$\langle c!a \rightarrow P \downarrow n = (\text{case } n \text{ of } 0 \Rightarrow \perp \mid \text{Suc } m \Rightarrow c!a \rightarrow (P \downarrow m)) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-read* :

$\langle c?a \in A \rightarrow P a \downarrow n = (\text{case } n \text{ of } 0 \Rightarrow \perp \mid \text{Suc } m \Rightarrow c?a \in A \rightarrow (P a \downarrow m)) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-ndet-write* :

$\langle c!!a \in A \rightarrow P a \downarrow n = (\text{case } n \text{ of } 0 \Rightarrow \perp \mid \text{Suc } m \Rightarrow c!!a \in A \rightarrow (P a \downarrow m)) \rangle$
 $\langle \text{proof} \rangle$

2.2 Constructiveness

lemma *Mprefix-constructive* : $\langle \text{constructive } (\lambda P. \Box a \in A \rightarrow P a) \rangle$
 $\langle \text{proof} \rangle$

lemma *Mndetprefix-constructive* : $\langle \text{constructive } (\lambda P. \Box a \in A \rightarrow P a) \rangle$
 $\langle \text{proof} \rangle$

lemma *write0-constructive* : $\langle \text{constructive } (\lambda P. a \rightarrow P) \rangle$
 $\langle \text{proof} \rangle$

lemma *write-constructive* : $\langle \text{constructive } (\lambda P. c!a \rightarrow P) \rangle$
 $\langle \text{proof} \rangle$

lemma *read-constructive* : $\langle \text{constructive } (\lambda P. c?a \in A \rightarrow P a) \rangle$
 $\langle \text{proof} \rangle$

lemma *ndet-write-constructive* : $\langle \text{constructive } (\lambda P. c!!a \in A \rightarrow P a) \rangle$
 $\langle \text{proof} \rangle$

3 Non Destructiveness of Choices

3.1 Equality

lemma *restriction-process_{ptick}-Ndet* : $\langle P \sqcap Q \downarrow n = (P \downarrow n) \sqcap (Q \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-GlobalNdet* :

$\langle (\Box a \in A. P a) \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \Box a \in A. (P a \downarrow n)) \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-GlobalDet* :
 $\langle (\Box a \in A. P a) \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \Box a \in A. (P a \downarrow n)) \rangle$
 $(\text{is } \langle ?lhs = (\text{if } n = 0 \text{ then } \perp \text{ else } ?rhs) \rangle)$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-Det*: $\langle P \Box Q \downarrow n = (P \downarrow n) \Box (Q \downarrow n) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-Sliding*: $\langle P \triangleright Q \downarrow n = (P \downarrow n) \triangleright (Q \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

3.2 Non Destructiveness

lemma *GlobalNdet-non-destructive* : $\langle \text{non-destructive } (\lambda P. \Box a \in A. P a) \rangle$
 $\langle \text{proof} \rangle$

lemma *Ndet-non-destructive* : $\langle \text{non-destructive } (\lambda(P, Q). P \Box Q) \rangle$
 $\langle \text{proof} \rangle$

lemma *GlobalDet-non-destructive* : $\langle \text{non-destructive } (\lambda P. \Box a \in A. P a) \rangle$
 $\langle \text{proof} \rangle$

lemma *Det-non-destructive* : $\langle \text{non-destructive } (\lambda(P, Q). P \Box Q) \rangle$
 $\langle \text{proof} \rangle$

corollary *Sliding-non-destructive* : $\langle \text{non-destructive } (\lambda(P, Q). P \triangleright Q) \rangle$
 $\langle \text{proof} \rangle$

4 Non Destructiveness of Renaming

4.1 Equality

lemma *restriction-process_{ptick}-Renaming*:
 $\langle \text{Renaming } P f g \downarrow n = \text{Renaming } (P \downarrow n) f g \rangle$ (is $\langle ?lhs = ?rhs \rangle$)
 $\langle \text{proof} \rangle$

4.2 Non Destructiveness

lemma *Renaming-non-destructive [simp]* :
 $\langle \text{non-destructive } (\lambda P. \text{Renaming } P f g) \rangle$
 $\langle \text{proof} \rangle$

5 Non Destructiveness of Sequential Composition

5.1 Refinement

lemma *restriction-process_{ptick}-Seq-FD* :
 $\langle P ; Q \downarrow n \sqsubseteq_{FD} (P \downarrow n) ; (Q \downarrow n) \rangle$ (**is** $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$)
 $\langle proof \rangle$

corollary *restriction-process_{ptick}-MultiSeq-FD* :
 $\langle (SEQ\ l \in@ L. P\ l) \downarrow n \sqsubseteq_{FD} SEQ\ l \in@ L. (P\ l \downarrow n) \rangle$
 $\langle proof \rangle$

5.2 Non Destructiveness

lemma *Seq-non-destructive* :
 $\langle non\text{-destructive } (\lambda(P :: ('a, 'r) process_{ptick}, Q). P ; Q) \rangle$
 $\langle proof \rangle$

6 Non Destructiveness of Synchronization Product

6.1 Preliminaries

lemma *D-Sync-optimized* :
 $\langle \mathcal{D} (P \llbracket A \rrbracket Q) =$
 $\{v @ w \mid t\ u\ v\ w. tF\ v \wedge ftF\ w \wedge$
 $v\ setinterleaves\ ((t, u), range\ tick \cup ev\ 'A) \wedge$
 $(t \in \mathcal{D}\ P \wedge u \in \mathcal{T}\ Q \vee t \in \mathcal{D}\ Q \wedge u \in \mathcal{T}\ P)\} \rangle$
(is $\langle - = ?rhs \rangle$)
 $\langle proof \rangle$

lemma *tickFree-interleave-iff* :
 $\langle t\ setinterleaves\ ((u, v), S) \implies tF\ t \longleftrightarrow tF\ u \wedge tF\ v \rangle$
 $\langle proof \rangle$

lemma *interleave-subsetL* :
 $\langle tF\ t \implies \{a. ev\ a \in set\ u\} \subseteq A \implies$
 $t\ setinterleaves\ ((u, v), range\ tick \cup ev\ 'A) \implies t = v \rangle$
for $t\ u\ v :: \langle ('a, 'r) trace_{ptick} \rangle$
 $\langle proof \rangle$

lemma *interleave-subsetR* :
 $\langle tF\ t \implies \{a. ev\ a \in set\ v\} \subseteq A \implies$
 $t\ setinterleaves\ ((u, v), range\ tick \cup ev\ 'A) \implies t = u \rangle$
 $\langle proof \rangle$

lemma *interleave-imp-lengthLR-le* :
 $\langle t \text{ setinterleaves } ((u, v), S) \implies$
 $\text{length } u \leq \text{length } t \wedge \text{length } v \leq \text{length } t \rangle$
 $\langle \text{proof} \rangle$

lemma *interleave-le-prefixLR* :
 $\langle t \text{ setinterleaves } ((u, v), S) \implies u' \leq u \implies v' \leq v \implies$
 $(\exists t' \leq t. \exists v'' \leq v'. t' \text{ setinterleaves } ((u', v''), S)) \vee$
 $(\exists t' \leq t. \exists u'' \leq u'. t' \text{ setinterleaves } ((u'', v'), S)) \rangle$
 $\langle \text{proof} \rangle$

lemma *restriction-process_{ptick}-Sync-FD-div-oneside* :
assumes $\langle tF \ u \rangle \langle ftF \ v \rangle \langle t-P \in \mathcal{D} \ (P \downarrow n) \rangle \langle t-Q \in \mathcal{T} \ (Q \downarrow n) \rangle$
 $\langle u \text{ setinterleaves } ((t-P, t-Q), \text{range tick} \cup \text{ev } A) \rangle$
shows $\langle u @ v \in \mathcal{D} \ (P \llbracket A \rrbracket \ Q \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

6.2 Refinement

lemma *restriction-process_{ptick}-Sync-FD* :
 $\langle P \llbracket A \rrbracket \ Q \downarrow n \sqsubseteq_{FD} (P \downarrow n) \llbracket A \rrbracket \ (Q \downarrow n) \rangle$ (**is** $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$)
 $\langle \text{proof} \rangle$

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* :
 $\langle P \llbracket A \rrbracket \ Q \downarrow n = (P \downarrow n) \llbracket A \rrbracket \ (Q \downarrow n) \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
if $\langle \alpha(P) \subseteq A \vee \alpha(Q) \subseteq A \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-MultiSync-FD* :
 $\langle \llbracket A \rrbracket \ m \in \# \ M. P \ l \downarrow n \sqsubseteq_{FD} \llbracket A \rrbracket \ m \in \# \ M. (P \ l \downarrow n) \rangle$
 $\langle \text{proof} \rangle$

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

corollary *strict-events-of-subset-restriction-process_{ptick}-MultiSync* :
 $\langle \llbracket A \rrbracket \ m \in \# \ M. P \ m \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \llbracket A \rrbracket \ m \in \# \ M. (P \ m \downarrow n)) \rangle$
— *if $n = 0$ then \perp else* - is necessary because we can have $M = \{\#\}$.
if $\langle \bigwedge m. m \in \# \ M \implies \alpha(P \ m) \subseteq A \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-process_{ptick}-Par* :
 $\langle P \parallel Q \downarrow n = (P \downarrow n) \parallel (Q \downarrow n) \rangle$

⟨proof⟩

corollary *restriction-process_{ptick}-MultiPar* :

⟨|| $m \in \# M$. $P \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } || m \in \# M. (P \downarrow n))$ ⟩

⟨proof⟩

6.3 Non Destructiveness

lemma *Sync-non-destructive* :

⟨non-destructive $(\lambda(P :: ('a, 'r) \text{ process}_{ptick}, Q). P \llbracket A \rrbracket Q)$ ⟩

⟨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 $(\lambda(P :: ('a, 'r) \text{ process}_{ptick}, Q). P \Theta a \in A. Q a)$ ⟩

⟨proof⟩

lemma *ThrowR-constructive-if-disjoint-initials* :

⟨constructive $(\lambda Q :: 'a \Rightarrow ('a, 'r) \text{ process}_{ptick}. P \Theta a \in A. Q a)$ ⟩

if $\langle A \cap \{e. \text{ev } e \in P^0\} = \{\} \rangle$

⟨proof⟩

8 Non Destructiveness of Interrupt

8.1 Refinement

lemma *restriction-process_{ptick}-Interrupt-FD* :
 $\langle P \triangle Q \downarrow n \sqsubseteq_{FD} (P \downarrow n) \triangle (Q \downarrow n) \rangle$ (is $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$)
 $\langle proof \rangle$

8.2 Non Destructiveness

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

9 Non too Destructiveness of After

9.1 Equality

lemma *initials-restriction-process_{ptick}*: $\langle (P \downarrow n)^0 = (if\ n = 0\ then\ UNIV\ else\ P^0) \rangle$
 $\langle proof \rangle$

lemma (in *After*) *restriction-process_{ptick}-After*:
 $\langle P\ after\ e\ \downarrow\ n = (if\ ev\ e \in P^0\ then\ (P\ \downarrow\ Suc\ n)\ after\ e\ else\ \Psi\ P\ e\ \downarrow\ n) \rangle$
 $\langle proof \rangle$

lemma (in *AfterExt*) *restriction-process_{ptick}-After_{tick}*:
 $\langle P\ after_{\checkmark}\ e\ \downarrow\ n =$
 $(case\ of\ \checkmark(r) \Rightarrow \Omega\ P\ r\ \downarrow\ n \mid ev\ x \Rightarrow if\ e \in P^0\ then\ (P\ \downarrow\ Suc\ n)$
 $after_{\checkmark}\ e\ else\ \Psi\ P\ x\ \downarrow\ n) \rangle$
 $\langle proof \rangle$

lemma (in *AfterExt*) *restriction-process_{ptick}-After_{trace}*:
 $\langle t \in \mathcal{T}\ P \Rightarrow tF\ t \Rightarrow P\ after_{\mathcal{T}}\ t\ \downarrow\ n = (P\ \downarrow\ (n + length\ t))\ after_{\mathcal{T}}\ t \rangle$
 $\langle proof \rangle$

9.2 Non too Destructiveness

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

lemma (in *AfterExt*) *non-too-destructive-on-After_{tick}* :
 $\langle non-too-destructive-on (\lambda P. P\ after_{\checkmark}\ e) \{P. e \in P^0\} \rangle$
if $\langle \bigwedge r. e = \checkmark(r) \Rightarrow non-too-destructive-on\ \Omega\ \{P. \checkmark(r) \in P^0\} \rangle$
 $\langle proof \rangle$

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

lemma (in *AfterExt*) *non-too-destructive-After_{tick}* :
 ⟨*non-too-destructive* ($\lambda P. P$ after \checkmark e)⟩
if * : ⟨ $\bigwedge a. e = \text{ev } a \implies \text{non-too-destructive-on } \Psi \{P. \text{ev } a \notin P^0\}$ ⟩
 ⟨ $\bigwedge r. e = \checkmark(r) \implies \text{non-too-destructive } (\lambda P. \Omega P r)$ ⟩
 ⟨*proof*⟩

lemma (in *AfterExt*) *restriction-shift-After_{trace}* :
 ⟨*restriction-shift* ($\lambda P. P$ after τ t) ($- \text{int } (\text{length } t)$)⟩
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 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* :
 ⟨ $\exists P Q :: ('a, 'r) \text{process}_{ptick}. P \downarrow n = Q \downarrow n \wedge (P \setminus S) \downarrow \text{Suc } 0 \neq$
 $(Q \setminus S) \downarrow \text{Suc } 0$ ⟩ **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*] :
 $\langle \text{adm}_\downarrow (\lambda x. t \in \mathcal{D} (f x)) \rangle$
and *restriction-adm-notin-D* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{adm}_\downarrow (\lambda x. t \notin \mathcal{D} (f x)) \rangle$
and *restriction-adm-in-F* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{adm}_\downarrow (\lambda x. (t, X) \in \mathcal{F} (f x)) \rangle$
and *restriction-adm-notin-F* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{adm}_\downarrow (\lambda x. (t, X) \notin \mathcal{F} (f x)) \rangle$ **if** $\langle \text{cont}_\downarrow f \rangle$
for $f :: \langle 'b :: \text{restriction} \Rightarrow ('a, 'r) \text{process}_{\text{ptick}} \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-adm-in-T* [*restriction-adm-process_{ptick}-simpset*] :
:
 $\langle \text{cont}_\downarrow f \implies \text{adm}_\downarrow (\lambda x. t \in \mathcal{T} (f x)) \rangle$
and *restriction-adm-notin-T* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{cont}_\downarrow f \implies \text{adm}_\downarrow (\lambda x. t \notin \mathcal{T} (f x)) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-adm-in-initials* [*restriction-adm-process_{ptick}-simpset*] :
:
 $\langle \text{cont}_\downarrow f \implies \text{adm}_\downarrow (\lambda x. e \in (f x)^0) \rangle$
and *restriction-adm-notin-initials* [*restriction-adm-process_{ptick}-simpset*] :
:
 $\langle \text{cont}_\downarrow f \implies \text{adm}_\downarrow (\lambda x. e \notin (f x)^0) \rangle$
 $\langle \text{proof} \rangle$

11.2 Refining

corollary *restriction-adm-leF* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{cont}_\downarrow f \implies \text{cont}_\downarrow g \implies \text{adm}_\downarrow (\lambda x. f x \sqsubseteq_F g x) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-adm-leD* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{cont}_\downarrow f \implies \text{cont}_\downarrow g \implies \text{adm}_\downarrow (\lambda x. f x \sqsubseteq_D g x) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-adm-leT* [*restriction-adm-process_{ptick}-simpset*] :
 $\langle \text{cont}_\downarrow f \implies \text{cont}_\downarrow g \implies \text{adm}_\downarrow (\lambda x. f x \sqsubseteq_T g x) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-adm-leFD* [*restriction-adm-process_{ptick}-simpset*] :
:
 $\langle \text{cont}_\downarrow f \implies \text{cont}_\downarrow g \implies \text{adm}_\downarrow (\lambda x. f x \sqsubseteq_{FD} g x) \rangle$
 $\langle \text{proof} \rangle$

corollary *restriction-adm-leDT* [*restriction-adm-process_{ptick}-simpset*]

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

11.2.1 Transitions

lemma (in *After*) *restriction-cont-After* [*restriction-adm-simpset*] :
 $\langle cont_{\downarrow} (\lambda x. f x \text{ after } a) \rangle$ **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*]
:
 $\langle cont_{\downarrow} (\lambda x. f x \text{ after}_{\checkmark} e) \rangle$ **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*]
:
 $\langle cont_{\downarrow} (\lambda x. f x \text{ after}_{\tau} t) \rangle$ **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*:
 $\langle \bigwedge f g :: 'b \Rightarrow ('a, 'r) process_{ptick}. cont_{\downarrow} f \implies cont_{\downarrow} g \implies adm_{\downarrow} (\lambda x. f x \rightsquigarrow_{\tau} g x) \rangle$
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 $\langle adm_{\downarrow} (\lambda x. f x \rightsquigarrow_e g x) \rangle$
 $\langle proof \rangle$

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*:
 $\langle \bigwedge f g :: 'b \Rightarrow ('a, 'r) process_{ptick}. cont_{\downarrow} f \implies cont_{\downarrow} g \implies adm_{\downarrow} (\lambda x. f x \rightsquigarrow_{\tau} g x) \rangle$
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 $\langle adm_{\downarrow} (\lambda x. f x \rightsquigarrow_{\checkmark r} (g x)) \rangle$
 $\langle proof \rangle$

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*:
 $\langle \bigwedge f g :: 'b \Rightarrow ('a, 'r) process_{ptick}. cont_{\downarrow} f \implies cont_{\downarrow} g \implies adm_{\downarrow} (\lambda x. f x \rightsquigarrow_{\tau} g x) \rangle$
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 $\langle \text{adm}_\downarrow (\lambda x. f x \rightsquigarrow^* t (g x)) \rangle$
 $\langle \text{proof} \rangle$

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*]
 :
 $\langle \text{constructive } (\lambda x. \Box a \in A \rightarrow f a x) \rangle$ **if** $\langle (\bigwedge a. a \in A \implies \text{non-destructive } (f a)) \rangle$
 $\langle \text{proof} \rangle$

lemma *Mndetprefix-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle \text{constructive } (\lambda x. \Box a \in A \rightarrow f a x) \rangle$ **if** $\langle (\bigwedge a. a \in A \implies \text{non-destructive } (f a)) \rangle$
 $\langle \text{proof} \rangle$

corollary *write0-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle \text{non-destructive } f \implies \text{constructive } (\lambda x. a \rightarrow f x) \rangle$
 $\langle \text{proof} \rangle$

corollary *write-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle \text{non-destructive } f \implies \text{constructive } (\lambda x. c!a \rightarrow f x) \rangle$
 $\langle \text{proof} \rangle$

corollary *read-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle (\bigwedge a. a \in A \implies \text{non-destructive } (f a)) \implies \text{constructive } (\lambda x. c?a \in A \rightarrow f a x) \rangle$
 $\langle \text{proof} \rangle$

corollary *ndet-write-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle (\bigwedge a. a \in A \implies \text{non-destructive } (f a)) \implies \text{constructive } (\lambda x. c!!a \in A \rightarrow f a x) \rangle$

$\langle proof \rangle$

12.2 Choices

lemma *GlobalNdet-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

:

$\langle (\bigwedge a. a \in A \implies \text{non-destructive } (f a)) \implies \text{non-destructive } (\lambda x. \sqcap a \in A. f a x) \rangle$

$\langle (\bigwedge a. a \in A \implies \text{constructive } (f a)) \implies \text{constructive } (\lambda x. \sqcap a \in A. f a x) \rangle$

$\langle proof \rangle$

lemma *GlobalDet-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

:

$\langle (\bigwedge a. a \in A \implies \text{non-destructive } (f a)) \implies \text{non-destructive } (\lambda x. \sqcap a \in A. f a x) \rangle$

$\langle (\bigwedge a. a \in A \implies \text{constructive } (f a)) \implies \text{constructive } (\lambda x. \sqcap a \in A. f a x) \rangle$

$\langle proof \rangle$

lemma *Ndet-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

:

$\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f x \sqcap g x) \rangle$

$\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f x \sqcap g x) \rangle$

$\langle proof \rangle$

lemma *Det-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

:

$\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f x \sqcap g x) \rangle$

$\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f x \sqcap g x) \rangle$

$\langle proof \rangle$

lemma *Sliding-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

:

$\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f x \triangleright g x) \rangle$

$\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f x \triangleright g x) \rangle$

$\langle proof \rangle$

12.3 Renaming

lemma *Renaming-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

:

$\langle \text{non-destructive } P \implies \text{non-destructive } (\lambda x. \text{Renaming } (P x) f g) \rangle$

$\langle \text{constructive } P \implies \text{constructive } (\lambda x. \text{Renaming } (P x) f g) \rangle$

$\langle proof \rangle$

12.4 Sequential Composition

lemma *Seq-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f \ x ; g \ x) \rangle$
 $\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f \ x ; g \ x) \rangle$
 $\langle \text{proof} \rangle$

lemma *MultiSeq-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle (\bigwedge l. l \in \text{set } L \implies \text{non-destructive } (f \ l)) \implies \text{non-destructive } (\lambda x. \text{SEQ } l \in @ L. f \ l \ x) \rangle$
 $\langle (\bigwedge l. l \in \text{set } L \implies \text{constructive } (f \ l)) \implies \text{constructive } (\lambda x. \text{SEQ } l \in @ L. f \ l \ x) \rangle$
 $\langle \text{proof} \rangle$

corollary *MultiSeq-non-destructive* : $\langle \text{non-destructive } (\lambda P. \text{SEQ } l \in @ L. P \ l) \rangle$
 $\langle \text{proof} \rangle$

12.5 Synchronization Product

lemma *Sync-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f \ \llbracket S \rrbracket g \ x) \rangle$
 $\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f \ \llbracket S \rrbracket g \ x) \rangle$
 $\langle \text{proof} \rangle$

lemma *MultiSync-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :
 $\langle (\bigwedge m. m \in \# M \implies \text{non-destructive } (f \ m)) \implies \text{non-destructive } (\lambda x. \llbracket S \rrbracket m \in \# M. f \ m \ x) \rangle$
 $\langle (\bigwedge m. m \in \# M \implies \text{constructive } (f \ m)) \implies \text{constructive } (\lambda x. \llbracket S \rrbracket m \in \# M. f \ m \ x) \rangle$
 $\langle \text{proof} \rangle$

corollary *MultiSync-non-destructive* : $\langle \text{non-destructive } (\lambda P. \llbracket S \rrbracket m \in \# M. P \ m) \rangle$
 $\langle \text{proof} \rangle$

12.6 Throw

lemma *Throw-restriction-shift-process_{ptick}* [*restriction-shift-process_{ptick}-simpset*]
 :

$\langle \text{non-destructive } f \implies (\bigwedge a. a \in A \implies \text{non-destructive } (g a)) \implies$
 $\text{non-destructive } (\lambda x. f x \Theta a \in A. g a x) \rangle$
 $\langle \text{constructive } f \implies (\bigwedge a. a \in A \implies \text{constructive } (g a)) \implies \text{constructive } (\lambda x. f x \Theta a \in A. g a x) \rangle$
 $\langle \text{proof} \rangle$

12.7 Interrupt

lemma *Interrupt-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

$\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f$
 $x \Delta g x) \rangle$
 $\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f x \Delta g x) \rangle$
 $\langle \text{proof} \rangle$

12.8 After

lemma (in *After*) *After-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

$\langle \text{non-too-destructive } \Psi \implies \text{constructive } f \implies \text{non-destructive } (\lambda x.$
 $f x \text{ after } a) \rangle$
 $\langle \text{non-too-destructive } \Psi \implies \text{non-destructive } f \implies \text{non-too-destructive } (\lambda x. f x \text{ after } a) \rangle$
 $\langle \text{proof} \rangle$

lemma (in *AfterExt*) *After_{tick}-restriction-shift-process_{ptick} [restriction-shift-process_{ptick}-simpset]*

$\langle \llbracket \text{non-too-destructive } \Psi; \text{non-too-destructive } \Omega; \text{constructive } f \rrbracket$
 $\implies \text{non-destructive } (\lambda x. f x \text{ after } \surd e) \rangle$
 $\langle \llbracket \text{non-too-destructive } \Psi; \text{non-too-destructive } \Omega; \text{non-destructive } f \rrbracket$
 $\implies \text{non-too-destructive } (\lambda x. f x \text{ after } \surd e) \rangle$
 $\langle \text{proof} \rangle$

12.9 Illustration

declare *restriction-shift-process_{ptick}-simpset [simp]*

notepad begin

$\langle \text{proof} \rangle$

end

Finally, we add the following new law. A similar result involving the least fixed-point operator inherited from HOLCF was already available: *Renaming-fix*.

lemma *Renaming-restriction-fix* :

$\langle \text{Renaming } (v X. \varphi X) f g = (v X. ((\lambda P. \text{Renaming } P f g) \circ \varphi \circ$
 $(\lambda P. \text{Renaming } P (\text{inv } f) (\text{inv } g))) X) \rangle$

(**is** $\langle \text{Renaming } (v X. \varphi X) f g = (v X. ?\varphi' X) \rangle$) **if** $\langle \text{constructive } \varphi \rangle$
and $\langle \text{inj } f \rangle$ **and** $\langle \text{inj } g \rangle$ **for** $\varphi :: \langle ('a, 'r) \text{ process}_{ptick} \Rightarrow ('a, 'r)$
 $\text{process}_{ptick} \rangle$
 $\langle \text{proof} \rangle$

end