HOL-CSP_PTick
Parameterized Termination for Sequential Composition and Synchronization Product

Benoît Ballenghien

February 10, 2026

Abstract

Recently, parameterized termination has been introduced in `HOL-CSP`, allowing the termination event `tick` to carry a result value, in a way analogous to the `return` of a state monad. This conservative extension of the CSP theory required the generalization of several denotational definitions and the adaptation of numerous proofs. Since Isabelle2025, this work has been completed for the `HOL-CSP`, `HOL-CSPM`, and `HOL-CSP_OpSem` sessions. However, for two operators—namely sequential composition and the synchronization product—the most direct generalizations turn out to be conceptually unsatisfactory, in particular with respect to their interaction with $SKIP$. To address this issue, we introduce in this entry generalized versions of these operators that fully exploit the expressive power of parameterized termination; in particular, the resulting notion of sequential composition satisfies the monad laws. Building on these definitions, we establish a range of algebraic and operational laws, as well as fundamental properties such as continuity and non-destructiveness.

Contents

1 Introduction **11**
- 1.1 Motivations . . . 11
- 1.2 The Global Architecture of `HOL-CSP_PTick` . . . 12

2 Finite Ticks Predicate **15**
- 2.1 Definitions . . . 15
- 2.2 Properties . . . 16
  - 2.2.1 Constant Processes . . . 16
  - 2.2.2 Other properties . . . 16
- 2.3 Laws . . . 18
  - 2.3.1 Laws of $\mathbb{F}_{\checkmark}(P)$ . . . 18
  - 2.3.2 Laws of $\mathbb{F}_{\checkmark}{\Rightarrow}(f)$ . . . 20

3 Generalization of the Sequential Composition **23**
- 3.1 Definition . . . 23
  - 3.1.1 Preliminaries . . . 23
  - 3.1.2 Formal Definition . . . 25
- 3.2 Projections . . . 25

4 Generalization of the Synchronization Product **29**
- 4.1 Trace Interleaving . . . 29
  - 4.1.1 Motivation . . . 29
  - 4.1.2 Definition . . . 30
  - 4.1.3 First Properties . . . 32
  - 4.1.4 Lengths . . . 36
  - 4.1.5 Trace Prefix Interleaving . . . 36
  - 4.1.6 Hiding Events . . . 37
- 4.2 Synchronization Product . . . 39
  - 4.2.1 Definition . . . 39
  - 4.2.2 Projections . . . 41
  - 4.2.3 First Properties . . . 42

5 Some Work on Renaming **45**
5.1 Tick Swap Operator . . . 45
5.1.1 Preliminaries . . . 45
5.1.2 The Operator . . . 51
5.2 Splitting the Renaming Operator . . . 58
5.2.1 Renaming only Events . . . 59
5.2.2 Renaming only Ticks . . . 60
5.2.3 Properties . . . 61
5.3 Renaming and Generalized Synchronization Product . . . 63

6 Commutativity and Associativity of Synchronization **65**
6.1 Commutativity . . . 65
6.1.1 Motivation . . . 65
6.1.2 Formalization . . . 65
6.1.3 First Properties . . . 66
6.1.4 Commutativity . . . 66
6.2 Associativity . . . 67
6.2.1 Motivation . . . 67
6.2.2 Formalization . . . 67
6.2.3 First Properties . . . 68
6.2.4 Associativity for the Traces . . . 68
6.2.5 Associativity . . . 69

7 First Laws **71**
7.1 Behaviour with Constant Processes . . . 71
7.1.1 The Laws of $\bot$ . . . 71
7.1.2 The Laws of *STOP* . . . 71
7.1.3 The Laws of *SKIP* . . . 72
7.2 Associativity of Sequential Composition . . . 74
7.3 Distributivity of Non-Determinism . . . 74
7.3.1 Sequential Composition . . . 74
7.3.2 Synchronization Product . . . 74

8 Communications **77**
8.1 Step Laws . . . 77
8.1.1 Sequential Composition . . . 77
8.1.2 Synchronization Product . . . 77
8.2 Extended step Laws . . . 78
8.2.1 Sequential Composition . . . 78
8.2.2 Synchronization Product . . . 78
8.3 Read and Write Laws . . . 83
8.3.1 Sequential Composition . . . 83
8.3.2 Synchronization Product . . . 83

9 Operational Semantics Laws **101**
9.1 Behaviour of *initials* . . . 101
9.1.1 *TickSwap* . . . 101
9.1.2 Sequential Composition . . . 101
9.1.3 Synchronization Product . . . 101
9.2 Laws of After . . . 101
9.2.1 Sequential Composition . . . 101
9.2.2 Synchronization Product . . . 103
9.3 Small Steps Transitions . . . 104
9.3.1 Extension of the After Operator . . . 104
9.3.2 Sequential Composition . . . 104
9.3.3 Generic Operational Semantics as Locales . . . 106

10 Declensions of the Generalized Synchronization Product **111**
10.1 Interpretations . . . 111
10.1.1 Classical Version . . . 111
10.1.2 Product Type . . . 111
10.1.3 List Type . . . 112
10.2 Associativities . . . 114
10.2.1 Classical Version . . . 114
10.2.2 Product Type . . . 114
10.2.3 List Type . . . 115
10.3 Properties . . . 115
10.3.1 Actual Generalization . . . 115
10.3.2 Other Properties . . . 116
10.4 Ticks Length and Conversions . . . 116
10.4.1 Ticks Length . . . 116
10.4.2 Conversions . . . 120
10.5 First Laws . . . 122
10.6 Operational Laws . . . 124
10.6.1 Classical Version . . . 124
10.6.2 Product Type . . . 125
10.6.3 List Type . . . 125

11 Architectural Versions **129**
11.1 Sequential Composition . . . 129
11.1.1 Definition . . . 129
11.1.2 First Properties . . . 129
11.1.3 Behaviour with binary version . . . 130
11.1.4 Other Properties . . . 130
11.1.5 Behaviour with injectivity . . . 130
11.2 Synchronization Product . . . 131
11.2.1 Definition . . . 131
11.2.2 First properties . . . 132

11.2.3 Properties . . . 132
11.2.4 Behaviour with binary version . . . 133
11.2.5 Behaviour with injectivity . . . 133
11.2.6 Permuting the Sequence . . . 133

12 Events and Ticks **137**
12.1 Preliminaries . . . 137
12.2 Sequential Composition . . . 137
12.2.1 Events . . . 137
12.2.2 Ticks . . . 137
12.3 Synchronization Product . . . 138
12.3.1 Events . . . 138
12.3.2 Ticks . . . 138
12.4 Architectural Operators . . . 138
12.4.1 Events . . . 138
12.4.2 Ticks . . . 139

13 Continuity Rules **141**
13.1 Sequential Composition . . . 141
13.1.1 Monotonicity . . . 141
13.1.2 Preliminaries . . . 141
13.1.3 Continuity . . . 142
13.2 Synchronization Product . . . 143
13.2.1 Monotonicity . . . 143
13.2.2 Preliminaries . . . 143
13.2.3 Continuity . . . 143

14 Monotonicity Properties **145**
14.0.1 Sequential Composition . . . 145
14.0.2 Multiple Sequential Composition . . . 145
14.0.3 Synchronization Product . . . 146
14.0.4 Multiple Synchronization Product . . . 146

15 Non Destructiveness Rules **147**
15.1 Synchronization Product . . . 147
15.1.1 Refinement . . . 147
15.1.2 Non Destructiveness . . . 148
15.1.3 Setup . . . 148

16 Other Laws **149**
16.1 Laws of Renaming . . . 149
16.1.1 Renaming and Sequential Composition . . . 149
16.1.2 Renaming and Synchronization Product . . . 150
16.2 Laws of Hiding . . . 150

16.3 Hiding and Sequential Composition . . . 150
16.4 Hiding and Synchronization Product . . . 150
16.5 Other Laws of Synchronization Product . . . 151
16.5.1 Synchronization Set can be restricted . . . 151
16.5.2 Some Refinements . . . 152

17 Deadlock Results **153**
17.1 First Results . . . 153
17.1.1 Non Terminating . . . 153
17.1.2 Deadlock Free . . . 153
17.2 Renaming and reference Processes . . . 154
17.2.1 Alternative Definitions with restriction fixed-point Operator . . . 155
17.2.2 Stronger Results . . . 155
17.3 Data Independence . . . 156
17.3.1 An interesting equivalence . . . 156
17.3.2 *STOP* and *SKIP* synchronized with *DF A* . . . 156
17.3.3 Finally, *deadlock-free* $(P \mid\mid\mid Q)$ . . . 157

18 Conclusion **159**
18.1 Main Entry Point . . . 159
18.2 Conclusion . . . 159
18.2.1 Summary . . . 159
18.2.2 Sequential Composition . . . 160
18.2.3 Synchronization Product . . . 161

Chapter 1

Introduction

1.1 Motivations

Recently, the question arose whether `HOL-CSP` could accommodate a parameterized notion of termination.[1] The idea is very simple: replace at the very beginning of the formalization

datatype $'a$ *event* = *ev* $'a$ | *tick* (‹✓›)

(isomorphic to option type) by

datatype ($'a$, $'r$) $event_{ptick}$ = *ev* $'a$ | *tick* $'r$ (‹✓$'$(-$'$)›)

(isomorphic to sum type), so that the explicit termination event carries a return value.

Certain definitions must therefore be adapted (mainly by replacing ✓ with *range tick*). For example, a trace t was said to be tick-free if ✓ $\notin$ *set* t. In this new setup, such a trace instead satisfies *range tick* $\cap$ *set* $t = \{\}$. Surprisingly, once these few intuitive adjustments have been made, most of the existing Isar proofs remain valid with little to no modification. This generalization has already been carried out, and the AFP entries for `HOL-CSP`, `HOL-CSPM`, and `HOL-CSP_OpSem` have all been updated accordingly [2, 1, 3]. More recently, `HOL-CSP_RS` [5] has been added as well. However, two operators do not behave as satisfactorily as one might hope.

Firstly, sequential composition no longer admits *SKIP* as a neutral element. In the classical theory, we have *Skip* **;** $P = P$ and P **;** *Skip* $= P$. But in the generalized setting, *SKIP* carries a value and if the first law can still be adapted and proven: *SKIP* r **;** $P = P$, the second one only holds when the return type is *unit* (which amounts to ignoring the generalization). From a

[1] This idea was sparked by an innocent remark from Simon Foster, which we later explored in depth.

broader perspective, one would in fact like the right-hand process to depend on the return value of the left-hand process, which is not the case in the current framework.

Secondly, the synchronization product does not properly support synchronized termination. Classically, we have *Skip* $[\![S]\!]$ *Skip* = *Skip*, adapted in the last version of HOL-CSP as *SKIP r* $[\![A]\!]$ *SKIP s* = (*if r* = *s then SKIP r else STOP*). When restricted to $'a$ *process* (which is ($'a$, *unit*) $process_{ptick}$) the behavior is fine, but with general return values deadlocks may occur. One would rather expect a law like *SKIP r* $[\![A]\!]$ *SKIP s* = *SKIP* (*r*, *s*), yet defining such an operator raises non-trivial technical challenges.

In this entry, we propose generalized definitions for sequential composition and synchronization product that not only respect the invariant *is-process* but also fulfill the expectations outlined above. Beyond this substantial work, we establish algebraic and operational properties of these operators, as well as the lemmas required for fixed-point reasoning. In particular, it can be pointed out that the resulting sequential composition operator fulfills the laws of a monad.

1.2 The Global Architecture of HOL-CSP_PTick

Our formalization attempts to take full advantage of parallelization, explaining the shape of the session graph shown in Figure 1.1.

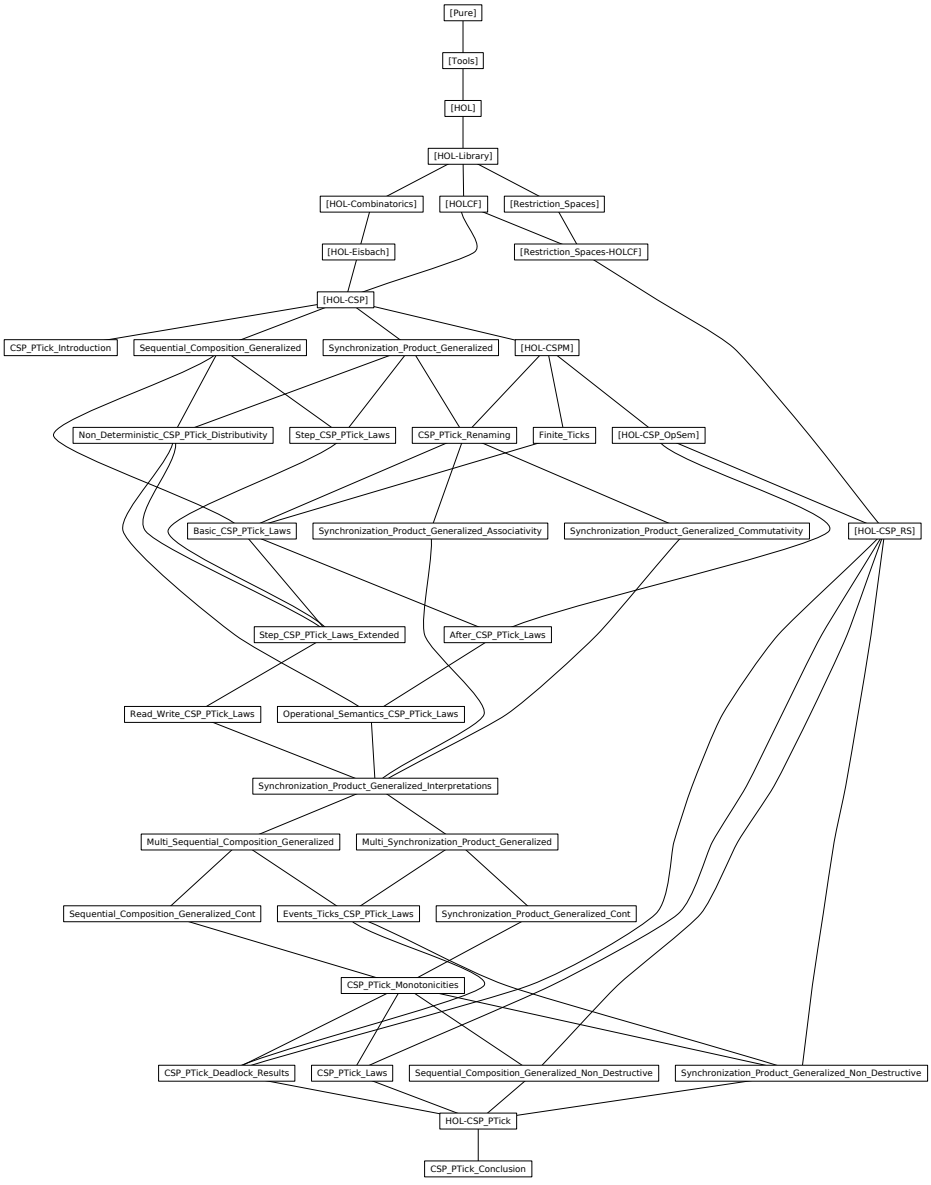

Figure 1.1: The overall architecture

Chapter 2

Finite Ticks Predicate

2.1 Definitions

Due to our generalization, the generalized sequential composition will require this additional assumption for continuity. Intuitively, having an infinite number of possible terminations after a given trace will lead to a infinite branching preventing continuity, to a certain extent like what happens with global non deterministic choice.

definition *finite-all-ticks* :: ‹('a, 'r) $process_{ptick}$ ⇒ *bool*›
  where ‹*finite-all-ticks* $P \equiv \forall\, t \in \mathcal{T}\ P.$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$›

lemma *finite-all-ticksI* : ‹$(\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}) \Longrightarrow$ *finite-all-ticks* P›
  ⟨*proof*⟩

lemma *finite-all-ticksD* : ‹*finite-all-ticks* $P \Longrightarrow$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$›
  ⟨*proof*⟩

Actually, when a *tick* only appears in divergences, it will not matter for continuity. We therefore introduce the modified predicate, which is much more useful.

definition *finite-ticks* :: ‹('a, 'r) $process_{ptick}$ ⇒ *bool*› (‹$\mathbb{F}_{\checkmark}$'(-')›)
  where ‹$\mathbb{F}_{\checkmark}(P) \equiv \forall\, t \in \mathcal{T}\ P.$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P - \mathcal{D}\ P\}$›

lemma *finite-ticksI* :
  ‹$(\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow t \notin \mathcal{D}\ P \Longrightarrow$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}) \Longrightarrow \mathbb{F}_{\checkmark}(P)$›
  ⟨*proof*⟩

lemma *finite-ticksD* :
  ‹$\mathbb{F}_{\checkmark}(P) \Longrightarrow t \notin \mathcal{D}\ P \Longrightarrow$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$›
  ⟨*proof*⟩

lemma *finite-all-ticks-imp-finite-ticks* [*simp*] : ‹*finite-all-ticks* $P \Longrightarrow \mathbb{F}_{\checkmark}(P)$›

⟨*proof*⟩

lemma *finite-all-ticks-is-finite-ticks-or-finite-UNIV* :
‹*finite-all-ticks P* ⟷ (*if* $\mathcal{D}$ *P* = {} *then* $\mathbb{F}_{\checkmark}(P)$ *else finite* (*UNIV* :: *'r set*))›
— This is justifying why *finite-all-ticks* is not really interesting.
for *P* :: ‹('*a*, '*r*) *process*$_{ptick}$›
⟨*proof*⟩

We also introduce the concept that a function can preserve *finite-ticks*. Unfortunately, we will not succeed in proving continuity under this condition for generalized sequential composition.

definition *finite-ticks-fun* :: ‹(('*a*, '*r*) *process*$_{ptick}$ ⇒ ('*b*, '*s*) *process*$_{ptick}$) ⇒ *bool*›
(‹$\mathbb{F}_{\checkmark\Rightarrow}$'(-')›)
where ‹$\mathbb{F}_{\checkmark\Rightarrow}(f)$ ≡ ∀ *P*. $\mathbb{F}_{\checkmark}(P)$ ⟶ $\mathbb{F}_{\checkmark}(f\ P)$›

lemma *finite-ticks-funI*: ‹(⋀*P*. $\mathbb{F}_{\checkmark}(P)$ ⟹ $\mathbb{F}_{\checkmark}(f\ P)$) ⟹ $\mathbb{F}_{\checkmark\Rightarrow}(f)$›
⟨*proof*⟩

lemma *finite-ticks-funD*: ‹$\mathbb{F}_{\checkmark\Rightarrow}(f)$ ⟹ $\mathbb{F}_{\checkmark}(P)$ ⟹ $\mathbb{F}_{\checkmark}(f\ P)$›
⟨*proof*⟩

2.2 Properties

named-theorems *finite-ticks-simps*
named-theorems *finite-ticks-fun-simps*

2.2.1 Constant Processes

lemma *finite-ticks-BOT* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(\bot)$›
⟨*proof*⟩

lemma *finite-ticks-fun-BOT* [*finite-ticks-fun-simps*] : ‹$\mathbb{F}_{\checkmark\Rightarrow}(\bot)$›
⟨*proof*⟩

lemma *finite-ticks-SKIP* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}$(*SKIP r*)›
⟨*proof*⟩

lemma *finite-ticks-STOP* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}$(*STOP*)›
⟨*proof*⟩

lemma *finite-ticks-SKIPS-iff* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}$(*SKIPS R*) ⟷ *finite R*›
⟨*proof*⟩

2.2.2 Other properties

lemma *finite-strict-ticks-of-imp-finite-ticks* [*finite-ticks-simps*] :
‹*finite* ✓**s**(*P*) ⟹ $\mathbb{F}_{\checkmark}(P)$›
⟨*proof*⟩

lemma *finite-strict-ticks-of-image-imp-finite-ticks-fun* [*finite-ticks-fun-simps*] :
‹$(\bigwedge x.\ \mathit{finite}\ \checkmark\mathbf{s}(f\ x)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f)$›
⟨*proof*⟩

lemma *anti-mono-finite-ticks* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P)$› **if** ‹$P \sqsubseteq Q$› ‹$\mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

lemma *anti-mono-finite-ticks-fun* [*finite-ticks-fun-simps*] :
‹$f \sqsubseteq g \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f)$›
⟨*proof*⟩

lemma *finite-ticks-LUB-iff* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark}(\bigsqcup i.\ Y\ i) \longleftrightarrow (\forall\, i.\ \mathbb{F}_{\checkmark}(Y\ i))$› **if** ‹*chain* Y›
⟨*proof*⟩

lemma *adm-finite-ticks* [*finite-ticks-simps*] : ‹*adm* $(\lambda P.\ \mathbb{F}_{\checkmark}(P))$›
⟨*proof*⟩

lemma *finite-ticks-fix* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(\mu\ X.\ f\ X)$› **if** ‹*cont* f› **and** ‹$\mathbb{F}_{\checkmark\Rightarrow}(f)$›
⟨*proof*⟩

lemma *adm-finite-ticks-fun* [*finite-ticks-fun-simps*] : ‹*adm* $(\lambda f.\ \mathbb{F}_{\checkmark\Rightarrow}(f))$›
⟨*proof*⟩

lemma *finite-ticks-fun-fix* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\mu\ X.\ f\ X)$› **if** ‹*cont* f› **and** ‹$\bigwedge x.\ \mathbb{F}_{\checkmark\Rightarrow}(x) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-id* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\mathit{id})$› ‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-const-iff* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ P) \longleftrightarrow \mathbb{F}_{\checkmark}(P)$›
⟨*proof*⟩

lemma *finite-ticks-fun-comp* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ g\ (f\ x))$›
⟨*proof*⟩

2.3 Laws

2.3.1 Laws of $\mathbb{F}_{\checkmark}(P)$

lemma *finite-ticks-Ndet* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \sqcap Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P)$› ‹$\mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

lemma *finite-ticks-Det* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \mathbin{\square} Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P)$› ‹$\mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

lemma *finite-ticks-Sliding* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P) \Longrightarrow \mathbb{F}_{\checkmark}(Q) \Longrightarrow \mathbb{F}_{\checkmark}(P \rhd Q)$›
⟨*proof*⟩

lemma *finite-ticks-Interrupt* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \mathbin{\triangle} Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P)$› ‹$\mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

lemma *finite-ticks-Throw* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \mathbin{\Theta} a{\in}A.\ Q\ a)$› **if** ‹$\mathbb{F}_{\checkmark}(P)$› ‹$\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark}(Q\ a)$›
⟨*proof*⟩

lemma *finite-ticks-Renaming* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(\mathit{Renaming}\ P\ f\ g)$› **if** ‹*finitary f*› ‹*finitary g*› ‹$\mathbb{F}_{\checkmark}(P)$›
⟨*proof*⟩

lemma *finite-ticks-Seq* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \mathbin{;} Q)$› **if** ‹$\mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

lemma *finite-ticks-Sync* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \mathbin{[\![} S \mathbin{]\!]} Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

corollary ‹$\mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q) \Longrightarrow \mathbb{F}_{\checkmark}(P \mid\mid Q)$›
and ‹$\mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q) \Longrightarrow \mathbb{F}_{\checkmark}(P \mid\mid\mid Q)$›
⟨*proof*⟩

lemma *finite-ticks-GlobalNdet* [*finite-ticks-simps*] :
‹*finite* $A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark}(P\ a)) \Longrightarrow \mathbb{F}_{\checkmark}(\sqcap a{\in}A.\ P\ a)$›
— We can't expect *infinite* A here, see $\mathbb{F}_{\checkmark}(SKIPS\ R) = finite\ R$.
⟨*proof*⟩

lemma *finite-ticks-GlobalDet* [*finite-ticks-simps*] :
‹*finite* $A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark}(P\ a)) \Longrightarrow \mathbb{F}_{\checkmark}(\Box a{\in}A.\ P\ a)$›
⟨*proof*⟩

lemma ‹$L = [] \Longrightarrow \mathbb{F}_{\checkmark}(SEQ\ l{\in}@L.\ P\ l)$› ⟨*proof*⟩

lemma *finite-ticks-MultiSeq-nonempty* [*finite-ticks-simps*] :
‹$L \neq [] \Longrightarrow \mathbb{F}_{\checkmark}(P\ (last\ L)) \Longrightarrow \mathbb{F}_{\checkmark}(SEQ\ l{\in}@L.\ P\ l)$›
⟨*proof*⟩

lemma *finite-ticks-MultiSync* [*finite-ticks-simps*] :
‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark}(P\ m)) \Longrightarrow \mathbb{F}_{\checkmark}([\![S]\!]\ m{\in}\#M.\ P\ m)$›
⟨*proof*⟩

corollary ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark}(P\ m)) \Longrightarrow \mathbb{F}_{\checkmark}(||\ \ m{\in}\#M.\ P\ m)$›
and ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark}(P\ m)) \Longrightarrow \mathbb{F}_{\checkmark}(|||\ m{\in}\#M.\ P\ m)$›
⟨*proof*⟩

lemma *finite-ticks-Mprefix-iff* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(\Box a{\in}A \to P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
⟨*proof*⟩

lemma *finite-ticks-Mndetprefix-iff* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(\sqcap a{\in}A \to P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
⟨*proof*⟩

lemma *finite-ticks-write0-iff* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(a \to P) \longleftrightarrow \mathbb{F}_{\checkmark}(P)$›
⟨*proof*⟩

lemma *finite-ticks-write-iff* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(c!a \to P) \longleftrightarrow \mathbb{F}_{\checkmark}(P)$›
⟨*proof*⟩

lemma *finite-ticks-read-iff* :
‹$\mathbb{F}_{\checkmark}(c?a{\in}A \to P\ a) \longleftrightarrow (\forall\, b{\in}c\ `\ A.\ \mathbb{F}_{\checkmark}(P\ (inv\text{-}into\ A\ c\ b)))$›
⟨*proof*⟩

lemma *finite-ticks-inj-on-read-iff* [*finite-ticks-simps*] :
‹*inj-on* $c\ A \Longrightarrow \mathbb{F}_{\checkmark}(c?a{\in}A \to P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
⟨*proof*⟩

lemma *finite-ticks-ndet-write-iff* :
‹$\mathbb{F}_{\checkmark}(c!!a{\in}A \to P\ a) \longleftrightarrow (\forall\, b{\in}c\ `\ A.\ \mathbb{F}_{\checkmark}(P\ (inv\text{-}into\ A\ c\ b)))$›

⟨*proof*⟩

lemma *finite-ticks-inj-on-ndet-write-iff* [*finite-ticks-simps*] :
‹*inj-on c A* ⟹ $\mathbb{F}_{\checkmark}(c!!a{\in}A \to P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
⟨*proof*⟩

2.3.2 Laws of $\mathbb{F}_{\checkmark\Rightarrow}(f)$

lemma *finite-ticks-fun-Det* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \ \Box\ g\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Ndet* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \sqcap g\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Sliding* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \rhd g\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Interrupt* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \ \triangle\ g\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Throw* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g\ a)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ \Theta\ a{\in}A.\ g\ a\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Renaming* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(P) \Longrightarrow$ *finitary f* ⟹ *finitary g* ⟹ $\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.$ *Renaming* $(P\ x)\ f\ g)$›
⟨*proof*⟩

lemma *finite-ticks-fun-RenamingF* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(P) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ (P\ x)\ [\![a := b]\!]\ [\![c := d]\!])$›
⟨*proof*⟩

lemma *finite-ticks-fun-Seq* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ ;\ g\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Sync* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ [\![S]\!]\ g\ x)$›
⟨*proof*⟩

corollary ‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \ ||\ \ g\ x)$›
and ‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \ |||\ g\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-GlobalNdet* [*finite-ticks-fun-simps*] :
‹*finite* $A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ a)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ \sqcap a{\in}A.\ f\ a\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-GlobalDet* :
‹*finite* $A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ a)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ \Box a{\in}A.\ f\ a\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-MultiSeq* [*finite-ticks-fun-simps*] :
‹$L = [] \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ SEQ\ l{\in}@L.\ f\ l\ x)$›
‹$L \neq [] \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ (last\ L)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ SEQ\ l{\in}@L.\ f\ l\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-MultiSync* [*finite-ticks-fun-simps*] :
‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ m)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ [\![S]\!]\ m{\in}\#M.\ f\ m\ x)$›
⟨*proof*⟩

corollary ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ m)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ ||\ \ m{\in}\#M.\ f\ m\ x)$›
and ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ m)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ |||\ m{\in}\#M.\ f\ m\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Mprefix-iff* :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ \Box a{\in}A \to f\ a\ x) \longleftrightarrow (\forall\, a \in A.\ \mathbb{F}_{\checkmark\Rightarrow}(f\ a))$›
⟨*proof*⟩

lemma *finite-ticks-fun-Mprefix* [*finite-ticks-fun-simps*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ a)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ \Box a{\in}A \to f\ a\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-Mndetprefix-iff* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ \sqcap a{\in}A \to f\ a\ x) \longleftrightarrow (\forall\, a \in A.\ \mathbb{F}_{\checkmark\Rightarrow}(f\ a))$›
⟨*proof*⟩

lemma *finite-ticks-fun-Mndetprefix* [*finite-ticks-fun-simps*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ a)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ \sqcap a{\in}A \to f\ a\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-write0-iff* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ a \to f\ x) \longleftrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f)$›
⟨*proof*⟩

lemma *finite-ticks-fun-write-iff* [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ c!a \to f\ x) \longleftrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f)$›
⟨*proof*⟩

lemma *finite-ticks-fun-read-iff* :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ c?a{\in}A \to f\ a\ x) \longleftrightarrow (\forall\, b{\in}c\ `\ A.\ \mathbb{F}_{\checkmark\Rightarrow}(f\ (\mathit{inv\text{-}into}\ A\ c\ b)))$›
⟨*proof*⟩

lemma *finite-ticks-fun-read* [*finite-ticks-fun-simps*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ a\ x)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ c?a{\in}A \to f\ a\ x)$›
⟨*proof*⟩

lemma *finite-ticks-fun-ndet-write-iff* :
‹$\mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ c!!a{\in}A \to f\ a\ x) \longleftrightarrow (\forall\, b{\in}c\ `\ A.\ \mathbb{F}_{\checkmark\Rightarrow}(f\ (\mathit{inv\text{-}into}\ A\ c\ b)))$›
⟨*proof*⟩

lemma *finite-ticks-fun-ndet-write* [*finite-ticks-fun-simps*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ a\ x)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ c!!a{\in}A \to f\ a\ x)$›
⟨*proof*⟩

Chapter 3

Generalization of the Sequential Composition

3.1 Definition

For the sequential composition, the generalization seems quite straightforward. In a nutshell, we just replace Q with $Q\ r$ in the definition of P **;** Q since Q is now of type $'r \Rightarrow ('a, 'r)\ process_{ptick}$ (instead of $('a, 'r)\ process_{ptick}$).

lift-definition Seq_{ptick} ::
‹$[('a, 'r)\ process_{ptick}, 'r \Rightarrow ('a, 'r)\ process_{ptick}] \Rightarrow ('a, 'r)\ process_{ptick}$› (**infixl** ‹;✓› 74)
is ‹$\lambda P\ Q.\ (\{(t, X)\ |t\ X.\ (t, X \cup range\ tick) \in \mathcal{F}\ P \wedge tF\ t\} \cup$
$\{(t\ @\ u, X)\ |t\ u\ r\ X.\ t\ @\ [✓(r)] \in \mathcal{T}\ P \wedge (u, X) \in \mathcal{F}\ (Q\ r)\} \cup$
$\{(t, X).\ t \in \mathcal{D}\ P\},$
$\mathcal{D}\ P \cup \{t\ @\ u\ |t\ u\ r.\ t\ @\ [✓(r)] \in \mathcal{T}\ P \wedge u \in \mathcal{D}\ (Q\ r)\})$›
⟨*proof*⟩

Except that this is not a fully satisfactory definition yet. Indeed, here, the right-hand side argument must produce processes whose terminations keep the same type. In other words, Q is of type $'r \Rightarrow ('a, 'r)\ process_{ptick}$ while we would like to have in full generality $'r \Rightarrow ('a, 's)\ process_{ptick}$. The final definition given below is not immediate, and involves a precise understanding of the behaviour of the sequential composition.

3.1.1 Preliminaries

The first key for generalizing the definition is to see that $map\ (ev \circ of\text{-}ev)$ allows for changing the type of termination in tick-free traces.

lemma *tickFree-map-ev-of-ev-same-type-is* : ‹$tF\ t \Longrightarrow map\ (ev \circ of\text{-}ev)\ t = t$›
— In this case the type of termination remains unchanged.
⟨*proof*⟩

lemma *tickFree-map-ev-of-ev-eq-iff* :
‹*tF t* ⟹ *map* (*ev* ∘ *of-ev*) *t* = *t'* ⟹ *t* = *map* (*ev* ∘ *of-ev*) *t'*›
⟨*proof*⟩

lemma *tickFree-map-ev-of-ev-inj* :
‹*tF t* ⟹ *tF t'* ⟹ *map* (*ev* ∘ *of-ev*) *t* = *map* (*ev* ∘ *of-ev*) *t'* ⟷ *t* = *t'*›
⟨*proof*⟩

lemma *map-ev-of-ev-map-ev-of-ev* [*simp*] :
‹*map* (*ev* ∘ *of-ev*) (*map* (*ev* ∘ *of-ev*) *t*) = *map* (*ev* ∘ *of-ev*) *t*› ⟨*proof*⟩

lemma *map-ev-of-ev-map-ev-of-ev-simplified* [*simp*] :
‹*map* (*ev* ∘ *of-ev* ∘ (*ev* ∘ *of-ev*)) *t* = *map* (*ev* ∘ *of-ev*) *t*› ⟨*proof*⟩

lemma *tickFree-map-ev-of-ev-eq-imp-ev-mem-iff* :
‹*tF t'* ⟹ *t* = *map* (*ev* ∘ *of-ev*) *t'* ⟹ *ev a* ∈ *set t* ⟷ *ev a* ∈ *set t'*›
⟨*proof*⟩

The second key is to understand that $X \cup \mathit{range\ tick}$ can be rewritten as $(\mathit{ev} \circ \mathit{of\text{-}ev})\ `\ (X \cap \mathit{range\ ev}) \cup \mathit{range\ tick}$, and that this second expression also allows for changing the type of termination.

definition *ref-Seq*$_{ptick}$ :: ‹('*a*, '*r*) *event*$_{ptick}$ *set* ⇒ ('*a*, '*s*) *event*$_{ptick}$ *set*›
where ‹*ref-Seq*$_{ptick}$ *X* ≡ (*ev* ∘ *of-ev*) ` (*X* ∩ *range ev*) ∪ *range tick*›

lemma *ref-Seq*$_{ptick}$*-same-type-is* : ‹*ref-Seq*$_{ptick}$ *X* = *X* ∪ *range tick*›
— In this case the type of termination remains unchanged.
⟨*proof*⟩

lemma *mono-ref-Seq*$_{ptick}$: ‹*X* ⊆ *Y* ⟹ *ref-Seq*$_{ptick}$ *X* ⊆ *ref-Seq*$_{ptick}$ *Y*›
⟨*proof*⟩

lemma *ref-Seq*$_{ptick}$*-idem* : ‹*ref-Seq*$_{ptick}$ (*ref-Seq*$_{ptick}$ *X*) = *ref-Seq*$_{ptick}$ *X*›
⟨*proof*⟩

lemma *ref-Seq*$_{ptick}$*-comp-ref-Seq*$_{ptick}$: ‹*ref-Seq*$_{ptick}$ ∘ *ref-Seq*$_{ptick}$ = *ref-Seq*$_{ptick}$›
⟨*proof*⟩

lemma *ref-Seq*$_{ptick}$*-eq-iff* :
‹*ref-Seq*$_{ptick}$ *X* = *ref-Seq*$_{ptick}$ *Y* ⟷ *X* ∩ *range ev* = *Y* ∩ *range ev*›
⟨*proof*⟩

lemma *ref-Seq*$_{ptick}$*-is-map-event*$_{ptick}$*-image* :

‹*ref-Seq*$_{ptick}$ X = *map-event*$_{ptick}$ *id g* ‘ (X ∩ *range ev*) ∪ *range tick*›
— Note that g is free here and does not matter.
⟨*proof*⟩

lemma *ref-Seq*$_{ptick}$*-union-image-ev* :
‹*ref-Seq*$_{ptick}$ (X ∪ *ev* ‘ S) = *ref-Seq*$_{ptick}$ X ∪ *ev* ‘ S›
⟨*proof*⟩

lemma *ref-Seq*$_{ptick}$*-UNIV* : ‹*ref-Seq*$_{ptick}$ *UNIV* = *UNIV*›
⟨*proof*⟩

3.1.2 Formal Definition

definition *div-Seq*$_{ptick}$ ::
‹[('*a*, '*r*) *process*$_{ptick}$, '*r* ⇒ ('*a*, '*s*) *process*$_{ptick}$] ⇒ ('*a*, '*s*) *trace*$_{ptick}$ *set*›
where ‹*div-Seq*$_{ptick}$ P Q ≡
{*map* (*ev* ∘ *of-ev*) t @ u |t u. t ∈ $\mathcal{D}$ P ∧ *tF* t ∧ *ftF* u} ∪
{*map* (*ev* ∘ *of-ev*) t @ u |t u r. t @ [✓(r)] ∈ $\mathcal{T}$ P ∧ *tF* t ∧ u ∈ $\mathcal{D}$ (Q r)}›

definition *fail-Seq*$_{ptick}$ ::
‹[('*a*, '*r*) *process*$_{ptick}$, '*r* ⇒ ('*a*, '*s*) *process*$_{ptick}$] ⇒ ('*a*, '*s*) *failure*$_{ptick}$ *set*›
where ‹*fail-Seq*$_{ptick}$ P Q ≡
{(*map* (*ev* ∘ *of-ev*) t, X) |t X. (t, *ref-Seq*$_{ptick}$ X) ∈ $\mathcal{F}$ P ∧ *tF* t} ∪
{(*map* (*ev* ∘ *of-ev*) t @ u, X) |t u r X. t @ [✓(r)] ∈ $\mathcal{T}$ P ∧ *tF* t ∧ (u, X)
∈ $\mathcal{F}$ (Q r)} ∪
{(*map* (*ev* ∘ *of-ev*) t @ u, X) |t u X. t ∈ $\mathcal{D}$ P ∧ *tF* t ∧ *ftF* u}›
— *tF* t is trivial when t @ [✓(r)] ∈ $\mathcal{T}$ P, but we add it for proof automation

lift-definition *Seq*$_{ptick}$ ::
‹[('*a*, '*r*) *process*$_{ptick}$, '*r* ⇒ ('*a*, '*s*) *process*$_{ptick}$] ⇒ ('*a*, '*s*) *process*$_{ptick}$› (**infixl**
‹;✓› 74)
is ‹λP Q. (*fail-Seq*$_{ptick}$ P Q, *div-Seq*$_{ptick}$ P Q)›
⟨*proof*⟩

3.2 Projections

lemma *F-Seq*$_{ptick}$: ‹$\mathcal{F}$ (P ;✓ Q) = *fail-Seq*$_{ptick}$ P Q›
⟨*proof*⟩

lemma *D-Seq*$_{ptick}$: ‹$\mathcal{D}$ (P ;✓ Q) = *div-Seq*$_{ptick}$ P Q›
⟨*proof*⟩

lemma *T-Seq*$_{ptick}$*-bis* :
‹$\mathcal{T}$ (P ;✓ Q) = {*map* (*ev* ∘ *of-ev*) t |t. (t, *range tick*) ∈ $\mathcal{F}$ P ∧ *tF* t} ∪
{*map* (*ev* ∘ *of-ev*) t @ u |t u r. t @ [✓(r)] ∈ $\mathcal{T}$ P ∧ *tF* t ∧ u ∈ $\mathcal{T}$ (Q
r)} ∪

$\{map\ (ev \circ of\text{-}ev)\ t\ @\ u\ |t\ u.\ t \in \mathcal{D}\ P \wedge tF\ t \wedge ftF\ u\}\rangle$
⟨*proof*⟩

lemma *T-Seq*$_{ptick}$:
$\langle \mathcal{T}\ (P\ ;_{\checkmark}\ Q) = \{map\ (ev \circ of\text{-}ev)\ t\ |t.\ t \in \mathcal{T}\ P \wedge tF\ t\} \cup$
$\{map\ (ev \circ of\text{-}ev)\ t\ @\ u\ |t\ u\ r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge tF\ t \wedge u \in \mathcal{T}\ (Q\ r)\} \cup$
$\{map\ (ev \circ of\text{-}ev)\ t\ @\ u\ |t\ u.\ t \in \mathcal{D}\ P \wedge tF\ t \wedge ftF\ u\}\rangle$
— Often easier to use
⟨*proof*⟩

lemmas *Seq*$_{ptick}$*-projs* = *F-Seq*$_{ptick}$ *D-Seq*$_{ptick}$ *T-Seq*$_{ptick}$ *fail-Seq*$_{ptick}$*-def div-Seq*$_{ptick}$*-def*

lemma *mono-Seq*$_{ptick}$*-eq* : $\langle P\ ;_{\checkmark}\ Q = P'\ ;_{\checkmark}\ Q' \rangle$ **if** $* : \langle P = P' \rangle\ \langle \bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow Q\ r = Q'\ r \rangle$
for $P\ P' :: \langle ('a, 'r)\ process_{ptick} \rangle$ **and** $Q\ Q' :: \langle 'r \Rightarrow ('a, 's)\ process_{ptick} \rangle$
⟨*proof*⟩

Note that this definition allowing for changing the type of termination is actually a generalization of the first idea that we mentioned at the beginning. Indeed, when we enforce the type of P and Q to be $('a, 'r)\ process_{ptick}$ and $'r \Rightarrow ('a, 's)\ process_{ptick}$ respectively, the projections can be rewritten as follows.

lemma *F-Seq*$_{ptick}$*-same-type* :
$\langle \mathcal{F}\ (P\ ;_{\checkmark}\ Q) = \{(t, X)\ |t\ X.\ (t, X \cup range\ tick) \in \mathcal{F}\ P \wedge tF\ t\} \cup$
$\{(t\ @\ u, X)\ |t\ u\ r\ X.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge (u, X) \in \mathcal{F}\ (Q\ r)\} \cup$
$\{(t, X).\ t \in \mathcal{D}\ P\}\rangle$
⟨*proof*⟩

lemma *D-Seq*$_{ptick}$*-same-type* : $\langle \mathcal{D}\ (P\ ;_{\checkmark}\ Q) = \mathcal{D}\ P \cup \{t\ @\ u\ |t\ u\ r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge u \in \mathcal{D}\ (Q\ r)\}\rangle$
⟨*proof*⟩

lemma *T-Seq*$_{ptick}$*-same-type-bis* :
$\langle \mathcal{T}\ (P\ ;_{\checkmark}\ Q) = \{t.\ (t, range\ tick) \in \mathcal{F}\ P \wedge tF\ t\} \cup$
$\{t\ @\ u\ |t\ u\ r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge u \in \mathcal{T}\ (Q\ r)\} \cup$
$\mathcal{D}\ P\rangle$
⟨*proof*⟩

lemma *T-Seq*$_{ptick}$*-same-type* :
$\langle \mathcal{T}\ (P\ ;_{\checkmark}\ Q) = \{t \in \mathcal{T}\ P.\ tF\ t\} \cup \{t\ @\ u\ |t\ u\ r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge u \in \mathcal{T}\ (Q\ r)\} \cup \mathcal{D}\ P\rangle$
— Often easier to use
⟨*proof*⟩

lemmas *Seq$_{ptick}$-same-type-projs = F-Seq$_{ptick}$-same-type D-Seq$_{ptick}$-same-type T-Seq$_{ptick}$-same-type*

Chapter 4

Generalization of the Synchronization Product

4.1 Trace Interleaving

4.1.1 Motivation

The notion of trace interleaving found in `HOL-CSP` does not allow us to precisely handle termination. Indeed, as soon as $r \neq s$, we cannot have t *setinterleaves* (([✓(r)], [✓(s)]), *range tick* ∪ *ev* ‘ *A*).

lemma ‹$r \neq s \Longrightarrow \neg\ t$ *setinterleaves* (([✓(r)], [✓(s)]), *range tick* ∪ *ev* ‘ *A*)› ⟨*proof*⟩

The actual issue of this previous definition is that no distinction is done between the “regular” events (like *ev a*) and the terminations (like ✓(*r*)). Here, while we still want the same behaviour for regular events, we want instead the interleaving of ✓(*r*) and ✓(*s*) to be ✓((*r*, *s*)). But we would also like this interleaving to generalize the old one, i.e. be able to prevent sometimes two ticks from being combined. Our solution is therefore to rely on a parameter: *tick-join* of type $'r \Rightarrow 's \Rightarrow 't$ *option* whose role is to specify how two ticks can be combined (or not).

bundle *option-type-syntax*
begin

no-notation *floor* (‹(‹*open-block notation*=‹*mixfix floor*››⌊-⌋)›)
no-notation *ceiling* (‹(‹*open-block notation*=‹*mixfix ceiling*››⌈-⌉)›)

notation *Some* (‹(‹*open-block notation*=‹*mixfix Some*››⌊-⌋)›)
notation *the* (‹(‹*open-block notation*=‹*mixfix the*››⌈-⌉)›)
notation *None* (‹◇›)

end

unbundle *option-type-syntax*

4.1.2 Definition

type-synonym $('a, 'r, 's, 't)\ setinterleaving_{ptick}\text{-}args =$
$\langle('r \Rightarrow 's \Rightarrow 't\ option) \times ('a, 'r)\ trace_{ptick} \times 'a\ set \times ('a, 's)\ trace_{ptick}\rangle$

fun $setinterleaving_{ptick}$ ::
$\langle('a, 'r, 's, 't)\ setinterleaving_{ptick}\text{-}args \Rightarrow ('a, 't)\ trace_{ptick}\ set\rangle$
where $Nil\text{-}setinterleaving_{ptick}\text{-}Nil$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, [], A, []) = \{[]\}\rangle$

| $ev\text{-}setinterleaving_{ptick}\text{-}Nil$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, ev\ a \#\ u, A, []) =$
$(\ if\ a \in A\ then\ \{\}$
$\ else\ \{ev\ a \#\ t|\ t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, u, A, [])\}\rangle$
| $tick\text{-}setinterleaving_{ptick}\text{-}Nil$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, \checkmark(r) \#\ u, A, []) = \{\}\rangle$

| $Nil\text{-}setinterleaving_{ptick}\text{-}ev$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, [], A, ev\ b \#\ v) =$
$(\ if\ b \in A\ then\ \{\}$
$\ else\ \{ev\ b \#\ t|\ t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, [], A, v)\}\rangle$
| $Nil\text{-}setinterleaving_{ptick}\text{-}tick$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, [], A, \checkmark(s) \#\ v) = \{\}\rangle$

| $ev\text{-}setinterleaving_{ptick}\text{-}ev$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, ev\ a \#\ u, A, ev\ b \#\ v) =$
$(\ if\ a \in A$
$\ then\quad if\ b \in A$
$\qquad then\quad if\ a = b$
$\qquad\quad then\ \{ev\ a \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, u, A, v)\}$
$\qquad\quad else\ \{\}$
$\qquad else\ \{ev\ b \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, ev\ a \#\ u, A, v)\}$
$\quad else\quad if\ b \in A$
$\qquad then\ \{ev\ a \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, u, A, ev\ b \#\ v)\}$
$\qquad else\ \{ev\ a \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, u, A, ev\ b \#\ v)\} \cup$
$\qquad\quad \{ev\ b \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, ev\ a \#\ u, A, v)\})\rangle$
| $ev\text{-}setinterleaving_{ptick}\text{-}tick$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, ev\ a \#\ u, A, \checkmark(s) \#\ v) =$
$(\ if\ a \in A\ then\ \{\}$
$\ else\ \{ev\ a \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, u, A, \checkmark(s) \#\ v)\})\rangle$
| $tick\text{-}setinterleaving_{ptick}\text{-}ev$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, \checkmark(r) \#\ u, A, ev\ b \#\ v) =$
$(\ if\ b \in A\ then\ \{\}$
$\ else\ \{ev\ b \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, \checkmark(r) \#\ u, A, v)\})\rangle$
| $tick\text{-}setinterleaving_{ptick}\text{-}tick$:
$\langle setinterleaving_{ptick}\ (tick\text{-}join, \checkmark(r) \#\ u, A, \checkmark(s) \#\ v) =$
$(case\ tick\text{-}join\ r\ s$
$\ of\ \lfloor r\text{-}s \rfloor \Rightarrow \{\checkmark(r\text{-}s) \#\ t\ |t.\ t \in setinterleaving_{ptick}\ (tick\text{-}join, u, A, v)\}$
$\quad |\ \Diamond \Rightarrow \{\})\rangle$

lemmas $setinterleaving_{ptick}$-*induct*
[*case-names Nil-setinterleaving*$_{ptick}$-*Nil ev-setinterleaving*$_{ptick}$-*Nil*
tick-setinterleaving$_{ptick}$-*Nil Nil-setinterleaving*$_{ptick}$-*ev*
Nil-setinterleaving$_{ptick}$-*tick ev-setinterleaving*$_{ptick}$-*ev*
ev-setinterleaving$_{ptick}$-*tick tick-setinterleaving*$_{ptick}$-*ev*
tick-setinterleaving$_{ptick}$-*tick*,
induct type: *setinterleaving*$_{ptick}$-*args*] = *setinterleaving*$_{ptick}$.*induct*

lemma *Cons-setinterleaving*$_{ptick}$-*Nil* :
‹*setinterleaving*$_{ptick}$ (*tick-join*, *e* # *u*, *A*, []) =
(*case e of ev a* ⇒
(*if a* ∈ *A then* {}
else {*ev a* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, [])})
| ✓(*r*) ⇒ {})›
⟨*proof*⟩

lemma *Nil-setinterleaving*$_{ptick}$-*Cons* :
‹*setinterleaving*$_{ptick}$ (*tick-join*, [], *A*, *e* # *v*) =
(*case e of ev a* ⇒
(*if a* ∈ *A then* {}
else {*ev a* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, [], *A*, *v*)})
| ✓(*r*) ⇒ {})›
⟨*proof*⟩

lemma *Cons-setinterleaving*$_{ptick}$-*Cons* :
‹*setinterleaving*$_{ptick}$ (*tick-join*, *e* # *u*, *A*, *f* # *v*) =
(*case e of ev a* ⇒
(*case f of ev b* ⇒
if a ∈ *A*
then *if b* ∈ *A*
then *if a* = *b*
then {*ev a* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, *v*)}
else {}
else {*ev b* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *ev a* # *u*, *A*, *v*)}
else *if b* ∈ *A*
then {*ev a* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, *ev b* # *v*)}
else {*ev a* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, *ev b* # *v*)} ∪
{*ev b* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *ev a* # *u*, *A*, *v*)}
| ✓(*s*) ⇒ *if a* ∈ *A then* {}
else {*ev a* # *t* |*t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, ✓(*s*)
v)})
| ✓(*r*) ⇒
(*case f of ev b* ⇒
if b ∈ *A then* {}
else {*ev b* # *t*| *t*. *t* ∈ *setinterleaving*$_{ptick}$ (*tick-join*, ✓(*r*) # *u*, *A*, *v*)}
| ✓(*s*) ⇒

(*case tick-join r s of* $\lfloor$*r-s*$\rfloor$ $\Rightarrow$
{✓(*r-s*) # *t* |*t*. *t* $\in$ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, *v*)}
| ◊ $\Rightarrow$ {}))))›
⟨*proof*⟩

lemmas *setinterleaving*$_{ptick}$*-simps* =
Cons-setinterleaving$_{ptick}$*-Nil Nil-setinterleaving*$_{ptick}$*-Cons Cons-setinterleaving*$_{ptick}$*-Cons*

abbreviation *setinterleaves*$_{ptick}$ ::
‹[('*a*, '*t*) *trace*$_{ptick}$, '*r* $\Rightarrow$ '*s* $\Rightarrow$ '*t option*,
('*a*, '*r*) *trace*$_{ptick}$, ('*a*, '*s*) *trace*$_{ptick}$, '*a set*] $\Rightarrow$ *bool*›
(‹(- /(*setinterleaves*✓)-/ '(()'(-, -')(), -'))› [63,0,0,0,0] 64)
where ‹*t setinterleaves*✓*tick-join* ((*u*, *v*), *A*) ≡
t $\in$ *setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, *v*)›

4.1.3 First Properties

First of all: this formalization may seem tricky, but is actually a generalization of the old setup.

theorem *setinterleaves-is-setinterleaves*$_{ptick}$:
‹*t setinterleaves* ((*u*, *v*), *range tick* ∪ *ev* ‘ *A*) ⟷
t setinterleaves✓λ*r s*. *if r* = *s then* $\lfloor$*r*$\rfloor$ *else* ◊ ((*u*, *v*), *A*)›
for *t* :: ‹('*a*, '*r*) *trace*$_{ptick}$›
⟨*proof*⟩

corollary *setinterleaves-is-setinterleaves*$_{ptick}$*-unit* :
‹*t setinterleaves* ((*u*, *v*), *insert* ✓ (*ev* ‘ *A*)) ⟷
t setinterleaves✓λ*r s*. $\lfloor$*r*$\rfloor$ ((*u*, *v*), *A*)› (**is** ‹*?lhs* ⟷ *?rhs*›)
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-sym* :
— Of course not suitable for simplifier.
‹*t setinterleaves*✓λ*s r*. *tick-join r s* ((*v*, *u*), *A*) ⟷
t setinterleaves✓λ*r s*. *tick-join r s* ((*u*, *v*), *A*)›
⟨*proof*⟩

lemma *setinterleaves*$_{Pair}$*-UNIV-iff* :
‹*t setinterleaves*✓λ*r s*. $\lfloor$(*r*, *s*)$\rfloor$ ((*u*, *v*), *UNIV*) ⟷
u = *map* (*map-event*$_{ptick}$ *id fst*) *t* ∧
v = *map* (*map-event*$_{ptick}$ *id snd*) *t*› **for** *t* :: ‹('*a*, '*r* × '*s*) *trace*$_{ptick}$›

⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-empty* :
‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), {}) ⟹
ev a ∈ *set t* ⟷ *ev a* ∈ *set u* ∨ *ev a* ∈ *set v*›
for *u* :: ‹('*a*, '*r*) *trace*$_{ptick}$ ›
⟨*proof*⟩

lemma *tickFree-setinterleaves*$_{ptick}$*-any-tick-join* :
‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟷
t setinterleaves$_{\checkmark\,tick\text{-}join'}$ ((*u*, *v*), *A*)›
if ‹*tF t* ∨ *tF u* ∨ *tF v*›
⟨*proof*⟩

lemma *tickFree-setinterleaves*$_{ptick}$*-iff* :
‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *tF t* ⟷ *tF u* ∧ *tF v*›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-tickFree-imp* :
‹*tF u* ∨ *tF v* ⟹ *t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *tF t* ∧ *tF u* ∧ *tF v*›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-NilL-iff* :
‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ (([], *v*), *A*) ⟷
tF v ∧ *set v* ∩ *ev* ‘ *A* = {} ∧ *t* = *map ev* (*map of-ev v*)›
for *tick-join* :: ‹'*r* ⇒ '*s* ⇒ '*t option*›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-NilR-iff* :
‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*u*, []), *A*) ⟷
tF u ∧ *set u* ∩ *ev* ‘ *A* = {} ∧ *t* = *map ev* (*map of-ev u*)›
for *tick-join* :: ‹'*r* ⇒ '*s* ⇒ '*t option*›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-subsetL* :
‹*tF t* ⟹ {*a*. *ev a* ∈ *set u*} ⊆ *A* ⟹
t setinterleaves$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
t = *map ev* (*map of-ev v*)›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-subsetR* :
‹*tF t* ⟹ {*a*. *ev a* ∈ *set v*} ⊆ *A* ⟹

t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$
t = *map ev* (*map of-ev u*)›
⟨*proof*⟩

lemma *Nil-setinterleaves*$_{ptick}$:
‹[] *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow u = [] \wedge v = []$›
⟨*proof*⟩

lemma *front-tickFree-setinterleaves*$_{ptick}$*-iff* :
‹t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$ *ftF* $t \longleftrightarrow$ *ftF* $u \wedge$ *ftF* v›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-snoc-notinL* :
‹t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow a \notin A \Longrightarrow$
t @ [*ev a*] *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u$ @ [*ev a*], $v),\ A)$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-snoc-notinR* :
‹t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow a \notin A \Longrightarrow$
t @ [*ev a*] *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $(u,\ v$ @ [*ev a*]$),\ A)$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-snoc-inside* :
‹t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow a \in A \Longrightarrow$
t @ [*ev a*] *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u$ @ [*ev a*], v @ [*ev a*]$),\ A)$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-snoc-tick* :
‹t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$ *tick-join* $r\ s = \lfloor r\text{-}s \rfloor \Longrightarrow$
t @ [✓(r-s)] *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u$ @ [✓(r)], v @ [✓(s)]$),\ A)$›
⟨*proof*⟩

lemma *Cons-tick-setinterleaves*$_{ptick}E$:
‹✓(r-s) # t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$
$(\bigwedge u'\ v'\ r\ s.\ [\![$*tick-join* $r\ s = \lfloor r\text{-}s \rfloor$; $u = \checkmark(r)\ \#\ u'$; $v = \checkmark(s)\ \#\ v'$;
t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v'),\ A)]\!] \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*›
⟨*proof*⟩

lemma *Cons-ev-setinterleaves*$_{ptick}E$:
‹*ev a* # t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$
$(\bigwedge u'.\ a \notin A \Longrightarrow u =$ *ev a* # $u' \Longrightarrow t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v),\ A) \Longrightarrow$

thesis) $\Longrightarrow$
($\bigwedge v'$. $a \notin A \Longrightarrow v = ev\ a\ \#\ v' \Longrightarrow t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v'),\ A) \Longrightarrow$ *thesis*) $\Longrightarrow$
($\bigwedge u'\ v'$. $a \in A \Longrightarrow u = ev\ a\ \#\ u' \Longrightarrow v = ev\ a\ \#\ v' \Longrightarrow$
t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v'),\ A) \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*›
⟨*proof*⟩

lemma *rev-setinterleaves*$_{ptick}$*-rev-rev-iff* :
‹*rev* t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((rev\ u,\ rev\ v),\ A)$
$\longleftrightarrow$ t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A)$›
for u :: ‹$('a,\ 'r)$ *trace*$_{ptick}$› **and** v :: ‹$('a,\ 's)$ *trace*$_{ptick}$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-preserves-ev-notin-set* :
‹⟦$ev\ a \notin set\ u$; $ev\ a \notin set\ v$; t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A)$⟧ $\Longrightarrow$ $ev\ a \notin set\ t$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-inj-preserves-tick-notin-set* :
‹⟦*tick-join* $r\ s = \lfloor r\text{-}s \rfloor$; $\checkmark(r) \notin set\ u \vee \checkmark(s) \notin set\ v$;
t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A)$⟧ $\Longrightarrow$ $\checkmark(r\text{-}s) \notin set\ t$›
— This is a weakened injectivity property.
if *inj-tick-join* : ‹$\bigwedge r'\ s'$. *tick-join* $r'\ s' = \lfloor r\text{-}s \rfloor \Longrightarrow r' = r \wedge s' = s$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-preserves-ev-inside-set* :
‹⟦$ev\ a \in set\ u$; $ev\ a \in set\ v$; t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A)$⟧ $\Longrightarrow$ $ev\ a \in set\ t$›
⟨*proof*⟩

lemma *ev-notin-both-sets-imp-empty-setinterleaving*$_{ptick}$:
‹⟦$ev\ a \in set\ u \wedge ev\ a \notin set\ v \vee ev\ a \notin set\ u \wedge ev\ a \in set\ v$; $a \in A$⟧ $\Longrightarrow$
setinterleaving$_{ptick}$ $(tick\text{-}join,\ u,\ A,\ v) = \{\}$›
⟨*proof*⟩

lemma *setinterleaves*$_{ptick}$*-snoc-tick-snoc-tickE*:
‹($\bigwedge t'$ *r-s*. *tick-join* $r\ s = \lfloor r\text{-}s \rfloor \Longrightarrow t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$
$t = t'$ @ $[\checkmark(r\text{-}s)] \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*›
if ‹t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u$ @ $[\checkmark(r)],\ v$ @ $[\checkmark(s)]),\ A)$›
⟨*proof*⟩

lemma *snoc-tick-setinterleaves*$_{ptick}E$:
‹($\bigwedge u'\ v'\ r\ s$. ⟦*tick-join* $r\ s = \lfloor r\text{-}s \rfloor$; t *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v'),\ A)$;
$u = u'$ @ $[\checkmark(r)]$; $v = v'$ @ $[\checkmark(s)]$⟧ $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*›
if ‹t @ $[\checkmark(r\text{-}s)]$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A)$›

⟨*proof*⟩

4.1.4 Lengths

lemma *length-setinterleaves$_{ptick}$-eq-sum-minus-filterL* :
‹*t setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v), A) ⟹*
length t = length u + length v − length (filter (λe. e ∈ range tick ∪ ev ‘ A) u)›
⟨*proof*⟩

lemma *length-setinterleaves$_{ptick}$-eq-sum-minus-filterR* :
‹*t setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v), A) ⟹*
length t = length u + length v − length (filter (λe. e ∈ range tick ∪ ev ‘ A) v)›
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-eq-length* :
‹*t setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v), A) ⟹*
t′ setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v), A) ⟹ length t = length t′›
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-imp-lengthLR-le* :
‹*t setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v), A) ⟹*
length u ≤ length t ∧ length v ≤ length t›
⟨*proof*⟩

4.1.5 Trace Prefix Interleaving

We start with versions involving (@) before giving corollaries about the prefix ordering on traces.

lemma *setinterleaves$_{ptick}$-appendL* :
‹*t setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u1 @ u2, v), A) ⟹*
∃ t1 t2 v1 v2. t = t1 @ t2 ∧ v = v1 @ v2 ∧
t1 setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u1, v1), A) ∧
t2 setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u2, v2), A)›
⟨*proof*⟩

corollary *setinterleaves$_{ptick}$-appendR* :
‹*∃ t1 t2 u1 u2. t = t1 @ t2 ∧ u = u1 @ u2 ∧*
t1 setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u1, v1), A) ∧
t2 setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u2, v2), A)›
if ‹*t setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v1 @ v2), A)*›
⟨*proof*⟩

lemma *append-setinterleaves$_{ptick}$* :
‹*t1 @ t2 setinterleaves$_{\checkmark\, tick\text{-}join}$ ((u, v), A) ⟹*

$\exists\, u1\ u2\ v1\ v2.\ u = u1\ @\ u2 \wedge v = v1\ @\ v2 \wedge$
$t1\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u1,\ v1),\ A) \wedge$
$t2\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u2,\ v2),\ A)\rangle$
⟨*proof*⟩

corollary $setinterleaves_{ptick}$-*le-prefixL* :
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ A) \Longrightarrow u' \leq u \Longrightarrow$
$\exists\, t' \leq t.\ \exists\, v' \leq v.\ t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u',\ v'),\ A)\rangle$
⟨*proof*⟩

corollary $setinterleaves_{ptick}$-*le-prefixR* :
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ A) \Longrightarrow v' \leq v \Longrightarrow$
$\exists\, t' \leq t.\ \exists\, u' \leq u.\ t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u',\ v'),\ A)\rangle$
⟨*proof*⟩

corollary *le-prefix-*$setinterleaves_{ptick}$:
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ A) \Longrightarrow t' \leq t \Longrightarrow$
$\exists\, u' \leq u.\ \exists\, v' \leq v.\ t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u',\ v'),\ A)\rangle$
⟨*proof*⟩

lemma $setinterleaves_{ptick}$-*less-prefixL* :
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ A) \Longrightarrow u' < u \Longrightarrow$
$\exists\, t'\ v'.\ t' < t \wedge v' \leq v \wedge t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u',\ v'),\ A)\rangle$
⟨*proof*⟩

corollary $setinterleaves_{ptick}$-*less-prefixR* :
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ A) \Longrightarrow v' < v \Longrightarrow$
$\exists\, t'\ u'.\ t' < t \wedge u' \leq u \wedge t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u',\ v'),\ A)\rangle$
⟨*proof*⟩

lemma $setinterleaves_{ptick}$-*le-prefixLR* :
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ A) \Longrightarrow u' \leq u \Longrightarrow v' \leq v \Longrightarrow$
$(\exists\, t' \leq t.\ \exists\, v'' \leq v'.\ t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u',\ v''),\ A)) \vee$
$(\exists\, t' \leq t.\ \exists\, u'' \leq u'.\ t'\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u'',\ v'),\ A))\rangle$
⟨*proof*⟩

4.1.6 Hiding Events

lemma $setinterleaves_{ptick}$-*trace-hide* :
$\langle t\ setinterleaves_{\checkmark\, tick\text{-}join}\ ((u,\ v),\ S) \Longrightarrow$

trace-hide t (ev ‘ A) setinterleaves$_{\checkmark\, tick\text{-}join}$
((trace-hide u (ev ‘ A), trace-hide v (ev ‘ A)), S)›
⟨*proof*⟩

lemma *trace-hide-map-map-event*$_{ptick}$:
‹*trace-hide (map (map-event*$_{ptick}$ *f g) t) S =*
map (map-event$_{ptick}$ *f g) (trace-hide t (map-event*$_{ptick}$ *f g −‘ S))*›
⟨*proof*⟩

lemma *tickFree-trace-hide-map-ev-comp-of-ev* :
‹*tF t* ⟹ *trace-hide (map (ev* ∘ *of-ev) t) (ev ‘ A) =*
map (ev ∘ *of-ev) (trace-hide t (ev ‘ A))*›
⟨*proof*⟩

lemma *tickFree-disjoint-setinterleaves*$_{ptick}$*-appendL* :
‹*tF u1* ⟹ {*a. ev a* ∈ *set u1*} ∩ *A* = {} ⟹ *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u2*, *v*), *A*)
⟹ *map (ev* ∘ *of-ev) u1* @ *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*)›
⟨*proof*⟩

corollary *tickFree-disjoint-setinterleaves*$_{ptick}$*-appendR* :
‹⟦*tF v1*; {*a. ev a* ∈ *set v1*} ∩ *A* = {}; *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v2*), *A*)⟧
⟹ *map (ev* ∘ *of-ev) v1* @ *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v1* @ *v2*), *A*)›
⟨*proof*⟩

lemma *tickFree-disjoint-setinterleaves*$_{ptick}$*-append-tailL* :
‹*t* @ *map (ev* ∘ *of-ev) u2 setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*)›
if ‹*tF u2*› ‹{*a. ev a* ∈ *set u2*} ∩ *A* = {}› ‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1*, *v*), *A*)›
⟨*proof*⟩

corollary *tickFree-disjoint-setinterleaves*$_{ptick}$*-append-tailR* :
‹⟦*tF v2*; {*a. ev a* ∈ *set v2*} ∩ *A* = {}; *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v1*), *A*)⟧
⟹ *t* @ *map (ev* ∘ *of-ev) v2 setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v1* @ *v2*), *A*)›
⟨*proof*⟩

lemma *disjoint-trace-hide-setinterleaves*$_{ptick}$:
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$
((trace-hide u (ev ‘ A), trace-hide v (ev ‘ A)), S) ⟹
∃ *t′*. *t* = *trace-hide t′ (ev ‘ A)* ∧
t′ setinterleaves$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *S*)› **if** ‹*A* ∩ *S* = {}›
for *t* :: ‹('*a*, '*t*) *trace*$_{ptick}$› **and** *u* :: ‹('*a*, '*r*) *trace*$_{ptick}$› **and** *v* :: ‹('*a*, '*s*) *trace*$_{ptick}$›
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-inj-map-map-event$_{ptick}$-iff-weak* :
‹*map* (*map-event$_{ptick}$ f id*) *t setinterleaves$_{\checkmark\,tick\text{-}join}$*
((*map* (*map-event$_{ptick}$ f id*) *u*, *map* (*map-event$_{ptick}$ f id*) *v*), *f* ‘ *A*) ⟷
t setinterleaves$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), *A*)› **if** ‹*inj f*›
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-inj-map-map-event$_{ptick}$-iff-strong* :
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$*
((*map* (*map-event$_{ptick}$ f id*) *u*, *map* (*map-event$_{ptick}$ f id*) *v*), *f* ‘ *A*) ⟷
(∃ *t'*. *t* = *map* (*map-event$_{ptick}$ f id*) *t'* ∧
t' setinterleaves$_{\checkmark\,tick\text{-}join}$ ((*u*, *v*), *A*))› **if** ‹*inj f*›
— We could probably prove a stronger version with *inj-on f* (*A* ∪ {*a*. *ev a* ∈ *set u* ∨ *ev a* ∈ *set v*}) instead of *inj f*.
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-append-setinterleaves$_{ptick}$* :
‹*t1* @ *t2 setinterleaves$_{\checkmark\,tick\text{-}join}$* ((*u1* @ *u2*, *v1* @ *v2*), *A*)›
if ‹*t1 setinterleaves$_{\checkmark\,tick\text{-}join}$* ((*u1*, *v1*), *A*)›
and ‹*t2 setinterleaves$_{\checkmark\,tick\text{-}join}$* ((*u2*, *v2*), *A*)›
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-set-subsetL* :
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$* ((*u*, *v*), *A*) ⟹
{*a*. *ev a* ∈ *set* (*drop n u*)} ⊆ {*a*. *ev a* ∈ *set* (*drop n t*)}›
⟨*proof*⟩

lemma *setinterleaves$_{ptick}$-set-subsetR* :
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$* ((*u*, *v*), *A*) ⟹
{*a*. *ev a* ∈ *set* (*drop n v*)} ⊆ {*a*. *ev a* ∈ *set* (*drop n t*)}›
⟨*proof*⟩

4.2 Synchronization Product

4.2.1 Definition

definition *super-ref-Sync$_{ptick}$* ::
‹['*r* ⇒ '*s* ⇒ '*t option*, ('*a*, '*r*) *refusal$_{ptick}$*, '*a set*, ('*a*, '*s*) *refusal$_{ptick}$*] ⇒ ('*a*, '*t*) *refusal$_{ptick}$*›

where ‹*super-ref-Sync*$_{ptick}$ *tick-join X-P A X-Q* ≡
$\{ev\ a\ |a.\ ev\ a \in X\text{-}P \wedge ev\ a \in X\text{-}Q \vee (a \in A \wedge (ev\ a \in X\text{-}P \vee ev\ a \in X\text{-}Q))\} \cup$
$\{\checkmark(r\text{-}s)\ |r\ s\ r\text{-}s.\ tick\text{-}join\ r\ s = \lfloor r\text{-}s \rfloor \wedge (\checkmark(r) \in X\text{-}P \vee \checkmark(s) \in X\text{-}Q)\} \cup$
— This is the last addition: since we generalize with the parameter *tick-join*, we must add the following term to refuse the unreachable ticks.
$\{\checkmark(r\text{-}s)\ |r\text{-}s.\ \nexists r\ s.\ tick\text{-}join\ r\ s = \lfloor r\text{-}s \rfloor\}$›

For proving that the invariant *is-process* is preserved, we will need a kind of injectivity for the parameter *tick-join*. We implement this through a **locale**.

locale *Sync*$_{ptick}$*-locale* =
fixes *tick-join* :: ‹$'r \Rightarrow 's \Rightarrow 't$ *option*› (**infixl** ‹$\otimes\checkmark$› *100*)
assumes *inj-tick-join* :
‹$r \otimes\checkmark s = \lfloor r\text{-}s \rfloor \Longrightarrow r' \otimes\checkmark s' = \lfloor r\text{-}s \rfloor \Longrightarrow r' = r \wedge s' = s$›
begin

sublocale *Sync*$_{ptick}$*-locale-sym* : *Sync*$_{ptick}$*-locale* ‹$\lambda s\ r.\ r \otimes\checkmark s$›
⟨*proof*⟩

lift-definition *Sync*$_{ptick}$ ::
‹[($'a$, $'r$) *process*$_{ptick}$, $'a$ *set*, ($'a$, $'s$) *process*$_{ptick}$] ⇒ ($'a$, $'t$) *process*$_{ptick}$›
(‹(- $[\![$-$]\!]_{\checkmark}$ -)› [*70*, *0*, *71*] *70*)
is ‹$\lambda P\ A\ Q.$
$(\{(t, X).\ \exists\ t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q.$
$(t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \wedge$
$t\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A) \wedge$
$X \subseteq super\text{-}ref\text{-}Sync_{ptick}\ (\otimes\checkmark)\ X\text{-}P\ A\ X\text{-}Q\} \cup$
$\{(t\ @\ u, X)\ |t\ u\ t\text{-}P\ t\text{-}Q\ X.$
$ftF\ u \wedge (tF\ t \vee u = []) \wedge t\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A) \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\},$
$\{t\ @\ u\ |t\ u\ t\text{-}P\ t\text{-}Q.$
$ftF\ u \wedge (tF\ t \vee u = []) \wedge t\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A) \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\})$›
⟨*proof*⟩

Here $X \subseteq$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)$ *X-P A X-Q* may seem surprising (instead of for example $X =$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)$ *X-P A X-Q*, closer to the specification of *Sync*). Actually, edge cases in the behaviour of *tick* ensure that a definition with the latter would violate the invariant.

end

abbreviation (**in** *Sync*$_{ptick}$*-locale*) *Inter*$_{ptick}$ ::
‹[($'a$, $'r$) *process*$_{ptick}$, ($'a$, $'s$) *process*$_{ptick}$] ⇒
($'a$, $'t$) *process*$_{ptick}$› (‹(- $|||_{\checkmark}$ -)› [*72*, *73*] *72*)
where ‹$P\ |||_{\checkmark}\ Q \equiv P\ [\![\ \{\}\ ]\!]_{\checkmark}\ Q$›

abbreviation (**in** $Sync_{ptick}$-*locale*) Par_{ptick} ::
‹[('a, 'r) $process_{ptick}$, ('a, 's) $process_{ptick}$] ⇒
('a, 't) $process_{ptick}$› (‹(- $||_{✓}$ -)› [74, 75] 74)
where ‹P $||_{✓}$ Q ≡ P ⟦ $UNIV$ ⟧$_{✓}$ Q›

notation (**in** $Sync_{ptick}$-*locale*) $Sync_{ptick}$-*locale-sym*.$Sync_{ptick}$
(‹(- ⟦-⟧$_{✓sym}$ -)› [70, 0, 71] 70)

notation (**in** $Sync_{ptick}$-*locale*) $Sync_{ptick}$-*locale-sym*.$Inter_{ptick}$
(‹(- |||$_{✓sym}$ -)› [72, 73] 72)

notation (**in** $Sync_{ptick}$-*locale*) $Sync_{ptick}$-*locale-sym*.Par_{ptick}
(‹(- $||_{✓sym}$ -)› [74, 75] 74)

4.2.2 Projections

context $Sync_{ptick}$-*locale* **begin**

lemma *D-*$Sync_{ptick}$′ :
‹$\mathcal{D}$ (P ⟦A⟧$_{✓}$ Q) =
{t @ u |t u *t-P t-Q*.
ftF u ∧ (*tF* t ∨ u = []) ∧ t *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), A) ∧
(*t-P* ∈ $\mathcal{D}$ P ∧ *t-Q* ∈ $\mathcal{T}$ Q ∨ *t-P* ∈ $\mathcal{T}$ P ∧ *t-Q* ∈ $\mathcal{D}$ Q)}›
⟨*proof*⟩

corollary *D-*$Sync_{ptick}$:
— This version is easier to use.
‹$\mathcal{D}$ (P ⟦A⟧$_{✓}$ Q) =
{t @ u |t u *t-P t-Q*.
tF t ∧ *ftF* u ∧ t *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), A) ∧
(*t-P* ∈ $\mathcal{D}$ P ∧ *t-Q* ∈ $\mathcal{T}$ Q ∨ *t-P* ∈ $\mathcal{T}$ P ∧ *t-Q* ∈ $\mathcal{D}$ Q)}›
(**is** ‹- = *?rhs*›)
⟨*proof*⟩

lemma *F-*$Sync_{ptick}$′ :
‹$\mathcal{F}$ (P ⟦A⟧$_{✓}$ Q) =
{(t, X). ∃ *t-P t-Q X-P X-Q*.
(*t-P*, *X-P*) ∈ $\mathcal{F}$ P ∧ (*t-Q*, *X-Q*) ∈ $\mathcal{F}$ Q ∧
t *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), A) ∧
X ⊆ *super-ref-*$Sync_{ptick}$ (⊗✓) *X-P* A *X-Q*} ∪
{(t @ u, X) |t u *t-P t-Q* X.
ftF u ∧ (*tF* t ∨ u = []) ∧ t *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), A) ∧
(*t-P* ∈ $\mathcal{D}$ P ∧ *t-Q* ∈ $\mathcal{T}$ Q ∨ *t-P* ∈ $\mathcal{T}$ P ∧ *t-Q* ∈ $\mathcal{D}$ Q)}›
⟨*proof*⟩

lemma *F-*$Sync_{ptick}$:
‹$\mathcal{F}$ (P ⟦A⟧$_{✓}$ Q) =
{(t, X). ∃ *t-P t-Q X-P X-Q*.

$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ Q \wedge$
t *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$
$X \subseteq$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)$ $X\text{-}P\ A\ X\text{-}Q\} \cup$
$\{(t\ @\ u,\ X)\ |t\ u\ t\text{-}P\ t\text{-}Q\ X.$
$tF\ t \wedge ftF\ u \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}\rangle$
⟨*proof*⟩

lemma *T-Sync*$_{ptick}$$'$:
$\langle \mathcal{T}\ (P\ [\![A]\!]_{\checkmark}\ Q) =$
$\{t.\ \exists\, t\text{-}P\ t\text{-}Q.\ t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)\}$
$\cup$
$\{t\ @\ u\ |t\ u\ t\text{-}P\ t\text{-}Q.$
$ftF\ u \wedge (tF\ t \vee u = []) \wedge$
t *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}\rangle$
⟨*proof*⟩

lemma *T-Sync*$_{ptick}$:
$\langle \mathcal{T}\ (P\ [\![A]\!]_{\checkmark}\ Q) =$
$\{t.\ \exists\, t\text{-}P\ t\text{-}Q.\ t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)\}$
$\cup$
$\{t\ @\ u\ |t\ u\ t\text{-}P\ t\text{-}Q.$
$tF\ t \wedge ftF\ u \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}\rangle$
⟨*proof*⟩

lemmas *Sync*$_{ptick}$*-projs'* = *F-Sync*$_{ptick}$$'$ *D-Sync*$_{ptick}$$'$ *T-Sync*$_{ptick}$$'$
— Classical versions, but the ones below are often more convenient to use.

lemmas *Sync*$_{ptick}$*-projs* = *F-Sync*$_{ptick}$ *D-Sync*$_{ptick}$ *T-Sync*$_{ptick}$

lemma (**in** *Sync*$_{ptick}$*-locale*) *Sync*$_{ptick}$*-same-tick-join-on-strict-ticks-of* :
$\langle$*Sync*$_{ptick}$*-locale.Sync*$_{ptick}$ *tick-join'* $P\ S\ Q = P\ [\![S]\!]_{\checkmark}\ Q\rangle$
if ⟨*Sync*$_{ptick}$*-locale tick-join'*⟩ **and** $\langle \bigwedge r\ s.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow s \in \checkmark\mathbf{s}(Q) \Longrightarrow$ *tick-join'* $r\ s = r \otimes\checkmark\ s\rangle$
⟨*proof*⟩

4.2.3 First Properties

abbreviation *range-tick-join* :: ⟨$'t$ *set*⟩
where ⟨*range-tick-join* $\equiv \{r\text{-}s\ |r\text{-}s\ r\ s.\ r \otimes\checkmark\ s = \lfloor r\text{-}s \rfloor\}\rangle$

lemma *setinterleaves*$_{ptick}$*-imp-set-range-tick-join* :
$\langle t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((u,\ v),\ A) \Longrightarrow$
$\{r\text{-}s.\ \checkmark(r\text{-}s) \in$ *set* $t\} \subseteq$ *range-tick-join*$\rangle$

⟨*proof*⟩

end

lemma
— Of course not suitable for simplifier.
‹*t setinterleaves*$_{✓\lambda s\ r.\ tick\text{-}join\ r\ s}$ $((v,\ u),\ A) \longleftrightarrow$
t setinterleaves$_{✓\lambda r\ s.\ tick\text{-}join\ r\ s}$ $((u,\ v),\ A)$›
⟨*proof*⟩

lemma *super-ref-Sync*$_{ptick}$*-sym* :
— Of course not suitable for simplifier.
‹*super-ref-Sync*$_{ptick}$ $(\lambda s\ r.\ tick\text{-}join\ r\ s)$ *X-Q S X-P* =
super-ref-Sync$_{ptick}$ $(\lambda r\ s.\ tick\text{-}join\ r\ s)$ *X-P S X-Q*›
⟨*proof*⟩

lemma *super-ref-Sync*$_{ptick}$*-mono* :
‹$A \subseteq A' \Longrightarrow$ *X-P* $\subseteq$ *X-P*$' \Longrightarrow$ *X-Q* $\subseteq$ *X-Q*$' \Longrightarrow$
super-ref-Sync$_{ptick}$ *tick-join X-P A X-Q* $\subseteq$
super-ref-Sync$_{ptick}$ *tick-join X-P*$'$ *A*$'$ *X-Q*$'$›
⟨*proof*⟩

context *Sync*$_{ptick}$*-locale* **begin**

lemma *Sync*$_{ptick}$*-sym* : ‹*Q* $[\![A]\!]_{✓sym}$ *P* = *P* $[\![A]\!]_{✓}$ *Q*›
⟨*proof*⟩

lemma *interpretable-inj-on-range-tick-join* :
‹*inj-on g range-tick-join* $\Longrightarrow$
Sync$_{ptick}$*-locale* $(\lambda r\ s.$ *case r* $\otimes$✓ *s of* $\lfloor$*r-s*$\rfloor$ $\Rightarrow$ $\lfloor$*g r-s*$\rfloor$ | $\Diamond \Rightarrow \Diamond)$›
⟨*proof*⟩

lemma *inj-on-map-map-event*$_{ptick}$*-setinterleaves*$_{ptick}$:
‹*t setinterleaves*$_{✓(\otimes✓)}$ $((u,\ v),\ A) \Longrightarrow$
map (*map-event*$_{ptick}$ *id g*) *t*
setinterleaves$_{✓\lambda r\ s.\ case\ r\ \otimes✓\ s\ of\ \lfloor r\text{-}s\rfloor \Rightarrow \lfloor g\ r\text{-}s\rfloor \mid \Diamond \Rightarrow \Diamond}$ $((u,\ v),\ A)$›
(**is** ‹- $\Longrightarrow$ - *setinterleaves*$_{✓\ ?tick\text{-}join'}$ $((u,\ v),\ A)$›)
if *inj-on-g* : ‹*inj-on g range-tick-join*›
⟨*proof*⟩

lemma *vimage-inj-on-subset-super-ref-Sync$_{ptick}$-iff* :
‹*map-event$_{ptick}$ id g −‘ X ⊆*
super-ref-Sync$_{ptick}$ (⊗✓) X-P A X-Q ⟷
X ⊆ super-ref-Sync$_{ptick}$ (λr s. case r ⊗✓ s of ⌊r-s⌋ ⇒ ⌊g r-s⌋ | ◊ ⇒ ◊) X-P A X-Q›
(**is** ‹*?lhs1 ⊆ ?lhs2 ⟷ X ⊆ ?rhs*›)
if *inj-on-g* : ‹*inj-on g range-tick-join*›
⟨*proof*⟩

The two following lemmas are necessary for the proof of continuity.

lemma *finite-setinterleaves$_{ptick}$-tick-join* :
‹*finite {(u, v). t setinterleaves$_{✓(⊗✓)}$ ((u, v), A)}*›
(**is** ‹*finite {(u, v). ?f t u v}*›)
⟨*proof*⟩

lemma *finite-setinterleaves$_{ptick}$-tick-join-Sync$_{ptick}$*:
‹*finite {(t-P, t-Q, u). u setinterleaves$_{✓(⊗✓)}$ ((t-P, t-Q), A) ∧*
(∃ v. t = u @ v ∧ ftF v ∧ (tF u ∨ v = []))}›
(**is** ‹*finite {(t-P, t-Q, u). ?f u t-P t-Q ∧ ?g t u}*›)
⟨*proof*⟩

end

Chapter 5

Some Work on Renaming

unbundle *option-type-syntax*

This chapter contains several developments related to the *Renaming* operator. Some are not directly related to this session and may be moved to `HOL-CSP` or `HOL-CSPM` in the future, while others specifically concern the operator $Sync_{ptick}$-*locale*.$Sync_{ptick}$.

5.1 Tick Swap Operator

We want to define an operator for swapping the values inside termination. Intuitively, we want *TickSwap* (*SKIP* (r, s)) = *SKIP* (s, r).

5.1.1 Preliminaries

Swapping an Event

We start by defining *tick-swap*, which is swapping the values inside termination but only for an event. Then this will be generalized to a trace, a refusal and a failure.

fun *tick-swap* :: ‹$('a, 'r \times 's)$ $event_{ptick}$ ⇒ $('a, 's \times 'r)$ $event_{ptick}$›
  where ‹*tick-swap* $(ev\ a)$ = $ev\ a$›
  | ‹*tick-swap* ✓$((r, s))$ = ✓$((s, r))$›

lemma *tick-swap-tick* : ‹*tick-swap* ✓(*r-s*) = (*case r-s of* (r, s) ⇒ ✓$((s, r))$)›
  ⟨*proof*⟩

lemma *tick-swap-tick-swap* [*simp*] : ‹*tick-swap* (*tick-swap e*) = *e*›
⟨*proof*⟩

lemma *tick-swap-comp-tick-swap* [*simp*] : ‹*tick-swap* ∘ *tick-swap* = *id*›
⟨*proof*⟩

lemma *inj-tick-swap* [*simp*] : ‹*inj tick-swap*›
⟨*proof*⟩

lemma *surj-tick-swap* [*simp*] : ‹*surj tick-swap*›
⟨*proof*⟩

lemma *bij-tick-swap* [*simp*] : ‹*bij tick-swap*›
⟨*proof*⟩

lemma *bij-betw-tick-swap* :
‹*bij-betw tick-swap* (*range ev*) (*range ev*)›
‹*bij-betw tick-swap* (*range tick*) (*range tick*)›
⟨*proof*⟩

lemma *ev-eq-tick-swap-iff* [*simp*] : ‹*ev a* = *tick-swap e* ⟷ *e* = *ev a*›
and *tick-swap-eq-ev-iff* [*simp*] : ‹*tick-swap e* = *ev a* ⟷ *e* = *ev a*›
and *tick-eq-tick-swap-iff* [*simp*] : ‹✓((*r*, *s*)) = *tick-swap e* ⟷ *e* = ✓((*s*, *r*))›
and *tick-swap-eq-tick-iff* [*simp*] : ‹*tick-swap e* = ✓((*r*, *s*)) ⟷ *e* = ✓((*s*, *r*))›
⟨*proof*⟩

Swapping a Trace

fun *trace-tick-swap* :: ‹('*a*, ('*r* × '*s*)) *trace*$_{ptick}$ ⇒ ('*a*, ('*s* × '*r*)) *trace*$_{ptick}$›
where ‹*trace-tick-swap* [] = []›
| ‹*trace-tick-swap* (*ev a* # *t*) = *ev a* # *trace-tick-swap t*›
| ‹*trace-tick-swap* (✓((*r*, *s*)) # *t*) = ✓((*s*, *r*)) # *trace-tick-swap t*›

lemma *trace-tick-swap-tick-Cons* :
‹*trace-tick-swap* (✓(*r-s*) # *t*) = (*case r-s of* (*r*, *s*) ⇒ ✓((*s*, *r*)) # *trace-tick-swap t*)›
⟨*proof*⟩

lemma *trace-tick-swap-def* : ‹*trace-tick-swap* = *map tick-swap*›
⟨*proof*⟩

lemma *trace-tick-swap-append* : ‹*trace-tick-swap* (*t* @ *u*) = *trace-tick-swap t* @ *trace-tick-swap u*›
⟨*proof*⟩

lemma *trace-tick-swap-singl* [*simp*] : ‹*trace-tick-swap* [*e*] = [*tick-swap e*]›
⟨*proof*⟩

lemma *trace-tick-swap-comp-trace-tick-swap* [*simp*] : ‹*trace-tick-swap* ∘ *trace-tick-swap*

$= id\rangle$
⟨*proof*⟩

lemma *trace-tick-swap-trace-tick-swap* [*simp*] : ‹*trace-tick-swap* (*trace-tick-swap t*)
$= t\rangle$
⟨*proof*⟩

lemma *inj-trace-tick-swap* [*simp*] : ‹*inj trace-tick-swap*›
⟨*proof*⟩

lemma *surj-trace-tick-swap* [*simp*] : ‹*surj trace-tick-swap*›
⟨*proof*⟩

lemma *bij-trace-tick-swap* [*simp*] : ‹*bij trace-tick-swap*›
⟨*proof*⟩

lemma *strict-mono-trace-tick-swap* : ‹*strict-mono trace-tick-swap*›
⟨*proof*⟩

lemma *image-trace-tick-swap-min-elems* :
‹*trace-tick-swap* ‘ (*min-elems T*) = *min-elems* (*trace-tick-swap* ‘ *T*)›
⟨*proof*⟩

lemma *Nil-eq-trace-tick-swap-iff* [*iff*] : ‹[] = *trace-tick-swap t* ⟷ *t* = []›
and *trace-tick-swap-eq-Nil-iff* [*iff*] : ‹*trace-tick-swap t* = [] ⟷ *t* = []›
⟨*proof*⟩

lemma *ev-Cons-eq-trace-tick-swap-iff* [*iff*] :
‹*ev a* # *t* = *trace-tick-swap u* ⟷ *u* = *ev a* # *trace-tick-swap t*›
and *trace-tick-swap-eq-ev-Cons-iff* [*iff*] :
‹*trace-tick-swap u* = *ev a* # *t* ⟷ *u* = *ev a* # *trace-tick-swap t*›
⟨*proof*⟩

lemma *tick-Cons-eq-trace-tick-swap-iff* [*iff*] :
‹✓((*r*, *s*)) # *t* = *trace-tick-swap u* ⟷ *u* = ✓((*s*, *r*)) # *trace-tick-swap t*›
and *trace-tick-swap-eq-tick-Cons-iff* [*iff*] :
‹*trace-tick-swap u* = ✓((*r*, *s*)) # *t* ⟷ *u* = ✓((*s*, *r*)) # *trace-tick-swap t*›
⟨*proof*⟩

lemma *snoc-ev-eq-trace-tick-swap-iff* [*iff*] :
‹*t* @ [*ev a*] = *trace-tick-swap u* ⟷ *u* = *trace-tick-swap t* @ [*ev a*]›
and *trace-tick-swap-eq-snoc-ev-iff* [*iff*] :
‹*trace-tick-swap u* = *t* @ [*ev a*] ⟷ *u* = *trace-tick-swap t* @ [*ev a*]›
⟨*proof*⟩

lemma *snoc-tick-eq-trace-tick-swap-iff* [*iff*] :
‹t @ [✓((r, s))] = *trace-tick-swap* $u \longleftrightarrow u$ = *trace-tick-swap* t @ [✓((s, r))]›
and *trace-tick-swap-eq-snoc-tick-iff* [*iff*] :
‹*trace-tick-swap* $u = t$ @ [✓((r, s))] $\longleftrightarrow u$ = *trace-tick-swap* t @ [✓((s, r))]›
⟨*proof*⟩

lemma *trace-tick-swap-eq-ev-ConsE* :
‹*trace-tick-swap* u = *ev* a # $t \Longrightarrow (\bigwedge u'.\ u$ = *ev* a # $u' \Longrightarrow t$ = *trace-tick-swap* $u' \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*›
and *trace-tick-swap-eq-tick-ConsE* :
‹*trace-tick-swap* u = ✓((r, s)) # $t \Longrightarrow (\bigwedge u'.\ u$ = ✓((s, r)) # $u' \Longrightarrow t$ = *trace-tick-swap* $u' \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*›
and *trace-tick-swap-eq-snoc-evE* :
‹*trace-tick-swap* $u = t$ @ [*ev* a] $\Longrightarrow (\bigwedge u'.\ u = u'$ @ [*ev* a] $\Longrightarrow t$ = *trace-tick-swap* $u' \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*›
and *trace-tick-swap-eq-snoc-tickE* :
‹*trace-tick-swap* $u = t$ @ [✓((r, s))] $\Longrightarrow (\bigwedge u'.\ u = u'$ @ [✓((s, r))] $\Longrightarrow t$ = *trace-tick-swap* $u' \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*›
⟨*proof*⟩

lemma *trace-tick-swap-tickFree* :
‹*tF* $t \Longrightarrow$ *trace-tick-swap* t = *map* (*ev* ∘ *of-ev*) t› **for** t :: ‹('*a*, ('*r* × '*s*)) $trace_{ptick}$›
⟨*proof*⟩

lemma *trace-tick-swap-front-tickFree* :
‹*trace-tick-swap* t = (*if tF* t *then map* (*ev* ∘ *of-ev*) t
else map (*ev* ∘ *of-ev*) (*butlast* t) @ [*case last* t *of* ✓((r, s)) ⇒ ✓((s, r))])›
if ‹*ftF* t› **for** t :: ‹('*a*, ('*r* × '*s*)) $trace_{ptick}$›
⟨*proof*⟩

lemma *tickFree-trace-tick-swap-iff* [*simp*] : ‹*tF* (*trace-tick-swap* t) $\longleftrightarrow$ *tF* t›
⟨*proof*⟩

lemma *front-tickFree-trace-tick-swap-iff* [*simp*] : ‹*ftF* (*trace-tick-swap* t) $\longleftrightarrow$ *ftF* t›
⟨*proof*⟩

Swapping a Refusal

definition *refusal-tick-swap* :: ‹('*a*, ('*r* × '*s*)) $refusal_{ptick}$ ⇒ ('*a*, ('*s* × '*r*)) $refusal_{ptick}$›
where ‹*refusal-tick-swap* X = *tick-swap* ‘ X›

lemma *refusal-tick-swap-empty* [*simp*] : ‹*refusal-tick-swap* {} = {}›
⟨*proof*⟩

lemma *refusal-tick-swap-insert* [*simp*] :
‹*refusal-tick-swap* (*insert x X*) = *insert* (*tick-swap x*) (*refusal-tick-swap X*)›
⟨*proof*⟩

lemma *refusal-tick-swap-union* :
‹*refusal-tick-swap* (*X* ∪ *Y*) = *refusal-tick-swap X* ∪ *refusal-tick-swap Y*›
⟨*proof*⟩

lemma *refusal-tick-swap-diff* :
‹*refusal-tick-swap* (*X* − *Y*) = *refusal-tick-swap X* − *refusal-tick-swap Y*›
⟨*proof*⟩

lemma *refusal-tick-swap-inter* :
‹*refusal-tick-swap* (*X* ∩ *Y*) = *refusal-tick-swap X* ∩ *refusal-tick-swap Y*›
⟨*proof*⟩

lemma *refusal-tick-swap-singl* : ‹*refusal-tick-swap* {*e*} = {*tick-swap e*}› ⟨*proof*⟩

lemma *refusal-tick-swap-comp-refusal-tick-swap* [*simp*] :
‹*refusal-tick-swap* ∘ *refusal-tick-swap* = *id*›
⟨*proof*⟩

lemma *refusal-tick-swap-refusal-tick-swap* [*simp*] :
‹*refusal-tick-swap* (*refusal-tick-swap X*) = *X*›
⟨*proof*⟩

lemma *inj-refusal-tick-swap* [*simp*] : ‹*inj refusal-tick-swap*›
⟨*proof*⟩

lemma *surj-refusal-tick-swap* [*simp*] : ‹*surj refusal-tick-swap*›
⟨*proof*⟩

lemma *bij-refusal-tick-swap* [*simp*] : ‹*bij refusal-tick-swap*›
⟨*proof*⟩

lemma *strict-mono-refusal-tick-swap* : ‹*strict-mono refusal-tick-swap*›
⟨*proof*⟩

lemma *empty-eq-refusal-tick-swap-iff* [*iff*] : ‹{} = *refusal-tick-swap X* ⟷ *X* = {}›
and *refusal-tick-swap-eq-empty-iff* [*iff*] : ‹*refusal-tick-swap X* = {} ⟷ *X* = {}›
⟨*proof*⟩

lemma *insert-ev-eq-refusal-tick-swap-iff* [*iff*] :
‹*insert* (*ev a*) *X* = *refusal-tick-swap Y* ⟷ *Y* = *insert* (*ev a*) (*refusal-tick-swap X*)›

and *refusal-tick-swap-eq-insert-ev-iff* [*iff*] :
‹*refusal-tick-swap Y* =*insert* (*ev a*) *X* ⟷ *Y* = *insert* (*ev a*) (*refusal-tick-swap X*)›
⟨*proof*⟩

lemma *insert-tick-eq-refusal-tick-swap-iff* [*iff*] :
‹*insert* ✓((*r*, *s*)) *X* = *refusal-tick-swap Y* ⟷ *Y* = *insert* ✓((*s*, *r*)) (*refusal-tick-swap X*)›
and *refusal-tick-swap-eq-insert-tick-iff* [*iff*] :
‹*refusal-tick-swap Y* = *insert* ✓((*r*, *s*)) *X* ⟷ *Y* = *insert* ✓((*s*, *r*)) (*refusal-tick-swap X*)›
⟨*proof*⟩

lemma *refusal-tick-swap-eq-insert-evE* :
‹*refusal-tick-swap Y* = *insert* (*ev a*) *X* ⟹ (⋀*Y'*. *Y* = *insert* (*ev a*) *Y'* ⟹ *X* = *refusal-tick-swap Y'* ⟹ *thesis*) ⟹ *thesis*›
and *refusal-tick-swap-eq-insert-tickE* :
‹*refusal-tick-swap Y* = *insert* ✓((*r*, *s*)) *X* ⟹ (⋀*Y'*. *Y* = *insert* ✓((*s*, *r*)) *Y'* ⟹ *X* = *refusal-tick-swap Y'* ⟹ *thesis*) ⟹ *thesis*›
⟨*proof*⟩

lemma *refusal-tick-swap-tickFree* :
‹*X* ⊆ *range ev* ⟹ *refusal-tick-swap X* = (*ev* ∘ *of-ev*) ‘ *X*›
⟨*proof*⟩

lemma *tickFree-refusal-tick-swap-iff* :
‹*refusal-tick-swap X* ⊆ *range ev* ⟷ *X* ⊆ *range ev*›
⟨*proof*⟩

The old version of interleaving of traces is not affected.

lemma *setinterleaves-imp-setinterleaves-trace-tick-swap* :
‹*t setinterleaves* ((*u*, *v*), *S*) ⟹
trace-tick-swap t setinterleaves ((*trace-tick-swap u*, *trace-tick-swap v*), *refusal-tick-swap S*)›
⟨*proof*⟩

lemma *trace-tick-swap-setinterleaves-iff* :
‹*trace-tick-swap t setinterleaves* ((*u*, *v*), *S*) ⟷
t setinterleaves ((*trace-tick-swap u*, *trace-tick-swap v*), *refusal-tick-swap S*)›
⟨*proof*⟩

Swapping a Failure

definition *failure-tick-swap* :: ‹('*a*, ('*r* × '*s*)) *failure*$_{ptick}$ ⇒ ('*a*, ('*s* × '*r*)) *failure*$_{ptick}$›
where ‹*failure-tick-swap F* ≡ *case F of* (*t*, *X*) ⇒ (*trace-tick-swap t*, *refusal-tick-swap X*)›

lemma *failure-tick-swap-empty* [*simp*] : ‹*failure-tick-swap* ([], {}) = ([], {})›
⟨*proof*⟩

lemma *failure-tick-swap-comp-failure-tick-swap* [*simp*] :
‹*failure-tick-swap* ∘ *failure-tick-swap* = *id*›
⟨*proof*⟩

lemma *failure-tick-swap-failure-tick-swap* [*simp*] :
‹*failure-tick-swap* (*failure-tick-swap F*) = *F*›
⟨*proof*⟩

lemma *inj-failure-tick-swap* [*simp*] : ‹*inj failure-tick-swap*›
⟨*proof*⟩

lemma *surj-failure-tick-swap* [*simp*] : ‹*surj failure-tick-swap*›
⟨*proof*⟩

lemma *bij-failure-tick-swap* [*simp*] : ‹*bij failure-tick-swap*›
⟨*proof*⟩

lemma *empty-eq-failure-tick-swap-iff* [*iff*] : ‹([], {}) = *failure-tick-swap F* ⟷ *F* = ([], {})›
and *failure-tick-swap-eq-empty-iff* [*iff*] : ‹*failure-tick-swap F* = ([], {}) ⟷ *F* = ([], {})›
⟨*proof*⟩

5.1.2 The Operator

Definition

lift-definition *TickSwap* :: ‹('*a*, '*r* × '*s*) *process*$_{ptick}$ ⇒ ('*a*, '*s* × '*r*) *process*$_{ptick}$›
is ‹λ*P*. ({(*t*, *X*). *failure-tick-swap* (*t*, *X*) ∈ *F P*}, {*t*. *trace-tick-swap t* ∈ *D P*})›
— One might expect λ*P*. (*failure-tick-swap* ‘ *F P*, *trace-tick-swap* ‘ *D P*) instead. This is equivalent, see the projections below, but easier for the following proof obligation.
⟨*proof*⟩

Projections

lemma *F-TickSwap'* : ‹*F* (*TickSwap P*) = {(*t*, *X*). *failure-tick-swap* (*t*, *X*) ∈ *F P*}›
⟨*proof*⟩

lemma *D-TickSwap'* : ‹*D* (*TickSwap P*) = {*t*. *trace-tick-swap t* ∈ *D P*}›
⟨*proof*⟩

lemma *T-TickSwap'* : ‹$\mathcal{T}$ (*TickSwap P*) = {*t*. *trace-tick-swap t* ∈ $\mathcal{T}$ *P*}›
⟨*proof*⟩

lemmas *TickSwap-projs'* = *F-TickSwap' D-TickSwap' T-TickSwap'*

This is not very intuitive. The following lemmas are more intuitive.

lemma *F-TickSwap* : ‹$\mathcal{F}$ (*TickSwap P*) = *failure-tick-swap* ‘ $\mathcal{F}$ *P*›
⟨*proof*⟩

lemma *D-TickSwap* : ‹$\mathcal{D}$ (*TickSwap P*) = *trace-tick-swap* ‘ $\mathcal{D}$ *P*›
⟨*proof*⟩

lemma *T-TickSwap* : ‹$\mathcal{T}$ (*TickSwap P*) = *trace-tick-swap* ‘ $\mathcal{T}$ *P*›
⟨*proof*⟩

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

We finally give the following versions, sometimes more convenient to use.

lemma *F-TickSwap''* : ‹$\mathcal{F}$ (*TickSwap P*) = {(*trace-tick-swap t*, *refusal-tick-swap X*)| *t X*. (*t*, *X*) ∈ $\mathcal{F}$ *P*}›
⟨*proof*⟩

lemma *D-TickSwap''* : ‹$\mathcal{D}$ (*TickSwap P*) = {*trace-tick-swap t*| *t*. *t* ∈ $\mathcal{D}$ *P*}›
⟨*proof*⟩

lemma *T-TickSwap''* : ‹$\mathcal{T}$ (*TickSwap P*) = {*trace-tick-swap t*| *t*. *t* ∈ $\mathcal{T}$ *P*}›
⟨*proof*⟩

lemmas *TickSwap-projs''* = *F-TickSwap'' D-TickSwap'' T-TickSwap''*

Properties

lemma *events-TickSwap* [*simp*] : ‹*events-of* (*TickSwap P*) = *events-of P*›
⟨*proof*⟩

lemma *ticks-TickSwap* [*simp*] : ‹*ticks-of* (*TickSwap P*) = {(*s*, *r*). (*r*, *s*) ∈ *ticks-of P*}›
⟨*proof*⟩

lemma *strict-ticks-TickSwap* [*simp*] :
‹*strict-ticks-of* (*TickSwap P*) = {(*s*, *r*). (*r*, *s*) ∈ *strict-ticks-of P*}›
⟨*proof*⟩

lemma *trace-tick-swap-image-setinterleaving*$_{Pair}$:
‹*trace-tick-swap* ‘ *setinterleaving*$_{ptick}$ (λ*r s*. ⌊(*r*, *s*)⌋, *u*, *A*, *v*) =
setinterleaving$_{ptick}$ (λ*r s*. ⌊(*r*, *s*)⌋, *v*, *A*, *u*)›
for *u* :: ‹('*a*, '*r*) *trace*$_{ptick}$› **and** *v* :: ‹('*a*, '*s*) *trace*$_{ptick}$›
⟨*proof*⟩

lemma *trace-tick-swap-setinterleaves$_{Pair}$-iff* [*iff*] :
‹*trace-tick-swap t setinterleaves*$_{✓\lambda r\ s.\ \lfloor(r,\ s)\rfloor}$ $((u,\ v),\ A) \longleftrightarrow$
t setinterleaves$_{✓\lambda r\ s.\ \lfloor(r,\ s)\rfloor}$ $((v,\ u),\ A)$›
⟨*proof*⟩

The following theorem is a bridge with the existing operators: *TickSwap* can be expressed via the *Renaming* operator.

lemma *tick-swap-is-map-event$_{ptick}$* : ‹*tick-swap* = *map-event$_{ptick}$ id prod.swap*›
⟨*proof*⟩

lemma *trace-tick-swap-is-map-map-event$_{ptick}$* :
‹*trace-tick-swap* = *map* (*map-event$_{ptick}$ id prod.swap*)›
⟨*proof*⟩

lemma *refusal-tick-swap-is-image-map-event$_{ptick}$* :
‹*refusal-tick-swap* = (‘) (*map-event$_{ptick}$ id prod.swap*)›
⟨*proof*⟩

theorem *TickSwap-is-Renaming* :
‹*TickSwap P* = *Renaming P id prod.swap*› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *TickSwap-TickSwap* [*simp*] : ‹*TickSwap* (*TickSwap P*) = *P*›
⟨*proof*⟩

lemma *TickSwap-comp-TickSwap* [*simp*] : ‹*TickSwap* ∘ *TickSwap* = *id*›
⟨*proof*⟩

lemma *TickSwap-eq-iff-eq-TickSwap* : ‹*TickSwap P* = *Q* ⟷ *P* = *TickSwap Q*›
⟨*proof*⟩

lemma *inj-TickSwap* [*simp*] : ‹*inj TickSwap*›
⟨*proof*⟩

lemma *surj-TickSwap* [*simp*] : ‹*surj TickSwap*›
⟨*proof*⟩

lemma *bij-TickSwap* [*simp*] : ‹*bij TickSwap*›
⟨*proof*⟩

lemma *strict-mono-TickSwap* : ‹*strict-mono TickSwap*›
⟨*proof*⟩

Monotonicity Properties

lemma *mono-TickSwap* : ‹*P* ⊑ *Q* ⟹ *TickSwap P* ⊑ *TickSwap Q*›

⟨*proof*⟩

lemma *mono-TickSwap-FD* : ‹$P \sqsubseteq_{FD} Q \Longrightarrow TickSwap\ P \sqsubseteq_{FD} TickSwap\ Q$›
and *mono-TickSwap-DT* : ‹$P \sqsubseteq_{DT} Q \Longrightarrow TickSwap\ P \sqsubseteq_{DT} TickSwap\ Q$›
and *mono-TickSwap-F* : ‹$P \sqsubseteq_{F} Q \Longrightarrow TickSwap\ P \sqsubseteq_{F} TickSwap\ Q$›
and *mono-TickSwap-D* : ‹$P \sqsubseteq_{D} Q \Longrightarrow TickSwap\ P \sqsubseteq_{D} TickSwap\ Q$›
and *mono-TickSwap-T* : ‹$P \sqsubseteq_{T} Q \Longrightarrow TickSwap\ P \sqsubseteq_{T} TickSwap\ Q$›
⟨*proof*⟩

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

lemma *le-approx-TickSwap-iff* : ‹$TickSwap\ P \sqsubseteq TickSwap\ Q \longleftrightarrow P \sqsubseteq Q$›
and *FD-TickSwap-iff* : ‹$TickSwap\ P \sqsubseteq_{FD} TickSwap\ Q \longleftrightarrow P \sqsubseteq_{FD} Q$›
and *DT-TickSwap-iff* : ‹$TickSwap\ P \sqsubseteq_{DT} TickSwap\ Q \longleftrightarrow P \sqsubseteq_{DT} Q$›
and *F-TickSwap-iff* : ‹$TickSwap\ P \sqsubseteq_{F} TickSwap\ Q \longleftrightarrow P \sqsubseteq_{F} Q$›
and *D-TickSwap-iff* : ‹$TickSwap\ P \sqsubseteq_{D} TickSwap\ Q \longleftrightarrow P \sqsubseteq_{D} Q$›
and *T-TickSwap-iff* : ‹$TickSwap\ P \sqsubseteq_{T} TickSwap\ Q \longleftrightarrow P \sqsubseteq_{T} Q$›
⟨*proof*⟩

lemmas *le-TickSwap-iff* = *le-approx-TickSwap-iff FD-TickSwap-iff DT-TickSwap-iff F-TickSwap-iff D-TickSwap-iff T-TickSwap-iff*

Continuity

Continuity is a direct corollary of the continuity of *Renaming*.

lemma *TickSwap-cont*[*simp*] : ‹$cont\ P \Longrightarrow cont\ (\lambda x.\ TickSwap\ (P\ x))$›
⟨*proof*⟩

Algebraic Laws

Constant Processes **lemma** *TickSwap-STOP* [*simp*] : ‹$TickSwap\ STOP = STOP$›
⟨*proof*⟩

lemma *TickSwap-is-STOP-iff* [*simp*] : ‹$TickSwap\ P = STOP \longleftrightarrow P = STOP$›
⟨*proof*⟩

lemma *TickSwap-BOT* [*simp*] : ‹$TickSwap\ \bot = \bot$›
⟨*proof*⟩

lemma *TickSwap-is-BOT-iff* [*simp*] : ‹$TickSwap\ P = \bot \longleftrightarrow P = \bot$›
⟨*proof*⟩

lemma *TickSwap-SKIP* [*simp*] : ‹$TickSwap\ (SKIP\ (r,\ s)) = SKIP\ (s,\ r)$›
⟨*proof*⟩

lemma *TickSwap-is-SKIP-iff* [*simp*] : ‹*TickSwap P* = *SKIP* (*r*, *s*) ⟷ *P* = *SKIP* (*s*, *r*)›
⟨*proof*⟩

lemma *TickSwap-SKIPS* [*simp*] : ‹*TickSwap* (*SKIPS R-S*) = *SKIPS* {(*s*, *r*). (*r*, *s*) ∈ *R-S*}›
⟨*proof*⟩

lemma *TickSwap-is-SKIPS-iff* [*simp*] :
‹*TickSwap P* = *SKIPS R-S* ⟷ *P* = *SKIPS* {(*s*, *r*). (*r*, *s*) ∈ *R-S*}›
⟨*proof*⟩

Binary (or less) Operators **lemma** *TickSwap-Ndet* [*simp*] : ‹*TickSwap* (*P* ⊓ *Q*) = *TickSwap P* ⊓ *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-is-Ndet-iff* [*simp*] : ‹*TickSwap P* = *Q* ⊓ *R* ⟷ *P* = *TickSwap Q* ⊓ *TickSwap R*›
⟨*proof*⟩

lemma *TickSwap-Det* [*simp*] :
‹*TickSwap* (*P* □ *Q*) = *TickSwap P* □ *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-is-Det-iff* [*simp*] : ‹*TickSwap P* = *Q* □ *R* ⟷ *P* = *TickSwap Q* □ *TickSwap R*›
⟨*proof*⟩

lemma *TickSwap-Sliding* [*simp*] : ‹*TickSwap* (*P* ▷ *Q*) = *TickSwap P* ▷ *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-is-Sliding-iff* [*simp*] : ‹*TickSwap P* = *Q* ▷ *R* ⟷ *P* = *TickSwap Q* ▷ *TickSwap R*›
⟨*proof*⟩

lemma *TickSwap-Sync* [*simp*] :
‹*TickSwap* (*P* ⟦*S*⟧ *Q*) = *TickSwap P* ⟦*S*⟧ *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-is-Sync-iff* [*simp*] :
‹*TickSwap P* = *Q* ⟦*S*⟧ *R* ⟷ *P* = *TickSwap Q* ⟦*S*⟧ *TickSwap R*›
⟨*proof*⟩

lemma *TickSwap-Seq* [*simp*] :
‹*TickSwap* (*P* ; *Q*) = *TickSwap P* ; *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-is-Seq-iff* [*simp*] :
‹*TickSwap P* = *Q* ; *R* ⟷ *P* = *TickSwap Q* ; *TickSwap R*›
⟨*proof*⟩

lemma *TickSwap-Renaming* [*simp*] :
‹*TickSwap* (*Renaming P f g*) =
Renaming (*TickSwap P*) *f* (*prod.swap* ∘ *g* ∘ *prod.swap*)›
⟨*proof*⟩

lemma *TickSwap-Renaming′* :
‹*TickSwap* (*Renaming P f g*) = *Renaming P f* (*prod.swap* ∘ *g*)›
⟨*proof*⟩

lemma *TickSwap-is-Renaming-iff* [*simp*] :
‹*TickSwap P* = *Renaming Q f g* ⟷ *P* = *Renaming* (*TickSwap Q*) *f* (*prod.swap* ∘ *g* ∘ *prod.swap*)›
⟨*proof*⟩

lemma *TickSwap-Hiding* [*simp*] : ‹*TickSwap* (*P* \ *S*) = *TickSwap P* \ *S*›
⟨*proof*⟩

lemma *TickSwap-is-Hiding-iff* [*simp*] : ‹*TickSwap P* = *Q* \ *S* ⟷ *P* = *TickSwap Q* \ *S*›
⟨*proof*⟩

lemma *TickSwap-Interrupt* [*simp*] :
‹*TickSwap* (*P* △ *Q*) = *TickSwap P* △ *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-is-Interrupt-iff* [*simp*] :
‹*TickSwap P* = *Q* △ *R* ⟷ *P* = *TickSwap Q* △ *TickSwap R*›
⟨*proof*⟩

lemma *TickSwap-Throw* [*simp*] :
‹*TickSwap* (*P* Θ *a* ∈ *A*. *Q a*) = *TickSwap P* Θ *a* ∈ *A*. *TickSwap* (*Q a*)›
⟨*proof*⟩

lemma *TickSwap-is-Throw-iff* [*simp*] :
‹*TickSwap P* = *Q* Θ *a* ∈ *A*. *R a* ⟷ *P* = *TickSwap Q* Θ *a* ∈ *A*. *TickSwap* (*R a*)›

⟨*proof*⟩

Architectural Operators **lemma** *TickSwap-GlobalNdet* [*simp*] :
‹*TickSwap* (⊓*a* ∈ *A*. *P a*) = ⊓*a* ∈ *A*. *TickSwap* (*P a*)›
⟨*proof*⟩

lemma *TickSwap-is-GlobalNdet-iff* [*simp*] :
‹*TickSwap P* = ⊓*a* ∈ *A*. *Q a* ⟷ *P* = ⊓*a* ∈ *A*. *TickSwap* (*Q a*)›
⟨*proof*⟩

lemma *TickSwap-GlobalDet* [*simp*] :
‹*TickSwap* (□*a* ∈ *A*. *P a*) = □*a* ∈ *A*. *TickSwap* (*P a*)›
⟨*proof*⟩

lemma *TickSwap-is-GlobalDet-iff* [*simp*] :
‹*TickSwap P* = □*a* ∈ *A*. *Q a* ⟷ *P* = □*a* ∈ *A*. *TickSwap* (*Q a*)›
⟨*proof*⟩

lemma *TickSwap-MultiSync* [*simp*] :
‹*TickSwap* (⟦*S*⟧ *m* ∈# *M*. *P m*) = ⟦*S*⟧ *m* ∈# *M*. *TickSwap* (*P m*)›
⟨*proof*⟩

lemma *TickSwap-is-TickSwap-MultiSync-iff* [*simp*] :
‹*TickSwap P* = ⟦*S*⟧ *m* ∈# *M*. *Q m* ⟷ *P* = ⟦*S*⟧ *m* ∈# *M*. *TickSwap* (*Q m*)›
⟨*proof*⟩

lemma *TickSwap-MultiSeq* [*simp*] :
‹*L* ≠ [] ⟹ *TickSwap* (*SEQ l* ∈@ *L*. *P l*) = *SEQ l* ∈@ *L*. *TickSwap* (*P l*)›
⟨*proof*⟩

lemma *TickSwap-is-MultiSeq-iff* [*simp*] :
‹*L* ≠ [] ⟹ *TickSwap P* = *SEQ l* ∈@ *L*. *Q l* ⟷ *P* = *SEQ l* ∈@ *L*. *TickSwap* (*Q l*)›
⟨*proof*⟩

Communications **lemma** *TickSwap-write0* [*simp*] : ‹*TickSwap* (*e* → *P*) = *e* → *TickSwap P*›
⟨*proof*⟩

lemma *TickSwap-is-write0-iff* [*simp*] : ‹*TickSwap P* = *e* → *Q* ⟷ *P* = *e* → *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-write* [*simp*] : ‹*TickSwap* (*c*!*e* → *P*) = *c*!*e* → *TickSwap P*›
⟨*proof*⟩

lemma *TickSwap-is-write-iff* [*simp*] : ‹*TickSwap P* = *c*!*e* → *Q* ⟷ *P* = *c*!*e* → *TickSwap Q*›
⟨*proof*⟩

lemma *TickSwap-Mprefix* [*simp*] :
‹*TickSwap* (□*a* ∈ *A* → *P a*) = □*a* ∈ *A* → *TickSwap* (*P a*)›
⟨*proof*⟩

lemma *TickSwap-is-Mprefix-iff* [*simp*] :
‹*TickSwap P* = (□*a* ∈ *A* → *Q a*) ⟷ *P* = □*a* ∈ *A* → *TickSwap* (*Q a*)›
⟨*proof*⟩

lemma *TickSwap-read* [*simp*] : ‹*TickSwap* (*c*?*a*∈*A* → *P a*) = *c*?*a*∈*A* → *TickSwap* (*P a*)›
⟨*proof*⟩

lemma *TickSwap-is-read-iff* [*simp*] :
‹*TickSwap P* = *c*?*a*∈*A* → *Q a* ⟷ *P* = *c*?*a*∈*A* → *TickSwap* (*Q a*)›
⟨*proof*⟩

lemma *TickSwap-Mndetprefix* [*simp*] :
‹*TickSwap* (⊓*a* ∈ *A* → *P a*) = ⊓*a* ∈ *A* → *TickSwap* (*P a*)›
⟨*proof*⟩

lemma *TickSwap-is-Mndetprefix-iff* [*simp*] :
‹*TickSwap P* = (⊓*a* ∈ *A* → *Q a*) ⟷ *P* = ⊓*a* ∈ *A* → *TickSwap* (*Q a*)›
⟨*proof*⟩

lemma *TickSwap-ndet-write* [*simp*] : ‹*TickSwap* (*c*!!*a*∈*A* → *P a*) = *c*!!*a*∈*A* → *TickSwap* (*P a*)›
⟨*proof*⟩

lemma *TickSwap-is-ndet-write-iff* [*simp*] :
‹*TickSwap P* = *c*!!*a*∈*A* → *Q a* ⟷ *P* = *c*!!*a*∈*A* → *TickSwap* (*Q a*)›
⟨*proof*⟩

5.2 Splitting the Renaming Operator

We split the *Renaming* operator in two: the first one only renames the "regular" events, the second one only the ticks.

5.2.1 Renaming only Events

abbreviation *RenamingEv* :: ‹[('a, 'r) $process_{ptick}$, 'a ⇒ 'b] ⇒ ('b, 'r) $process_{ptick}$›
where ‹*RenamingEv P f* ≡ *Renaming P f id*›

lemma *RenamingEv-id-unfolded* [*iff*] :
‹*Renaming P f* (λ*r*. *r*) = *RenamingEv P f*› ⟨*proof*⟩

lemmas *strict-ticks-of-RenamingEv-subset* = *strict-ticks-of-Renaming-subset* [**where** *g* = *id*, *simplified*]
and *strict-ticks-of-inj-on-RenamingEv* = *strict-ticks-of-inj-on-Renaming* [**where** *g* = *id*, *simplified*]

lemmas *monos-RenamingEv* = *monos-Renaming*[**where** *g* = *id*]

lemma *RenamingEv-SKIP* : ‹*RenamingEv* (*SKIP r*) *f* = *SKIP r*› ⟨*proof*⟩

lemma *RenamingEv-cont* :
‹*cont P* ⟹ *finitary f* ⟹ *cont* (λ*x*. *RenamingEv* (*P x*) *f*)› ⟨*proof*⟩

lemma *RenamingEv-Seq* :
‹*RenamingEv* (*P* ; *Q*) *f* = *RenamingEv P f* ; *RenamingEv Q f*›
⟨*proof*⟩

declare *Renaming-id* [*simp*]

lemmas *RenamingEv-id* = *Renaming-id*
and *RenamingEv-STOP* = *Renaming-STOP* [**where** *g* = *id*]
and *RenamingEv-BOT* = *Renaming-BOT* [**where** *g* = *id*]
and *RenamingEv-is-STOP-iff* = *Renaming-is-STOP-iff* [**where** *g* = *id*]
and *RenamingEv-is-BOT-iff* = *Renaming-is-BOT-iff* [**where** *g* = *id*]

lemmas *RenamingEv-Det* = *Renaming-Det* [**where** *g* = *id*]
and *RenamingEv-Ndet* = *Renaming-Ndet* [**where** *g* = *id*]
and *RenamingEv-Sliding* = *Renaming-Sliding* [**where** *g* = *id*]
and *RenamingEv-Interrupt* = *Renaming-Interrupt* [**where** *g* = *id*]
and *RenamingEv-write0* = *Renaming-write0* [**where** *g* = *id*]
and *RenamingEv-write* = *Renaming-write* [**where** *g* = *id*]
and *RenamingEv-comp* = *Renaming-comp* [*of - - - id id*, *simplified*]
and *RenamingEv-inv* = *Renaming-inv* [**where** *g* = *id*, *simplified*]
and *inv-RenamingEv* = *inv-Renaming* [**where** *g* = *id*, *simplified*]

lemmas *bij-RenamingEv-Sync* = *bij-Renaming-Sync* [**where** $g = id$, *simplified*]
and *bij-RenamingEv-Hiding* = *bij-Renaming-Hiding* [**where** $g = id$, *simplified*]
and *inj-on-RenamingEv-Throw* = *inj-on-Renaming-Throw* [**where** $g = id$]
and *RenamingEv-fix* = *Renaming-fix* [**where** $g = id$, *simplified*]

lemmas *RenamingEv-distrib-GlobalDet* = *Renaming-distrib-GlobalDet* [**where** $g = id$]
and *RenamingEv-distrib-GlobalNDet* = *Renaming-distrib-GlobalNdet* [**where** $g = id$]
and *RenamingEv-Mprefix* = *Renaming-Mprefix* [**where** $g = id$]
and *RenamingEv-Mndetprefix* = *Renaming-Mndetprefix* [**where** $g = id$]
and *RenamingEv-read* = *Renaming-read* [**where** $g = id$]
and *RenamingEv-ndet-write* = *Renaming-ndet-write* [**where** $g = id$]

5.2.2 Renaming only Ticks

abbreviation *RenamingTick* :: ‹$[('a, 'r)\ process_{ptick}, 'r \Rightarrow 's] \Rightarrow ('a, 's)\ process_{ptick}$›
where ‹*RenamingTick P g* ≡ *Renaming P id g*›

lemma *RenamingTick-id-unfolded* [*iff*] :
‹*Renaming P* (λ*a*. *a*) *g* = *RenamingTick P g*› ⟨*proof*⟩

lemmas *strict-ticks-of-RenamingTick-subset* = *strict-ticks-of-Renaming-subset* [**where** $f = id$]
and *strict-ticks-of-inj-on-RenamingTick* = *strict-ticks-of-inj-on-Renaming* [**where** $f = id$, *simplified*]

lemmas *monos-RenamingTick* = *monos-Renaming*[**where** $f = id$]

lemma *RenamingTick-cont* :
‹*cont P* ⟹ *finitary g* ⟹ *cont* (λ*x*. *RenamingTick* (*P x*) *g*)› ⟨*proof*⟩

lemmas *RenamingTick-id* = *Renaming-id*
and *RenamingTick-STOP* = *Renaming-STOP* [**where** $f = id$]
and *RenamingTick-SKIP* = *Renaming-SKIP* [**where** $f = id$]
and *RenamingTick-BOT* = *Renaming-BOT* [**where** $f = id$]
and *RenamingTick-is-STOP-iff* = *Renaming-is-STOP-iff* [**where** $f = id$]
and *RenamingTick-is-BOT-iff* = *Renaming-is-BOT-iff* [**where** $f = id$]

lemmas *RenamingTick-Seq* = *Renaming-Seq*[**where** $f = id$]
and *RenamingTick-Det* = *Renaming-Det* [**where** $f = id$]
and *RenamingTick-Ndet* = *Renaming-Ndet* [**where** $f = id$]
and *RenamingTick-Sliding* = *Renaming-Sliding* [**where** $f = id$]

and *RenamingTick-Interrupt* = *Renaming-Interrupt* [**where** $f = id$]
and *RenamingTick-write0* = *Renaming-write0* [**where** $f = id$, *simplified*]
and *RenamingTick-write* = *Renaming-write* [**where** $f = id$, *simplified*]
and *RenamingTick-comp* = *Renaming-comp* [*of* - *id id* , *simplified*]
and *RenamingTick-inv* = *Renaming-inv* [**where** $f = id$, *simplified*]
and *inv-RenamingTick* = *inv-Renaming* [**where** $f = id$, *simplified*]

lemmas *bij-RenamingTick-Sync* = *bij-Renaming-Sync* [**where** $f = id$, *simplified*]

and *RenamingTick-fix* = *Renaming-fix* [**where** $f = id$, *simplified*]

— The assumption *bij g* is actually not necessary for *RenamingTick* and (\), see below.

lemma *RenamingTick-Throw* :
‹*RenamingTick* (*P* Θ *a*∈*A*. *Q a*) *g* = *RenamingTick P g* Θ *a*∈*A*. *RenamingTick* (*Q a*) *g*›
⟨*proof*⟩

lemmas *RenamingTick-distrib-GlobalDet* = *Renaming-distrib-GlobalDet* [**where** $f = id$]
and *RenamingTick-distrib-GlobalNDet* = *Renaming-distrib-GlobalNdet* [**where** $f = id$]
and *RenamingTick-Mprefix* = *Renaming-Mprefix-image-inj* [**where** $f = id$, *simplified*]
and *RenamingTick-Mndetprefix* = *Renaming-Mndetprefix-inj* [**where** $f = id$, *simplified*]
and *RenamingTick-read* = *Renaming-read* [**where** $f = id$, *simplified*]
and *RenamingTick-ndet-write* = *Renaming-ndet-write* [**where** $f = id$, *simplified*]

lemma *RenamingEv-RenamingTick-is-Renaming* :
‹*RenamingEv* (*RenamingTick P g*) *f* = *Renaming P f g*›
and *RenamingTick-RenamingEv-is-Renaming* :
‹*RenamingTick* (*RenamingEv P f*) *g* = *Renaming P f g*›
⟨*proof*⟩

5.2.3 Properties

lemma *isInfHidden-seqRun-imp-tickFree-seqRun* :
‹*isInfHidden-seqRun x P A t* ⟹ *tF* (*seqRun t x i*)›
⟨*proof*⟩

lemma *tickFree-map-map-event$_{ptick}$-is* :
‹*tF t* $\Longrightarrow$ *map* (*map-event$_{ptick}$ f g*) *t* = *map* (*ev* $\circ$ *f* $\circ$ *of-ev*) *t*›
⟨*proof*⟩

lemma *RenamingTick-Hiding* :
‹*RenamingTick* (*P* $\setminus$ *A*) *g* = *RenamingTick P g* $\setminus$ *A*›
(**is** ‹*?lhs* = *?rhs*›) **for** *P* :: ‹('*a*, '*r*) *process$_{ptick}$*›
⟨*proof*⟩

corollary *bij-Renaming-Hiding* :
‹*Renaming* (*P* $\setminus$ *S*) *f g* = *Renaming P f g* $\setminus$ *f* ‘ *S*› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*bij f*›
— We already have ⟦*bij fa*; *bij ga*⟧ $\Longrightarrow$ *Renaming* (*Pa* $\setminus$ *Sa*) *fa ga* = *Renaming Pa fa ga* $\setminus$ *fa* ‘ *Sa*, but he assumption *bij g* is actually not necessary.
⟨*proof*⟩

lemma *Renaming-is-restrictable-on-events-of-strict-ticks-of* :
‹*Renaming P f g* = *Renaming P f' g'*›
if *fun-hyps* : ‹$\bigwedge$*a*. *a* $\in$ α(*P*) $\Longrightarrow$ *f a* = *f' a*›
‹$\bigwedge$*r*. *r* $\in$ ✓**s**(*P*) $\Longrightarrow$ *g r* = *g' r*›
for *f f'* :: ‹'*a* $\Rightarrow$ '*b*› **and** *g g'* :: ‹'*r* $\Rightarrow$ '*t*›
— probably also possible to strengthen with *strict-events-of*
⟨*proof*⟩

corollary *Renaming-is-restrictable-on-events-of-ticks-of* :
‹⟦$\bigwedge$*a*. *a* $\in$ α(*P*) $\Longrightarrow$ *f a* = *f' a*; $\bigwedge$*r*. *r* $\in$ ✓*s*(*P*) $\Longrightarrow$ *g r* = *g' r*⟧
$\Longrightarrow$ *Renaming P f g* = *Renaming P f' g'*›
⟨*proof*⟩

corollary *RenamingEv-is-restrictable-on-events-of* :
‹($\bigwedge$*a*. *a* $\in$ α(*P*) $\Longrightarrow$ *f a* = *f' a*) $\Longrightarrow$ *RenamingEv P f* = *RenamingEv P f'*›
⟨*proof*⟩

corollary *RenamingTick-is-restrictable-on-strict-ticks-of* :
‹($\bigwedge$*r*. *r* $\in$ ✓**s**(*P*) $\Longrightarrow$ *g r* = *g' r*) $\Longrightarrow$ *RenamingTick P g* = *RenamingTick P g'*›
⟨*proof*⟩

corollary *RenamingTick-is-restrictable-on-ticks-of* :
‹($\bigwedge$*r*. *r* $\in$ ✓*s*(*P*) $\Longrightarrow$ *g r* = *g' r*) $\Longrightarrow$ *RenamingTick P g* = *RenamingTick P g'*›
⟨*proof*⟩

5.3 Renaming and Generalized Synchronization Product

lemma (**in** $Sync_{ptick}$-*locale*) *inj-on-RenamingTick-Sync*$_{ptick}$:
‹*RenamingTick* (*P* $[\![S]\!]_{\checkmark}$ *Q*) *g* =
Sync$_{ptick}$-*locale.Sync*$_{ptick}$ (λ*r s*. *case r* ⊗✓ *s of* ⌊*r-s*⌋ ⇒ ⌊*g r-s*⌋ | ◊ ⇒ ◊) *P S Q*›
(**is** ‹*?lhs* = *?rhs*›)
if *inj-on-g* : ‹*inj-on g range-tick-join*›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*locale*) *inj-RenamingTick-Sync*$_{ptick}$*-inj-RenamingTick* :
‹*RenamingTick P g* $[\![S]\!]_{\checkmark}$ *RenamingTick Q h* =
Sync$_{ptick}$-*locale.Sync*$_{ptick}$ (λ*r s*. *g r* ⊗✓ *h s*) *P S Q*› (**is** ‹*?lhs* = *?rhs*›)
if ‹*inj g*› **and** ‹*inj h*›

for *P* :: ‹('*a*, '*r*') *process*$_{ptick}$› **and** *Q* :: ‹('*a*, '*s*') *process*$_{ptick}$›
⟨*proof*⟩

Chapter 6

Commutativity and Associativity of Synchronization

6.1 Commutativity

6.1.1 Motivation

The classical synchronization product is commutative: $P \ [\![A]\!] \ Q = Q \ [\![A]\!] \ P$ but in our generalization such a law cannot be obtained in all generality. Imagine for example that the $(\otimes\checkmark)$ parameter is actually $\lambda r\ s.\ \lfloor(r,\ s)\rfloor$: we easily figure out that in this case the corresponding law should be something like $P \ [\![A]\!]_{\checkmark Pair} \ Q = \mathit{TickSwap} \ (Q \ [\![A]\!]_{\checkmark Pair} \ P)$. More generally, in the **locale**, when writing $P \ [\![A]\!]_{\checkmark} \ Q$, P is of type $('a,\ 'r)\ \mathit{process}_{ptick}$ while Q is of type $('a,\ 's)\ \mathit{process}_{ptick}$ so we want to find an abstract setup in which we can establish a quasi-commutativity. This is done in the next subsection.

6.1.2 Formalization

locale Sync_{ptick}*-comm-locale* =
  Sync_{ptick}*-locale* ‹$(\otimes\checkmark)$› **for** *tick-join* :: ‹$'r \Rightarrow 's \Rightarrow 't\ \mathit{option}$› (**infixl** ‹$\otimes\checkmark$› *100*)
+
fixes *tick-join-rev* :: ‹$'s \Rightarrow 'r \Rightarrow 'u\ \mathit{option}$› (**infixl** ‹$\otimes\checkmark_{rev}$› *100*)
  and *tick-join-conv* :: ‹$'t \Rightarrow 'u$› (‹$\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$›)
  and *tick-join-rev-conv* :: ‹$'u \Rightarrow 't$› (‹$\otimes\checkmark_{rev} \Rightarrow \otimes\checkmark$›)
assumes *tick-join-None-iff* :
  ‹$r \otimes\checkmark\ s = \Diamond \longleftrightarrow s \otimes\checkmark_{rev}\ r = \Diamond$›
  and *tick-join-Some-imp* :
  ‹$r \otimes\checkmark\ s = \lfloor r\text{-}s\rfloor \Longrightarrow s \otimes\checkmark_{rev}\ r = \lfloor\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}\ r\text{-}s\rfloor$›
  and *tick-join-rev-Some-imp* :
  ‹$s \otimes\checkmark_{rev}\ r = \lfloor s\text{-}r\rfloor \Longrightarrow r \otimes\checkmark\ s = \lfloor\otimes\checkmark_{rev} \Rightarrow \otimes\checkmark\ s\text{-}r\rfloor$›
begin

There is an obvious symmetry over the variables.

sublocale $Sync_{ptick}$-*comm-locale-sym* :
$Sync_{ptick}$-*comm-locale* ‹$(\otimes\checkmark_{rev})$› ‹$(\otimes\checkmark)$› ‹$\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$› ‹$\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$›
⟨*proof*⟩

notation $Sync_{ptick}$-*comm-locale-sym.*$Sync_{ptick}$ (‹(- $[\![$-$]\!]_{\checkmark rev}$ -)› [70, 0, 71] 70)
notation $Sync_{ptick}$-*comm-locale-sym.*$Inter_{ptick}$ (‹(- $|||_{\checkmark rev}$ -)› [72, 73] 72)
notation $Sync_{ptick}$-*comm-locale-sym.*Par_{ptick} (‹(- $||_{\checkmark rev}$ -)› [74, 75] 74)

6.1.3 First Properties

lemma *tick-join-conv-image-range-tick-join* :
‹$\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ ‘ *range-tick-join* = $Sync_{ptick}$-*comm-locale-sym.range-tick-join*›
⟨*proof*⟩

lemma *tick-join-rev-conv-comp-tick-join-conv* [*simp*] :
‹*r-s* ∈ *range-tick-join* ⟹ $\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ ($\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ *r-s*) = *r-s*›
⟨*proof*⟩

lemma *inj-on-tick-join-conv* : ‹*inj-on* $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ *range-tick-join*›
⟨*proof*⟩

lemma *bij-betw-tick-join-conv* :
‹*bij-betw* $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ *range-tick-join* $Sync_{ptick}$-*comm-locale-sym.range-tick-join*›
⟨*proof*⟩

lemma *map-tick-join-rev-conv-map-tick-join-conv* :
‹{*r-s*. ✓(*r-s*) ∈ *set t*} ⊆ *range-tick-join* ⟹
map (*map-event*$_{ptick}$ *id* $\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$) (*map* (*map-event*$_{ptick}$ *id* $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$) *t*)
= *t*›
⟨*proof*⟩

end

6.1.4 Commutativity

context $Sync_{ptick}$-*comm-locale* **begin**

lemma $setinterleaves_{ptick}$-*imp-*$setinterleaves_{ptick}$-*rev* :
‹*t* $setinterleaves_{\checkmark(\otimes\checkmark)}$ ((*u*, *v*), *A*) ⟹
map (*map-event*$_{ptick}$ *id* $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$) *t*
$setinterleaves_{\checkmark(\otimes\checkmark_{rev})}$ ((*v*, *u*), *A*)›
— Finally not used, and probably obtainable as a corollary of *t* $setinterleaves_{\checkmark(\otimes\checkmark)}$
((*u*, *v*), *A*) ⟹ *map* (*map-event*$_{ptick}$ *id* $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$) *t* $setinterleaves_{\checkmark}$*λr s. case r* $\otimes\checkmark$ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ =
((*u*, *v*), *A*)
⟨*proof*⟩

lemma *vimage-tick-join-rev-conv-subset-super-ref-Sync$_{ptick}$-iff* :
‹*map-event$_{ptick}$ id* ⊗✓$_{rev}$⇒⊗✓ −‘ *X* ⊆ *super-ref-Sync$_{ptick}$* (⊗✓$_{rev}$) *X-Q A X-P*
⟷ *X* ⊆ *super-ref-Sync$_{ptick}$* (⊗✓) *X-P A X-Q*›
(**is** ‹*?lhs1* ⊆ *?lhs2* ⟷ *X* ⊆ *?rhs*›)
— Same: finally not used, and probably obtainable as a corollary of (*map-event$_{ptick}$ id* ⊗✓$_{rev}$⇒⊗✓ −‘ *?X* ⊆ *super-ref-Sync$_{ptick}$* (⊗✓$_{rev}$) *?X-P ?A ?X-Q*) = (*?X* ⊆ *super-ref-Sync$_{ptick}$* (λ*r s. case r* ⊗✓$_{rev}$ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊⊗✓$_{rev}$⇒⊗✓ *r-s*⌋) *?X-P ?A ?X-Q*).
⟨*proof*⟩

In the end, the proof is quite simple: mainly a corollary of *inj-on g range-tick-join* ⟹ *RenamingTick* (*P* ⟦*S*⟧✓ *Q*) *g* = *Sync$_{ptick}$-locale.Sync$_{ptick}$* (λ*r s. case r* ⊗✓ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊*g r-s*⌋) *P S Q*.

theorem *Sync$_{ptick}$-commute* :
‹*RenamingTick* (*P* ⟦*A*⟧✓ *Q*) ⊗✓⇒⊗✓$_{rev}$ = *Q* ⟦*A*⟧✓$_{rev}$ *P*›
⟨*proof*⟩

end

6.2 Associativity

6.2.1 Motivation

The classical synchronization product is associative: *P* ⟦*A*⟧ (*Q* ⟦*A*⟧ *R*) = *P* ⟦*A*⟧ *Q* ⟦*A*⟧ *R* but in our generalization such a law cannot be obtained in all generality. We already encountered a similar issue for the commutativity: we have to find a setup in which the different combinations of the ticks that we need make sense, and prove the quasi-associativity.

6.2.2 Formalization

locale *Sync$_{ptick}$-assoc-locale* =
Sync$_{ptick1}$: *Sync$_{ptick}$-locale* ‹(⊗✓*1*)› +
Sync$_{ptick2}$: *Sync$_{ptick}$-locale* ‹(⊗✓*2*)› +
Sync$_{ptick3}$: *Sync$_{ptick}$-locale* ‹(⊗✓*3*)› +
Sync$_{ptick4}$: *Sync$_{ptick}$-locale* ‹(⊗✓*4*)›
for *tick-join1* :: ‹*'r* ⇒ *'s* ⇒ *'t option*› (**infixl** ‹⊗✓*1*› *100*)
and *tick-join2* :: ‹*'t* ⇒ *'u* ⇒ *'v option*› (**infixl** ‹⊗✓*2*› *100*)
and *tick-join3* :: ‹*'r* ⇒ *'w* ⇒ *'x option*› (**infixl** ‹⊗✓*3*› *100*)
and *tick-join4* :: ‹*'s* ⇒ *'u* ⇒ *'w option*› (**infixl** ‹⊗✓*4*› *100*) +
fixes *tick-assoc-ren* :: ‹*'v* ⇒ *'x*› (‹⊗✓*2*⇒⊗✓*3*›)
and *tick-assoc-ren-conv* :: ‹*'x* ⇒ *'v*› (‹⊗✓*3*⇒⊗✓*2*›)
assumes *None-assms-tick-join* :

‹$r \otimes\!\checkmark 1\ s = \Diamond \Longrightarrow s \otimes\!\checkmark 4\ u = \Diamond \vee r \otimes\!\checkmark 3\ \lceil s \otimes\!\checkmark 4\ u\rceil = \Diamond$›
‹$r \otimes\!\checkmark 1\ s \neq \Diamond \Longrightarrow \lceil r \otimes\!\checkmark 1\ s\rceil \otimes\!\checkmark 2\ u = \Diamond \Longrightarrow s \otimes\!\checkmark 4\ u = \Diamond \vee r \otimes\!\checkmark 3\ \lceil s \otimes\!\checkmark 4\ u\rceil = \Diamond$›
‹$s \otimes\!\checkmark 4\ u = \Diamond \Longrightarrow r \otimes\!\checkmark 1\ s = \Diamond \vee \lceil r \otimes\!\checkmark 1\ s\rceil \otimes\!\checkmark 2\ u = \Diamond$›
‹$s \otimes\!\checkmark 4\ u \neq \Diamond \Longrightarrow r \otimes\!\checkmark 3\ \lceil s \otimes\!\checkmark 4\ u\rceil = \Diamond \Longrightarrow r \otimes\!\checkmark 1\ s = \Diamond \vee \lceil r \otimes\!\checkmark 1\ s\rceil \otimes\!\checkmark 2\ u = \Diamond$›
and *tick-assoc-ren-hyp* :
‹$r \otimes\!\checkmark 1\ s = \lfloor t\rfloor \Longrightarrow t \otimes\!\checkmark 2\ u = \lfloor v\rfloor \Longrightarrow \lceil r \otimes\!\checkmark 3\ \lceil s \otimes\!\checkmark 4\ u\rceil\rceil = \otimes\!\checkmark 2 \Rightarrow \otimes\!\checkmark 3\ v$›
and *tick-assoc-ren-conv-hyp* :
‹$s \otimes\!\checkmark 4\ u = \lfloor w\rfloor \Longrightarrow r \otimes\!\checkmark 3\ w = \lfloor x\rfloor \Longrightarrow \lceil\lceil r \otimes\!\checkmark 1\ s\rceil \otimes\!\checkmark 2\ u\rceil = \otimes\!\checkmark 3 \Rightarrow \otimes\!\checkmark 2\ x$›
begin

There is a symmetry over the variables.

sublocale *Sync$_{ptick}$-assoc-locale-sym* :
Sync$_{ptick}$-assoc-locale ‹$\lambda u\ s.\ s \otimes\!\checkmark 4\ u$› ‹$\lambda w\ r.\ r \otimes\!\checkmark 3\ w$› ‹$\lambda u\ t.\ t \otimes\!\checkmark 2\ u$›
‹$\lambda s\ r.\ r \otimes\!\checkmark 1\ s$› ‹$\otimes\!\checkmark 3 \Rightarrow \otimes\!\checkmark 2$› ‹$\otimes\!\checkmark 2 \Rightarrow \otimes\!\checkmark 3$›
⟨*proof*⟩

end

6.2.3 First Properties

lemma (**in** *Sync$_{ptick}$-assoc-locale*) *tick-assoc-ren-tick-assoc-ren-conv* :
‹$\exists r\ s\ u\ w.\ s \otimes\!\checkmark 4\ u = \lfloor w\rfloor \wedge r \otimes\!\checkmark 3\ w = \lfloor x\rfloor \Longrightarrow$
$\otimes\!\checkmark 2 \Rightarrow \otimes\!\checkmark 3\ (\otimes\!\checkmark 3 \Rightarrow \otimes\!\checkmark 2\ x) = x$›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-assoc-locale*) *tick-assoc-ren-conv-tick-assoc-ren* :
‹$\exists r\ s\ t\ u.\ r \otimes\!\checkmark 1\ s = \lfloor t\rfloor \wedge t \otimes\!\checkmark 2\ u = \lfloor v\rfloor \Longrightarrow \otimes\!\checkmark 3 \Rightarrow \otimes\!\checkmark 2\ (\otimes\!\checkmark 2 \Rightarrow \otimes\!\checkmark 3\ v) = v$›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-assoc-locale*) *inj-on-tick-assoc-ren* :
‹*inj-on* $\otimes\!\checkmark 2 \Rightarrow \otimes\!\checkmark 3\ \{v.\ \exists r\ s\ t\ u.\ r \otimes\!\checkmark 1\ s = \lfloor t\rfloor \wedge t \otimes\!\checkmark 2\ u = \lfloor v\rfloor\}$›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-assoc-locale*) *inj-on-tick-assoc-ren-conv* :
‹*inj-on* $\otimes\!\checkmark 3 \Rightarrow \otimes\!\checkmark 2\ \{x.\ \exists r\ s\ u\ w.\ s \otimes\!\checkmark 4\ u = \lfloor w\rfloor \wedge r \otimes\!\checkmark 3\ w = \lfloor x\rfloor\}$›
⟨*proof*⟩

6.2.4 Associativity for the Traces

lemma (**in** *Sync$_{ptick}$-assoc-locale*) *setinterleaves$_{ptick}$-assoc-left* :
‹$[\![t_t$ *setinterleaves*$_{\checkmark(\otimes\checkmark 1)}$ $((t_r,\ t_s),\ A);$
t_v *setinterleaves*$_{\checkmark(\otimes\checkmark 2)}$ $((t_t,\ t_u),\ A)]\!] \Longrightarrow$
$\exists t_w.$ *map* (*map-event$_{ptick}$ id* $\otimes\!\checkmark 2 \Rightarrow \otimes\!\checkmark 3$) t_v *setinterleaves*$_{\checkmark(\otimes\checkmark 3)}$ $((t_r,\ t_w),\ A) \wedge$
t_w *setinterleaves*$_{\checkmark(\otimes\checkmark 4)}$ $((t_s,\ t_u),\ A)$›

⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*assoc-locale*) $setinterleaves_{ptick}$-*assoc-right* :
‹t_w $setinterleaves_{✓(⊗✓4)}$ $((t_s,\ t_u),\ A) \Longrightarrow$
t_x $setinterleaves_{✓(⊗✓3)}$ $((t_r,\ t_w),\ A) \Longrightarrow$
$\exists\, t_t.$ *map* ($map\text{-}event_{ptick}$ *id* $⊗✓3⇒⊗✓2$) t_x $setinterleaves_{✓(⊗✓2)}$ $((t_t,\ t_u),\ A)$
$\wedge$
t_t $setinterleaves_{✓(⊗✓1)}$ $((t_r,\ t_s),\ A)$›
⟨*proof*⟩

6.2.5 Associativity

context $Sync_{ptick}$-*assoc-locale*
begin

notation $Sync_{ptick}1.Sync_{ptick}$ (‹(- $[\![\text{-}]\!]_{✓1}$ -)› [70, 0, 71] 70)
notation $Sync_{ptick}2.Sync_{ptick}$ (‹(- $[\![\text{-}]\!]_{✓2}$ -)› [70, 0, 71] 70)
notation $Sync_{ptick}3.Sync_{ptick}$ (‹(- $[\![\text{-}]\!]_{✓3}$ -)› [70, 0, 71] 70)
notation $Sync_{ptick}4.Sync_{ptick}$ (‹(- $[\![\text{-}]\!]_{✓4}$ -)› [70, 0, 71] 70)

lemma $Sync_{ptick}$-*assoc-oneside* :
‹P $[\![S]\!]_{✓3}$ (Q $[\![S]\!]_{✓4}$ R) $\sqsubseteq_{FD}$ *RenamingTick* (P $[\![S]\!]_{✓1}$ Q $[\![S]\!]_{✓2}$ R) $⊗✓2⇒⊗✓3$›
(**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
⟨*proof*⟩

end

lemma (**in** $Sync_{ptick}$-*locale*) *strict-ticks-of-*$Sync_{ptick}$-*subset* :
‹$✓\mathbf{s}(P$ $[\![S]\!]_{✓}$ $Q) \subseteq \{r\text{-}s \mid r\text{-}s\ r\ s.\ r \otimes✓\ s = \lfloor r\text{-}s \rfloor \wedge$
$r \in ✓\mathbf{s}(P) \wedge s \in ✓\mathbf{s}(Q)\}$› (**is** ‹- $\subseteq$ *?S*›)
⟨*proof*⟩

theorem (**in** $Sync_{ptick}$-*assoc-locale*) $Sync_{ptick}$-*assoc* :
‹P $[\![S]\!]_{✓3}$ (Q $[\![S]\!]_{✓4}$ R) = *RenamingTick* (P $[\![S]\!]_{✓1}$ Q $[\![S]\!]_{✓2}$ R) $⊗✓2⇒⊗✓3$› (**is**
‹*?lhs* = *?rhs*›)
⟨*proof*⟩

Chapter 7

First Laws

unbundle *option-type-syntax*

7.1 Behaviour with Constant Processes

By "basic" laws we mean the behaviour of $\bot$, *STOP* and *SKIP*, plus the associativity of some concerned operators.

lemma Seq_{ptick}*-const* [*simp*] : $\langle P \mathbin{;_{\checkmark}} (\lambda r.\ Q) = P \mathbin{;} Q\rangle$
— Very basic law.
⟨*proof*⟩

7.1.1 The Laws of $\bot$

lemma Seq_{ptick}*-is-BOT-iff* : $\langle P \mathbin{;_{\checkmark}} Q = \bot \longleftrightarrow P = \bot \vee (\exists\, r.\ [\checkmark(r)] \in \mathcal{T}\ P \wedge Q\ r = \bot)\rangle$
⟨*proof*⟩

lemma $BOT\text{-}Seq_{ptick}$ [*simp*] : $\langle \bot \mathbin{;_{\checkmark}} P = \bot\rangle$ ⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) $Sync_{ptick}$*-is-BOT-iff* : $\langle P\ [\![S]\!]_{\checkmark}\ Q = \bot \longleftrightarrow P = \bot \vee Q = \bot\rangle$
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) $Sync_{ptick}$*-BOT* [*simp*] : $\langle P\ [\![S]\!]_{\checkmark}\ \bot = \bot\rangle$ **and** $BOT\text{-}Sync_{ptick}$ [*simp*] : $\langle \bot\ [\![S]\!]_{\checkmark}\ Q = \bot\rangle$
⟨*proof*⟩

7.1.2 The Laws of *STOP*

lemma Seq_{ptick}*-is-STOP-iff* :
$\langle P \mathbin{;_{\checkmark}} Q = STOP \longleftrightarrow \mathcal{T}\ P \subseteq insert\ []\ \{[\checkmark(r)] |\ r.\ True\} \wedge (\forall\, r.\ [\checkmark(r)] \in \mathcal{T}\ P \longrightarrow Q\ r = STOP)\rangle$ (**is** $\langle ?lhs \longleftrightarrow ?rhs\rangle$)

⟨*proof*⟩

lemma Seq_{ptick}*-is-STOP-iff-bis* :
‹$P \;;_{\checkmark}\; Q = STOP \longleftrightarrow SKIPS\ \{r.\ [\checkmark(r)] \in \mathcal{T}\ P\} \sqsubseteq_{DT} P \wedge (\forall r.\ [\checkmark(r)] \in \mathcal{T}\ P \longrightarrow Q\ r = STOP)$›
(**is** ‹*?lhs* ⟷ *?rhs*›)
⟨*proof*⟩

corollary $STOP\text{-}Seq_{ptick}$ [*simp*] : ‹$STOP \;;_{\checkmark}\; P = STOP$›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) $STOP\text{-}Sync_{ptick}\text{-}STOP$ [*simp*] : ‹$STOP\ [\![S]\!]_{\checkmark}\ STOP = STOP$›
⟨*proof*⟩

More powerful Laws **lemma** (**in** $Sync_{ptick}$*-locale*) $Inter_{ptick}$*-STOP* :
— Here, g is a free parameter.
‹$P\ |||_{\checkmark}\ STOP = RenamingTick\ (P\ ;\ STOP)$
$(\lambda r.\ the\ (tick\text{-}join\ r\ (g\ r)))$› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) $STOP\text{-}Inter_{ptick}$:
‹$STOP\ |||_{\checkmark}\ Q = RenamingTick\ (Q\ ;\ STOP)$
$(\lambda s.\ the\ (tick\text{-}join\ (g\ s)\ s))$›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) Par_{ptick}*-STOP* [*simp*] : ‹$P\ ||_{\checkmark}\ STOP = (if\ P = \bot\ then\ \bot\ else\ STOP)$›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) $STOP\text{-}Par_{ptick}$ [*simp*] : ‹$STOP\ ||_{\checkmark}\ P = (if\ P = \bot\ then\ \bot\ else\ STOP)$›
⟨*proof*⟩

7.1.3 The Laws of *SKIP*

Sequential Composition

SKIP is neutral for Seq_{ptick}!

lemma $SKIP\text{-}Seq_{ptick}$ [*simp*] : ‹$SKIP\ r \;;_{\checkmark}\; P = P\ r$›
⟨*proof*⟩

lemma Seq_{ptick}*-SKIP* [*simp*] : ‹$P \;;_{\checkmark}\; SKIP = P$›

⟨*proof*⟩

lemma *SKIPS-Seq*$_{ptick}$ [*simp*] : ‹*SKIPS R* ;$_{\checkmark}$ *P* = ⊓*r* ∈ *R*. *P r*›
⟨*proof*⟩

lemma *finite-ticks-Seq*$_{ptick}$ [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}$(*P* ;$_{\checkmark}$ *Q*)›
if ‹$\mathbb{F}_{\checkmark}$(*P*)› **and** ‹(⋀*r*. *r* ∈ ✓**s**(*P*) ⟹ $\mathbb{F}_{\checkmark}$(*Q r*))›
⟨*proof*⟩

lemma *finite-ticks-fun-Seq*$_{ptick}$*-bis* :
‹$\mathbb{F}_{\checkmark\Rightarrow}$(*f*) ⟹ (⋀*x r*. *r* ∈ ✓**s**(*f x*) ⟹ $\mathbb{F}_{\checkmark}$(*x*) ⟹ $\mathbb{F}_{\checkmark}$(*g x r*)) ⟹ $\mathbb{F}_{\checkmark\Rightarrow}$(λ*x*. *f x* ;$_{\checkmark}$ *g x*)›
⟨*proof*⟩

lemma *finite-ticks-fun-Seq*$_{ptick}$ [*finite-ticks-fun-simps*] :
— Big approximation.
‹$\mathbb{F}_{\checkmark\Rightarrow}$(*f*) ⟹ (⋀*r*. *r* ∈ (⋃*x*. ✓**s**(*f x*)) ⟹ $\mathbb{F}_{\checkmark\Rightarrow}$(λ*x*. *g x r*)) ⟹ $\mathbb{F}_{\checkmark\Rightarrow}$(λ*x*. *f x* ;$_{\checkmark}$ *g x*)›
⟨*proof*⟩

Synchronization Product

The generalization of the synchronization product was essentially motivated by the following theorem (in comparison to *SKIP r* ⟦*A*⟧ *SKIP s* = (*if r* = *s then SKIP r else STOP*)).

theorem (**in** *Sync*$_{ptick}$*-locale*) *SKIP-Sync*$_{ptick}$*-SKIP* [*simp*] :
‹*SKIP r* ⟦*A*⟧$_{\checkmark}$ *SKIP s* = (*case tick-join r s of* ⌊*r-s*⌋ ⇒ *SKIP r-s* | ◇ ⇒ *STOP*)›
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$*-locale*) *STOP-Sync*$_{ptick}$*-SKIP* [*simp*] : ‹*STOP* ⟦*A*⟧$_{\checkmark}$ *SKIP s* = *STOP*›
and *SKIP-Sync*$_{ptick}$*-STOP* [*simp*] : ‹*SKIP r* ⟦*A*⟧$_{\checkmark}$ *STOP* = *STOP*›
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$*-locale*) *Mprefix-Sync*$_{ptick}$*-SKIP* :
‹□*a* ∈ *A* → *P a* ⟦*S*⟧$_{\checkmark}$ *SKIP r* =
□*a* ∈ (*A* − *S*) → (*P a* ⟦*S*⟧$_{\checkmark}$ *SKIP r*)› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

corollary (**in** *Sync*$_{ptick}$*-locale*) *SKIP-Sync*$_{ptick}$*-Mprefix* :
‹*SKIP r* ⟦*S*⟧$_{\checkmark}$ □*b* ∈ *B* → *Q b* = □*b* ∈ (*B* − *S*) → (*SKIP r* ⟦*S*⟧$_{\checkmark}$ *Q b*)› (**is** ‹*?lhs* = *?rhs*›)

⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*locale*) *finite-ticks-*$Sync_{ptick}$ [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P\ [\![S]\!]_{\checkmark}\ Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P)$› **and** ‹$\mathbb{F}_{\checkmark}(Q)$›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*locale*) *finite-ticks-fun-*$Sync_{ptick}$ [*finite-ticks-fun-simps*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ [\![S]\!]_{\checkmark}\ g\ x)$›
⟨*proof*⟩

7.2 Associativity of Sequential Composition

lemma Seq_{ptick}-*assoc* : ‹$P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R) = P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R$›
for $P :: \langle('a, 'r)\ process_{ptick}\rangle$
and $Q :: \langle'r \Rightarrow ('a, 's)\ process_{ptick}\rangle$
and $R :: \langle's \Rightarrow ('a, 't)\ process_{ptick}\rangle$
⟨*proof*⟩

7.3 Distributivity of Non-Determinism

7.3.1 Sequential Composition

lemma Seq_{ptick}-*distrib-GlobalNdet-left* :
‹$P\ ;_{\checkmark}\ (\lambda r.\ \sqcap\ a{\in}A.\ Q\ a\ r) = (\mathit{if}\ A = \{\}\ \mathit{then}\ P\ ;_{\checkmark}\ (\lambda r.\ STOP)\ \mathit{else}\ \sqcap\ a{\in}A.\ (P\ ;_{\checkmark}\ Q\ a))$›
⟨*proof*⟩

lemma Seq_{ptick}-*distrib-GlobalNdet-right* : ‹$(\sqcap\ a{\in}A.\ P\ a)\ ;_{\checkmark}\ Q = \sqcap\ a{\in}A.\ (P\ a\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

lemma Seq_{ptick}-*distrib-Ndet-left* : ‹$P\ ;_{\checkmark}\ (\lambda r.\ Q\ r \sqcap R\ r) = (P\ ;_{\checkmark}\ Q) \sqcap (P\ ;_{\checkmark}\ R)$›
⟨*proof*⟩

lemma Seq_{ptick}-*distrib-Ndet-right* : ‹$P \sqcap Q\ ;_{\checkmark}\ R = (P\ ;_{\checkmark}\ R) \sqcap (Q\ ;_{\checkmark}\ R)$›
⟨*proof*⟩

7.3.2 Synchronization Product

context $Sync_{ptick}$-*locale* **begin**

lemma $Sync_{ptick}$-*distrib-GlobalNdet-left* :
‹$P\ [\![S]\!]_{\checkmark} \sqcap\ a{\in}A.\ Q\ a = (\mathit{if}\ A = \{\}\ \mathit{then}\ P\ [\![S]\!]_{\checkmark}\ STOP\ \mathit{else} \sqcap\ a{\in}A.\ (P\ [\![S]\!]_{\checkmark}\ Q\ a))$›

(**is** ‹*?lhs* = (*if* $A = \{\}$ *then* P $[\![S]\!]_{\checkmark}$ *STOP else ?rhs*)›)
⟨*proof*⟩

lemma $Sync_{ptick}$*-distrib-GlobalNdet-right* :
‹$\sqcap$ $a{\in}A$. P a $[\![S]\!]_{\checkmark}$ Q = (*if* $A = \{\}$ *then STOP* $[\![S]\!]_{\checkmark}$ Q *else* $\sqcap$ $a{\in}A$. (P a $[\![S]\!]_{\checkmark}$ Q))›
(**is** ‹*?lhs* = (*if* $A = \{\}$ *then STOP* $[\![S]\!]_{\checkmark}$ Q *else ?rhs*)›)
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) $Sync_{ptick}$*-GlobalNdet-cartprod*:
‹($\sqcap$ (a, b) $\in$ $A \times B$. (P a $[\![S]\!]_{\checkmark}$ Q b)) =
(*if* $A = \{\} \vee B = \{\}$ *then STOP else* ($\sqcap a \in A$. P a) $[\![S]\!]_{\checkmark}$ ($\sqcap b \in B$. Q b))›
⟨*proof*⟩

lemma $Sync_{ptick}$*-distrib-Ndet-left* :
‹P $[\![S]\!]_{\checkmark}$ $Q \sqcap R = (P$ $[\![S]\!]_{\checkmark}$ $Q) \sqcap (P$ $[\![S]\!]_{\checkmark}$ $R)$›
⟨*proof*⟩

corollary $Sync_{ptick}$*-distrib-Ndet-right* :
‹$P \sqcap Q$ $[\![S]\!]_{\checkmark}$ $R = (P$ $[\![S]\!]_{\checkmark}$ $R) \sqcap (Q$ $[\![S]\!]_{\checkmark}$ $R)$›
⟨*proof*⟩

end

Chapter 8

Communications

8.1 Step Laws

8.1.1 Sequential Composition

lemma *Mprefix-Seq$_{ptick}$*: ‹$\Box a \in A \rightarrow P\ a\ ;_{\checkmark}\ Q = \Box a \in A \rightarrow (P\ a\ ;_{\checkmark}\ Q)$› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

8.1.2 Synchronization Product

lemma (**in** *Sync$_{ptick}$-locale*) *Mprefix-Sync$_{ptick}$-Mprefix-bis* :
‹$\Box a{\in}(A \cup A') \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}(B \cup B') \rightarrow Q\ b =$
$(\Box a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}(B \cup B') \rightarrow Q\ b))\ \Box$
$(\Box b{\in}B \rightarrow (\Box a{\in}(A \cup A') \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box x{\in}(A' \cap B') \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))$›
(**is** ‹*?lhs1* $[\![S]\!]_{\checkmark}$ *?lhs2* = *?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*›)
if *sets-assms*: ‹$A \cap S = \{\}$› ‹$A' \subseteq S$› ‹$B \cap S = \{\}$› ‹$B' \subseteq S$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *Mprefix-Sync$_{ptick}$-Mprefix*:
— This version is easier to use.
‹$\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b =$
$(\Box a{\in}(A - S) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b))\ \Box$
$(\Box b{\in}(B - S) \rightarrow (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box x{\in}(A \cap B \cap S) \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *Mprefix-Sync$_{ptick}$-Mprefix-for-procomata*:
— Specialized version for Proc-Omata.
‹$\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b =$
$(\Box a{\in}(A - S - B) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b))\ \Box$
$(\Box b{\in}(B - S - A) \rightarrow (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box x{\in}(A \cap B - S) \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b) \sqcap (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ x))\ \Box$

$(\Box x{\in}(A \cap B \cap S) \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))\rangle$
⟨*proof*⟩

unbundle *option-type-syntax*

8.2 Extended step Laws

8.2.1 Sequential Composition

lemma *Mndetprefix-Seq*$_{ptick}$: $\langle \sqcap a \in A \rightarrow P\ a\ ;_{\checkmark}\ Q = \sqcap a \in A \rightarrow (P\ a\ ;_{\checkmark}\ Q)\rangle$
⟨*proof*⟩

8.2.2 Synchronization Product

Behaviour of *SKIPS*

lemma (**in** *Sync*$_{ptick}$*-locale*) *SKIPS-Sync*$_{ptick}$*-SKIPS* :
$\langle SKIPS\ R\ [\![A]\!]_{\checkmark}\ SKIPS\ S = \sqcap(r,\ s) \in R \times S.\ (case\ r \otimes\checkmark\ s\ of\ \lfloor r\text{-}s \rfloor \Rightarrow SKIP\ r\text{-}s \mid \Diamond \Rightarrow STOP)\rangle$
⟨*proof*⟩

In order for the right-hand side to be rewritten as a SKIPS, an assumption is required: the ticks involved must be able to be combined.

lemma *GlobalNdet-prod-SKIP-is-SKIPS* :
$\langle \sqcap(r,\ s) \in R \times S.\ SKIP\ \lceil tick\text{-}join\ r\ s \rceil =$
$SKIPS\ ((the \circ (\lambda(r,\ s).\ tick\text{-}join\ r\ s))\ `\ (R \times S))\rangle$
⟨*proof*⟩

lemma *GlobalNdet-prod-case-SKIP-STOP-is-GlobalNdet-prod-SKIP-iff* :
$\langle \sqcap(r,\ s) \in R \times S.\ (case\ tick\text{-}join\ r\ s\ of\ \Diamond \Rightarrow STOP \mid \lfloor r\text{-}s \rfloor \Rightarrow SKIP\ r\text{-}s) =$
$\sqcap(r,\ s) \in R \times S.\ SKIP\ \lceil tick\text{-}join\ r\ s \rceil$
$\longleftrightarrow (\forall r\ s.\ r \in R \longrightarrow s \in S \longrightarrow tick\text{-}join\ r\ s \neq \Diamond)\rangle$
(**is** $\langle ?lhs1 = ?lhs2 \longleftrightarrow ?rhs\rangle$)
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$*-locale*) *SKIPS-Sync*$_{ptick}$*-SKIPS-bis* :
$\langle SKIPS\ R\ [\![A]\!]_{\checkmark}\ SKIPS\ S = SKIPS\ ((the \circ (\lambda(r,\ s).\ r \otimes\checkmark\ s))\ `\ (R \times S))\rangle$
if $\langle \bigwedge r\ s.\ r \in R \Longrightarrow s \in S \Longrightarrow r \otimes\checkmark\ s \neq \Diamond\rangle$
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$*-locale*)
SKIPS-Sync$_{ptick}$*-STOP* [*simp*] : $\langle SKIPS\ R\ [\![A]\!]_{\checkmark}\ STOP = STOP\rangle$
and *STOP-Sync*$_{ptick}$*-SKIPS* [*simp*] : $\langle STOP\ [\![A]\!]_{\checkmark}\ SKIPS\ S = STOP\rangle$
⟨*proof*⟩

Derived step Laws with Non-Determinism

context $Sync_{ptick}$-*locale* **begin**

lemma $Mprefix\text{-}Inter_{ptick}\text{-}Mprefix$:
$\langle \Box a{\in}A \to P\ a\ |||_{\checkmark}\ \Box b{\in}B \to Q\ b =$
$(\Box a{\in}A \to (P\ a\ |||_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box\ (\Box b{\in}B \to (\Box a \in A \to P\ a\ |||_{\checkmark}\ Q\ b))\rangle$
⟨*proof*⟩

lemma $Mprefix\text{-}Par_{ptick}\text{-}Mprefix$: $\langle \Box a{\in}A \to P\ a\ ||_{\checkmark}\ \Box b{\in}B \to Q\ b = \Box x{\in}(A \cap B) \to (P\ x\ ||_{\checkmark}\ Q\ x)\rangle$
⟨*proof*⟩

lemma $Mprefix\text{-}Sync_{ptick}\text{-}Mprefix\text{-}subset$:
$\langle [\![A \subseteq S;\ B \subseteq S]\!] \Longrightarrow \Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b\ = \Box x{\in}(A \cap B) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)\rangle$
⟨*proof*⟩

lemma $Mprefix\text{-}Sync_{ptick}\text{-}Mprefix\text{-}indep$:
$\langle [\![A \cap S = \{\};\ B \cap S = \{\}]\!] \Longrightarrow$
$\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b =$
$(\Box a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box\ (\Box b{\in}B \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\rangle$
⟨*proof*⟩

lemma $Mprefix\text{-}Sync_{ptick}\text{-}Mprefix\text{-}left$:
$\langle [\![A \cap S = \{\};\ B \subseteq S]\!] \Longrightarrow \Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b = \Box a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b)\rangle$
⟨*proof*⟩

lemma $Mprefix\text{-}Sync_{ptick}\text{-}Mprefix\text{-}right$:
$\langle [\![A \subseteq S;\ B \cap S = \{\}]\!] \Longrightarrow \Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b = \Box b{\in}B \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)\rangle$
⟨*proof*⟩

lemma $Mprefix\text{-}Sync_{ptick}\text{-}STOP$: $\langle \Box a \in A \to P\ a\ [\![S]\!]_{\checkmark}\ STOP = \Box a \in (A - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ STOP)\rangle$
⟨*proof*⟩

lemma $STOP\text{-}Sync_{ptick}\text{-}Mprefix$: $\langle STOP\ [\![S]\!]_{\checkmark}\ \Box b \in B \to Q\ b = \Box b \in (B - S) \to (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)\rangle$
⟨*proof*⟩

Mixing deterministic and non deterministic prefix choices **lemma** $Mndetprefix\text{-}Sync_{ptick}\text{-}Mprefix$:
$\langle (\sqcap a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b) =$
$(\ \ if\ A = \{\}\ then\ STOP\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b)$
$\ \ else\ \sqcap a{\in}A.\ (if\ a \in S\ then\ STOP\ else\ (a \to (P\ a\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b))))\ \Box$
$\Box b{\in}(B - S) \to ((a \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(if\ a \in B \cap S\ then\ (a \to (P\ a\ [\![S]\!]_{\checkmark}\ Q\ a))\ else\ STOP))\rangle$

⟨*proof*⟩

lemma *Mprefix-Sync*$_{ptick}$*-Mndetprefix* :
‹(□$a \in A \to P\ a$) [[S]]$_{\checkmark}$ (⊓$b \in B \to Q\ b$) =
(*if* $B = \{\}$ *then* (□$a \in A \to P\ a$) [[S]]$_{\checkmark}$ *STOP*
else ⊓b∈B. (*if* $b \in S$ *then STOP else* ($b \to$ ((□$a \in A \to P\ a$) [[S]]$_{\checkmark}$ $Q\ b$))) □
(□a∈($A - S$) $\to$ ($P\ a$ [[S]]$_{\checkmark}$ ($b \to Q\ b$))) □
(*if* $b \in A \cap S$ *then* ($b \to$ ($P\ b$ [[S]]$_{\checkmark}$ $Q\ b$)) *else STOP*))›
⟨*proof*⟩

In particular, we can obtain the theorem for *Mndetprefix* synchronized with *STOP*.

lemma *Mndetprefix-Sync*$_{ptick}$*-STOP* :
‹(⊓$a \in A \to P\ a$) [[S]]$_{\checkmark}$ *STOP* =
(*if* $A \cap S = \{\}$ *then* ⊓$a \in A \to$ ($P\ a$ [[S]]$_{\checkmark}$ *STOP*)
else (⊓$a \in (A - S) \to$ ($P\ a$ [[S]]$_{\checkmark}$ *STOP*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if* $A \cap S = \{\}$ *then ?rhs1 else ?rhs2* ⊓ *STOP*)›)
⟨*proof*⟩

lemma *STOP-Sync*$_{ptick}$*-Mndetprefix* :
‹*STOP* [[S]]$_{\checkmark}$ (⊓$b \in B \to Q\ b$) =
(*if* $B \cap S = \{\}$ *then* ⊓$b \in B \to$ (*STOP* [[S]]$_{\checkmark}$ $Q\ b$)
else (⊓$b \in (B - S) \to$ (*STOP* [[S]]$_{\checkmark}$ $Q\ b$)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if* $B \cap S = \{\}$ *then ?rhs1 else ?rhs2* ⊓ *STOP*)›)
⟨*proof*⟩

corollary *Mndetprefix-Sync*$_{ptick}$*-Mprefix-subset* :
‹(⊓$a \in A \to P\ a$) [[S]]$_{\checkmark}$ (□$b \in B \to Q\ b$) =
(*if* $A \subseteq B$ *then* ⊓ $a \in A \to$ ($P\ a$ [[S]]$_{\checkmark}$ $Q\ a$)
else (⊓$a \in (A \cap B) \to$ ($P\ a$ [[S]]$_{\checkmark}$ $Q\ a$)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if* $A \subseteq B$ *then ?rhs1 else ?rhs2*)›) **if** ‹$A \subseteq S$› ‹$B \subseteq S$›
⟨*proof*⟩

corollary *Mprefix-Sync*$_{ptick}$*-Mndetprefix-subset* :
‹□$a \in A \to P\ a$ [[S]]$_{\checkmark}$ ⊓$b \in B \to Q\ b$ =
(*if* $B \subseteq A$ *then* ⊓$b \in B \to$ ($P\ b$ [[S]]$_{\checkmark}$ $Q\ b$)
else (⊓$b \in (A \cap B) \to$ ($P\ b$ [[S]]$_{\checkmark}$ $Q\ b$)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if* $B \subseteq A$ *then ?rhs1 else ?rhs2*)›) **if** ‹$A \subseteq S$› ‹$B \subseteq S$›
⟨*proof*⟩

corollary *Mndetprefix-Sync*$_{ptick}$*-Mprefix-indep* :
‹(⊓$a \in A \to P\ a$) [[S]]$_{\checkmark}$ (□$b \in B \to Q\ b$) =
(*if* $A = \{\}$ *then* □b∈$B \to$ (*STOP* [[S]]$_{\checkmark}$ $Q\ b$)
else ⊓a∈A. ($a \to$ ($P\ a$ [[S]]$_{\checkmark}$ (□$b \in B \to Q\ b$))) □
(□b∈$B \to$ (($a \to P\ a$) [[S]]$_{\checkmark}$ $Q\ b$)))›

if $\langle A \cap S = \{\}\rangle$ **and** $\langle B \cap S = \{\}\rangle$
⟨*proof*⟩

corollary *Mprefix-Sync*$_{ptick}$*-Mndetprefix-indep* :
$\langle(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B \to Q\ b) =$
(*if* $B = \{\}$ *then* $\Box a \in A \to (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
else $\sqcap b{\in}B.\ (b \to ((\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (b \to Q\ b))))\rangle$
if $\langle A \cap S = \{\}\rangle\ \langle B \cap S = \{\}\rangle$
⟨*proof*⟩

corollary *Mndetprefix-Sync*$_{ptick}$*-Mprefix-left* :
$\langle(\sqcap a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b) =$
(*if* $A = \{\}$ *then* $STOP\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b)$
else $\sqcap a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b)))\rangle$
if $\langle A \cap S = \{\}\rangle$ **and** $\langle B \subseteq S\rangle$
⟨*proof*⟩

corollary *Mndetprefix-Sync*$_{ptick}$*-Mprefix-right* :
$\langle(\sqcap a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b) =$
(*if* $A = \{\}$ *then* $STOP\ [\![S]\!]_{\checkmark}\ (\Box b \in B \to Q\ b)$
else $\Box b{\in}B \to ((\sqcap a{\in}A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))\rangle$
if $\langle A \subseteq S\rangle$ **and** $\langle B \cap S = \{\}\rangle$
⟨*proof*⟩

corollary *Mprefix-Sync*$_{ptick}$*-Mndetprefix-left* :
$\langle(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B \to Q\ b) =$
(*if* $B = \{\}$ *then* $(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ STOP$
else $\Box a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B \to Q\ b)))\rangle$
if $\langle A \cap S = \{\}\rangle\ \langle B \subseteq S\rangle$
⟨*proof*⟩

corollary *Mprefix-Sync*$_{ptick}$*-Mndetprefix-right* :
$\langle(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B \to Q\ b) =$
(*if* $B = \{\}$ *then* $(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ STOP$
else $\sqcap b{\in}B \to ((\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))\rangle$
if $\langle A \subseteq S\rangle\ \langle B \cap S = \{\}\rangle$
⟨*proof*⟩

corollary *Mndetprefix-Par*$_{ptick}$*-Mprefix* :
$\langle\sqcap a \in A \to P\ a\ ||_{\checkmark}\ \Box b \in B \to Q\ b =$
(*if* $A \subseteq B$ *then* $\sqcap a \in A \to (P\ a\ ||_{\checkmark}\ Q\ a)$ *else* $(\sqcap a \in (A \cap B) \to (P\ a\ ||_{\checkmark}\ Q\ a))$
$\sqcap\ STOP)\rangle$
⟨*proof*⟩

corollary *Mprefix-Par*$_{ptick}$*-Mndetprefix* :

$\langle \Box a \in A \to P\ a\ ||_{\checkmark}\ \sqcap b \in B \to Q\ b =$
$(if\ B \subseteq A\ then\ \sqcap b \in B \to (P\ b\ ||_{\checkmark}\ Q\ b)\ else\ (\sqcap b \in (A \cap B) \to (P\ b\ ||_{\checkmark}\ Q\ b)) \sqcap STOP)\rangle$
⟨*proof*⟩

corollary *Mndetprefix-Inter*$_{ptick}$*-Mprefix* :
$\langle \sqcap a \in A \to P\ a\ |||_{\checkmark}\ \Box b \in B \to Q\ b =$
$(\ \ if\ A = \{\}\ then\ \Box b \in B \to RenamingTick\ (Q\ b\ ;\ STOP)\ (\lambda s.\ the\ (tick\text{-}join\ (g\ s)\ s))$
$\quad else\ \sqcap a{\in}A.\ (a \to (P\ a\ |||_{\checkmark}\ \Box b \in B \to Q\ b))\ \Box$
$\qquad (\Box b{\in}B \to (a \to P\ a\ |||_{\checkmark}\ Q\ b)))\rangle$
⟨*proof*⟩

corollary *Mprefix-Inter*$_{ptick}$*-Mndetprefix* :
$\langle \Box a \in A \to P\ a\ |||_{\checkmark}\ \sqcap b \in B \to Q\ b =$
$(\ \ if\ B = \{\}\ then\ \Box a \in A \to RenamingTick\ (P\ a\ ;\ STOP)\ (\lambda r.\ the\ (tick\text{-}join\ r\ (g\ r)))$
$\quad else\ \sqcap b{\in}B.\ (b \to (\Box a \in A \to P\ a\ |||_{\checkmark}\ Q\ b))\ \Box$
$\qquad (\Box a{\in}A \to (P\ a\ |||_{\checkmark}\ b \to Q\ b)))\rangle$
⟨*proof*⟩

Mixing two non deterministic prefix choices **lemma** *Mndetprefix-Sync*$_{ptick}$*-Mndetprefix* :
$\langle \sqcap a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \to Q\ b =$
$(\ \ if\ A = \{\}\ then\quad if\ B \cap S = \{\}\ then\ \sqcap b{\in}B \to (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
$\qquad\qquad else\ (\sqcap x \in (B - S) \to (STOP\ [\![S]\!]_{\checkmark}\ Q\ x)) \sqcap STOP$
$\quad else\quad if\ B = \{\}\ then\quad if\ A \cap S = \{\}\ then\ \sqcap a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
$\qquad\qquad\qquad else\ (\sqcap x \in (A - S) \to (P\ x\ [\![S]\!]_{\checkmark}\ STOP)) \sqcap STOP$
$\qquad else\ \sqcap b{\in}B.\ \sqcap a{\in}A.$
$\qquad\quad (if\ a \in S\ then\ STOP\ else\ a \to (P\ a\ [\![S]\!]_{\checkmark}\ b \to Q\ b))\ \Box$
$\qquad\quad (if\ b \in S\ then\ STOP\ else\ b \to (a \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$\qquad\quad (if\ a = b \wedge b \in S\ then\ b \to (P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)\ else\ STOP))\rangle$
(**is** ‹*?lhs* = (*if A* = {} *then if B* ∩ *S* = {} *then ?mv-right else ?mv-right′* ⊓ *STOP*
else if B = {} *then if A* ∩ *S* = {} *then ?mv-left else ?mv-left′* ⊓ *STOP*
else ?huge-mess)›)
⟨*proof*⟩

lemma *Mndetprefix-Sync*$_{ptick}$*-Mndetprefix-subset* :
$\langle \sqcap a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \to Q\ b =$
$(\ \ if\ \exists\, b.\ A = \{b\} \wedge B = \{b\}$
$\quad then\ (THE\ b.\ B = \{b\}) \to (P\ (THE\ a.\ A = \{a\})\ [\![S]\!]_{\checkmark}\ Q\ (THE\ b.\ B = \{b\}))$
$\quad else\ (\sqcap x \in (A \cap B) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)) \sqcap STOP)\rangle$
(**is** ‹*?lhs* = (*if ?cond then ?rhs1 else ?rhs2*)›) **if** ‹*A* ⊆ *S*› ‹*B* ⊆ *S*›
⟨*proof*⟩

lemma *Mndetprefix-Sync$_{ptick}$-Mndetprefix-indep* :
‹$A \cap S = \{\} \Longrightarrow B \cap S = \{\} \Longrightarrow$
$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b =$
(*if* $A = \{\}$ *then* $\sqcap b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
else *if* $B = \{\}$ *then* $\sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
else $\sqcap b{\in}B.\ \sqcap a{\in}A.$
$((a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ b))) \ \Box$
$((b \rightarrow (a \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))))$›
⟨*proof*⟩

lemma *Mndetprefix-Sync$_{ptick}$-Mndetprefix-left* :
‹$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b = \sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$A \cap S = \{\}$› ‹$B \subseteq S$›
⟨*proof*⟩

end

corollary (**in** *Sync$_{ptick}$-locale*) *Mndetprefix-Sync$_{ptick}$-Mndetprefix-right* :
‹$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b = \sqcap b{\in}B \rightarrow (\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$A \subseteq S$› ‹$B \cap S = \{\}$›
⟨*proof*⟩

8.3 Read and Write Laws

8.3.1 Sequential Composition

lemma *read-Seq$_{ptick}$* : ‹$c?a{\in}A \rightarrow P\ a\ ;_{\checkmark}\ Q = c?a{\in}A \rightarrow (P\ a\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

lemma *write0-Seq$_{ptick}$* : ‹$a \rightarrow P\ ;_{\checkmark}\ Q = a \rightarrow (P\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

lemma *ndet-write-Seq$_{ptick}$* : ‹$c!!a{\in}A \rightarrow P\ a\ ;_{\checkmark}\ Q = c!!a{\in}A \rightarrow (P\ a\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

lemma *write-Seq$_{ptick}$* : ‹$c!a \rightarrow P\ ;_{\checkmark}\ Q = c!a \rightarrow (P\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

8.3.2 Synchronization Product

General Laws

context *Sync$_{ptick}$-locale* **begin**

read **lemma** *read-Sync$_{ptick}$-read* :
— This is the general case.
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
($c?a{\in}(A - c -` S) \to (P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)) \Box$
($d?b{\in}(B - d -` S) \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
($\Box x{\in}(c\ `\ A \cap d\ `\ B \cap S) \to (P$ (*inv-into* $A\ c\ x$) $[\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d\ x$)))›
(**is** ‹*?lhs* = *?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*›)
if ‹$c\ `\ A \cap S = \{\} \vee c\ `\ A \subseteq S \vee$ *inj-on* $c\ A$›
‹$d\ `\ B \cap S = \{\} \vee d\ `\ B \subseteq S \vee$ *inj-on* $d\ B$›
— Assumptions may seem strange, but the motivation is that when $A \setminus c -` S \neq \{\}$ (which is equivalent to $\neg\ c\ `\ A \subseteq S$), we need to ensure that *inv-into* ($A \setminus c -` S$) $c$ is equal to *inv-into* $A\ c$. This requires $A \setminus c -` S = A$ (which is equivalent to $c\ `\ A \cap S = \{\}$) or *inj-on* $c\ A$. We need obviously a similar assumption for B.
⟨*proof*⟩

Enforce read **lemma** *read-Sync$_{ptick}$-read-forced-read-left* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
($c?a{\in}(A - c -` S) \to (P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)) \Box$
($d?b{\in}(B - d -` S) \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
($c?x{\in}(A \cap c -` (d\ `\ B \cap S)) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$)›
(**is** ‹*?lhs* = *?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*›)
if ‹$c\ `\ A \cap S = \{\} \vee$ *inj-on* $c\ A$›
‹$d\ `\ B \cap S = \{\} \vee$ *inj-on* $d\ B$›
‹$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow d\ b \in S \Longrightarrow a = b$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *read-Sync$_{ptick}$-read-forced-read-right*:
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
($c?a{\in}(A - c -` S) \to (P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)) \Box$
($d?b{\in}(B - d -` S) \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
($d?x{\in}(B \cap d -` (c\ `\ A \cap S)) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$)›
(**is** ‹*?lhs* = *?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*›)
if ‹$c\ `\ A \cap S = \{\} \vee$ *inj-on* $c\ A$›
‹$d\ `\ B \cap S = \{\} \vee$ *inj-on* $d\ B$›
‹$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow d\ b \in S \Longrightarrow a = b$›
⟨*proof*⟩

Special Cases **lemma** *read-Sync$_{ptick}$-read-subset* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
$\Box x{\in}(c\ `\ A \cap d\ `\ B) \to (P$ (*inv-into* $A\ c\ x$) $[\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d\ x$))›
if ‹$c\ `\ A \subseteq S$› ‹$d\ `\ B \subseteq S$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-subset-forced-read-left* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = c?x{\in}(A \cap c -` d\ `\ B) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
if ‹$c\ `\ A \subseteq S$› ‹$d\ `\ B \subseteq S$› ‹*inj-on* $c\ A$› ‹*inj-on* $d\ B$›
‹$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow d\ b \in S \Longrightarrow a = b$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-subset-forced-read-right* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = d?x{\in}(B \cap d -` c `` A) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
if ‹$c `` A \subseteq S$› ‹$d `` B \subseteq S$› ‹*inj-on c A*› ‹*inj-on d B*›
‹$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow d\ b \in S \Longrightarrow a = b$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-indep* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
$(c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (d?b{\in}B \to Q\ b))) \Box (d?b{\in}B \to ((c?a{\in}A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))$›
if ‹$c `` A \cap S = \{\}$› ‹$d `` B \cap S = \{\}$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-left* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (d?b{\in}B \to Q\ b))$›
if ‹$c `` A \cap S = \{\}$› ‹$d `` B \subseteq S$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-right* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = d?b{\in}B \to ((c?a{\in}A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b)$›
if ‹$c `` A \subseteq S$› ‹$d `` B \cap S = \{\}$›
⟨*proof*⟩

corollary *read-Par$_{ptick}$-read* :
‹$c?a{\in}A \to P\ a\ ||_{\checkmark}\ d?b{\in}B \to Q\ b =$
$\Box x{\in}(c `` A \cap d `` B) \to (P\ (\mathit{inv\text{-}into}\ A\ c\ x)\ ||_{\checkmark}\ Q\ (\mathit{inv\text{-}into}\ B\ d\ x))$›
⟨*proof*⟩

corollary *read-Par$_{ptick}$-read-forced-read-left* :
‹$[\![\mathit{inj\text{-}on}\ c\ A;\ \mathit{inj\text{-}on}\ d\ B;\ \bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow a = b]\!] \Longrightarrow$
$c?a{\in}A \to P\ a\ ||_{\checkmark}\ d?b{\in}B \to Q\ b = c?x{\in}(A \cap c -` d `` B) \to (P\ x\ ||_{\checkmark}\ Q\ x)$›
⟨*proof*⟩

corollary *read-Par$_{ptick}$-read-forced-read-right* :
‹$[\![\mathit{inj\text{-}on}\ c\ A;\ \mathit{inj\text{-}on}\ d\ B;\ \bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow a = b]\!] \Longrightarrow$
$c?a{\in}A \to P\ a\ ||_{\checkmark}\ d?b{\in}B \to Q\ b = d?x{\in}(B \cap d -` c `` A) \to (P\ x\ ||_{\checkmark}\ Q\ x)$›
⟨*proof*⟩

corollary *read-Inter$_{ptick}$-read* :
‹$[\![\mathit{inj\text{-}on}\ c\ A;\ \mathit{inj\text{-}on}\ d\ B;\ \bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow a = b]\!] \Longrightarrow$
$c?a{\in}A \to P\ a\ |||_{\checkmark}\ d?b{\in}B \to Q\ b =$
$(c?a{\in}A \to (P\ a\ |||_{\checkmark}\ d?b{\in}B \to Q\ b)) \Box (d?b{\in}B \to (c?a{\in}A \to P\ a\ |||_{\checkmark}\ Q\ b))$›
⟨*proof*⟩

Same channel Some results can be rewritten when we have the same channel.

lemma *read-Sync$_{ptick}$-read-forced-read-same-chan* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ c?b{\in}B \to Q\ b =$
$(c?a{\in}(A - c -` S) \to (P\ a\ [\![S]\!]_{\checkmark}\ c?b{\in}B \to Q\ b)) \Box$
$(c?b{\in}(B - c -` S) \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
$(c?x{\in}(A \cap B \cap c -` S) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))$›
(**is** ‹*?lhs* = *?rhs1* □ *?rhs2* □ *?rhs3*›)
if ‹c ` $A \cap S = \{\} \vee$ *inj-on c A*› ‹$c$ ` $B \cap S = \{\} \vee$ *inj-on c B*›
‹$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = c\ b \Longrightarrow c\ b \in S \Longrightarrow a = b$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-forced-read-same-chan-weaker* :
— Easier with a stronger assumption.
‹*inj-on c* $(A \cup B) \Longrightarrow$
$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ c?b{\in}B \to Q\ b =$
$(c?a{\in}(A - c -` S) \to (P\ a\ [\![S]\!]_{\checkmark}\ c?b{\in}B \to Q\ b)) \Box$
$(c?b{\in}(B - c -` S) \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
$(c?x{\in}(A \cap B \cap c -` S) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-read-subset-forced-read-same-chan* :
— In the subset case, the assumption *inj-on c* $(A \cup B)$ is equivalent. The result is not weaker anymore.
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ c?b{\in}B \to Q\ b = c?x{\in}(A \cap B) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
if ‹c ` $A \subseteq S$› ‹$c$ ` $B \subseteq S$› ‹*inj-on c* $(A \cup B)$›
⟨*proof*⟩

read **and** *ndet-write*. **lemma** *ndet-write-Sync$_{ptick}$-read* :
‹$c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
(*if* $A = \{\}$ *then STOP* $[\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b$
else $\sqcap a{\in}c$ ` A. (*if* $a \in S$ *then STOP else* $a \to (P$ (*inv-into A c a*) $[\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)) \Box$
$(\Box b{\in}(d$ ` $B - S) \to (a \to P$ (*inv-into A c a*) $[\![S]\!]_{\checkmark}\ Q$ (*inv-into B d b*))) □
(*if* $a \in d$ ` $B \cap S$ *then* $a \to (P$ (*inv-into A c a*) $[\![S]\!]_{\checkmark}\ Q$ (*inv-into B d a*)) *else STOP*)›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-ndet-write* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b =$
(*if* $B = \{\}$ *then* $c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ STOP$
else $\sqcap b{\in}d$ ` B. (*if* $b \in S$ *then STOP else* $b \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into B d b*))) □
$(\Box a{\in}(c$ ` $A - S) \to (P$ (*inv-into A c a*) $[\![S]\!]_{\checkmark}\ b \to Q$ (*inv-into B d b*))) □
(*if* $b \in c$ ` $A \cap S$ *then* $b \to (P$ (*inv-into A c b*) $[\![S]\!]_{\checkmark}\ Q$ (*inv-into B d b*)) *else STOP*)›

⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-read-subset* :
‹c ‘ $A \subseteq S \Longrightarrow d$ ‘ $B \subseteq S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
(*if* c ‘ $A \subseteq d$ ‘ B *then* $\sqcap a{\in}c$ ‘ $A \rightarrow (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ a))$
else $(\sqcap a{\in}(c$ ‘ $A \cap d$ ‘ $B) \rightarrow (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ a))) \sqcap \textit{STOP})$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-ndet-write-subset* :
‹c ‘ $A \subseteq S \Longrightarrow d$ ‘ $B \subseteq S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* d ‘ $B \subseteq c$ ‘ A *then* $\sqcap b{\in}d$ ‘ $B \rightarrow (P\ (\textit{inv-into}\ A\ c\ b)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ b))$
else $(\sqcap b{\in}(c$ ‘ $A \cap d$ ‘ $B) \rightarrow (P\ (\textit{inv-into}\ A\ c\ b)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ b))) \sqcap \textit{STOP})$›
⟨*proof*⟩
lemma *ndet-write-Sync$_{ptick}$-read-subset-same-chan*:
‹$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ c?b{\in}B \rightarrow Q\ b =$
(*if* $A \subseteq B$ *then* $c!!a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ Q\ a)$ *else* $(c!!a{\in}(A \cap B) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ Q\ a)) \sqcap \textit{STOP})$›
if ‹c ‘ $A \subseteq S$› ‹c ‘ $B \subseteq S$› ‹*inj-on* c $(A \cup B)$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *read-Sync$_{ptick}$-ndet-write-subset-same-chan*:
‹$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ c!!b{\in}B \rightarrow Q\ b =$
(*if* $B \subseteq A$ *then* $c!!b{\in}B \rightarrow (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b)$ *else* $(c!!b{\in}(A \cap B) \rightarrow (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b)) \sqcap \textit{STOP})$›
if ‹c ‘ $A \subseteq S$› ‹c ‘ $B \subseteq S$› ‹*inj-on* c $(A \cup B)$›
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-read-indep* :
‹c ‘ $A \cap S = \{\} \Longrightarrow d$ ‘ $B \cap S = \{\} \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
(*if* $A = \{\}$ *then* $d?b{\in}B \rightarrow\ (\textit{STOP}\ [\![S]\!]_{\checkmark}\ Q\ b)$
else $\sqcap a{\in}c$ ‘ $A.\ (a \rightarrow (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b)) \Box$
$(d?b{\in}B \rightarrow (a \rightarrow P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ b)))$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-ndet-write-indep* :
‹c ‘ $A \cap S = \{\} \Longrightarrow d$ ‘ $B \cap S = \{\} \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* $B = \{\}$ *then* $c?a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \textit{STOP})$
else $\sqcap b{\in}d$ ‘ $B.\ (b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ b))) \Box$
$(c?a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (\textit{inv-into}\ B\ d\ b))))$›

⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-read-left* :
‹$c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = c!!a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹c ‘ $A \cap S = \{\}$› ‹d ‘ $B \subseteq S$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-ndet-write-left* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b = c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹c ‘ $A \cap S = \{\}$› ‹d ‘ $B \subseteq S$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *ndet-write-Sync$_{ptick}$-read-right* :
‹$c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = d?b{\in}B \to (c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$›
if ‹c ‘ $A \subseteq S$› ‹d ‘ $B \cap S = \{\}$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *read-Sync$_{ptick}$-ndet-write-right* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b = d!!b{\in}B \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$›
if ‹c ‘ $A \subseteq S$› ‹d ‘ $B \cap S = \{\}$›
⟨*proof*⟩

read **and** *write*. **lemma** *write-Sync$_{ptick}$-read* :
‹$c!a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
($if\ c\ a \in S\ then\ STOP\ else\ c!a \to (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)$) $\Box$
($\Box b{\in}(d$ ‘ $B - S) \to (c!a \to P\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ b))$) $\Box$
($if\ c\ a \in d$ ‘ $B \cap S\ then\ c!a \to (P\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ (c\ a)))\ else\ STOP$)›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-write* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!b \to Q =$
($if\ d\ b \in S\ then\ STOP\ else\ d!b \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q)$) $\Box$
($\Box a{\in}(c$ ‘ $A - S) \to (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ d!b \to Q)$) $\Box$
($if\ d\ b \in c$ ‘ $A \cap S\ then\ d!b \to (P\ (inv\text{-}into\ A\ c\ (d\ b))\ [\![S]\!]_{\checkmark}\ Q)\ else\ STOP$)›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-read-subset* :
‹$c\ a \in S \Longrightarrow d$ ‘ $B \subseteq S \Longrightarrow$
$c!a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
($if\ c\ a \in d$ ‘ $B\ then\ c!a \to (P\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ (c\ a)))\ else\ STOP$)›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-write-subset* :
‹c ‘ $A \subseteq S \Longrightarrow d\ b \in S \Longrightarrow$
$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!b \to Q =$
($if\ d\ b \in c$ ‘ $A\ then\ d!b \to (P\ (inv\text{-}into\ A\ c\ (d\ b))\ [\![S]\!]_{\checkmark}\ Q)\ else\ STOP$)›

⟨*proof*⟩

lemma *write-Sync$_{ptick}$-read-subset-same-chan*:
‹$c\ a \in S \Longrightarrow c\ `\ B \subseteq S \Longrightarrow$ *inj-on* c (*insert* $a\ B$) $\Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ c?b{\in}B \rightarrow Q\ b = (\textit{if}\ a \in B\ \textit{then}\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ a)\ \textit{else}\ STOP)$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-write-subset-same-chan*:
‹$c\ `\ A \subseteq S \Longrightarrow c\ b \in S \Longrightarrow$ *inj-on* c (*insert* $b\ A$) $\Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ c!b \rightarrow Q = (\textit{if}\ b \in A\ \textit{then}\ c!b \rightarrow (P\ b\ [\![S]\!]_{\checkmark}\ Q)\ \textit{else}\ STOP)$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-read-indep* :
‹$c\ a \notin S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
$(c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b)) \Box (d?b{\in}B \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ b))$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-write-indep* :
‹$c\ `\ A \cap S = \{\} \Longrightarrow d\ b \notin S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q =$
$(d!b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)) \Box (c?a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q))$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-read-left* :
‹$c\ a \notin S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b = c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b)$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-write-left* :
‹$c\ `\ A \cap S = \{\} \Longrightarrow d\ b \in S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = c?a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-read-right* :
‹$c\ a \in S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b = d?b{\in}B \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ b)$›
⟨*proof*⟩

lemma *read-Sync$_{ptick}$-write-right* :
‹$c\ `\ A \subseteq S \Longrightarrow d\ b \notin S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)$›
⟨*proof*⟩

ndet-write **and** *ndet-write* **lemma** *ndet-write-Sync$_{ptick}$-ndet-write* :
‹$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
$(\ \textit{if}\ A = \{\}\ \textit{then}\ \ \textit{if}\ d\ `\ B \cap S = \{\}\ \textit{then}\ d!!b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$

else $(\sqcap x{\in}d\ `\ (B - d -`\ S) \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ x)))$
$\sqcap$ *STOP*
else *if* $B = \{\}$ *then* *if* $c\ `\ A \cap S = \{\}$ *then* $c!!a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
else $(\sqcap x{\in}c\ `\ (A - c -`\ S) \rightarrow (P\ (inv\text{-}into\ A\ c\ x)\ [\![S]\!]_{\checkmark}$
STOP)) $\sqcap$ *STOP*
else $\sqcap b{\in}d\ `\ B.\ \sqcap a{\in}c\ `\ A.$
(*if* $a \in S$ *then STOP else* $a \rightarrow (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (inv\text{-}into$
$B\ d\ b))) \Box$
(*if* $b \in S$ *then STOP else* $b \rightarrow (a \rightarrow P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into$
$B\ d\ b))) \Box$
(*if* $a = b \wedge b \in S$ *then* $b \rightarrow (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d$
$b))\ else\ STOP))\rangle$
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-ndet-write-subset* :
$\langle c\ `\ A \subseteq S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* $\exists b.\ c\ `\ A = \{b\} \wedge d\ `\ B = \{b\}$
then $(THE\ b.\ d\ `\ B = \{b\}) \rightarrow (P\ (inv\text{-}into\ A\ c\ (THE\ a.\ c\ `\ A = \{a\}))\ [\![S]\!]_{\checkmark}\ Q$
$(inv\text{-}into\ B\ d\ (THE\ b.\ d\ `\ B = \{b\})))$
else $(\sqcap x{\in}(c\ `\ A \cap d\ `\ B) \rightarrow (P\ (inv\text{-}into\ A\ c\ x)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ x))) \sqcap$
STOP)$\rangle$
⟨*proof*⟩

corollary *inj-on-ndet-write-Sync*$_{ptick}$*-ndet-write-subset* :
$\langle c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* $\exists b.\ c\ `\ A = \{b\} \wedge d\ `\ B = \{b\}$
then $d\ (THE\ b.\ B = \{b\}) \rightarrow (P\ (THE\ a.\ A = \{a\})\ [\![S]\!]_{\checkmark}\ Q\ (THE\ b.\ B = \{b\}))$
else $(\sqcap x{\in}(c\ `\ A \cap d\ `\ B) \rightarrow (P\ (inv\text{-}into\ A\ c\ x)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ x))) \sqcap$
STOP)$\rangle$
if $\langle inj\text{-}on\ c\ A\rangle\ \langle inj\text{-}on\ d\ B\rangle\ \langle c\ `\ A \subseteq S\rangle\ \langle d\ `\ B \subseteq S\rangle$
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-ndet-write-indep* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* $A = \{\}$ *then* $d!!b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
else *if* $B = \{\}$ *then* $c!!a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
else $\sqcap b{\in}d\ `\ B.\ \sqcap a{\in}c\ `\ A.$
$((a \rightarrow (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (inv\text{-}into\ B\ d\ b)))) \Box$
$((b \rightarrow (a \rightarrow P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ b)))))\rangle$
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-ndet-write-left* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$

$c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b = c!!a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b)\rangle$
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-ndet-write-right* :
$\langle c\ `\ A \subseteq S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b = d!!b{\in}B \to (c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)\rangle$
⟨*proof*⟩

ndet-write **and** *write* **lemma** *write-Sync$_{ptick}$-ndet-write* :
$\langle c!a \to P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b =$
(*if* $B = \{\}$ *then* $c!a \to P\ [\![S]\!]_{\checkmark}\ STOP$
else $\sqcap b{\in}d\ `\ B.$ (*if* $b \in S$ *then* $STOP$ *else* $b \to (c!a \to P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d\ b$))) $\Box$
(*if* $c\ a \in S$ *then* $STOP$ *else* $c!a \to (P\ [\![S]\!]_{\checkmark}\ b \to Q$ (*inv-into* $B\ d\ b$))) $\Box$
(*if* $b = c\ a \wedge c\ a \in S$ *then* $c!a \to (P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d$ ($c\ a$))) *else* $STOP$))⟩
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-write* :
$\langle c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d!b \to Q =$
(*if* $A = \{\}$ *then* $STOP\ [\![S]\!]_{\checkmark}\ d!b \to Q$
else $\sqcap a{\in}c\ `\ A.$ (*if* $a \in S$ *then* $STOP$ *else* $a \to (P$ (*inv-into* $A\ c\ a$) $[\![S]\!]_{\checkmark}\ d!b \to Q$)) $\Box$
(*if* $d\ b \in S$ *then* $STOP$ *else* $d!b \to (a \to P$ (*inv-into* $A\ c\ a$) $[\![S]\!]_{\checkmark}\ Q$)) $\Box$
(*if* $a = d\ b \wedge d\ b \in S$ *then* $d!b \to (P$ (*inv-into* $A\ c\ a$) $[\![S]\!]_{\checkmark}\ Q$) *else* $STOP$))⟩
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-ndet-write-subset* :
$\langle c!a \to P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b =$
(*if* $c\ a \notin d\ `\ B$ *then* $STOP$ *else if* $d\ `\ B = \{c\ a\}$ *then* $c!a \to (P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d$ ($c\ a$)))
else $(c!a \to (P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d$ ($c\ a$)))) $\sqcap\ STOP$)⟩ **if** $\langle c\ a \in S\rangle$ $\langle d\ `\ B \subseteq S\rangle$
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *ndet-write-Sync$_{ptick}$-write-subset* :
$\langle (c!!a{\in}A \to P\ a)\ [\![S]\!]_{\checkmark}\ (d!b \to Q) =$
(*if* $d\ b \notin c\ `\ A$ *then* $STOP$ *else if* $c\ `\ A = \{d\ b\}$ *then* $d!b \to (P$ (*inv-into* $A\ c$ ($d\ b$)) $[\![S]\!]_{\checkmark}\ Q$)
else $(d!b \to (P$ (*inv-into* $A\ c$ ($d\ b$)) $[\![S]\!]_{\checkmark}\ Q$)) $\sqcap\ STOP$)⟩ **if** $\langle c\ `\ A \subseteq S\rangle$ $\langle d\ b \in S\rangle$
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-ndet-write-indep* :
‹$c\ a \notin S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
$(\ \ if\ B = \{\}\ then\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ STOP)$
$\ \ else\ \sqcap b{\in}d\ `\ B.\ (c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (inv\text{-}into\ B\ d\ b))) \Box$
$(b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ b))))$›
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-write-indep* :
‹$c\ `\ A \cap S = \{\} \Longrightarrow d\ b \notin S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q =$
$(\ \ if\ A = \{\}\ then\ d!b \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q)$
$\ \ else\ \sqcap a{\in}c\ `\ A.\ (a \rightarrow (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)) \Box$
$(d!b \rightarrow (a \rightarrow P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q)))$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-ndet-write-left* :
‹$c\ a \notin S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b = c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b)$›
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-write-left* :
‹$c\ `\ A \cap S = \{\} \Longrightarrow d\ b \in S \Longrightarrow c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = c!!a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-ndet-write-right* :
‹$c\ a \in S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b = d!!b{\in}B \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ b)$›
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-write-right* :
‹$c\ `\ A \subseteq S \Longrightarrow d\ b \notin S \Longrightarrow c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)$›
⟨*proof*⟩

write **and** *write* **lemma** *write-Sync$_{ptick}$-write* :
‹$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q =$
$(if\ d\ b \in S\ then\ STOP\ else\ d!b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)) \Box$
$(if\ c\ a \in S\ then\ STOP\ else\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)) \Box$
$(if\ c\ a = d\ b \wedge d\ b \in S\ then\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q)\ else\ STOP)$›
⟨*proof*⟩

lemma *write-Inter$_{ptick}$-write* :
‹$c!a \rightarrow P\ |||_{\checkmark}\ d!b \rightarrow Q = (c!a \rightarrow (P\ |||_{\checkmark}\ d!b \rightarrow Q)) \Box (d!b \rightarrow (c!a \rightarrow P\ |||_{\checkmark}\ Q))$›
⟨*proof*⟩

lemma *write-Par*$_{ptick}$*-write* :
‹$c!a \rightarrow P \;||_{\checkmark}\; d!b \rightarrow Q = (\mathit{if}\ c\ a = d\ b\ \mathit{then}\ c!a \rightarrow (P \;||_{\checkmark}\; Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write-Sync*$_{ptick}$*-write-subset* :
‹$c\ a \in S \Longrightarrow d\ b \in S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = (\mathit{if}\ c\ a = d\ b\ \mathit{then}\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write-Sync*$_{ptick}$*-write-indep* :
‹$c\ a \notin S \Longrightarrow d\ b \notin S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = (c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)) \;\Box\; (d!b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q))$›
⟨*proof*⟩

lemma *write-Sync*$_{ptick}$*-write-left* :
‹$c\ a \notin S \Longrightarrow d\ b \in S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)$›
⟨*proof*⟩

lemma *write-Sync*$_{ptick}$*-write-right* :
‹$c\ a \in S \Longrightarrow d\ b \notin S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)$›
⟨*proof*⟩

read **and** $(\rightarrow)$**.** **lemma** *write0-Sync*$_{ptick}$*-read* :
‹$a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
$(\mathit{if}\ a \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b)) \;\Box$
$(\Box b{\in}(d\ `\ B - S) \rightarrow (a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ (\mathit{inv\text{-}into}\ B\ d\ b))) \;\Box$
$(\mathit{if}\ a \in d\ `\ B \cap S\ \mathit{then}\ a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ (\mathit{inv\text{-}into}\ B\ d\ a))\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *read-Sync*$_{ptick}$*-write0* :
‹$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
$(\mathit{if}\ b \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)) \;\Box$
$(\Box a{\in}(c\ `\ A - S) \rightarrow (P\ (\mathit{inv\text{-}into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ b \rightarrow Q)) \;\Box$
$(\mathit{if}\ b \in c\ `\ A \cap S\ \mathit{then}\ b \rightarrow (P\ (\mathit{inv\text{-}into}\ A\ c\ b)\ [\![S]\!]_{\checkmark}\ Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-read-subset* :
‹$a \in S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
$(\mathit{if}\ a \in d\ `\ B\ \mathit{then}\ a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ (\mathit{inv\text{-}into}\ B\ d\ a))\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *read-Sync*$_{ptick}$*-write0-subset* :
‹$c\ `\ A \subseteq S \Longrightarrow b \in S \Longrightarrow$

$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q =$
$(if\ b \in c\ `\ A\ then\ b \to (P\ (inv\text{-}into\ A\ c\ b)\ [\![S]\!]_{\checkmark}\ Q)\ else\ STOP)\rangle$
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-read-subset-same-chan*:
$\langle a \in S \Longrightarrow B \subseteq S \Longrightarrow$
$a \to P\ [\![S]\!]_{\checkmark}\ id?b{\in}B \to Q\ b = (if\ a \in B\ then\ a \to (P\ [\![S]\!]_{\checkmark}\ Q\ a)\ else\ STOP)\rangle$
⟨*proof*⟩

lemma *read-Sync*$_{ptick}$*-write0-subset-same-chan*:
$\langle A \subseteq S \Longrightarrow b \in S \Longrightarrow$
$id?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q = (if\ b \in A\ then\ b \to (P\ b\ [\![S]\!]_{\checkmark}\ Q)\ else\ STOP)\rangle$
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-read-indep* :
$\langle a \notin S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
$(a \to (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)) \Box (d?b{\in}B \to (a \to P\ [\![S]\!]_{\checkmark}\ Q\ b))\rangle$
⟨*proof*⟩

lemma *read-Sync*$_{ptick}$*-write0-indep* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow b \notin S \Longrightarrow$
$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q =$
$(b \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q)) \Box (c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ b \to Q))\rangle$
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-read-left* :
$\langle a \notin S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = a \to (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)\rangle$
⟨*proof*⟩

lemma *read-Sync*$_{ptick}$*-write0-left* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow b \in S \Longrightarrow c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q = c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ b \to Q)\rangle$
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-read-right* :
$\langle a \in S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = d?b{\in}B \to (a \to P\ [\![S]\!]_{\checkmark}\ Q\ b)\rangle$
⟨*proof*⟩

lemma *read-Sync*$_{ptick}$*-write0-right* :
$\langle c\ `\ A \subseteq S \Longrightarrow b \notin S \Longrightarrow c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q = b \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q)\rangle$
⟨*proof*⟩

ndet-write **and** $(\to)$ **lemma** *write0-Sync*$_{ptick}$*-ndet-write* :
$\langle a \to P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b =$
$(\ \ if\ B = \{\}\ then\ a \to P\ [\![S]\!]_{\checkmark}\ STOP$

else $\sqcap b{\in}d$ ‘ B. (*if* $b \in S$ *then* *STOP* *else* $b \rightarrow (a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ b))) \sqcap$
(*if* $a \in S$ *then* *STOP* *else* $a \rightarrow (P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (\textit{inv-into}\ B\ d\ b))) \sqcap$
(*if* $b = a \wedge a \in S$ *then* $a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ a))$ *else* *STOP*))›
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-write0* :
‹$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
(*if* $A = \{\}$ *then* *STOP* $[\![S]\!]_{\checkmark}\ b \rightarrow Q$
else $\sqcap a{\in}c$ ‘ A. (*if* $a \in S$ *then* *STOP* *else* $a \rightarrow (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ b \rightarrow Q)) \sqcap$
(*if* $b \in S$ *then* *STOP* *else* $b \rightarrow (a \rightarrow P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q)) \sqcap$
(*if* $a = b \wedge b \in S$ *then* $b \rightarrow (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q)$ *else* *STOP*))›
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-ndet-write-subset* :
‹$a \in S \Longrightarrow d$ ‘ $B \subseteq S \Longrightarrow$
$a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* $a \notin d$ ‘ B *then* *STOP* *else if* d ‘ $B = \{a\}$ *then* $a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ a))$
else $(a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ a))) \sqcap$ *STOP*)›
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-write0-subset* :
‹c ‘ $A \subseteq S \Longrightarrow b \in S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
(*if* $b \notin c$ ‘ A *then* *STOP* *else if* c ‘ $A = \{b\}$ *then* $b \rightarrow (P\ (\textit{inv-into}\ A\ c\ b)\ [\![S]\!]_{\checkmark}\ Q)$
else $(b \rightarrow (P\ (\textit{inv-into}\ A\ c\ b)\ [\![S]\!]_{\checkmark}\ Q)) \sqcap$ *STOP*)›
⟨*proof*⟩

lemma *write0-Sync*$_{ptick}$*-ndet-write-indep* :
‹$a \notin S \Longrightarrow d$ ‘ $B \cap S = \{\} \Longrightarrow$
$a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
(*if* $B = \{\}$ *then* $a \rightarrow (P\ [\![S]\!]_{\checkmark}\ \textit{STOP})$
else $\sqcap b{\in}d$ ‘ B. $(a \rightarrow (P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (\textit{inv-into}\ B\ d\ b))) \sqcap$
$(b \rightarrow (a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ b)))$)›
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-write0-indep* :
‹c ‘ $A \cap S = \{\} \Longrightarrow b \notin S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
(*if* $A = \{\}$ *then* $b \rightarrow (\textit{STOP}\ [\![S]\!]_{\checkmark}\ Q)$
else $\sqcap a{\in}c$ ‘ A. $(a \rightarrow (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ b \rightarrow Q)) \sqcap$
$(b \rightarrow (a \rightarrow P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q))$)›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-ndet-write-left* :
$\langle a \notin S \Longrightarrow d \mathbin{`} B \subseteq S \Longrightarrow a \rightarrow P \ [\![S]\!]_{\checkmark} \ d!!b{\in}B \rightarrow Q \ b = a \rightarrow (P \ [\![S]\!]_{\checkmark} \ d!!b{\in}B \rightarrow Q \ b)\rangle$
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-write0-left* :
$\langle c \mathbin{`} A \cap S = \{\} \Longrightarrow b \in S \Longrightarrow c!!a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ b \rightarrow Q = c!!a{\in}A \rightarrow (P \ a \ [\![S]\!]_{\checkmark} \ b \rightarrow Q)\rangle$
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-ndet-write0-right* :
$\langle a \in S \Longrightarrow d \mathbin{`} B \cap S = \{\} \Longrightarrow a \rightarrow P \ [\![S]\!]_{\checkmark} \ d!!b{\in}B \rightarrow Q \ b = d!!b{\in}B \rightarrow (a \rightarrow P \ [\![S]\!]_{\checkmark} \ Q \ b)\rangle$
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-write0-right* :
$\langle c \mathbin{`} A \subseteq S \Longrightarrow b \notin S \Longrightarrow c!!a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ b \rightarrow Q = b \rightarrow (c!!a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ Q)\rangle$
⟨*proof*⟩

$(\rightarrow)$ **and** $(\rightarrow)$ **lemma** *write0-Sync$_{ptick}$-write0* :
$\langle a \rightarrow P \ [\![S]\!]_{\checkmark} \ b \rightarrow Q =$
$(\mathit{if}\ b \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ b \rightarrow (a \rightarrow P \ [\![S]\!]_{\checkmark} \ Q)) \ \Box$
$(\mathit{if}\ a \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ a \rightarrow (P \ [\![S]\!]_{\checkmark} \ b \rightarrow Q)) \ \Box$
$(\mathit{if}\ a = b \wedge b \in S\ \mathit{then}\ a \rightarrow (P \ [\![S]\!]_{\checkmark} \ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write0-bis* :
$\langle (a \rightarrow P) \ [\![S]\!]_{\checkmark} \ (b \rightarrow Q) =$
$($ *if* $a \in S$
then *if* $b \in S$
then *if* $a = b$
then $a \rightarrow (P \ [\![S]\!]_{\checkmark} \ Q)$
else STOP
else $(b \rightarrow ((a \rightarrow P) \ [\![S]\!]_{\checkmark} \ Q))$
else *if* $b \in S$
then $a \rightarrow (P \ [\![S]\!]_{\checkmark} \ (b \rightarrow Q))$
else $(a \rightarrow (P \ [\![S]\!]_{\checkmark} \ (b \rightarrow Q))) \ \Box \ (b \rightarrow ((a \rightarrow P) \ [\![S]\!]_{\checkmark} \ Q)))\rangle$
⟨*proof*⟩

lemma *write0-Inter$_{ptick}$-write0* :
$\langle a \rightarrow P \ |||_{\checkmark} \ b \rightarrow Q = (a \rightarrow (P \ |||_{\checkmark} \ b \rightarrow Q)) \ \Box \ (b \rightarrow (a \rightarrow P \ |||_{\checkmark} \ Q))\rangle$
⟨*proof*⟩

lemma *write0-Par$_{ptick}$-write0* :
$\langle a \rightarrow P \ ||_{\checkmark} \ b \rightarrow Q = (\mathit{if}\ a = b\ \mathit{then}\ a \rightarrow (P \ ||_{\checkmark} \ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write0-subset* :
‹$a \in S \Longrightarrow b \in S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = (\mathit{if}\ a = b\ \mathit{then}\ a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write0-indep* :
‹$a \notin S \Longrightarrow b \notin S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = (a \rightarrow (P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q)) \; \Box \; (b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q))$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write0-left* :
‹$a \notin S \Longrightarrow b \in S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = a \rightarrow (P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q)$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write0-right* :
‹$a \in S \Longrightarrow b \notin S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q)$›
⟨*proof*⟩

write **and** $(\rightarrow)$ **lemma** *write0-Sync$_{ptick}$-write* :
‹$a \rightarrow P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q =$
$(\mathit{if}\ d\ b \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ d!b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q)) \; \Box$
$(\mathit{if}\ a \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ a \rightarrow (P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q)) \; \Box$
$(\mathit{if}\ a = d\ b \wedge d\ b \in S\ \mathit{then}\ a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-write0* :
‹$c!a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q =$
$(\mathit{if}\ b \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ b \rightarrow (c!a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q)) \; \Box$
$(\mathit{if}\ c\ a \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ c!a \rightarrow (P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q)) \; \Box$
$(\mathit{if}\ c\ a = b \wedge b \in S\ \mathit{then}\ c!a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write-subset* :
‹$a \in S \Longrightarrow d\ b \in S \Longrightarrow$
$a \rightarrow P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q = (\mathit{if}\ a = d\ b\ \mathit{then}\ a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-write0-subset* :
‹$c\ a \in S \Longrightarrow b \in S \Longrightarrow$
$c!a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = (\mathit{if}\ c\ a = b\ \mathit{then}\ c!a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q)\ \mathit{else}\ \mathit{STOP})$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write-indep* :
‹$a \notin S \Longrightarrow d\ b \notin S \Longrightarrow$
$a \rightarrow P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q = (a \rightarrow (P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q)) \; \Box \; (d!b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q))$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-write0-indep* :
‹$c\ a \notin S \Longrightarrow b \notin S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q = (c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q)) \Box (b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q))$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write-left* :
‹$a \notin S \Longrightarrow d\ b \in S \Longrightarrow a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-write0-left* :
‹$c\ a \notin S \Longrightarrow b \in S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q = c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q)$›
⟨*proof*⟩

lemma *write0-Sync$_{ptick}$-write-right* :
‹$a \in S \Longrightarrow d\ b \notin S \Longrightarrow a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)$›
⟨*proof*⟩

lemma *write-Sync$_{ptick}$-write0-right* :
‹$c\ a \in S \Longrightarrow b \notin S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q = b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)$›
⟨*proof*⟩

Synchronization with *SKIP* and *STOP*

SKIP Without injectivity, the result is a trivial corollary of *read c A P* ≡ *Mprefix* (*c* ‘ *A*) (*P* ∘ *inv-into A c*) and *Mprefix A P* $[\![S]\!]_{\checkmark}$ *SKIP r* = □*a*∈(*A* \ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*).

lemma *read-Sync$_{ptick}$-SKIP* :
‹$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ SKIP\ r = c?a{\in}(A - c -` S) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ SKIP\ r)$› **if** ‹*inj-on c A*›
⟨*proof*⟩

lemma *SKIP-Sync$_{ptick}$-read* :
‹$SKIP\ r\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b = d?b{\in}(B - d -` S) \rightarrow (SKIP\ r\ [\![S]\!]_{\checkmark}\ Q\ b)$› **if** ‹*inj-on d B*›
⟨*proof*⟩

corollary *write-Sync$_{ptick}$-SKIP* :
‹$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ SKIP\ s = (if\ c\ a \in S\ then\ STOP\ else\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ SKIP\ s))$›
and *SKIP-Sync$_{ptick}$-write* :
‹$SKIP\ r\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = (if\ d\ b \in S\ then\ STOP\ else\ d!b \rightarrow (SKIP\ r\ [\![S]\!]_{\checkmark}\ Q))$›
⟨*proof*⟩

corollary *write0-Sync$_{ptick}$-SKIP* :
‹$a \rightarrow P\ [\![S]\!]_{\checkmark}\ SKIP\ s = (if\ a \in S\ then\ STOP\ else\ a \rightarrow (P\ [\![S]\!]_{\checkmark}\ SKIP\ s))$›
and *SKIP-Sync$_{ptick}$-write0* :

‹*SKIP r* $[\![S]\!]_{\checkmark}$ *b* → *Q* = (*if b* ∈ *S then STOP else b* → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q*))›
⟨*proof*⟩

lemma *ndet-write-Sync*$_{ptick}$*-SKIP* :
‹*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *SKIP r* =
(*if c ‘ A* ∩ *S* = {} *then c*!!*a*∈*A* → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*)
else (*c*!!*a*∈(*A* − *c* −‘ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if - then ?rhs1 else ?rhs2* ⊓ *STOP*)›) **if** ‹*inj-on c A*›
⟨*proof*⟩

corollary (**in** *Sync*$_{ptick}$*-locale*) *SKIP-Sync*$_{ptick}$*-ndet-write* :
‹*inj-on d B* ⟹ *SKIP r* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* =
(*if d ‘ B* ∩ *S* = {} *then d*!!*b*∈*B* → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)
else (*d*!!*b*∈(*B* − *d* −‘ *S*) → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)) ⊓ *STOP*)›
⟨*proof*⟩

corollary (**in** *Sync*$_{ptick}$*-locale*) *Mndetprefix-Sync*$_{ptick}$*-SKIP* :
‹⊓*a* ∈ *A* → *P a* $[\![S]\!]_{\checkmark}$ *SKIP r* =
(*if A* ∩ *S* = {} *then* ⊓*a* ∈ *A* → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*)
else (⊓*a* ∈ (*A* − *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*)) ⊓ *STOP*)›
⟨*proof*⟩

corollary (**in** *Sync*$_{ptick}$*-locale*) *Sync*$_{ptick}$*-SKIP-Mndetprefix* :
‹*SKIP r* $[\![S]\!]_{\checkmark}$ ⊓*b* ∈ *B* → *Q b* =
(*if B* ∩ *S* = {} *then* ⊓*b* ∈ *B* → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)
else (⊓*b* ∈ (*B* − *S*) → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)) ⊓ *STOP*)›
⟨*proof*⟩

STOP Without injectivity, the result is a trivial corollary of *read c A P* ≡ *Mprefix* (*c ‘ A*) (*P* ∘ *inv-into A c*) and *Mprefix A P* $[\![S]\!]_{\checkmark}$ *SKIP r* = □*a*∈(*A* \ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*).

lemma *read-Sync*$_{ptick}$*-STOP* :
‹*c*?*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *STOP* = *c*?*a*∈(*A* − *c* −‘ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *STOP*)› **if** ‹*inj-on c A*›
⟨*proof*⟩

lemma *STOP-Sync*$_{ptick}$*-read* :
‹*STOP* $[\![S]\!]_{\checkmark}$ *d*?*b*∈*B* → *Q b* = *d*?*b*∈(*B* − *d* −‘ *S*) → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)› **if** ‹*inj-on d B*›
⟨*proof*⟩

corollary *write-Sync*$_{ptick}$*-STOP* :
‹*c*!*a* → *P* $[\![S]\!]_{\checkmark}$ *STOP* = (*if c a* ∈ *S then STOP else c*!*a* → (*P* $[\![S]\!]_{\checkmark}$ *STOP*))›

and *STOP-Sync$_{ptick}$-write* :
‹*STOP* $[\![S]\!]_{\checkmark}$ *d*!*b* → *Q* = (*if d b* ∈ *S then STOP else d*!*b* → (*STOP* $[\![S]\!]_{\checkmark}$ *Q*))›
⟨*proof*⟩

corollary *write0-Sync$_{ptick}$-STOP* :
‹*a* → *P* $[\![S]\!]_{\checkmark}$ *STOP* = (*if a* ∈ *S then STOP else a* → (*P* $[\![S]\!]_{\checkmark}$ *STOP*))›
and *STOP-Sync$_{ptick}$-write0* :
‹*STOP* $[\![S]\!]_{\checkmark}$ *b* → *Q* = (*if b* ∈ *S then STOP else b* → (*STOP* $[\![S]\!]_{\checkmark}$ *Q*))›
⟨*proof*⟩

lemma *ndet-write-Sync$_{ptick}$-STOP* :
‹*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *STOP* =
(*if c* ‘ *A* ∩ *S* = {} *then c*!!*a*∈*A* → (*P a* $[\![S]\!]_{\checkmark}$ *STOP*)
else (*c*!!*a*∈(*A* − *c* −‘ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *STOP*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if - then ?rhs1 else ?rhs2* ⊓ *STOP*)›) **if** ‹*inj-on c A*›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *STOP-Sync$_{ptick}$-ndet-write* :
‹*inj-on d B* ⟹ *STOP* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* =
(*if d* ‘ *B* ∩ *S* = {} *then d*!!*b*∈*B* → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)
else (*d*!!*b*∈(*B* − *d* −‘ *S*) → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)) ⊓ *STOP*)›
⟨*proof*⟩

end

Chapter 9

Operational Semantics Laws

9.1 Behaviour of *initials*

9.1.1 *TickSwap*

lemma *initials-TickSwap* :
‹(*TickSwap P*)^0 = (*if P* = ⊥ *then UNIV*
else {*ev a* |*a*. *ev a* ∈ P^0} ∪ {✓((*s*, *r*)) |*r s*. ✓((*r*, *s*)) ∈ P^0})›
⟨*proof*⟩

9.1.2 Sequential Composition

lemma *initials-Seq*$_{ptick}$:
‹(*P* ;✓ *Q*)^0 = (*if P* = ⊥ *then UNIV*
else {*ev a* |*a*. *ev a* ∈ P^0} ∪ (⋃ *r*∈{*r*. ✓(*r*) ∈ P^0}. (*Q r*)^0))›
(**is** ‹- = (*if* - *then* - *else ?rhs*)›)
⟨*proof*⟩

9.1.3 Synchronization Product

lemma (**in** *Sync*$_{ptick}$-*locale*) *initials-Sync*$_{ptick}$:
‹(*P* ⟦*S*⟧✓ *Q*)^0 =
(*if P* = ⊥ ∨ *Q* = ⊥ *then UNIV else*
{*ev a* |*a*. *a* ∈ *S* ∧ *ev a* ∈ P^0 ∧ *ev a* ∈ Q^0 ∨ *a* ∉ *S* ∧ (*ev a* ∈ P^0 ∨ *ev a* ∈ Q^0)}
∪
{✓(*r-s*) |*r-s r s*. *tick-join r s* = *Some r-s* ∧ ✓(*r*) ∈ P^0 ∧ ✓(*s*) ∈ Q^0})›
(**is** ‹(*P* ⟦*S*⟧✓ *Q*)^0 = (*if P* = ⊥ ∨ *Q* = ⊥ *then UNIV else ?rhs-ev* ∪ *?rhs-tick*)›)
⟨*proof*⟩

9.2 Laws of After

9.2.1 Sequential Composition

locale *AfterDuplicated-same-events* = *AfterDuplicated* Ψ$_\alpha$ Ψ$_\beta$
for Ψ$_\alpha$ :: ‹('*a*, '*r*) *process*$_{ptick}$ ⇒ '*a* ⇒ ('*a*, '*r*) *process*$_{ptick}$›
and Ψ$_\beta$ :: ‹('*a*, '*s*) *process*$_{ptick}$ ⇒ '*a* ⇒ ('*a*, '*s*) *process*$_{ptick}$›

begin

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

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

lemma *skippable-not-initialL-After-Seq*$_{ptick}$:
‹$(P \;;_{\checkmark}\; Q)\ after_\beta\ a = (\ \ if\ (\exists r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0)$
$then\ \sqcap r{\in}\{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.\ Q\ r\ after_\beta\ a$
$else\ \Psi_\beta\ (P \;;_{\checkmark}\; Q)\ a)$›
(**is** ‹$(P \;;_{\checkmark}\; Q)\ after_\beta\ a = (if\ ?prem\ then\ ?rhs\ else\ -)$›) **if** ‹$ev\ a \notin P^0$›
⟨*proof*⟩

lemma *skippable-initialL-initialR-After-Seq*$_{ptick}$:
‹$(P \;;_{\checkmark}\; Q)\ after_\beta\ a = (P\ after_\alpha\ a \;;_{\checkmark}\; Q) \sqcap (\sqcap r{\in}\{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.$
$Q\ r\ after_\beta\ a)$›
(**is** ‹$(P \;;_{\checkmark}\; Q)\ after_\beta\ a = (P\ after_\alpha\ a \;;_{\checkmark}\; Q) \sqcap\ ?rhs$›)
if *assms* : ‹$\exists r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0$› ‹$ev\ a \in P^0$›
⟨*proof*⟩

lemma *not-initialL-not-initialR-After-Seq*$_{ptick}$:
‹$ev\ a \notin P^0 \Longrightarrow (\bigwedge r.\ \checkmark(r) \in P^0 \Longrightarrow ev\ a \notin (Q\ r)^0) \Longrightarrow$
$(P \;;_{\checkmark}\; Q)\ after_\beta\ a = \Psi_\beta\ (P \;;_{\checkmark}\; Q)\ a$›
⟨*proof*⟩

lemma *After-Seq*$_{ptick}$:
‹$(P \;;_{\checkmark}\; Q)\ after_\beta\ a =$
$(\ \ if\ \forall r.\ \checkmark(r) \in P^0 \longrightarrow ev\ a \notin (Q\ r)^0$
$\ then\ \ \ if\ ev\ a \in P^0\ then\ P\ after_\alpha\ a \;;_{\checkmark}\; Q\ else\ \Psi_\beta\ (P \;;_{\checkmark}\; Q)\ a$
$\ else\ \ \ if\ ev\ a \in P^0$
$\quad then\ (P\ after_\alpha\ a \;;_{\checkmark}\; Q) \sqcap (\sqcap r{\in}\{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.\ Q\ r\ after_\beta\ a)$
$\quad else\ \sqcap r{\in}\{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.\ Q\ r\ after_\beta\ a)$›
⟨*proof*⟩

end

9.2.2 Synchronization Product

Because of the types, we have to extend the **locale**.

locale *After-Sync$_{ptick}$-locale* = *Sync$_{ptick}$-locale tick-join* +
  After$_{lhs}$: *After* Ψ_{lhs} + *After$_{rhs}$* : *After* Ψ_{rhs} + *After$_{ptick}$* : *After* Ψ_{ptick}
  for *tick-join* :: ‹$'r \Rightarrow 's \Rightarrow 't$ *option*›
    and Ψ_{lhs} :: ‹[($'a$, $'r$) *process$_{ptick}$*, $'a$] $\Rightarrow$ ($'a$, $'r$) *process$_{ptick}$*›
    and Ψ_{rhs} :: ‹[($'a$, $'s$) *process$_{ptick}$*, $'a$] $\Rightarrow$ ($'a$, $'s$) *process$_{ptick}$*›
    and Ψ_{ptick} :: ‹[($'a$, $'t$) *process$_{ptick}$*, $'a$] $\Rightarrow$ ($'a$, $'t$) *process$_{ptick}$*›
begin

notation *After$_{lhs}$.After* (**infixl** ‹*after$_{lhs}$*› *86*)
notation *After$_{rhs}$.After* (**infixl** ‹*after$_{rhs}$*› *86*)
notation *After$_{ptick}$.After* (**infixl** ‹*after$_{ptick}$*› *86*)

sublocale *After-Sync$_{ptick}$-locale-sym* :
  After-Sync$_{ptick}$-locale ‹$\lambda s\ r.$ *tick-join r s*› Ψ_{rhs} Ψ_{lhs} Ψ_{ptick}
  ⟨*proof*⟩

lemma *initialL-not-initialR-not-in-After-Sync$_{ptick}$*:
  ‹(P $[\![S]\!]_{\checkmark}$ Q) *after$_{ptick}$* a = P *after$_{lhs}$* a $[\![S]\!]_{\checkmark}$ Q› (**is** ‹*?lhs* = *?rhs*›)
  if *initial-hyps*: ‹*ev* $a \in P^0$› ‹*ev* $a \notin Q^0$› **and** *notin*: ‹$a \notin S$›
⟨*proof*⟩

lemma (**in** *After-Sync$_{ptick}$-locale*) *not-initialL-initialR-not-in-After-Sync$_{ptick}$*:
  ‹(P $[\![S]\!]_{\checkmark}$ Q) *after$_{ptick}$* a = P $[\![S]\!]_{\checkmark}$ Q *after$_{rhs}$* a› (**is** ‹*?lhs* = *?rhs*›)
  if *initial-hyps*: ‹*ev* $a \notin P^0$› ‹*ev* $a \in Q^0$› **and** *notin*: ‹$a \notin S$›
  ⟨*proof*⟩

lemma *not-initialL-in-After-Sync$_{ptick}$*:
  ‹*ev* $a \notin P^0 \Longrightarrow a \in S \Longrightarrow$
   (P $[\![S]\!]_{\checkmark}$ Q) *after$_{ptick}$* a = (*if* $Q = \bot$ *then* $\bot$ *else* Ψ_{ptick} (P $[\![S]\!]_{\checkmark}$ Q) a)›
  ⟨*proof*⟩

lemma *not-initialR-in-After-Sync$_{ptick}$*:
  ‹*ev* $a \notin Q^0 \Longrightarrow a \in S \Longrightarrow$
   (P $[\![S]\!]_{\checkmark}$ Q) *after$_{ptick}$* a = (*if* $P = \bot$ *then* $\bot$ *else* Ψ_{ptick} (P $[\![S]\!]_{\checkmark}$ Q) a)›
  ⟨*proof*⟩

lemma *initialL-initialR-in-After-Sync$_{ptick}$*:
  ‹(P $[\![S]\!]_{\checkmark}$ Q) *after$_{ptick}$* a = P *after$_{lhs}$* a $[\![S]\!]_{\checkmark}$ Q *after$_{rhs}$* a› (**is** ‹*?lhs* = *?rhs*›)
  if *initial-hyps*: ‹*ev* $a \in P^0$› ‹*ev* $a \in Q^0$› **and** *inside*: ‹$a \in S$›
⟨*proof*⟩

lemma *initialL-initialR-not-in-After-Sync*$_{ptick}$:
‹(*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *after*$_{ptick}$ *a* = (*P after*$_{lhs}$ *a* ⟦*S*⟧$_{\checkmark}$ *Q*) ⊓ (*P* ⟦*S*⟧$_{\checkmark}$ *Q after*$_{rhs}$ *a*)›
(**is** ‹*?lhs* = *?rhs1* ⊓ *?rhs2*›)
if *initial-hyps*: ‹*ev a* ∈ *P*^{0}› ‹*ev a* ∈ *Q*^{0}› **and** *notin*: ‹*a* ∉ *S*›
⟨*proof*⟩

lemma *not-initialL-not-initialR-After-Sync*$_{ptick}$:
‹*ev a* ∉ *P*^{0} ⟹ *ev a* ∉ *Q*^{0} ⟹ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *after*$_{ptick}$ *a* = Ψ$_{ptick}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *a*›
⟨*proof*⟩

Finally, the monster theorem !

theorem *After-Sync*$_{ptick}$:
‹(*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *after*$_{ptick}$ *a* =
(*if P* = ⊥ ∨ *Q* = ⊥ *then* ⊥
else *if ev a* ∈ *P*^{0} ∧ *ev a* ∈ *Q*^{0}
then *if a* ∈ *S then P after*$_{lhs}$ *a* ⟦*S*⟧$_{\checkmark}$ *Q after*$_{rhs}$ *a*
else (*P after*$_{lhs}$ *a* ⟦*S*⟧$_{\checkmark}$ *Q*) ⊓ (*P* ⟦*S*⟧$_{\checkmark}$ *Q after*$_{rhs}$ *a*)
else *if ev a* ∈ *P*^{0} ∧ *a* ∉ *S then P after*$_{lhs}$ *a* ⟦*S*⟧$_{\checkmark}$ *Q*
else *if ev a* ∈ *Q*^{0} ∧ *a* ∉ *S then P* ⟦*S*⟧$_{\checkmark}$ *Q after*$_{rhs}$ *a*
else Ψ$_{ptick}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *a*)›
⟨*proof*⟩

end

9.3 Small Steps Transitions

9.3.1 Extension of the After Operator

9.3.2 Sequential Composition

locale *AfterExtDuplicated-same-events* = *AfterExtDuplicated* Ψ$_\alpha$ Ω$_\alpha$ Ψ$_\beta$ Ω$_\beta$
for Ψ$_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
and Ω$_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*r*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
and Ψ$_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
and Ω$_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*s*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›

sublocale *AfterExtDuplicated-same-events* ⊆ *AfterDuplicated-same-events* ⟨*proof*⟩

context *AfterExtDuplicated-same-events*
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** ‹$after_{\checkmark\alpha}$› *86*)
notation $After_{tick\beta}.After_{tick}$ (**infixl** ‹$after_{\checkmark\beta}$› *86*)

lemma $After_{tick}$-Seq_{ptick} :
‹$(P \mathbin{;_{\checkmark}} Q)\ after_{\checkmark\beta}\ e =$
(*case e of* $\checkmark(r) \Rightarrow \Omega_{\beta}\ (P \mathbin{;_{\checkmark}} Q)\ r$
| *ev a* ⇒
if $\forall r.\ \checkmark(r) \in P^0 \longrightarrow ev\ a \notin (Q\ r)^0$
then *if ev a* $\in P^0$
then P $after_{\checkmark\alpha}$ *ev a* $;_{\checkmark}$ *Q else* $\Psi_{\beta}\ (P \mathbin{;_{\checkmark}} Q)\ a$
else *if ev a* $\in P^0$
then $(P\ after_{\checkmark\alpha}\ ev\ a \mathbin{;_{\checkmark}} Q) \sqcap$
$(\sqcap r \in \{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.\ Q\ r\ after_{\checkmark\beta}\ ev\ a)$
else $\sqcap r \in \{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.\ Q\ r\ after_{\checkmark\beta}\ ev\ a$)›
⟨*proof*⟩

end

Synchronization Product

locale $AfterExt$-$Sync_{ptick}$-*locale* = $Sync_{ptick}$-*locale tick-join* +
$AfterExt_{lhs}$: *AfterExt* $\Psi_{lhs}\ \Omega_{lhs}$ +
$AfterExt_{rhs}$: *AfterExt* $\Psi_{rhs}\ \Omega_{rhs}$ +
$AfterExt_{ptick}$: *AfterExt* $\Psi_{ptick}\ \Omega_{ptick}$
for *tick-join* :: ‹$'r \Rightarrow 's \Rightarrow 't$ *option*›
and Ψ_{lhs} :: ‹$[('a, 'r)\ process_{ptick}, 'a] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ω_{lhs} :: ‹$[('a, 'r)\ process_{ptick}, 'r] \Rightarrow ('a, 'r)\ process_{ptick}$›
and Ψ_{rhs} :: ‹$[('a, 's)\ process_{ptick}, 'a] \Rightarrow ('a, 's)\ process_{ptick}$›
and Ω_{rhs} :: ‹$[('a, 's)\ process_{ptick}, 's] \Rightarrow ('a, 's)\ process_{ptick}$›
and Ψ_{ptick} :: ‹$[('a, 't)\ process_{ptick}, 'a] \Rightarrow ('a, 't)\ process_{ptick}$›
and Ω_{ptick} :: ‹$[('a, 't)\ process_{ptick}, 't] \Rightarrow ('a, 't)\ process_{ptick}$›
begin

sublocale $After$-$Sync_{ptick}$-*locale tick-join* $\Psi_{lhs}\ \Psi_{rhs}\ \Psi_{ptick}$ ⟨*proof*⟩

sublocale $AfterExt$-$Sync_{ptick}$-*locale-sym* :
$AfterExt$-$Sync_{ptick}$-*locale* ‹$\lambda s\ r.$ *tick-join r s*› $\Psi_{rhs}\ \Omega_{rhs}\ \Psi_{lhs}\ \Omega_{lhs}\ \Psi_{ptick}\ \Omega_{ptick}$
⟨*proof*⟩

notation $AfterExt_{lhs}.After_{tick}$ (**infixl** ‹$after_{\checkmark lhs}$› *86*)
notation $AfterExt_{rhs}.After_{tick}$ (**infixl** ‹$after_{\checkmark rhs}$› *86*)

notation $AfterExt_{ptick}.After_{tick}$ (**infixl** ‹$after\checkmark_{ptick}$› *86*)

theorem $After_{tick}$-$Sync_{ptick}$:
‹$(P\ [\![S]\!]\checkmark\ Q)\ after\checkmark_{ptick}\ e =$
(*case e of* ✓(*r-s*) ⇒ $\Omega_{ptick}\ (P\ [\![S]\!]\checkmark\ Q)$ *r-s*
| *ev a* ⇒
if $P = \bot \vee Q = \bot$ *then* $\bot$
else if ev a $\in P^0 \wedge$ *ev a* $\in Q^0$
then if $a \in S$ *then* $P\ after\checkmark_{lhs}$ *ev a* $[\![S]\!]\checkmark\ Q\ after\checkmark_{rhs}$ *ev a*
else $(P\ after\checkmark_{lhs}$ *ev a* $[\![S]\!]\checkmark\ Q) \sqcap (P\ [\![S]\!]\checkmark\ Q\ after\checkmark_{rhs}$ *ev a*)
else if ev a $\in P^0 \wedge a \notin S$ *then* $P\ after\checkmark_{lhs}$ *ev a* $[\![S]\!]\checkmark\ Q$
else if ev a $\in Q^0 \wedge a \notin S$ *then* $P\ [\![S]\!]\checkmark\ Q\ after\checkmark_{rhs}$ *ev a*
else $\Psi_{ptick}\ (P\ [\![S]\!]\checkmark\ Q)\ a$)›
⟨*proof*⟩

end

9.3.3 Generic Operational Semantics as Locales

Sequential Composition

locale *OpSemTransitionsDuplicated-same-events* =
OpSemTransitionsDuplicated $\Psi_\alpha\ \Omega_\alpha\ \tau$-$trans_\alpha\ \Psi_\beta\ \Omega_\beta\ \tau$-$trans_\beta$
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 τ-$trans_\alpha$ :: ‹$[('a, 'r)\ process_{ptick}, ('a, 'r)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹$\alpha\rightsquigarrow\tau$› *50*)
and Ψ_β :: ‹$[('a, 's)\ process_{ptick}, 'a] \Rightarrow ('a, 's)\ process_{ptick}$›
and Ω_β :: ‹$[('a, 's)\ process_{ptick}, 's] \Rightarrow ('a, 's)\ process_{ptick}$›
and τ-$trans_\beta$ :: ‹$[('a, 's)\ process_{ptick}, ('a, 's)\ process_{ptick}] \Rightarrow bool$› (**infixl** ‹$\beta\rightsquigarrow\tau$› *50*)

sublocale *OpSemTransitionsDuplicated-same-events* ⊆ *AfterExtDuplicated-same-events*
⟨*proof*⟩

context *OpSemTransitionsDuplicated-same-events* **begin**

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

lemma τ-$trans\text{-}Seq_{ptick}R$: ‹$P\ ;\checkmark\ Q\ _\beta\rightsquigarrow_\tau\ Q'$› **if** ‹$P\ _\alpha\rightsquigarrow\checkmark_r\ P'$› **and** ‹$Q\ r\ _\beta\rightsquigarrow_\tau\ Q'$›
⟨*proof*⟩

lemma ‹✓$(r) \in P^0 \Longrightarrow Q\ r\ _\beta\rightsquigarrow_e\ Q' \Longrightarrow P\ ;\checkmark\ Q\ _\beta\rightsquigarrow_e\ Q'$› **for** P :: ‹$('a, 'r)\ process_{ptick}$›
⟨*proof*⟩

end

locale *OpSemTransitionsSeq*$_{ptick}$ =
OpSemTransitionsDuplicated-same-events 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 :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
and Ω_β :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*s*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
and τ-*trans*$_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, ('*a*, '*s*) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\beta\!\rightsquigarrow_\tau$› *50*) +
assumes τ-*trans-Seq*$_{ptick}L$: ‹*P* $_\alpha\!\rightsquigarrow_\tau$ *P'* ⟹ *P* ;$_\checkmark$ *Q* $_\beta\!\rightsquigarrow_\tau$ *P'* ;$_\checkmark$ *Q*›
begin

lemma *ev-trans-Seq*$_{ptick}L$: ‹*P* $_\alpha\!\rightsquigarrow_e$ *P'* ⟹ *P* ;$_\checkmark$ *Q* $_\beta\!\rightsquigarrow_e$ *P'* ;$_\checkmark$ *Q*›
⟨*proof*⟩

lemmas *Seq*$_{ptick}$-*OpSem-rules* = τ-*trans-Seq*$_{ptick}L$ *ev-trans-Seq*$_{ptick}L$ τ-*trans-Seq*$_{ptick}R$

end

Synchronization Product

locale *OpSemTransitions-Sync*$_{ptick}$-*locale* = *Sync*$_{ptick}$-*locale* ‹(⊗✓)› +
OpSemTransitions$_{lhs}$: *OpSemTransitions* Ψ_{lhs} Ω_{lhs} ‹($_{lhs}\!\rightsquigarrow_\tau$)› +
OpSemTransitions$_{rhs}$: *OpSemTransitions* Ψ_{rhs} Ω_{rhs} ‹($_{rhs}\!\rightsquigarrow_\tau$)› +
OpSemTransitions$_{ptick}$: *OpSemTransitions* Ψ_{ptick} Ω_{ptick} ‹($_{ptick}\!\rightsquigarrow_\tau$)›
for *tick-join* :: ‹'*r* ⇒ '*s* ⇒ '*t option*› (**infixl** ‹⊗✓› *100*)
and Ψ_{lhs} :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
and Ω_{lhs} :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*r*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
and τ-*trans*$_{lhs}$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, ('*a*, '*r*) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_{lhs}\!\rightsquigarrow_\tau$› *50*)
and Ψ_{rhs} :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
and Ω_{rhs} :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*s*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
and τ-*trans*$_{rhs}$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, ('*a*, '*s*) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_{rhs}\!\rightsquigarrow_\tau$› *50*)
and Ψ_{ptick} :: ‹[('*a*, '*t*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*t*) *process*$_{ptick}$›
and Ω_{ptick} :: ‹[('*a*, '*t*) *process*$_{ptick}$, '*t*] ⇒ ('*a*, '*t*) *process*$_{ptick}$›
and τ-*trans*$_{ptick}$ :: ‹[('*a*, '*t*) *process*$_{ptick}$, ('*a*, '*t*) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_{ptick}\!\rightsquigarrow_\tau$› *50*) +
assumes τ-*trans-Sync*$_{ptick}L$: ‹*P* $_{lhs}\!\rightsquigarrow_\tau$ *P'* ⟹ *P* $[\![A]\!]_\checkmark$ *Q* $_{ptick}\!\rightsquigarrow_\tau$ *P'* $[\![A]\!]_\checkmark$ *Q*›
and τ-*trans-Sync*$_{ptick}R$: ‹*Q* $_{rhs}\!\rightsquigarrow_\tau$ *Q'* ⟹ *P* $[\![A]\!]_\checkmark$ *Q* $_{ptick}\!\rightsquigarrow_\tau$ *P* $[\![A]\!]_\checkmark$ *Q'*›
begin

sublocale *AfterExt-Sync$_{ptick}$-locale* ⟨*proof*⟩

sublocale *OpSemTransitions-Sync$_{ptick}$-locale-sym* :
OpSemTransitions-Sync$_{ptick}$-locale
‹λ*s r*. *r* ⊗✓ *s*› Ψ$_{rhs}$ Ω$_{rhs}$ ‹(*$_{rhs}$*⇝τ)› Ψ$_{lhs}$ Ω$_{lhs}$ ‹(*$_{lhs}$*⇝τ)› Ψ$_{ptick}$ Ω$_{ptick}$ ‹(*$_{ptick}$*⇝τ)›
⟨*proof*⟩

notation *OpSemTransitions$_{lhs}$.ev-trans* (‹- *$_{lhs}$*⇝- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitions$_{lhs}$.tick-trans* (‹- *$_{lhs}$*⇝✓- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitions$_{rhs}$.ev-trans* (‹- *$_{rhs}$*⇝- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitions$_{rhs}$.tick-trans* (‹- *$_{rhs}$*⇝✓- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitions$_{ptick}$.ev-trans* (‹- *$_{ptick}$*⇝- -› [*50*, *3*, *51*] *50*)
notation *OpSemTransitions$_{ptick}$.tick-trans* (‹- *$_{ptick}$*⇝✓- -› [*50*, *3*, *51*] *50*)

We do not need the assumptions τ-*trans-Sync$_{ptick}$L* τ-*trans-Sync$_{ptick}$R* for the three following lemmas.

lemma τ-*trans-SKIP-Sync$_{ptick}$L* :
‹*P* ⟦*A*⟧✓ *Q* *$_{ptick}$*⇝τ *SKIP r* ⟦*A*⟧✓ *Q*› **if** ‹*P* *$_{lhs}$*⇝✓*r P'*›
⟨*proof*⟩

lemma τ-*trans-SKIP-Sync$_{ptick}$R* :
‹*P* ⟦*A*⟧✓ *Q* *$_{ptick}$*⇝τ *P* ⟦*A*⟧✓ *SKIP s*› **if** ‹*Q* *$_{rhs}$*⇝✓*s Q'*›
⟨*proof*⟩

lemma *tick-trans-SKIP-Sync$_{ptick}$-SKIP*:
‹*r* ⊗✓ *s* = *Some r-s* ⟹ *SKIP r* ⟦*A*⟧✓ *SKIP s* *$_{ptick}$*⇝✓*r-s* Ω$_{ptick}$ (*SKIP r-s*) *r-s*›
⟨*proof*⟩

lemma *ev-trans-Sync$_{ptick}$L* :
‹*a* ∉ *A* ⟹ *P* *$_{lhs}$*⇝*a P'* ⟹ *P* ⟦*A*⟧✓ *Q* *$_{ptick}$*⇝*a P'* ⟦*A*⟧✓ *Q*›
⟨*proof*⟩

lemma *ev-trans-Sync$_{ptick}$R* :
‹*a* ∉ *A* ⟹ *Q* *$_{rhs}$*⇝*a Q'* ⟹ *P* ⟦*A*⟧✓ *Q* *$_{ptick}$*⇝*a P* ⟦*A*⟧✓ *Q'*›
⟨*proof*⟩

lemma *ev-trans-Sync$_{ptick}$LR* :
‹*a* ∈ *A* ⟹ *P* *$_{lhs}$*⇝*a P'* ⟹ *Q* *$_{rhs}$*⇝*a Q'* ⟹ *P* ⟦*A*⟧✓ *Q* *$_{ptick}$*⇝*a P'* ⟦*A*⟧✓ *Q'*›
⟨*proof*⟩

lemmas *Sync$_{ptick}$-OpSem-rules* = τ-*trans-Sync$_{ptick}$L* τ-*trans-Sync$_{ptick}$R*
ev-trans-Sync$_{ptick}$L ev-trans-Sync$_{ptick}$R

ev-trans-Sync$_{ptick}$*LR*
τ*-trans-SKIP-Sync*$_{ptick}$*L* τ*-trans-SKIP-Sync*$_{ptick}$*R*
tick-trans-SKIP-Sync$_{ptick}$*-SKIP*

end

Chapter 10

Declensions of the Generalized Synchronization Product

unbundle *option-type-syntax*

10.1 Interpretations

For practical reasons, we directly interpret $Sync_{ptick}$-*comm-locale*. Then, the laws of associativity will be derived manually (instead of globally interpreting the locale $Sync_{ptick}$-*assoc-locale*).

10.1.1 Classical Version

The following interpretation is initially the reason we wanted the parameter ($\otimes\checkmark$) to be of type $'r \Rightarrow 's \Rightarrow 't$ *option* instead of just $'r \Rightarrow 's \Rightarrow 't$ (we wanted the operator *Sync* already defined in `HOL-CSP` to indeed be a particular case of the new one).

interpretation $Sync_{Classic}$: $Sync_{ptick}$-*comm-locale*
  ‹$\lambda r\ s.$ *if* $r = s$ *then* $\lfloor r \rfloor$ *else* $\Diamond$›
  ‹$\lambda s\ r.$ *if* $s = r$ *then* $\lfloor s \rfloor$ *else* $\Diamond$› *id id*
  ⟨*proof*⟩

notation $Sync_{Classic}.Sync_{ptick}$ (‹(- $[\![$-$]\!]_{\checkmark Classic}$ -)› [70, 0, 71] 70)
notation $Sync_{Classic}.Inter_{ptick}$ (‹(- $|||_{\checkmark Classic}$ -)› [72, 73] 72)
notation $Sync_{Classic}.Par_{ptick}$ (‹(- $||_{\checkmark Classic}$ -)› [74, 75] 74)

10.1.2 Product Type

interpretation $Sync_{Pair}$: $Sync_{ptick}$-*comm-locale*

‹λr s. ⌊(r, s)⌋› ‹λs r. ⌊(s, r)⌋› *prod.swap prod.swap*
⟨*proof*⟩

notation $Sync_{Pair}.Sync_{ptick}$ (‹(- $[\![$-$]\!]_{\checkmark Pair}$ -)› [70, 0, 71] 70)
notation $Sync_{Pair}.Inter_{ptick}$ (‹(- $|||_{\checkmark Pair}$ -)› [72, 73] 72)
notation $Sync_{Pair}.Par_{ptick}$ (‹(- $||_{\checkmark Pair}$ -)› [74, 75] 74)

10.1.3 List Type

Pair

interpretation $Sync_{Pairlist}$: $Sync_{ptick}$-comm-locale
‹λr s. ⌊[r, s]⌋› ‹λs r. ⌊[s, r]⌋›
‹λrs. [rs ! Suc 0, rs ! 0]› ‹λrs. [rs ! Suc 0, rs ! 0]›
⟨*proof*⟩

notation $Sync_{Pairlist}.Sync_{ptick}$ (‹(- $[\![$-$]\!]_{\checkmark Pairlist}$ -)› [70, 0, 71] 70)
notation $Sync_{Pairlist}.Inter_{ptick}$ (‹(- $|||_{\checkmark Pairlist}$ -)› [72, 73] 72)
notation $Sync_{Pairlist}.Par_{ptick}$ (‹(- $||_{\checkmark Pairlist}$ -)› [74, 75] 74)

Right List

Here, we want to have one process of type ('a, 'r) $process_{ptick}$ on the left hand side, and one of type ('a, 'r *list*) $process_{ptick}$ on the right hand side.

interpretation $Sync_{Rlist}$: $Sync_{ptick}$-comm-locale
‹λr s. ⌊r # s⌋› ‹λs r. ⌊s @ [r]⌋›
‹*rotate1*› ‹λrs. *if rs* = [] *then* [] *else last rs # butlast rs*›
— λrs. *last rs # butlast rs* is not injective.
⟨*proof*⟩

notation $Sync_{Rlist}.Sync_{ptick}$ (‹(- $[\![$-$]\!]_{\checkmark Rlist}$ -)› [70, 0, 71] 70)
notation $Sync_{Rlist}.Inter_{ptick}$ (‹(- $|||_{\checkmark Rlist}$ -)› [72, 73] 72)
notation $Sync_{Rlist}.Par_{ptick}$ (‹(- $||_{\checkmark Rlist}$ -)› [74, 75] 74)

Left List

Here, we want to have one process of type ('a, 'r *list*) $process_{ptick}$ on the left hand side, and one of type ('a, 'r) $process_{ptick}$ on the right hand side. There is no need to do a new interpretation, the operator we are looking for is actually the symmetric of the one we defined just above.

notation $Sync_{Rlist}.Sync_{ptick}$-comm-locale-sym.$Sync_{ptick}$ (‹(- $[\![$-$]\!]_{\checkmark Llist}$ -)› [70, 0, 71] 70)
notation $Sync_{Rlist}.Sync_{ptick}$-comm-locale-sym.$Inter_{ptick}$ (‹(- $|||_{\checkmark Llist}$ -)› [72, 73] 72)
notation $Sync_{Rlist}.Sync_{ptick}$-comm-locale-sym.$Par_{ptick}$ (‹(- $||_{\checkmark Llist}$ -)› [74, 75] 74)

Arbitrary Lists

We believed for a long time that it was not possible to handle the case where both processes have their ticks of type *'r list*. Indeed the concatenation on the lists is not injective, resulting in the impossibility of interpreting *Sync$_{ptick}$-locale*. But it turns out that by adding some control on the length of the lists, we actually can!

Control on one side **context fixes** *lenL* :: *nat* **begin**

global-interpretation *Sync$_{ListslenL}$* : *Sync$_{ptick}$-comm-locale*
‹*λr s. if length r = lenL then ⌊r @ s⌋ else ◊*›
‹*λs r. if length r = lenL then ⌊s @ r⌋ else ◊*›
‹*λrs. drop lenL rs @ take lenL rs*›
‹*λrs. rev (take lenL (rev rs)) @ rev (drop lenL (rev rs))*›
⟨*proof*⟩

end

abbreviation *Sync$_{ListslenL}$-syntax* ::
‹[('*a*, '*r list*) *process$_{ptick}$*, *nat*, '*a set*, ('*a*, '*r list*) *process$_{ptick}$*]
⇒ ('*a*, '*r list*) *process$_{ptick}$*› (‹(- -([-]$_{✓ListslenL}$) -)› [*70*, *0*, *0*, *71*] *70*)
where ‹*P* $_{lenL}[\![A]\!]_{✓ListslenL}$ *Q* ≡ *Sync$_{ListslenL}$.Sync$_{ptick}$ lenL P A Q*›

abbreviation *Inter$_{ListslenL}$-syntax* ::
‹[('*a*, '*r list*) *process$_{ptick}$*, *nat*, ('*a*, '*r list*) *process$_{ptick}$*]
⇒ ('*a*, '*r list*) *process$_{ptick}$*› (‹(- -(|||$_{✓ListslenL}$) -)› [*72*, *0*, *73*] *72*)
where ‹*P* $_{lenL}|||_{✓ListslenL}$ *Q* ≡ *Sync$_{ListslenL}$.Inter$_{ptick}$ lenL P Q*›

abbreviation *Par$_{ListslenL}$-syntax* ::
‹[('*a*, '*r list*) *process$_{ptick}$*, *nat*, ('*a*, '*r list*) *process$_{ptick}$*]
⇒ ('*a*, '*r list*) *process$_{ptick}$*› (‹(- -(||$_{✓ListslenL}$) -)› [*74*, *0*, *75*] *75*)
where ‹*P* $_{lenL}||_{✓ListslenL}$ *Q* ≡ *Sync$_{ListslenL}$.Par$_{ptick}$ lenL P Q*›

The control is done on the left process, so with the symmetric version of this operator we control the ticks length of the right one.

abbreviation *Sync$_{ListslenR}$-syntax* ::
‹[('*a*, '*r list*) *process$_{ptick}$*, *nat*, '*a set*, ('*a*, '*r list*) *process$_{ptick}$*]
⇒ ('*a*, '*r list*) *process$_{ptick}$*› (‹(- -([-]$_{✓ListslenR}$) -)› [*70*, *0*, *0*, *71*] *70*)
where ‹*P* $_{lenL}[\![A]\!]_{✓ListslenR}$ *Q* ≡ *Sync$_{ListslenL}$.Sync$_{ptick}$-comm-locale-sym.Sync$_{ptick}$ lenL P A Q›

abbreviation *Inter$_{ListslenR}$-syntax* ::
‹[('*a*, '*r list*) *process$_{ptick}$*, *nat*, ('*a*, '*r list*) *process$_{ptick}$*]
⇒ ('*a*, '*r list*) *process$_{ptick}$*› (‹(- -(|||$_{✓ListslenR}$) -)› [*72*, *0*, *73*] *72*)
where ‹*P* $_{lenL}|||_{✓ListslenR}$ *Q* ≡ *Sync$_{ListslenL}$.Sync$_{ptick}$-comm-locale-sym.Inter$_{ptick}$ lenL P Q›

abbreviation $Par_{ListslenR}$-*syntax* ::
‹[('a, 'r list) $process_{ptick}$, *nat*, ('a, 'r list) $process_{ptick}$]
⇒ ('a, 'r list) $process_{ptick}$› (‹(- -($||_{\checkmark ListslenR}$) -)› [74, 0, 75] 75)
where ‹*P* $_{lenL}||_{\checkmark ListslenR}$ *Q* ≡ $Sync_{ListslenL}$.$Sync_{ptick}$-*locale-sym*.Par_{ptick} *lenL P Q*›

Control on both sides **context fixes** *lenL* :: *nat* **and** *lenR* :: *nat* **begin**

global-interpretation $Sync_{Lists}$: $Sync_{ptick}$-*comm-locale*
‹λ*r s*. *if length r* = *lenL* ∧ *length s* = *lenR then* ⌊*r* @ *s*⌋ *else* ◊›
‹λ*s r*. *if length s* = *lenR* ∧ *length r* = *lenL then* ⌊*s* @ *r*⌋ *else* ◊›
‹λ*rs*. *drop lenL rs* @ *take lenL rs*›
‹λ*rs*. *drop lenR rs* @ *take lenR rs*›
⟨*proof*⟩

end

abbreviation $Sync_{Lists}$-*syntax* ::
‹[('a, 'r list) $process_{ptick}$, *nat*, 'a *set*, *nat*, ('a, 'r list) $process_{ptick}$]
⇒ ('a, 'r list) $process_{ptick}$› (‹(- -($[\![$-$]\!]_{\checkmark}$)- -)› [70, 0, 0, 0, 71] 70)
where ‹*P* $_{lenL}[\![A]\!]_{\checkmark lenR}$ *Q* ≡ $Sync_{Lists}$.$Sync_{ptick}$ *lenL lenR P A Q*›

abbreviation $Inter_{Lists}$-*syntax* ::
‹[('a, 'r list) $process_{ptick}$, *nat*, *nat*, ('a, 'r list) $process_{ptick}$]
⇒ ('a, 'r list) $process_{ptick}$› (‹(- -($|||_{\checkmark}$)- -)› [72, 0, 0, 73] 72)
where ‹*P* $_{lenL}|||_{\checkmark lenR}$ *Q* ≡ $Sync_{Lists}$.$Inter_{ptick}$ *lenL lenR P Q*›

abbreviation Par_{Lists}-*syntax* ::
‹[('a, 'r list) $process_{ptick}$, *nat*, *nat*, ('a, 'r list) $process_{ptick}$]
⇒ ('a, 'r list) $process_{ptick}$› (‹(- -($||_{\checkmark}$)- -)› [74, 0, 0, 75] 75)
where ‹*P* $_{lenL}||_{\checkmark lenR}$ *Q* ≡ $Sync_{Lists}$.Par_{ptick} *lenL lenR P Q*›

10.2 Associativities

10.2.1 Classical Version

lemma $Sync_{Classic}$-*assoc* :
‹*P* $[\![S]\!]_{\checkmark Classic}$ (*Q* $[\![S]\!]_{\checkmark Classic}$ *R*) = *P* $[\![S]\!]_{\checkmark Classic}$ *Q* $[\![S]\!]_{\checkmark Classic}$ *R*›
⟨*proof*⟩

10.2.2 Product Type

lemma $Sync_{Pair}$-*assoc* :
‹*P* $[\![S]\!]_{\checkmark Pair}$ (*Q* $[\![S]\!]_{\checkmark Pair}$ *R*) = *RenamingTick* (*P* $[\![S]\!]_{\checkmark Pair}$ *Q* $[\![S]\!]_{\checkmark Pair}$ *R*) (λ((*r*, *s*), *t*). (*r*, *s*, *t*))›
⟨*proof*⟩

10.2.3 List Type

lemma $Sync_{Rlist}$-$Sync_{Pairlist}$-$assoc$:
$‹P\ [\![S]\!]_{✓Rlist}\ (Q\ [\![S]\!]_{✓Pairlist}\ R) = (P\ [\![S]\!]_{✓Pairlist}\ Q)\ [\![S]\!]_{✓Llist}\ R›$
⟨*proof*⟩

lemma $Sync_{Rlist}$-$Sync_{Llist}$-$assoc$:
$‹P\ [\![S]\!]_{✓Rlist}\ (Q\ [\![S]\!]_{✓Llist}\ R) = (P\ [\![S]\!]_{✓Rlist}\ Q)\ [\![S]\!]_{✓Llist}\ R›$
⟨*proof*⟩

lemma $Sync_{Rlist}$-$Sync_{ListslenL}$-$assoc$:
$‹P\ [\![S]\!]_{✓Rlist}\ (Q\ lenQ[\![S]\!]_{✓ListslenL}\ R) = (P\ [\![S]\!]_{✓Rlist}\ Q)\ Suc\ lenQ[\![S]\!]_{✓ListslenL}\ R›$
⟨*proof*⟩

lemma $Sync_{ListslenR}$-$Sync_{Llist}$-$assoc$:
$‹P\ Suc\ lenQ[\![S]\!]_{✓ListslenR}\ (Q\ [\![S]\!]_{✓Llist}\ R) = (P\ lenQ[\![S]\!]_{✓ListslenR}\ Q)\ [\![S]\!]_{✓Llist}\ R›$
⟨*proof*⟩

lemma $Sync_{Lists}$-$assoc$:
$‹P\ lenP[\![S]\!]_{✓lenQ\ +\ lenR}\ (Q\ lenQ[\![S]\!]_{✓lenR}\ R) =$
$P\ lenP[\![S]\!]_{✓lenQ}\ Q\ lenP\ +\ lenQ[\![S]\!]_{✓lenR}\ R›$
⟨*proof*⟩

lemma $Sync_{Rlist}$-$Sync_{Rlist}$-$assoc$:
$‹P\ [\![S]\!]_{✓Rlist}\ (Q\ [\![S]\!]_{✓Rlist}\ R) = (P\ [\![S]\!]_{✓Pairlist}\ Q)\ Suc\ (Suc\ 0)[\![S]\!]_{✓ListslenL}\ R›$
⟨*proof*⟩

lemma $Sync_{ListslenR}$-$Sync_{Pairlist}$-$assoc$:
$‹P\ Suc\ (Suc\ 0)[\![S]\!]_{✓ListslenR}\ (Q\ [\![S]\!]_{✓Pairlist}\ R) = (P\ [\![S]\!]_{✓Llist}\ Q)\ [\![S]\!]_{✓Llist}\ R›$
⟨*proof*⟩

10.3 Properties

10.3.1 Actual Generalization

We can actually recover the classical synchronization product defined in session `HOL-CSP` as a particular case of our generalization.

theorem $Sync_{Classic}$-is-$Sync$: $‹P\ [\![A]\!]_{✓Classic}\ Q = P\ [\![A]\!]\ Q›$
⟨*proof*⟩

10.3.2 Other Properties

lemma ‹$Sync_{Lists}$.$Sync_{ptick}$-locale-sym.$Sync_{ptick}$ *lenL lenR Q A P* = *P* $_{lenL}[\![A]\!]_{\checkmark lenR}$ *Q*›
⟨*proof*⟩

corollary *TickSwap-Sync*$_{Pair}$ [*simp*] : ‹*TickSwap* (*P* $[\![S]\!]_{\checkmark Pair}$ *Q*) = *Q* $[\![S]\!]_{\checkmark Pair}$ *P*›
⟨*proof*⟩

lemma *TickSwap-is-Sync*$_{Pair}$*-iff* [*simp*] :
‹*TickSwap P* = *Q* $[\![S]\!]_{\checkmark Pair}$ *R* ⟷ *P* = *R* $[\![S]\!]_{\checkmark Pair}$ *Q*›
⟨*proof*⟩

corollary *Sync*$_{Classic}$*-commute* : ‹*P* $[\![S]\!]_{\checkmark Classic}$ *Q* = *Q* $[\![S]\!]_{\checkmark Classic}$ *P*›
⟨*proof*⟩

lemma ‹*RenamingTick* (*P* $_{lenL}[\![S]\!]_{\checkmark lenR}$ *Q*) (λ*r-s*. *drop lenL r-s* @ *take lenL r-s*) =
Q $_{lenR}[\![S]\!]_{\checkmark lenL}$ *P*›
⟨*proof*⟩

10.4 Ticks Length and Conversions

Through *RenamingTick*, conversions can be established between the interpretations. For this, we sometimes need an assumption about the length of the ticks.

10.4.1 Ticks Length

Definition and first Properties

definition *is-ticks-length* ::
‹*nat* ⇒ ('*a*, '*r list*) *process*$_{ptick}$ ⇒ *bool*› (‹(*length*$_{\checkmark}$)-('(-'))›)
where ‹*length*$_{\checkmark n}$(*P*) ≡ ∀ *rs* ∈ ✓**s**(*P*). *length rs* = *n*›

We might imagine ∀ *rs*∈✓*s*(*P*). *length rs* = *n* instead. But when the process *P* has divergences, the predicate would not hold. Additionally, we only need the control about traces that are not divergences.

lemma *is-ticks-lengthI* : ‹(⋀*rs*. *rs* ∈ ✓**s**(*P*) ⟹ *length rs* = *n*) ⟹ *length*$_{\checkmark n}$(*P*)›
⟨*proof*⟩

lemma *is-ticks-lengthD* : ‹*length*$_{\checkmark n}$(*P*) ⟹ *rs* ∈ ✓**s**(*P*) ⟹ *length rs* = *n*›
⟨*proof*⟩

lemma *is-ticks-length-unique* :
— Not suitable for simplifier.
‹$length_{\checkmark}\, n(P) \longleftrightarrow \checkmark\mathbf{s}(P) = \{\} \vee (\forall\, m.\ length_{\checkmark}\, m(P) \longleftrightarrow m = n)$›
⟨*proof*⟩

lemma *empty-strict-ticks-of-imp-is-ticks-length* :
‹$\checkmark\mathbf{s}(P) = \{\} \Longrightarrow length_{\checkmark}\, n(P)$›
⟨*proof*⟩

lemma *nonempty-strict-ticks-of-imp-is-ticks-length-unique* :
‹$\checkmark\mathbf{s}(P) \neq \{\} \Longrightarrow length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, m(P) \Longrightarrow m = n$›
⟨*proof*⟩

Behaviour

named-theorems *is-ticks-length-simp*
named-theorems *is-ticks-length-intro*

Constant Processes **lemma** *is-ticks-length-STOP* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(STOP)$› ⟨*proof*⟩

lemma *is-ticks-length-BOT* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(\bot)$› ⟨*proof*⟩

lemma *is-ticks-length-SKIP-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(SKIP\ rs) \longleftrightarrow length\ rs = n$›
⟨*proof*⟩

lemma *is-ticks-length-SKIPS-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(SKIPS\ R) \longleftrightarrow (\forall\, rs \in R.\ length\ rs = n)$›
⟨*proof*⟩

Binary (or less) Operators **lemma** *is-ticks-length-Ndet* [*is-ticks-length-intro*]
:
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \sqcap Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Det* [*is-ticks-length-intro*] :
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \mathbin{\square} Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Sliding* [*is-ticks-length-intro*] :
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \rhd Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Sync* [*is-ticks-length-intro*] :
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \mathbin{[\![S]\!]} Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Seq* [*is-ticks-length-intro*] :
‹*non-terminating* $P \vee$ *length*$_{\checkmark} n(Q) \Longrightarrow$ *length*$_{\checkmark} n(P \mathbin{;} Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Hiding* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark} n(P \setminus S)$› **if** ‹*length*$_{\checkmark} n(P)$›
⟨*proof*⟩

lemma *is-ticks-length-Interrupt* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark} n(P) \Longrightarrow$ *length*$_{\checkmark} n(Q) \Longrightarrow$ *length*$_{\checkmark} n(P \triangle Q)$›
⟨*proof*⟩
lemma *strict-ticks-Throw-subset* :
‹$\checkmark\mathbf{s}(P \Theta\ a{\in}A.\ Q\ a) \subseteq \checkmark\mathbf{s}(P) \cup (\bigcup a{\in}A \cap \alpha(P).\ \checkmark\mathbf{s}(Q\ a))$›
⟨*proof*⟩

lemma *is-ticks-length-Throw* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark} n(P \Theta\ a \in A.\ Q\ a)$›
if ‹*length*$_{\checkmark} n(P)$› ‹$\bigwedge a.\ a \in \alpha(P) \Longrightarrow$ *length*$_{\checkmark} n(Q\ a)$›
⟨*proof*⟩

lemma *is-ticks-length-Renaming* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark} n$(*Renaming* $P\ f\ g$) › **if** ‹$\bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow$ *length* $(g\ r) = n$›
⟨*proof*⟩

Architectural Operators **lemma** *is-ticks-length-GlobalNdet* [*is-ticks-length-intro*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *length*$_{\checkmark} n(P\ a)) \Longrightarrow$ *length*$_{\checkmark} n(\sqcap a \in A.\ P\ a)$›
⟨*proof*⟩

lemma *is-ticks-length-GlobalDet* [*is-ticks-length-intro*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *length*$_{\checkmark} n(P\ a)) \Longrightarrow$ *length*$_{\checkmark} n(\Box a \in A.\ P\ a)$›
⟨*proof*⟩

lemma *is-ticks-length-MultiSync* [*is-ticks-length-intro*] :
‹$(\bigwedge m.\ m \in$ *set-mset* $M \Longrightarrow$ *length*$_{\checkmark} n(P\ m)) \Longrightarrow$ *length*$_{\checkmark} n([\![S]\!]\ m \in\# M.\ P\ m)$›
⟨*proof*⟩

lemma *is-ticks-length-MultiSeq* [*is-ticks-length-intro*] :
‹$L \neq [] \Longrightarrow$ *length*$_{\checkmark} n(P$ (*last* $L)) \Longrightarrow$ *length*$_{\checkmark} n(SEQ\ l \in@\ L.\ P\ l)$›
⟨*proof*⟩

Communications **lemma** *is-ticks-length-write0-iff* [*is-ticks-length-simp*] :
‹*length*$_{\checkmark} n(e \rightarrow P) \longleftrightarrow$ *length*$_{\checkmark} n(P)$›
⟨*proof*⟩

lemma *is-ticks-length-write-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark n}(c!e \rightarrow P) \longleftrightarrow length_{\checkmark n}(P)$›
⟨*proof*⟩

lemma *is-ticks-length-Mprefix-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark n}(\Box a \in A \rightarrow P\ a) = (\forall\, a \in A.\ length_{\checkmark n}(P\ a))$›
⟨*proof*⟩

lemma *is-ticks-length-read-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark n}(c?a{\in}A \rightarrow P\ a) = (\forall\, b \in c\ `\ A.\ length_{\checkmark n}(P\ (inv\text{-}into\ A\ c\ b)))$›
⟨*proof*⟩

corollary ‹$inj\text{-}on\ c\ A \Longrightarrow length_{\checkmark n}(c?a{\in}A \rightarrow P\ a) = (\forall\, a \in A.\ length_{\checkmark n}(P\ a))$›
⟨*proof*⟩

lemma *is-ticks-length-Mndetprefix-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark n}(\sqcap a \in A \rightarrow P\ a) = (\forall\, a \in A.\ length_{\checkmark n}(P\ a))$›
⟨*proof*⟩

lemma *is-ticks-length-ndet-write-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark n}(c!!a{\in}A \rightarrow P\ a) = (\forall\, b \in c\ `\ A.\ length_{\checkmark n}(P\ (inv\text{-}into\ A\ c\ b)))$›
⟨*proof*⟩

corollary ‹$inj\text{-}on\ c\ A \Longrightarrow length_{\checkmark n}(c!!a{\in}A \rightarrow P\ a) = (\forall\, a \in A.\ length_{\checkmark n}(P\ a))$›
⟨*proof*⟩

Generalizations **lemma** *strict-ticks-of-Seq$_{ptick}$-subset* : ‹$\checkmark\mathbf{s}(P\ ;_{\checkmark}\ Q) \subseteq \bigcup \{\checkmark\mathbf{s}(Q\ r)\ |r.\ r \in \checkmark\mathbf{s}(P)\}$›
⟨*proof*⟩

lemma *non-terminating-Seq$_{ptick}$* :
‹$P\ ;_{\checkmark}\ Q = RenamingTick\ P\ g$› **if** ‹*non-terminating P*›
⟨*proof*⟩

lemma *is-ticks-length-Seq$_{ptick}$* [*is-ticks-length-intro*] :
‹$non\text{-}terminating\ P \vee (\forall\, r{\in}\checkmark\mathbf{s}(P).\ length_{\checkmark n}(Q\ r)) \Longrightarrow length_{\checkmark n}(P\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Sync$_{ptick}$* :
‹$length_{\checkmark n}(Sync_{ptick}\text{-}locale.Sync_{ptick}\ tick\text{-}join\ P\ A\ Q)$›
— We cannot work directly inside the locale since in this context the types of ticks $'t$ cannot be set to $'r\ list$.
if ‹$Sync_{ptick}\text{-}locale\ tick\text{-}join$›
and ‹$\bigwedge r\ s.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow s \in \checkmark\mathbf{s}(Q) \Longrightarrow$

case tick-join r s of ◊ ⇒ True | ⌊r-s⌋ ⇒ length r-s = n›
⟨*proof*⟩

lemma *is-ticks-length-One-RenamingTick-singl* [*is-ticks-length-simp*] :
‹$length_{\checkmark Suc\ 0}(RenamingTick\ P\ (\lambda r.\ [r]))$›
⟨*proof*⟩

lemma *is-ticks-length-Two-Sync*$_{Pairlist}$ [*is-ticks-length-simp*] :
‹$length_{\checkmark Suc\ (Suc\ 0)}(P\ [\![S]\!]_{\checkmark Pairlist}\ Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Suc-Sync*$_{Rlist}$ [*is-ticks-length-intro*] :
‹$length_{\checkmark} n(Q) \Longrightarrow length_{\checkmark Suc\ n}(P\ [\![S]\!]_{\checkmark Rlist}\ Q)$›
⟨*proof*⟩

The equivalence is false.

lemma *False* **if** ‹$\bigwedge P\ Q\ n.\ length_{\checkmark Suc\ n}(P\ [\![S]\!]_{\checkmark Rlist}\ Q) \Longrightarrow length_{\checkmark} n(Q)$›
⟨*proof*⟩

lemma *is-ticks-length-Suc-Sync*$_{Llist}$ [*is-ticks-length-intro*] :
‹$length_{\checkmark} n(P) \Longrightarrow length_{\checkmark Suc\ n}(P\ [\![S]\!]_{\checkmark Llist}\ Q)$›
⟨*proof*⟩

lemma *is-ticks-length-sum-Sync*$_{ListslenL}$ [*is-ticks-length-intro*] :
‹$length_{\checkmark} m(Q) \Longrightarrow length_{\checkmark} n + m(P\ n[\![S]\!]_{\checkmark ListslenL}\ Q)$›
⟨*proof*⟩

lemma *is-ticks-length-sum-Sync*$_{ListslenR}$ [*is-ticks-length-intro*] :
‹$length_{\checkmark} n(P) \Longrightarrow length_{\checkmark} n + m(P\ m[\![S]\!]_{\checkmark ListslenR}\ Q)$›
⟨*proof*⟩

lemma *is-ticks-length-sum-Sync*$_{Lists}$ [*is-ticks-length-intro*] :
‹$length_{\checkmark} n + m(P\ n[\![S]\!]_{\checkmark} m\ Q)$›
⟨*proof*⟩

10.4.2 Conversions

lemma *Sync*$_{Pairlist}$*-to-Sync*$_{Rlist}$:
‹$P\ [\![S]\!]_{\checkmark Pairlist}\ Q = P\ [\![S]\!]_{\checkmark Rlist}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
⟨*proof*⟩

lemma *Sync*$_{Pairlist}$*-to-Sync*$_{Llist}$:
‹$P\ [\![S]\!]_{\checkmark Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ [\![S]\!]_{\checkmark Llist}\ Q$›
⟨*proof*⟩

lemma $Sync_{Rlist}$-to-$Sync_{ListslenL}$:
‹$P\ [\![S]\!]_{✓Rlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ _{Suc\ 0}[\![S]\!]_{✓ListslenL}\ Q$›
⟨*proof*⟩

lemma $Sync_{Llist}$-to-$Sync_{ListslenR}$:
‹$P\ [\![S]\!]_{✓Llist}\ Q = P\ _{Suc\ 0}[\![S]\!]_{✓ListslenR}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
⟨*proof*⟩

lemma $Sync_{ListslenL}$-to-$Sync_{Lists}$:
‹$length_{✓}m(Q) \Longrightarrow P\ n[\![S]\!]_{✓ListslenL}\ Q = P\ n[\![S]\!]_{✓}m\ Q$›
⟨*proof*⟩

lemma $Sync_{ListslenR}$-to-$Sync_{Lists}$:
‹$length_{✓}n(P) \Longrightarrow P\ m[\![S]\!]_{✓ListslenR}\ Q = P\ n[\![S]\!]_{✓}m\ Q$›
⟨*proof*⟩

corollary $Sync_{ListslenL}$-is-$Sync_{ListslenR}$:
‹$length_{✓}n(P) \Longrightarrow length_{✓}m(Q) \Longrightarrow P\ n[\![S]\!]_{✓ListslenL}\ Q = P\ m[\![S]\!]_{✓ListslenR}\ Q$›
⟨*proof*⟩

corollary $Sync_{Pairlist}$-to-$Sync_{ListslenL}$:
‹$P\ [\![S]\!]_{✓Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ _{Suc\ 0}[\![S]\!]_{✓ListslenL}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
⟨*proof*⟩

corollary $Sync_{Pairlist}$-to-$Sync_{ListslenR}$:
‹$P\ [\![S]\!]_{✓Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ _{Suc\ 0}[\![S]\!]_{✓ListslenR}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
⟨*proof*⟩

corollary $Sync_{Rlist}$-to-$Sync_{Lists}$:
‹$length_{✓}m(Q) \Longrightarrow P\ [\![S]\!]_{✓Rlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ _{Suc\ 0}[\![S]\!]_{✓}m\ Q$›
⟨*proof*⟩

corollary $Sync_{Llist}$-to-$Sync_{Lists}$:
‹$length_{✓}n(P) \Longrightarrow P\ [\![S]\!]_{✓Llist}\ Q = P\ n[\![S]\!]_{✓Suc\ 0}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
⟨*proof*⟩

corollary $Sync_{Pairlist}$-to-$Sync_{Lists}$:
‹$P\ [\![S]\!]_{✓Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ _{Suc\ 0}[\![S]\!]_{✓Suc\ 0}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
⟨*proof*⟩

lemma $Sync_{Pair}$-to-$Sync_{Pairlist}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). [*r*, *s*]) = *P* $[\![S]\!]_{✓Pairlist}$ *Q*›
⟨*proof*⟩

lemma $Sync_{Pairlist}$-to-$Sync_{Pair}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pairlist}$ *Q*) (λ*rs*. (*rs* ! *0*, *rs* ! *Suc 0*)) = *P* $[\![S]\!]_{✓Pair}$ *Q*›
⟨*proof*⟩

lemma $Sync_{Pair}$-to-$Sync_{Rlist}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* # *s*) = *P* $[\![S]\!]_{✓Rlist}$ *Q*›
⟨*proof*⟩

lemma $Sync_{Pair}$-to-$Sync_{Llist}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ [*s*]) = *P* $[\![S]\!]_{✓Llist}$ *Q*›
⟨*proof*⟩

lemma $Sync_{Pair}$-to-$Sync_{ListslenL}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ *s*) = *P* $_n[\![S]\!]_{✓ListslenL}$ *Q*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$length_{✓n}$(*P*)›
⟨*proof*⟩

corollary $Sync_{Pair}$-to-$Sync_{ListslenR}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ *s*) = *P* $_n[\![S]\!]_{✓ListslenR}$ *Q*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$length_{✓n}$(*Q*)›
⟨*proof*⟩

corollary $Sync_{Pair}$-to-$Sync_{Lists}$:
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ *s*) = *P* $_n[\![S]\!]_{✓m}$ *Q*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$length_{✓n}$(*P*)› **and** ‹$length_{✓m}$(*Q*)›
⟨*proof*⟩

10.5 First Laws

corollary $Inter_{Classic}$-*STOP* [*simp*] :
‹*P* $|||_{✓Classic}$ *STOP* = *P* ; *STOP*›
⟨*proof*⟩

corollary $Inter_{Pair}$-*STOP* :
‹*P* $|||_{✓Pair}$ *STOP* = *RenamingTick* (*P* ; *STOP*) (λ*r*. (*r*, *g r*))›
⟨*proof*⟩

corollary $Inter_{Pairlist}$-*STOP* :
‹*P* $|||_{✓Pairlist}$ *STOP* = *RenamingTick* (*P* ; *STOP*) (λ*r*. [*r*, *g r*])›
⟨*proof*⟩

corollary $Inter_{Rlist}$-*STOP* :

‹P $|||_{\checkmark Rlist}$ $STOP$ = $RenamingTick$ (P ; $STOP$) ($\lambda r.$ r # g r)›
⟨$proof$⟩

corollary $Inter_{Llist}$-$STOP$:
‹P $|||_{\checkmark Llist}$ $STOP$ = $RenamingTick$ (P ; $STOP$) ($\lambda r.$ r @ [g r])›
⟨$proof$⟩

corollary $Inter_{ListslenL}$-$STOP$:
‹P $_n|||_{\checkmark ListslenL}$ $STOP$ =
$RenamingTick$ (P ; $STOP$) ($\lambda r.$ *if length* r = n *then* r @ g r *else undefined*)›
⟨$proof$⟩

corollary $Inter_{ListslenR}$-$STOP$:
‹P $_n|||_{\checkmark ListslenR}$ $STOP$ =
$RenamingTick$ (P ; $STOP$) ($\lambda r.$ *if length* (g r) = n *then* r @ g r *else undefined*)›
⟨$proof$⟩

corollary $Inter_{Lists}$-$STOP$:
‹P $_n|||_{\checkmark m}$ $STOP$ =
$RenamingTick$ (P ; $STOP$) ($\lambda r.$ *if length* r = n ∧ *length* (g r) = m *then* r @ g r *else undefined*)›
⟨$proof$⟩

corollary $STOP$-$Inter_{Classic}$ [$simp$] :
‹$STOP$ $|||_{\checkmark Classic}$ Q = Q ; $STOP$›
⟨$proof$⟩

corollary $STOP$-$Inter_{Pair}$:
‹$STOP$ $|||_{\checkmark Pair}$ Q = $RenamingTick$ (Q ; $STOP$) ($\lambda s.$ (g s, s))›
⟨$proof$⟩

corollary $STOP$-$Inter_{Pairlist}$:
‹$STOP$ $|||_{\checkmark Pairlist}$ Q = $RenamingTick$ (Q ; $STOP$) ($\lambda s.$ [g s, s])›
⟨$proof$⟩

corollary $STOP$-$Inter_{Rlist}$:
‹$STOP$ $|||_{\checkmark Rlist}$ Q = $RenamingTick$ (Q ; $STOP$) ($\lambda s.$ g s # s)›
⟨$proof$⟩

corollary $STOP$-$Inter_{Llist}$:
‹$STOP$ $|||_{\checkmark Llist}$ Q = $RenamingTick$ (Q ; $STOP$) ($\lambda s.$ g s @ [s])›
⟨$proof$⟩

corollary $STOP$-$Inter_{ListslenL}$:
‹$STOP$ $_n|||_{\checkmark ListslenL}$ Q =
$RenamingTick$ (Q ; $STOP$) ($\lambda r.$ *if length* (g r) = n *then* g r @ r *else undefined*)›
⟨$proof$⟩

corollary *STOP-Inter*$_{ListslenR}$:
‹*STOP* $_n|||_{\checkmark ListslenR}$ Q =
RenamingTick (Q ; *STOP*) (λr. *if length* $r = n$ *then* $g\ r$ @ r *else undefined*)›
⟨*proof*⟩

corollary *STOP-Inter*$_{Lists}$:
‹*STOP* $_n|||_{\checkmark m}$ Q =
RenamingTick (Q ; *STOP*) (λr. *if length* ($g\ r$) = $n \wedge$ *length* $r = m$ *then* $g\ r$ @ r *else undefined*)›
⟨*proof*⟩

corollary *SKIP-Sync*$_{Classic}$*-SKIP* :
‹*SKIP* r $[\![A]\!]_{\checkmark Classic}$ *SKIP* s =
(*if* $r = s$ *then SKIP* r *else STOP*)› ⟨*proof*⟩

corollary *SKIP-Sync*$_{Pair}$*-SKIP* :
‹*SKIP* r $[\![A]\!]_{\checkmark Pair}$ *SKIP* s = *SKIP* (r, s)› ⟨*proof*⟩

corollary *SKIP-Sync*$_{Pairlist}$*-SKIP* :
‹*SKIP* r $[\![A]\!]_{\checkmark Pairlist}$ *SKIP* s = *SKIP* [r, s]› ⟨*proof*⟩

corollary *SKIP-Sync*$_{Rlist}$*-SKIP* :
‹*SKIP* r $[\![A]\!]_{\checkmark Rlist}$ *SKIP* s = *SKIP* (r # s)› ⟨*proof*⟩

corollary *SKIP-Sync*$_{Llist}$*-SKIP* :
‹*SKIP* r $[\![A]\!]_{\checkmark Llist}$ *SKIP* s = *SKIP* (r @ [s])› ⟨*proof*⟩

corollary *SKIP-Sync*$_{ListslenL}$*-SKIP* :
‹*SKIP* r $_n[\![A]\!]_{\checkmark ListslenL}$ *SKIP* s =
(*if length* $r = n$ *then SKIP* (r @ s) *else STOP*)› ⟨*proof*⟩

corollary *SKIP-Sync*$_{ListslenR}$*-SKIP* :
‹*SKIP* r $_n[\![A]\!]_{\checkmark ListslenR}$ *SKIP* s =
(*if length* $s = n$ *then SKIP* (r @ s) *else STOP*)› ⟨*proof*⟩

corollary *SKIP-Sync*$_{Lists}$*-SKIP* :
‹*SKIP* r $_n[\![A]\!]_{\checkmark m}$ *SKIP* s =
(*if length* $r = n \wedge$ *length* $s = m$ *then SKIP* (r @ s) *else STOP*)› ⟨*proof*⟩

10.6 Operational Laws

10.6.1 Classical Version

locale *After-Sync*$_{Classic}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s$. *if* $r = s$ *then* $\lfloor r \rfloor$ *else* $\diamondsuit$›
begin

— Just checking...
lemma ‹$Sync_{ptick}\ P\ S\ Q = P\ [\![S]\!]_{\checkmark Classic}\ Q$› ⟨*proof*⟩

end

locale *AfterExt-Sync*$_{Classic}$*-locale* =
AfterExt-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.\ if\ r = s\ then\ \lfloor r \rfloor\ else\ \Diamond$›

sublocale *AfterExt-Sync*$_{Classic}$*-locale* ⊆ *After-Sync*$_{Classic}$*-locale*
⟨*proof*⟩

locale *OpSemTransitions-Sync*$_{Classic}$*-locale* =
OpSemTransitions-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.\ if\ r = s\ then\ \lfloor r \rfloor\ else\ \Diamond$›

sublocale *OpSemTransitions-Sync*$_{Classic}$*-locale* ⊆ *AfterExt-Sync*$_{Classic}$*-locale*
⟨*proof*⟩

10.6.2 Product Type

locale *After-Sync*$_{Pair}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor (r,\ s) \rfloor$›
begin

— Just checking...
lemma ‹$Sync_{ptick}\ P\ S\ Q = P\ [\![S]\!]_{\checkmark Pair}\ Q$› ⟨*proof*⟩

end

locale *AfterExt-Sync*$_{Pair}$*-locale* =
AfterExt-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor (r,\ s) \rfloor$›

sublocale *AfterExt-Sync*$_{Pair}$*-locale* ⊆ *After-Sync*$_{Pair}$*-locale*
⟨*proof*⟩

locale *OpSemTransitions-Sync*$_{Pair}$*-locale* =
OpSemTransitions-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor (r,\ s) \rfloor$›

sublocale *OpSemTransitions-Sync*$_{Pair}$*-locale* ⊆ *AfterExt-Sync*$_{Pair}$*-locale*
⟨*proof*⟩

10.6.3 List Type

Pair

locale *After-Sync*$_{Pairlist}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor [r,\ s] \rfloor$›
begin

— Just checking...
lemma ‹$Sync_{ptick}\ P\ S\ Q = P\ [\![S]\!]_{\checkmark Pairlist}\ Q$› ⟨*proof*⟩

end

locale *AfterExt-Sync$_{Pairlist}$-locale* =
AfterExt-Sync$_{ptick}$-locale ‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$›

sublocale *AfterExt-Sync$_{Pairlist}$-locale* ⊆ *After-Sync$_{Pairlist}$-locale*
⟨*proof*⟩

locale *OpSemTransitions-Sync$_{Pairlist}$-locale* =
OpSemTransitions-Sync$_{ptick}$-locale ‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$›

sublocale *OpSemTransitions-Sync$_{Pairlist}$-locale* ⊆ *AfterExt-Sync$_{Pairlist}$-locale*
⟨*proof*⟩

Right List

locale *After-Sync$_{Rlist}$-locale* = *After-Sync$_{ptick}$-locale* ‹$\lambda r\ s.\ \lfloor r\ \#\ s\rfloor$›
begin

— Just checking...
lemma ‹*Sync$_{ptick}$ P S Q* = *P* $[\![S]\!]_{\checkmark Rlist}$ *Q*› ⟨*proof*⟩

end

locale *AfterExt-Sync$_{Rlist}$-locale* =
AfterExt-Sync$_{ptick}$-locale ‹$\lambda r\ s.\ \lfloor r\ \#\ s\rfloor$›

sublocale *AfterExt-Sync$_{Rlist}$-locale* ⊆ *After-Sync$_{Rlist}$-locale*
⟨*proof*⟩

locale *OpSemTransitions-Sync$_{Rlist}$-locale* =
OpSemTransitions-Sync$_{ptick}$-locale ‹$\lambda r\ s.\ \lfloor r\ \#\ s\rfloor$›

sublocale *OpSemTransitions-Sync$_{Rlist}$-locale* ⊆ *AfterExt-Sync$_{Rlist}$-locale*
⟨*proof*⟩

Left List

locale *After-Sync$_{Llist}$-locale* = *After-Sync$_{ptick}$-locale* ‹$\lambda r\ s.\ \lfloor r\ @\ [s]\rfloor$›
begin

— Just checking...
lemma ‹*Sync$_{ptick}$ P S Q* = *P* $[\![S]\!]_{\checkmark Llist}$ *Q*› ⟨*proof*⟩

end

locale *AfterExt-Sync$_{Llist}$-locale* =
AfterExt-Sync$_{ptick}$-locale ‹$\lambda r\ s.\ \lfloor r\ @\ [s]\rfloor$›

sublocale *AfterExt-Sync$_{Llist}$-locale* ⊆ *After-Sync$_{Llist}$-locale*

⟨*proof*⟩

locale *OpSemTransitions-Sync*$_{Llist}$*-locale* =
OpSemTransitions-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor r\ @\ [s]\rfloor$›

sublocale *OpSemTransitions-Sync*$_{Llist}$*-locale* ⊆ *AfterExt-Sync*$_{Llist}$*-locale*
⟨*proof*⟩

Arbitrary Lists

Control on left side **locale** *After-Sync*$_{ListslenL}$*-locale* =
After-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.$ *if length* r = *lenL then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
for *lenL* :: *nat*
begin

— Just checking...
lemma ‹*Sync*$_{ptick}$ P S Q = P $_{lenL}[\![S]\!]_{\checkmark ListslenL}$ Q› ⟨*proof*⟩

end

locale *AfterExt-Sync*$_{ListslenL}$*-locale* =
AfterExt-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.$ *if length* r = *lenL then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
for *lenL* :: *nat*

sublocale *AfterExt-Sync*$_{ListslenL}$*-locale* ⊆ *After-Sync*$_{ListslenL}$*-locale*
⟨*proof*⟩

locale *OpSemTransitions-Sync*$_{ListslenL}$*-locale* =
OpSemTransitions-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.$ *if length* r = *lenL then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
for *lenL* :: *nat*

sublocale *OpSemTransitions-Sync*$_{ListslenL}$*-locale* ⊆ *AfterExt-Sync*$_{ListslenL}$*-locale*
⟨*proof*⟩

Control on right side **locale** *After-Sync*$_{ListslenR}$*-locale* =
After-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.$ *if length* s = *lenR then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
for *lenR* :: *nat*
begin

— Just checking...
lemma ‹*Sync*$_{ptick}$ P S Q = P $_{lenR}[\![S]\!]_{\checkmark ListslenR}$ Q› ⟨*proof*⟩

end

locale *AfterExt-Sync*$_{ListslenR}$*-locale* =
AfterExt-Sync$_{ptick}$*-locale* ‹$\lambda r\ s.$ *if length* s = *lenR then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
for *lenR* :: *nat*

sublocale *AfterExt-Sync*$_{ListslenR}$*-locale* ⊆ *After-Sync*$_{ListslenR}$*-locale*

⟨*proof*⟩

locale *OpSemTransitions-Sync*$_{Listslen_R}$*-locale* =
OpSemTransitions-Sync$_{ptick}$*-locale* ‹*λr s. if length r = lenL then* ⌊*r @ s*⌋ *else* ◊›
for *lenL* :: *nat*

sublocale *OpSemTransitions-Sync*$_{Listslen_R}$*-locale* ⊆ *AfterExt-Sync*$_{Listslen_R}$*-locale*
⟨*proof*⟩

Control on both sides **locale** *After-Sync*$_{Lists}$*-locale* =
After-Sync$_{ptick}$*-locale*
‹*λr s. if length r = lenL ∧ length s = lenR then* ⌊*r @ s*⌋ *else* ◊›
for *lenL lenR* :: *nat*
begin

— Just checking...
lemma ‹*Sync*$_{ptick}$ *P S Q* = *P* $_{lenL}[\![S]\!]_{✓lenR}$ *Q*› ⟨*proof*⟩

end

locale *AfterExt-Sync*$_{Lists}$*-locale* =
AfterExt-Sync$_{ptick}$*-locale*
‹*λr s. if length r = lenL ∧ length s = lenR then* ⌊*r @ s*⌋ *else* ◊›
for *lenL lenR* :: *nat*

sublocale *AfterExt-Sync*$_{Lists}$*-locale* ⊆ *After-Sync*$_{Lists}$*-locale*
⟨*proof*⟩

locale *OpSemTransitions-Sync*$_{Lists}$*-locale* =
OpSemTransitions-Sync$_{ptick}$*-locale*
‹*λr s. if length r = lenL ∧ length s = lenR then* ⌊*r @ s*⌋ *else* ◊›
for *lenL lenR* :: *nat*

sublocale *OpSemTransitions-Sync*$_{Lists}$*-locale* ⊆ *AfterExt-Sync*$_{Lists}$*-locale*
⟨*proof*⟩

Chapter 11

Architectural Versions

11.1 Sequential Composition

11.1.1 Definition

fun $MultiSeq_{ptick}$ :: ‹['b list, 'b ⇒ 'r ⇒ ('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
where $MultiSeq_{ptick}$-Nil : ‹$MultiSeq_{ptick}$ [] P = SKIP›
| $MultiSeq_{ptick}$-Cons : ‹$MultiSeq_{ptick}$ (l # L) P = (λr. P l r ;✓ $MultiSeq_{ptick}$ L P)›

syntax -$MultiSeq_{ptick}$ ::
‹[pttrn, 'b list, 'b ⇒ 'r ⇒ ('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
(‹(3$SEQ_{✓}$ - ∈@ -./ -)› [78,78,77] 77)
syntax-consts -$MultiSeq_{ptick}$ ⇌ $MultiSeq_{ptick}$
translations $SEQ_{✓}$ p ∈@ L. P ⇌ CONST $MultiSeq_{ptick}$ L (λp. P)

11.1.2 First Properties

lemma ‹$SEQ_{✓}$ p ∈@ []. P p = SKIP›
and ‹$SEQ_{✓}$ p ∈@ [a]. P p = (λr. P a r)›
and ‹$SEQ_{✓}$ p ∈@ [a, b]. P p = (λr. P a r ;✓ P b)›
and ‹$SEQ_{✓}$ p ∈@ [a, b, c]. P p = (λr. P a r ;✓ P b ;✓ P c)›
⟨proof⟩

lemma ‹$SEQ_{✓}$ p ∈@ [1::int .. 3]. P p = (λr. P 1 r ;✓ P 2 ;✓ P 3)›
⟨proof⟩

lemma ‹($SEQ_{✓}$ p ∈@ []. P p) = SKIP› ⟨proof⟩

lemma ‹($SEQ_{✓}$ l ∈@ (a # L). P l) = (λr. P a r ;✓ $SEQ_{✓}$ l ∈@ L. P l)› ⟨proof⟩

lemma *MultiSeq$_{ptick}$-singl* [*simp*] : ‹*SEQ$_{\checkmark}$ l ∈@ [a]. P l = P a*› ⟨*proof*⟩

lemma *MultiSeq$_{ptick}$-snoc* : ‹*SEQ$_{\checkmark}$ l ∈@ (L @ [a]). P l = (λr. (SEQ$_{\checkmark}$ l ∈@ L. P l) r ;$_{\checkmark}$ P a)*›
⟨*proof*⟩

lemma *mono-MultiSeq$_{ptick}$-eq*:
‹(⋀*l. l ∈ set L ⟹ P l = Q l*) ⟹ *SEQ$_{\checkmark}$ l ∈@ L. P l = SEQ$_{\checkmark}$ l ∈@ L. Q l*›
⟨*proof*⟩

lemma *MultiSeq$_{ptick}$-const* [*simp*] :
‹(*SEQ$_{\checkmark}$ l ∈@ L. (λr. P l)) =*
(if L = [] then SKIP else (λr. SEQ l ∈@ L. P l))›
⟨*proof*⟩

11.1.3 Behaviour with binary version

lemma *MultiSeq$_{ptick}$-append*:
‹*SEQ$_{\checkmark}$ l ∈@ (L1 @ L2). P l = (λr. (SEQ$_{\checkmark}$ l ∈@ L1. P l) r ;$_{\checkmark}$ SEQ$_{\checkmark}$ l ∈@ L2. P l)*›
⟨*proof*⟩

11.1.4 Other Properties

lemma *MultiSeq$_{ptick}$-SKIP-neutral*:
‹*P a = SKIP ⟹ SEQ$_{\checkmark}$ l ∈@ (L1 @ [a] @ L2). P l = SEQ$_{\checkmark}$ l ∈@ (L1 @ L2). P l*›
⟨*proof*⟩

lemma *MultiSeq$_{ptick}$-BOT-absorb*:
‹*P a = ⊥ ⟹ SEQ$_{\checkmark}$ l ∈@ (L1 @ [a] @ L2). P l = (λr. (SEQ$_{\checkmark}$ l ∈@ L1. P l) r ;$_{\checkmark}$ ⊥)*›
⟨*proof*⟩

lemma *MultiSeq$_{ptick}$-STOP-absorb*:
‹*P a = (λr. STOP) ⟹ SEQ$_{\checkmark}$ l ∈@ (L1 @ [a] @ L2). P l =*
(λr. (SEQ$_{\checkmark}$ l ∈@ L1. P l) r ; STOP)›
⟨*proof*⟩

lemma *is-ticks-length-MultiSeq$_{ptick}$* [*is-ticks-length-intro*] :
‹*length$_{\checkmark}$n((SEQ$_{\checkmark}$ l ∈@ L. P l) r)*›
if ‹*L ≠ []*› **and** ‹⋀*r′. r′ ∈ ✓s((SEQ$_{\checkmark}$ l ∈@ (butlast L). P l) r) ⟹ length$_{\checkmark}$n(P (last L) r′)*›
⟨*proof*⟩

11.1.5 Behaviour with injectivity

lemma *inj-on-mapping-over-MultiSeq$_{ptick}$*:

‹*inj-on f* (*set L*) ⟹
SEQ$_{\checkmark}$ *l* ∈@ *L*. *P l* = *SEQ*$_{\checkmark}$ *l* ∈@ *map f L*. *P* (*inv-into* (*set L*) *f l*)›
⟨*proof*⟩

unbundle *no funcset-syntax*
— Inherited from *HOL−Combinatorics.List-Permutation*.

11.2 Synchronization Product

11.2.1 Definition

The generalized synchronization product is not really commutative (see *RenamingTick* (*P* ⟦*A*⟧$_{\checkmark}$ *Q*) ⊗✓⇒⊗✓$_{rev}$ = *Q* ⟦*A*⟧$_{\checkmark rev}$ *P*). We therefore define the architectural version on a list.

fun *MultiSync*$_{ptick}$ ::
‹['*a set*, '*b list*, '*b* ⇒ ('*a*, '*r*) *process*$_{ptick}$] ⇒ ('*a*, '*r list*) *process*$_{ptick}$›
where ‹*MultiSync*$_{ptick}$ *S* [] *P* = *STOP*›
| ‹*MultiSync*$_{ptick}$ *S* [*l*] *P* = *RenamingTick* (*P l*) (λ*r*. [*r*])›
| ‹*MultiSync*$_{ptick}$ *S* (*l* # *m* # *L*) *P* = *P l* ⟦*S*⟧$_{\checkmark Rlist}$ *MultiSync*$_{ptick}$ *S* (*m* # *L*) *P*›

syntax -*MultiSync*$_{ptick}$ ::
‹[*pttrn*, '*a set*, '*b list*, ('*a*, '*r*) *process*$_{ptick}$] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
(‹(3⟦-⟧$_{\checkmark}$ - ∈@ -./ -)› [78,78,78,77] 77)
syntax-consts -*MultiSync*$_{ptick}$ ⇌ *MultiSync*$_{ptick}$
translations ⟦*S*⟧$_{\checkmark}$ *l* ∈@ *L*. *P* ⇌ *CONST MultiSync*$_{ptick}$ *S L* (λ*l*. *P*)

Special case of *MultiSync*$_{ptick}$ *S P* when *S* = {}.

abbreviation *MultiInter*$_{ptick}$ ::
‹['*b list*, '*b* ⇒ ('*a*, '*r*) *process*$_{ptick}$] ⇒ ('*a*, '*r list*) *process*$_{ptick}$›
where ‹*MultiInter*$_{ptick}$ *L P* ≡ *MultiSync*$_{ptick}$ {} *L P*›

syntax -*MultiInter*$_{ptick}$ ::
‹[*pttrn*, '*b list*, ('*a*, '*r*) *process*$_{ptick}$] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
(‹(3|||$_{\checkmark}$ -∈@-./ -)› [78,78,77] 77)
syntax-consts -*MultiInter*$_{ptick}$ ⇌ *MultiInter*$_{ptick}$
translations |||$_{\checkmark}$ *l* ∈@ *L*. *P* ⇌ *CONST MultiInter*$_{ptick}$ *L* (λ*l*. *P*)

Special case of *MultiSync*$_{ptick}$ *S P* when *S* = *UNIV*.

abbreviation *MultiPar*$_{ptick}$ ::
‹['*b list*, '*b* ⇒ ('*a*, '*r*) *process*$_{ptick}$] ⇒ ('*a*, '*r list*) *process*$_{ptick}$›
where ‹*MultiPar*$_{ptick}$ *L P* ≡ *MultiSync*$_{ptick}$ *UNIV L P*›

syntax -*MultiPar*$_{ptick}$ ::
‹[*pttrn*, '*b list*, ('*a*, '*r*) *process*$_{ptick}$] ⇒ ('*a*, '*r*) *process*$_{ptick}$›

(‹(3||$_{\checkmark}$ -∈@-./ -)› [78,78,77] 77)
syntax-consts -$MultiPar_{ptick}$ ⇌ $MultiPar_{ptick}$
translations ||$_{\checkmark}$ l ∈@ L. P ⇌ $CONST\ MultiPar_{ptick}\ L\ (\lambda l.\ P)$

11.2.2 First properties

lemma *is-ticks-length-MultiSync$_{ptick}$* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark}$ *length* L([$\![S]\!]_{\checkmark}$ l ∈@ L. P l)›
⟨*proof*⟩

lemma *MultiSync$_{ptick}$-Cons* :
‹[$\![S]\!]_{\checkmark}$ m ∈@ (l # L). P m =
(*if* L = [] *then RenamingTick* (P l) (λr. [r])
else P l [$\![S]\!]_{\checkmark Rlist}$ [$\![S]\!]_{\checkmark}$ m ∈@ L. P m)›
⟨*proof*⟩

lemma *mono-MultiSync$_{ptick}$-eq* :
‹(⋀l. l ∈ *set* L ⟹ P l = Q l) ⟹ [$\![S]\!]_{\checkmark}$ l ∈@ L. P l = [$\![S]\!]_{\checkmark}$ l ∈@ L. Q l›
⟨*proof*⟩

lemma *mono-MultiSync$_{ptick}$-eq2*:
‹(⋀l. l ∈ *set* L ⟹ P (f l) = Q l) ⟹ [$\![S]\!]_{\checkmark}$ l ∈@ *map* f L. P l = [$\![S]\!]_{\checkmark}$ l ∈@ L. Q l›
⟨*proof*⟩
lemma ‹([$\![S]\!]_{\checkmark}$ l ∈@ []. P l) = *STOP*›
and ‹([$\![S]\!]_{\checkmark}$ l ∈@ [a]. P l) = *RenamingTick* (P a) (λr. [r])›
and ‹([$\![S]\!]_{\checkmark}$ l ∈@ [a, b]. P l) = P a [$\![S]\!]_{\checkmark Rlist}$ *RenamingTick* (P b) (λr. [r])›
and ‹([$\![S]\!]_{\checkmark}$ l ∈@ [a, b, c]. P l) = P a [$\![S]\!]_{\checkmark Rlist}$ (P b [$\![S]\!]_{\checkmark Rlist}$ *RenamingTick* (P c) (λr. [r]))›
⟨*proof*⟩

11.2.3 Properties

lemma *MultiSync$_{ptick}$-is-BOT-iff*:
‹[$\![S]\!]_{\checkmark}$ l ∈@ L. P l = ⊥ ⟷ (∃ l ∈ *set* L. P l = ⊥)›
⟨*proof*⟩

lemma *MultiSync$_{ptick}$-BOT-absorb*:
‹l ∈ *set* L ⟹ P l = ⊥ ⟹ [$\![S]\!]_{\checkmark}$ l ∈@ L. P l = ⊥›
⟨*proof*⟩

lemma *MultiSync$_{ptick}$-SKIP-id* :
‹[$\![S]\!]_{\checkmark}$ r ∈@ L. *SKIP* r = (*if* L = [] *then STOP else SKIP* L)›
⟨*proof*⟩

11.2.4 Behaviour with binary version

lemma *MultiSync$_{ptick}$-append* :
‹*L1* ≠ [] ⟹ *L2* ≠ [] ⟹
⟦*S*⟧$_{✓}$ *l* ∈@ (*L1* @ *L2*). *P l* =
⟦*S*⟧$_{✓}$ *l* ∈@ *L1*. *P l* $_{length\ L1}$⟦*S*⟧$_{✓length\ L2}$ ⟦*S*⟧$_{✓}$ *l* ∈@ *L2*. *P l*›
⟨*proof*⟩

11.2.5 Behaviour with injectivity

lemma *inj-on-mapping-over-MultiSync$_{ptick}$*:
‹*inj-on f* (*set L*) ⟹
⟦*S*⟧$_{✓}$ *l* ∈@ *L*. *P l* = ⟦*S*⟧$_{✓}$ *l* ∈@ *map f L*. *P* (*inv-into* (*set L*) *f l*)›
⟨*proof*⟩

11.2.6 Permuting the Sequence

A particular Case

lemma *MultiSync$_{ptick}$-snoc* :
‹⟦*S*⟧$_{✓}$ *m* ∈@ (*L* @ [*l*]). *P m* =
(*if L* = [] *then RenamingTick* (*P l*) (λ*r*. [*r*])
else ⟦*S*⟧$_{✓}$ *m* ∈@ *L*. *P m* ⟦*S*⟧$_{✓Llist}$ *P l*)›
⟨*proof*⟩

At the beginning, we wanted to prove the following property.

theorem *MultiSync$_{ptick}$-rev* :
‹⟦*S*⟧$_{✓}$ *l* ∈@ (*rev L*). *P l* = *RenamingTick* (⟦*S*⟧$_{✓}$ *l* ∈@ *L*. *P l*) *rev*›
⟨*proof*⟩

This has just been established for *rev L*, which is a particular permutation of the list *L*. It turns out that it actually holds for any permutation. The rest of this file constitutes the proof.

Arbitrary Permutation

Some preliminary results **lemma** *permute-list-transpose-eq-list-update* :
‹*i* < *length xs* ⟹ *j* < *length xs* ⟹
permute-list (*Transposition.transpose i j*) *xs* = *xs*[*i* := *xs*!*j*, *j* := *xs*!*i*]›
⟨*proof*⟩

lemma *inj-on-permute-list-transpose* :
‹*i* < *n* ⟹ *j* < *n* ⟹ *inj-on* (*permute-list* (*Transposition.transpose i j*)) {*xs*. *n* ≤ *length xs*}›
⟨*proof*⟩

lemma *rev-permute-list-transpose* :
‹*i* < *length L* ⟹ *j* < *length L* ⟹
rev (*permute-list* (*Transposition.transpose i j*) *L*) =

permute-list (*Transposition.transpose* (*length L* − *Suc i*) (*length L* − *Suc j*)) (*rev L*)›
⟨*proof*⟩

lemma *permute-list-transpose-rev* :
‹*i* < *length L* ⟹ *j* < *length L* ⟹
permute-list (*Transposition.transpose i j*) (*rev L*) =
rev (*permute-list* (*Transposition.transpose* (*length L* − *Suc i*) (*length L* − *Suc j*)) *L*)›
⟨*proof*⟩

lemma *tickFree-map-map-event*$_{ptick}$*-id-eq* :
‹*tF t* ⟹ *map* (*map-event*$_{ptick}$ *id g*) *t* = *t*›
and *tickFree-mem-T-RenamingTick-iff-mem-T* :
‹*tF t* ⟹ *t* ∈ 𝒯 (*RenamingTick P g*) ⟷ *t* ∈ 𝒯 *P*›
and *tickFree-mem-D-RenamingTick-iff-mem-D* :
‹*tF t* ⟹ *t* ∈ 𝒟 (*RenamingTick P g*) ⟷ *t* ∈ 𝒟 *P*›
for *P* :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** *g* :: ‹'*r* ⇒ '*r*›
— Necessarily here, antecedents and images for *g* share the same type.
⟨*proof*⟩

The proof We start by proving that the *RenamingTick* of the right-hand side process Q by a transposition can be "taken to the outside" of the synchronization $P\ [\![S]\!]_{✓Rlist}\ Q$.

lemma *Sync*$_{Rlist}$*-RenamingTick-permute-list-transpose* :
‹*P* [⟦*S*⟧]$_{✓Rlist}$ *RenamingTick Q* (*permute-list* (*Transposition.transpose i j*)) =
RenamingTick (*P* [⟦*S*⟧]$_{✓Rlist}$ *Q*) (*permute-list* (*Transposition.transpose* (*Suc i*) (*Suc j*)))›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*i* < *n*› ‹*j* < *n*› ‹⋀*rs*. *rs* ∈ ✓**s**(*Q*) ⟹ *n* ≤ *length rs*›
⟨*proof*⟩

lemma *RenamingTick-permute-list-transpose-Sync*$_{ListslenL}$:
‹*RenamingTick P* (*permute-list* (*Transposition.transpose i j*)) $_n$[⟦*S*⟧]$_{✓ListslenL}$ *Q* =
RenamingTick (*P* $_n$[⟦*S*⟧]$_{✓ListslenL}$ *Q*) (*permute-list* (*Transposition.transpose i j*))›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*i* < *n*› ‹*j* < *n*› **for** *P* :: ‹('*a*, '*r list*) *process*$_{ptick}$›
⟨*proof*⟩

Then, we establish the result when the permutation is only a transposition.

lemma *MultiSync*$_{ptick}$*-permute-list-transpose* :
‹*i* < *length L* ⟹ *j* < *length L* ⟹
[⟦*S*⟧]$_{✓}$ *l* ∈@ *permute-list* (*Transposition.transpose i j*) *L*. *P l* =
RenamingTick ([⟦*S*⟧]$_{✓}$ *l* ∈@ *L*. *P l*) (*permute-list* (*Transposition.transpose i j*))›
for *L* :: ‹'*b list*›

⟨*proof*⟩

Finally, the proof of the general version relies on the fact that a permutation can be written as finite product of transpositions.

theorem $MultiSync_{ptick}$*-permute-list* :
‹$[\![S]\!]_{\checkmark}$ l ∈@ *permute-list f L*. P l =
RenamingTick ($[\![S]\!]_{\checkmark}$ l ∈@ L. P l) (*permute-list f*)›
if *f-permutes* : ‹*f permutes* {..<*length L*}›
⟨*proof*⟩

Chapter 12

Events and Ticks

12.1 Preliminaries

lemma *strict-events-of-memE-optimized-tickFree* :
‹($\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow t \notin \mathcal{D}\ P \Longrightarrow ev\ a \in set\ t \Longrightarrow tF\ t \Longrightarrow thesis$) $\Longrightarrow$ *thesis*› **if** ‹$a \in \boldsymbol{\alpha}(P)$›
⟨*proof*⟩

lemma *events-of-memE-optimized-tickFree* :
‹($\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow ev\ a \in set\ t \Longrightarrow tF\ t \Longrightarrow thesis$) $\Longrightarrow$ *thesis*› **if** ‹$a \in \alpha(P)$›
⟨*proof*⟩

12.2 Sequential Composition

12.2.1 Events

lemma *events-of-Seq$_{ptick}$* : ‹$\alpha(P\ ;_{\checkmark}\ Q) = \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$›
⟨*proof*⟩
lemma *events-of-Seq$_{ptick}$-subset* : ‹$\alpha(P\ ;_{\checkmark}\ Q) \subseteq \alpha(P) \cup (\bigcup r.\ \alpha(Q\ r))$›
⟨*proof*⟩
corollary *events-of-Seq-subset* : ‹$\alpha(P\ ;\ Q) \subseteq \alpha(P) \cup \alpha(Q)$›
⟨*proof*⟩

lemma *strict-events-of-Seq$_{ptick}$-subset* : ‹$\boldsymbol{\alpha}(P\ ;_{\checkmark}\ Q) \subseteq \boldsymbol{\alpha}(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\boldsymbol{\alpha}(Q\ r))$›
⟨*proof*⟩

12.2.2 Ticks

lemma *ticks-of-Seq$_{ptick}$* :
‹$\checkmark s(P\ ;_{\checkmark}\ Q) = (if\ \mathcal{D}\ P = \{\}\ then\ (\bigcup r \in \checkmark\mathbf{s}(P).\ \checkmark s(Q\ r))\ else\ UNIV)$›
⟨*proof*⟩

lemma ‹$\checkmark\mathbf{s}(P\ ;_{\checkmark}\ Q) \subseteq \bigcup\ \{\checkmark\mathbf{s}(Q\ r)\ |r.\ r \in \checkmark\mathbf{s}(P)\}$›
— Already proven earlier in the construction.
⟨*proof*⟩

12.3 Synchronization Product

12.3.1 Events

lemma (**in** *Sync*$_{ptick}$-*locale*) *events-of-Sync*$_{ptick}$-*subset* : ‹$\alpha(P\ [\![S]\!]_{\checkmark}\ Q) \subseteq \alpha(P) \cup \alpha(Q)$›
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$-*locale*) *events-of-Inter*$_{ptick}$: ‹$\alpha(P\ |||_{\checkmark}\ Q) = \alpha(P) \cup \alpha(Q)$›
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$-*locale*) *strict-events-of-Sync*$_{ptick}$-*subset* :
‹$\boldsymbol{\alpha}(P\ [\![S]\!]_{\checkmark}\ Q) \subseteq \boldsymbol{\alpha}(P) \cup \boldsymbol{\alpha}(Q)$›
⟨*proof*⟩

12.3.2 Ticks

lemma (**in** *Sync*$_{ptick}$-*locale*)
‹$\checkmark\mathbf{s}(P\ [\![S]\!]_{\checkmark}\ Q) \subseteq \{r\text{-}s\ |r\text{-}s\ r\ s.\ r \otimes\checkmark\ s = \mathit{Some}\ r\text{-}s \wedge r \in \checkmark\mathbf{s}(P) \wedge s \in \checkmark\mathbf{s}(Q)\}$›
— Already proven earlier in the construction.
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$-*locale*) *ticks-of-no-div-Sync*$_{ptick}$-*subset* :
‹$\mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q) = \{\} \Longrightarrow$
$\checkmark s(P\ [\![S]\!]_{\checkmark}\ Q) \subseteq \{r\text{-}s\ |r\text{-}s\ r\ s.\ \mathit{tick\text{-}join}\ r\ s = \mathit{Some}\ r\text{-}s \wedge r \in \checkmark s(P) \wedge s \in \checkmark s(Q)\}$›
⟨*proof*⟩

12.4 Architectural Operators

12.4.1 Events

lemma *events-of-MultiSeq-subset* :

‹$\alpha(\mathit{SEQ}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \alpha(P\ l))$›
⟨*proof*⟩

lemma *events-of-MultiSeq*$_{ptick}$-*subset* :
‹$\alpha((\mathit{SEQ}_{\checkmark}\ l \in@\ L.\ P\ l)\ r) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \alpha(P\ l\ r))$›
⟨*proof*⟩

lemma *strict-events-of-MultiSeq-subset* :

‹$\boldsymbol{\alpha}(\mathit{SEQ}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \boldsymbol{\alpha}(P\ l))$›

⟨*proof*⟩

lemma *strict-events-of-MultiSeq*$_{ptick}$*-subset* :
‹$\boldsymbol{\alpha}((SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r) \subseteq (\bigcup l \in set\ L.\ \bigcup r.\ \boldsymbol{\alpha}(P\ l\ r))$›
⟨*proof*⟩

lemma *events-of-MultiSync*$_{ptick}$*-subset* :
‹$\alpha([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in set\ L.\ \alpha(P\ l))$›
⟨*proof*⟩

lemma *events-of-MultiInter*$_{ptick}$:
‹$\alpha(|||_{\checkmark}\ l \in@\ L.\ P\ l) = (\bigcup l \in set\ L.\ \alpha(P\ l))$›
⟨*proof*⟩

lemma *strict-events-of-MultiSync*$_{ptick}$*-subset* :
‹$\boldsymbol{\alpha}([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in set\ L.\ \boldsymbol{\alpha}(P\ l))$›
⟨*proof*⟩

12.4.2 Ticks

We only look at *strict-ticks-of* lemmas: *ticks-of* is harder to deal with because it requires more control on the divergences.

lemma *strict-ticks-of-MultiSeq*$_{ptick}$*-subset* :
‹$\checkmark\mathbf{s}((SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r) \subseteq (if\ L = []\ then\ \{r\}\ else\ (\bigcup r.\ \checkmark\mathbf{s}(P\ (last\ L)\ r)))$›
⟨*proof*⟩

lemma *strict-ticks-of-MultiSeq-subset* :
‹$\checkmark\mathbf{s}(SEQ\ l \in@\ L.\ P\ l) \subseteq (if\ L = []\ then\ \{undefined\}\ else\ (\bigcup r.\ \checkmark\mathbf{s}(P\ (last\ L))))$›
⟨*proof*⟩

lemma *strict-ticks-of-MultiSync*$_{ptick}$*-subset* :
‹$\checkmark\mathbf{s}([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l) \subseteq$
$\{l.\ length\ l = length\ L \wedge (\forall i < length\ L.\ l\ !\ i \in \checkmark\mathbf{s}(P\ (L\ !\ i)))\}$›
⟨*proof*⟩

Chapter 13

Continuity Rules

13.1 Sequential Composition

13.1.1 Monotonicity

lemma *tickFree-mem-min-elems-D* : ‹$t \in$ *min-elems* ($\mathcal{D}$ P) $\Longrightarrow$ *tF* t›
⟨*proof*⟩

lemma *mono-Seq*$_{ptick}$: ‹P ;$_{\checkmark}$ $R \sqsubseteq Q$;$_{\checkmark}$ S› **if** ‹$P \sqsubseteq Q$› **and** ‹$R \sqsubseteq S$›
for P Q :: ‹('a, 'r) *process*$_{ptick}$› **and** R S :: ‹'$r \Rightarrow$ ('a, 's) *process*$_{ptick}$›
⟨*proof*⟩

13.1.2 Preliminaries

context begin

private lemma *chain-Seq*$_{ptick}$*-left*: ‹*chain* $Y \Longrightarrow$ *chain* ($\lambda i.$ Y i ;$_{\checkmark}$ S)›
⟨*proof*⟩ **lemma** *chain-Seq*$_{ptick}$*-right*: ‹*chain* $Y \Longrightarrow$ *chain* ($\lambda i.$ S ;$_{\checkmark}$ Y i)›
⟨*proof*⟩ **lemma** *cont-left-prem-Seq*$_{ptick}$:
‹($\bigsqcup i.$ Y i) ;$_{\checkmark}$ S = ($\bigsqcup i.$ Y i ;$_{\checkmark}$ S)› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*chain* Y›
— We have to add this hypothesis in the generalization.
⟨*proof*⟩

lemma ‹*finite* $R \Longrightarrow$ *chain* $Y \Longrightarrow \sqcap r \in R.$ ($\bigsqcup i.$ Y i r) = ($\bigsqcup i. \sqcap r \in R.$ Y i r)›
⟨*proof*⟩

lemma *infinite-GlobalNdet-not-cont* :
— This is a counter example.
defines *Y-def* : ‹$Y \equiv \lambda i$ r :: *nat*. *if* $r \leq i$ *then* *STOP* *else* $\bot$ :: (*nat*, *nat*) *process*$_{ptick}$›
shows ‹*chain* Y› ‹$\sqcap r \in$ *UNIV*. ($\bigsqcup i.$ Y i r) $\neq$ ($\bigsqcup i. \sqcap r \in$ *UNIV*. Y i r)›
⟨*proof*⟩

The same counter-example works for Seq_{ptick}.

lemma *infinite-Seq$_{ptick}$-not-cont* :
— This is a counter example.
defines *P-def* : ‹$P \equiv SKIPS\ UNIV$ :: $(nat, nat)\ process_{ptick}$›
and *Y-def* : ‹$Y \equiv \lambda i\ r :: nat.\ if\ r \leq i\ then\ STOP\ else\ \bot$ :: $(nat, nat)\ process_{ptick}$›
shows ‹$chain\ Y$› ‹$P\ ;_{\checkmark}\ (\bigsqcup i.\ Y\ i) \neq (\bigsqcup i.\ P\ ;_{\checkmark}\ Y\ i)$›
⟨*proof*⟩

We must therefore find a condition under which Seq_{ptick} is continuous.

private lemma *cont-right-prem-Seq$_{ptick}$* :
‹$S\ ;_{\checkmark}\ (\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ S\ ;_{\checkmark}\ Y\ i)$› (**is** ‹*?lhs* = *?rhs*›) **if** ‹$chain\ Y$› **and** ‹$\mathbb{F}_{\checkmark}(S)$›
— We have to add this hypothesis in the generalization.
⟨*proof*⟩

13.1.3 Continuity

We then spent a lot of time trying to prove the continuity under the assumption of *finite-ticks-fun*.

lemma *Seq$_{ptick}$-cont* [*simp*] : ‹$cont\ (\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
if ‹$cont\ f$› **and** ‹$cont\ g$› **and** ‹$\mathbb{F}_{\checkmark\Rightarrow}(f)$›
for g :: ‹- ⇒ - ⇒ $('a, 's)\ process_{ptick}$›
⟨*proof*⟩

We could therefore only prove the weaker following version.

lemma *Seq$_{ptick}$-cont* [*simp*] : ‹$cont\ (\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
if ‹$cont\ f$› **and** ‹$cont\ g$› **and** ‹$\bigwedge x.\ \mathbb{F}_{\checkmark}(f\ x)$›
for g :: ‹- ⇒ - ⇒ $('a, 's)\ process_{ptick}$›
⟨*proof*⟩

end

corollary ‹$cont\ f \Longrightarrow cont\ g \Longrightarrow cont\ (\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
for f :: ‹$'b :: cpo \Rightarrow ('a, 'r :: finite)\ process_{ptick}$›
⟨*proof*⟩

lemma *MultiSeq$_{ptick}$-cont*[*simp*]:
‹$[\![\bigwedge l.\ l \in set\ L \Longrightarrow cont\ (f\ l);\ \bigwedge l\ r\ x.\ l \in set\ (butlast\ L) \Longrightarrow \mathbb{F}_{\checkmark}(f\ l\ x\ r)]\!]$
$\Longrightarrow cont\ (\lambda x.\ (SEQ_{\checkmark}\ l \in@\ L.\ f\ l\ x)\ r)$›
⟨*proof*⟩

13.2 Synchronization Product

context $Sync_{ptick}$-*locale* **begin**

13.2.1 Monotonicity

lemma *mono-Sync*$_{ptick}$: $\langle P \ [\![A]\!]_{\checkmark} \ Q \sqsubseteq P' \ [\![A]\!]_{\checkmark} \ Q' \rangle$ **if** $\langle P \sqsubseteq P' \rangle$ **and** $\langle Q \sqsubseteq Q' \rangle$
⟨*proof*⟩

13.2.2 Preliminaries

lemma *chain-Sync*$_{ptick}$-*left* : $\langle chain \ Y \Longrightarrow chain \ (\lambda i. \ Y \ i \ [\![A]\!]_{\checkmark} \ Q) \rangle$
and *chain-Sync*$_{ptick}$-*right* : $\langle chain \ Z \Longrightarrow chain \ (\lambda i. \ P \ [\![A]\!]_{\checkmark} \ Z \ i) \rangle$
⟨*proof*⟩

lemma *cont-left-prem-Sync*$_{ptick}$:
$\langle (\bigsqcup i. \ Y \ i) \ [\![A]\!]_{\checkmark} \ Q = (\bigsqcup i. \ Y \ i \ [\![A]\!]_{\checkmark} \ Q) \rangle$ **if** *chain*: $\langle chain \ Y \rangle$
⟨*proof*⟩

lemma (**in** *Sync*$_{ptick}$-*locale*) *cont-right-prem-Sync*$_{ptick}$:
$\langle P \ [\![A]\!]_{\checkmark} \ (\bigsqcup i. \ Z \ i) = (\bigsqcup i. \ P \ [\![A]\!]_{\checkmark} \ Z \ i) \rangle$ **if** $\langle chain \ Z \rangle$
⟨*proof*⟩

13.2.3 Continuity

lemma *Sync*$_{ptick}$-*cont*[*simp*]: $\langle cont \ (\lambda x. \ f \ x \ [\![A]\!]_{\checkmark} \ g \ x) \rangle$ **if** $\langle cont \ f \rangle$ $\langle cont \ g \rangle$
⟨*proof*⟩

end

lemma *MultiSync*$_{ptick}$-*cont* [*simp*] :
$\langle (\bigwedge l. \ l \in set \ L \Longrightarrow cont \ (P \ l)) \Longrightarrow cont \ (\lambda x. \ [\![S]\!]_{\checkmark} \ l \in@ \ L. \ P \ l \ x) \rangle$
⟨*proof*⟩

Chapter 14

Monotonicity Properties

14.0.1 Sequential Composition

lemma *mono-Seq$_{ptick}$-FD* : $\langle P \sqsubseteq_{FD} P' \Longrightarrow (\bigwedge r.\ Q\ r \sqsubseteq_{FD} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{FD} P'\ ;_{\checkmark}\ Q'\rangle$
⟨*proof*⟩

lemma *mono-Seq$_{ptick}$-DT* : $\langle P \sqsubseteq_{DT} P' \Longrightarrow (\bigwedge r.\ Q\ r \sqsubseteq_{DT} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{DT} P'\ ;_{\checkmark}\ Q'\rangle$
⟨*proof*⟩

lemma *mono-Seq$_{ptick}$-F-right* : $\langle(\bigwedge r.\ Q\ r \sqsubseteq_{F} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{F} P\ ;_{\checkmark}\ Q'\rangle$
⟨*proof*⟩

lemma *mono-Seq$_{ptick}$-D-right* : $\langle(\bigwedge r.\ Q\ r \sqsubseteq_{D} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{D} P\ ;_{\checkmark}\ Q'\rangle$
⟨*proof*⟩

lemma *mono-Seq$_{ptick}$-T-right* : $\langle(\bigwedge r.\ Q\ r \sqsubseteq_{T} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{T} P\ ;_{\checkmark}\ Q'\rangle$
⟨*proof*⟩

Left Sequence monotonicity doesn't hold for ($\sqsubseteq_F$), ($\sqsubseteq_D$) and ($\sqsubseteq_T$).

lemmas *monos-Seq$_{ptick}$* = *mono-Seq$_{ptick}$ mono-Seq$_{ptick}$-FD mono-Seq$_{ptick}$-DT mono-Seq$_{ptick}$-F-right mono-Seq$_{ptick}$-D-right mono-Seq$_{ptick}$-T-right*

14.0.2 Multiple Sequential Composition

lemma *mono-MultiSeq$_{ptick}$* :
$\langle(\bigwedge x\ r.\ x \in set\ L \Longrightarrow P\ x\ r \sqsubseteq Q\ x\ r) \Longrightarrow$
$(SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r \sqsubseteq (SEQ_{\checkmark}\ l \in@\ L.\ Q\ l)\ r\rangle$
⟨*proof*⟩

lemma *mono-MultiSeq$_{ptick}$-FD* :
$\langle(\bigwedge x\ r.\ x \in set\ L \Longrightarrow P\ x\ r \sqsubseteq_{FD} Q\ x\ r) \Longrightarrow$
$(SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r \sqsubseteq_{FD} (SEQ_{\checkmark}\ l \in@\ L.\ Q\ l)\ r\rangle$
and *mono-MultiSeq$_{ptick}$-DT* :

$\langle(\bigwedge x\ r.\ x \in set\ L \Longrightarrow P\ x\ r \sqsubseteq_{DT} Q\ x\ r) \Longrightarrow$
$(SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r \sqsubseteq_{DT} (SEQ_{\checkmark}\ l \in@\ L.\ Q\ l)\ r\rangle$
⟨*proof*⟩

lemmas *monos-MultiSeq*$_{ptick}$ =
mono-MultiSeq$_{ptick}$ *mono-MultiSeq*$_{ptick}$*-FD mono-MultiSeq*$_{ptick}$*-FD*

14.0.3 Synchronization Product

context *Sync*$_{ptick}$*-locale* **begin**

lemma *mono-Sync*$_{ptick}$*-DT* :
$\langle P \sqsubseteq_{DT} P' \Longrightarrow Q \sqsubseteq_{DT} Q' \Longrightarrow P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_{DT} P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$
⟨*proof*⟩

lemma *mono-Sync*$_{ptick}$*-FD* : $\langle P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_{FD} P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$
if $\langle P \sqsubseteq_{FD} P'\rangle$ **and** $\langle Q \sqsubseteq_{FD} Q'\rangle$
⟨*proof*⟩

lemmas *monos-Sync*$_{ptick}$ = *mono-Sync*$_{ptick}$ *mono-Sync*$_{ptick}$*-FD mono-Sync*$_{ptick}$*-DT*

end

14.0.4 Multiple Synchronization Product

lemma *mono-MultiSync*$_{ptick}$:
$\langle(\bigwedge l.\ l \in set\ L \Longrightarrow P\ l \sqsubseteq Q\ l) \Longrightarrow [\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l \sqsubseteq [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l\rangle$
⟨*proof*⟩

lemma *mono-MultiSync*$_{ptick}$*-FD* :
$\langle(\bigwedge l.\ l \in set\ L \Longrightarrow P\ l \sqsubseteq_{FD} Q\ l) \Longrightarrow [\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l \sqsubseteq_{FD} [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l\rangle$
⟨*proof*⟩

lemma *mono-MultiSync*$_{ptick}$*-DT* :
$\langle(\bigwedge l.\ l \in set\ L \Longrightarrow P\ l \sqsubseteq_{DT} Q\ l) \Longrightarrow [\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l \sqsubseteq_{DT} [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l\rangle$
⟨*proof*⟩

lemmas *monos-MultiSync*$_{ptick}$ =
mono-MultiSync$_{ptick}$ *mono-MultiSync*$_{ptick}$*-FD mono-MultiSync*$_{ptick}$*-DT*

Chapter 15

Non Destructiveness Rules

⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

15.1 Synchronization Product

15.1.1 Refinement

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

lemma (**in** *Sync*$_{ptick}$*-locale*) *restriction-process*$_{ptick}$*-Sync*$_{ptick}$*-FD* :
  ‹*P* $[\![A]\!]_{✓}$ *Q* ↓ *n* $\sqsubseteq_{FD}$ (*P* ↓ *n*) $[\![A]\!]_{✓}$ (*Q* ↓ *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 (**in** *Sync*$_{ptick}$*-locale*) *strict-events-of-subset-restriction-process*$_{ptick}$*-Sync*$_{ptick}$
:
  ‹*P* $[\![A]\!]_{✓}$ *Q* ↓ *n* = (*P* ↓ *n*) $[\![A]\!]_{✓}$ (*Q* ↓ *n*)› (**is** ‹*?lhs* = *?rhs*›)
  if ‹***α***(*P*) ⊆ *A* ∨ ***α***(*Q*) ⊆ *A*›
⟨*proof*⟩

corollary *restriction-process*$_{ptick}$*-MultiSync*$_{ptick}$*-FD* :
  ‹$[\![A]\!]_{✓}$ *l* ∈@ *L*. *P l* ↓ *n* $\sqsubseteq_{FD}$ $[\![A]\!]_{✓}$ *l* ∈@ *L*. (*P l* ↓ *n*)›
⟨*proof*⟩

The generalization of the lemma $\boldsymbol{\alpha}(P) \subseteq A \vee \boldsymbol{\alpha}(Q) \subseteq A \Longrightarrow P \llbracket A \rrbracket_{\checkmark} Q \downarrow n = (P \downarrow n) \llbracket A \rrbracket_{\checkmark} (Q \downarrow n)$ is not straightforward. We can already observe with only three processes that one can not expect the first synchronization to have its strict alphabets contained in the synchronization set. Therefore, we have to assume the condition on at least *length L − 1* processes.

corollary *strict-events-of-subset-restriction-process*$_{ptick}$*-MultiSync*$_{ptick}$:
‹$\llbracket A \rrbracket_{\checkmark}$ $l \in@$ L. $P\ l \downarrow n = (if\ n = 0\ then \perp else \llbracket A \rrbracket_{\checkmark}\ l \in@\ L.\ (P\ l \downarrow n))$›
— *if n = 0 then ⊥ else* - is necessary because we can have $L = []$.
if ‹$\bigwedge l.\ l \in set\ (tl\ L) \Longrightarrow \boldsymbol{\alpha}(P\ l) \subseteq A$›
⟨*proof*⟩

corollary (**in** *Sync*$_{ptick}$*-locale*) *restriction-process*$_{ptick}$*-Par*$_{ptick}$:
‹$P \mid\mid_{\checkmark} Q \downarrow n = (P \downarrow n) \mid\mid_{\checkmark} (Q \downarrow n)$›
⟨*proof*⟩

corollary *restriction-process*$_{ptick}$*-MultiPar*$_{ptick}$:
‹$\mid\mid_{\checkmark} l \in@ L.\ P\ l \downarrow n = (if\ n = 0\ then \perp else \mid\mid_{\checkmark} l \in@ L.\ (P\ l \downarrow n))$›
⟨*proof*⟩

15.1.2 Non Destructiveness

lemma (**in** *Sync*$_{ptick}$*-locale*) *Sync*$_{ptick}$*-non-destructive* :
‹*non-destructive* $(\lambda(P,\ Q).\ P \llbracket A \rrbracket_{\checkmark} Q)$›
⟨*proof*⟩

15.1.3 Setup

lemma (**in** *Sync*$_{ptick}$*-locale*) *Sync*$_{ptick}$*-restriction-shift-process*$_{ptick}$
[*restriction-shift-process*$_{ptick}$*-simpset*, *simp*] :
‹*non-destructive f* $\Longrightarrow$ *non-destructive g* $\Longrightarrow$ *non-destructive* $(\lambda x.\ f\ x \llbracket S \rrbracket_{\checkmark}\ g\ x)$›
‹*constructive f* $\Longrightarrow$ *constructive g* $\Longrightarrow$ *constructive* $(\lambda x.\ f\ x \llbracket S \rrbracket_{\checkmark}\ g\ x)$›
⟨*proof*⟩

lemma *MultiSync*$_{ptick}$*-restriction-shift-process*$_{ptick}$
[*restriction-shift-process*$_{ptick}$*-simpset*, *simp*] :
‹$(\bigwedge l.\ l \in set\ L \Longrightarrow$ *non-destructive* $(f\ l)) \Longrightarrow$ *non-destructive* $(\lambda x.\ \llbracket S \rrbracket_{\checkmark}\ l \in@\ L.\ f\ l\ x)$›
‹$(\bigwedge l.\ l \in set\ L \Longrightarrow$ *constructive* $(f\ l)) \Longrightarrow$ *constructive* $(\lambda x.\ \llbracket S \rrbracket_{\checkmark}\ l \in@\ L.\ f\ l\ x)$›
⟨*proof*⟩

corollary *MultiSync*$_{ptick}$*-non-destructive* : ‹*non-destructive* $(\lambda P.\ \llbracket S \rrbracket_{\checkmark}\ l \in@\ L.\ P\ l)$›
⟨*proof*⟩

Chapter 16

Other Laws

declare [[*metis-instantiate*]]

16.1 Laws of Renaming

16.1.1 Renaming and Sequential Composition

lemma *FD-Renaming-Seq*$_{ptick}$:
‹*Renaming P f g* ;$_{✓}$ (λ*g-r*. ⊓$r \in \{r \in$ ✓**s**(P). *g-r* = *g r*$\}$. *Renaming* (*Q r*) *f g*′)
⊑$_{FD}$ *Renaming* (*P* ;$_{✓}$ *Q*) *f g*′› (**is** ‹*?lhs* ⊑$_{FD}$ *?rhs*›)
⟨*proof*⟩

lemma *inj-on-Renaming-Seq*$_{ptick}$:
‹*Renaming* (*P* ;$_{✓}$ *Q*) *f g*′ =
Renaming P f g ;$_{✓}$ (λ*g-r*. *Renaming* (*Q* (*THE r*. *r* ∈ ✓**s**$(P) \wedge$ *g-r* = *g r*)) *f g*′)›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*inj-on g* ✓**s**(P)›
— This assumption is necessary, otherwise we cannot know which tick triggered *Q*.
⟨*proof*⟩

When ′*r* is set on *unit*, we recover the version that we had before the generalization.

lemma ‹*Renaming* (*P* ;$_{✓}$ *Q*) *f g* = *Renaming P f g* ;$_{✓}$ (λ*r*. *Renaming* (*Q* ()) *f g*)›
⟨*proof*⟩

lemma *TickSwap-Seq*$_{ptick}$ [*simp*] :
‹*TickSwap* (*P* ;$_{✓}$ *Q*) = *TickSwap P* ;$_{✓}$ (λ(*s*, *r*). *TickSwap* (*Q* (*r*, *s*)))› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *TickSwap-is-Seq$_{ptick}$-iff* [*simp*] :
‹*TickSwap P = Q* ;$_{\checkmark}$ *R* ⟷ *P = TickSwap Q* ;$_{\checkmark}$ (λ(*r*, *s*). *TickSwap* (*R* (*s*, *r*)))›
⟨*proof*⟩

16.1.2 Renaming and Synchronization Product

theorem (**in** *Sync$_{ptick}$-locale*) *inj-RenamingEv-Sync$_{ptick}$* :
‹*RenamingEv* (*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *f* = *RenamingEv P f* ⟦*f* ‘ *S*⟧$_{\checkmark}$ *RenamingEv Q f*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*inj f*›
⟨*proof*⟩

16.2 Laws of Hiding

16.3 Hiding and Sequential Composition

We start by giving a counter example when the assumption $\mathbb{F}_{\checkmark}(P)$ is not satisfied.

notepad begin
⟨*proof*⟩

end

In general, only one refinement is holding.

theorem *Hiding-Seq-FD-Seq-Hiding* :
‹(*P* ;$_{\checkmark}$ *Q*) \ *S* ⊑$_{FD}$ (*P* \ *S*) ;$_{\checkmark}$ (λ*r*. *Q r* \ *S*)› (**is** ‹*?lhs* ⊑$_{FD}$ *?rhs*›)
⟨*proof*⟩

16.4 Hiding and Synchronization Product

lemma *setinterleaves$_{ptick}$-imp-superset-ev* :
‹*t setinterleaves*$_{\checkmark}$*$_{tick\text{-}join}$* ((*u*, *v*), *A*) ⟹
{*ev a* |*a*. *ev a* ∈ *set u*} ∪ {*ev a* |*a*. *ev a* ∈ *set v*} ⊆ {*ev a* |*a*. *ev a* ∈ *set t*}›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-locale*) *disjoint-isInfHidden-seqRunL-to-Sync$_{ptick}$* :
assumes ‹*A* ∩ *S* = {}› **and** ‹*isInfHidden-seqRun x P A t-P*›
and ‹*t-Q* ∈ $\mathcal{T}$ *Q*› **and** ‹*t setinterleaves*$_{\checkmark}$*$_{(\otimes\checkmark)}$* ((*t-P*, *t-Q*), *S*)›
shows ‹*isInfHidden-seqRun* (*ev* ∘ *of-ev* ∘ *x*) (*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *A t*›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-locale*) *disjoint-isInfHidden-seqRunR-to-Sync$_{ptick}$* :
‹⟦*A* ∩ *S* = {}; *isInfHidden-seqRun x Q A t-Q*; *t-P* ∈ $\mathcal{T}$ *P*;
t setinterleaves$_{\checkmark}$*$_{(\otimes\checkmark)}$* ((*t-P*, *t-Q*), *S*)⟧ ⟹
isInfHidden-seqRun (*ev* ∘ *of-ev* ∘ *x*) (*P* ⟦*S*⟧$_{\checkmark}$ *Q*) *A t*›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*locale*) *disjoint-Hiding-Sync*$_{ptick}$*-FD-Sync*$_{ptick}$*-Hiding-aux* :
— This lemma avoids duplication of the proof work.
assumes ‹$A \cap S = \{\}$› ‹*tF u*› ‹*ftF v*› ‹*t-P* $\in \mathcal{D}$ $(P \setminus A)$› ‹*t-Q* $\in \mathcal{T}$ $(Q \setminus A)$›
and $*$: ‹*u setinterleaves*$_{✓(\otimes✓)}$ $((t\text{-}P,\ t\text{-}Q),\ S)$›
shows ‹u @ $v \in \mathcal{D}$ $(P\ [\![S]\!]_{✓}\ Q \setminus A)$›
⟨*proof*⟩

theorem (**in** $Sync_{ptick}$-*locale*) *disjoint-Hiding-Sync*$_{ptick}$*-FD-Sync*$_{ptick}$*-Hiding* :
‹$P\ [\![S]\!]_{✓}\ Q \setminus A \sqsubseteq_{FD} (P \setminus A)\ [\![S]\!]_{✓}\ (Q \setminus A)$› **if** ‹$A \cap S = \{\}$›
⟨*proof*⟩

theorem (**in** $Sync_{ptick}$-*locale*) *disjoint-finite-Hiding-Sync*$_{ptick}$:
‹$P\ [\![S]\!]_{✓}\ Q \setminus A = (P \setminus A)\ [\![S]\!]_{✓}\ (Q \setminus A)$› **if** ‹$A \cap S = \{\}$› **and** ‹*finite A*›
— Monster theorem!
⟨*proof*⟩

lemma *disjoint-Hiding-MultiSync*$_{ptick}$*-FD-MultiSync*$_{ptick}$*-Hiding* :
‹$[\![S]\!]_{✓}\ l \in@\ L.\ P\ l \setminus A \sqsubseteq_{FD} [\![S]\!]_{✓}\ l \in@\ L.\ (P\ l \setminus A)$› **if** ‹$A \cap S = \{\}$›
⟨*proof*⟩

lemma *disjoint-finite-Hiding-MultiSync*$_{ptick}$:
‹$[\![S]\!]_{✓}\ l \in@\ L.\ P\ l \setminus A = [\![S]\!]_{✓}\ l \in@\ L.\ (P\ l \setminus A)$› **if** ‹$A \cap S = \{\}$› **and** ‹*finite A*›
⟨*proof*⟩

16.5 Other Laws of Synchronization Product

16.5.1 Synchronization Set can be restricted

lemma *setinterleaves*$_{ptick}$*-is-restrictable-on-superset-events-of* :
‹$\{a.\ ev\ a \in set\ u \vee ev\ a \in set\ v\} \subseteq A \Longrightarrow$
t *setinterleaves*$_{✓\,tick\text{-}join}$ $((u,\ v),\ S) \longleftrightarrow$
t *setinterleaves*$_{✓\,tick\text{-}join}$ $((u,\ v),\ S \cap A)$›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*locale*) *Sync*$_{ptick}$*-is-restrictable-on-events-of* :
‹$P\ [\![S]\!]_{✓}\ Q = P\ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{✓}\ Q$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *Sync$_{ptick}$-is-restrictable-on-superset-events-of* :
‹$P\ [\![S]\!]_{\checkmark}\ Q = P\ [\![S \cap A]\!]_{\checkmark}\ Q$› **if** ‹$\alpha(P) \cup \alpha(Q) \subseteq A$›
⟨*proof*⟩

lemma ‹*tF* $t \Longrightarrow \{a.\ ev\ a \in set\ u\} \cap S = \{\} \Longrightarrow a \in S \Longrightarrow$
$\neg\ t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ ev\ a\ \#\ v),\ S)$›
⟨*proof*⟩

16.5.2 Some Refinements

context *Sync$_{ptick}$-locale* **begin**

lemma *Mndetprefix-Sync$_{ptick}$-Det-distr-FD* :
‹$(\sqcap\ a \in A \rightarrow (P\ a\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \rightarrow Q\ b))) \ \Box$
$(\sqcap\ b \in B \rightarrow ((\sqcap\ a \in A \rightarrow P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ Q\ b))$
$\sqsubseteq_{FD}\ (\sqcap\ a \in A \rightarrow P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \rightarrow Q\ b)$›
(**is** ‹*?lhs1* $\Box$ *?lhs2* $\sqsubseteq_{FD}$ *?rhs*›)
if ‹$A \neq \{\}$› ‹$B \neq \{\}$› ‹$A \cap C = \{\}$› ‹$B \cap C = \{\}$›
⟨*proof*⟩

lemmas *Mndetprefix-Sync$_{ptick}$-Det-distr-F* =
Mndetprefix-Sync$_{ptick}$-Det-distr-FD[*THEN leFD-imp-leF*]

lemmas *Mndetprefix-Sync$_{ptick}$-Det-distr-D* =
Mndetprefix-Sync$_{ptick}$-Det-distr-FD[*THEN leFD-imp-leD*]

lemmas *Mndetprefix-Sync$_{ptick}$-Det-distr-T* =
Mndetprefix-Sync$_{ptick}$-Det-distr-F[*THEN leF-imp-leT*]

lemma *Mndetprefix-Sync$_{ptick}$-Det-distr-DT* :
‹$[\![A \neq \{\};\ B \neq \{\};\ A \cap C = \{\};\ B \cap C = \{\}]\!] \Longrightarrow$
$(\sqcap\ a \in A \rightarrow (P\ a\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \rightarrow Q\ b))) \ \Box$
$(\sqcap\ b \in B \rightarrow ((\sqcap\ a \in A \rightarrow P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ Q\ b))$
$\sqsubseteq_{DT}\ (\sqcap\ a \in A \rightarrow P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \rightarrow Q\ b)$›
⟨*proof*⟩

end

Chapter 17

Deadlock Results

17.1 First Results

17.1.1 Non Terminating

Keep in mind *lifelock-free*$_{SKIPS}$ $P = (\mathcal{D}\ P = \{\})$.

Sequential Composition

lemma ‹*non-terminating* $P \Longrightarrow P\ ;_{\checkmark}\ Q =$ *RenamingTick* $P\ g$›
— Already proven earlier.
⟨*proof*⟩

Synchronization Product

lemma (**in** *Sync*$_{ptick}$-*locale*) *non-terminating-Sync*$_{ptick}$:
‹*non-terminating* $P \Longrightarrow$ *lifelock-free*$_{SKIPS}$ $Q \Longrightarrow$ *non-terminating* $(P\ [\![A]\!]_{\checkmark}\ Q)$›
‹*lifelock-free*$_{SKIPS}$ $P \Longrightarrow$ *non-terminating* $Q \Longrightarrow$ *non-terminating* $(P\ [\![A]\!]_{\checkmark}\ Q)$›
⟨*proof*⟩

17.1.2 Deadlock Free

Sequential Composition

lemma ‹*deadlock-free* $P \Longrightarrow$ *deadlock-free* $(P\ ;_{\checkmark}\ Q)$›
⟨*proof*⟩

The next lemma is of course more interesting.

lemma *deadlock-free*$_{SKIPS}$-*Seq*$_{ptick}$:
‹*deadlock-free*$_{SKIPS}$ $(P\ ;_{\checkmark}\ Q)$›
if *df-assms* : ‹*deadlock-free*$_{SKIPS}$ P› ‹$\bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow$ *deadlock-free*$_{SKIPS}$ $(Q\ r)$›
⟨*proof*⟩

corollary *deadlock-free-Seq$_{ptick}$* :
‹⟦*deadlock-free$_{SKIPS}$ P*; ⋀*r*. *r* ∈ ✓**s**(*P*) ⟹ *deadlock-free* (*Q r*)⟧
⟹ *deadlock-free* (*P* ;✓ *Q*)›
⟨*proof*⟩

Synchronization Product

context *Sync$_{ptick}$-locale* **begin**

lemma *deadlock-free-Det-bis* :
‹*P* = *STOP* ∧ *Q* ≠ *STOP* ∨ *deadlock-free P* ⟹
Q = *STOP* ∧ *P* ≠ *STOP* ∨ *deadlock-free Q* ⟹ *deadlock-free* (*P* □ *Q*)›
⟨*proof*⟩

lemma *deadlock-free-Mprefix-Sync$_{ptick}$-Mprefix* :
assumes *not-all-empty*: ‹¬ *A* ⊆ *S* ∨ ¬ *B* ⊆ *S* ∨ *A* ∩ *B* ∩ *S* ≠ {}›
and ‹⋀*a*. *a* ∈ *A* − *S* ⟹ *deadlock-free* (*P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*)›
and ‹⋀*b*. *b* ∈ *B* − *S* ⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)›
and ‹⋀*x*. *x* ∈ *A* ∩ *B* ∩ *S* ⟹ *deadlock-free* (*P x* ⟦*S*⟧✓ *Q x*)›
shows ‹*deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ □*b* ∈ *B* → *Q b*)›
⟨*proof*⟩

lemma *deadlock-free-Mprefix-Sync$_{ptick}$-Mprefix-subset* :
‹⟦*A* ⊆ *S*; *B* ⊆ *S*; *A* ∩ *B* ≠ {};
⋀*x*. *x* ∈ *A* ∩ *B* ∩ *S* ⟹ *deadlock-free* (*P x* ⟦*S*⟧✓ *Q x*)⟧
⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*)›
and *deadlock-free-Mprefix-Sync$_{ptick}$-Mprefix-indep* :
‹⟦*A* ∩ *S* = {}; *B* ∩ *S* = {}; *A* ≠ {} ∨ *B* ≠ {};
⋀*a*. *a* ∈ *A* − *S* ⟹ *deadlock-free* (*P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*);
⋀*b*. *b* ∈ *B* − *S* ⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)⟧
⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*)›
and *deadlock-free-Mprefix-Sync$_{ptick}$-Mprefix-right* :
‹⟦*A* ⊆ *S*; *B* ∩ *S* = {}; *B* ≠ {};
⋀*b*. *b* ∈ *B* − *S* ⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)⟧
⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*)›
and *deadlock-free-Mprefix-Sync$_{ptick}$-Mprefix-left* :
‹⟦*A* ∩ *S* = {}; *B* ⊆ *S*; *A* ≠ {};
⋀*a*. *a* ∈ *A* − *S* ⟹ *deadlock-free* (*P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*)⟧
⟹ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧✓ □*b*∈*B* → *Q b*)›
⟨*proof*⟩

end

17.2 Renaming and reference Processes

lemma *DF-empty* [*simp*] : ‹*DF* {} = *STOP*›
and *DF$_{SKIPS}$-empty* [*simp*] : ‹*DF$_{SKIPS}$* {} {} = *STOP*›

and *RUN-empty* [*simp*] : ‹$RUN\ \{\} = STOP$›
and *CHAOS-empty* [*simp*] : ‹$CHAOS\ \{\} = STOP$›
and *$CHAOS_{SKIPS}$-empty* [*simp*] : ‹$CHAOS_{SKIPS}\ \{\}\ \{\} = STOP$›
⟨*proof*⟩

17.2.1 Alternative Definitions with restriction fixed-point Operator

For now, we have lemmas such as $DF\ (f\ `\ A) \sqsubseteq_{FD} Renaming\ (DF\ A)\ f\ g$, but the other refinement is requiring *finitary* assumptions ($[\![$*finitary* $f$; *finitary* $g]\!] \Longrightarrow Renaming\ (DF\ A)\ f\ g \sqsubseteq_{FD} DF\ (f\ `\ A)$).

lemma *DF-restriction-fix-def* : ‹$DF\ A = (\upsilon\ X.\ \sqcap a \in A \rightarrow X)$›
⟨*proof*⟩

lemma *DF_{SKIPS}-restriction-fix-def* : ‹$DF_{SKIPS}\ A\ R = (\upsilon\ X.\ (\sqcap a \in A \rightarrow X) \sqcap SKIPS\ R)$›
⟨*proof*⟩

lemma *RUN-restriction-fix-def* : ‹$RUN\ A = (\upsilon\ X.\ \Box a \in A \rightarrow X)$›
⟨*proof*⟩

lemma *CHAOS-restriction-fix-def* : ‹$CHAOS\ A = (\upsilon\ X.\ STOP \sqcap (\Box a \in A \rightarrow X))$›
⟨*proof*⟩

lemma *$CHAOS_{SKIPS}$-restriction-fix-def* : ‹$CHAOS_{SKIPS}\ A\ R = (\upsilon\ X.\ SKIPS\ R \sqcap STOP \sqcap (\Box a \in A \rightarrow X))$›
⟨*proof*⟩

17.2.2 Stronger Results

With *restriction-fix* induction, removing these assumptions is trivial.

lemma *Renaming-DF* : ‹$Renaming\ (DF\ A)\ f\ g = DF\ (f\ `\ A)$›
⟨*proof*⟩

lemma *Renaming-DF_{SKIPS}* : ‹$Renaming\ (DF_{SKIPS}\ A\ R)\ f\ g = DF_{SKIPS}\ (f\ `\ A)\ (g\ `\ R)$›
⟨*proof*⟩

lemma *Renaming-RUN* : ‹$Renaming\ (RUN\ A)\ f\ g = RUN\ (f\ `\ A)$›
⟨*proof*⟩

lemma *Renaming-CHAOS* : ‹$Renaming\ (CHAOS\ A)\ f\ g = CHAOS\ (f\ `\ A)$›
⟨*proof*⟩

lemma *Renaming-$CHAOS_{SKIPS}$* : ‹$Renaming\ (CHAOS_{SKIPS}\ A\ R)\ f\ g = CHAOS_{SKIPS}\ (f\ `\ A)\ (g\ `\ R)$›
⟨*proof*⟩

17.3 Data Independence

When working with the new interleaving $P \ [\![\{\}]\!]_{\checkmark} \ Q$, we intuitively expect it to be *deadlock-free* when both P and Q are. The purpose of this section is to prove it.

17.3.1 An interesting equivalence

lemma (**in** $Sync_{ptick}$*-locale*) *deadlock-free-of-*$Sync_{ptick}$*-iff-DF-FD-DF-*$Sync_{ptick}$*-DF*:
‹$(\forall P\ Q.\ \textit{deadlock-free}\ P \longrightarrow \textit{deadlock-free}\ Q \longrightarrow \textit{deadlock-free}\ (P\ [\![S]\!]_{\checkmark}\ Q))$
$\longleftrightarrow \textit{DF UNIV} \sqsubseteq_{FD} (\textit{DF UNIV}\ [\![S]\!]_{\checkmark}\ \textit{DF UNIV})$› (**is** ‹*?lhs* $\longleftrightarrow$ *?rhs*›)
⟨*proof*⟩

17.3.2 *STOP* and *SKIP* synchronized with *DF A*

The two results below form a stronger (and generalized) version of $r = s \Longrightarrow (\textit{DF}\ A \sqsubseteq_{FD} \textit{DF}\ A\ [\![S]\!]\ \textit{SKIP}\ r) = (A \cap S = \{\})$.

context $Sync_{ptick}$*-locale* **begin**

lemma (**in** $Sync_{ptick}$*-locale*) *DF-FD-DF-*$Sync_{ptick}$*-SKIPS-imp-disjoint* :
‹$A \cap S = \{\}$› **if** ‹$\textit{DF}\ A \sqsubseteq_{FD} \textit{DF}\ A\ [\![S]\!]_{\checkmark}\ \textit{SKIPS}\ R$›
⟨*proof*⟩

lemma *disjoint-imp-DF-eq-DF-*$Sync_{ptick}$*-SKIPS* :
‹$\textit{DF}\ A = \textit{DF}\ A\ [\![S]\!]_{\checkmark}\ \textit{SKIPS}\ R$› **if** ‹$A \cap S = \{\}$›
⟨*proof*⟩

corollary *DF-FD-DF-*$Sync_{ptick}$*-STOP-imp-disjoint* :
‹$\textit{DF}\ A \sqsubseteq_{FD} \textit{DF}\ A\ [\![S]\!]_{\checkmark}\ \textit{STOP} \Longrightarrow A \cap S = \{\}$›
and *DF-FD-DF-*$Sync_{ptick}$*-SKIP-imp-disjoint* :
‹$\textit{DF}\ A \sqsubseteq_{FD} \textit{DF}\ A\ [\![S]\!]_{\checkmark}\ \textit{SKIP}\ r \Longrightarrow A \cap S = \{\}$›
and *disjoint-imp-DF-eq-DF-*$Sync_{ptick}$*-STOP* :
‹$A \cap S = \{\} \Longrightarrow \textit{DF}\ A = \textit{DF}\ A\ [\![S]\!]_{\checkmark}\ \textit{STOP}$›
and *disjoint-imp-DF-eq-DF-*$Sync_{ptick}$*-SKIP* :
‹$A \cap S = \{\} \Longrightarrow \textit{DF}\ A = \textit{DF}\ A\ [\![S]\!]_{\checkmark}\ \textit{SKIP}\ r$›
⟨*proof*⟩

end

corollary (**in** $Sync_{ptick}$*-locale*) *DF-FD-SKIPS-*$Sync_{ptick}$*-DF-imp-disjoint* :
‹$\textit{DF}\ A \sqsubseteq_{FD} \textit{SKIPS}\ R\ [\![S]\!]_{\checkmark}\ \textit{DF}\ A \Longrightarrow A \cap S = \{\}$›
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$*-locale*) *disjoint-imp-DF-eq-SKIPS-*$Sync_{ptick}$*-DF* :

$\langle A \cap S = \{\} \Longrightarrow DF\ A = SKIPS\ R\ [\![S]\!]_{\checkmark}\ DF\ A\rangle$
⟨*proof*⟩

corollary (**in** $Sync_{ptick}$-*locale*) *DF-FD-STOP-Sync*$_{ptick}$*-DF-imp-disjoint* :
$\langle DF\ A \sqsubseteq_{FD} STOP\ [\![S]\!]_{\checkmark}\ DF\ A \Longrightarrow A \cap S = \{\}\rangle$
and *DF-FD-SKIP-Sync*$_{ptick}$*-DF-imp-disjoint* :
$\langle DF\ A \sqsubseteq_{FD} SKIP\ r\ [\![S]\!]_{\checkmark}\ DF\ A \Longrightarrow A \cap S = \{\}\rangle$
and *disjoint-imp-DF-eq-STOP-Sync*$_{ptick}$*-DF* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = STOP\ [\![S]\!]_{\checkmark}\ DF\ A\rangle$
and *disjoint-imp-DF-eq-SKIP-Sync*$_{ptick}$*-DF* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = SKIP\ r\ [\![S]\!]_{\checkmark}\ DF\ A\rangle$
⟨*proof*⟩

17.3.3 Finally, *deadlock-free* $(P \mid\mid\mid Q)$

theorem (**in** $Sync_{ptick}$-*locale*) *DF-F-DF-Sync*$_{ptick}$*-DF-weak* : $\langle DF\ (A \cup B) \sqsubseteq_F DF\ A\ [\![S]\!]_{\checkmark}\ DF\ B\rangle$
if *nonempty*: $\langle A \neq \{\}\rangle$ $\langle B \neq \{\}\rangle$
and *intersect-hyp*: $\langle B \cap S = \{\} \vee (\exists\, y.\ B \cap S = \{y\} \wedge A \cap S \subseteq \{y\})\rangle$
⟨*proof*⟩

theorem (**in** $Sync_{ptick}$-*locale*) *DF-F-DF-Sync*$_{ptick}$*-DF* :
$\langle DF\ (A \cup B) \sqsubseteq_F DF\ A\ [\![S]\!]_{\checkmark}\ DF\ B\rangle$ **if** $\langle A \neq \{\}\rangle$ $\langle B \neq \{\}\rangle$
and $\langle A \cap S = \{\} \vee (\exists\, a.\ A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \vee$
$B \cap S = \{\} \vee (\exists\, b.\ B \cap S = \{b\} \wedge A \cap S \subseteq \{b\})\rangle$
⟨*proof*⟩

lemma (**in** $Sync_{ptick}$-*locale*) *DF-FD-DF-Sync*$_{ptick}$*-DF* :
$\langle DF\ (A \cup B) \sqsubseteq_{FD} DF\ A\ [\![S]\!]_{\checkmark}\ DF\ B\rangle$ **if** $\langle A \neq \{\}\rangle$ $\langle B \neq \{\}\rangle$
and $\langle A \cap S = \{\} \vee (\exists\, a.\ A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \vee$
$B \cap S = \{\} \vee (\exists\, b.\ B \cap S = \{b\} \wedge A \cap S \subseteq \{b\})\rangle$
⟨*proof*⟩

theorem (**in** $Sync_{ptick}$-*locale*) *DF-FD-DF-Sync*$_{ptick}$*-DF-iff*:
$\langle DF\ (A \cup B) \sqsubseteq_{FD} DF\ A\ [\![S]\!]_{\checkmark}\ DF\ B \longleftrightarrow$
(*if* $A = \{\}$ *then* $B \cap S = \{\}$
else if $B = \{\}$ *then* $A \cap S = \{\}$
else $A \cap S = \{\} \vee (\exists\, a.\ A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \vee$
$B \cap S = \{\} \vee (\exists\, b.\ B \cap S = \{b\} \wedge A \cap S \subseteq \{b\}))\rangle$
(**is** ⟨*?FD-ref* $\longleftrightarrow$ (*if* $A = \{\}$ *then* $B \cap S = \{\}$
else if $B = \{\}$ *then* $A \cap S = \{\}$
else ?cases)⟩)
⟨*proof*⟩

lemma *DF-FD-DF-MultiSync$_{ptick}$-DF* :
‹$[\![\bigwedge l.\ l \in set\ L \Longrightarrow X\ l \neq \{\};\ \exists s.\ (\bigcup\ l \in set\ L.\ X\ l) \cap S \subseteq \{s\}]\!]$
$\Longrightarrow DF\ (\bigcup\ l \in set\ L.\ X\ l) \sqsubseteq_{FD} [\![S]\!]_{\checkmark}\ l \in@\ L.\ (DF\ (X\ l) :: ('a, 'r)\ process_{ptick})$›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-locale*) ‹$DF\ \{a\} = DF\ \{a\}\ [\![S]\!]_{\checkmark}\ STOP \longleftrightarrow a \notin S$›
⟨*proof*⟩

lemma (**in** *Sync$_{ptick}$-locale*) ‹$DF\ \{a\}\ [\![S]\!]_{\checkmark}\ STOP = STOP \longleftrightarrow a \in S$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *DF-FD-DF-Inter$_{ptick}$-DF* : ‹$DF\ A \sqsubseteq_{FD} DF\ A\ |||_{\checkmark}\ DF\ A$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *DF-UNIV-FD-DF-UNIV-Inter$_{ptick}$-DF-UNIV*:
‹$DF\ UNIV \sqsubseteq_{FD} DF\ UNIV\ |||_{\checkmark}\ DF\ UNIV$›
⟨*proof*⟩

corollary (**in** *Sync$_{ptick}$-locale*) *Inter$_{ptick}$-deadlock-free* :
‹*deadlock-free P* $\Longrightarrow$ *deadlock-free Q* $\Longrightarrow$ *deadlock-free* $(P\ |||_{\checkmark}\ Q)$›
⟨*proof*⟩

theorem *MultiInter$_{ptick}$-deadlock-free* :
‹$[\![L \neq []; \bigwedge l.\ l \in set\ L \Longrightarrow deadlock\text{-}free\ (P\ l)]\!] \Longrightarrow$
deadlock-free $(|||_{\checkmark}\ l \in@\ L.\ P\ l)$›
⟨*proof*⟩

Chapter 18

Conclusion

18.1 Main Entry Point

This is where the session `HOL-CSP_PTick` should be imported from.

declare *finite-ticks-simps* [*simp*]
declare *finite-ticks-fun-simps* [*simp*]

unbundle *no option-type-syntax*

18.2 Conclusion

18.2.1 Summary

In this session, we introduced generalized versions of the sequential composition and synchronization operators, thus completing the generalization of `HOL-CSP` (and its extensions) to support parameterized termination. The main motivation was to propagate return values across processes, so that algebraic laws such as those involving *SKIP* continue to hold in a natural way. While the sequential composition adapts relatively smoothly, the synchronization product required a more substantial redesign: the interleaving theory of the classical *Sync* operator could not be reused, and the failures specification had to be carefully adjusted.

Overall, the results confirm that the parameterized setting integrates well with the broader CSP framework. Most classical laws remain valid with only minor modifications, and the new operators exhibit the algebraic and operational properties one expects. The formalization is fairly extensive and provides a solid foundation for further developments of CSP theories with enriched termination behavior.

18.2.2 Sequential Composition

The new version of the sequential composition is of type $('a, 'r)\ process_{ptick} \Rightarrow ('r \Rightarrow ('a, 's)\ process_{ptick}) \Rightarrow ('a, 's)\ process_{ptick}$, so that the process on the right-hand side is now parameterized with the value returned by the process on the left-hand side. The main motivation for this generalization was to have *SKIP* as neutral element. This is now the case.

$$P \mathbin{;\!\!\checkmark} SKIP = P \qquad SKIP\ r \mathbin{;\!\!\checkmark} Q = Q\ r$$

Additionally, with the following associativity property :

$$P \mathbin{;\!\!\checkmark} (\lambda r.\ Q\ r \mathbin{;\!\!\checkmark} R) = P \mathbin{;\!\!\checkmark} Q \mathbin{;\!\!\checkmark} R$$

we can conclude that this generalized sequential composition fulfills the monad laws.
Unsurprisingly, the correspondence with classical version is very intuitive.

$$P \mathbin{;\!\!\checkmark} (\lambda r.\ Q) = P \mathbin{;} Q$$

The expected step law has also been established.

$$\Box a{\in}A \rightarrow P\ a \mathbin{;\!\!\checkmark} Q = \Box a{\in}A \rightarrow (P\ a \mathbin{;\!\!\checkmark} Q)$$

Additionally, in the same way as described in [4], operational laws have been derived.

$$\frac{P\ {}_{\alpha}{\rightsquigarrow}_{\tau}\ P'}{P \mathbin{;\!\!\checkmark} Q\ {}_{\beta}{\rightsquigarrow}_{\tau}\ P' \mathbin{;\!\!\checkmark} Q} \qquad \frac{a\ {}_{\alpha}{\rightsquigarrow}_{P}\ P'}{a \mathbin{;\!\!\checkmark} Q\ {}_{\beta}{\rightsquigarrow}_{P}\ P' \mathbin{;\!\!\checkmark} Q}$$

$$\frac{r\ {}_{\alpha}{\rightsquigarrow}\checkmark_{P}\ P' \qquad Q\ P\ {}_{\beta}{\rightsquigarrow}_{\tau}\ Q'}{r \mathbin{;\!\!\checkmark} Q\ {}_{\beta}{\rightsquigarrow}_{\tau}\ Q'}$$

The continuity has only be obtained under a kind of finiteness assumption, but non-destructiveness holds in general.
Finally, an architectural version is defined. It satisfies the following property.

$$SEQ\checkmark\ l \in@\ (L1\ @\ L2).\ P\ l = (\lambda r.\ (SEQ\checkmark\ l \in@\ L1.\ P\ l)\ r \mathbin{;\!\!\checkmark} SEQ\checkmark\ l \in@\ L2.\ P\ l)$$

18.2.3 Synchronization Product

The main motivation for generalizing the synchronization product was to have a satisfying handling of the synchronization of two terminations. Indeed, with the *Sync* operator inherited from `HOL-CSP`, the returned values were lost (most of the time).

$$SKIP\ r\ [\![A]\!]\ SKIP\ s = (\mathsf{if}\ r = s\ \mathsf{then}\ SKIP\ r\ \mathsf{else}\ STOP)$$

With the new definition, this is not the case anymore.

$$SKIP\ r\ [\![A]\!]_{\checkmark}\ SKIP\ s = (\mathsf{case}\ r\ \otimes\checkmark\ s\ \mathsf{of}\ None \Rightarrow STOP \mid Some\ r\text{-}s \Rightarrow SKIP\ r\text{-}s)$$

This law is directly extracted from the core of the construction, which is done in a very abstract way through a **locale** specification. The operator is then declined in several variations, leading to the following rules.

$$SKIP\ r\ [\![A]\!]_{\checkmark Pair}\ SKIP\ s = SKIP\ (r,\ s)$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Pairlist}\ SKIP\ s = SKIP\ [r,\ s]$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Rlist}\ SKIP\ s = SKIP\ (r \cdot s)$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Llist}\ SKIP\ s = SKIP\ (r\ @\ [s])$$
$$SKIP\ r\ {}_n[\![A]\!]_{\checkmark ListslenL}\ SKIP\ s = (\mathsf{if}\ |r| = n\ \mathsf{then}\ SKIP\ (r\ @\ s)\ \mathsf{else}\ STOP)$$
$$SKIP\ r\ {}_n[\![A]\!]_{\checkmark ListslenR}\ SKIP\ s = (\mathsf{if}\ |s| = n\ \mathsf{then}\ SKIP\ (r\ @\ s)\ \mathsf{else}\ STOP)$$
$$SKIP\ r\ {}_n[\![A]\!]_{\checkmark m}\ SKIP\ s = (\mathsf{if}\ |r| = n \wedge |s| = m\ \mathsf{then}\ SKIP\ (r\ @\ s)\ \mathsf{else}\ STOP)$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Classic}\ SKIP\ s = (\mathsf{if}\ r = s\ \mathsf{then}\ SKIP\ r\ \mathsf{else}\ STOP)$$

Moreover, the last declension is proved to be equal to the old version, ensuring that this work is actually a generalization.

$$P\ [\![A]\!]_{\checkmark Classic}\ Q = P\ [\![A]\!]\ Q$$

We also established commutativity and associativity, modulo renaming the ticks. The underlying abstract setup is quite obscure, so we will only display here the pair versions.

$$RenamingTick\ (P\ [\![A]\!]_{\checkmark Pair}\ Q)\ prod.swap = Q\ [\![A]\!]_{\checkmark Pair}\ P$$
$$P\ [\![A]\!]_{\checkmark Pair}\ (Q\ [\![A]\!]_{\checkmark Pair}\ R) =$$
$$RenamingTick\ (P\ [\![A]\!]_{\checkmark Pair}\ Q\ [\![A]\!]_{\checkmark Pair}\ R)\ (\lambda((r,\ s),\ t).\ (r,\ s,\ t))$$

Again, the expected step law has been established.

$$\Box a{\in}A \to P\ a\ ;\!\!\checkmark\ Q = \Box a{\in}A \to (P\ a\ ;\!\!\checkmark\ Q)$$

In this abstract setup, the operational laws have also been derived.

$$\frac{P\ {}_{lhs}\!\!\rightsquigarrow_{\tau}\ P'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ P'\ [\![A]\!]_{\checkmark}\ Q} \qquad \frac{Q\ {}_{rhs}\!\!\rightsquigarrow_{\tau}\ Q'}{A\ [\![P]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ A\ [\![P]\!]_{\checkmark}\ Q'}$$

$$\frac{a \notin A \qquad P\ {}_{lhs}\!\!\rightsquigarrow_{a}\ P'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{a}\ P'\ [\![A]\!]_{\checkmark}\ Q} \qquad \frac{a \notin A \qquad Q\ {}_{rhs}\!\!\rightsquigarrow_{a}\ Q'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{a}\ P\ [\![A]\!]_{\checkmark}\ Q'}$$

$$\frac{a \in A \qquad P\ {}_{lhs}\!\!\rightsquigarrow_{a}\ P' \qquad Q\ {}_{rhs}\!\!\rightsquigarrow_{a}\ Q'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{a}\ P'\ [\![A]\!]_{\checkmark}\ Q'}$$

$$\frac{P\ {}_{lhs}\!\!\rightsquigarrow_{\checkmark r}\ P'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ SKIP\ r\ [\![A]\!]_{\checkmark}\ Q}$$

$$\frac{Q\ {}_{rhs}\!\!\rightsquigarrow_{\checkmark s}\ Q'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ P\ [\![A]\!]_{\checkmark}\ SKIP\ s}$$

$$\frac{r \otimes\!\checkmark\ s = Some\ r\text{-}s}{SKIP\ r\ [\![A]\!]_{\checkmark}\ SKIP\ s\ {}_{ptick}\!\!\rightsquigarrow_{\checkmark r\text{-}s}\ \Omega_{ptick}\ (SKIP\ r\text{-}s)\ r\text{-}s}$$

Continuity and non-destructiveness hold in general, and an architectural version is defined. It satisfies the following property.

$$\frac{L1 \neq [] \qquad L2 \neq []}{[\![S]\!]_{\checkmark}\ l \in@\ (L1\ @\ L2).\ P\ l = [\![S]\!]_{\checkmark}\ l \in@\ L1.\ P\ l\ {}_{|L1|}[\![S]\!]_{\checkmark|L2|}\ [\![S]\!]_{\checkmark}\ l \in@\ L2.\ P\ l}$$

It is defined on a list (while its counterpart *MultiSync* based on the *Sync* operator is defined on a multiset) because the order of appearance of the ticks matters. However, as long as we keep track of the positions, we can permute the list. This is summarized by the following theorem.

$$\frac{f\ permutes\ \{..<|L|\}}{[\![S]\!]_{\checkmark}\ l \in@\ permute\text{-}list\ f\ L.\ P\ l = RenamingTick\ ([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l)\ (permute\text{-}list\ f)}$$

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, S. Taha, B. Wolff, and L. Ye. Hol-csp version 2.0. *Archive of Formal Proofs*, April 2019. https://isa-afp.org/entries/HOL-CSP.html, Formal proof development.

[3] B. Ballenghien and B. Wolff. Operational semantics formally proven in hol-csp. *Archive of Formal Proofs*, December 2023. https://isa-afp.org/entries/HOL-CSP_OpSem.html, Formal proof development.

[4] 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)*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] B. Ballenghien and B. Wolff. Csp semantics over restriction spaces. *Archive of Formal Proofs*, May 2025. https://isa-afp.org/entries/HOL-CSP_RS.html, Formal proof development.