

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	17
2.2.2	Other properties	17
2.3	Laws	19
2.3.1	Laws of $\mathbb{F}_{\checkmark}(P)$	19
2.3.2	Laws of $\mathbb{F}_{\checkmark} \Rightarrow (f)$	26
3	Generalization of the Sequential Composition	29
3.1	Definition	29
3.1.1	Preliminaries	29
3.1.2	Formal Definition	31
3.2	Projections	36
4	Generalization of the Synchronization Product	39
4.1	Trace Interleaving	39
4.1.1	Motivation	39
4.1.2	Definition	40
4.1.3	First Properties	42
4.1.4	Lengths	50
4.1.5	Trace Prefix Interleaving	51
4.1.6	Hiding Events	65
4.2	Synchronization Product	79
4.2.1	Definition	79
4.2.2	Projections	86
4.2.3	First Properties	91

5	Some Work on Renaming	97
5.1	Tick Swap Operator	97
5.1.1	Preliminaries	97
5.1.2	The Operator	106
5.2	Splitting the Renaming Operator	115
5.2.1	Renaming only Events	115
5.2.2	Renaming only Ticks	116
5.2.3	Properties	118
5.3	Renaming and Generalized Synchronization Product	125
6	Commutativity and Associativity of Synchronization	133
6.1	Commutativity	133
6.1.1	Motivation	133
6.1.2	Formalization	133
6.1.3	First Properties	134
6.1.4	Commutativity	135
6.2	Associativity	137
6.2.1	Motivation	137
6.2.2	Formalization	137
6.2.3	First Properties	138
6.2.4	Associativity for the Traces	138
6.2.5	Associativity	145
7	First Laws	153
7.1	Behaviour with Constant Processes	153
7.1.1	The Laws of $\perp$	153
7.1.2	The Laws of <i>STOP</i>	153
7.1.3	The Laws of <i>SKIP</i>	158
7.2	Associativity of Sequential Composition	165
7.3	Distributivity of Non-Determinism	170
7.3.1	Sequential Composition	170
7.3.2	Synchronization Product	171
8	Communications	175
8.1	Step Laws	175
8.1.1	Sequential Composition	175
8.1.2	Synchronization Product	176
8.2	Extended step Laws	186
8.2.1	Sequential Composition	186
8.2.2	Synchronization Product	186
8.3	Read and Write Laws	199
8.3.1	Sequential Composition	199
8.3.2	Synchronization Product	199

9	Operational Semantics Laws	225
9.1	Behaviour of <i>initials</i>	225
9.1.1	<i>TickSwap</i>	225
9.1.2	Sequential Composition	225
9.1.3	Synchronization Product	226
9.2	Laws of After	228
9.2.1	Sequential Composition	228
9.2.2	Synchronization Product	233
9.3	Small Steps Transitions	242
9.3.1	Extension of the After Operator	242
9.3.2	Sequential Composition	242
9.3.3	Generic Operational Semantics as Locales	244
10	Declensions of the Generalized Synchronization Product	249
10.1	Interpretations	249
10.1.1	Classical Version	249
10.1.2	Product Type	249
10.1.3	List Type	250
10.2	Associativities	252
10.2.1	Classical Version	252
10.2.2	Product Type	252
10.2.3	List Type	253
10.3	Properties	255
10.3.1	Actual Generalization	255
10.3.2	Other Properties	256
10.4	Ticks Length and Conversions	256
10.4.1	Ticks Length	256
10.4.2	Conversions	264
10.5	First Laws	267
10.6	Operational Laws	270
10.6.1	Classical Version	270
10.6.2	Product Type	270
10.6.3	List Type	271
11	Architectural Versions	275
11.1	Sequential Composition	275
11.1.1	Definition	275
11.1.2	First Properties	275
11.1.3	Behaviour with binary version	276
11.1.4	Other Properties	276
11.1.5	Behaviour with injectivity	277
11.2	Synchronization Product	277
11.2.1	Definition	277
11.2.2	First properties	278

11.2.3	Properties	279
11.2.4	Behaviour with binary version	279
11.2.5	Behaviour with injectivity	280
11.2.6	Permuting the Sequence	280
12	Events and Ticks	301
12.1	Preliminaries	301
12.2	Sequential Composition	302
12.2.1	Events	302
12.2.2	Ticks	304
12.3	Synchronization Product	304
12.3.1	Events	304
12.3.2	Ticks	306
12.4	Architectural Operators	306
12.4.1	Events	306
12.4.2	Ticks	307
13	Continuity Rules	309
13.1	Sequential Composition	309
13.1.1	Monotonicity	309
13.1.2	Preliminaries	311
13.1.3	Continuity	315
13.2	Synchronization Product	317
13.2.1	Monotonicity	317
13.2.2	Preliminaries	319
13.2.3	Continuity	320
14	Monotonicity Properties	323
14.0.1	Sequential Composition	323
14.0.2	Multiple Sequential Composition	324
14.0.3	Synchronization Product	324
14.0.4	Multiple Synchronization Product	325
15	Non Destructiveness Rules	327
15.1	Synchronization Product	327
15.1.1	Refinement	327
15.1.2	Non Destructiveness	332
15.1.3	Setup	333
16	Other Laws	335
16.1	Laws of Renaming	335
16.1.1	Renaming and Sequential Composition	335
16.1.2	Renaming and Synchronization Product	342
16.2	Laws of Hiding	349

16.3	Hiding and Sequential Composition	349
16.4	Hiding and Synchronization Product	353
16.5	Other Laws of Synchronization Product	365
16.5.1	Synchronization Set can be restricted	365
16.5.2	Some Refinements	367
17	Deadlock Results	369
17.1	First Results	369
17.1.1	Non Terminating	369
17.1.2	Deadlock Free	369
17.2	Renaming and reference Processes	371
17.2.1	Alternative Definitions with restriction fixed-point Op- erator	371
17.2.2	Stronger Results	372
17.3	Data Independence	372
17.3.1	An interesting equivalence	373
17.3.2	<i>STOP</i> and <i>SKIP</i> synchronized with <i>DF A</i>	373
17.3.3	Finally, <i>deadlock-free</i> ($P \parallel Q$)	374
18	Conclusion	381
18.1	Main Entry Point	381
18.2	Conclusion	381
18.2.1	Summary	381
18.2.2	Sequential Composition	382
18.2.3	Synchronization Product	383

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* ($\langle \checkmark \rangle$)

(isomorphic to option type) by

datatype (*'a, 'r*) *event_{ptick}* = *ev 'a* | *tick 'r* ($\langle \checkmark'(-) \rangle$)

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

Certain definitions must therefore be adapted (mainly by replacing $\checkmark$ with *range tick*). For example, a trace *t* was said to be tick-free if $\checkmark \notin \text{set } t$. In this new setup, such a trace instead satisfies *range tick* $\cap \text{set } t = \{\}$. 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

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

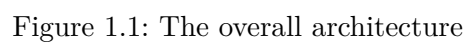
broader perspective, one would in fact like the right-hand process to depend on the return value of the left-hand process, which is not the case in the current framework.

Secondly, the synchronization product does not properly support synchronized termination. Classically, we have $Skip \llbracket S \rrbracket Skip = Skip$, adapted in the last version of HOL-CSP as $SKIP\ r \llbracket A \rrbracket SKIP\ s = (if\ r = s\ then\ SKIP\ r\ else\ STOP)$. 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 $SKIP\ r \llbracket A \rrbracket SKIP\ s = SKIP\ (r, s)$, 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](#).



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* :: $\langle ('a, 'r) \text{ process}_{ptick} \Rightarrow \text{bool} \rangle$
where $\langle \text{finite-all-ticks } P \equiv \forall t \in \mathcal{T} P. \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$

lemma *finite-all-ticksI* : $\langle (\bigwedge t. t \in \mathcal{T} P \Rightarrow \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\}) \Rightarrow \text{finite-all-ticks } P \rangle$
by (*simp add: finite-all-ticks-def*)

lemma *finite-all-ticksD* : $\langle \text{finite-all-ticks } P \Rightarrow \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$
by (*simp add: finite-all-ticks-def*)
(meson is-processT3-TR-append not-finite-existsD)

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* :: $\langle ('a, 'r) \text{ process}_{ptick} \Rightarrow \text{bool} \rangle (\mathbb{F}_{\checkmark}'(-))$
where $\langle \mathbb{F}_{\checkmark}'(P) \equiv \forall t \in \mathcal{T} P. \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P - \mathcal{D} P\} \rangle$

lemma *finite-ticksI* :
 $\langle (\bigwedge t. t \in \mathcal{T} P \Rightarrow t \notin \mathcal{D} P \Rightarrow \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\}) \Rightarrow \mathbb{F}_{\checkmark}'(P) \rangle$
by (*simp add: finite-ticks-def*)
(metis (mono-tags, lifting) Collect-cong append-T-imp-tickFree front-tickFree-Cons-iff is-processT7 is-processT9 not-Cons-self2 not-finite-existsD)

lemma *finite-ticksD* :
 $\langle \mathbb{F}_{\checkmark}'(P) \Rightarrow t \notin \mathcal{D} P \Rightarrow \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$
by (*simp add: finite-ticks-def*)

(metis (lifting) Collect-cong is-processT3-TR-append
is-processT9 not-finite-existsD)

lemma *finite-all-ticks-imp-finite-ticks* [simp] : $\langle \text{finite-all-ticks } P \implies \mathbb{F}_{\checkmark}(P) \rangle$
by (simp add: finite-all-ticksD finite-ticksI)

lemma *finite-all-ticks-is-finite-ticks-or-finite-UNIV* :
 $\langle \text{finite-all-ticks } P \longleftrightarrow (\text{if } \mathcal{D} P = \{\} \text{ then } \mathbb{F}_{\checkmark}(P) \text{ else finite (UNIV :: 'r set)}) \rangle$
 — This is justifying why *finite-all-ticks* is not really interesting.
 for $P :: \langle ('a, 'r) \text{ process}_{ptick} \rangle$
proof (rule iffI)
 show $\langle \text{if } \mathcal{D} P = \{\} \text{ then } \mathbb{F}_{\checkmark}(P) \text{ else finite (UNIV :: 'r set)} \rangle$
 if $\langle \text{finite-all-ticks } P \rangle$
proof (split if-split, intro conjI impI)
 from $\langle \text{finite-all-ticks } P \rangle$ show $\langle \mathcal{D} P = \{\} \implies \mathbb{F}_{\checkmark}(P) \rangle$
 by (simp add: finite-ticksI finite-all-ticksD)
 next
 assume $\langle \mathcal{D} P \neq \{\} \rangle$
 with nonempty-divE obtain t where $\langle tF t \rangle \langle t \in \mathcal{D} P \rangle$ by blast
 hence $\langle t @ [\checkmark(r)] \in \mathcal{D} P \rangle$ for r by (simp add: is-processT7)
 with $\langle \text{finite-all-ticks } P \rangle$ show $\langle \text{finite (UNIV :: 'r set)} \rangle$
 by (metis (mono-tags, lifting) Collect-cong D-T UNIV-I $\langle t \in \mathcal{D} P \rangle$
 finite-all-ticks-def mem-Collect-eq top-set-def)
 qed
 next
 show $\langle \text{if } \mathcal{D} P = \{\} \text{ then } \mathbb{F}_{\checkmark}(P) \text{ else finite (UNIV :: 'r set)} \implies \text{finite-all-ticks } P \rangle$
 by (simp add: finite-ticksD finite-all-ticks-def split: if-split-asm)
 (meson rev-finite-subset subset-UNIV)
 qed

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* :: $\langle (('a, 'r) \text{ process}_{ptick} \Rightarrow ('b, 's) \text{ process}_{ptick}) \Rightarrow \text{bool} \rangle$
 $(\langle \mathbb{F}_{\checkmark \Rightarrow} '(-) \rangle)$
 where $\langle \mathbb{F}_{\checkmark \Rightarrow}(f) \equiv \forall P. \mathbb{F}_{\checkmark}(P) \longrightarrow \mathbb{F}_{\checkmark}(f P) \rangle$

lemma *finite-ticks-funI*: $\langle (\bigwedge P. \mathbb{F}_{\checkmark}(P) \implies \mathbb{F}_{\checkmark}(f P)) \implies \mathbb{F}_{\checkmark \Rightarrow}(f) \rangle$
 by (simp add: finite-ticks-fun-def)

lemma *finite-ticks-funD*: $\langle \mathbb{F}_{\checkmark \Rightarrow}(f) \implies \mathbb{F}_{\checkmark}(P) \implies \mathbb{F}_{\checkmark}(f P) \rangle$
 by (simp add: finite-ticks-fun-def)

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*] : $\langle \mathbf{F}_{\checkmark}(\perp) \rangle$
by (*simp add: finite-ticks-def BOT-projs*)

lemma *finite-ticks-fun-BOT* [*finite-ticks-fun-simps*] : $\langle \mathbf{F}_{\checkmark \Rightarrow}(\perp) \rangle$
by (*simp add: finite-ticks-fun-def finite-ticks-BOT*)

lemma *finite-ticks-SKIP* [*finite-ticks-simps*] : $\langle \mathbf{F}_{\checkmark}(\text{SKIP } r) \rangle$
by (*simp add: finite-ticks-def SKIP-projs*)

lemma *finite-ticks-STOP* [*finite-ticks-simps*] : $\langle \mathbf{F}_{\checkmark}(\text{STOP}) \rangle$
by (*simp add: finite-ticks-def T-STOP*)

lemma *finite-ticks-SKIPS-iff* [*finite-ticks-simps*] : $\langle \mathbf{F}_{\checkmark}(\text{SKIPS } R) \longleftrightarrow \text{finite } R \rangle$
by (*auto simp add: finite-ticks-def SKIPS-projs*)

2.2.2 Other properties

lemma *finite-strict-ticks-of-imp-finite-ticks* [*finite-ticks-simps*] :
 $\langle \text{finite } \checkmark \mathbf{s}(P) \implies \mathbf{F}_{\checkmark}(P) \rangle$
by (*metis (mono-tags, lifting) finite-subset finite-ticksI is-processT9 mem-Collect-eq strict-ticks-of-memI subsetI*)

lemma *finite-strict-ticks-of-image-imp-finite-ticks-fun* [*finite-ticks-fun-simps*] :
 $\langle (\bigwedge x. \text{finite } \checkmark \mathbf{s}(f x)) \implies \mathbf{F}_{\checkmark \Rightarrow}(f) \rangle$
by (*simp add: finite-strict-ticks-of-imp-finite-ticks finite-ticks-funI*)

lemma *anti-mono-finite-ticks* [*finite-ticks-simps*] :
 $\langle \mathbf{F}_{\checkmark}(P) \rangle \text{ if } \langle P \sqsubseteq Q \rangle \langle \mathbf{F}_{\checkmark}(Q) \rangle$
proof (*rule finite-ticksI*)
fix *t* **assume** $\langle t \in \mathcal{T} P \rangle \langle t \notin \mathcal{D} P \rangle$
have $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} = \{r. t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$
by (*meson* $\langle t \notin \mathcal{D} P \rangle$ *is-processT9 le-approx2T* $\langle P \sqsubseteq Q \rangle$)
also have $\langle \text{finite } \dots \rangle$
proof (*rule* $\langle \mathbf{F}_{\checkmark}(Q) \rangle$ [*THEN finite-ticksD*])
from $\langle t \notin \mathcal{D} P \rangle$ *le-approx1* $\langle P \sqsubseteq Q \rangle$ **show** $\langle t \notin \mathcal{D} (Q) \rangle$ **by** *blast*
qed
finally show $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$.
qed

lemma *anti-mono-finite-ticks-fun* [*finite-ticks-fun-simps*] :
 $\langle f \sqsubseteq g \implies \mathbf{F}_{\checkmark \Rightarrow}(g) \implies \mathbf{F}_{\checkmark \Rightarrow}(f) \rangle$
by (*metis anti-mono-finite-ticks finite-ticks-fun-def fun-below-iff*)

lemma *finite-ticks-LUB-iff* [*finite-ticks-fun-simps*] :

$\langle \mathbb{F}_{\checkmark}(\bigsqcup i. Y i) \longleftrightarrow (\forall i. \mathbb{F}_{\checkmark}(Y i)) \rangle$ **if** $\langle \text{chain } Y \rangle$
proof *safe*
from *anti-mono-finite-ticks is-ub-the lub* $\langle \text{chain } Y \rangle$
show $\langle \mathbb{F}_{\checkmark}(\bigsqcup i. Y i) \implies \mathbb{F}_{\checkmark}(Y i) \rangle$ **for** i **by** *blast*
next
show $\langle \mathbb{F}_{\checkmark}(\bigsqcup i. Y i) \rangle$ **if** $\langle \forall i. \mathbb{F}_{\checkmark}(Y i) \rangle$
proof (*rule finite-ticksI*)
fix t **assume** $\langle t \in \mathcal{T}(\bigsqcup i. Y i) \rangle$ $\langle t \notin \mathcal{D}(\bigsqcup i. Y i) \rangle$
from $\langle t \notin \mathcal{D}(\bigsqcup i. Y i) \rangle$ **obtain** j **where** $\langle t \notin \mathcal{D}(Y j) \rangle$
by (*metis D-LUB-2 chain Y limproc-is-the lub*)
have $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T}(\bigsqcup i. Y i)\} = \{r. t @ [\checkmark(r)] \in \mathcal{T}(Y j)\} \rangle$
by (*meson chain Y t not in D (Y j) is-processT9 is-ub-the lub le-approx2T*)
also have $\langle \text{finite } \dots \rangle$
by (*fact* $\langle \forall i. \mathbb{F}_{\checkmark}(Y i) \rangle$ [*THEN spec, THEN finite-ticksD, OF* $\langle t \notin \mathcal{D}(Y j) \rangle$])
finally show $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T}(\bigsqcup i. Y i)\} \rangle$.
qed
qed

lemma *adm-finite-ticks [finite-ticks-simps]* : $\langle \text{adm } (\lambda P. \mathbb{F}_{\checkmark}(P)) \rangle$
by (*rule admI (simp add: finite-ticks-LUB-iff)*)

lemma *finite-ticks-fix [finite-ticks-simps]* :
 $\langle \mathbb{F}_{\checkmark}(\mu X. f X) \rangle$ **if** $\langle \text{cont } f \rangle$ **and** $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \rangle$
proof (*induct rule: fix-ind*)
show $\langle \text{adm finite-ticks} \rangle$ **by** (*fact adm-finite-ticks*)
next
show $\langle \mathbb{F}_{\checkmark}(\perp) \rangle$ **by** (*fact finite-ticks-BOT*)
next
show $\langle \mathbb{F}_{\checkmark}((\lambda X. f X) \cdot X) \rangle$ **if** $\langle \mathbb{F}_{\checkmark}(X) \rangle$ **for** X
by (*simp add: cont f*) (*fact finite-ticks-funD[OF* $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \rangle$ $\langle \mathbb{F}_{\checkmark}(X) \rangle$])
qed

lemma *adm-finite-ticks-fun [finite-ticks-fun-simps]* : $\langle \text{adm } (\lambda f. \mathbb{F}_{\checkmark} \Rightarrow (f)) \rangle$
by (*simp add: admI ch2ch-fun finite-ticks-LUB-iff finite-ticks-fun-def lub-fun*)

lemma *finite-ticks-fun-fix [finite-ticks-fun-simps]* :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (\mu X. f X) \rangle$ **if** $\langle \text{cont } f \rangle$ **and** $\langle \bigwedge x. \mathbb{F}_{\checkmark} \Rightarrow (x) \implies \mathbb{F}_{\checkmark} \Rightarrow (f x) \rangle$
proof (*induct f rule: cont-fix-ind*)
from $\langle \text{cont } f \rangle$ **show** $\langle \text{cont } f \rangle$.
next
from *adm-finite-ticks-fun* **show** $\langle \text{adm } (\lambda f. \mathbb{F}_{\checkmark} \Rightarrow (f)) \rangle$.
next
from *finite-ticks-fun-BOT* **show** $\langle \mathbb{F}_{\checkmark} \Rightarrow (\perp) \rangle$.
next
from $\langle \bigwedge y. \mathbb{F}_{\checkmark} \Rightarrow (y) \implies \mathbb{F}_{\checkmark} \Rightarrow (f y) \rangle$ **show** $\langle \mathbb{F}_{\checkmark} \Rightarrow (x) \implies \mathbb{F}_{\checkmark} \Rightarrow (f x) \rangle$ **for** x .
qed

lemma *finite-ticks-fun-id* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (id) \rangle \langle \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. x) \rangle$
by (*simp-all add: finite-ticks-funI*)

lemma *finite-ticks-fun-const-iff* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. P) \rangle \longleftrightarrow \langle \mathbb{F}_{\checkmark}(P) \rangle$
by (*meson finite-ticks-STOP finite-ticks-fun-def*)

lemma *finite-ticks-fun-comp* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (g) \rangle \Longrightarrow \langle \mathbb{F}_{\checkmark} \Rightarrow (f) \rangle \Longrightarrow \langle \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. g (f x)) \rangle$
by (*simp add: finite-ticks-fun-def*)

2.3 Laws

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

lemma *finite-ticks-Ndet* [*finite-ticks-simps*] :
 $\langle \mathbb{F}_{\checkmark}(P \sqcap Q) \rangle \text{ if } \langle \mathbb{F}_{\checkmark}(P) \rangle \langle \mathbb{F}_{\checkmark}(Q) \rangle$
proof (*rule finite-ticksI*)
fix *t* **assume** $\langle t \in \mathcal{T} (P \sqcap Q) \rangle \langle t \notin \mathcal{D} (P \sqcap Q) \rangle$
from $\langle t \in \mathcal{T} (P \sqcap Q) \rangle$
have $\langle t \in \mathcal{T} P \wedge t \in \mathcal{T} Q \vee t \in \mathcal{T} P \wedge (\forall r. t @ [\checkmark(r)] \notin \mathcal{T} Q) \vee (\forall r. t @ [\checkmark(r)] \notin \mathcal{T} P) \wedge t \in \mathcal{T} Q \rangle$
unfolding *T-Ndet* **by** (*metis Un-iff is-processT3-TR-append*)
with $\langle \mathbb{F}_{\checkmark}(P) \rangle \langle \mathbb{F}_{\checkmark}(Q) \rangle \langle t \notin \mathcal{D} (P \sqcap Q) \rangle$ **show** $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} (P \sqcap Q)\} \rangle$
by (*auto simp add: Ndet-projs dest: finite-ticksD*)
qed

lemma *finite-ticks-Det* [*finite-ticks-simps*] :
 $\langle \mathbb{F}_{\checkmark}(P \sqcup Q) \rangle \text{ if } \langle \mathbb{F}_{\checkmark}(P) \rangle \langle \mathbb{F}_{\checkmark}(Q) \rangle$
proof –
have $\langle \mathbb{F}_{\checkmark}(P \sqcup Q) \rangle = \langle \mathbb{F}_{\checkmark}(P \sqcap Q) \rangle$ **by** (*simp add: finite-ticks-def Det-projs Ndet-projs*)
with $\langle \mathbb{F}_{\checkmark}(P) \rangle \langle \mathbb{F}_{\checkmark}(Q) \rangle$ **show** $\langle \mathbb{F}_{\checkmark}(P \sqcup Q) \rangle$ **by** (*simp add: finite-ticks-Ndet*)
qed

lemma *finite-ticks-Sliding* [*finite-ticks-simps*] :
 $\langle \mathbb{F}_{\checkmark}(P) \rangle \Longrightarrow \langle \mathbb{F}_{\checkmark}(Q) \rangle \Longrightarrow \langle \mathbb{F}_{\checkmark}(P \triangleright Q) \rangle$
by (*simp add: Sliding-def finite-ticks-Ndet finite-ticks-Det*)

lemma *finite-ticks-Interrupt* [*finite-ticks-simps*] :
 $\langle \mathbb{F}_{\checkmark}(P \triangle Q) \rangle \text{ if } \langle \mathbb{F}_{\checkmark}(P) \rangle \langle \mathbb{F}_{\checkmark}(Q) \rangle$
proof (*cases* $\langle Q = \perp \rangle$)

```

  show  $\langle Q = \perp \implies \mathbf{F}_{\checkmark}(P \triangle Q) \rangle$  by (simp add: finite-ticks-BOT)
next
  show  $\langle \mathbf{F}_{\checkmark}(P \triangle Q) \rangle$  if  $\langle Q \neq \perp \rangle$ 
  proof (rule finite-ticksI)
    fix  $t$  assume  $\langle t \in \mathcal{T}(P \triangle Q) \rangle$   $\langle t \notin \mathcal{D}(P \triangle Q) \rangle$ 
    have  $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T}(P \triangle Q)\} \subseteq$ 
       $\{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \cup$ 
       $(\bigcup u \in \{u. \exists v. t = u @ v \wedge u \in \mathcal{T} P\}. \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in$ 
 $\mathcal{T} Q\}) \rangle$ 
    by (simp add: subset-iff T-Interrupt)
      (metis Prefix-Order.prefix-length-le Prefix-Order.same-prefix-nil
        append-eq-append-conv-if append-eq-first-pref-spec butlast-append but-
last-snoc)
    moreover have  $\langle \text{finite } \dots \rangle$ 
    proof (rule finite-UnI)
      from D-Interrupt  $\langle t \notin \mathcal{D}(P \triangle Q) \rangle$  have  $\langle t \notin \mathcal{D} P \rangle$  by blast
      thus  $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$  by (simp add: finite-ticksD  $\langle \mathbf{F}_{\checkmark}(P) \rangle$ )
    next
      show  $\langle \text{finite } (\bigcup u \in \{u. \exists v. t = u @ v \wedge u \in \mathcal{T} P\}. \{r. \text{drop}(\text{length } u) t @$ 
 $[\checkmark(r)] \in \mathcal{T} Q\}) \rangle$ 
      proof (rule finite-UN-I)
        show  $\langle \text{finite } \{u. \exists v. t = u @ v \wedge u \in \mathcal{T} P\} \rangle$  by (prove-finite-subset-of-prefixes
 $t$ )
      next
        fix  $u$  assume  $\langle u \in \{u. \exists v. t = u @ v \wedge u \in \mathcal{T} P\} \rangle$ 
        then obtain  $v$  where  $\langle u \in \mathcal{T} P \rangle$   $\langle t = u @ v \rangle$  by blast
        with  $\langle t \in \mathcal{T}(P \triangle Q) \rangle$  append-T-imp-tickFree consider  $\langle tF u \rangle \mid \langle v = [] \rangle$ 
      by blast
        thus  $\langle \text{finite } \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$ 
        proof cases
          assume  $\langle tF u \rangle$ 
          with  $\langle u \in \mathcal{T} P \rangle$   $\langle t \notin \mathcal{D}(P \triangle Q) \rangle$  have  $\langle v \notin \mathcal{D} Q \rangle$ 
            by (simp add: D-Interrupt  $\langle t = u @ v \rangle$ )
          thus  $\langle tF u \implies \text{finite } \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$ 
            by (simp add:  $\langle t = u @ v \rangle$  finite-ticksD  $\langle \mathbf{F}_{\checkmark}(Q) \rangle$ )
        next
          from BOT-iff-Nil-D  $\langle Q \neq \perp \rangle$  have  $\langle [] \notin \mathcal{D} Q \rangle$  by blast
          with  $\langle \mathbf{F}_{\checkmark}(Q) \rangle$  finite-ticksD have  $\langle \text{finite } \{r. [\checkmark(r)] \in \mathcal{T} Q\} \rangle$  by force
          thus  $\langle v = [] \implies \text{finite } \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$ 
            by (simp add:  $\langle t = u @ v \rangle$ )
        qed
      qed
    qed
  ultimately show  $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T}(P \triangle Q)\} \rangle$  by (fact finite-subset)
qed
qed

```

lemma finite-ticks-Throw [finite-ticks-simps] :

$\langle \mathbb{F}_{\checkmark}(P \Theta a \in A. Q a) \rangle$ if $\langle \mathbb{F}_{\checkmark}(P) \rangle \langle \bigwedge a. a \in A \implies \mathbb{F}_{\checkmark}(Q a) \rangle$
proof (*rule finite-ticksI*)
fix t **assume** $\langle t \in \mathcal{T} (P \Theta a \in A. Q a) \rangle \langle t \notin \mathcal{D} (P \Theta a \in A. Q a) \rangle$
then consider $\langle t \in \mathcal{T} P \rangle \langle \text{set } t \cap \text{ev } 'A = \{\}' \rangle$
 $| t1 \ a \ t2$ **where** $\langle t = t1 \ @ \ \text{ev } a \ \# \ t2 \rangle \langle t1 \ @ \ [\text{ev } a] \in \mathcal{T} P \rangle$
 $\langle \text{set } t1 \cap \text{ev } 'A = \{\}' \rangle \langle a \in A \rangle \langle t2 \in \mathcal{T} (Q a) \rangle$
unfolding *Throw-projs* **by** *blast*
thus $\langle \text{finite } \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} (P \Theta a \in A. Q a)\} \rangle$
proof *cases*
assume $\langle t \in \mathcal{T} P \rangle \langle \text{set } t \cap \text{ev } 'A = \{\}' \rangle$
hence $\langle \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} (P \Theta a \in A. Q a)\} \subseteq \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} P \rangle$
by (*auto simp add: T-Throw D-T is-processT7 disjoint-iff image-iff*)
(*metis (no-types) butlast.simps(2) butlast-append butlast-snoc in-set-conv-decomp*)
moreover have $\langle \text{finite } \dots \rangle$
proof (*rule* $\langle \mathbb{F}_{\checkmark}(P) \rangle$ *[THEN finite-ticksD]*)
from $\langle \text{set } t \cap \text{ev } 'A = \{\}' \rangle$
have $\langle t \in \mathcal{D} P \implies (\text{if } tF \ t \text{ then } t \text{ else butlast } t) \in \mathcal{D} (P \Theta a \in A. Q a) \rangle$
by (*cases t rule: rev-cases, simp-all add: D-Throw*)
(*metis D-imp-front-tickFree* $\langle \text{set } t \cap \text{ev } 'A = \{\}' \rangle$ *append.right-neutral*)
butlast-snoc
div-butlast-when-non-tickFree-iff front-tickFree-Nil front-tickFree-nonempty-append-imp
not-Cons-self2 not-is-ev tickFree-Cons-iff tickFree-append-iff
with $\langle t \notin \mathcal{D} (P \Theta a \in A. Q a) \rangle$ *D-imp-front-tickFree div-butlast-when-non-tickFree-iff*
show $\langle t \notin \mathcal{D} P \rangle$ **by** *blast*
qed
ultimately show $\langle \text{finite } \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} (P \Theta a \in A. Q a)\} \rangle$ **by** (*fact finite-subset*)
next
fix $t1 \ a \ t2$ **assume** $* : \langle t = t1 \ @ \ \text{ev } a \ \# \ t2 \rangle \langle t1 \ @ \ [\text{ev } a] \in \mathcal{T} P \rangle$
 $\langle \text{set } t1 \cap \text{ev } 'A = \{\}' \rangle \langle a \in A \rangle \langle t2 \in \mathcal{T} (Q a) \rangle$
from $\langle t \notin \mathcal{D} (P \Theta a \in A. Q a) \rangle$
have $\langle t \notin \{t1 \ @ \ t2 \mid t1 \ t2. t1 \in \mathcal{D} P \wedge tF \ t1 \wedge \text{set } t1 \cap \text{ev } 'A = \{\}' \wedge ftF \ t2\} \rangle$
by (*simp add: D-Throw UnI1*)

with $*$ **have** $\langle \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} (P \Theta a \in A. Q a)\} = \{r. t2 \ @ \ [\checkmark(r)] \in \mathcal{T} (Q a)\} \rangle$
by (*simp add: T-Throw, safe*)
(*metis Cons-eq-appendI append-assoc butlast-snoc front-tickFree-charn*
non-tickFree-tick tickFree-Nil tickFree-append-iff tickFree-imp-front-tickFree,
solves $\langle \text{simp add: Throw-T-third-clause-breaker} \rangle$, *metis*)
also have $\langle \text{finite } \dots \rangle$
proof (*rule* $\langle \bigwedge a. a \in A \implies \mathbb{F}_{\checkmark}(Q a) \rangle$ *[THEN finite-ticksD, OF* $\langle a \in A \rangle$ *])*
from $\langle t \notin \mathcal{D} (P \Theta a \in A. Q a) \rangle \langle t1 \ @ \ [\text{ev } a] \in \mathcal{T} P \rangle \langle \text{set } t1 \cap \text{ev } 'A = \{\}' \rangle$
show $\langle t2 \notin \mathcal{D} (Q a) \rangle$ **by** (*auto simp add: D-Throw* $\langle t = t1 \ @ \ \text{ev } a \ \# \ t2 \rangle \langle a \in A \rangle$)
qed
finally show $\langle \text{finite } \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} (P \Theta a \in A. Q a)\} \rangle$.
qed
qed

lemma *finite-ticks-Renaming* [*finite-ticks-simps*] :

$\langle \mathbb{F}_{\checkmark}(\text{Renaming } P \text{ } f \text{ } g) \rangle$ **if** $\langle \text{finitary } f \rangle \langle \text{finitary } g \rangle \langle \mathbb{F}_{\checkmark}(P) \rangle$

proof (*rule finite-ticksI*)

fix t **assume** $\langle t \notin \mathcal{D}(\text{Renaming } P \text{ } f \text{ } g) \rangle$

hence $\langle \{s. t @ [\checkmark(s)] \in \mathcal{T}(\text{Renaming } P \text{ } f \text{ } g)\} \subseteq$
 $(\bigcup u \in \{u. t = \text{map}(\text{map-event}_{ptick} f g) u \wedge u \in \mathcal{T} P\}. \{g \text{ } r \mid r. u @ [\checkmark(r)]$
 $\in \mathcal{T} P\}) \rangle$

by (*auto simp add: subset-iff Renaming-projs append-eq-map-conv tick-eq-map-event_{ptick}-iff*)
(use is-processT3-TR-append in blast,
metis append-Nil butlast-append event_{ptick}.disc(2) front-tickFree-iff-tickFree-butlast
map-event_{ptick}-tickFree snoc-eq-iff-butlast tickFree-butlast)

moreover have $\langle \text{finite } \dots \rangle$

proof (*rule finite-UN-I*)

have $\langle \text{finitary}(\text{map-event}_{ptick} f g) \rangle$ **by** (*simp add: Cont-RenH2* $\langle \text{finitary } f \rangle$
 $\langle \text{finitary } g \rangle$)

have $\langle \{u. t = \text{map}(\text{map-event}_{ptick} f g) u \wedge u \in \mathcal{T} P\} \subseteq \{u. t = \text{map}$
 $(\text{map-event}_{ptick} f g) u\} \rangle$ **by** *blast*

moreover from *Cont-RenH4* $\langle \text{finitary}(\text{map-event}_{ptick} f g) \rangle$ **have** $\langle \text{finite } \dots \rangle$

by *blast*

ultimately show $\langle \text{finite } \{u. t = \text{map}(\text{map-event}_{ptick} f g) u \wedge u \in \mathcal{T} P\} \rangle$ **by**
(fact finite-subset)

next

fix u **assume** $\langle u \in \{u. t = \text{map}(\text{map-event}_{ptick} f g) u \wedge u \in \mathcal{T} P\} \rangle$

hence $\langle t = \text{map}(\text{map-event}_{ptick} f g) u \rangle \langle u \in \mathcal{T} P \rangle$ **by** *simp-all*

with $\langle t \notin \mathcal{D}(\text{Renaming } P \text{ } f \text{ } g) \rangle$ **have** $\langle u \notin \mathcal{D} P \rangle$

by (*simp add: D-Renaming*)
(metis (no-types, opaque-lifting) D-imp-front-tickFree append-Nil append-Nil2
div-butlast-when-non-tickFree-iff front-tickFree-chn map-butlast
map-event_{ptick}-tickFree snoc-eq-iff-butlast tickFree-Nil)

thus $\langle \text{finite } \{g \text{ } r \mid r. u @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$

by (*simp add: finite-ticksD* $\langle \mathbb{F}_{\checkmark}(P) \rangle$)

qed

ultimately show $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T}(\text{Renaming } P \text{ } f \text{ } g)\} \rangle$ **by** (*fact fi-*
nite-subset)

qed

lemma *finite-ticks-Seq* [*finite-ticks-simps*] :

$\langle \mathbb{F}_{\checkmark}(P ; Q) \rangle$ **if** $\langle \mathbb{F}_{\checkmark}(Q) \rangle$

proof (*cases* $\langle Q = \perp \rangle$)

from *not-finite-existsD* **show** $\langle Q = \perp \implies \mathbb{F}_{\checkmark}(P ; Q) \rangle$

by (*auto simp add: finite-ticks-def Seq-projs BOT-projs*)

next

show $\langle \mathbb{F}_{\checkmark}(P ; Q) \rangle$ **if** $\langle Q \neq \perp \rangle$

proof (*rule finite-ticksI*)

fix t **assume** $\langle t \notin \mathcal{D}(P ; Q) \rangle$

hence $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T}(P ; Q)\} \subseteq$

$(\bigcup u \in \{u. \exists v r. t = u @ v \wedge u @ [\checkmark(r)] \in \mathcal{T} P\}. \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\})$
by (*auto simp add: Seq-projs intro: is-processT9*)
(metis (no-types, opaque-lifting) T-imp-front-tickFree append-butlast-last-id
append-eq-conv-conj butlast-append butlast-snoc front-tickFree-nonempty-append-imp
last-appendR list.distinct(1) non-tickFree-tick tickFree-append-iff)
moreover have $\langle \text{finite } \dots \rangle$
proof (*rule finite-UN-I*)
show $\langle \text{finite } \{u. \exists v r. t = u @ v \wedge u @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$
by (*prove-finite-subset-of-prefixes t*)
next
fix u **assume** $\langle u \in \{u. \exists v r. t = u @ v \wedge u @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$
then obtain $v r$ **where** $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle t = u @ v \rangle$ **by** *blast*
with *append-T-imp-tickFree* **consider** $\langle tF u \rangle \mid \langle v = [] \rangle$ **by** *blast*
thus $\langle \text{finite } \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$
proof cases
assume $\langle tF u \rangle$
with $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle t \notin \mathcal{D}(P; Q) \rangle$ **have** $\langle v \notin \mathcal{D} Q \rangle$
by (*auto simp add: D-Seq* $\langle t = u @ v \rangle$)
thus $\langle tF u \implies \text{finite } \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$
by (*simp add:* $\langle t = u @ v \rangle$ *finite-ticksD* $\langle F_{\checkmark}(Q) \rangle$)
next
from *BOT-iff-Nil-D* $\langle Q \neq \perp \rangle$ **have** $\langle [] \notin \mathcal{D} Q \rangle$ **by** *blast*
with $\langle F_{\checkmark}(Q) \rangle$ *finite-ticksD* **have** $\langle \text{finite } \{r. [\checkmark(r)] \in \mathcal{T} Q\} \rangle$ **by** *force*
thus $\langle v = [] \implies \text{finite } \{r. \text{drop}(\text{length } u) t @ [\checkmark(r)] \in \mathcal{T} Q\} \rangle$
by (*simp add:* $\langle t = u @ v \rangle$)
qed
qed
ultimately show $\langle \text{finite } \{r. t @ [\checkmark(r)] \in \mathcal{T}(P; Q)\} \rangle$ **by** (*fact finite-subset*)
qed
qed

lemma *finite-ticks-Sync* [*finite-ticks-simps*] :

$\langle F_{\checkmark}(P \llbracket S \rrbracket Q) \rangle$ **if** $\langle F_{\checkmark}(P) \vee F_{\checkmark}(Q) \rangle$
proof (*rule finite-ticksI*)
fix t **assume** $\langle t \notin \mathcal{D}(P \llbracket S \rrbracket Q) \rangle$
have $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T}(P \llbracket S \rrbracket Q)\} \subseteq$
 $(\bigcup (t-P, t-Q) \in \{(t-P, t-Q). t \text{ setinterleaves } ((t-P, t-Q), \text{range tick} \cup \text{ev } S)\}.$
 $\{r. t-P @ [\checkmark(r)] \in \mathcal{T} P \wedge t-P \notin \mathcal{D} P \wedge t-Q @ [\checkmark(r)] \in \mathcal{T} Q \wedge$
 $t-Q \notin \mathcal{D} Q\}) \rangle$
(is $\langle - \subseteq ?rhs \rangle$ *)*
proof (*rule subsetI*)
fix r **assume** $\langle r \in \{r. t @ [\checkmark(r)] \in \mathcal{T}(P \llbracket S \rrbracket Q)\} \rangle$
hence $\langle t @ [\checkmark(r)] \in \mathcal{T}(P \llbracket S \rrbracket Q) \rangle$ **..**
moreover from $\langle t \notin \mathcal{D}(P \llbracket S \rrbracket Q) \rangle$ **have** $\langle t @ [\checkmark(r)] \notin \mathcal{D}(P \llbracket S \rrbracket Q) \rangle$
by (*meson is-processT9*)
ultimately obtain $t-P \ t-Q$ **where** $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle \langle t-P \notin \mathcal{D} P \rangle$

$\langle t-Q \notin \mathcal{D} \ Q \rangle$
 $\langle (t \ @ \ [\checkmark(r)]) \text{ setinterleaves } ((t-P, t-Q), \text{ range tick } \cup \text{ ev } 'S) \rangle$
by (*simp add: Sync-projs*)
 $(\text{metis (no-types, lifting) append.right-neutral front-tickFree-Nil setinterleaving-sym})$
from *this(1-4) SyncWithTick-imp-NTF[OF this(5)]* **show** $\langle r \in ?rhs \rangle$
by *simp (metis T-imp-front-tickFree front-tickFree-append-iff is-processT7 not-Cons-self2)*
qed
moreover have $\langle \text{finite } \dots \rangle$
proof (*rule finite-UN-I, safe*)
show $\langle \text{finite } \{(t-P, t-Q). t \text{ setinterleaves } ((t-P, t-Q), \text{ range tick } \cup \text{ ev } 'S)\} \rangle$
by (*fact finite-interleaves*)
next
from $\langle \mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q) \rangle$ *finite-ticksD*
show $\langle \text{finite } \{r. t-P \ @ \ [\checkmark(r)] \in \mathcal{T} \ P \wedge t-P \notin \mathcal{D} \ P \wedge$
 $t-Q \ @ \ [\checkmark(r)] \in \mathcal{T} \ Q \wedge t-Q \notin \mathcal{D} \ Q\} \rangle$ **for** $t-P \ t-Q$ **by** *fastforce*
qed
ultimately show $\langle \text{finite } \{r. t \ @ \ [\checkmark(r)] \in \mathcal{T} \ (P \llbracket S \rrbracket \ Q)\} \rangle$ **by** (*fact finite-subset*)
qed
corollary $\langle \mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q) \implies \mathbb{F}_{\checkmark}(P \parallel Q) \rangle$
and $\langle \mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q) \implies \mathbb{F}_{\checkmark}(P \parallel\!\!\parallel Q) \rangle$
by (*fact finite-ticks-Sync*) $+$

lemma *finite-ticks-GlobalNdet [finite-ticks-simps] :*
 $\langle \text{finite } A \implies (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark}(P \ a)) \implies \mathbb{F}_{\checkmark}(\bigcap a \in A. P \ a) \rangle$
— We can't expect *infinite* A here, see $\mathbb{F}_{\checkmark}(\text{SKIPS } R) = \text{finite } R$.
by (*induct A rule: induct-subset-empty-single*)
 $(\text{simp-all add: GlobalNdet-distrib-unit finite-ticks-Ndet finite-ticks-STOP})$

lemma *finite-ticks-GlobalDet [finite-ticks-simps] :*
 $\langle \text{finite } A \implies (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark}(P \ a)) \implies \mathbb{F}_{\checkmark}(\bigcap a \in A. P \ a) \rangle$
by (*induct A rule: finite-induct*)
 $(\text{simp-all add: GlobalDet-distrib-unit-bis finite-ticks-Det finite-ticks-STOP})$

lemma $\langle L = [] \implies \mathbb{F}_{\checkmark}(\text{SEQ } l \in @L. P \ l) \rangle$ **by** (*simp add: finite-ticks-SKIP*)

lemma *finite-ticks-MultiSeq-nonempty [finite-ticks-simps] :*
 $\langle L \neq [] \implies \mathbb{F}_{\checkmark}(P \ (\text{last } L)) \implies \mathbb{F}_{\checkmark}(\text{SEQ } l \in @L. P \ l) \rangle$
by (*induct L rule: rev-induct*) (*simp-all add: finite-ticks-Seq*)

lemma *finite-ticks-MultiSync [finite-ticks-simps] :*
 $\langle (\bigwedge m. m \in \# \ M \implies \mathbb{F}_{\checkmark}(P \ m)) \implies \mathbb{F}_{\checkmark}(\llbracket S \rrbracket \ m \in \# \ M. P \ m) \rangle$
by (*induct M rule: induct-subset-mset-empty-single*)
 $(\text{simp-all add: finite-ticks-Sync finite-ticks-STOP})$

corollary $\langle (\bigwedge m. m \in \# M \implies \mathbf{F}_{\checkmark}(P\ m)) \implies \mathbf{F}_{\checkmark}(\parallel m \in \# M. P\ m) \rangle$
and $\langle (\bigwedge m. m \in \# M \implies \mathbf{F}_{\checkmark}(P\ m)) \implies \mathbf{F}_{\checkmark}(\parallel\parallel m \in \# M. P\ m) \rangle$
by (*fact finite-ticks-MultiSync*) $+$

lemma *finite-ticks-Mprefix-iff* [*finite-ticks-simps*] :

$\langle \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \longleftrightarrow (\forall a \in A. \mathbf{F}_{\checkmark}(P\ a)) \rangle$

proof (*safe intro!; finite-ticksI*)

fix $t\ a$ **assume** $\langle \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \rangle \langle a \in A \rangle \langle t \in \mathcal{T}(P\ a) \rangle \langle t \notin \mathcal{D}(P\ a) \rangle$

have $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T}(P\ a)\} = \{r. (ev\ a \# t) @ [\checkmark(r)] \in \mathcal{T}(\Box a \in A \rightarrow P\ a)\} \rangle$

by (*auto simp add: $\langle a \in A \rangle$ T-Mprefix*)

also have $\langle \text{finite} \dots \rangle$

by (*rule $\langle \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \rangle$ [THEN finite-ticksD]*)

(*simp add: D-Mprefix $\langle t \notin \mathcal{D}(P\ a) \rangle$*)

finally show $\langle \text{finite} \{r. t @ [\checkmark(r)] \in \mathcal{T}(P\ a)\} \rangle$.

next

fix t **assume** $\langle \forall a \in A. \mathbf{F}_{\checkmark}(P\ a) \rangle \langle t \in \mathcal{T}(\Box a \in A \rightarrow P\ a) \rangle \langle t \notin \mathcal{D}(\Box a \in A \rightarrow P\ a) \rangle$

from $\langle t \in \mathcal{T}(\Box a \in A \rightarrow P\ a) \rangle$ **consider** $\langle t = [] \rangle \mid u\ a$ **where** $\langle a \in A \rangle \langle t = ev\ a \# u \rangle$

by (*auto simp add: T-Mprefix*)

thus $\langle \text{finite} \{r. t @ [\checkmark(r)] \in \mathcal{T}(\Box a \in A \rightarrow P\ a)\} \rangle$

proof *cases*

show $\langle t = [] \implies \text{finite} \{r. t @ [\checkmark(r)] \in \mathcal{T}(\Box a \in A \rightarrow P\ a)\} \rangle$ **by** (*simp add:*

T-Mprefix)

next

fix $u\ a$ **assume** $\langle a \in A \rangle \langle t = ev\ a \# u \rangle$

hence $\langle \{r. t @ [\checkmark(r)] \in \mathcal{T}(\Box a \in A \rightarrow P\ a)\} = \{r. u @ [\checkmark(r)] \in \mathcal{T}(P\ a)\} \rangle$

by (*simp add: set-eq-iff T-Mprefix*)

also have $\langle \text{finite} \dots \rangle$

by (*rule $\langle \forall a \in A. \mathbf{F}_{\checkmark}(P\ a) \rangle$ [THEN bspec, OF $\langle a \in A \rangle$, THEN finite-ticksD]*)

(*use $\langle t \notin \mathcal{D}(\Box a \in A \rightarrow P\ a) \rangle$ in $\langle \text{simp add: } \langle t = ev\ a \# u \rangle$ D-Mprefix $\langle a \in A \rangle$*)

finally show $\langle \text{finite} \{r. t @ [\checkmark(r)] \in \mathcal{T}(\Box a \in A \rightarrow P\ a)\} \rangle$.

qed

qed

lemma *finite-ticks-Mndetprefix-iff* [*finite-ticks-simps*] :

$\langle \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \longleftrightarrow (\forall a \in A. \mathbf{F}_{\checkmark}(P\ a)) \rangle$

proof $-$

have $\langle \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \longleftrightarrow \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \rangle$

by (*simp add: finite-ticks-def Mndetprefix-projs Mprefix-projs*)

thus $\langle \mathbf{F}_{\checkmark}(\Box a \in A \rightarrow P\ a) \longleftrightarrow (\forall a \in A. \mathbf{F}_{\checkmark}(P\ a)) \rangle$ **by** (*simp add: finite-ticks-Mprefix-iff*)

qed

lemma *finite-ticks-write0-iff* [*finite-ticks-simps*] : $\langle \mathbf{F}_{\checkmark}(a \rightarrow P) \longleftrightarrow \mathbf{F}_{\checkmark}(P) \rangle$

by (*simp add: write0-def finite-ticks-Mprefix-iff*)

lemma *finite-ticks-write-iff* [*finite-ticks-simps*] : $\langle \mathbb{F}_{\checkmark}(c!a \rightarrow P) \longleftrightarrow \mathbb{F}_{\checkmark}(P) \rangle$
by (*simp add: write-def finite-ticks-Mprefix-iff*)

lemma *finite-ticks-read-iff* :
 $\langle \mathbb{F}_{\checkmark}(c?a \in A \rightarrow P\ a) \longleftrightarrow (\forall b \in c.\ 'A. \mathbb{F}_{\checkmark}(P\ (\text{inv-into}\ A\ c\ b))) \rangle$
by (*simp add: read-def finite-ticks-Mprefix-iff*)

lemma *finite-ticks-inj-on-read-iff* [*finite-ticks-simps*] :
 $\langle \text{inj-on}\ c\ A \implies \mathbb{F}_{\checkmark}(c?a \in A \rightarrow P\ a) \longleftrightarrow (\forall a \in A. \mathbb{F}_{\checkmark}(P\ a)) \rangle$
by (*simp add: read-def finite-ticks-Mprefix-iff*)

lemma *finite-ticks-ndet-write-iff* :
 $\langle \mathbb{F}_{\checkmark}(c!!a \in A \rightarrow P\ a) \longleftrightarrow (\forall b \in c.\ 'A. \mathbb{F}_{\checkmark}(P\ (\text{inv-into}\ A\ c\ b))) \rangle$
by (*simp add: ndet-write-def finite-ticks-Mndetprefix-iff*)

lemma *finite-ticks-inj-on-ndet-write-iff* [*finite-ticks-simps*] :
 $\langle \text{inj-on}\ c\ A \implies \mathbb{F}_{\checkmark}(c!!a \in A \rightarrow P\ a) \longleftrightarrow (\forall a \in A. \mathbb{F}_{\checkmark}(P\ a)) \rangle$
by (*simp add: ndet-write-def finite-ticks-Mndetprefix-iff*)

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

lemma *finite-ticks-fun-Det* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \implies \mathbb{F}_{\checkmark} \Rightarrow (g) \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. f\ x \sqcap g\ x) \rangle$
by (*simp add: finite-ticks-Det finite-ticks-fun-def*)

lemma *finite-ticks-fun-Ndet* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \implies \mathbb{F}_{\checkmark} \Rightarrow (g) \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. f\ x \sqcap g\ x) \rangle$
by (*simp add: finite-ticks-Ndet finite-ticks-fun-def*)

lemma *finite-ticks-fun-Sliding* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \implies \mathbb{F}_{\checkmark} \Rightarrow (g) \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. f\ x \triangleright g\ x) \rangle$
by (*simp add: finite-ticks-Sliding finite-ticks-fun-def*)

lemma *finite-ticks-fun-Interrupt* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \implies \mathbb{F}_{\checkmark} \Rightarrow (g) \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. f\ x \triangle g\ x) \rangle$
by (*simp add: finite-ticks-Interrupt finite-ticks-fun-def*)

lemma *finite-ticks-fun-Throw* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (f) \implies (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} \Rightarrow (g\ a)) \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. f\ x \ominus a \in A. g\ a\ x) \rangle$
by (*simp add: finite-ticks-Throw finite-ticks-fun-def*)

lemma *finite-ticks-fun-Renaming* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (P) \implies \text{finitary}\ f \implies \text{finitary}\ g \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. \text{Renaming}\ (P\ x)\ f\ g) \rangle$
by (*simp add: finite-ticks-Renaming finite-ticks-fun-def*)

lemma *finite-ticks-fun-RenamingF* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} \Rightarrow (P) \implies \mathbb{F}_{\checkmark} \Rightarrow (\lambda x. (P\ x)\ [\![a := b]\!] [\![c := d]\!]) \rangle$
by (*simp add: finite-ticks-fun-Renaming*)

lemma *finite-ticks-fun-Seq* [*finite-ticks-fun-simps*] :

$\langle \mathbb{F}_{\checkmark} (g) \implies \mathbb{F}_{\checkmark} (\lambda x. f x ; g x) \rangle$

by (*simp add: finite-ticks-Seq finite-ticks-fun-def*)

lemma *finite-ticks-fun-Sync* [*finite-ticks-fun-simps*] :

$\langle \mathbb{F}_{\checkmark} (f) \implies \mathbb{F}_{\checkmark} (g) \implies \mathbb{F}_{\checkmark} (\lambda x. f x \llbracket S \rrbracket g x) \rangle$

by (*simp add: finite-ticks-Sync finite-ticks-fun-def*)

corollary $\langle \mathbb{F}_{\checkmark} (f) \implies \mathbb{F}_{\checkmark} (g) \implies \mathbb{F}_{\checkmark} (\lambda x. f x \parallel g x) \rangle$

and $\langle \mathbb{F}_{\checkmark} (f) \implies \mathbb{F}_{\checkmark} (g) \implies \mathbb{F}_{\checkmark} (\lambda x. f x \parallel\parallel g x) \rangle$

by (*fact finite-ticks-fun-Sync*)⁺

lemma *finite-ticks-fun-GlobalNdet* [*finite-ticks-fun-simps*] :

$\langle \text{finite } A \implies (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} (f a)) \implies \mathbb{F}_{\checkmark} (\lambda x. \bigwedge a \in A. f a x) \rangle$

by (*simp add: finite-ticks-GlobalNdet finite-ticks-fun-def*)

lemma *finite-ticks-fun-GlobalDet* :

$\langle \text{finite } A \implies (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} (f a)) \implies \mathbb{F}_{\checkmark} (\lambda x. \Box a \in A. f a x) \rangle$

by (*simp add: finite-ticks-GlobalDet finite-ticks-fun-def*)

lemma *finite-ticks-fun-MultiSeq* [*finite-ticks-fun-simps*] :

$\langle L = [] \implies \mathbb{F}_{\checkmark} (\lambda x. \text{SEQ } l \in @L. f l x) \rangle$

$\langle L \neq [] \implies \mathbb{F}_{\checkmark} (f (\text{last } L)) \implies \mathbb{F}_{\checkmark} (\lambda x. \text{SEQ } l \in @L. f l x) \rangle$

by (*simp-all add: finite-ticks-MultiSeq-nonempty finite-ticks-fun-def finite-ticks-SKIP*)

lemma *finite-ticks-fun-MultiSync* [*finite-ticks-fun-simps*] :

$\langle (\bigwedge m. m \in \# M \implies \mathbb{F}_{\checkmark} (f m)) \implies \mathbb{F}_{\checkmark} (\lambda x. \llbracket S \rrbracket m \in \# M. f m x) \rangle$

by (*simp add: finite-ticks-MultiSync finite-ticks-fun-def*)

corollary $\langle (\bigwedge m. m \in \# M \implies \mathbb{F}_{\checkmark} (f m)) \implies \mathbb{F}_{\checkmark} (\lambda x. \parallel m \in \# M. f m x) \rangle$

and $\langle (\bigwedge m. m \in \# M \implies \mathbb{F}_{\checkmark} (f m)) \implies \mathbb{F}_{\checkmark} (\lambda x. \parallel\parallel m \in \# M. f m x) \rangle$

by (*fact finite-ticks-fun-MultiSync*)⁺

lemma *finite-ticks-fun-Mprefix-iff* :

$\langle \mathbb{F}_{\checkmark} (\lambda x. \Box a \in A \rightarrow f a x) \longleftrightarrow (\forall a \in A. \mathbb{F}_{\checkmark} (f a)) \rangle$

by (*auto simp add: finite-ticks-fun-def finite-ticks-Mprefix-iff*)

lemma *finite-ticks-fun-Mprefix* [*finite-ticks-fun-simps*] :

$\langle (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} (f a)) \implies \mathbb{F}_{\checkmark} (\lambda x. \Box a \in A \rightarrow f a x) \rangle$

by (*simp add: finite-ticks-fun-Mprefix-iff*)

lemma *finite-ticks-fun-Mndetprefix-iff* [*finite-ticks-fun-simps*] :

$\langle \mathbb{F}_{\checkmark} (\lambda x. \bigwedge a \in A \rightarrow f a x) \longleftrightarrow (\forall a \in A. \mathbb{F}_{\checkmark} (f a)) \rangle$

by (*auto simp add: finite-ticks-fun-def finite-ticks-Mndetprefix-iff*)

lemma *finite-ticks-fun-Mndetprefix* [*finite-ticks-fun-simps*] :
 $\langle (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} (f a)) \implies \mathbb{F}_{\checkmark} (\lambda x. \bigwedge a \in A \rightarrow f a x) \rangle$
by (*simp add: finite-ticks-fun-Mndetprefix-iff*)

lemma *finite-ticks-fun-write0-iff* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} (\lambda x. a \rightarrow f x) \longleftrightarrow \mathbb{F}_{\checkmark} (f) \rangle$
by (*simp add: write0-def finite-ticks-fun-Mprefix-iff*)

lemma *finite-ticks-fun-write-iff* [*finite-ticks-fun-simps*] :
 $\langle \mathbb{F}_{\checkmark} (\lambda x. c!a \rightarrow f x) \longleftrightarrow \mathbb{F}_{\checkmark} (f) \rangle$
by (*simp add: write-def finite-ticks-fun-Mprefix-iff*)

lemma *finite-ticks-fun-read-iff* :
 $\langle \mathbb{F}_{\checkmark} (\lambda x. c?a \in A \rightarrow f a x) \longleftrightarrow (\forall b \in c \text{ ' } A. \mathbb{F}_{\checkmark} (f (inv\text{-}into A c b))) \rangle$
by (*simp add: read-def finite-ticks-fun-Mprefix-iff*)

lemma *finite-ticks-fun-read* [*finite-ticks-fun-simps*] :
 $\langle (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} (\lambda x. f a x)) \implies \mathbb{F}_{\checkmark} (\lambda x. c?a \in A \rightarrow f a x) \rangle$
by (*simp add: read-def o-def inv-into-into finite-ticks-fun-Mprefix-iff*)

lemma *finite-ticks-fun-ndet-write-iff* :
 $\langle \mathbb{F}_{\checkmark} (\lambda x. c!!a \in A \rightarrow f a x) \longleftrightarrow (\forall b \in c \text{ ' } A. \mathbb{F}_{\checkmark} (f (inv\text{-}into A c b))) \rangle$
by (*simp add: ndet-write-def finite-ticks-fun-Mndetprefix-iff*)

lemma *finite-ticks-fun-ndet-write* [*finite-ticks-fun-simps*] :
 $\langle (\bigwedge a. a \in A \implies \mathbb{F}_{\checkmark} (\lambda x. f a x)) \implies \mathbb{F}_{\checkmark} (\lambda x. c!!a \in A \rightarrow f a x) \rangle$
by (*simp add: ndet-write-def o-def inv-into-into finite-ticks-fun-Mndetprefix-iff*)

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 $'r \Rightarrow ('a, 'r) \text{ process}_{ptick}$ (instead of $('a, 'r) \text{ process}_{ptick}$).

lift-definition $Seq_{ptick} ::$

$\langle [('a, 'r) \text{ process}_{ptick}, 'r \Rightarrow ('a, 'r) \text{ process}_{ptick}] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$ (**infixl** $\langle ;, \checkmark \rangle$ 74)

is $\langle \lambda P \ Q. (\{(t, X) \mid t \ X. (t, X \cup \text{range tick}) \in \mathcal{F} \ P \wedge tF \ t\} \cup$
 $\{(t @ u, X) \mid t \ u \ r \ X. t @ [\checkmark(r)] \in \mathcal{T} \ P \wedge (u, X) \in \mathcal{F} \ (Q \ r)\} \cup$
 $\{(t, X). t \in \mathcal{D} \ P\},$
 $\mathcal{D} \ P \cup \{t @ u \mid t \ u \ r. t @ [\checkmark(r)] \in \mathcal{T} \ P \wedge u \in \mathcal{D} \ (Q \ r)\}) \rangle$

oops

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 $'r \Rightarrow ('a, 'r) \text{ process}_{ptick}$ while we would like to have in full generality $'r \Rightarrow ('a, 's) \text{ process}_{ptick}$. 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 $\text{map } (ev \circ of\text{-}ev)$ allows for changing the type of termination in tick-free traces.

lemma *tickFree-map-ev-of-ev-same-type-is* : $\langle tF \ t \implies \text{map } (ev \circ of\text{-}ev) \ t = t \rangle$

— In this case the type of termination remains unchanged.

by $(\text{induct } t) \text{ simp-all}$

lemma *tickFree-map-ev-of-ev-eq-iff* :
 $\langle tF\ t \implies \text{map}\ (ev \circ of\text{-}ev)\ t = t' \implies t = \text{map}\ (ev \circ of\text{-}ev)\ t' \rangle$
by (*induct t arbitrary: t'*) *auto*

lemma *tickFree-map-ev-of-ev-inj* :
 $\langle tF\ t \implies tF\ t' \implies \text{map}\ (ev \circ of\text{-}ev)\ t = \text{map}\ (ev \circ of\text{-}ev)\ t' \longleftrightarrow t = t' \rangle$
by (*induct t arbitrary: t'*) (*use event_{ptick}.expand in auto*)**+**

lemma *map-ev-of-ev-map-ev-of-ev [simp]* :
 $\langle \text{map}\ (ev \circ of\text{-}ev)\ (\text{map}\ (ev \circ of\text{-}ev)\ t) = \text{map}\ (ev \circ of\text{-}ev)\ t \rangle$ **by** *simp*

lemma *map-ev-of-ev-map-ev-of-ev-simplified [simp]* :
 $\langle \text{map}\ (ev \circ of\text{-}ev \circ (ev \circ of\text{-}ev))\ t = \text{map}\ (ev \circ of\text{-}ev)\ t \rangle$ **by** *simp*

lemma *tickFree-map-ev-of-ev-eq-imp-ev-mem-iff* :
 $\langle tF\ t' \implies t = \text{map}\ (ev \circ of\text{-}ev)\ t' \implies ev\ a \in \text{set}\ t \longleftrightarrow ev\ a \in \text{set}\ t' \rangle$
by (*induct t' arbitrary: t*) *auto*

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

definition *ref-Seq_{ptick}* :: $\langle ('a, 'r)\ event_{ptick}\ set \Rightarrow ('a, 's)\ event_{ptick}\ set \rangle$
where $\langle \text{ref-Seq}_{ptick}\ X \equiv (ev \circ of\text{-}ev)\ (X \cap \text{range ev}) \cup \text{range tick} \rangle$

lemma *ref-Seq_{ptick}-same-type-is* : $\langle \text{ref-Seq}_{ptick}\ X = X \cup \text{range tick} \rangle$
— In this case the type of termination remains unchanged.
by (*auto simp add: ref-Seq_{ptick}-def set-eq-iff image-iff*)
(*metis Int-iff event_{ptick}.exhaust event_{ptick}.sel(1) rangeI*)

lemma *mono-ref-Seq_{ptick}* : $\langle X \subseteq Y \implies \text{ref-Seq}_{ptick}\ X \subseteq \text{ref-Seq}_{ptick}\ Y \rangle$
unfolding *ref-Seq_{ptick}-def* **by** *fast*

lemma *ref-Seq_{ptick}-idem* : $\langle \text{ref-Seq}_{ptick}\ (\text{ref-Seq}_{ptick}\ X) = \text{ref-Seq}_{ptick}\ X \rangle$
by (*auto simp add: image-iff ref-Seq_{ptick}-def*)
(*metis Int-iff event_{ptick}.sel(1) rangeI,*
metis (lifting) Int-iff Un-iff event_{ptick}.sel(1) image-eqI rangeI)

lemma *ref-Seq_{ptick}-comp-ref-Seq_{ptick}* : $\langle \text{ref-Seq}_{ptick} \circ \text{ref-Seq}_{ptick} = \text{ref-Seq}_{ptick} \rangle$
by (*rule ext*) (*simp add: ref-Seq_{ptick}-idem*)

lemma *ref-Seq_{ptick}-eq-iff* :
 $\langle \text{ref-Seq}_{ptick}\ X = \text{ref-Seq}_{ptick}\ Y \longleftrightarrow X \cap \text{range ev} = Y \cap \text{range ev} \rangle$
proof (*rule iffI*)

show $\langle X \cap \text{range } ev = Y \cap \text{range } ev \implies \text{ref-Seq}_{ptick} X = \text{ref-Seq}_{ptick} Y \rangle$
by (*auto simp add: ref-Seq_{ptick}-def*)
next
show $\langle X \cap \text{range } ev = Y \cap \text{range } ev \rangle$ **if** $\langle \text{ref-Seq}_{ptick} X = \text{ref-Seq}_{ptick} Y \rangle$
proof (*rule set-eqI*)
show $\langle e \in X \cap \text{range } ev \longleftrightarrow e \in Y \cap \text{range } ev \rangle$ **for** e
using *that[unfolded set-eq-iff, THEN spec, of $\langle (ev \circ of-ev) e \rangle$]*
by (*auto simp add: ref-Seq_{ptick}-def*)
qed
qed

lemma *ref-Seq_{ptick}-is-map-event_{ptick}-image* :
 $\langle \text{ref-Seq}_{ptick} X = \text{map-event}_{ptick} \text{id } g \text{ '}(X \cap \text{range } ev) \cup \text{range tick}' \rangle$
 — Note that g is free here and does not matter.
by (*auto simp add: ref-Seq_{ptick}-def image-iff*)
 (*metis Int-iff eq-id-iff map-event_{ptick}-eq-ev-iff rangeI,*
metis Int-iff event_{ptick}.sel(1) rangeI)

lemma *ref-Seq_{ptick}-union-image-ev* :
 $\langle \text{ref-Seq}_{ptick} (X \cup ev \text{ ' } S) = \text{ref-Seq}_{ptick} X \cup ev \text{ ' } S \rangle$
by (*auto simp add: ref-Seq_{ptick}-def image-iff*)
 (*metis Int-iff Un-iff event_{ptick}.sel(1) image-eqI rangeI*)

lemma *ref-Seq_{ptick}-UNIV* : $\langle \text{ref-Seq}_{ptick} UNIV = UNIV \rangle$
by (*simp add: set-eq-iff ref-Seq_{ptick}-def image-iff*)
 (*meson event_{ptick}.exhaust*)

3.1.2 Formal Definition

definition *div-Seq_{ptick}* ::
 $\langle [(\text{'a}, \text{'r}) \text{ process}_{ptick}, \text{'r} \Rightarrow (\text{'a}, \text{'s}) \text{ process}_{ptick}] \Rightarrow (\text{'a}, \text{'s}) \text{ trace}_{ptick} \text{ set} \rangle$
where $\langle \text{div-Seq}_{ptick} P Q \equiv$
 $\{ \text{map } (ev \circ of-ev) t @ u \mid t u. t \in \mathcal{D} P \wedge tF t \wedge ftF u \} \cup$
 $\{ \text{map } (ev \circ of-ev) t @ u \mid t u r. t @ [\checkmark(r)] \in \mathcal{T} P \wedge tF t \wedge u \in \mathcal{D} (Q r) \} \rangle$

definition *fail-Seq_{ptick}* ::
 $\langle [(\text{'a}, \text{'r}) \text{ process}_{ptick}, \text{'r} \Rightarrow (\text{'a}, \text{'s}) \text{ process}_{ptick}] \Rightarrow (\text{'a}, \text{'s}) \text{ failure}_{ptick} \text{ set} \rangle$
where $\langle \text{fail-Seq}_{ptick} P Q \equiv$
 $\{ (\text{map } (ev \circ of-ev) t, X) \mid t X. (t, \text{ref-Seq}_{ptick} X) \in \mathcal{F} P \wedge tF t \} \cup$
 $\{ (\text{map } (ev \circ of-ev) t @ u, X) \mid t u r X. t @ [\checkmark(r)] \in \mathcal{T} P \wedge tF t \wedge (u, X) \in \mathcal{F} (Q r) \} \cup$
 $\{ (\text{map } (ev \circ of-ev) t @ u, X) \mid t u X. t \in \mathcal{D} P \wedge tF t \wedge ftF u \}$
 — $tF t$ is trivial when $t @ [\checkmark(r)] \in \mathcal{T} P$, but we add it for proof automation

lift-definition *Seq_{ptick}* ::
 $\langle [(\text{'a}, \text{'r}) \text{ process}_{ptick}, \text{'r} \Rightarrow (\text{'a}, \text{'s}) \text{ process}_{ptick}] \Rightarrow (\text{'a}, \text{'s}) \text{ process}_{ptick} \rangle$ (**infixl**
 $\langle ;\checkmark \rangle$ 74)

```

is  $\langle \lambda P Q. (fail-Seq_{ptick} P Q, div-Seq_{ptick} P Q) \rangle$ 
proof -
  show  $\langle ?thesis P Q \rangle$  (is  $\langle is-process(?f, ?d) \rangle$ ) for  $P$  and  $Q$ 
  proof (unfold is-process-def FAILURES-def DIVERGENCES-def fst-conv snd-conv,
    intro conjI allI impI)
    show  $\langle [], \{\} \rangle \in ?f$ 
    by (simp add: fail-Seqptick-def ref-Seqptick-def)
      (metis append-Nil is-processT1 trace-tick-continuation-or-all-tick-failuresE)
  next
    show  $\langle (t, X) \in ?f \implies ftF t \rangle$  for  $t X$ 
    by (auto simp add: fail-Seqptick-def div-Seqptick-def
      F-imp-front-tickFree D-imp-front-tickFree
      intro: front-tickFree-append)
  next
    show  $\langle (t @ u, \{\}) \in ?f \implies (t, \{\}) \in ?f \rangle$  for  $t u$ 
    proof (induct u arbitrary: t)
      show  $\langle (t @ [], \{\}) \in ?f \implies (t, \{\}) \in ?f \rangle$  for  $t$  by simp
    next
      fix  $t e u$  assume prem :  $\langle (t @ e \# u, \{\}) \in ?f \rangle$ 
      assume hyp :  $\langle (t @ u, \{\}) \in ?f \implies (t, \{\}) \in ?f \rangle$  for  $t$ 
      from prem have  $\langle (t @ [e] @ u, \{\}) \in ?f \rangle$  by simp
      with hyp have  $\langle (t @ [e], \{\}) \in ?f \rangle$  by presburger
      then consider (D-P)  $t' u$  where  $\langle t @ [e] = \text{map } (ev \circ of-ev) t' @ u \rangle \langle t' \in$ 
 $\mathcal{D} P \rangle \langle tF t' \rangle \langle ftF u \rangle$ 
      | (F-P)  $t' \text{ where } \langle t @ [e] = \text{map } (ev \circ of-ev) t' \rangle \langle (t', \text{range tick}) \in \mathcal{F} P \rangle$ 
 $\langle tF t' \rangle$ 
      | (F-Q)  $t' r u \text{ where } \langle t @ [e] = \text{map } (ev \circ of-ev) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T}$ 
 $P \rangle \langle (u, \{\}) \in \mathcal{F} (Q r) \rangle$ 
      by (auto simp add: fail-Seqptick-def div-Seqptick-def ref-Seqptick-def)
      thus  $\langle (t, \{\}) \in ?f \rangle$ 
    proof cases
      case D-P
      show ?thesis
      proof (cases u rule: rev-cases)
        assume  $\langle u = [] \rangle$ 
        have  $\langle \text{butlast } t', \{\} \rangle \in \mathcal{F} P$ 
        by (metis D-P(2) D-T prefixI T-F-spec append-butlast-last-id
          butlast.simps(1) is-processT3-TR)
        thus  $\langle (t, \{\}) \in ?f \rangle$ 
        by (elim trace-tick-continuation-or-all-tick-failuresE, simp-all add:
          fail-Seqptick-def ref-Seqptick-def)
          (metis (no-types, opaque-lifting) D-P(1)  $\langle u = [] \rangle$  append.right-neutral
            append-T-imp-tickFree butlast-snoc is-processT1 map-butlast not-Cons-self2,
            metis D-P(1,3)  $\langle u = [] \rangle$  append.right-neutral butlast-snoc
            front-tickFree-iff-tickFree-butlast map-butlast tickFree-imp-front-tickFree)
      next
        fix  $e' u'$  assume  $\langle u = u' @ [e'] \rangle$ 
        with D-P have  $\langle t = \text{map } (ev \circ of-ev) t' @ u' \rangle \langle t' \in \mathcal{D} P \rangle \langle tF t' \rangle \langle ftF u' \rangle$ 
        by (simp-all add: front-tickFree-append-iff)

```



```

      thus  $\langle (t, \{\}) \in ?f \rangle$  by (auto simp add: fail-Seqptick-def)
    qed
  next
    case F-P
    have  $\langle (\text{butlast } t', \{\}) \in \mathcal{F} P \rangle$ 
      by (metis F-P(1, 2) is-processT3 is-processT4-empty list.map-disc-iff
        snoc-eq-iff-butlast)
    with F-P(2) show  $\langle (t, \{\}) \in ?f \rangle$ 
      by (elim trace-tick-continuation-or-all-tick-failuresE, simp-all add: fail-Seqptick-def
        ref-Seqptick-def)
        (metis (no-types, lifting) F-P(1) T-imp-front-tickFree append.right-neutral
        butlast-snoc
        front-tickFree-iff-tickFree-butlast is-processT1 map-butlast,
        metis F-P(1) F-imp-front-tickFree butlast-snoc front-tickFree-iff-tickFree-butlast
        map-butlast)
    next
      case F-Q
      show  $\langle (t, \{\}) \in ?f \rangle$ 
      proof (cases u rule: rev-cases)
        assume  $\langle u = [] \rangle$ 
        have  $\langle (\text{butlast } t', \{\}) \in \mathcal{F} P \rangle$ 
          by (metis F-Q(2) T-F-spec append-butlast-last-id butlast.simps(1)
            is-processT3-TR-append)
        thus  $\langle (t, \{\}) \in ?f \rangle$ 
          by (elim trace-tick-continuation-or-all-tick-failuresE, simp-all add:
            fail-Seqptick-def ref-Seqptick-def)
            (metis (no-types, lifting) F-Q(1)  $\langle u = [] \rangle$  append-T-imp-tickFree
            butlast-snoc
            is-processT1 map-butlast not-Cons-self2 self-append-conv,
            metis F-Q(1, 2) T-imp-front-tickFree  $\langle u = [] \rangle$  append-self-conv
            butlast-snoc
            front-tickFree-iff-tickFree-butlast is-processT3-TR-append map-butlast)
      next
        from F-Q show  $\langle u = u' @ [e] \implies (t, \{\}) \in ?f \rangle$  for  $u' e'$ 
          by (simp add: fail-Seqptick-def)
          (metis append-T-imp-tickFree is-processT3 list.distinct(1))
      qed
    qed
  next
    fix t X Y assume  $\langle (t, Y) \in ?f \wedge X \subseteq Y \rangle$ 
    hence  $\langle (t, Y) \in ?f \rangle \langle X \subseteq Y \rangle$  by simp-all
    from  $\langle (t, Y) \in ?f \rangle$  consider (F-P)  $t'$  where  $\langle t = \text{map } (ev \circ \text{of-ev}) t' \rangle$ 
       $\langle (t', (ev \circ \text{of-ev}) ' (Y \cap \text{range } ev) \cup \text{range tick}) \in \mathcal{F} P \rangle \langle tF t' \rangle$ 
      | (F-Q)  $t' r u$  where  $\langle t = \text{map } (ev \circ \text{of-ev}) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF$ 
 $t' \rangle \langle (u, Y) \in \mathcal{F} (Q r) \rangle$ 
      | (D-P)  $t' u$  where  $\langle t = \text{map } (ev \circ \text{of-ev}) t' @ u \rangle \langle t' \in \mathcal{D} P \rangle \langle tF t' \rangle \langle tF u \rangle$ 
    by (auto simp add: fail-Seqptick-def ref-Seqptick-def)
    thus  $\langle (t, X) \in ?f \rangle$ 

```

```

proof cases
  case F-P
    from  $\langle X \subseteq Y \rangle$  have  $\langle (ev \circ of-ev) \text{ ' } (X \cap range\ ev) \cup range\ tick \subseteq$ 
       $(ev \circ of-ev) \text{ ' } (Y \cap range\ ev) \cup range\ tick \rangle$  by blast
    with  $F-P(2)$  have  $\langle (t', (ev \circ of-ev) \text{ ' } (X \cap range\ ev) \cup range\ tick) \in \mathcal{F}\ P \rangle$ 
      by (metis is-processT4)
    with  $F-P(1, 3)$  show  $\langle (t, X) \in ?f \rangle$ 
      by (auto simp add: fail-Seqptick-def ref-Seqptick-def)
  next
    case F-Q thus  $\langle (t, X) \in ?f \rangle$ 
      by (simp add: fail-Seqptick-def) (metis  $\langle X \subseteq Y \rangle$  is-processT4)
  next
    case D-P thus  $\langle (t, X) \in ?f \rangle$  by (auto simp add: fail-Seqptick-def)
qed
next
  fix  $t\ X\ Y$  assume  $* : \langle (t, X) \in ?f \wedge (\forall e. e \in Y \longrightarrow (t @ [e], \{\}) \notin ?f) \rangle$ 
  from  $*$  consider  $\langle t \in ?d \rangle$ 
    |  $(F-P)\ t'\ \mathbf{where}\ \langle t = map\ (ev \circ of-ev)\ t' \rangle \langle (t', (ev \circ of-ev) \text{ ' } (X \cap range\ ev) \cup range\ tick) \in \mathcal{F}\ P \rangle \langle tF\ t' \rangle$ 
    |  $(F-Q)\ t'\ r\ u\ \mathbf{where}\ \langle t = map\ (ev \circ of-ev)\ t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T}\ P \rangle \langle tF\ t' \rangle \langle (u, X) \in \mathcal{F}\ (Q\ r) \rangle$ 
  unfolding fail-Seqptick-def div-Seqptick-def ref-Seqptick-def by auto
  thus  $\langle (t, X \cup Y) \in ?f \rangle$ 
proof cases
  show  $\langle t \in ?d \implies (t, X \cup Y) \in ?f \rangle$ 
    by (simp add: div-Seqptick-def fail-Seqptick-def) (metis is-processT8)
  next
    case F-P
      have  $\langle (t', (ev \circ of-ev) \text{ ' } (X \cap range\ ev) \cup range\ tick \cup (ev \circ of-ev) \text{ ' } (Y \cap range\ ev)) \in \mathcal{F}\ P \rangle$ 
      proof (intro is-processT5[OF F-P(2)] allI impI)
        fix  $e :: \langle (a, r)\ event_{ptick} \rangle$  assume  $\langle e \in (ev \circ of-ev) \text{ ' } (Y \cap range\ ev) \rangle$ 
        then obtain  $a\ \mathbf{where}\ \langle e = ev\ a \rangle \langle ev\ a \in Y \rangle$  by auto
        from  $*[THEN\ conjunct2, rule-format, OF\ this(2), unfolded\ fail-Seq_{ptick}-def]$ 
         $F-P(1, 3)$ 
        show  $\langle (t' @ [e], \{\}) \notin \mathcal{F}\ P \rangle$ 
        apply (simp add: fail-Seqptick-def  $\langle e = ev\ a \rangle$  append-eq-map-conv ref-Seqptick-def)
        by (smt (verit, del-insts) append-Nil2 comp-apply eventptick.sel(1) is-processT1
          list.simps(8, 9) map-append tickFree-append-iff
          tickFree-map-ev-comp trace-tick-continuation-or-all-tick-failuresE)
      qed
    also have  $\langle (ev \circ of-ev) \text{ ' } (X \cap range\ ev) \cup range\ tick \cup (ev \circ of-ev) \text{ ' } (Y \cap range\ ev) =$ 
       $ref-Seq_{ptick}\ (X \cup Y) \rangle$  unfolding ref-Seqptick-def by blast
    finally show  $\langle (t, X \cup Y) \in ?f \rangle$ 
      using  $F-P(1, 3)$  by (auto simp add: fail-Seqptick-def)
  next

```

```

case  $F\text{-}Q$ 
from * have  $\langle \forall e. e \in Y \longrightarrow (u @ [e], \{\}) \notin \mathcal{F} (Q\ r) \rangle$ 
  by (simp add: fail-Seqptick-def F-Q(1, 2))
    (metis F-Q(2) append-T-imp-tickFree not-Cons-self2)
  with  $F\text{-}Q(3, 4)$  have  $\langle (u, X \cup Y) \in \mathcal{F} (Q\ r) \rangle$  by (simp add: is-processT5)
with  $F\text{-}Q(1-3)$  show  $\langle (t, X \cup Y) \in ?f \rangle$  by (auto simp add: fail-Seqptick-def)
qed
next
show  $\langle t \in ?d \wedge tF\ t \wedge ftF\ u \implies t @ u \in ?d \rangle$  for  $t\ u$ 
  by (simp add: div-Seqptick-def, elim conjE disjE exE)
    (solves <use front-tickFree-append in auto>,
     meson append.assoc is-processT7 tickFree-append-iff)
next
show  $\langle t \in ?d \implies (t, X) \in ?f \rangle$  for  $t\ X$ 
  by (simp add: div-Seqptick-def fail-Seqptick-def) (metis is-processT8)
next
show * :  $\langle t @ [\checkmark(r')] \in ?d \implies t \in ?d \rangle$  for  $t\ r'$ 
  by (simp add: div-Seqptick-def, elim disjE exE conjE)
    (metis butlast-append butlast-snoc front-tickFree-iff-tickFree-butlast non-tickFree-tick
     tickFree-append-iff tickFree-imp-front-tickFree tickFree-map-ev-comp,
     metis D-imp-front-tickFree butlast-append butlast-snoc
     div-butlast-when-non-tickFree-iff non-tickFree-tick
     tickFree-append-iff tickFree-map-ev-comp)

fix  $t\ r'\ X$  assume  $\langle (t @ [\checkmark(r')], \{\}) \in ?f \rangle$ 
then consider  $\langle t @ [\checkmark(r')] \in ?d \rangle$ 
  | ( $F\text{-}Q$ )  $t'\ r\ u$  where  $\langle t @ [\checkmark(r')] = \text{map } (ev \circ of\text{-}ev) t' @ u \rangle$ 
   $\langle t' @ [\checkmark(r)] \in \mathcal{T}\ P \rangle$   $\langle tF\ t' \rangle$   $\langle (u, X) \in \mathcal{F} (Q\ r) \rangle$ 
  by (auto simp add: fail-Seqptick-def div-Seqptick-def)
    (metis non-tickFree-tick tickFree-append-iff tickFree-map-ev-comp,
     metis F-T F-imp-front-tickFree nonTickFree-n-frontTickFree
     non-tickFree-tick tickFree-append-iff tickFree-map-ev-comp tick-T-F)
thus  $\langle (t, X - \{\checkmark(r')\}) \in ?f \rangle$ 
proof cases
  assume  $\langle t @ [\checkmark(r')] \in ?d \rangle$ 
  with * have  $\langle t \in ?d \rangle$  .
  thus  $\langle (t, X - \{\checkmark(r')\}) \in ?f \rangle$ 
    by (simp add: fail-Seqptick-def div-Seqptick-def) (metis is-processT8)
next
case  $F\text{-}Q$ 
from  $F\text{-}Q(1, 2)$  obtain  $u'$  where  $\langle u = u' @ [\checkmark(r')] \rangle$ 
  by (cases u rule: rev-cases, simp-all)
    (metis non-tickFree-tick tickFree-append-iff tickFree-map-ev-comp)
  with  $F\text{-}Q(4)$  have  $\langle (u', X - \{\checkmark(r')\}) \in \mathcal{F} (Q\ r) \rangle$  by (simp add: F-T
is-processT6-TR)
  with  $F\text{-}Q(1-3)$   $\langle u = u' @ [\checkmark(r')] \rangle$  show  $\langle (t, X - \{\checkmark(r')\}) \in ?f \rangle$ 
    by (auto simp add: fail-Seqptick-def)
qed
qed

```

qed

3.2 Projections

lemma $F\text{-Seq}_{ptick} : \langle \mathcal{F} (P ; \checkmark Q) = \text{fail-Seq}_{ptick} P Q \rangle$
 by (simp add: Failures.rep-eq FAILURES-def Seq_{ptick}.rep-eq)

lemma $D\text{-Seq}_{ptick} : \langle \mathcal{D} (P ; \checkmark Q) = \text{div-Seq}_{ptick} P Q \rangle$
 by (simp add: Divergences.rep-eq DIVERGENCES-def Seq_{ptick}.rep-eq)

lemma $T\text{-Seq}_{ptick}\text{-bis} :$
 $\langle \mathcal{T} (P ; \checkmark Q) = \{ \text{map } (ev \circ of\text{-}ev) t \mid t. (t, \text{range tick}) \in \mathcal{F} P \wedge tF t \} \cup$
 $\{ \text{map } (ev \circ of\text{-}ev) t @ u \mid t u r. t @ [\checkmark(r)] \in \mathcal{T} P \wedge tF t \wedge u \in \mathcal{T} (Q$
 $r) \} \cup$
 $\{ \text{map } (ev \circ of\text{-}ev) t @ u \mid t u. t \in \mathcal{D} P \wedge tF t \wedge ftF u \} \rangle$
 by (auto simp add: Traces.rep-eq TRACES-def F-Seq_{ptick} fail-Seq_{ptick}-def ref-Seq_{ptick}-def
 intro: is-processT4 simp flip: Failures.rep-eq)
 (metis, metis (lifting) image-empty inf-bot-left sup-bot-left, blast)

lemma $T\text{-Seq}_{ptick} :$
 $\langle \mathcal{T} (P ; \checkmark Q) = \{ \text{map } (ev \circ of\text{-}ev) t \mid t. t \in \mathcal{T} P \wedge tF t \} \cup$
 $\{ \text{map } (ev \circ of\text{-}ev) t @ u \mid t u r. t @ [\checkmark(r)] \in \mathcal{T} P \wedge tF t \wedge u \in \mathcal{T} (Q$
 $r) \} \cup$
 $\{ \text{map } (ev \circ of\text{-}ev) t @ u \mid t u. t \in \mathcal{D} P \wedge tF t \wedge ftF u \} \rangle$
 — Often easier to use
 by (auto simp add: T-Seq_{ptick}-bis F-T)
 (metis T-F-spec append.right-neutral is-processT1-TR
 trace-tick-continuation-or-all-tick-failuresE)

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

lemma $\text{mono-Seq}_{ptick}\text{-eq} : \langle P ; \checkmark Q = P' ; \checkmark Q' \rangle \text{ if } * : \langle P = P' \rangle \wedge r. r \in \checkmark s(P)$
 $\implies Q r = Q' r \rangle$
 for $P P' :: \langle 'a, 'r \rangle \text{process}_{ptick} \rangle$ and $Q Q' :: \langle 'r \Rightarrow ('a, 's) \text{process}_{ptick} \rangle$
proof (fold $*(1)$, subst Process-eq-spec-optimized, safe)
 { fix t and $Q Q' :: \langle 'r \Rightarrow ('a, 's) \text{process}_{ptick} \rangle$
 assume $\langle t \in \mathcal{D} (P ; \checkmark Q) \rangle$ and $* : \langle r \in \checkmark s(P) \implies Q r = Q' r \rangle$ for r
 from $\langle t \in \mathcal{D} (P ; \checkmark Q) \rangle$ consider $(D\text{-}P) t' u$ where $\langle t = \text{map } (ev \circ of\text{-}ev) t'$
 $@ u \rangle \langle t' \in \mathcal{D} P \rangle \langle tF t' \rangle \langle ftF u \rangle$
 $\mid (D\text{-}Q) t' r u$ where $\langle t = \text{map } (ev \circ of\text{-}ev) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF$
 $t' \rangle \langle u \in \mathcal{D} (Q r) \rangle$
 unfolding Seq_{ptick}-projs by blast
 hence $\langle t \in \mathcal{D} (P ; \checkmark Q') \rangle$
proof cases
 case D-P thus $\langle t \in \mathcal{D} (P ; \checkmark Q') \rangle$ by (auto simp add: Seq_{ptick}-projs)

```

next
  case D-Q thus  $\langle t \in \mathcal{D} (P ; \checkmark Q') \rangle$ 
    by (simp add: Seqptick-projs)
      (metis * D-imp-front-tickFree is-processT9 strict-ticks-of-memI)
  qed
} note $ = this
show  $\langle t \in \mathcal{D} (P ; \checkmark Q) \implies t \in \mathcal{D} (P ; \checkmark Q') \rangle$ 
  and  $\langle t \in \mathcal{D} (P ; \checkmark Q') \implies t \in \mathcal{D} (P ; \checkmark Q) \rangle$  for t
  by (erule $, simp add: *(2))+
next
{ fix t X and Q Q' ::  $\langle 'r \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$ 
  assume  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q) \rangle$  and same-div :  $\langle \mathcal{D} (P ; \checkmark Q) = \mathcal{D} (P ; \checkmark Q') \rangle$ 
  and * :  $\langle r \in \checkmark s(P) \implies Q r = Q' r \rangle$  for r
  from  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q) \rangle$ 
  consider (F-P) t' where  $\langle t = \text{map} (ev \circ of\text{-}ev) t' \rangle \langle (t', \text{ref-Seq}_{ptick} X) \in \mathcal{F} P \rangle \langle tF t' \rangle$ 
    | (F-Q) t' r u where  $\langle t = \text{map} (ev \circ of\text{-}ev) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF t' \rangle \langle (u, X) \in \mathcal{F} (Q r) \rangle$ 
    | (D-P) t' u where  $\langle t = \text{map} (ev \circ of\text{-}ev) t' @ u \rangle \langle t' \in \mathcal{D} P \rangle \langle tF t' \rangle \langle ftF u \rangle$ 
  unfolding Seqptick-projs by blast
  hence  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q') \rangle$ 
proof cases
  case F-P thus  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q') \rangle$  by (auto simp add: Seqptick-projs)
next
  case F-Q thus  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q') \rangle$ 
    by (simp add: Seqptick-projs)
      (metis * F-imp-front-tickFree is-processT9 strict-ticks-of-memI)
next
  case D-P thus  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q') \rangle$  by (auto simp add: Seqptick-projs)
qed
} note $ = this
show  $\langle \mathcal{D} (P ; \checkmark Q) = \mathcal{D} (P ; \checkmark Q') \implies (t, X) \in \mathcal{F} (P ; \checkmark Q) \implies (t, X) \in \mathcal{F} (P ; \checkmark Q') \rangle$ 
  and  $\langle \mathcal{D} (P ; \checkmark Q) = \mathcal{D} (P ; \checkmark Q') \implies (t, X) \in \mathcal{F} (P ; \checkmark Q') \implies (t, X) \in \mathcal{F} (P ; \checkmark Q) \rangle$  for t X
  by (erule $, simp add: *(2))+
qed

```

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 $(a, r) \text{ process}_{ptick}$ and $r \Rightarrow (a, s) \text{ process}_{ptick}$ respectively, the projections can be rewritten as follows.

lemma *F-Seq_{ptick}-same-type :*

$$\begin{aligned} \langle \mathcal{F} (P ; \checkmark Q) \rangle = & \{ (t, X) \mid t X. (t, X \cup \text{range tick}) \in \mathcal{F} P \wedge tF t \} \cup \\ & \{ (t @ u, X) \mid t u r X. t @ [\checkmark(r)] \in \mathcal{T} P \wedge (u, X) \in \mathcal{F} (Q r) \} \cup \\ & \{ (t, X). t \in \mathcal{D} P \} \end{aligned}$$

by (auto simp add: Seq_{ptick}-projs tickFree-map-ev-of-ev-same-type-is ref-Seq_{ptick}-same-type-is is-processT7)

(metis tickFree-map-ev-of-ev-same-type-is,
metis append-T-imp-tickFree not-Cons-self2,
metis D-T T-imp-front-tickFree T-nonTickFree-imp-decomp append.right-neutral
front-tickFree-nonempty-append-imp is-processT9 not-Cons-self2
tickFree-Nil tickFree-imp-front-tickFree)

lemma $D\text{-Seq}_{ptick}\text{-same-type} : \langle \mathcal{D} (P ; \checkmark Q) = \mathcal{D} P \cup \{t @ u \mid t u r. t @ [\checkmark(r)] \in \mathcal{T} P \wedge u \in \mathcal{D} (Q r)\} \rangle$
by (auto simp add: Seq_{ptick}-projs tickFree-map-ev-of-ev-same-type-is is-processT7)
(blast,
metis D-imp-front-tickFree butlast-snoc div-butlast-when-non-tickFree-iff
front-tickFree-charn front-tickFree-nonempty-append-imp
self-append-conv tickFree-Nil tickFree-map-ev-of-ev-same-type-is,
metis append-T-imp-tickFree not-Cons-self2)

lemma $T\text{-Seq}_{ptick}\text{-same-type-bis} :$
 $\langle \mathcal{T} (P ; \checkmark Q) = \{t. (t, \text{range tick}) \in \mathcal{F} P \wedge tF t\} \cup$
 $\{t @ u \mid t u r. t @ [\checkmark(r)] \in \mathcal{T} P \wedge u \in \mathcal{T} (Q r)\} \cup$
 $\mathcal{D} P \rangle$
by (auto simp add: Traces.rep-eq TRACES-def F-Seq_{ptick}-same-type simp flip:
Failures.rep-eq)
(meson is-processT4 sup-ge2, meson is-processT5-S7', blast)

lemma $T\text{-Seq}_{ptick}\text{-same-type} :$
 $\langle \mathcal{T} (P ; \checkmark Q) = \{t \in \mathcal{T} P. tF t\} \cup \{t @ u \mid t u r. t @ [\checkmark(r)] \in \mathcal{T} P \wedge u \in \mathcal{T} (Q r)\} \cup \mathcal{D} P \rangle$
— Often easier to use
by (auto simp add: T-Seq_{ptick}-same-type-bis F-T)
(metis T-F-spec append.right-neutral is-processT1-TR
trace-tick-continuation-or-all-tick-failuresE)

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

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* $(([\checkmark(r)], [\checkmark(s)]), \text{range tick} \cup \text{ev} \text{ ' } A)$.

lemma $\langle r \neq s \implies \neg t \text{ setinterleaves } (([\checkmark(r)], [\checkmark(s)]), \text{range tick} \cup \text{ev} \text{ ' } A) \rangle$ *by simp*

The actual issue of this previous definition is that no distinction is done between the “regular” events (like $\text{ev } a$) and the terminations (like $\checkmark(r)$). Here, while we still want the same behaviour for regular events, we want instead the interleaving of $\checkmark(r)$ and $\checkmark(s)$ to be $\checkmark((r, s))$. 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 $'r \Rightarrow 's \Rightarrow 't \text{ option}$ whose role is to specify how two ticks can be combined (or not).

bundle *option-type-syntax*
begin

no-notation *floor* $(\langle (\langle \text{open-block notation} = \langle \text{mixfix floor} \rangle \rangle [-]) \rangle)$
no-notation *ceiling* $(\langle (\langle \text{open-block notation} = \langle \text{mixfix ceiling} \rangle \rangle [-]) \rangle)$

notation *Some* $(\langle (\langle \text{open-block notation} = \langle \text{mixfix Some} \rangle \rangle [-]) \rangle)$
notation *the* $(\langle (\langle \text{open-block notation} = \langle \text{mixfix the} \rangle \rangle [-]) \rangle)$
notation *None* $(\langle \Diamond \rangle)$

end

unbundle *option-type-syntax*

4.1.2 Definition

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

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

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

| $\text{tick-setinterleaving}_{ptick}\text{-Nil} :$
 $\langle \text{setinterleaving}_{ptick} (\text{tick-join}, \checkmark(r) \# u, A, []) = \{ \} \rangle$

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

| $\text{Nil-setinterleaving}_{ptick}\text{-tick} :$
 $\langle \text{setinterleaving}_{ptick} (\text{tick-join}, [], A, \checkmark(s) \# v) = \{ \} \rangle$

| $\text{ev-setinterleaving}_{ptick}\text{-ev} :$
 $\langle \text{setinterleaving}_{ptick} (\text{tick-join}, \text{ev } a \# u, A, \text{ev } b \# v) =$
 $(\text{ if } a \in A$
 $\text{ then if } b \in A$
 $\text{ then if } a = b$
 $\text{ then } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, u, A, v) \}$
 $\text{ else } \{ \}$
 $\text{ else } \{ \text{ev } b \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, \text{ev } a \# u, A, v) \}$
 $\text{ else if } b \in A$
 $\text{ then } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, u, A, \text{ev } b \# v) \}$
 $\text{ else } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, u, A, \text{ev } b \# v) \} \cup$
 $\{ \text{ev } b \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, \text{ev } a \# u, A, v) \}) \rangle$

| $\text{ev-setinterleaving}_{ptick}\text{-tick} :$
 $\langle \text{setinterleaving}_{ptick} (\text{tick-join}, \text{ev } a \# u, A, \checkmark(s) \# v) =$
 $(\text{ if } a \in A \text{ then } \{}$
 $\text{ else } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, u, A, \checkmark(s) \# v) \}) \rangle$

| $\text{tick-setinterleaving}_{ptick}\text{-ev} :$
 $\langle \text{setinterleaving}_{ptick} (\text{tick-join}, \checkmark(r) \# u, A, \text{ev } b \# v) =$
 $(\text{ if } b \in A \text{ then } \{}$
 $\text{ else } \{ \text{ev } b \# t \mid t. t \in \text{setinterleaving}_{ptick} (\text{tick-join}, \checkmark(r) \# u, A, v) \}) \rangle$

| $\text{tick-setinterleaving}_{ptick}\text{-tick} :$
 $\langle \text{setinterleaving}_{ptick} (\text{tick-join}, \checkmark(r) \# u, A, \checkmark(s) \# v) =$
 $(\text{ case tick-join } r \text{ s}$

of $\lfloor r-s \rfloor \Rightarrow \{\checkmark(r-s) \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, u, A, v)\}$
 $\mid \Diamond \Rightarrow \{\}\rangle$

lemmas $\text{setinterleaving}_{ptick}\text{-induct}$

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

lemma $\text{Cons-setinterleaving}_{ptick}\text{-Nil} :$

$\langle \text{setinterleaving}_{ptick}(\text{tick-join}, e \# u, A, []) =$
 $(\text{case } e \text{ of } \text{ev } a \Rightarrow$
 $(\text{if } a \in A \text{ then } \{\}$
 $\text{else } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, u, A, []) \})$
 $\mid \checkmark(r) \Rightarrow \{\} \rangle$
by $(\text{cases } e) \text{ simp-all}$

lemma $\text{Nil-setinterleaving}_{ptick}\text{-Cons} :$

$\langle \text{setinterleaving}_{ptick}(\text{tick-join}, [], A, e \# v) =$
 $(\text{case } e \text{ of } \text{ev } a \Rightarrow$
 $(\text{if } a \in A \text{ then } \{\}$
 $\text{else } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, [], A, v) \})$
 $\mid \checkmark(r) \Rightarrow \{\} \rangle$
by $(\text{cases } e) \text{ simp-all}$

lemma $\text{Cons-setinterleaving}_{ptick}\text{-Cons} :$

$\langle \text{setinterleaving}_{ptick}(\text{tick-join}, e \# u, A, f \# v) =$
 $(\text{case } e \text{ of } \text{ev } a \Rightarrow$
 $(\text{case } f \text{ of } \text{ev } b \Rightarrow$
 $\text{if } a \in A$
 $\text{then if } b \in A$
 $\text{then if } a = b$
 $\text{then } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, u, A, v) \}$
 $\text{else } \{\}$
 $\text{else } \{ \text{ev } b \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, \text{ev } a \# u, A, v) \}$
 $\text{else if } b \in A$
 $\text{then } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, u, A, \text{ev } b \# v) \}$
 $\text{else } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, u, A, \text{ev } b \# v) \} \cup$
 $\{ \text{ev } b \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, \text{ev } a \# u, A, v) \}$
 $\mid \checkmark(s) \Rightarrow \text{if } a \in A \text{ then } \{\}$
 $\text{else } \{ \text{ev } a \# t \mid t. t \in \text{setinterleaving}_{ptick}(\text{tick-join}, u, A, \checkmark(s)$
 $\# v) \})$
 $\mid \checkmark(r) \Rightarrow$
 $(\text{case } f \text{ of } \text{ev } b \Rightarrow$

if $b \in A$ then $\{\}$
 else $\{ev\ b \# t \mid t. t \in setinterleaving_{ptick} (tick\text{-}join, \checkmark(r) \# u, A, v)\}$
 $\mid \checkmark(s) \Rightarrow$
 $(case\ tick\text{-}join\ r\ s\ of\ [r\text{-}s] \Rightarrow$
 $\{\checkmark(r\text{-}s) \# t \mid t. t \in setinterleaving_{ptick} (tick\text{-}join, u, A, v)\}$
 $\mid \Diamond \Rightarrow \{\})\})\rangle$
 by (cases e ; cases f) simp-all

lemmas $setinterleaving_{ptick}\text{-sims} =$
 $Cons\text{-}setinterleaving_{ptick}\text{-}Nil\ Nil\text{-}setinterleaving_{ptick}\text{-}Cons\ Cons\text{-}setinterleaving_{ptick}\text{-}Cons$

abbreviation $setinterleaves_{ptick} ::$
 $\langle [('a, 't)\ trace_{ptick}, ('r \Rightarrow 's \Rightarrow 't)\ option,$
 $('a, 'r)\ trace_{ptick}, ('a, 's)\ trace_{ptick}, 'a\ set] \Rightarrow bool \rangle$
 $\langle (- / (setinterleaves_{\checkmark}) - / '((-) (-, -) (-, -)) \rangle [63, 0, 0, 0, 0] 64)$
where $\langle t\ setinterleaves_{\checkmark}\ tick\text{-}join\ ((u, v), A) \equiv$
 $t \in setinterleaving_{ptick} (tick\text{-}join, u, A, v) \rangle$

4.1.3 First Properties

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

theorem $setinterleaves\text{-}is\text{-}setinterleaves_{ptick} :$
 $\langle t\ setinterleaves\ ((u, v), range\ tick \cup ev\ 'A) \longleftrightarrow$
 $t\ setinterleaves_{\checkmark}\lambda r\ s.\ if\ r = s\ then\ [r]\ else\ \Diamond\ ((u, v), A) \rangle$
for $t :: \langle ('a, 'r)\ trace_{ptick} \rangle$
by (induct $\langle (\lambda r :: 'r. \lambda s :: 'r. if\ r = s\ then\ [r]\ else\ \Diamond, u, A, v) \rangle$
 arbitrary: $t\ u\ v$) (simp-all add: image-iff)

corollary $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}\text{-}unit :$
 $\langle t\ setinterleaves\ ((u, v), insert\ \checkmark\ (ev\ 'A)) \longleftrightarrow$
 $t\ setinterleaves_{\checkmark}\lambda r\ s.\ [r]\ ((u, v), A) \rangle$ (is $\langle ?lhs \longleftrightarrow ?rhs \rangle$)

proof –

have $\langle ?lhs \longleftrightarrow t\ setinterleaves\ ((u, v), range\ tick \cup ev\ 'A) \rangle$
by (simp add: UNIV-unit)
also have $\langle \dots \longleftrightarrow ?rhs \rangle$
by (simp add: setinterleaves-is-setinterleaves $_{ptick}$)
finally show $\langle ?lhs \longleftrightarrow ?rhs \rangle$.

qed

lemma $setinterleaves_{ptick}\text{-}sym :$

— Of course not suitable for simplifier.

$\langle t \text{ setinterleaves} \checkmark_{\lambda s r. \text{ tick-join } r s} ((v, u), A) \longleftrightarrow$
 $t \text{ setinterleaves} \checkmark_{\lambda r s. \text{ tick-join } r s} ((u, v), A) \rangle$
by (induct $\langle (\text{tick-join}, u, A, v) \rangle$ arbitrary: $t u v$) (auto split: option.split)

lemma *setinterleaves_{Pair}-UNIV-iff* :

$\langle t \text{ setinterleaves} \checkmark_{\lambda r s. \lfloor (r, s) \rfloor} ((u, v), \text{UNIV}) \longleftrightarrow$
 $u = \text{map } (\text{map-event}_{\text{ptick}} \text{ id fst}) t \wedge$
 $v = \text{map } (\text{map-event}_{\text{ptick}} \text{ id snd}) t \rangle$ **for** $t :: \langle ('a, 'r \times 's) \text{ trace}_{\text{ptick}} \rangle$
by (induct $\langle (\lambda r :: 'r. \lambda s :: 's. \lfloor (r, s) \rfloor, u, \text{UNIV} :: 'a \text{ set}, v) \rangle$ arbitrary: $t u v$)
 (auto simp add: ev-eq-map-event_{ptick}-iff tick-eq-map-event_{ptick}-iff)

lemma *setinterleaves_{ptick}-empty* :

$\langle t \text{ setinterleaves} \checkmark_{\text{tick-join}} ((u, v), \{\}) \implies$
 $ev a \in \text{set } t \longleftrightarrow ev a \in \text{set } u \vee ev a \in \text{set } v \rangle$
for $u :: \langle ('a, 'r) \text{ trace}_{\text{ptick}} \rangle$
by (induct $\langle (\text{tick-join}, u, \{\}) :: 'a \text{ set}, v \rangle$ arbitrary: $t u v$)
 (auto split: option.split-asm)

lemma *tickFree-setinterleaves_{ptick}-any-tick-join* :

$\langle t \text{ setinterleaves} \checkmark_{\text{tick-join}} ((u, v), A) \longleftrightarrow$
 $t \text{ setinterleaves} \checkmark_{\text{tick-join}'} ((u, v), A) \rangle$
if $\langle tF t \vee tF u \vee tF v \rangle$

proof (rule iffI)

from $\langle tF t \vee tF u \vee tF v \rangle$
show $\langle t \text{ setinterleaves} \checkmark_{\text{tick-join}} ((u, v), A) \implies$
 $t \text{ setinterleaves} \checkmark_{\text{tick-join}'} ((u, v), A) \rangle$
for $\text{tick-join tick-join}'$
by (induct $\langle (\text{tick-join}, u, A, v) \rangle$ arbitrary: $t u v$)
 (auto split: if-split-asm option.split-asm)
thus $\langle t \text{ setinterleaves} \checkmark_{\text{tick-join}'} ((u, v), A) \implies$
 $t \text{ setinterleaves} \checkmark_{\text{tick-join}} ((u, v), A) \rangle$.

qed

lemma *tickFree-setinterleaves_{ptick}-iff* :

$\langle t \text{ setinterleaves} \checkmark_{\text{tick-join}} ((u, v), A) \implies tF t \longleftrightarrow tF u \wedge tF v \rangle$
by (induct $\langle (\text{tick-join}, u, A, v) \rangle$ arbitrary: $t u v$)
 (auto split: if-split-asm option.split-asm)

lemma *setinterleaves_{ptick}-tickFree-imp* :

$\langle tF u \vee tF v \implies t \text{ setinterleaves} \checkmark_{\text{tick-join}} ((u, v), A) \implies tF t \wedge tF u \wedge tF v \rangle$
by (induct $\langle (\text{tick-join}, u, A, v) \rangle$ arbitrary: $t u v$)

(*auto split: if-split-asm*)

lemma *setinterleaves_{ptick}-NilL-iff* :
 $\langle t \text{ setinterleaves}_{\text{tick-join}} ((\square, v), A) \longleftrightarrow$
 $tF v \wedge \text{set } v \cap \text{ev } \langle A = \{ \} \rangle \wedge t = \text{map ev } (\text{map of-ev } v) \rangle$
for *tick-join* :: $\langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$
by (*induct* $\langle (\text{tick-join}, \square :: ('a, 'r) \text{ trace}_{\text{ptick}}, A, v) \rangle$
arbitrary: t v) (*auto split: if-split-asm*)

lemma *setinterleaves_{ptick}-NilR-iff* :
 $\langle t \text{ setinterleaves}_{\text{tick-join}} ((u, \square), A) \longleftrightarrow$
 $tF u \wedge \text{set } u \cap \text{ev } \langle A = \{ \} \rangle \wedge t = \text{map ev } (\text{map of-ev } u) \rangle$
for *tick-join* :: $\langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$
by (*induct* $\langle (\text{tick-join}, u, A, \square :: ('a, 's) \text{ trace}_{\text{ptick}}) \rangle$
arbitrary: t u) (*auto split: if-split-asm*)

lemma *setinterleaves_{ptick}-subsetL* :
 $\langle tF t \implies \{a. \text{ev } a \in \text{set } u\} \subseteq A \implies$
 $t \text{ setinterleaves}_{\text{tick-join}} ((u, v), A) \implies$
 $t = \text{map ev } (\text{map of-ev } v) \rangle$
by (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: t u v*)
(auto simp add: subset-iff split: if-split-asm option.split-asm)

lemma *setinterleaves_{ptick}-subsetR* :
 $\langle tF t \implies \{a. \text{ev } a \in \text{set } v\} \subseteq A \implies$
 $t \text{ setinterleaves}_{\text{tick-join}} ((u, v), A) \implies$
 $t = \text{map ev } (\text{map of-ev } u) \rangle$
by (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: t u v*)
(auto simp add: subset-iff split: if-split-asm option.split-asm)

lemma *Nil-setinterleaves_{ptick}* :
 $\langle \square \text{ setinterleaves}_{\text{tick-join}} ((u, v), A) \implies u = \square \wedge v = \square \rangle$
by (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: u v*)
(simp-all split: if-split-asm option.split-asm)

lemma *front-tickFree-setinterleaves_{ptick}-iff* :
 $\langle t \text{ setinterleaves}_{\text{tick-join}} ((u, v), A) \implies ftF t \longleftrightarrow ftF u \wedge ftF v \rangle$
proof (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: t u v*)
case *Nil-setinterleaving_{ptick}-Nil* **thus** ?*case* **by** *simp*
next
case (*ev-setinterleaving_{ptick}-Nil* *a u*)
thus ?*case* **by** (*simp add: setinterleaves_{ptick}-NilR-iff split: if-split-asm*)
next
case (*tick-setinterleaving_{ptick}-Nil* *r u*) **thus** ?*case* **by** *simp*
next
case (*Nil-setinterleaving_{ptick}-ev* *b v*)

```

thus ?case by (simp add: setinterleavesptick-NilL-iff split: if-split-asm)
next
  case (Nil-setinterleavingptick-tick s v) thus ?case by simp
next
  case (ev-setinterleavingptick-ev a u b v)
  thus ?case by (simp split: if-split-asm)
    (metis eventptick.disc(1) front-tickFree-Cons-iff front-tickFree-Nil)+
next
  case (ev-setinterleavingptick-tick a u s v)
  thus ?case by (simp split: if-split-asm)
    (metis eventptick.disc(1) front-tickFree-Cons-iff front-tickFree-Nil)
next
  case (tick-setinterleavingptick-ev r u b v)
  thus ?case by (simp split: if-split-asm)
    (metis eventptick.disc(1) front-tickFree-Cons-iff front-tickFree-Nil)
next
  case (tick-setinterleavingptick-tick r u s v) thus ?case
  by (simp split: option.split-asm)
    (metis Nil-setinterleavesptick Nil-setinterleavingptick-Nil
      eventptick.disc(2) front-tickFree-Cons-iff singletonD)
qed

```

lemma setinterleaves_{ptick}-snoc-notinL :

$\langle t \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u, v), A) \implies a \notin A \implies$
 $t @ [ev\ a] \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u @ [ev\ a], v), A) \rangle$
by (induct $\langle (tick\text{-join}, u, A, v) \rangle$ arbitrary: $t\ u\ v$)
(auto split: if-split-asm option.split-asm)

lemma setinterleaves_{ptick}-snoc-notinR :

$\langle t \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u, v), A) \implies a \notin A \implies$
 $t @ [ev\ a] \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u, v @ [ev\ a]), A) \rangle$
by (induct $\langle (tick\text{-join}, u, A, v) \rangle$ arbitrary: $t\ u\ v$)
(auto split: if-split-asm option.split-asm)

lemma setinterleaves_{ptick}-snoc-inside :

$\langle t \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u, v), A) \implies a \in A \implies$
 $t @ [ev\ a] \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u @ [ev\ a], v @ [ev\ a]), A) \rangle$
by (induct $\langle (tick\text{-join}, u, A, v) \rangle$ arbitrary: $t\ u\ v$)
(auto split: if-split-asm option.split-asm)

lemma setinterleaves_{ptick}-snoc-tick :

$\langle t \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u, v), A) \implies tick\text{-join } r\ s = [r\text{-}s] \implies$
 $t @ [\checkmark(r\text{-}s)] \text{ setinterleaves}_{\checkmark} \text{ tick-join } ((u @ [\checkmark(r)], v @ [\checkmark(s)]), A) \rangle$
by (induct $\langle (tick\text{-join}, u, A, v) \rangle$ arbitrary: $t\ u\ v$)

(*auto split: if-split-asm option.split-asm*)

lemma *Cons-tick-setinterleaves_{ptick}E* :

$\langle \checkmark(r-s) \# t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$
 $(\bigwedge u' v' r s. \llbracket \text{tick-join } r s = \lfloor r-s \rfloor; u = \checkmark(r) \# u'; v = \checkmark(s) \# v';$
 $t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v'), A) \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$
by (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: t u v*)
(simp-all split: if-split-asm option.split-asm)

lemma *Cons-ev-setinterleaves_{ptick}E* :

$\langle \text{ev } a \# t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$
 $(\bigwedge u'. a \notin A \implies u = \text{ev } a \# u' \implies t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v), A) \implies$
 $\text{thesis}) \implies$
 $(\bigwedge v'. a \notin A \implies v = \text{ev } a \# v' \implies t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v'), A) \implies$
 $\text{thesis}) \implies$
 $(\bigwedge u' v'. a \in A \implies u = \text{ev } a \# u' \implies v = \text{ev } a \# v' \implies$
 $t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v'), A) \implies \text{thesis}) \implies \text{thesis} \rangle$

proof (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: u v t*)

case *Nil-setinterleaving_{ptick}-Nil* **thus** ?*case by simp*

next

case (*ev-setinterleaving_{ptick}-Nil b u*)

from *ev-setinterleaving_{ptick}-Nil.prem*s(1) **show** ?*case*

by (*simp add: ev-setinterleaving_{ptick}-Nil.prem*s(2) *split: if-split-asm*)

next

case (*tick-setinterleaving_{ptick}-Nil r u*) **thus** ?*case by simp*

next

case (*Nil-setinterleaving_{ptick}-ev c v*)

from *Nil-setinterleaving_{ptick}-ev.prem*s(1) **show** ?*case*

by (*simp add: Nil-setinterleaving_{ptick}-ev.prem*s(3) *split: if-split-asm*)

next

case (*Nil-setinterleaving_{ptick}-tick s v*) **thus** ?*case by simp*

next

case (*ev-setinterleaving_{ptick}-ev b u c v*)

from *ev-setinterleaving_{ptick}-ev.prem*s(1) **show** ?*case*

by (*simp add: ev-setinterleaving_{ptick}-ev.prem*s(2, 3, 4) *split: if-split-asm*)

(*use ev-setinterleaving_{ptick}-ev.prem*s(2, 3) **in** *presburger*)

next

case (*ev-setinterleaving_{ptick}-tick b u s v*)

from *ev-setinterleaving_{ptick}-tick.prem*s(1) **show** ?*case*

by (*simp add: ev-setinterleaving_{ptick}-tick.prem*s(2) *split: if-split-asm*)

next

case (*tick-setinterleaving_{ptick}-ev r u c v*)

from *tick-setinterleaving_{ptick}-ev.prem*s(1) **show** ?*case*

by (*simp add: tick-setinterleaving_{ptick}-ev.prem*s(3) *split: if-split-asm*)

next

case (*tick-setinterleaving_{ptick}-tick r u s v*)

thus ?*case by (simp split: option.split-asm)*

qed

```

lemma rev-setinterleavesptick-rev-rev-iff :
  ⟨rev t setinterleaves✓tick-join ((rev u, rev v), A)
  ⟷ t setinterleaves✓tick-join ((u, v), A)⟩
  for u :: ⟨('a, 'r) traceptick⟩ and v :: ⟨('a, 's) traceptick⟩
proof (rule iffI)
  show ⟨t setinterleaves✓tick-join ((u, v), A) ⟹
    rev t setinterleaves✓tick-join ((rev u, rev v), A)⟩
    for u :: ⟨('a, 'r) traceptick⟩ and v :: ⟨('a, 's) traceptick⟩ and t
proof (induct ⟨(tick-join, u, A, v)⟩ arbitrary: t u v)
  case Nil-setinterleavingptick-Nil thus ?case by simp
next
  case (ev-setinterleavingptick-Nil a u)
  thus ?case by (auto simp add: setinterleavesptick-snoc-notinL split: if-split-asm)
)
next
  case (tick-setinterleavingptick-Nil r v) thus ?case by simp
next
  case (Nil-setinterleavingptick-ev b v)
  thus ?case by (auto simp add: setinterleavesptick-snoc-notinR split: if-split-asm)
)
next
  case (Nil-setinterleavingptick-tick s v) thus ?case by simp
next
  case (ev-setinterleavingptick-ev a u b v)
from ev-setinterleavingptick-ev.prem
consider (both-in) t' where ⟨a ∈ A⟩ ⟨a = b⟩ ⟨t = ev a # t'⟩
  ⟨t' setinterleaves✓tick-join ((u, v), A)⟩
| (inR-mvL) t' where ⟨a ∉ A⟩ ⟨b ∈ A⟩ ⟨t = ev a # t'⟩
  ⟨t' setinterleaves✓tick-join ((u, ev b # v), A)⟩
| (inL-mvR) t' where ⟨a ∈ A⟩ ⟨b ∉ A⟩ ⟨t = ev b # t'⟩
  ⟨t' setinterleaves✓tick-join ((ev a # u, v), A)⟩
| (notin-mvL) t' where ⟨a ∉ A⟩ ⟨b ∉ A⟩ ⟨t = ev a # t'⟩
  ⟨t' setinterleaves✓tick-join ((u, ev b # v), A)⟩
| (notin-mvR) t' where ⟨a ∉ A⟩ ⟨b ∉ A⟩ ⟨t = ev b # t'⟩
  ⟨t' setinterleaves✓tick-join ((ev a # u, v), A)⟩
  by (auto split: if-split-asm)
thus ?case
proof cases
  case both-in thus ?thesis
  by (simp add: ev-setinterleavingptick-ev.hyps(1) setinterleavesptick-snoc-inside)
next
  case inR-mvL thus ?thesis
  by (metis ev-setinterleavingptick-ev.hyps(3) rev.simps(2) setinterleavesptick-snoc-notinL)
next
  case inL-mvR thus ?thesis
  by (metis ev-setinterleavingptick-ev.hyps(2) rev.simps(2) setinterleavesptick-snoc-notinR)

```

```

next
  case notin-mvL thus ?thesis
  by (metis ev-setinterleavingptick-ev.hyps(4) rev.simps(2) setinterleavesptick-snoc-notinL)
next
  case notin-mvR thus ?thesis
  by (metis ev-setinterleavingptick-ev.hyps(5) rev.simps(2) setinterleavesptick-snoc-notinR)
qed
next
  case (ev-setinterleavingptick-tick a u s v) thus ?case
  by (auto simp add: setinterleavesptick-snoc-notinL split: if-split-asm)
next
  case (tick-setinterleavingptick-ev r u b v) thus ?case
  by (auto simp add: setinterleavesptick-snoc-notinR split: if-split-asm)
next
  case (tick-setinterleavingptick-tick r u s v)
  from tick-setinterleavingptick-tick.prem
  obtain t' r-s where ⟨tick-join r s = ⌊r-s⌋⟩ ⟨t = ✓(r-s) # t'⟩
    ⟨t' setinterleaves✓tick-join ((u, v), A)⟩
  by (auto split: option.split-asm)
  from ⟨t' setinterleaves✓tick-join ((u, v), A)⟩
  have ⟨rev t' setinterleaves✓tick-join ((rev u, rev v), A)⟩
  by (simp add: ⟨tick-join r s = ⌊r-s⌋⟩ tick-setinterleavingptick-tick.hyps)
  hence ⟨rev t' @ [✓(r-s)] setinterleaves✓tick-join ((rev u @ [✓(r)], rev v @ [✓(s)]),
    A)⟩
  by (simp add: ⟨tick-join r s = ⌊r-s⌋⟩ setinterleavesptick-snoc-tick)
  thus ?case by (simp add: ⟨t = ✓(r-s) # t'⟩)
qed
from this[of ⟨rev t⟩ ⟨rev u⟩ ⟨rev v⟩, simplified]
show ⟨rev t setinterleaves✓tick-join ((rev u, rev v), A) ⟹
  t setinterleaves✓tick-join ((u, v), A)⟩ .
qed

```

lemma *setinterleaves_{ptick}-preserves-ev-notin-set* :

⟨[ev a ∉ set u; ev a ∉ set v; t setinterleaves_{✓tick-join} ((u, v), A)] ⟹ ev a ∉ set t⟩

by (induct ⟨(tick-join, u, A, v)⟩ arbitrary: t u v)
(auto split: if-split-asm option.split-asm)

lemma *setinterleaves_{ptick}-inj-preserves-tick-notin-set* :

⟨[tick-join r s = ⌊r-s⌋; ✓(r) ∉ set u ∨ ✓(s) ∉ set v;
t setinterleaves_{✓tick-join} ((u, v), A)] ⟹ ✓(r-s) ∉ set t⟩

— This is a weakened injectivity property.

if inj-tick-join : ⟨∧ r' s'. tick-join r' s' = ⌊r-s⌋ ⟹ r' = r ∧ s' = s⟩

by (induct ⟨(tick-join, u, A, v)⟩ arbitrary: t u v)
(auto split: if-split-asm option.split-asm, (metis inj-tick-join)+)

lemma *setinterleaves_{ptick}-preserves-ev-inside-set* :

⟨[ev a ∈ set u; ev a ∈ set v; t setinterleaves_{✓tick-join} ((u, v), A)] ⟹ ev a ∈ set

$t\rangle$

by $\langle \text{induct } \langle \text{tick-join}, u, A, v \rangle \text{ arbitrary: } t \ u \ v \rangle$
 $\langle \text{auto split: if-split-asm option.split-asm} \rangle$

lemma $\text{ev-notin-both-sets-imp-empty-setinterleaving}_{ptick} :$

$\langle \llbracket \text{ev } a \in \text{set } u \wedge \text{ev } a \notin \text{set } v \vee \text{ev } a \notin \text{set } u \wedge \text{ev } a \in \text{set } v; a \in A \rrbracket \implies$
 $\text{setinterleaving}_{ptick} (\text{tick-join}, u, A, v) = \{\} \rangle$
by $\langle \text{induct } \langle \text{tick-join}, u, A, v \rangle \text{ arbitrary: } u \ v \rangle$
 $\langle \text{simp-all, safe, auto split: option.split-asm} \rangle$

lemma $\text{setinterleaves}_{ptick}\text{-snoc-tick-snoc-tick}E:$

$\langle (\bigwedge t' \ r\text{-s. } \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor \implies t' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u, v), A) \implies$
 $t = t' @ [\checkmark(r\text{-s})] \implies \text{thesis}) \implies \text{thesis} \rangle$

if $\langle t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u @ [\checkmark(r)], v @ [\checkmark(s)]), A) \rangle$

proof –

from that have $\langle \text{rev } t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((\checkmark(r) \# \text{rev } u, \checkmark(s) \# \text{rev } v), A) \rangle$

by $\langle \text{metis (no-types) rev.simps(2) rev-rev-ident rev-setinterleaves}_{ptick}\text{-rev-rev-iff} \rangle$

then obtain $t' \ r\text{-s}$ **where** $\langle \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor \rangle \langle \text{rev } t = \checkmark(r\text{-s}) \# t' \rangle$

$\langle t' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((\text{rev } u, \text{rev } v), A) \rangle$

by $\langle \text{cases } t \text{ rule: rev-cases} \rangle \langle \text{simp-all split: option.split-asm} \rangle$

hence $\langle \text{rev } t' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u, v), A) \wedge t = \text{rev } t' @ [\checkmark(r\text{-s})] \rangle$

using $\text{rev-setinterleaves}_{ptick}\text{-rev-rev-iff}$ **by force**

with $\langle \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor \rangle$

show $\langle (\bigwedge t' \ r\text{-s. } \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor \implies t' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u, v), A) \implies$
 $t = t' @ [\checkmark(r\text{-s})] \implies \text{thesis}) \implies \text{thesis} \rangle$ **by blast**

qed

lemma $\text{snoc-tick-setinterleaves}_{ptick}E :$

$\langle (\bigwedge u' \ v' \ r \ s. \llbracket \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor; t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u', v'), A);$
 $u = u' @ [\checkmark(r)]; v = v' @ [\checkmark(s)] \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$

if $\langle t @ [\checkmark(r\text{-s})] \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u, v), A) \rangle$

proof –

from that have $\langle \text{rev } (t @ [\checkmark(r\text{-s})]) \text{ setinterleaves}_{\checkmark \text{tick-join}} ((\text{rev } u, \text{rev } v), A) \rangle$

by $\langle \text{metis (no-types) rev.simps(2) rev-rev-ident rev-setinterleaves}_{ptick}\text{-rev-rev-iff} \rangle$

hence $\langle \checkmark(r\text{-s}) \# \text{rev } t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((\text{rev } u, \text{rev } v), A) \rangle$ **by simp**

then obtain $u' \ v' \ r \ s$ **where** $\langle \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor \rangle$

$\langle \text{rev } t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u', v'), A) \rangle$

$\langle \text{rev } u = \checkmark(r) \# u' \rangle \langle \text{rev } v = \checkmark(s) \# v' \rangle$

by $\langle \text{elim Cons-tick-setinterleaves}_{ptick}E \rangle$

hence $\langle t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((\text{rev } u', \text{rev } v'), A) \wedge$

$u = \text{rev } u' @ [\checkmark(r)] \wedge v = \text{rev } v' @ [\checkmark(s)] \rangle$

using $\text{rev-setinterleaves}_{ptick}\text{-rev-rev-iff}$ **by fastforce**

with $\langle \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor \rangle$

show $\langle (\bigwedge u' \ v' \ r \ s. \llbracket \text{tick-join } r \ s = \lfloor r\text{-s} \rfloor; t \text{ setinterleaves}_{\checkmark \text{tick-join}} ((u', v'), A);$
 $u = u' @ [\checkmark(r)]; v = v' @ [\checkmark(s)] \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$ **by blast**

qed

4.1.4 Lengths

lemma *length-setinterleaves_{ptick}-eq-sum-minus-filterL* :

$\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$
 $\text{length } t = \text{length } u + \text{length } v - \text{length } (\text{filter } (\lambda e. e \in \text{range tick} \cup \text{ev } ^\circ A) u) \rangle$

proof (*induct t arbitrary: u v*)

case *Nil*

thus ?case **by** (*auto dest: Nil-setinterleaves_{ptick}*)

next

note *thms = Suc-diff-le le-add1 length-filter-le order-trans*

case (*Cons e t*)

from *Cons.prem*s **consider** (*mv-left*) *a u' where* $\langle a \notin A \rangle \langle e = \text{ev } a \rangle \langle u = \text{ev } a \# u' \rangle$

$\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v), A) \rangle$

| (*mv-right*) *a v' where* $\langle a \notin A \rangle \langle e = \text{ev } a \rangle \langle v = \text{ev } a \# v' \rangle$

$\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v'), A) \rangle$

| (*mv-both-ev*) *a u' v' where* $\langle a \in A \rangle \langle e = \text{ev } a \rangle \langle u = \text{ev } a \# u' \rangle \langle v = \text{ev } a \# v' \rangle$

$\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v'), A) \rangle$

| (*mv-both-tick*) *r s r-s u' v' where* $\langle \text{tick-join } r s = \lfloor r-s \rfloor \rangle \langle e = \checkmark(r-s) \rangle$

$\langle u = \checkmark(r) \# u' \rangle \langle v = \checkmark(s) \# v' \rangle \langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v'), A) \rangle$

by (*cases e*) (*auto elim: Cons-ev-setinterleaves_{ptick} E Cons-tick-setinterleaves_{ptick} E*)

thus ?case

proof *cases*

case *mv-left*

from *Cons.hyps*[*OF mv-left(4)*] **show** ?thesis

by (*simp add: mv-left(1-3) image-iff*) (*metis (no-types, lifting) thms*)

next

case *mv-right*

from *Cons.hyps*[*OF mv-right(4)*] **show** ?thesis

by (*simp add: mv-right(1-3) image-iff*) (*metis (no-types, lifting) thms*)

next

case *mv-both-ev*

from *Cons.hyps*[*OF mv-both-ev(5)*] **show** ?thesis

by (*simp add: mv-both-ev(1, 3, 4) image-iff*) (*metis (no-types, lifting) thms*)

next

case *mv-both-tick*

from *Cons.hyps*[*OF mv-both-tick(5)*] **show** ?thesis

by (*simp add: mv-both-tick(3, 4) image-iff*) (*metis (no-types, lifting) thms*)

qed

qed

lemma *length-setinterleaves_{ptick}-eq-sum-minus-filterR* :

$\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$

$\text{length } t = \text{length } u + \text{length } v - \text{length } (\text{filter } (\lambda e. e \in \text{range tick} \cup \text{ev } ^\circ A) v) \rangle$

by (*subst (asm) setinterleaves_{ptick}-sym*)

(*auto dest: length-setinterleaves_{ptick}-eq-sum-minus-filterL*)

lemma *setinterleaves_{ptick}-eq-length* :
 $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$
 $t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies \text{length } t = \text{length } t' \rangle$
by (*simp add: length-setinterleaves_{ptick}-eq-sum-minus-filterL*)

lemma *setinterleaves_{ptick}-imp-lengthLR-le* :
 $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$
 $\text{length } u \leq \text{length } t \wedge \text{length } v \leq \text{length } t \rangle$
by (*induct* $\langle (\text{tick-join}, u, A, v) \rangle$ *arbitrary: t u v*)
(fastforce split: if-split-asm option.split-asm)+

4.1.5 Trace Prefix Interleaving

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

lemma *setinterleaves_{ptick}-appendL* :
 $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u1 @ u2, v), A) \implies$
 $\exists t1 t2 v1 v2. t = t1 @ t2 \wedge v = v1 @ v2 \wedge$
 $t1 \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u1, v1), A) \wedge$
 $t2 \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$
proof (*induct* $\langle (\text{tick-join}, u1, A, v) \rangle$ *arbitrary: t u1 v*)
case *Nil-setinterleaving_{ptick}-Nil*
thus *?case by simp*
next
case (*ev-setinterleaving_{ptick}-Nil a u1*)
from *ev-setinterleaving_{ptick}-Nil.prem*s
have $\langle a \notin A \rangle \langle t = \text{ev } a \# \text{map ev (map of-ev (u1 @ u2))} \rangle$
 $\langle \text{map ev (map of-ev (u1 @ u2)) setinterleaves}_{\checkmark \text{ tick-join}} ((u1 @ u2, []), A) \rangle$
by (*simp-all add: setinterleaves_{ptick}-NilR-iff split: if-split-asm*)
from *ev-setinterleaving_{ptick}-Nil.hyps[OF* $\langle a \notin A \rangle$ *this(3)]*
obtain *t1 t2 v1 v2 where* $\langle \text{map ev (map of-ev (u1 @ u2))} = t1 @ t2 \rangle$
 $\langle [] = v1 @ v2 \rangle \langle t1 \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u1, v1), A) \rangle$
 $\langle t2 \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$ **by** *blast*
thus *?case*
by (*simp add:* $\langle a \notin A \rangle \langle t = \text{ev } a \# \text{map ev (map of-ev (u1 @ u2))} \rangle$)
(metis append-Cons)
next
case (*tick-setinterleaving_{ptick}-Nil r u1*)
from *tick-setinterleaving_{ptick}-Nil.prem*s **have** *False by simp*
thus *?case ..*
next
case (*Nil-setinterleaving_{ptick}-ev b v*)
thus *?case*
by (*cases u2, simp-all split: if-split-asm*)
(fastforce, metis Nil-setinterleaving_{ptick}-Nil self-append-conv2 singleton-iff)

```

next
  case (Nil-setinterleavingptick-tick s v)
  thus ?case by (cases u2, simp-all add: setinterleavingptick-simps
    split: eventptick.split-asm) fastforce+
next
  case (ev-setinterleavingptick-ev a u1 b v)
  from ev-setinterleavingptick-ev.premis [simplified]
  consider (mv-both) t' where ⟨a ∈ A⟩ ⟨b ∈ A⟩ ⟨a = b⟩ ⟨t = ev b # t'⟩ ⟨t'
setinterleaves✓tick-join ((u1 @ u2, v), A)⟩
    | (mvR-inL) t' where ⟨a ∈ A⟩ ⟨b ∉ A⟩ ⟨t = ev b # t'⟩ ⟨t' setinter-
leaves✓tick-join (((ev a # u1) @ u2, v), A)⟩
    | (mvL-inR) t' where ⟨a ∉ A⟩ ⟨b ∈ A⟩ ⟨t = ev a # t'⟩ ⟨t' setinter-
leaves✓tick-join ((u1 @ u2, ev b # v), A)⟩
    | (mvR-notin) t' where ⟨a ∉ A⟩ ⟨b ∉ A⟩ ⟨t = ev b # t'⟩ ⟨t' setinter-
leaves✓tick-join (((ev a # u1) @ u2, v), A)⟩
    | (mvL-notin) t' where ⟨a ∉ A⟩ ⟨b ∉ A⟩ ⟨t = ev a # t'⟩ ⟨t' setinter-
leaves✓tick-join ((u1 @ u2, ev b # v), A)⟩
  by (auto split: if-split-asm)
  thus ?case
  proof cases
    case mv-both
    from ev-setinterleavingptick-ev.hyps(1)[OF mv-both(1-3, 5)] obtain t1 t2 v1
v2
    where ⟨t' = t1 @ t2⟩ ⟨v = v1 @ v2⟩ ⟨t1 setinterleaves✓tick-join ((u1, v1),
A)⟩
    ⟨t2 setinterleaves✓tick-join ((u2, v2), A)⟩ by blast
    hence ⟨t = (ev b # t1) @ t2 ∧ ev b # v = (ev b # v1) @ v2 ∧
ev b # t1 setinterleaves✓tick-join ((ev a # u1, ev b # v1), A) ∧
t2 setinterleaves✓tick-join ((u2, v2), A)⟩ by (simp add: mv-both(1-4))
    thus ?thesis by blast
  next
    case mvR-inL
    from ev-setinterleavingptick-ev.hyps(2)[OF mvR-inL(1, 2, 4)] obtain t1 t2 v1
v2
    where ⟨t' = t1 @ t2⟩ ⟨v = v1 @ v2⟩ ⟨t1 setinterleaves✓tick-join ((ev a # u1,
v1), A)⟩
    ⟨t2 setinterleaves✓tick-join ((u2, v2), A)⟩ by blast
    hence ⟨t = (ev b # t1) @ t2 ∧ ev b # v = (ev b # v1) @ v2 ∧
ev b # t1 setinterleaves✓tick-join ((ev a # u1, ev b # v1), A) ∧
t2 setinterleaves✓tick-join ((u2, v2), A)⟩ by (simp add: mvR-inL(1-3))
    thus ?thesis by blast
  next
    case mvL-inR
    from ev-setinterleavingptick-ev.hyps(3)[OF mvL-inR(1, 2, 4)] obtain t1 t2 v1
v2
    where ⟨t' = t1 @ t2⟩ ⟨ev b # v = v1 @ v2⟩ ⟨t1 setinterleaves✓tick-join ((u1,
v1), A)⟩
    ⟨t2 setinterleaves✓tick-join ((u2, v2), A)⟩ by blast

```

hence $\langle t = (ev\ a \# t1) @ t2 \wedge ev\ b \# v = v1 @ v2 \wedge$
 $ev\ a \# t1\ setinterleaves_{\checkmark tick-join} ((ev\ a \# u1, v1), A) \wedge$
 $t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$
by (cases $v1$, simp-all add: mvL-inR(1, 3))
thus ?thesis **by** blast
next
case mvR-notin
from ev-setinterleaving_{ptick}-ev.hyps(5)[OF mvR-notin(1, 2, 4)] **obtain** $t1\ t2$
 $v1\ v2$
where $\langle t' = t1 @ t2 \rangle \langle v = v1 @ v2 \rangle \langle t1\ setinterleaves_{\checkmark tick-join} ((ev\ a \# u1,$
 $v1), A) \rangle$
 $\langle t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** blast
hence $\langle t = (ev\ b \# t1) @ t2 \wedge ev\ b \# v = (ev\ b \# v1) @ v2 \wedge$
 $ev\ b \# t1\ setinterleaves_{\checkmark tick-join} ((ev\ a \# u1, ev\ b \# v1), A) \wedge$
 $t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** (simp add: mvR-notin(1-3))
thus ?thesis **by** blast
next
case mvL-notin
from ev-setinterleaving_{ptick}-ev.hyps(4)[OF mvL-notin(1, 2, 4)] **obtain** $t1\ t2$
 $v1\ v2$
where $\langle t' = t1 @ t2 \rangle \langle ev\ b \# v = v1 @ v2 \rangle \langle t1\ setinterleaves_{\checkmark tick-join} ((u1,$
 $v1), A) \rangle$
 $\langle t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** blast
hence $\langle t = (ev\ a \# t1) @ t2 \wedge ev\ b \# v = v1 @ v2 \wedge$
 $ev\ a \# t1\ setinterleaves_{\checkmark tick-join} ((ev\ a \# u1, v1), A) \wedge$
 $t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$
by (cases $v1$, simp-all add: mvL-notin(1, 3))
thus ?thesis **by** blast
qed
next
case (ev-setinterleaving_{ptick}-tick a u1 s v)
from ev-setinterleaving_{ptick}-tick.premis **obtain** t'
where $\langle a \notin A \rangle \langle t = ev\ a \# t' \rangle \langle t'\ setinterleaves_{\checkmark tick-join} ((u1 @ u2, \checkmark(s) \#$
 $v), A) \rangle$
by (auto split: if-split-asm)
from ev-setinterleaving_{ptick}-tick.hyps[OF this(1, 3)] **obtain** $t1\ t2\ v1\ v2$
where $\langle t' = t1 @ t2 \rangle \langle \checkmark(s) \# v = v1 @ v2 \rangle$
 $\langle t1\ setinterleaves_{\checkmark tick-join} ((u1, v1), A) \rangle$
 $\langle t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** blast
hence $\langle t = (ev\ a \# t1) @ t2 \wedge \checkmark(s) \# v = v1 @ v2 \wedge$
 $ev\ a \# t1\ setinterleaves_{\checkmark tick-join} ((ev\ a \# u1, v1), A) \wedge$
 $t2\ setinterleaves_{\checkmark tick-join} ((u2, v2), A) \rangle$
by (cases $v1$, simp-all add: $\langle t = ev\ a \# t' \rangle \langle a \notin A \rangle$)
thus ?case **by** blast
next
case (tick-setinterleaving_{ptick}-ev r u1 b v)
from tick-setinterleaving_{ptick}-ev.premis **obtain** t'
where $\langle b \notin A \rangle \langle t = ev\ b \# t' \rangle \langle t'\ setinterleaves_{\checkmark tick-join} (((\checkmark(r) \# u1) @ u2,$

$v\rangle, A\rangle$
by (*auto split: if-split-asm*)
from $\text{tick-setinterleaving}_{\text{ptick}}\text{-ev.hyps}[OF \text{ this}(1, 3)]$ **obtain** $t1\ t2\ v1\ v2$
where $\langle t' = t1 \ @ \ t2 \rangle \langle v = v1 \ @ \ v2 \rangle$
 $\langle t1 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((\checkmark(r) \# u1, v1), A) \rangle$
 $\langle t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$ **by** *blast*
hence $\langle t = (ev\ b \# t1) \ @ \ t2 \wedge ev\ b \# v = (ev\ b \# v1) \ @ \ v2 \wedge$
 $ev\ b \# t1 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((\checkmark(r) \# u1, ev\ b \# v1), A) \wedge$
 $t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$
by (*simp add: $\langle t = ev\ b \# t' \rangle \langle b \notin A \rangle$*)
thus *?case* **by** *blast*
next
case ($\text{tick-setinterleaving}_{\text{ptick}}\text{-tick}\ r\ u1\ s\ v$)
from $\text{tick-setinterleaving}_{\text{ptick}}\text{-tick.prem}$ s **obtain** $r\text{-}s\ t'$
where $\langle \text{tick-join}\ r\ s = \lfloor r\text{-}s \rfloor \rangle \langle t = \checkmark(r\text{-}s) \# t' \rangle$
 $\langle t' \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u1 \ @ \ u2, v), A) \rangle$ **by** (*auto split: option.split-asm*)
from $\text{tick-setinterleaving}_{\text{ptick}}\text{-tick.hyps}[OF \text{ this}(1, 3)]$ **obtain** $t1\ t2\ v1\ v2$
where $\langle t' = t1 \ @ \ t2 \rangle \langle v = v1 \ @ \ v2 \rangle \langle t1 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u1, v1), A) \rangle$
 $\langle t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$ **by** *blast*
hence $\langle t = (\checkmark(r\text{-}s) \# t1) \ @ \ t2 \wedge \checkmark(s) \# v = (\checkmark(s) \# v1) \ @ \ v2 \wedge$
 $\checkmark(r\text{-}s) \# t1 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((\checkmark(r) \# u1, \checkmark(s) \# v1), A) \wedge$
 $t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$
by (*simp add: $\langle \text{tick-join}\ r\ s = \lfloor r\text{-}s \rfloor \rangle \langle t = \checkmark(r\text{-}s) \# t' \rangle$*)
thus *?case* **by** *blast*
qed

corollary $\text{setinterleaves}_{\text{ptick}}\text{-appendR}$:

$\langle \exists t1\ t2\ u1\ u2. t = t1 \ @ \ t2 \wedge u = u1 \ @ \ u2 \wedge$
 $t1 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u1, v1), A) \wedge$
 $t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$
if $\langle t \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u, v1 \ @ \ v2), A) \rangle$

proof –

from *that have* $\langle t \ \text{setinterleaves}_{\checkmark \lambda s\ r. \text{ tick-join}\ r\ s} ((v1 \ @ \ v2, u), A) \rangle$
using $\text{setinterleaves}_{\text{ptick}}\text{-sym}$ **by** *blast*
from $\text{setinterleaves}_{\text{ptick}}\text{-appendL}[OF \text{ this}]$
obtain $t1\ t2\ u1\ u2$ **where** $\langle t = t1 \ @ \ t2 \rangle \langle u = u1 \ @ \ u2 \rangle$
 $\langle t1 \ \text{setinterleaves}_{\checkmark \lambda s\ r. \text{ tick-join}\ r\ s} ((v1, u1), A) \rangle$
 $\langle t2 \ \text{setinterleaves}_{\checkmark \lambda s\ r. \text{ tick-join}\ r\ s} ((v2, u2), A) \rangle$ **by** *blast*
from $\text{this}(3, 4)$ **have** $\langle t1 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u1, v1), A) \rangle$
 $\langle t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u2, v2), A) \rangle$
using $\text{setinterleaves}_{\text{ptick}}\text{-sym}$ **by** *blast* +
with $\langle t = t1 \ @ \ t2 \rangle \langle u = u1 \ @ \ u2 \rangle$ **show** *?thesis* **by** *blast*
qed

lemma $\text{append-setinterleaves}_{\text{ptick}}$:

$\langle t1 \ @ \ t2 \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u, v), A) \implies$

$\exists u1\ u2\ v1\ v2. u = u1 \text{ @ } u2 \wedge v = v1 \text{ @ } v2 \wedge$
 $t1 \text{ setinterleaves}_{\checkmark tick-join} ((u1, v1), A) \wedge$
 $t2 \text{ setinterleaves}_{\checkmark tick-join} ((u2, v2), A)$

proof (induct $t1$ arbitrary: $u\ v$)

case *Nil*

hence $\langle u = [] \text{ @ } u \rangle \langle v = [] \text{ @ } v \rangle$
 $\langle [] \text{ setinterleaves}_{\checkmark tick-join} ([], []), A \rangle$
 $\langle t2 \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \rangle$ **by** *simp-all*

thus ?case **by** *blast*

next

case (*Cons e t1*)

from *Cons.prem*s **consider** (*mv-left*) $a\ u'$ **where** $\langle a \notin A \rangle \langle e = ev\ a \rangle \langle u = ev\ a \# u' \rangle$

$\langle t1 \text{ @ } t2 \text{ setinterleaves}_{\checkmark tick-join} ((u', v), A) \rangle$
 $| \text{ (mv-right) } a\ v' \text{ where } \langle a \notin A \rangle \langle e = ev\ a \rangle \langle v = ev\ a \# v' \rangle$
 $\langle t1 \text{ @ } t2 \text{ setinterleaves}_{\checkmark tick-join} ((u, v'), A) \rangle$
 $| \text{ (mv-both-ev) } a\ u'\ v' \text{ where } \langle a \in A \rangle \langle e = ev\ a \rangle \langle u = ev\ a \# u' \rangle \langle v = ev\ a \# v' \rangle$
 $\langle t1 \text{ @ } t2 \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$
 $| \text{ (mv-both-tick) } r\ s\ r-s\ u'\ v' \text{ where } \langle tick-join\ r\ s = [r-s] \rangle \langle e = \checkmark(r-s) \rangle$
 $\langle u = \checkmark(r) \# u' \rangle \langle v = \checkmark(s) \# v' \rangle \langle t1 \text{ @ } t2 \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$
by (*cases e*) (*auto elim: Cons-ev-setinterleaves_{ptick}E Cons-tick-setinterleaves_{ptick}E*)

thus ?case

proof *cases*

case *mv-left*

from *Cons.hyps*[*OF mv-left(4)*] **obtain** $u1\ u2\ v1\ v2$

where $\langle u' = u1 \text{ @ } u2 \rangle \langle t1 \text{ setinterleaves}_{\checkmark tick-join} ((u1, v1), A) \rangle$
and $*$: $\langle v = v1 \text{ @ } v2 \rangle \langle t2 \text{ setinterleaves}_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** *blast*

from *this(2)* **have** $\langle e \# t1 \text{ setinterleaves}_{\checkmark tick-join} ((ev\ a \# u1, v1), A) \rangle$
by (*cases v1*) (*auto simp add: \langle a \notin A \rangle \langle e = ev\ a \rangle setinterleaving_{ptick}-simps split: event_{ptick}.split*)

moreover from $\langle u' = u1 \text{ @ } u2 \rangle$ **have** $\langle u = (ev\ a \# u1) \text{ @ } u2 \rangle$
by (*simp add: mv-left(3)*)

ultimately show ?thesis **using** $*(1, 2)$ **by** *blast*

next

case *mv-right*

from *Cons.hyps*[*OF mv-right(4)*] **obtain** $u1\ u2\ v1\ v2$

where $\langle v' = v1 \text{ @ } v2 \rangle \langle t1 \text{ setinterleaves}_{\checkmark tick-join} ((u1, v1), A) \rangle$
and $*$: $\langle u = u1 \text{ @ } u2 \rangle \langle t2 \text{ setinterleaves}_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** *blast*

from *this(2)* **have** $\langle e \# t1 \text{ setinterleaves}_{\checkmark tick-join} ((u1, ev\ a \# v1), A) \rangle$
by (*cases u1*) (*auto simp add: \langle a \notin A \rangle \langle e = ev\ a \rangle setinterleaving_{ptick}-simps split: event_{ptick}.split*)

moreover from $\langle v' = v1 \text{ @ } v2 \rangle$ **have** $\langle v = (ev\ a \# v1) \text{ @ } v2 \rangle$
by (*simp add: mv-right(3)*)

ultimately show ?thesis **using** $*(1, 2)$ **by** *blast*

next

case *mv-both-ev*

from *Cons.hyps*[*OF mv-both-ev(5)*] **obtain** $u1\ u2\ v1\ v2$

where $\langle u' = u1 \ @ \ u2 \rangle \langle v' = v1 \ @ \ v2 \rangle \langle t1 \text{ setinterleaves}_{\checkmark tick-join} ((u1, v1), A) \rangle$
and $*$: $\langle t2 \text{ setinterleaves}_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** *blast*
from *this*(3) **have** $\langle e \# t1 \text{ setinterleaves}_{\checkmark tick-join} ((ev \ a \ \# \ u1, ev \ a \ \# \ v1), A) \rangle$
by (*simp add: $\langle a \in A \rangle \langle e = ev \ a \rangle$*)
moreover from $\langle u' = u1 \ @ \ u2 \rangle$ **have** $\langle u = (ev \ a \ \# \ u1) \ @ \ u2 \rangle$
by (*simp add: mv-both-ev(3)*)
moreover from $\langle v' = v1 \ @ \ v2 \rangle$ **have** $\langle v = (ev \ a \ \# \ v1) \ @ \ v2 \rangle$
by (*simp add: mv-both-ev(4)*)
ultimately show *?thesis* **using** $*$ **by** *blast*
next
case *mv-both-tick*
from *Cons.hyps*[*OF mv-both-tick(5)*] **obtain** $u1 \ u2 \ v1 \ v2$
where $\langle u' = u1 \ @ \ u2 \rangle \langle v' = v1 \ @ \ v2 \rangle \langle t1 \text{ setinterleaves}_{\checkmark tick-join} ((u1, v1), A) \rangle$
and $*$: $\langle t2 \text{ setinterleaves}_{\checkmark tick-join} ((u2, v2), A) \rangle$ **by** *blast*
from *this*(3) **have** $\langle e \# t1 \text{ setinterleaves}_{\checkmark tick-join} ((\checkmark(r) \ \# \ u1, \checkmark(s) \ \# \ v1), A) \rangle$
by (*simp add: mv-both-tick(1, 2)*)
moreover from $\langle u' = u1 \ @ \ u2 \rangle$ **have** $\langle u = (\checkmark(r) \ \# \ u1) \ @ \ u2 \rangle$
by (*simp add: mv-both-tick(3)*)
moreover from $\langle v' = v1 \ @ \ v2 \rangle$ **have** $\langle v = (\checkmark(s) \ \# \ v1) \ @ \ v2 \rangle$
by (*simp add: mv-both-tick(4)*)
ultimately show *?thesis* **using** $*$ **by** *blast*
qed
qed

corollary *setinterleaves_{ptick}-le-prefixL* :
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \implies u' \leq u \implies$
 $\exists t' \leq t. \exists v' \leq v. t' \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$
by (*auto elim!: prefixE dest!: setinterleaves_{ptick}-appendL intro: prefixI*)

corollary *setinterleaves_{ptick}-le-prefixR* :
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \implies v' \leq v \implies$
 $\exists t' \leq t. \exists u' \leq u. t' \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$
by (*auto elim!: prefixE dest!: setinterleaves_{ptick}-appendR intro: prefixI*)

corollary *le-prefix-setinterleaves_{ptick}* :
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \implies t' \leq t \implies$
 $\exists u' \leq u. \exists v' \leq v. t' \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$
by (*auto elim!: prefixE dest!: append-setinterleaves_{ptick} intro: prefixI*)

lemma *setinterleaves_{ptick}-less-prefixL* :

$\langle t \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \implies u' < u \implies$
 $\exists t' v'. t' < t \wedge v' \leq v \wedge t' \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$

proof (*induct* $\langle (tick-join, u, A, v) \rangle$ *arbitrary*: $t \ u \ u' \ v$)

case *Nil-setinterleaving_{ptick}-Nil* **thus** *?case by simp*

next

case (*ev-setinterleaving_{ptick}-Nil* $a \ u$)

from $\langle u' < ev \ a \ \# \ u \rangle$ **consider** $\langle u' = [] \mid u'' \text{ where } \langle u' = ev \ a \ \# \ u'' \rangle \langle u'' < u \rangle$

by (*metis Prefix-Order.prefix-Cons less-list-def*)

thus *?case*

proof *cases*

from *ev-setinterleaving_{ptick}-Nil.prem*(1)

show $\langle u' = [] \implies ?case \rangle$ **by** (*auto split: if-split-asm*)

next

fix u'' **assume** $\langle u' = ev \ a \ \# \ u'' \rangle \langle u'' < u \rangle$

from *ev-setinterleaving_{ptick}-Nil.prem*(1)

obtain t' **where** $\langle a \notin A \rangle \langle t = ev \ a \ \# \ t' \rangle \langle t' \text{ setinterleaves}_{\checkmark tick-join} ((u, []), A) \rangle$

by (*auto split: if-split-asm*)

from *ev-setinterleaving_{ptick}-Nil.hyps*[*OF* $\langle a \notin A \rangle$ *this*(3) $\langle u'' < u \rangle$]

obtain $t'' \ v'$ **where** $\langle t'' < t' \rangle \langle v' \leq [] \rangle \langle t'' \text{ setinterleaves}_{\checkmark tick-join} ((u'', v'), A) \rangle$

by *blast*

hence $\langle ev \ a \ \# \ t'' < t \wedge v' \leq [] \wedge ev \ a \ \# \ t'' \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$

by (*simp add:* $\langle u' = ev \ a \ \# \ u'' \rangle \langle t = ev \ a \ \# \ t' \rangle \langle a \notin A \rangle$)

thus *?case by blast*

qed

next

case (*tick-setinterleaving_{ptick}-Nil* $r \ u$) **thus** *?case by simp*

next

case (*Nil-setinterleaving_{ptick}-ev* $b \ v$) **thus** *?case by simp*

next

case (*Nil-setinterleaving_{ptick}-tick* $s \ v$) **thus** *?case by simp*

next

case (*ev-setinterleaving_{ptick}-ev* $a \ u \ b \ v$)

from $\langle u' < ev \ a \ \# \ u \rangle$ **consider** $\langle u' = [] \mid u'' \text{ where } \langle u' = ev \ a \ \# \ u'' \rangle \langle u'' < u \rangle$

by (*metis Prefix-Order.prefix-Cons less-list-def*)

thus *?case*

proof *cases*

from *ev-setinterleaving_{ptick}-ev.prem*(1)

show $\langle u' = [] \implies ?case \rangle$ **by** (*simp split: if-split-asm*) *force+*

next

fix u'' **assume** $\langle u' = ev \ a \ \# \ u'' \rangle \langle u'' < u \rangle$

hence $\langle ev \ a \ \# \ u'' < ev \ a \ \# \ u \rangle$ **by** *simp*

from *ev-setinterleaving_{ptick}-ev.prem*(1)

consider (*both-in*) t' **where** $\langle a \in A \rangle \langle b \in A \rangle \langle a = b \rangle \langle t = ev \ a \ \# \ t' \rangle$

$\langle t' \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \rangle$

```

| (inR-mvL) t' where  $\langle a \notin A \rangle \langle b \in A \rangle \langle t = \text{ev } a \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, \text{ev } b \# v), A) \rangle$ 
| (inL-mvR) t' where  $\langle a \in A \rangle \langle b \notin A \rangle \langle t = \text{ev } b \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\text{ev } a \# u, v), A) \rangle$ 
| (notin-mvL) t' where  $\langle a \notin A \rangle \langle b \notin A \rangle \langle t = \text{ev } a \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, \text{ev } b \# v), A) \rangle$ 
| (notin-mvR) t' where  $\langle a \notin A \rangle \langle b \notin A \rangle \langle t = \text{ev } b \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\text{ev } a \# u, v), A) \rangle$ 
by (auto split: if-split-asm)
thus ?case
proof cases
case both-in
from ev-setinterleavingptick-ev.hyps(1)[OF both-in(1-3, 5)  $\langle u'' < u \rangle$ ]
obtain t'' v' where  $\langle t'' < t' \wedge v' \leq v \wedge t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u'', v'),$ 
A)  $\rangle$  by blast
hence  $\langle \text{ev } a \# t'' < t \wedge \text{ev } b \# v' \leq \text{ev } b \# v \wedge$ 
 $\text{ev } a \# t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', \text{ev } b \# v'), A) \rangle$ 
by (simp add: both-in(2, 3, 4)  $\langle u' = \text{ev } a \# u'' \rangle$ )
thus ?thesis by blast
next
case inR-mvL
from ev-setinterleavingptick-ev.hyps(3)[OF inR-mvL(1, 2, 4)  $\langle u'' < u \rangle$ ]
obtain t'' v' where  $\langle t'' < t' \wedge v' \leq \text{ev } b \# v \wedge t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}}$ 
 $((u'', v'), A) \rangle$  by blast
hence  $\langle \text{ev } a \# t'' < t \wedge v' \leq \text{ev } b \# v \wedge$ 
 $\text{ev } a \# t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v'), A) \rangle$ 
by (cases v') (simp-all add: inR-mvL(1-3)  $\langle u' = \text{ev } a \# u'' \rangle$ )
thus ?thesis by blast
next
case inL-mvR
from ev-setinterleavingptick-ev.hyps(2)[OF inL-mvR(1, 2, 4)  $\langle \text{ev } a \# u'' <$ 
 $\text{ev } a \# u \rangle$ ]
obtain t'' v' where  $\langle t'' < t' \wedge v' \leq v \wedge t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\text{ev } a \#$ 
 $u'', v'), A) \rangle$  by blast
hence  $\langle \text{ev } b \# t'' < t \wedge \text{ev } b \# v' \leq \text{ev } b \# v \wedge$ 
 $\text{ev } b \# t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', \text{ev } b \# v'), A) \rangle$ 
by (simp add: inL-mvR(1-3)  $\langle u' = \text{ev } a \# u'' \rangle$ )
thus ?thesis by blast
next
case notin-mvL
from ev-setinterleavingptick-ev.hyps(4)[OF notin-mvL(1, 2, 4)  $\langle u'' < u \rangle$ ]
obtain t'' v' where  $\langle t'' < t' \wedge v' \leq \text{ev } b \# v \wedge t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}}$ 
 $((u'', v'), A) \rangle$  by blast
hence  $\langle \text{ev } a \# t'' < t \wedge v' \leq \text{ev } b \# v \wedge$ 
 $\text{ev } a \# t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u', v'), A) \rangle$ 
by (cases v') (simp-all add: notin-mvL(1-3)  $\langle u' = \text{ev } a \# u'' \rangle$ )
thus ?thesis by blast
next

```

```

    case notin-mvR
    from ev-setinterleavingptick-ev.hyps(5)[OF notin-mvR(1, 2, 4) ⟨ev a # u''
< ev a # u⟩]
    obtain t'' v' where ⟨t'' < t' ∧ v' ≤ v ∧ t'' setinterleaves✓tick-join ((ev a #
u'', v'), A)⟩ by blast
    hence ⟨ev b # t'' < t ∧ ev b # v' ≤ ev b # v ∧
ev b # t'' setinterleaves✓tick-join ((u', ev b # v'), A)⟩
    by (simp add: notin-mvR(1-3) ⟨u' = ev a # u''⟩)
    thus ?thesis by blast
qed
qed
next
case (ev-setinterleavingptick-tick a u s v)
from ⟨u' < ev a # u⟩ consider ⟨u' = []⟩ | u'' where ⟨u' = ev a # u''⟩ ⟨u'' <
u⟩
by (metis Prefix-Order.prefix-Cons less-list-def)
thus ?case
proof cases
from ev-setinterleavingptick-tick.prem(1)
show ⟨u' = [] ⟹ ?case⟩ by (simp split: if-split-asm) force+
next
fix u'' assume ⟨u' = ev a # u''⟩ ⟨u'' < u⟩
from ev-setinterleavingptick-tick.prem obtain t'
where ⟨a ∉ A⟩ ⟨t = ev a # t'⟩ ⟨t' setinterleaves✓tick-join ((u, ✓(s) # v), A)⟩
by (auto split: if-split-asm)
from ev-setinterleavingptick-tick.hyps[OF ⟨a ∉ A⟩ this(3) ⟨u'' < u⟩]
obtain t'' v' where ⟨t'' < t' ∧ v' ≤ ✓(s) # v ∧ t'' setinterleaves✓tick-join
((u'', v'), A)⟩ by blast
hence ⟨ev a # t'' < t ∧ v' ≤ ✓(s) # v ∧ ev a # t'' setinterleaves✓tick-join
((u', v'), A)⟩
by (cases v') (simp-all add: ⟨a ∉ A⟩ ⟨u' = ev a # u''⟩ ⟨t = ev a # t'⟩)
thus ?case by blast
qed
next
case (tick-setinterleavingptick-ev r u b v)
from ⟨u' < ✓(r) # u⟩ consider ⟨u' = []⟩ | u'' where ⟨u' = ✓(r) # u''⟩ ⟨u'' <
u⟩
by (metis Prefix-Order.prefix-Cons less-list-def)
thus ?case
proof cases
from tick-setinterleavingptick-ev.prem(1)
show ⟨u' = [] ⟹ ?case⟩ by (simp split: if-split-asm) force+
next
fix u'' assume ⟨u' = ✓(r) # u''⟩ ⟨u'' < u⟩
hence ⟨✓(r) # u'' < ✓(r) # u⟩ by simp
from tick-setinterleavingptick-ev.prem obtain t'
where ⟨b ∉ A⟩ ⟨t = ev b # t'⟩ ⟨t' setinterleaves✓tick-join ((✓(r) # u, v), A)⟩
by (auto split: if-split-asm)
from tick-setinterleavingptick-ev.hyps[OF ⟨b ∉ A⟩ this(3) ⟨✓(r) # u'' < ✓(r)

```

$\# u\rangle]$
obtain $t'' v'$ **where** $\langle t'' < t' \wedge v' \leq v \wedge t'' \text{ setinterleaves}_{\checkmark tick-join} ((\checkmark(r) \# u'', v'), A) \rangle$ **by** *blast*
hence $\langle ev\ b \# t'' < t \wedge ev\ b \# v' \leq ev\ b \# v \wedge ev\ b \# t'' \text{ setinterleaves}_{\checkmark tick-join} ((u', ev\ b \# v'), A) \rangle$
by (*simp add: $\langle b \notin A \rangle \langle u' = \checkmark(r) \# u'' \rangle \langle t = ev\ b \# t' \rangle$*)
thus *?case* **by** *blast*
qed
next
case (*tick-setinterleaving_{ptick}-tick* $r\ u\ s\ v$)
from $\langle u' < \checkmark(r) \# u \rangle$ **consider** $\langle u' = [] \rangle \mid u''$ **where** $\langle u' = \checkmark(r) \# u'' \rangle \langle u'' < u \rangle$
by (*metis Prefix-Order.prefix-Cons less-list-def*)
thus *?case*
proof *cases*
from *tick-setinterleaving_{ptick}-tick.prem*(1)
show $\langle u' = [] \implies ?case \rangle$ **by** (*force split: option.split-asm*)
next
fix u'' **assume** $\langle u' = \checkmark(r) \# u'' \rangle \langle u'' < u \rangle$
from *tick-setinterleaving_{ptick}-tick.prem*(1)
obtain $t' r-s$
where $\langle tick-join\ r\ s = \lfloor r-s \rfloor \rangle \langle t = \checkmark(r-s) \# t' \rangle \langle t' \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \rangle$
by (*auto split: option.split-asm*)
from *tick-setinterleaving_{ptick}-tick.hyps*[*OF this*(1, 3) $\langle u'' < u \rangle$]
obtain $t'' v'$ **where** $\langle t'' < t' \wedge v' \leq v \wedge t'' \text{ setinterleaves}_{\checkmark tick-join} ((u'', v'), A) \rangle$ **by** *blast*
hence $\langle \checkmark(r-s) \# t'' < t \wedge \checkmark(s) \# v' \leq \checkmark(s) \# v \wedge \checkmark(r-s) \# t'' \text{ setinterleaves}_{\checkmark tick-join} ((u', \checkmark(s) \# v'), A) \rangle$
by (*simp add: $\langle tick-join\ r\ s = \lfloor r-s \rfloor \rangle \langle u' = \checkmark(r) \# u'' \rangle \langle t = \checkmark(r-s) \# t' \rangle$*)
thus *?case* **by** *blast*
qed
qed

corollary *setinterleaves_{ptick}-less-prefixR* :
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \implies v' < v \implies \exists t' u'. t' < t \wedge u' \leq u \wedge t' \text{ setinterleaves}_{\checkmark tick-join} ((u', v'), A) \rangle$
using *setinterleaves_{ptick}-less-prefixL* *setinterleaves_{ptick}-sym* **by** *blast*

lemma *setinterleaves_{ptick}-le-prefixLR* :
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((u, v), A) \implies u' \leq u \implies v' \leq v \implies (\exists t' \leq t. \exists v'' \leq v'. t' \text{ setinterleaves}_{\checkmark tick-join} ((u', v''), A)) \vee (\exists t' \leq t. \exists u'' \leq u'. t' \text{ setinterleaves}_{\checkmark tick-join} ((u'', v'), A)) \rangle$
proof (*induct $\langle tick-join, u, A, v \rangle$ arbitrary: $t\ u\ u'\ v\ v'$*)
case *Nil-setinterleaving_{ptick}-Nil* **thus** *?case* **by** *simp*

```

next
  case (ev-setinterleavingptick-Nil a u) thus ?case by simp fastforce
next
  case (tick-setinterleavingptick-Nil r u) thus ?case by simp
next
  case (Nil-setinterleavingptick-ev b v) thus ?case by simp fastforce
next
  case (Nil-setinterleavingptick-tick s v) thus ?case by simp
next
  case (ev-setinterleavingptick-ev a u b v)
  show ?case
  proof (cases ⟨u' = [] ∨ v' = []⟩)
    show ⟨u' = [] ∨ v' = [] ⟹ ?case⟩ by force
  next
    assume ⟨¬ (u' = [] ∨ v' = [])⟩
    with ev-setinterleavingptick-ev.prem(2, 3)
    obtain u'' v'' where ⟨u' = ev a # u''⟩ ⟨u'' ≤ u⟩ ⟨v' = ev b # v''⟩ ⟨v'' ≤ v⟩
      by (meson Prefix-Order.prefix-Cons)
    from ev-setinterleavingptick-ev.prem(1)
    consider (both-in) t' where ⟨a ∈ A⟩ ⟨b ∈ A⟩ ⟨a = b⟩ ⟨t = ev a # t'⟩
      ⟨t' setinterleavestick-join ((u, v), A)⟩
    | (inR-mvL) t' where ⟨a ∉ A⟩ ⟨b ∈ A⟩ ⟨t = ev a # t'⟩
      ⟨t' setinterleavestick-join ((u, ev b # v), A)⟩
    | (inL-mvR) t' where ⟨a ∈ A⟩ ⟨b ∉ A⟩ ⟨t = ev b # t'⟩
      ⟨t' setinterleavestick-join ((ev a # u, v), A)⟩
    | (notin-mvL) t' where ⟨a ∉ A⟩ ⟨b ∉ A⟩ ⟨t = ev a # t'⟩
      ⟨t' setinterleavestick-join ((u, ev b # v), A)⟩
    | (notin-mvR) t' where ⟨a ∉ A⟩ ⟨b ∉ A⟩ ⟨t = ev b # t'⟩
      ⟨t' setinterleavestick-join ((ev a # u, v), A)⟩
    by (auto split: if-split-asm)
  thus ?case
  proof cases
    case both-in
    from ev-setinterleavingptick-ev.hyps(1)[OF both-in(1-3, 5) ⟨u'' ≤ u⟩ ⟨v'' ≤
v⟩]
    show ?thesis
    proof (elim disjE exE conjE)
      fix t'' v'''
      assume ⟨t'' ≤ t'⟩ ⟨v''' ≤ v''⟩ ⟨t'' setinterleavestick-join ((u'', v'''), A)⟩
      hence ⟨ev b # t'' ≤ t ∧ ev b # v''' ≤ v' ∧
        ev b # t'' setinterleavestick-join ((u', ev b # v'''), A)⟩
      by (simp add: ⟨u' = ev a # u''⟩ ⟨v' = ev b # v''⟩ both-in(2-4))
      thus ?thesis by blast
    next
      fix t'' u'''
      assume ⟨t'' ≤ t'⟩ ⟨u''' ≤ u''⟩ ⟨t'' setinterleavestick-join ((u''', v'''), A)⟩
      hence ⟨ev a # t'' ≤ t ∧ ev a # u''' ≤ u' ∧
        ev a # t'' setinterleavestick-join ((ev a # u''', v'), A)⟩

```

```

    by (simp add: ⟨u' = ev a # u''⟩ ⟨v' = ev b # v''⟩ both-in(2-4))
    thus ?thesis by blast
  qed
next
  case inR-mvL
  from ev-setinterleavingptick-ev.hyps(3)[OF inR-mvL(1, 2, 4) ⟨u'' ≤ u⟩ ⟨v' ≤
ev b # v⟩]
  show ?thesis
  proof (elim disjE exE conjE)
    fix t'' v'''
    assume ⟨t'' ≤ t'⟩ ⟨v''' ≤ v'⟩ ⟨t'' setinterleaves✓tick-join ((u'', v'''), A)⟩
    hence ⟨ev a # t'' ≤ t ∧ v''' ≤ v' ∧
      ev a # t'' setinterleaves✓tick-join ((u', v'''), A)⟩
    by (cases v''') (simp-all add: ⟨u' = ev a # u''⟩ ⟨v' = ev b # v''⟩ inR-mvL(1,
3))
    thus ?thesis by blast
  next
    fix t'' u'''
    assume ⟨t'' ≤ t'⟩ ⟨u''' ≤ u'⟩ ⟨t'' setinterleaves✓tick-join ((u''', v'), A)⟩
    hence ⟨ev a # t'' ≤ t ∧ ev a # u''' ≤ u' ∧
      ev a # t'' setinterleaves✓tick-join ((ev a # u''', v'), A)⟩
    by (simp add: ⟨u' = ev a # u''⟩ ⟨v' = ev b # v''⟩ inR-mvL(1, 3))
    thus ?thesis by blast
  qed
next
  case inL-mvR
  from ev-setinterleavingptick-ev.hyps(2)[OF inL-mvR(1, 2, 4) ⟨u' ≤ ev a #
u⟩ ⟨v'' ≤ v⟩]
  show ?thesis
  proof (elim disjE exE conjE)
    fix t'' v'''
    assume ⟨t'' ≤ t'⟩ ⟨v''' ≤ v''⟩ ⟨t'' setinterleaves✓tick-join ((u', v'''), A)⟩
    hence ⟨ev b # t'' ≤ t ∧ ev b # v''' ≤ v' ∧
      ev b # t'' setinterleaves✓tick-join ((u', ev b # v'''), A)⟩
    by (simp add: ⟨u' = ev a # u''⟩ ⟨v' = ev b # v''⟩ inL-mvR(2, 3))
    thus ?thesis by blast
  next
    fix t'' u'''
    assume ⟨t'' ≤ t'⟩ ⟨u''' ≤ u'⟩ ⟨t'' setinterleaves✓tick-join ((u''', v''), A)⟩
    hence ⟨ev b # t'' ≤ t ∧ u''' ≤ u' ∧
      ev b # t'' setinterleaves✓tick-join ((u''', v'), A)⟩
    by (cases u''') (simp-all add: ⟨u' = ev a # u''⟩ ⟨v' = ev b # v''⟩ inL-mvR(2,
3))
    thus ?thesis by blast
  qed
next
  case notin-mvL
  from ev-setinterleavingptick-ev.hyps(4)[OF notin-mvL(1, 2, 4) ⟨u'' ≤ u⟩ ⟨v'
≤ ev b # v⟩]

```

```

show ?thesis
proof (elim disjE exE conjE)
  fix t'' v'''
  assume <t'' ≤ t'> <v''' ≤ v'> <t'' setinterleaves✓tick-join ((u'', v'''), A)>
  hence <ev a # t'' ≤ t ∧ v''' ≤ v' ∧
        ev a # t'' setinterleaves✓tick-join ((u', v'''), A)>
  by (cases v''') (simp-all add: <u' = ev a # u''> <v' = ev b # v''> notin-mvL(1,
3))
    thus ?thesis by blast
next
fix t'' u'''
assume <t'' ≤ t'> <u''' ≤ u''> <t'' setinterleaves✓tick-join ((u''', v'), A)>
hence <ev a # t'' ≤ t ∧ ev a # u''' ≤ u' ∧
      ev a # t'' setinterleaves✓tick-join ((ev a # u''', v'), A)>
  by (simp add: <u' = ev a # u''> <v' = ev b # v''> notin-mvL(1, 3))
  thus ?thesis by blast
qed
next
case notin-mvR
from ev-setinterleavingptick-ev.hyps(5)[OF notin-mvR(1, 2, 4) <u' ≤ ev a #
u> <v'' ≤ v>]
show ?thesis
proof (elim disjE exE conjE)
  fix t'' v'''
  assume <t'' ≤ t'> <v''' ≤ v''> <t'' setinterleaves✓tick-join ((u', v'''), A)>
  hence <ev b # t'' ≤ t ∧ ev b # v''' ≤ v' ∧
        ev b # t'' setinterleaves✓tick-join ((u', ev b # v'''), A)>
  by (simp add: <u' = ev a # u''> <v' = ev b # v''> notin-mvR(2, 3))
  thus ?thesis by blast
next
fix t'' u'''
assume <t'' ≤ t'> <u''' ≤ u'> <t'' setinterleaves✓tick-join ((u''', v''), A)>
hence <ev b # t'' ≤ t ∧ u''' ≤ u' ∧
      ev b # t'' setinterleaves✓tick-join ((u''', v'), A)>
  by (cases u''') (simp-all add: <u' = ev a # u''> <v' = ev b # v''> notin-mvR(2,
3))
    thus ?thesis by blast
qed
qed
qed
next
case (ev-setinterleavingptick-tick a u s v)
show ?case
proof (cases <u' = [] ∨ v' = []>)
  show <u' = [] ∨ v' = [] ⇒ ?case> by force
next
assume <¬ (u' = [] ∨ v' = [])>
with ev-setinterleavingptick-tick.prem(2, 3)
obtain u'' v'' where <u' = ev a # u''> <u'' ≤ u> <v' = ✓(s) # v''> <v'' ≤ v>

```

```

    by (meson Prefix-Order.prefix-Cons)
  from ev-setinterleavingptick-tick.prem(1)
  obtain t' where ⟨a ∉ A⟩ ⟨t = ev a # t'⟩
    ⟨t' setinterleaves✓tick-join ((u, ✓(s) # v), A)⟩
    by (auto split: if-split-asm)
  from ev-setinterleavingptick-tick.hyps[OF this(1, 3) ⟨u'' ≤ u⟩ ⟨v' ≤ ✓(s) # v⟩]
  show ?case
  proof (elim disjE exE conjE)
    fix t'' v''' assume ⟨t'' ≤ t'⟩ ⟨v''' ≤ v'⟩ ⟨t'' setinterleaves✓tick-join ((u'', v'''),
A)⟩
    hence ⟨ev a # t'' ≤ t ∧ v''' ≤ v' ∧ ev a # t'' setinterleaves✓tick-join ((u',
v'''), A)⟩
    by (cases v''') (simp-all add: ⟨a ∉ A⟩ ⟨t = ev a # t'⟩ ⟨u' = ev a # u''⟩ ⟨v'
= ✓(s) # v''⟩)
    thus ?case by blast
  next
    fix t'' u''' assume ⟨t'' ≤ t'⟩ ⟨u''' ≤ u'⟩ ⟨t'' setinterleaves✓tick-join ((u''', v'),
A)⟩
    hence ⟨ev a # t'' ≤ t ∧ ev a # u''' ≤ u' ∧ ev a # t'' setinterleaves✓tick-join
((ev a # u''', v'), A)⟩
    by (simp add: ⟨a ∉ A⟩ ⟨t = ev a # t'⟩ ⟨u' = ev a # u''⟩ ⟨v' = ✓(s) # v''⟩)
    thus ?case by blast
  qed
qed
next
case (tick-setinterleavingptick-ev r u b v)
show ?case
proof (cases ⟨u' = [] ∨ v' = []⟩)
  show ⟨u' = [] ∨ v' = [] ⟹ ?case⟩ by force
next
  assume ⟨¬ (u' = [] ∨ v' = [])⟩
  with tick-setinterleavingptick-ev.prem(2, 3)
  obtain u'' v'' where ⟨u' = ✓(r) # u''⟩ ⟨u'' ≤ u⟩ ⟨v' = ev b # v''⟩ ⟨v'' ≤ v⟩
    by (meson Prefix-Order.prefix-Cons)
  from tick-setinterleavingptick-ev.prem(1)
  obtain t' where ⟨b ∉ A⟩ ⟨t = ev b # t'⟩
    ⟨t' setinterleaves✓tick-join ((✓(r) # u, v), A)⟩
    by (auto split: if-split-asm)
  from tick-setinterleavingptick-ev.hyps[OF this(1, 3) ⟨u' ≤ ✓(r) # u⟩ ⟨v'' ≤ v⟩]
  show ?case
  proof (elim disjE exE conjE)
    fix t'' v''' assume ⟨t'' ≤ t'⟩ ⟨v''' ≤ v''⟩ ⟨t'' setinterleaves✓tick-join ((u', v'''),
A)⟩
    hence ⟨ev b # t'' ≤ t ∧ ev b # v''' ≤ v' ∧ ev b # t'' setinterleaves✓tick-join
((u', ev b # v'''), A)⟩
    by (simp add: ⟨b ∉ A⟩ ⟨t = ev b # t'⟩ ⟨u' = ✓(r) # u''⟩ ⟨v' = ev b # v''⟩)
    thus ?case by blast
  next
    fix t'' u''' assume ⟨t'' ≤ t'⟩ ⟨u''' ≤ u'⟩ ⟨t'' setinterleaves✓tick-join ((u''', v''),

```



```

A)⟩
  hence ⟨ev b # t'' ≤ t ∧ u''' ≤ u' ∧ ev b # t'' setinterleaves✓tick-join ((u''',
v'), A)⟩
    by (cases u''') (simp-all add: ⟨b ∉ A⟩ ⟨t = ev b # t'⟩ ⟨u' = ✓(r) # u''⟩ ⟨v'
= ev b # v''⟩)
    thus ?case by blast
qed
qed
next
case (tick-setinterleavingptick-tick r u s v)
show ?case
proof (cases ⟨u' = [] ∨ v' = []⟩)
  show ⟨u' = [] ∨ v' = [] ⟹ ?case⟩ by force
next
assume ⟨¬ (u' = [] ∨ v' = [])⟩
with tick-setinterleavingptick-tick.prem(2, 3)
obtain u'' v'' where ⟨u' = ✓(r) # u''⟩ ⟨u'' ≤ u⟩ ⟨v' = ✓(s) # v''⟩ ⟨v'' ≤ v⟩
  by (meson Prefix-Order.prefix-Cons)
from tick-setinterleavingptick-tick.prem(1)
obtain t' r-s where ⟨t = ✓(r-s) # t'⟩ ⟨tick-join r s = [r-s]⟩
  ⟨t' setinterleaves✓tick-join ((u, v), A)⟩
  by (auto split: option.split-asm)
from tick-setinterleavingptick-tick.hyps[OF this(2, 3) ⟨u'' ≤ u⟩ ⟨v'' ≤ v⟩]
show ?case
proof (elim disjE exE conjE)
  fix t'' v'''
  assume ⟨t'' ≤ t'⟩ ⟨v''' ≤ v''⟩ ⟨t'' setinterleaves✓tick-join ((u'', v'''), A)⟩
  hence ⟨✓(r-s) # t'' ≤ t ∧ ✓(s) # v''' ≤ v' ∧
    ✓(r-s) # t'' setinterleaves✓tick-join ((u', ✓(s) # v'''), A)⟩
    by (simp add: ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s) # t'⟩
      ⟨u' = ✓(r) # u''⟩ ⟨v' = ✓(s) # v''⟩)
  thus ?case by blast
next
fix t'' u'''
assume ⟨t'' ≤ t'⟩ ⟨u''' ≤ u''⟩ ⟨t'' setinterleaves✓tick-join ((u''', v''), A)⟩
hence ⟨✓(r-s) # t'' ≤ t ∧ ✓(r) # u''' ≤ u' ∧
  ✓(r-s) # t'' setinterleaves✓tick-join ((✓(r) # u''', v'), A)⟩
  by (simp add: ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s) # t'⟩
    ⟨u' = ✓(r) # u''⟩ ⟨v' = ✓(s) # v''⟩)
  thus ?case by blast
qed
qed
qed

```

4.1.6 Hiding Events

lemma *setinterleaves_{ptick}-trace-hide* :

$$\langle t \text{ setinterleaves}_{\text{✓tick-join}} ((u, v), S) \implies \text{trace-hide } t \text{ (ev } \text{' } A) \text{ setinterleaves}_{\text{✓tick-join}}$$

```

      ((trace-hide u (ev ' A), trace-hide v (ev ' A)), S)
proof (induct ⟨(tick-join, u, S, v)⟩ arbitrary: t u v)
  case Nil-setinterleavingptick-Nil
  thus ?case by simp
next
  case (ev-setinterleavingptick-Nil a u)
  from ev-setinterleavingptick-Nil.premss obtain t' where ⟨a ∉ S⟩ ⟨t = ev a # t'⟩
    ⟨t' setinterleaves✓ tick-join ((u, []), S)⟩ by (auto split: if-split-asm)
  from ev-setinterleavingptick-Nil.hyps[OF this(1, 3)]
  show ?case by (simp add: image-iff[of ⟨ev -⟩] ⟨a ∉ S⟩ ⟨t = ev a # t'⟩)
next
  case (tick-setinterleavingptick-Nil r u)
  from tick-setinterleavingptick-Nil have False by simp
  thus ?case ..
next
  case (Nil-setinterleavingptick-ev b v)
  from Nil-setinterleavingptick-ev.premss obtain t' where ⟨b ∉ S⟩ ⟨t = ev b # t'⟩
    ⟨t' setinterleaves✓ tick-join (([], v), S)⟩ by (auto split: if-split-asm)
  from Nil-setinterleavingptick-ev.hyps[OF this(1, 3)]
  show ?case by (simp add: image-iff[of ⟨ev -⟩] ⟨b ∉ S⟩ ⟨t = ev b # t'⟩)
next
  case (Nil-setinterleavingptick-tick s v)
  from Nil-setinterleavingptick-tick.premss have False by simp
  thus ?case ..
next
  case (ev-setinterleavingptick-ev a u b v)
  from ev-setinterleavingptick-ev.premss
  consider (both-in) t' where ⟨a ∈ S⟩ ⟨b ∈ S⟩ ⟨a = b⟩ ⟨t = ev a # t'⟩
    ⟨t' setinterleaves✓ tick-join ((u, v), S)⟩
  | (inR-mvL) t' where ⟨a ∉ S⟩ ⟨b ∈ S⟩ ⟨t = ev a # t'⟩
    ⟨t' setinterleaves✓ tick-join ((u, ev b # v), S)⟩
  | (inL-mvR) t' where ⟨a ∈ S⟩ ⟨b ∉ S⟩ ⟨t = ev b # t'⟩
    ⟨t' setinterleaves✓ tick-join ((ev a # u, v), S)⟩
  | (notin-mvL) t' where ⟨a ∉ S⟩ ⟨b ∉ S⟩ ⟨t = ev a # t'⟩
    ⟨t' setinterleaves✓ tick-join ((u, ev b # v), S)⟩
  | (notin-mvR) t' where ⟨a ∉ S⟩ ⟨b ∉ S⟩ ⟨t = ev b # t'⟩
    ⟨t' setinterleaves✓ tick-join ((ev a # u, v), S)⟩
  by (auto split: if-split-asm)
  thus ?case
proof cases
  case both-in
  from ev-setinterleavingptick-ev.hyps(1)[OF both-in(1-3, 5)]
  show ?thesis by (simp add: both-in(2-5) image-iff[of ⟨ev -⟩])
next
  case inR-mvL
  from ev-setinterleavingptick-ev.hyps(3)[OF inR-mvL(1, 2, 4)]
  show ?thesis by (cases ⟨trace-hide v (ev ' A)⟩
    (auto simp add: inR-mvL(1-3) setinterleavingptick-simps
      split: if-split-asm eventptick.split))

```

```

next
  case inL-mvR
  from ev-setinterleavingptick-ev.hyps(2)[OF inL-mvR(1, 2, 4)]
  show ?thesis by (cases ⟨trace-hide u (ev ‘A)⟩
    (auto simp add: inL-mvR(1-3) setinterleavingptick-simps
      split: if-split-asm eventptick.split)
  next
    case notin-mvL
    from ev-setinterleavingptick-ev.hyps(4)[OF notin-mvL(1, 2, 4)]
    show ?thesis by (cases ⟨trace-hide v (ev ‘A)⟩
      (auto simp add: notin-mvL(1-3) setinterleavingptick-simps
        split: if-split-asm eventptick.split)
    next
      case notin-mvR
      from ev-setinterleavingptick-ev.hyps(5)[OF notin-mvR(1, 2, 4)]
      show ?thesis by (cases ⟨trace-hide u (ev ‘A)⟩
        (auto simp add: notin-mvR(1-3) setinterleavingptick-simps
          split: if-split-asm eventptick.split)
      qed
    next
      case (ev-setinterleavingptick-tick a u s v)
      from ev-setinterleavingptick-tick.prem obtain t' where ⟨a ∉ S⟩ ⟨t = ev a #
t'⟩
        ⟨t' setinterleaves ✓tick-join ((u, ✓(s) # v), S)⟩ by (auto split: if-split-asm)
      from ev-setinterleavingptick-tick.hyps[OF this(1, 3)]
      show ?case by (simp add: image-iff[of ⟨ev -⟩] image-iff[of ⟨✓(-)⟩] ⟨a ∉ S⟩ ⟨t =
ev a # t'⟩)
    next
      case (tick-setinterleavingptick-ev r u b v)
      from tick-setinterleavingptick-ev.prem obtain t' where ⟨b ∉ S⟩ ⟨t = ev b # t'⟩
        ⟨t' setinterleaves ✓tick-join ((✓(r) # u, v), S)⟩ by (auto split: if-split-asm)
      from tick-setinterleavingptick-ev.hyps[OF this(1, 3)]
      show ?case by (simp add: image-iff[of ⟨ev -⟩] image-iff[of ⟨✓(-)⟩] ⟨b ∉ S⟩ ⟨t =
ev b # t'⟩)
    next
      case (tick-setinterleavingptick-tick r u s v)
      from tick-setinterleavingptick-tick.prem
      obtain r-s t' where ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s) # t'⟩
        ⟨t' setinterleaves ✓tick-join ((u, v), S)⟩ by (auto split: option.split-asm)
      from tick-setinterleavingptick-tick.hyps[OF this(1, 3)]
      show ?case by (simp add: image-iff[of ⟨✓(-)⟩] ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s)
# t'⟩)
      qed
  qed

```

```

lemma trace-hide-map-map-eventptick :
  ⟨trace-hide (map (map-eventptick f g) t) S =
  map (map-eventptick f g) (trace-hide t (map-eventptick f g -‘ S))⟩
  by (induct t) simp-all

```

lemma *tickFree-trace-hide-map-ev-comp-of-ev* :
 $\langle tF\ t \implies \text{trace-hide} (\text{map} (ev \circ \text{of-ev})\ t) (ev\ 'A) =$
 $\text{map} (ev \circ \text{of-ev}) (\text{trace-hide}\ t (ev\ 'A)) \rangle$
by (*induct t*) (*auto simp add: image-iff*)

lemma *tickFree-disjoint-setinterleaves_{ptick}-appendL* :
 $\langle tF\ u1 \implies \{a. ev\ a \in \text{set}\ u1\} \cap A = \{\} \implies t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u2, v),$
 $A) \implies \text{map} (ev \circ \text{of-ev})\ u1\ @\ t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u1\ @\ u2, v), A) \rangle$
proof (*induct u1*)
case *Nil*
from *Nil.premis(3)* **show** *?case* **by** *simp*
next
case (*Cons e u1*)
from *Cons.premis(1, 2)* **obtain** *a*
where $\langle e = ev\ a \rangle \langle a \notin A \rangle \langle tF\ u1 \rangle \langle \{a. ev\ a \in \text{set}\ u1\} \cap A = \{\} \rangle$
by (*auto simp add: disjoint-iff is-ev-def*)
from *Cons.hyps[OF this(3, 4) Cons.premis(3)]*
have $\langle \text{map} (ev \circ \text{of-ev})\ u1\ @\ t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u1\ @\ u2, v), A) \rangle$.
with $\langle e = ev\ a \rangle \langle a \notin A \rangle$
show *?case* **by** (*cases v*)
 $(\text{auto simp add: setinterleaving}_{\text{ptick-simps comp-def split: event}_{\text{ptick-split}}})$
qed

corollary *tickFree-disjoint-setinterleaves_{ptick}-appendR* :
 $\langle \llbracket tF\ v1; \{a. ev\ a \in \text{set}\ v1\} \cap A = \{\}; t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u, v2), A) \rrbracket$
 $\implies \text{map} (ev \circ \text{of-ev})\ v1\ @\ t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u, v1\ @\ v2), A) \rangle$
by (*metis setinterleaves_{ptick}-sym tickFree-disjoint-setinterleaves_{ptick}-appendL*)

lemma *tickFree-disjoint-setinterleaves_{ptick}-append-tailL* :
 $\langle t\ @\ \text{map} (ev \circ \text{of-ev})\ u2\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u1\ @\ u2, v), A) \rangle$
if $\langle tF\ u2 \rangle \langle \{a. ev\ a \in \text{set}\ u2\} \cap A = \{\} \rangle \langle t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u1, v), A) \rangle$
proof –
have $\langle t\ @\ \text{map} (ev \circ \text{of-ev})\ u2\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((u1\ @\ u2, v), A) \longleftrightarrow$
 $\text{map} (ev \circ \text{of-ev})\ (\text{rev}\ u2)\ @\ \text{rev}\ t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((\text{rev}\ u2\ @\ \text{rev}\ u1,$
 $\text{rev}\ v), A) \rangle$
by (*subst rev-setinterleaves_{ptick}-rev-rev-iff[symmetric]*)
 $(\text{simp add: rev-map})$
also have ...
proof (*rule tickFree-disjoint-setinterleaves_{ptick}-appendL*)
show $\langle tF\ (\text{rev}\ u2) \rangle$ **by** (*simp add: that(1)*)
next
show $\langle \{a. ev\ a \in \text{set}\ (\text{rev}\ u2)\} \cap A = \{\} \rangle$ **by** (*simp add: that(2)*)
next
show $\langle \text{rev}\ t\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((\text{rev}\ u1, \text{rev}\ v), A) \rangle$

by (simp add: rev-setinterleaves_{ptick}-rev-rev-iff that(3))
qed
finally show ?thesis .
qed

corollary *tickFree-disjoint-setinterleaves_{ptick}-append-tailR* :
 $\langle \llbracket tF\ v2; \{a.\ ev\ a \in set\ v2\} \cap A = \{\};\ t\ setinterleaves_{\checkmark tick-join} ((u, v1), A) \rrbracket$
 $\implies t\ @\ map\ (ev \circ of-ev)\ v2\ setinterleaves_{\checkmark tick-join} ((u, v1\ @\ v2), A) \rangle$
by (metis setinterleaves_{ptick}-sym tickFree-disjoint-setinterleaves_{ptick}-append-tailL)

lemma *disjoint-trace-hide-setinterleaves_{ptick}* :
 $\langle t\ setinterleaves_{\checkmark tick-join} ((trace-hide\ u\ (ev\ 'A),\ trace-hide\ v\ (ev\ 'A)),\ S) \implies$
 $\exists t'.\ t = trace-hide\ t'\ (ev\ 'A) \wedge$
 $t'\ setinterleaves_{\checkmark tick-join} ((u, v), S) \rangle$ if $\langle A \cap S = \{\} \rangle$
for $t :: \langle 'a, 't \rangle trace_{ptick}$ and $u :: \langle 'a, 'r \rangle trace_{ptick}$ and $v :: \langle 'a, 's \rangle trace_{ptick}$
proof –
let $?th = trace-hide$ and $?A = \langle ev\ 'A \rangle$
show $\langle t\ setinterleaves_{\checkmark tick-join} ((?th\ u\ ?A,\ ?th\ v\ ?A),\ S) \implies \exists t'.\ t = ?th\ t'\ ?A \wedge t'\ setinterleaves_{\checkmark tick-join} ((u, v), S) \rangle$
proof (induct $\langle (tick-join, u, S, v) \rangle$ arbitrary: $t\ u\ v$)
case *Nil-setinterleaving_{ptick}-Nil*
then show ?case by simp
next
case (ev-setinterleaving_{ptick}-Nil a u)
from ev-setinterleaving_{ptick}-Nil.prem
consider t' where $\langle a \notin S \rangle \langle a \notin A \rangle \langle t = ev\ a \# t' \rangle$
 $\langle t'\ setinterleaves_{\checkmark tick-join} ((?th\ u\ ?A,\ ?th\ []\ ?A),\ S) \rangle$
| $\langle a \in A \rangle \langle t\ setinterleaves_{\checkmark tick-join} ((?th\ u\ ?A,\ ?th\ []\ ?A),\ S) \rangle$
by (auto split: if-split-asm)
thus ?case
proof cases
fix t' assume $\langle a \notin S \rangle \langle a \notin A \rangle \langle t = ev\ a \# t' \rangle$
 $\langle t'\ setinterleaves_{\checkmark tick-join} ((?th\ u\ ?A,\ ?th\ []\ ?A),\ S) \rangle$
from ev-setinterleaving_{ptick}-Nil.hyps[OF this(1, 4)] obtain t''
where $\langle t' = ?th\ t''\ ?A \wedge t''\ setinterleaves_{\checkmark tick-join} ((u, []), S) \rangle$..
hence $\langle t = ?th\ (ev\ a \# t'')\ ?A \wedge ev\ a \# t''\ setinterleaves_{\checkmark tick-join} ((ev\ a \#$
 $u, []), S) \rangle$
by (simp add: $\langle a \notin A \rangle \langle a \notin S \rangle \langle t = ev\ a \# t' \rangle$ image-iff[of $\langle ev\ - \rangle$])
thus ?case ..
next
assume $\langle a \in A \rangle$
with $\langle A \cap S = \{\} \rangle$ have $\langle a \notin S \rangle$ by blast
moreover assume $\langle t\ setinterleaves_{\checkmark tick-join} ((?th\ u\ ?A,\ ?th\ []\ ?A),\ S) \rangle$
ultimately obtain t' where $\langle t = ?th\ t'\ ?A \rangle \langle t'\ setinterleaves_{\checkmark tick-join} ((u,$

```

 $\square$ ),  $S$ )  

  using  $ev\text{-}setinterleaving_{ptick}\text{-}Nil.hyps$  by blast  

  hence  $\langle t = ?th (ev\ a \# t')\ ?A \wedge ev\ a \# t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((ev\ a \#$   

 $u, \square), S) \rangle$   

  by (simp add:  $\langle a \in A \rangle \langle a \notin S \rangle$ )  

  thus ?case ..  

qed  

next  

case (tick-setinterleaving $_{ptick}\text{-}Nil\ r\ u$ )  

from tick-setinterleaving $_{ptick}\text{-}Nil.prem$ s have False by (simp add: image-iff[ $of$   

 $\langle \checkmark(-) \rangle$ ])  

thus ?case ..  

next  

case (Nil-setinterleaving $_{ptick}\text{-}ev\ b\ v$ )  

from Nil-setinterleaving $_{ptick}\text{-}ev.prem$ s  

consider  $t'$  where  $\langle b \notin S \rangle \langle b \notin A \rangle \langle t = ev\ b \# t' \rangle$   

 $\langle t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((?th\ \square\ ?A, ?th\ v\ ?A), S) \rangle$   

 $| \langle b \in A \rangle \langle t \text{ setinterleaves}_{\checkmark tick\text{-}join} ((?th\ \square\ ?A, ?th\ v\ ?A), S) \rangle$   

by (auto split: if-split-asm)  

thus ?case  

proof cases  

fix  $t'$  assume  $\langle b \notin S \rangle \langle b \notin A \rangle \langle t = ev\ b \# t' \rangle$   

 $\langle t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((?th\ \square\ ?A, ?th\ v\ ?A), S) \rangle$   

from Nil-setinterleaving $_{ptick}\text{-}ev.hyps[OF\ this(1, 4)]$  obtain  $t''$   

where  $\langle t' = ?th\ t''\ ?A \wedge t'' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((\square, v), S) \rangle ..$   

hence  $\langle t = ?th (ev\ b \# t'')\ ?A \wedge ev\ b \# t'' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((\square, ev\ b$   

 $\# v), S) \rangle$   

by (simp add:  $\langle b \notin A \rangle \langle b \notin S \rangle \langle t = ev\ b \# t' \rangle$  image-iff[ $of\ \langle ev\ - \rangle$ ])  

thus ?case ..  

next  

assume  $\langle b \in A \rangle$   

with  $\langle A \cap S = \{\} \rangle$  have  $\langle b \notin S \rangle$  by blast  

moreover assume  $\langle t \text{ setinterleaves}_{\checkmark tick\text{-}join} ((?th\ \square\ ?A, ?th\ v\ ?A), S) \rangle$   

ultimately obtain  $t'$  where  $\langle t = ?th\ t'\ ?A \rangle \langle t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((\square,$   

 $v), S) \rangle$   

using Nil-setinterleaving $_{ptick}\text{-}ev.hyps$  by blast  

hence  $\langle t = ?th (ev\ b \# t')\ ?A \wedge ev\ b \# t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((\square, ev\ b$   

 $\# v), S) \rangle$   

by (simp add:  $\langle b \in A \rangle \langle b \notin S \rangle$ )  

thus ?case ..  

qed  

next  

case (Nil-setinterleaving $_{ptick}\text{-}tick\ s\ v$ )  

from Nil-setinterleaving $_{ptick}\text{-}tick.prem$ s have False by (simp add: image-iff[ $of$   

 $\langle \checkmark(-) \rangle$ ])  

thus ?case ..  

next  

case (ev-setinterleaving $_{ptick}\text{-}ev\ a\ u\ b\ v$ )

```

```

show ?case
proof (cases ⟨a ∈ A⟩; cases ⟨b ∈ A⟩)
  assume ⟨a ∈ A⟩ ⟨b ∈ A⟩
  with ev-setinterleavingptick-ev.prem
  have * : ⟨t setinterleaves✓tick-join ((?th (ev a # u) ?A, ?th v ?A), S)⟩
    ⟨t setinterleaves✓tick-join ((?th u ?A, ?th (ev b # v) ?A), S)⟩ by simp-all
  from ⟨A ∩ S = {}⟩ ⟨a ∈ A⟩ ⟨b ∈ A⟩ have ⟨a ∉ S⟩ ⟨b ∉ S⟩ by blast+
  from ev-setinterleavingptick-ev.hyps(4)[OF this *(2)]
    ev-setinterleavingptick-ev.hyps(5)[OF this *(1)]
  obtain t' where ⟨t = ?th t' ?A⟩
    ⟨t' setinterleaves✓tick-join ((ev a # u, v), S)⟩ ∨
    ⟨t' setinterleaves✓tick-join ((u, ev b # v), S)⟩ by blast
  hence ⟨t = ?th (ev b # t') ?A ∧ ev b # t' setinterleaves✓tick-join ((ev a #
u, ev b # v), S)⟩ ∨
    t = ?th (ev a # t') ?A ∧ ev a # t' setinterleaves✓tick-join ((ev a # u,
ev b # v), S)⟩
    by (auto simp add: ⟨a ∈ A⟩ ⟨b ∈ A⟩ ⟨a ∉ S⟩ ⟨b ∉ S⟩)
  thus ?case by blast
next
  assume ⟨a ∈ A⟩ ⟨b ∉ A⟩
  with ev-setinterleavingptick-ev.prem
  have * : ⟨t setinterleaves✓tick-join ((?th u ?A, ?th (ev b # v) ?A), S)⟩ by
simp
  from ⟨A ∩ S = {}⟩ ⟨a ∈ A⟩ have ⟨a ∉ S⟩ by blast
  from ev-setinterleavingptick-ev.hyps(3)[OF ⟨a ∉ S⟩ - *(1)]
    ev-setinterleavingptick-ev.hyps(4)[OF ⟨a ∉ S⟩ - *] obtain t'
    where ⟨t = ?th t' ?A⟩ ⟨t' setinterleaves✓tick-join ((u, ev b # v), S)⟩ by
blast
  hence ⟨t = ?th (ev a # t') ?A ∧
    ev a # t' setinterleaves✓tick-join ((ev a # u, ev b # v), S)⟩
    by (simp add: ⟨a ∈ A⟩ ⟨a ∉ S⟩)
  thus ?case ..
next
  assume ⟨a ∉ A⟩ ⟨b ∈ A⟩
  with ev-setinterleavingptick-ev.prem
  have * : ⟨t setinterleaves✓tick-join ((?th (ev a # u) ?A, ?th v ?A), S)⟩ by
simp
  from ⟨A ∩ S = {}⟩ ⟨b ∈ A⟩ have ⟨b ∉ S⟩ by blast
  from ev-setinterleavingptick-ev.hyps(2)[OF - ⟨b ∉ S⟩ *]
    ev-setinterleavingptick-ev.hyps(5)[OF - ⟨b ∉ S⟩ *] obtain t'
    where ⟨t = ?th t' ?A⟩ ⟨t' setinterleaves✓tick-join ((ev a # u, v), S)⟩ by
blast
  hence ⟨t = ?th (ev b # t') ?A ∧
    ev b # t' setinterleaves✓tick-join ((ev a # u, ev b # v), S)⟩
    by (simp add: ⟨b ∈ A⟩ ⟨b ∉ S⟩)
  thus ?case ..
next
  assume ⟨a ∉ A⟩ ⟨b ∉ A⟩
  hence ⟨?th (ev a # u) ?A = ev a # ?th u ?A⟩

```

```

  <?th (ev b # v) ?A = ev b # ?th v ?A> by auto
from ev-setinterleavingptick-ev.premis[unfolded this]
have <t setinterleaves✓tick-join ((ev a # ?th u ?A, ev b # ?th v ?A), S)> .
then consider (mv-both) t' where <a ∈ S> <b ∈ S> <a = b> <t = ev a # t'>
  <t' setinterleaves✓tick-join ((?th u ?A, ?th v ?A), S)>
| (mvL) t' where <a ∉ S> <t = ev a # t'>
  <t' setinterleaves✓tick-join ((?th u ?A, ev b # ?th v ?A), S)>
| (mvR) t' where <b ∉ S> <t = ev b # t'>
  <t' setinterleaves✓tick-join ((ev a # ?th u ?A, ?th v ?A), S)>
  by (auto split: if-split-asm)
thus ?case
proof cases
  case mv-both
  from ev-setinterleavingptick-ev.hyps(1)[OF mv-both(1-3, 5)] obtain t''
    where <t' = ?th t'' ?A ∧ t'' setinterleaves✓tick-join ((u, v), S)> ..
  hence <t = ?th (ev b # t'') ?A ∧ ev b # t'' setinterleaves✓tick-join ((ev a
# u, ev b # v), S)>
    by (simp add: mv-both(2-4) <b ∉ A> image-iff[of <ev ->] )
  thus ?thesis ..
next
  case mvL
  from ev-setinterleavingptick-ev.hyps(3, 4)
    [OF mvL(1) - mvL(3)[folded <?th (ev b # v) ?A = ev b # ?th v ?A>]]
  obtain t'' where <t' = ?th t'' ?A>
    <t'' setinterleaves✓tick-join ((u, ev b # v), S)> by blast
  hence <t = ?th (ev a # t'') ?A ∧
    ev a # t'' setinterleaves✓tick-join ((ev a # u, ev b # v), S)>
    by (simp add: mvL(1, 2) <a ∉ A> image-iff[of <ev ->])
  thus ?thesis ..
next
  case mvR
  from ev-setinterleavingptick-ev.hyps(2, 5)
    [OF - mvR(1) mvR(3)[folded <?th (ev a # u) ?A = ev a # ?th u ?A>]]
  obtain t'' where <t' = ?th t'' ?A>
    <t'' setinterleaves✓tick-join ((ev a # u, v), S)> by blast
  hence <t = ?th (ev b # t'') ?A ∧
    ev b # t'' setinterleaves✓tick-join ((ev a # u, ev b # v), S)>
    by (simp add: mvR(1, 2) <b ∉ A> image-iff[of <ev ->])
  thus ?thesis ..
qed
qed
next
  case (ev-setinterleavingptick-tick a u s v)
  from ev-setinterleavingptick-tick.premis
  consider t' where <a ∉ S> <a ∉ A> <t = ev a # t'>
    <t' setinterleaves✓tick-join ((?th u ?A, ?th (✓(s) # v) ?A), S)>
  | <a ∈ A> <t setinterleaves✓tick-join ((?th u ?A, ?th (✓(s) # v) ?A), S)>
    by (auto split: if-split-asm)

```



```

thus ?case
proof cases
  fix t' assume  $\langle a \notin S \rangle \langle a \notin A \rangle \langle t = \text{ev } a \# t' \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?th \ u \ ?A, ?th (\checkmark(s) \# v) \ ?A), S) \rangle$ 
  from  $\text{ev-setinterleaving}_{\text{ptick-tick.hyps}}[OF \ \text{this}(1, 4)]$  obtain t''
    where  $\langle t' = ?th \ t'' \ ?A \rangle \langle t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, \checkmark(s) \# v), S) \rangle$  by
blast
  hence  $\langle t = ?th \ (\text{ev } a \# t'') \ ?A \wedge$ 
     $\text{ev } a \# t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\text{ev } a \# u, \checkmark(s) \# v), S) \rangle$ 
  by (simp add:  $\langle a \notin A \rangle \langle a \notin S \rangle \langle t = \text{ev } a \# t' \rangle$  image-iff[of  $\langle \text{ev } - \rangle$ ])
  thus ?case ..
next
  assume  $\langle a \in A \rangle$ 
  with  $\langle A \cap S = \{\} \rangle$  have  $\langle a \notin S \rangle$  by blast
  moreover assume  $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?th \ u \ ?A, ?th (\checkmark(s) \# v) \ ?A),$ 
S)  $\rangle$ 
  ultimately obtain t' where  $\langle t = ?th \ t' \ ?A \rangle \langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u,$ 
 $\checkmark(s) \# v), S) \rangle$ 
  using  $\text{ev-setinterleaving}_{\text{ptick-tick.hyps}}$  by blast
  hence  $\langle t = ?th \ (\text{ev } a \# t') \ ?A \wedge \text{ev } a \# t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\text{ev } a \#$ 
 $u, \checkmark(s) \# v), S) \rangle$ 
  by (simp add:  $\langle a \in A \rangle \langle a \notin S \rangle$ )
  thus ?case ..
qed
next
case ( $\text{tick-setinterleaving}_{\text{ptick-ev } r \ u \ b \ v}$ )
from  $\text{tick-setinterleaving}_{\text{ptick-ev.prem}}$ 
consider t' where  $\langle b \notin S \rangle \langle b \notin A \rangle \langle t = \text{ev } b \# t' \rangle$ 
   $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?th (\checkmark(r) \# u) \ ?A, ?th \ v \ ?A), S) \rangle$ 
  |  $\langle b \in A \rangle \langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?th (\checkmark(r) \# u) \ ?A, ?th \ v \ ?A), S) \rangle$ 
  by (auto split: if-split-asm)
thus ?case
proof cases
  fix t' assume  $\langle b \notin S \rangle \langle b \notin A \rangle \langle t = \text{ev } b \# t' \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?th (\checkmark(r) \# u) \ ?A, ?th \ v \ ?A), S) \rangle$ 
  from  $\text{tick-setinterleaving}_{\text{ptick-ev.hyps}}[OF \ \text{this}(1, 4)]$  obtain t''
    where  $\langle t' = ?th \ t'' \ ?A \rangle \langle t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\checkmark(r) \# u, v), S) \rangle$  by
blast
  hence  $\langle t = ?th \ (\text{ev } b \# t'') \ ?A \wedge$ 
     $\text{ev } b \# t'' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\checkmark(r) \# u, \text{ev } b \# v), S) \rangle$ 
  by (simp add:  $\langle b \notin A \rangle \langle b \notin S \rangle \langle t = \text{ev } b \# t' \rangle$  image-iff[of  $\langle \text{ev } - \rangle$ ])
  thus ?case ..
next
  assume  $\langle b \in A \rangle$ 
  with  $\langle A \cap S = \{\} \rangle$  have  $\langle b \notin S \rangle$  by blast
  moreover assume  $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?th (\checkmark(r) \# u) \ ?A, ?th \ v \ ?A),$ 
S)  $\rangle$ 
  ultimately obtain t' where  $\langle t = ?th \ t' \ ?A \rangle \langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((\checkmark(r)$ 

```

```

# u, v), S)⟩
  using tick-setinterleavingptick-ev.hyps by blast
  hence ⟨t = ?th (ev b # t') ?A ∧ ev b # t' setinterleaves✓ tick-join ((✓(r) #
u, ev b # v), S)⟩
    by (simp add: ⟨b ∈ A⟩ ⟨b ∉ S⟩)
  thus ?case ..
qed
next
case (tick-setinterleavingptick-tick r u s v)
from tick-setinterleavingptick-tick.prem obtain r-s t'
  where ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s) # t'⟩
  ⟨t' setinterleaves✓ tick-join ((?th u ?A, ?th v ?A), S)⟩
  by (auto split: if-split-asm option.split-asm)
from tick-setinterleavingptick-tick.hyps[OF this(1, 3)] obtain t''
  where ⟨t' = ?th t'' ?A⟩ ⟨t'' setinterleaves✓ tick-join ((u, v), S)⟩ by blast
  hence ⟨t = ?th (✓(r-s) # t'') ?A ∧
    ✓(r-s) # t'' setinterleaves✓ tick-join ((✓(r) # u, ✓(s) # v), S)⟩
    by (simp add: ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s) # t'⟩ image-iff[of ⟨✓(-)⟩])
  thus ?case ..
qed
qed

```

lemma *setinterleaves_{ptick}-inj-map-map-event_{ptick}-iff-weak* :

```

⟨map (map-eventptick f id) t setinterleaves✓ tick-join
((map (map-eventptick f id) u, map (map-eventptick f id) v), f ' A) ⟷
t setinterleaves✓ tick-join ((u, v), A)⟩ if ⟨inj f⟩
by (induct ⟨(tick-join, u, A, v)⟩ arbitrary: t u v)
  (auto simp add: image-iff map-eventptick-eq-ev-iff map-eventptick-eq-tick-iff
    dest!: injD[OF ⟨inj f⟩] split: option.split-asm)

```

lemma *setinterleaves_{ptick}-inj-map-map-event_{ptick}-iff-strong* :

```

⟨t setinterleaves✓ tick-join
((map (map-eventptick f id) u, map (map-eventptick f id) v), f ' A) ⟷
(∃ t'. t = map (map-eventptick f id) t' ∧
t' setinterleaves✓ tick-join ((u, v), A))⟩ if ⟨inj f⟩

```

— We could probably prove a stronger version with *inj-on* f ($A \cup \{a. \text{ev } a \in \text{set } u \vee \text{ev } a \in \text{set } v\}$) instead of *inj* f .

proof —

```

let ?map = ⟨map (map-eventptick f id)⟩
have ⟨t setinterleaves✓ tick-join ((?map u, ?map v), f ' A) ⟷ ∃ t'. t = ?map t'⟩
proof (induct ⟨(tick-join, u, A, v)⟩ arbitrary: t u v)
  case Nil-setinterleavingptick-Nil
  thus ?case by simp
next
case (ev-setinterleavingptick-Nil a u)

```

```

from ev-setinterleavingptick-Nil.prems obtain  $t'$ 
  where  $\langle a \notin A \rangle \langle t = \text{ev } (f \ a) \# t' \rangle \langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?map \ u, ?map$ 
 $\square), f \ 'A) \rangle$ 
    by (auto split: if-split-asm)
from ev-setinterleavingptick-Nil.hyps[OF this(1, 3)]
obtain  $t''$  where  $\langle t' = ?map \ t'' \rangle \dots$ 
hence  $\langle t = ?map \ (\text{ev } a \# t'') \rangle$  by (simp add: \langle t = \text{ev } (f \ a) \# t' \rangle)
thus ?case ..
next
  case (tick-setinterleavingptick-Nil r u)
from tick-setinterleavingptick-Nil.prems have False by simp
thus ?case ..
next
  case (Nil-setinterleavingptick-ev b v)
from Nil-setinterleavingptick-ev.prems obtain  $t'$ 
  where  $\langle b \notin A \rangle \langle t = \text{ev } (f \ b) \# t' \rangle \langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?map \ [], ?map$ 
 $v), f \ 'A) \rangle$ 
    by (auto split: if-split-asm)
from Nil-setinterleavingptick-ev.hyps[OF this(1, 3)]
obtain  $t''$  where  $\langle t' = ?map \ t'' \rangle \dots$ 
hence  $\langle t = ?map \ (\text{ev } b \# t'') \rangle$  by (simp add: \langle t = \text{ev } (f \ b) \# t' \rangle)
thus ?case ..
next
  case (Nil-setinterleavingptick-tick s v)
from Nil-setinterleavingptick-tick.prems have False by simp
thus ?case ..
next
  case (ev-setinterleavingptick-ev a u b v)
from ev-setinterleavingptick-ev.prems
consider (mv-left)  $t'$  where  $\langle a \notin A \rangle \langle t = \text{ev } (f \ a) \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?map \ u, ?map \ (\text{ev } b \# v)), f \ 'A) \rangle$ 
| (mv-right)  $t'$  where  $\langle b \notin A \rangle \langle t = \text{ev } (f \ b) \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?map \ (\text{ev } a \# u), ?map \ v), f \ 'A) \rangle$ 
| (mv-both)  $t'$  where  $\langle a \in A \rangle \langle b \in A \rangle \langle a = b \rangle \langle t = \text{ev } (f \ b) \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((?map \ u, ?map \ v), f \ 'A) \rangle$ 
by (auto simp add: image-iff split: if-split-asm dest!: injD[OF \langle inj f \rangle])
thus ?case
proof cases
  case mv-left
from ev-setinterleavingptick-ev.hyps(3, 4)[OF mv-left(1) - mv-left(3)]
obtain  $t''$  where  $\langle t' = ?map \ t'' \rangle$  by blast
hence  $\langle t = ?map \ (\text{ev } a \# t'') \rangle$  by (simp add: mv-left(2))
thus ?thesis ..
next
  case mv-right
from ev-setinterleavingptick-ev.hyps(2, 5)[OF - mv-right(1, 3)]
obtain  $t''$  where  $\langle t' = ?map \ t'' \rangle$  by blast
hence  $\langle t = ?map \ (\text{ev } b \# t'') \rangle$  by (simp add: mv-right(2))
thus ?thesis ..

```

```

next
  case mv-both
  from ev-setinterleavingptick-ev.hyps(1)[OF mv-both(1-3, 5)]
  obtain t'' where ⟨t' = ?map t'⟩ ..
  hence ⟨t = ?map (ev b # t')⟩ by (simp add: mv-both(4))
  thus ?thesis ..
qed
next
  case (ev-setinterleavingptick-tick a u s v)
  from ev-setinterleavingptick-tick.prem obtain t'
  where ⟨a ∉ A⟩ ⟨t = ev (f a) # t'⟩ ⟨t' setinterleaves✓ tick-join ((?map u, ?map
(✓(s) # v)), f ' A)⟩
  by (auto split: if-split-asm)
  from ev-setinterleavingptick-tick.hyps[OF this(1, 3)]
  obtain t'' where ⟨t' = ?map t''⟩ ..
  hence ⟨t = ?map (ev a # t'')⟩ by (simp add: ⟨t = ev (f a) # t'⟩)
  thus ?case ..
next
  case (tick-setinterleavingptick-ev r u b v)
  from tick-setinterleavingptick-ev.prem obtain t'
  where ⟨b ∉ A⟩ ⟨t = ev (f b) # t'⟩ ⟨t' setinterleaves✓ tick-join ((?map (✓(r)
# u), ?map v), f ' A)⟩
  by (auto split: if-split-asm)
  from tick-setinterleavingptick-ev.hyps[OF this(1, 3)]
  obtain t'' where ⟨t' = ?map t''⟩ ..
  hence ⟨t = ?map (ev b # t'')⟩ by (simp add: ⟨t = ev (f b) # t'⟩)
  thus ?case ..
next
  case (tick-setinterleavingptick-tick r u s v)
  from tick-setinterleavingptick-tick.prem obtain r-s t'
  where ⟨tick-join r s = [r-s]⟩ ⟨t = ✓(r-s) # t'⟩
  ⟨t' setinterleaves✓ tick-join ((?map u, ?map v), f ' A)⟩
  by (auto split: option.split-asm)
  from tick-setinterleavingptick-tick.hyps[OF this(1, 3)]
  obtain t'' where ⟨t' = ?map t''⟩ ..
  hence ⟨t = ?map (✓(r-s) # t'')⟩ by (simp add: ⟨t = ✓(r-s) # t'⟩)
  thus ?case ..
qed
with setinterleavesptick-inj-map-map-eventptick-iff-weak[OF ⟨inj f⟩]
show ?thesis by blast
qed

```

```

lemma setinterleavesptick-append-setinterleavesptick :
  ⟨t1 @ t2 setinterleaves✓ tick-join ((u1 @ u2, v1 @ v2), A)⟩
  if ⟨t1 setinterleaves✓ tick-join ((u1, v1), A)⟩
  and ⟨t2 setinterleaves✓ tick-join ((u2, v2), A)⟩

```

```

using that(1) proof (induct  $\langle \text{tick-join}, u1, A, v1 \rangle$  arbitrary:  $t1\ u1\ v1$ )
case Nil-setinterleavingptick-Nil
from Nil-setinterleavingptick-Nil.prem(1) have  $\langle t1 = [] \rangle$  by simp
with that(2) show ?case by simp
next
case (ev-setinterleavingptick-Nil a u1)
from ev-setinterleavingptick-Nil.prem(1) obtain  $t1'$  where  $\langle a \notin A \rangle \langle t1 = ev$ 
 $a \# t1' \rangle$ 
 $\langle t1' \text{ setinterleaves}_{\checkmark}^{\text{tick-join}} ((u1, []), A) \rangle$  by (auto split: if-split-asm)
from ev-setinterleavingptick-Nil.hyps[OF this(1, 3)]
show ?case
by (cases v2)
(auto simp add:  $\langle a \notin A \rangle \langle t1 = ev\ a \# t1' \rangle$  setinterleavingptick-simps
split: eventptick.split)
next
case (tick-setinterleavingptick-Nil r u1)
from tick-setinterleavingptick-Nil.prem(1) have False by simp
thus ?case ..
next
case (Nil-setinterleavingptick-ev b v1)
from Nil-setinterleavingptick-ev.prem(1) obtain  $t1'$  where  $\langle b \notin A \rangle \langle t1 = ev\ b$ 
 $\# t1' \rangle$ 
 $\langle t1' \text{ setinterleaves}_{\checkmark}^{\text{tick-join}} ([], v1), A \rangle$  by (auto split: if-split-asm)
from Nil-setinterleavingptick-ev.hyps[OF this(1, 3)]
show ?case
by (cases u2)
(auto simp add:  $\langle b \notin A \rangle \langle t1 = ev\ b \# t1' \rangle$  setinterleavingptick-simps
split: eventptick.split)
next
case (Nil-setinterleavingptick-tick s v1)
from Nil-setinterleavingptick-tick.prem(1) have False by simp
thus ?case ..
next
case (ev-setinterleavingptick-ev a u1 b v1)
from ev-setinterleavingptick-ev.prem
consider (mv-both)  $t'$  where  $\langle a \in A \rangle \langle b \in A \rangle \langle a = b \rangle \langle t1 = ev\ a \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark}^{\text{tick-join}} ((u1, v1), A) \rangle$ 
| (mvL)  $t'$  where  $\langle a \notin A \rangle \langle t1 = ev\ a \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark}^{\text{tick-join}} ((u1, ev\ b \# v1), A) \rangle$ 
| (mvR)  $t'$  where  $\langle b \notin A \rangle \langle t1 = ev\ b \# t' \rangle$ 
 $\langle t' \text{ setinterleaves}_{\checkmark}^{\text{tick-join}} ((ev\ a \# u1, v1), A) \rangle$ 
by (auto split: if-split-asm)
thus ?case
proof cases
case mv-both
from ev-setinterleavingptick-ev.hyps(1)[OF mv-both(1-3, 5)]
show ?thesis by (simp add: mv-both(2-4))
next
case mvL

```

```

    from ev-setinterleavingptick-ev.hyps(3, 4)[OF mvL(1) - mvL(3)]
    show ?thesis by (simp add: mvL(1, 2))
  next
    case mvR
    from ev-setinterleavingptick-ev.hyps(2, 5)[OF - mvR(1, 3)]
    show ?thesis by (simp add: mvR(1, 2))
  qed
next
  case (ev-setinterleavingptick-tick a u1 s v1)
  from ev-setinterleavingptick-tick.prem(1)
  obtain t1' where ⟨a ∉ A⟩ ⟨t1 = ev a # t1'⟩
    ⟨t1' setinterleaves✓ tick-join ((u1, ✓(s) # v1), A)⟩ by (auto split: if-split-asm)
  from ev-setinterleavingptick-tick.hyps[OF this(1, 3)]
  show ?case
    by (cases v2)
      (auto simp add: ⟨a ∉ A⟩ ⟨t1 = ev a # t1'⟩ setinterleavingptick-sims
        split: eventptick.split)
next
  case (tick-setinterleavingptick-ev r u1 b v1)
  from tick-setinterleavingptick-ev.prem(1) obtain t1' where ⟨b ∉ A⟩ ⟨t1 = ev
b # t1'⟩
    ⟨t1' setinterleaves✓ tick-join ((✓(r) # u1, v1), A)⟩ by (auto split: if-split-asm)
  from tick-setinterleavingptick-ev.hyps[OF this(1, 3)]
  show ?case
    by (cases u2)
      (auto simp add: ⟨b ∉ A⟩ ⟨t1 = ev b # t1'⟩ setinterleavingptick-sims
        split: eventptick.split)
next
  case (tick-setinterleavingptick-tick r u1 s v1)
  from tick-setinterleavingptick-tick.prem(1) obtain r-s t1'
    where ⟨tick-join r s = ⌊r-s⌋⟩ ⟨t1 = ✓(r-s) # t1'⟩
    ⟨t1' setinterleaves✓ tick-join ((u1, v1), A)⟩
    by (auto split: option.split-asm)
  from tick-setinterleavingptick-tick.hyps[OF this(1, 3)]
  show ?case by (simp add: ⟨tick-join r s = ⌊r-s⌋⟩ ⟨t1 = ✓(r-s) # t1'⟩)
qed

```

```

lemma setinterleavesptick-set-subsetL :
  ⟨t setinterleaves✓ tick-join ((u, v), A) ⟹
  {a. ev a ∈ set (drop n u)} ⊆ {a. ev a ∈ set (drop n t)}⟩
proof (induct t arbitrary: n u v)
  case Nil
  thus ?case by (auto dest: Nil-setinterleavesptick)
next
  case (Cons e t)
  from Cons.prem consider (mv-left) a u' where ⟨a ∉ A⟩ ⟨e = ev a⟩ ⟨u = ev a

```

```

# u'
  <t setinterleaves✓tick-join ((u', v), A)>
  | (mv-right) a v' where <a ∉ A> <e = ev a> <v = ev a # v'>
    <t setinterleaves✓tick-join ((u, v'), A)>
  | (mv-both-ev) a u' v' where <a ∈ A> <e = ev a> <u = ev a # u'> <v = ev a #
v'>
    <t setinterleaves✓tick-join ((u', v'), A)>
  | (mv-both-tick) r s r-s u' v' where <tick-join r s = ⌊r-s⌋> <e = ✓(r-s)>
    <u = ✓(r) # u'> <v = ✓(s) # v'> <t setinterleaves✓tick-join ((u', v'), A)>
by (cases e) (auto elim: Cons-ev-setinterleavesptickE Cons-tick-setinterleavesptickE)
thus ?case
proof cases
  case mv-left
    from Cons.hyps[OF mv-left(4)] show ?thesis
    by (cases n, simp-all add: mv-left(2, 3) subset-iff) (metis drop0)
  next
    case mv-right
      from Cons.hyps[OF mv-right(4)] show ?thesis
      by (cases n, simp-all add: subset-iff)
        (metis drop0, meson Suc-n-not-le-n in-mono nle-le set-drop-subset-set-drop)
    next
      case mv-both-ev
        from Cons.hyps[OF mv-both-ev(5)] show ?thesis
        by (cases n, simp-all add: mv-both-ev(2, 3) subset-iff) (metis drop0)
      next
        case mv-both-tick
          from Cons.hyps[OF mv-both-tick(5)] show ?thesis
          by (cases n, simp-all add: mv-both-tick(3) subset-iff) (metis drop0)
    qed
  qed

lemma setinterleavesptick-set-subsetR :
  <t setinterleaves✓tick-join ((u, v), A) ⇒
  {a. ev a ∈ set (drop n v)} ⊆ {a. ev a ∈ set (drop n t)}>
by (rule setinterleavesptick-set-subsetL)
  (fact setinterleavesptick-sym[THEN iffD2])

```

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 ∈ X-P ∧ ev a ∈ X-Q ∨ (a ∈ A ∧ (ev a ∈ X-P ∨ ev a ∈
 X-Q))}> ∪
 {✓(r-s) | r s r-s. tick-join r s = ⌊r-s⌋ ∧ (✓(r) ∈ X-P ∨ ✓(s) ∈ X-Q)} ∪
 — This is the last addition: since we generalize with the parameter *tick-join*,

we must add the following term to refuse the unreachable ticks.

$$\{\checkmark(r-s) \mid r-s. \nexists r s. \text{ tick-join } r s = \lfloor r-s \rfloor\}$$

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* :: $\langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$ (**infixl** $\langle \otimes \checkmark \rangle$ 100)
 assumes *inj-tick-join* :
 $\langle r \otimes \checkmark s = \lfloor r-s \rfloor \implies r' \otimes \checkmark s' = \lfloor r-s \rfloor \implies r' = r \wedge s' = s \rangle$
begin

sublocale *Sync_{ptick}-locale-sym* : *Sync_{ptick}-locale* $\langle \lambda s r. r \otimes \checkmark s \rangle$
by *unfold-locale* (*simp add: inj-tick-join*)

lift-definition *Sync_{ptick}* ::

$$\langle [(\langle 'a, 'r \rangle \text{ process}_{\text{ptick}}, 'a \text{ set}, (\langle 'a, 's \rangle \text{ process}_{\text{ptick}}] \Rightarrow (\langle 'a, 't \rangle \text{ process}_{\text{ptick}}) \rangle$$

$$\langle \langle \cdot \rangle \llbracket \cdot \rrbracket \checkmark \cdot \rangle [70, 0, 71] 70 \rangle$$

is $\langle \lambda P A Q.$

$$\{ \langle (t, X). \exists t-P t-Q X-P X-Q.$$

$$(t-P, X-P) \in \mathcal{F} P \wedge (t-Q, X-Q) \in \mathcal{F} Q \wedge$$

$$t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), A) \wedge$$

$$X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\otimes \checkmark) X-P A X-Q \} \cup$$

$$\{ \langle t @ u, X \rangle \mid t u t-P t-Q X.$$

$$\text{ftF } u \wedge (tF t \vee u = []) \wedge t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), A) \wedge$$

$$(t-P \in \mathcal{D} P \wedge t-Q \in \mathcal{T} Q \vee t-P \in \mathcal{T} P \wedge t-Q \in \mathcal{D} Q) \},$$

$$\{ \langle t @ u \mid t u t-P t-Q.$$

$$\text{ftF } u \wedge (tF t \vee u = []) \wedge t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), A) \wedge$$

$$(t-P \in \mathcal{D} P \wedge t-Q \in \mathcal{T} Q \vee t-P \in \mathcal{T} P \wedge t-Q \in \mathcal{D} Q) \} \rangle$$

proof –

show $\langle ?thesis P A Q \rangle$

(**is** $\langle \text{is-process} (?f, ?d) \rangle$) **for** *P* **and** *Q* :: $\langle (\langle 'a, 's \rangle \text{ process}_{\text{ptick}}) \rangle$ **and** *A*

proof (*unfold is-process-def FAILURES-def DIVERGENCES-def fst-conv snd-conv, intro conjI impI allI*)

have $\langle ([], \{\}) \in \mathcal{F} P \rangle$ **and** $\langle ([], \{\}) \in \mathcal{F} Q \rangle$ **by** (*simp-all add: is-processT1*)

with *Nil-setinterleaving_{ptick}-Nil* **show** $\langle ([], \{\}) \in ?f \rangle$ **by** *fast*

next

show $\langle (t, X) \in ?f \implies \text{ftF } t \rangle$ **for** *t X*

by *simp* (*metis* (*no-types, opaque-lifting*) *D-T F-imp-front-tickFree T-imp-front-tickFree append.right-neutral front-tickFree-append front-tickFree-setinterleaves_{ptick}-iff*)

next

fix *t u* **assume** $\langle t @ u, \{\} \rangle \in ?f$

then consider (*fail*) *t-P t-Q X-P X-Q* **where**

$$\langle (t-P, X-P) \in \mathcal{F} P \rangle \langle (t-Q, X-Q) \in \mathcal{F} Q \rangle \langle t @ u \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), A) \rangle$$

| (*div*) *t' u' t-P t-Q* **where**

$$\langle t @ u = t' @ u' \rangle \langle \text{ftF } u' \rangle \langle tF t' \vee u' = [] \rangle \langle t' \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q),$$

$A\rangle$
 $\langle t\text{-}P \in \mathcal{D} P \wedge t\text{-}Q \in \mathcal{T} Q \vee t\text{-}P \in \mathcal{T} P \wedge t\text{-}Q \in \mathcal{D} Q \rangle$ **by** *simp blast*
thus $\langle (t, \{\}) \in ?f \rangle$
proof *cases*
case *fail*
from *fail*(3) **obtain** $t' u'$
where $*$: $\langle t' \leq t\text{-}P \rangle \langle u' \leq t\text{-}Q \rangle \langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t', u'), A) \rangle$
by (*auto dest!*: *append-setinterleaves_{ptick} intro: prefixI*)
from *fail*(1, 2) $*(1, 2)$ *F-T is-processT3-TR* **have** $\langle t' \in \mathcal{T} P \rangle \langle u' \in \mathcal{T} Q \rangle$
by *blast+*
thus $\langle (t, \{\}) \in ?f \rangle$ **by** *simp (metis T-F-spec *(3))*
next
case *div*
show $\langle (t, \{\}) \in ?f \rangle$
proof (*cases* $\langle \text{length } t' \leq \text{length } t \rangle$)
assume $\langle \text{length } t' \leq \text{length } t \rangle$
with *div*(1-3) **have** $\langle \text{ftF } (\text{take } (\text{length } t - \text{length } t') u') \wedge$
 $(\text{tF } t' \vee \text{take } (\text{length } t - \text{length } t') u' = []) \wedge$
 $t = t' @ \text{take } (\text{length } t - \text{length } t') u' \rangle$
by (*simp add: append-eq-conv-conj*)
(*metis append-take-drop-id front-tickFree-dw-closed*)
with *div*(4, 5) **show** $\langle (t, \{\}) \in ?f \rangle$ **by** *blast*
next
assume $\langle \neg \text{length } t' \leq \text{length } t \rangle$
with *div* **obtain** r' **where** $\langle t' = t @ r' \rangle$
by (*metis append-eq-append-conv-if append-take-drop-id*)
with *div*(4) **obtain** $t'' u''$
where $*$: $\langle t'' \leq t\text{-}P \rangle \langle u'' \leq t\text{-}Q \rangle \langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t'', u''), A) \rangle$
by (*auto dest!*: *append-setinterleaves_{ptick} intro: prefixI*)
from $*(1, 2)$ **have** $\langle t'' \in \mathcal{T} P \wedge u'' \in \mathcal{T} Q \rangle$ **by** (*meson D-T div(5)*
is-processT3-TR)
hence $\$$: $\langle (t'', \{\}) \in \mathcal{F} P \rangle \langle (u'', \{\}) \in \mathcal{F} Q \rangle$ **by** (*simp-all add: T-F*)
have $\$ \$$: $\langle \{ \text{ev } a \mid a. \text{ev } a \in \{\} \wedge \text{ev } a \in \{\} \vee (a \in A \wedge (\text{ev } a \in \{\} \vee \text{ev } a \in \{\})) \} \rangle \cup$
 $\langle \checkmark(r \otimes \checkmark s) \mid r s. \checkmark(r) \in \{\} \vee \checkmark(s) \in \{\} \rangle = \{\} \rangle$ **by** *simp*
show $\langle (t, \{\}) \in ?f \rangle$ **by** (*auto intro!*: $\$ *(3)$)
qed
qed
next
{ **fix** $t X Y$
assume $\langle (t, Y) \in ?f \wedge X \subseteq Y \rangle$
then consider $\langle t \in ?d \rangle$
| (*fail*) $t\text{-}P t\text{-}Q X\text{-}P X\text{-}Q$ **where**
 $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} Q \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), A) \rangle$
 $\langle Y \subseteq \text{super-ref-Sync}_{\text{ptick}} (\otimes\checkmark) X\text{-}P A X\text{-}Q \rangle$ **by** *blast*
thus $\langle (t, X) \in ?f \rangle$
proof *cases*
show $\langle t \in ?d \implies (t, X) \in ?f \rangle$ **by** *blast*

```

next
case fail
define  $X-P'$  where  $\langle X-P' \equiv X-P \cap (\{ev\ a \mid a. ev\ a \in X\} \cup$ 
 $\{\checkmark(r) \mid r\ s\ r-s. r \otimes \checkmark\ s = \lfloor r-s \rfloor \wedge \checkmark(r-s) \in X\}) \rangle$ 
define  $X-Q'$  where  $\langle X-Q' \equiv X-Q \cap (\{ev\ a \mid a. ev\ a \in X\} \cup$ 
 $\{\checkmark(s) \mid r\ s\ r-s. r \otimes \checkmark\ s = \lfloor r-s \rfloor \wedge \checkmark(r-s) \in X\}) \rangle$ 
have  $\langle (t-P, X-P') \in \mathcal{F}\ P \rangle$  unfolding  $X-P'$ -def by (meson fail(1) inf-le1
process-charn)
moreover have  $\langle (t-Q, X-Q') \in \mathcal{F}\ Q \rangle$  unfolding  $X-Q'$ -def by (meson
fail(2) inf-le1 process-charn)
moreover have  $\langle X \subseteq super-ref-Sync_{ptick} (\otimes \checkmark) X-P' \ A\ X-Q' \rangle$ 
by (subst  $\langle (t, Y) \in ?f \wedge X \subseteq Y \rangle [THEN\ conjunct2, THEN\ Int-absorb1,$ 
symmetric])
(use fail(4) in  $\langle fastforce\ simp\ add: X-P'-def\ X-Q'-def\ subset-iff\ su-$ 
per-ref-Syncptick-def  $\rangle$ )
ultimately show  $\langle (t, X) \in ?f \rangle$  using fail(3) by simp blast
qed } note processT4 = this

fix t X Y
assume  $\langle (t, X) \in ?f \wedge (\forall e. e \in Y \longrightarrow (t @ [e], \{\}) \notin ?f) \rangle$ 
then consider  $\langle t \in ?d \rangle \mid \langle (t, X) \in ?f \wedge t \notin ?d \rangle$  by linarith
thus  $\langle (t, X \cup Y) \in ?f \rangle$ 
proof cases
show  $\langle t \in ?d \implies (t, X \cup Y) \in ?f \rangle$  by blast
next
assume  $\langle (t, X) \in ?f \wedge t \notin ?d \rangle$ 
then obtain  $t-P\ X-P\ t-Q\ X-Q$ 
where  $assms : \langle (t-P, X-P) \in \mathcal{F}\ P \rangle \langle (t-Q, X-Q) \in \mathcal{F}\ Q \rangle$ 
 $\langle t\ setinterleaves_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), A) \rangle$ 
 $\langle X \subseteq super-ref-Sync_{ptick} (\otimes \checkmark) X-P\ A\ X-Q \rangle$  by blast
have  $assms5 : \langle e \in Y \implies t @ [e]\ setinterleaves_{\checkmark(\otimes \checkmark)} ((t', u'), A) \implies$ 
 $((t', \{\}) \in \mathcal{F}\ P \longrightarrow (u', \{\}) \notin \mathcal{F}\ Q) \wedge$ 
 $((u', \{\}) \in \mathcal{F}\ Q \longrightarrow (t', \{\}) \notin \mathcal{F}\ P) \rangle$  for  $e\ t'\ u'$ 
using  $\langle (t, X) \in ?f \wedge (\forall e. e \in Y \longrightarrow (t @ [e], \{\}) \notin ?f) \rangle$  by auto

define  $Y-ev-inside$  and  $Y-ev-notin$  and  $Y-tick$ 
where  $*$  :  $\langle Y-ev-inside \equiv \{a. ev\ a \in Y \wedge a \in A\} \rangle$ 
 $\langle Y-ev-notin \equiv \{a. ev\ a \in Y \wedge a \notin A\} \rangle$ 
 $\langle Y-tick \equiv \{r-s \mid r\ s\ r-s. r \otimes \checkmark\ s = \lfloor r-s \rfloor \wedge \checkmark(r-s) \in Y\} \rangle$ 

define  $Y-ev-inside-P$  and  $Y-ev-inside-Q$  and  $Y-ev-notin-P$ 
and  $Y-ev-notin-Q$  and  $Y-tick-P$  and  $Y-tick-Q$ 
where  $** : \langle Y-ev-inside-P \equiv \{a \in Y-ev-inside. (t-P @ [ev\ a], \{\}) \notin \mathcal{F}\ P\} \rangle$ 
 $\langle Y-ev-inside-Q \equiv \{a \in Y-ev-inside. (t-Q @ [ev\ a], \{\}) \notin \mathcal{F}\ Q\} \rangle$ 
 $\langle Y-ev-notin-P \equiv \{a \in Y-ev-notin. (t-P @ [ev\ a], \{\}) \notin \mathcal{F}\ P\} \rangle$ 
 $\langle Y-ev-notin-Q \equiv \{a \in Y-ev-notin. (t-Q @ [ev\ a], \{\}) \notin \mathcal{F}\ Q\} \rangle$ 
 $\langle Y-tick-P \equiv \{r-s \in Y-tick. \exists r\ s. r \otimes \checkmark\ s = \lfloor r-s \rfloor \wedge (t-P @ [\checkmark(r)], \{\}) \notin$ 
 $\mathcal{F}\ P\} \rangle$ 
 $\langle Y-tick-Q \equiv \{r-s \in Y-tick. \exists r\ s. r \otimes \checkmark\ s = \lfloor r-s \rfloor \wedge (t-Q @ [\checkmark(s)], \{\}) \notin$ 

```

$\notin \mathcal{F} Q\rangle$

have $\in : \langle \forall a \in Y\text{-ev-inside}. (t\text{-}P @ [ev\ a], \{\}) \notin \mathcal{F} P \vee (t\text{-}Q @ [ev\ a], \{\}) \notin \mathcal{F} Q \rangle$
proof (rule ccontr)
assume $\neg (\forall a \in Y\text{-ev-inside}. (t\text{-}P @ [ev\ a], \{\}) \notin \mathcal{F} P \vee (t\text{-}Q @ [ev\ a], \{\}) \notin \mathcal{F} Q) \rangle$
then obtain a where facts : $\langle a \in A \rangle \langle ev\ a \in Y \rangle \langle (t\text{-}P @ [ev\ a], \{\}) \in \mathcal{F} P \rangle$
 $\langle (t\text{-}Q @ [ev\ a], \{\}) \in \mathcal{F} Q \rangle$
unfolding * by blast
have $\langle t @ [ev\ a] \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P @ [ev\ a], t\text{-}Q @ [ev\ a]), A) \rangle$
by (simp add: facts(1) assms(3) setinterleaves_{ptick-snoc-inside})
with facts(2-4) assms5 show False by blast
qed
hence $\mathcal{L}\mathcal{L} : \langle Y\text{-ev-inside-}P \cup Y\text{-ev-inside-}Q = Y\text{-ev-inside} \rangle$ **by** (auto simp add: **)

have $\in\in : \langle \forall a \in Y\text{-ev-notin}. (t\text{-}P @ [ev\ a], \{\}) \notin \mathcal{F} P \vee (t\text{-}Q @ [ev\ a], \{\}) \notin \mathcal{F} Q \rangle$
proof (rule ccontr)
assume $\neg (\forall a \in Y\text{-ev-notin}. (t\text{-}P @ [ev\ a], \{\}) \notin \mathcal{F} P \vee (t\text{-}Q @ [ev\ a], \{\}) \notin \mathcal{F} Q) \rangle$
then obtain a where facts : $\langle a \notin A \rangle \langle ev\ a \in Y \rangle \langle (t\text{-}P @ [ev\ a], \{\}) \in \mathcal{F} P \rangle$
 $\langle (t\text{-}Q @ [ev\ a], \{\}) \in \mathcal{F} Q \rangle$ **unfolding * by blast**
have $\langle t @ [ev\ a] \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q @ [ev\ a]), A) \vee$
 $t @ [ev\ a] \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P @ [ev\ a], t\text{-}Q), A) \rangle$
by (simp add: facts(1) assms(3) setinterleaves_{ptick-snoc-notinL})
with facts assms(1-3) assms5 show False by (metis is-processT4-empty)
qed
hence $\mathcal{L}\mathcal{L}\mathcal{L} : \langle Y\text{-ev-notin-}P \cup Y\text{-ev-notin-}Q = Y\text{-ev-notin} \rangle$ **by** (auto simp add: **)

have $\in\in\in : \langle \forall r\text{-}s \in Y\text{-tick}. \exists r\ s. r \otimes \checkmark s = \lfloor r\text{-}s \rfloor \wedge ((t\text{-}P @ [\checkmark(r)], \{\}) \notin \mathcal{F} P \vee (t\text{-}Q @ [\checkmark(s)], \{\}) \notin \mathcal{F} Q) \rangle$
proof (rule ccontr)
assume $\neg (\forall r\text{-}s \in Y\text{-tick}. \exists r\ s. r \otimes \checkmark s = \lfloor r\text{-}s \rfloor \wedge ((t\text{-}P @ [\checkmark(r)], \{\}) \notin \mathcal{F} P \vee (t\text{-}Q @ [\checkmark(s)], \{\}) \notin \mathcal{F} Q)) \rangle$
then obtain r-s r s where facts : $\langle \checkmark(r\text{-}s) \in Y \rangle \langle r \otimes \checkmark s = \lfloor r\text{-}s \rfloor \rangle$
 $\langle (t\text{-}P @ [\checkmark(r)], \{\}) \in \mathcal{F} P \rangle \langle (t\text{-}Q @ [\checkmark(s)], \{\}) \in \mathcal{F} Q \rangle$
unfolding * by blast
have $\langle t @ [\checkmark(r\text{-}s)] \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P @ [\checkmark(r)], t\text{-}Q @ [\checkmark(s)]), A) \rangle$
by (simp add: facts(2) assms(3) setinterleaves_{ptick-snoc-tick})
with facts assms5 show False by blast
qed
hence $\mathcal{L}\mathcal{L}\mathcal{L}\mathcal{L} : \langle Y\text{-tick-}P \cup Y\text{-tick-}Q = Y\text{-tick} \rangle$ **unfolding ** by blast**

define $X\text{-}P'$ and $X\text{-}Q'$

```

where *** :  $\langle X-P' \equiv X-P \cup ev \text{ ' } Y\text{-ev-inside-}P \cup ev \text{ ' } Y\text{-ev-notin-}P \cup$ 
 $\{\checkmark(r) \mid r \text{ s } r\text{-s. } r \otimes \checkmark s = \lfloor r\text{-s} \rfloor \wedge r\text{-s} \in Y\text{-tick-}P\} \rangle$ 
 $\langle X-Q' \equiv X-Q \cup ev \text{ ' } Y\text{-ev-inside-}Q \cup ev \text{ ' } Y\text{-ev-notin-}Q \cup$ 
 $\{\checkmark(s) \mid r \text{ s } r\text{-s. } r \otimes \checkmark s = \lfloor r\text{-s} \rfloor \wedge r\text{-s} \in Y\text{-tick-}Q\} \rangle$ 

have $ :  $\langle (t-P, X-P') \in \mathcal{F} P \rangle \langle (t-Q, X-Q') \in \mathcal{F} Q \rangle$ 
by (auto simp add: ** *** intro!: is-processT5 assms dest: inj-tick-join)

have  $\langle Y \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A X-Q' \rangle$ 
proof (rule subsetI)
show  $\langle e \in \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A X-Q' \rangle$  if  $\langle e \in Y \rangle$  for  $e$ 
proof (cases  $e$ )
from  $\langle e \in Y \rangle$  show  $\langle e = ev \ a \implies e \in \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A$ 
 $X-Q' \rangle$  for  $a$ 
by (cases  $\langle a \in A \rangle$ , simp-all add: * ** *** image-iff super-ref-Syncptick-def)
(use *(1)  $\in$  in blast,
meson $(2) assms(1, 3) assms5 is-processT4-empty
setinterleavesptick-snoc-notinL setinterleavesptick-snoc-notinR)
next
show  $\langle e \in \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A X-Q' \rangle$  if  $\langle e = \checkmark(r-s) \rangle$  for  $r-s$ 
proof (cases  $\langle \exists r \text{ s. } r \otimes \checkmark s = \lfloor r\text{-s} \rfloor \rangle$ )
assume  $\langle \exists r \text{ s. } r \otimes \checkmark s = \lfloor r\text{-s} \rfloor \rangle$ 
then obtain  $r \text{ s}$  where  $\langle r \otimes \checkmark s = \lfloor r\text{-s} \rfloor \rangle$  by blast
with  $\langle e \in Y \rangle \langle e = \checkmark(r-s) \rangle$  have  $\langle r-s \in Y\text{-tick} \rangle$ 
by (auto simp add: *)
thus  $\langle e \in \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A X-Q' \rangle$ 
by (simp add: * super-ref-Syncptick-def)
(metis (mono-tags, lifting) *(3) ***(1,2)  $\mathcal{L}\mathcal{L}\mathcal{L}\mathcal{L}$ 
Un-iff mem-Collect-eq  $\langle e = \checkmark(r-s) \rangle$ )
next
show  $\langle \nexists r \text{ s. } r \otimes \checkmark s = \lfloor r\text{-s} \rfloor \implies$ 
 $e \in \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A X-Q' \rangle$ 
by (simp add:  $\langle e = \checkmark(r-s) \rangle$  super-ref-Syncptick-def)
qed
qed
qed
moreover from assms(4) have  $\langle X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P' A X-Q' \rangle$ 
by (fastforce simp add: *** subset-iff super-ref-Syncptick-def)
ultimately show  $\langle (t, X \cup Y) \in ?f \rangle$  using $ assms(3) by auto
qed
next
show processT9:  $\langle t \in ?d \rangle$  if  $\langle t @ [\checkmark(r-s)] \in ?d \rangle$  for  $t \text{ } r\text{-s}$ 
proof -
from  $\langle t @ [\checkmark(r-s)] \in ?d \rangle$  obtain  $u \text{ } v \text{ } t\text{-}P \text{ } t\text{-}Q$ 
where  $assms : \langle ftF \text{ } v \rangle \langle tF \text{ } u \vee v = [] \rangle$ 
 $\langle t @ [\checkmark(r-s)] = u @ v \rangle$ 
 $\langle u \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t\text{-}P, t\text{-}Q), A) \rangle$ 
 $\langle t\text{-}P \in \mathcal{D} P \wedge t\text{-}Q \in \mathcal{T} Q \vee t\text{-}P \in \mathcal{T} P \wedge t\text{-}Q \in \mathcal{D} Q \rangle$  by blast
from assms(2) show  $\langle t \in ?d \rangle$ 

```

```

proof (elim disjE)
  assume  $\langle tF\ u \rangle$ 
  with assms(3) obtain  $v'$  where  $\langle v = v' @ [\checkmark(r-s)] \rangle \langle t = u @ v' \rangle$ 
    by (cases v rule: rev-cases) auto
  from  $\langle v = v' @ [\checkmark(r-s)] \rangle$  assms(1) front-tickFree-dw-closed
  have  $\langle ftF\ v' \rangle$  by blast
  with  $\langle t = u @ v' \rangle \langle tF\ u \rangle$  assms(1, 4, 5) show  $\langle t \in ?d \rangle$  by blast
next
  assume  $\langle v = [] \rangle$ 
  with assms(3) obtain  $u'$  where  $\langle u = u' @ [\checkmark(r-s)] \rangle \langle t = u' \rangle$  by auto
  from snoc-tick-setinterleavesptickE[OF assms(4)[unfolded this(1)]]
  obtain  $r\ s\ t-P'\ t-Q'$  where  $\langle r \otimes \checkmark\ s = \lfloor r-s \rfloor \rangle$ 
     $\langle u' \text{ setinterleaves } \checkmark(\otimes \checkmark) ((t-P', t-Q'), A) \rangle$ 
     $\langle t-P = t-P' @ [\checkmark(r)] \rangle \langle t-Q = t-Q' @ [\checkmark(s)] \rangle$  by metis
  with assms(5)  $\langle t = u' \rangle$  show  $\langle t \in ?d \rangle$ 
    by simp (metis append.right-neutral front-tickFree-Nil
      is-processT3-TR-append is-processT9)
qed
qed

fix  $t\ X\ r-s$ 
assume  $\langle t @ [\checkmark(r-s)], \{\} \rangle \in ?f$ 
then consider (div)  $\langle t @ [\checkmark(r-s)] \rangle \in ?d$ 
  | (fail)  $t-P\ t-Q\ X-P\ X-Q$ 
    where  $\langle t-P, X-P \rangle \in \mathcal{F}\ P \rangle \langle t-Q, X-Q \rangle \in \mathcal{F}\ Q \rangle$ 
     $\langle t @ [\checkmark(r-s)] \rangle \text{ setinterleaves } \checkmark(\otimes \checkmark) ((t-P, t-Q), A) \rangle$  by auto
thus  $\langle t, X - \{\checkmark(r-s)\} \rangle \in ?f$ 
proof cases
  show  $\langle t @ [\checkmark(r-s)] \rangle \in ?d \implies \langle t, X - \{\checkmark(r-s)\} \rangle \in ?f$  by (drule processT9)
simp
next
  case fail
  from fail(3)[THEN snoc-tick-setinterleavesptickE]
  obtain  $r\ s\ t-P'\ t-Q'$  where  $\ast : \langle r \otimes \checkmark\ s = \lfloor r-s \rfloor \rangle$ 
     $\langle t \text{ setinterleaves } \checkmark(\otimes \checkmark) ((t-P', t-Q'), A) \rangle$ 
     $\langle t-P = t-P' @ [\checkmark(r)] \rangle \langle t-Q = t-Q' @ [\checkmark(s)] \rangle$  by metis
  from fail(1, 2) have  $\langle t-P' @ [\checkmark(r)] \rangle \in \mathcal{T}\ P \rangle \langle t-Q' @ [\checkmark(s)] \rangle \in \mathcal{T}\ Q \rangle$ 
    by (simp-all add: *(3, 4) F-T)
  hence  $\langle t-P', UNIV - \{\checkmark(r)\} \rangle \in \mathcal{F}\ P \rangle$ 
     $\langle t-Q', UNIV - \{\checkmark(s)\} \rangle \in \mathcal{F}\ Q \rangle$  by (meson is-processT6-TR) +
  moreover have  $\langle X - \{\checkmark(r-s)\} \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) (UNIV - \{\checkmark(r)\})$ 
   $A\ (UNIV - \{\checkmark(s)\}) \rangle$ 
    by (simp add: subset-iff super-ref-Syncptick-def)
    (metis *(1) eventptick.exhaust option.inject)
  ultimately show  $\langle t, X - \{\checkmark(r-s)\} \rangle \in ?f$  using *(2) by fast
qed
next
show  $\langle s \in ?d \wedge tF\ s \wedge ftF\ t \implies s @ t \in ?d \rangle$  for  $s\ t$ 
  using front-tickFree-append by fastforce

```

next
show $\langle s \in ?d \implies (s, X) \in ?f \rangle$ **for** $s \ X$ **by** *blast*
qed
qed

Here $X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P \ A \ X-Q$ may seem surprising (instead of for example $X = \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P \ A \ X-Q$, 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}* ::
 $\langle [(a, 'r) \text{ process}_{ptick}, (a, 's) \text{ process}_{ptick}] \Rightarrow$
 $(a, 't) \text{ process}_{ptick} \rangle \langle (- \parallel_{\checkmark} -) \rangle [72, 73] \ 72)$
where $\langle P \parallel_{\checkmark} Q \equiv P \llbracket \{\} \rrbracket_{\checkmark} Q \rangle$

abbreviation (in *Sync_{ptick}-locale*) *Par_{ptick}* ::
 $\langle [(a, 'r) \text{ process}_{ptick}, (a, 's) \text{ process}_{ptick}] \Rightarrow$
 $(a, 't) \text{ process}_{ptick} \rangle \langle (- \parallel_{\checkmark} -) \rangle [74, 75] \ 74)$
where $\langle P \parallel_{\checkmark} Q \equiv P \llbracket UNIV \rrbracket_{\checkmark} Q \rangle$

notation (in *Sync_{ptick}-locale*) *Sync_{ptick}-locale-sym.Sync_{ptick}*
 $\langle (- \llbracket_{\checkmark} \rrbracket_{sym} -) \rangle [70, 0, 71] \ 70)$

notation (in *Sync_{ptick}-locale*) *Sync_{ptick}-locale-sym.Inter_{ptick}*
 $\langle (- \parallel_{\checkmark} \rrbracket_{sym} -) \rangle [72, 73] \ 72)$

notation (in *Sync_{ptick}-locale*) *Sync_{ptick}-locale-sym.Par_{ptick}*
 $\langle (- \parallel_{\checkmark} \rrbracket_{sym} -) \rangle [74, 75] \ 74)$

4.2.2 Projections

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

lemma *D-Sync_{ptick}'* :
 $\langle \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) =$
 $\{ t @ u \mid t \ u \ t-P \ t-Q.$
 $\text{ftF } u \wedge (tF \ t \vee u = []) \wedge t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t-P, t-Q), A) \wedge$
 $(t-P \in \mathcal{D} \ P \wedge t-Q \in \mathcal{T} \ Q \vee t-P \in \mathcal{T} \ P \wedge t-Q \in \mathcal{D} \ Q) \} \rangle$
by (*simp add: Divergences.rep-eq Sync_{ptick}.rep-eq DIVERGENCES-def*)

corollary *D-Sync_{ptick}* :

— This version is easier to use.

$\langle \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) =$
 $\{ t @ u \mid t \ u \ t-P \ t-Q.$
 $tF \ t \wedge \text{ftF } u \wedge t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t-P, t-Q), A) \wedge$
 $(t-P \in \mathcal{D} \ P \wedge t-Q \in \mathcal{T} \ Q \vee t-P \in \mathcal{T} \ P \wedge t-Q \in \mathcal{D} \ Q) \} \rangle$
(is $\langle - = ?rhs \rangle$)

proof (*intro subset-antisym subsetI*)

show $\langle d \in ?rhs \implies d \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$ **for** d
by (*auto simp add: D-Sync_{ptick}'*)
next
fix d **assume** $\langle d \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
then obtain $t \ u \ t\text{-}P \ t\text{-}Q$
where $*$: $\langle d = t @ u \rangle \langle ftF \ u \rangle \langle tF \ t \vee u = [] \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), A) \rangle$
 $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$
unfolding $D\text{-Sync}_{ptick}'$ **by** *blast*
show $\langle d \in ?rhs \rangle$
proof (*cases* $\langle tF \ t \rangle$)
from $*$ **show** $\langle tF \ t \implies d \in ?rhs \rangle$ **by** *blast*
next
assume $\langle \neg tF \ t \rangle$
with $*(1, 3)$ **have** $\langle u = [] \rangle \langle d = t \rangle$ **by** *simp-all*
from $D\text{-imp-front-tickFree}$ $\langle d = t \rangle \langle d \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
have $\langle ftF \ t \rangle$ **by** *blast*
with $\langle \neg tF \ t \rangle$ **obtain** $r\text{-}s \ t'$ **where** $\langle t = t' @ [\checkmark(r\text{-}s)] \rangle$
by (*meson nonTickFree-n-frontTickFree*)
with $*(4)$ **obtain** $r \ t\text{-}P' \ s \ t\text{-}Q'$
where $**$: $\langle r \otimes \checkmark \ s = \lfloor r\text{-}s \rfloor \rangle$
 $\langle t\text{-}P = t\text{-}P' @ [\checkmark(r)] \rangle \langle t\text{-}Q = t\text{-}Q' @ [\checkmark(s)] \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P', t\text{-}Q'), A) \rangle$
by (*auto simp add:* $\langle t = t' @ [\checkmark(r\text{-}s)] \rangle$
elim: snoc-tick-setinterleaves_{ptick}E)
have $\langle t\text{-}P' \in \mathcal{D} \ P \wedge t\text{-}Q' \in \mathcal{T} \ Q \vee t\text{-}P' \in \mathcal{T} \ P \wedge t\text{-}Q' \in \mathcal{D} \ Q \rangle$
by (*metis* $*(5) \ ***(2, 3)$ *is-processT3-TR-append is-processT9*)
with $*(4)$ $\langle d = t \rangle \langle ftF \ t \rangle \langle t = t' @ [\checkmark(r\text{-}s)] \rangle$
front-tickFree-nonempty-append-imp **show** $\langle d \in ?rhs \rangle$ **by** *blast*
qed
qed

lemma $F\text{-Sync}_{ptick}' :$

$\langle \mathcal{F} (P \llbracket A \rrbracket_{\checkmark} Q) =$
 $\{(t, X). \exists t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q.$
 $(t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \wedge$
 $t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), A) \wedge$
 $X \subseteq \text{super-ref-Sync}_{ptick} (\otimes\checkmark) X\text{-}P \ A \ X\text{-}Q\} \cup$
 $\{(t @ u, X) \mid t \ u \ t\text{-}P \ t\text{-}Q \ X.$
 $ftF \ u \wedge (tF \ t \vee u = []) \wedge t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), A) \wedge$
 $(t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q)\}\}$
by (*simp add: Failures.rep-eq Sync_{ptick}.rep-eq FAILURES-def*)

lemma $F\text{-Sync}_{ptick} :$

$\langle \mathcal{F} (P \llbracket A \rrbracket_{\checkmark} Q) =$
 $\{(t, X). \exists t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q.$
 $(t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \wedge$
 $t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), A) \wedge$


```

next
  show  $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \implies t \in \mathcal{D} (tjoin\text{-}interpreted.Sync_{ptick} P S Q) \rangle$  for  $t$ 
  by (simp add:  $D\text{-}Sync_{ptick} \text{ } tjoin\text{-}interpreted.D\text{-}Sync_{ptick}$ )
    (metis tickFree-setinterleaves $_{ptick}$ -any-tick-join)
next
  fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} (tjoin\text{-}interpreted.Sync_{ptick} P S Q) \rangle$ 
   $\langle t \notin \mathcal{D} (tjoin\text{-}interpreted.Sync_{ptick} P S Q) \rangle$ 
  then obtain  $t\text{-}P \ X\text{-}P \ t\text{-}Q \ X\text{-}Q$  where  $*$  :  $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} Q \rangle$ 
   $\langle t \text{ setinterleaves}_{\checkmark} \text{ tick-join}' ((t\text{-}P, t\text{-}Q), S) \rangle$ 
   $\langle X \subseteq \text{super-ref-}Sync_{ptick} \text{ tick-join}' X\text{-}P S X\text{-}Q \rangle$ 
  unfolding  $tjoin\text{-}interpreted.Sync_{ptick}\text{-}projs$  by blast
  define  $X\text{-}P\text{-}plus$  where  $\langle X\text{-}P\text{-}plus \equiv X\text{-}P \cup \{\checkmark(r) \mid r. t\text{-}P @ [\checkmark(r)] \notin \mathcal{T} P - \mathcal{D} P \} \rangle$ 
  define  $X\text{-}Q\text{-}plus$  where  $\langle X\text{-}Q\text{-}plus \equiv X\text{-}Q \cup \{\checkmark(s) \mid s. t\text{-}Q @ [\checkmark(s)] \notin \mathcal{T} Q - \mathcal{D} Q \} \rangle$ 
  have  $\langle t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle$ 
  proof (cases  $\langle tF t \rangle$ )
    show  $\langle tF t \implies t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle$ 
    using  $*(3)$  tickFree-setinterleaves $_{ptick}$ -any-tick-join by blast
  next
    assume  $\langle \neg tF t \rangle$ 
    then obtain  $t' r\text{-}s$  where  $\langle tF t' \rangle \langle t = t' @ [\checkmark(r-s)] \rangle$ 
    by (metis  $F\text{-}imp\text{-}front\text{-}tickFree \langle (t, X) \in \mathcal{F} (tjoin\text{-}interpreted.Sync_{ptick} P S Q) \rangle$ )
    front-tickFree-append-iff nonTickFree-n-frontTickFree not-Cons-self2)
    with  $*(3)$  obtain  $t\text{-}P' r \ t\text{-}Q' s$  where  $**$  :  $\langle tick\text{-}join' r s = \lfloor r-s \rfloor \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark} \text{ tick-join}' ((t\text{-}P', t\text{-}Q'), S) \rangle$ 
     $\langle t\text{-}P = t\text{-}P' @ [\checkmark(r)] \rangle \langle t\text{-}Q = t\text{-}Q' @ [\checkmark(s)] \rangle$ 
    by (auto elim: snoc-tick-setinterleaves $_{ptick} E$ )
    have  $\langle r \in \checkmark s(P) \wedge s \in \checkmark s(Q) \rangle$ 
    proof (rule ccontr)
      assume  $\langle \neg (r \in \checkmark s(P) \wedge s \in \checkmark s(Q)) \rangle$ 
      hence  $\langle t\text{-}P' @ [\checkmark(r)] \in \mathcal{D} P \vee t\text{-}Q' @ [\checkmark(s)] \in \mathcal{D} Q \rangle$ 
      by (metis  $*(1, 2) ** (3, 4) F\text{-}T \text{ strict-ticks-of-memI}$ )
      with  $\langle t \notin \mathcal{D} (tjoin\text{-}interpreted.Sync_{ptick} P S Q) \rangle$  show False
      by (simp add:  $tjoin\text{-}interpreted.D\text{-}Sync_{ptick}$ )
      (metis  $*(1-3) ** (3, 4) F\text{-}T \text{ append.right-neutral front-tickFree-Nil}$ )
    qed
    moreover from  $*(2)$  have  $\langle t' \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t\text{-}P', t\text{-}Q'), S) \rangle$ 
    using  $\langle tF t' \rangle$  tickFree-setinterleaves $_{ptick}$ -any-tick-join by blast
    ultimately show  $\langle t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle$ 
    by (subst rev-setinterleaves $_{ptick}$ -rev-rev-iff[symmetric],
      subst (asm) rev-setinterleaves $_{ptick}$ -rev-rev-iff[symmetric])
      (use  $*(1)$  that(2) in  $\langle auto \text{ simp add: } \langle t = t' @ [\checkmark(r-s)] \rangle ** (3, 4) \rangle$ )
    qed
  moreover from  $*(1)$  is-processT5-S7' is-processT8 is-processT9
  have  $\langle (t\text{-}P, X\text{-}P\text{-}plus) \in \mathcal{F} P \rangle$  by (fastforce simp add:  $X\text{-}P\text{-}plus\text{-}def$ )

```

moreover from $\ast(2)$ *is-process* $T5-S7'$ *is-process* $T8$ *is-process* $T9$
 have $\langle (t-Q, X-Q\text{-plus}) \in \mathcal{F} \ Q \rangle$ by (fastforce simp add: $X-Q\text{-plus-def}$)
 moreover have $\langle e \in X \implies e \in \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P\text{-plus } S \ X-Q\text{-plus} \rangle$
 for e
 using $\ast(4)$ [THEN set-mp, of e]
 by (cases e , simp-all add: $X-P\text{-plus-def}$ $X-Q\text{-plus-def}$ $\text{super-ref-Sync}_{ptick}\text{-def}$ subset-iff)
 (metis strict-ticks-of-memI that(2) tjoin-interpreted.inj-tick-join)
 ultimately show $\langle (t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ by (simp add: $F\text{-Sync}_{ptick}$) blast
 next
 fix $t \ X$ assume $\langle (t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle \langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 then obtain $t-P \ X-P \ t-Q \ X-Q$ where $\ast : \langle (t-P, X-P) \in \mathcal{F} \ P \rangle \langle (t-Q, X-Q) \in \mathcal{F} \ Q \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P \ S \ X-Q \rangle$
 unfolding $\text{Sync}_{ptick}\text{-projs}$ by blast
 define $X-P\text{-plus}$ where $\langle X-P\text{-plus} \equiv X-P \cup \{\checkmark(r) \mid r. t-P @ [\checkmark(r)] \notin \mathcal{T} \ P - \mathcal{D} \ P \} \rangle$
 define $X-Q\text{-plus}$ where $\langle X-Q\text{-plus} \equiv X-Q \cup \{\checkmark(s) \mid s. t-Q @ [\checkmark(s)] \notin \mathcal{T} \ Q - \mathcal{D} \ Q \} \rangle$
 have $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}'} ((t-P, t-Q), S) \rangle$
 proof (cases $\langle tF \ t \rangle$)
 show $\langle tF \ t \implies t \text{ setinterleaves}_{\checkmark \text{ tick-join}'} ((t-P, t-Q), S) \rangle$
 using $\ast(3)$ tickFree-setinterleaves_{ptick}-any-tick-join by blast
 next
 assume $\langle \neg tF \ t \rangle$
 then obtain $t' \ r-s$ where $\langle tF \ t' \rangle \langle t = t' @ [\checkmark(r-s)] \rangle$
 by (metis $F\text{-imp-front-tickFree}$ $\langle (t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ $\text{front-tickFree-append-iff}$ $\text{nonTickFree-n-frontTickFree}$ not-Cons-self2)
 with $\ast(3)$ obtain $t-P' \ r \ t-Q' \ s$ where $\ast\ast : \langle r \otimes \checkmark \ s = \lfloor r-s \rfloor \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P', t-Q'), S) \rangle$
 $\langle t-P = t-P' @ [\checkmark(r)] \rangle \langle t-Q = t-Q' @ [\checkmark(s)] \rangle$
 by (auto elim: $\text{snoc-tick-setinterleaves}_{ptick} E$)
 have $\langle r \in \checkmark s(P) \wedge s \in \checkmark s(Q) \rangle$
 proof (rule ccontr)
 assume $\langle \neg (r \in \checkmark s(P) \wedge s \in \checkmark s(Q)) \rangle$
 hence $\langle t-P' @ [\checkmark(r)] \in \mathcal{D} \ P \vee t-Q' @ [\checkmark(s)] \in \mathcal{D} \ Q \rangle$
 by (metis $\ast(1, 2)$ $\ast\ast(3, 4)$ $F-T$ $\text{strict-ticks-of-memI}$)
 with $\langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ show False
 by (simp add: $D\text{-Sync}_{ptick}$)
 (metis $\ast(1-3)$ $\ast\ast(3, 4)$ $F-T$ $\text{append.right-neutral}$ $\text{front-tickFree-Nil}$)
 qed
 moreover from $\ast(2)$ have $\langle t' \text{ setinterleaves}_{\checkmark \text{ tick-join}'} ((t-P', t-Q'), S) \rangle$
 using $\langle tF \ t' \rangle$ tickFree-setinterleaves_{ptick}-any-tick-join by blast
 ultimately show $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}'} ((t-P, t-Q), S) \rangle$
 by (subst rev-setinterleaves_{ptick}-rev-rev-iff[symmetric],
 subst (asm) rev-setinterleaves_{ptick}-rev-rev-iff[symmetric])
 (use $\ast(1)$ that(2) in $\langle \text{auto simp add: } \langle t = t' @ [\checkmark(r-s)] \rangle \ast\ast(3, 4) \rangle$)

qed
moreover from $\ast(1)$ *is-process* $T5$ - $S7'$ *is-process* $T8$ *is-process* $T9$
have $\langle (t-P, X-P-plus) \in \mathcal{F} P \rangle$ **by** (*fastforce simp add: X-P-plus-def*)
moreover from $\ast(2)$ *is-process* $T5$ - $S7'$ *is-process* $T8$ *is-process* $T9$
have $\langle (t-Q, X-Q-plus) \in \mathcal{F} Q \rangle$ **by** (*fastforce simp add: X-Q-plus-def*)
moreover have $\langle e \in X \implies e \in \text{super-ref-Sync}_{ptick} \text{ tick-join}' X-P-plus S$
 $X-Q-plus \rangle$ **for** e
using $\ast(4)$ [*THEN set-mp, of e*]
by (*cases e, simp-all add: X-P-plus-def X-Q-plus-def super-ref-Sync_{ptick}-def*
subset-iff)
(metis strict-ticks-of-memI that(2) inj-tick-join)
ultimately show $\langle (t, X) \in \mathcal{F} (tjoin-interpreted.Sync_{ptick} P S Q) \rangle$
by (*simp add: tjoin-interpreted.F-Sync_{ptick}*) *blast*
qed
qed

4.2.3 First Properties

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

lemma *setinterleaves_{ptick}-imp-set-range-tick-join* :
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u, v), A) \implies$
 $\{r-s. \checkmark(r-s) \in \text{set } t\} \subseteq \text{range-tick-join} \rangle$
by (*induct* $\langle ((\otimes\checkmark), u, A, v) \rangle$ *arbitrary: t u v*)
(auto simp add: subset-iff split: if-split-asm option.split-asm)+
end

lemma
— Of course not suitable for simplifier.
 $\langle t \text{ setinterleaves}_{\checkmark\lambda s \ r. \text{ tick-join } r \ s} ((v, u), A) \longleftrightarrow$
 $t \text{ setinterleaves}_{\checkmark\lambda r \ s. \text{ tick-join } r \ s} ((u, v), A) \rangle$
by (*fact setinterleaves_{ptick}-sym*)

lemma *super-ref-Sync_{ptick}-sym* :
— Of course not suitable for simplifier.
 $\langle \text{super-ref-Sync}_{ptick} (\lambda s \ r. \text{ tick-join } r \ s) X-Q \ S \ X-P =$
 $\text{super-ref-Sync}_{ptick} (\lambda r \ s. \text{ tick-join } r \ s) X-P \ S \ X-Q \rangle$
by (*auto simp add: super-ref-Sync_{ptick}-def*)

lemma *super-ref-Sync_{ptick}-mono* :
 $\langle A \subseteq A' \implies X-P \subseteq X-P' \implies X-Q \subseteq X-Q' \implies$
 $\text{super-ref-Sync}_{ptick} \text{ tick-join } X-P \ A \ X-Q \subseteq$
 $\text{super-ref-Sync}_{ptick} \text{ tick-join } X-P' \ A' \ X-Q' \rangle$
by (*auto simp add: super-ref-Sync_{ptick}-def*)

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

lemma $\text{Sync}_{\text{ptick}}\text{-sym} : \langle Q \llbracket A \rrbracket_{\text{sym}} P = P \llbracket A \rrbracket Q \rangle$

proof (rule *Process-eq-optimizedI*)

show $\langle t \in \mathcal{D} (Q \llbracket A \rrbracket_{\text{sym}} P) \implies t \in \mathcal{D} (P \llbracket A \rrbracket Q) \rangle$ **for** t
by (*simp add: Sync_{ptick}-locale-sym.D-Sync_{ptick} D-Sync_{ptick}*)
(subst setinterleaves_{ptick}-sym, blast)

next

show $\langle t \in \mathcal{D} (P \llbracket A \rrbracket Q) \implies t \in \mathcal{D} (Q \llbracket A \rrbracket_{\text{sym}} P) \rangle$ **for** t
by (*simp add: Sync_{ptick}-locale-sym.D-Sync_{ptick} D-Sync_{ptick}*)
(subst setinterleaves_{ptick}-sym, blast)

next

show $\langle (t, X) \in \mathcal{F} (Q \llbracket A \rrbracket_{\text{sym}} P) \implies (t, X) \in \mathcal{F} (P \llbracket A \rrbracket Q) \rangle$ **for** $t X$
by (*simp add: Sync_{ptick}-locale-sym.F-Sync_{ptick} F-Sync_{ptick}*)
(subst (1 2) setinterleaves_{ptick}-sym,
subst super-ref-Sync_{ptick}-sym, blast)

next

show $\langle (t, X) \in \mathcal{F} (P \llbracket A \rrbracket Q) \implies (t, X) \in \mathcal{F} (Q \llbracket A \rrbracket_{\text{sym}} P) \rangle$ **for** $t X$
by (*simp add: Sync_{ptick}-locale-sym.F-Sync_{ptick} F-Sync_{ptick}*)
(subst (1 2) setinterleaves_{ptick}-sym,
subst super-ref-Sync_{ptick}-sym, blast)

qed

lemma *interpretable-inj-on-range-tick-join* :

$\langle \text{inj-on } g \text{ range-tick-join} \implies$
 $\text{Sync}_{\text{ptick}}\text{-locale } (\lambda r s. \text{ case } r \otimes \checkmark s \text{ of } [r-s] \Rightarrow [g \ r-s] \mid \Diamond \Rightarrow \Diamond) \rangle$
by (*unfold-locales, simp split: option.split-asm*)
(metis (mono-tags, lifting) inj-onD inj-tick-join mem-Collect-eq)

lemma *inj-on-map-map-event_{ptick}-setinterleaves_{ptick}* :

$\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u, v), A) \implies$
 $\text{map } (\text{map-event}_{\text{ptick}} \text{ id } g) t$
 $\text{setinterleaves}_{\checkmark\lambda r s. \text{ case } r \otimes \checkmark s \text{ of } [r-s] \Rightarrow [g \ r-s] \mid \Diamond \Rightarrow \Diamond} ((u, v), A) \rangle$
(is $\langle - \implies - \text{ setinterleaves}_{\checkmark?tick\text{-join}'} ((u, v), A) \rangle$
if *inj-on-g* : $\langle \text{inj-on } g \text{ range-tick-join} \rangle$

proof (*induct* $\langle ((\otimes\checkmark), u, A, v) \rangle$ *arbitrary: t u v*)

case (*tick-setinterleaving_{ptick}-tick* $r \ u \ s \ v$)
from *tick-setinterleaving_{ptick}-tick.prem*s [*simplified*]
obtain $r-s \ t'$ **where** $*$: $\langle r \otimes \checkmark s = [r-s] \rangle \langle t = \checkmark(r-s) \# t' \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u, v), A) \rangle$
by (*auto split: option.split-asm*)
from *tick-setinterleaving_{ptick}-tick.hyps*[*OF* $*(1, 3)$]
have $\langle \text{map } (\text{map-event}_{\text{ptick}} \text{ id } g) t'$
 $\text{setinterleaves}_{\checkmark?tick\text{-join}'} ((u, v), A) \rangle$.

thus ?case by (simp add: *(1, 2))
qed auto

lemma *image-inj-on-subset-super-ref-Sync_{ptick}-iff* :
 $\langle \text{map-event}_{\text{ptick}} \text{id } g - ' X \subseteq$
 $\text{super-ref-Sync}_{\text{ptick}} (\otimes \checkmark) X-P A X-Q \longleftrightarrow$
 $X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\lambda r s. \text{case } r \otimes \checkmark s \text{ of } [r-s] \Rightarrow [g r-s] \mid \Diamond \Rightarrow \Diamond) X-P A$
 $X-Q \rangle$
 (is $\langle ?lhs1 \subseteq ?lhs2 \longleftrightarrow X \subseteq ?rhs \rangle$)
 if *inj-on-g* : $\langle \text{inj-on } g \text{ range-tick-join} \rangle$
proof –
 let *?tick-join'* = $\langle \lambda r s. \text{case } r \otimes \checkmark s \text{ of } [r-s] \Rightarrow [g r-s] \mid \Diamond \Rightarrow \Diamond \rangle$
 interpret *Sync_{ptick}'* : *Sync_{ptick}-locale* ?tick-join'
 by (intro interpretable-inj-on-range-tick-join inj-on-g)
 from *inv-into-f-f inj-on-g* have *expanded-tick-join* :
 $\langle \text{tick-join} =$
 $(\lambda r s. \text{case } ?tick\text{-join}' r s \text{ of } \Diamond \Rightarrow \Diamond \mid [r-s] \Rightarrow [\text{inv-into range-tick-join } g r-s]) \rangle$
 by (fastforce split: split: option.split)
 let ?f1 = $\langle \text{map-event}_{\text{ptick}} \text{id } g \rangle$
 let ?f2 = $\langle \text{map-event}_{\text{ptick}} \text{id } (\text{inv-into range-tick-join } g) \rangle$
 show $\langle ?lhs1 \subseteq ?lhs2 \longleftrightarrow X \subseteq ?rhs \rangle$
proof (intro iffI subsetI)
 show $\langle e \in ?rhs \rangle$ if $\langle ?lhs1 \subseteq ?lhs2 \rangle \langle e \in X \rangle$ for *e*
proof (cases e)
 fix *a* assume $\langle e = \text{ev } a \rangle$
 with $\langle e \in X \rangle$ have $\langle ?f2 e \in ?f1 - ' X \rangle$ by simp
 with $\langle ?lhs1 \subseteq ?lhs2 \rangle$ have $\langle ?f2 e \in ?lhs2 \rangle$ by blast
 with $\langle e = \text{ev } a \rangle$ show $\langle e \in ?rhs \rangle$
 by (auto simp add: super-ref-Sync_{ptick}-def)
 next
 show $\langle e \in ?rhs \rangle$ if $\langle e = \checkmark(r-s) \rangle$ for *r-s*
proof (cases $\langle \exists r s. ?tick\text{-join}' r s = [r-s] \rangle$)
 from $\langle e = \checkmark(r-s) \rangle$ show $\langle \nexists r s. ?tick\text{-join}' r s = [r-s] \implies e \in ?rhs \rangle$
 by (simp add: super-ref-Sync_{ptick}-def)
 next
 assume $\langle \exists r s. ?tick\text{-join}' r s = [r-s] \rangle$
 with $\langle e = \checkmark(r-s) \rangle \langle e \in X \rangle$
 have $\langle ?f2 e \in ?f1 - ' X \rangle$
 by (auto split: option.split-asm)
 (metis (no-types, lifting) expanded-tick-join option.simps(5))
 with $\langle ?lhs1 \subseteq ?lhs2 \rangle$ have $\langle ?f2 e \in ?lhs2 \rangle$ by blast
 with $\langle e = \checkmark(r-s) \rangle$ show $\langle e \in ?rhs \rangle$
 by (simp add: super-ref-Sync_{ptick}-def)
 (metis (no-types, lifting) expanded-tick-join option.simps(5))
 qed
 qed
 next

```

show  $\langle e \in ?lhs2 \rangle$  if  $\langle X \subseteq ?rhs \rangle$  and  $\langle e \in ?lhs1 \rangle$  for  $e$ 
proof (cases  $e$ )
  fix  $a$  assume  $\langle e = ev\ a \rangle$ 
  with  $\langle e \in ?lhs1 \rangle$  have  $\langle ev\ a \in X \rangle$  by simp
  with  $\langle X \subseteq ?rhs \rangle$  have  $\langle ev\ a \in ?rhs \rangle$  by blast
  thus  $\langle e \in ?lhs2 \rangle$  by (auto simp add:  $\langle e = ev\ a \rangle$  super-ref-Syncptick-def)
next
show  $\langle e \in ?lhs2 \rangle$  if  $\langle e = \checkmark(s-r) \rangle$  for  $s-r$ 
proof (cases  $\langle \exists s\ r. tick\ join\ s\ r = \lfloor s-r \rfloor \rangle$ )
  from  $\langle e = \checkmark(s-r) \rangle$  show  $\langle \nexists s\ r. tick\ join\ s\ r = \lfloor s-r \rfloor \implies e \in ?lhs2 \rangle$ 
  by (simp add: super-ref-Syncptick-def)
next
assume  $\langle \exists s\ r. tick\ join\ s\ r = \lfloor s-r \rfloor \rangle$ 
with  $\langle e = \checkmark(s-r) \rangle$   $\langle e \in ?lhs1 \rangle$ 
have  $\langle \checkmark(g\ s-r) \in X \rangle$  by simp
with  $\langle X \subseteq ?rhs \rangle$  have  $\langle \checkmark(g\ s-r) \in ?rhs \rangle$  by blast
with  $\langle e = \checkmark(s-r) \rangle$  show  $\langle e \in ?lhs2 \rangle$ 
  by (simp add: super-ref-Syncptick-def)
  (metis Syncptick'.inj-tick-join option.simps(5))
qed
qed
qed
qed

```

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

```

lemma finite-setinterleavesptick-tick-join :
   $\langle finite\ \{(u, v). t\ setinterleaves_{\checkmark(\otimes\checkmark)}((u, v), A)\} \rangle$ 
  (is  $\langle finite\ \{(u, v). ?f\ t\ u\ v\} \rangle$ )
proof (induct  $t$ )
  have  $\langle \{(u, v). ?f\ []\ u\ v\} = \{([], [])\} \rangle$  by (auto simp add: Nil-setinterleavesptick)
  thus  $\langle finite\ \{(u, v). ?f\ []\ u\ v\} \rangle$  by simp
next
fix  $e\ t$  assume  $\langle finite\ \{(u, v). ?f\ t\ u\ v\} \rangle$ 
have  $*$  :  $\langle \{(x \# u, v) \mid u\ v. ?f\ t\ u\ v\} = (\lambda(u, v). (x \# u, v))\ ' \{(u, v). ?f\ t\ u\ v\} \rangle$ 
   $\langle \{(u, y \# v) \mid u\ v. ?f\ t\ u\ v\} = (\lambda(u, v). (u, y \# v))\ ' \{(u, v). ?f\ t\ u\ v\} \rangle$ 
   $\langle \{(x \# u, y \# v) \mid u\ v. ?f\ t\ u\ v\} = (\lambda(u, v). (x \# u, y \# v))\ ' \{(u, v). ?f\ t\ u\ v\} \rangle$ 
  by (auto)
for  $x\ y$  by auto
show  $\langle finite\ \{(u, v). ?f\ (e \# t)\ u\ v\} \rangle$ 
proof (cases  $e$ )
  fix  $a$  assume  $\langle e = ev\ a \rangle$ 
  hence  $\langle ?f\ (e \# t)\ u\ v \implies$ 
     $u \neq [] \wedge hd\ u = ev\ a \wedge ?f\ t\ (tl\ u)\ v \vee$ 
     $v \neq [] \wedge hd\ v = ev\ a \wedge ?f\ t\ u\ (tl\ v) \vee$ 
     $u \neq [] \wedge hd\ u = ev\ a \wedge v \neq [] \wedge hd\ v = ev\ a \wedge ?f\ t\ (tl\ u)\ (tl\ v) \rangle$  for  $u\ v$ 
  by (cases  $e$ ) (auto elim: Cons-ev-setinterleavesptick E Cons-tick-setinterleavesptick E)
  hence  $\langle \{(u, v). ?f\ (e \# t)\ u\ v\} \subseteq$ 
     $\{(ev\ a \# u, v) \mid u\ v. ?f\ t\ u\ v\} \cup$ 
     $\{(u, ev\ a \# v) \mid u\ v. ?f\ t\ u\ v\} \cup$ 
     $\{(u, ev\ a \# v) \mid u\ v. ?f\ t\ u\ v\} \cup$ 

```

$\{(ev\ a\ \# \ u, \ ev\ a\ \# \ v) \mid u\ v.\ ?f\ t\ u\ v\}$
 by (simp add: subset-iff) (metis list.collapse)
 moreover have $\langle finite\ \{(ev\ a\ \# \ u, \ v) \mid u\ v.\ ?f\ t\ u\ v\} \rangle$
 by (simp add: *(1) $\langle finite\ \{(u, \ v).\ ?f\ t\ u\ v\} \rangle$)
 moreover have $\langle finite\ \{(u, \ ev\ a\ \# \ v) \mid u\ v.\ ?f\ t\ u\ v\} \rangle$
 by (simp add: *(2) $\langle finite\ \{(u, \ v).\ ?f\ t\ u\ v\} \rangle$)
 moreover have $\langle finite\ \{(ev\ a\ \# \ u, \ ev\ a\ \# \ v) \mid u\ v.\ ?f\ t\ u\ v\} \rangle$
 by (simp add: *(3) $\langle finite\ \{(u, \ v).\ ?f\ t\ u\ v\} \rangle$)
 ultimately show $\langle finite\ \{(u, \ v).\ ?f\ (e\ \# \ t)\ u\ v\} \rangle$
 by (simp add: finite-subset)
 next
 show $\langle finite\ \{(u, \ v).\ ?f\ (e\ \# \ t)\ u\ v\} \rangle$ if $\langle e = \checkmark(r-s) \rangle$ for $r-s$
 proof (cases $\langle r-s \in range\ tick\ join \rangle$)
 assume $\langle r-s \in range\ tick\ join \rangle$
 then obtain $r\ s$ where $\langle r \otimes \checkmark\ s = [r-s] \rangle$ by blast
 hence $\langle ?f\ (e\ \# \ t)\ u\ v \implies$
 $u \neq [] \wedge hd\ u = \checkmark(r) \wedge v \neq [] \wedge hd\ v = \checkmark(s) \wedge ?f\ t\ (tl\ u)\ (tl\ v) \rangle$ for $u\ v$
 by (cases u ; cases v)
 (auto simp add: $\langle e = \checkmark(r-s) \rangle$ setinterleaving_{ptick}-simps inj-tick-join
 split: event_{ptick}.splits option.split-asm if-split-asm)
 hence $\langle \{(u, \ v).\ ?f\ (e\ \# \ t)\ u\ v\} \subseteq \{(\checkmark(r) \# \ u, \ \checkmark(s) \# \ v) \mid u\ v.\ ?f\ t\ u\ v\} \rangle$
 by (simp add: subset-iff) (metis list.collapse)
 moreover have $\langle finite\ \{(\checkmark(r) \# \ u, \ \checkmark(s) \# \ v) \mid u\ v.\ ?f\ t\ u\ v\} \rangle$
 by (simp add: *(3) $\langle finite\ \{(u, \ v).\ ?f\ t\ u\ v\} \rangle$)
 ultimately show $\langle finite\ \{(u, \ v).\ ?f\ (e\ \# \ t)\ u\ v\} \rangle$
 by (simp add: finite-subset)
 next
 assume $\langle r-s \notin range\ tick\ join \rangle$
 hence $\langle \neg\ ?f\ (e\ \# \ t)\ u\ v \rangle$ for $u\ v$
 by (cases u ; cases v)
 (auto simp add: $\langle e = \checkmark(r-s) \rangle$ setinterleaving_{ptick}-simps
 split: event_{ptick}.splits option.split-asm)
 thus $\langle finite\ \{(u, \ v).\ ?f\ (e\ \# \ t)\ u\ v\} \rangle$ by simp
 qed
 qed
 qed

lemma *finite-setinterleaves_{ptick}-tick-join-Sync_{ptick}:*

$\langle finite\ \{(t-P, \ t-Q, \ u).\ u\ setinterleaves_{\checkmark(\otimes \checkmark)}((t-P, \ t-Q), \ A) \wedge$
 $(\exists v. \ t = u @ v \wedge ftF\ v \wedge (tF\ u \vee v = []))\} \rangle$
 (is $\langle finite\ \{(t-P, \ t-Q, \ u).\ ?f\ u\ t-P\ t-Q \wedge ?g\ t\ u\} \rangle$)

proof –

have $\langle \{(t-P, \ t-Q, \ u) \mid t-P\ t-Q.\ ?f\ u\ t-P\ t-Q\} \subseteq$
 $(\lambda(t-P, \ t-Q).\ (t-P, \ t-Q, \ u))\ \langle \{(t-P, \ t-Q).\ ?f\ u\ t-P\ t-Q\} \rangle$ for u by auto
 hence $\langle finite\ \{(t-P, \ t-Q, \ u) \mid t-P\ t-Q.\ ?f\ u\ t-P\ t-Q\} \rangle$ for u
 by (rule finite-subset) (simp add: finite-setinterleaves_{ptick}-tick-join)
 moreover have $\langle \{(t-P, \ t-Q, \ u).\ ?f\ u\ t-P\ t-Q \wedge ?g\ t\ u\} \subseteq$
 $(\bigcup u \in \{u. \ u \leq t\}.\ \{(t-P, \ t-Q, \ u) \mid t-P\ t-Q.\ ?f\ u\ t-P\ t-Q\}) \rangle$

```

    unfolding less-eq-list-def prefix-def by blast
    moreover have  $\langle \text{finite } \{u. u \leq t\} \rangle$  by (prove-finite-subset-of-prefixes t)
    ultimately show ?thesis by (simp add: finite-subset)
qed

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 × 's) eventptick ⇒ ('a, 's × 'r) eventptick⟩  
  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)))⟩  
  by (cases r-s) simp
```

```
lemma tick-swap-tick-swap [simp] : ⟨tick-swap (tick-swap e) = e⟩  
proof (cases e)  
  show ⟨e = ev a ⇒ tick-swap (tick-swap e) = e⟩ for a by simp
```

```

next
  show  $\langle e = \checkmark(r-s) \implies \text{tick-swap} (\text{tick-swap } e) = e \rangle$  for  $r-s$ 
  by (cases  $r-s$ ) simp-all
qed

lemma tick-swap-comp-tick-swap [simp] :  $\langle \text{tick-swap} \circ \text{tick-swap} = \text{id} \rangle$ 
  by (rule ext) simp

lemma inj-tick-swap [simp] :  $\langle \text{inj tick-swap} \rangle$ 
  by (metis injI tick-swap-tick-swap)

lemma surj-tick-swap [simp] :  $\langle \text{surj tick-swap} \rangle$ 
  by (metis surjI tick-swap-tick-swap)

lemma bij-tick-swap [simp] :  $\langle \text{bij tick-swap} \rangle$ 
  by (simp add: bij-betw-def)

lemma bij-betw-tick-swap :
   $\langle \text{bij-betw tick-swap (range ev)} \rangle$ 
   $\langle \text{bij-betw tick-swap (range tick)} \rangle$ 
  by (auto simp add: bij-betw-def inj-on-def set-eq-iff image-iff)

lemma ev-eq-tick-swap-iff [simp] :  $\langle \text{ev } a = \text{tick-swap } e \iff e = \text{ev } a \rangle$ 
  and tick-swap-eq-ev-iff [simp] :  $\langle \text{tick-swap } e = \text{ev } a \iff e = \text{ev } a \rangle$ 
  and tick-eq-tick-swap-iff [simp] :  $\langle \checkmark((r, s)) = \text{tick-swap } e \iff e = \checkmark((s, r)) \rangle$ 
  and tick-swap-eq-tick-iff [simp] :  $\langle \text{tick-swap } e = \checkmark((r, s)) \iff e = \checkmark((s, r)) \rangle$ 
  by (cases  $e$ , auto)+

```

Swapping a Trace

```

fun trace-tick-swap ::  $\langle ('a, ('r \times 's)) \text{ trace}_{ptick} \Rightarrow ('a, ('s \times 'r)) \text{ trace}_{ptick} \rangle$ 
  where  $\langle \text{trace-tick-swap } [] = [] \rangle$ 
  |  $\langle \text{trace-tick-swap } (\text{ev } a \# t) = \text{ev } a \# \text{trace-tick-swap } t \rangle$ 
  |  $\langle \text{trace-tick-swap } (\checkmark((r, s)) \# t) = \checkmark((s, r)) \# \text{trace-tick-swap } t \rangle$ 

lemma trace-tick-swap-tick-Cons :
   $\langle \text{trace-tick-swap } (\checkmark(r-s) \# t) = (\text{case } r-s \text{ of } (r, s) \Rightarrow \checkmark((s, r)) \# \text{trace-tick-swap } t) \rangle$ 
  by (cases  $r-s$ ) simp

lemma trace-tick-swap-def :  $\langle \text{trace-tick-swap} = \text{map tick-swap} \rangle$ 
proof (rule ext)
  show  $\langle \text{trace-tick-swap } t = \text{map tick-swap } t \rangle$  for  $t :: \langle ('a, ('r \times 's)) \text{ trace}_{ptick} \rangle$ 
  by (induct  $t$  rule: trace-tick-swap.induct) simp-all
qed

```

lemma *trace-tick-swap-append* : $\langle \text{trace-tick-swap } (t @ u) = \text{trace-tick-swap } t @ \text{trace-tick-swap } u \rangle$
by (*simp add: trace-tick-swap-def*)

lemma *trace-tick-swap-singl* [*simp*] : $\langle \text{trace-tick-swap } [e] = [\text{tick-swap } e] \rangle$
by (*cases e*) *auto*

lemma *trace-tick-swap-comp-trace-tick-swap* [*simp*] : $\langle \text{trace-tick-swap} \circ \text{trace-tick-swap} = \text{id} \rangle$
by (*simp add: trace-tick-swap-def*)

lemma *trace-tick-swap-trace-tick-swap* [*simp*] : $\langle \text{trace-tick-swap } (\text{trace-tick-swap } t) = t \rangle$
by (*metis comp-def id-apply trace-tick-swap-comp-trace-tick-swap*)

lemma *inj-trace-tick-swap* [*simp*] : $\langle \text{inj trace-tick-swap} \rangle$
by (*metis injI trace-tick-swap-trace-tick-swap*)

lemma *surj-trace-tick-swap* [*simp*] : $\langle \text{surj trace-tick-swap} \rangle$
by (*metis surjI trace-tick-swap-trace-tick-swap*)

lemma *bij-trace-tick-swap* [*simp*] : $\langle \text{bij trace-tick-swap} \rangle$
by (*simp add: bij-betw-def*)

lemma *strict-mono-trace-tick-swap* : $\langle \text{strict-mono trace-tick-swap} \rangle$
by (*unfold trace-tick-swap-def*)
(rule strict-mono-map, simp add: strict-monoI)

lemma *image-trace-tick-swap-min-elems* :
 $\langle \text{trace-tick-swap } '(\text{min-elems } T) = \text{min-elems } (\text{trace-tick-swap } 'T) \rangle$
proof (*intro subset-antisym subsetI*)
show $\langle t \in \text{trace-tick-swap } ' \text{min-elems } T \implies t \in \text{min-elems } (\text{trace-tick-swap } 'T) \rangle$ **for** *t*
by (*auto simp add: min-elems-def less-list-def less-eq-list-def prefix-def*)
(metis Prefix-Order.prefixI Prefix-Order.same-prefix-nil
trace-tick-swap-append trace-tick-swap-trace-tick-swap)
next
show $\langle t \in \text{min-elems } (\text{trace-tick-swap } 'T) \implies t \in \text{trace-tick-swap } ' \text{min-elems } T \rangle$ **for** *t*
by (*auto simp add: min-elems-def less-list-def less-eq-list-def prefix-def image-iff*)
(metis trace-tick-swap-append trace-tick-swap-trace-tick-swap)
qed

lemma *Nil-eq-trace-tick-swap-iff* [*iff*] : $\langle [] = \text{trace-tick-swap } t \longleftrightarrow t = [] \rangle$
and *trace-tick-swap-eq-Nil-iff* [*iff*] : $\langle \text{trace-tick-swap } t = [] \longleftrightarrow t = [] \rangle$
by (*metis trace-tick-swap.simps(1) trace-tick-swap-trace-tick-swap*)**+**

lemma *ev-Cons-eq-trace-tick-swap-iff* [iff] :
 $\langle \text{ev } a \# t = \text{trace-tick-swap } u \longleftrightarrow u = \text{ev } a \# \text{trace-tick-swap } t \rangle$
and *trace-tick-swap-eq-ev-Cons-iff* [iff] :
 $\langle \text{trace-tick-swap } u = \text{ev } a \# t \longleftrightarrow u = \text{ev } a \# \text{trace-tick-swap } t \rangle$
by (metis trace-tick-swap.simps(2) trace-tick-swap-trace-tick-swap)+

lemma *tick-Cons-eq-trace-tick-swap-iff* [iff] :
 $\langle \checkmark((r, s)) \# t = \text{trace-tick-swap } u \longleftrightarrow u = \checkmark((s, r)) \# \text{trace-tick-swap } t \rangle$
and *trace-tick-swap-eq-tick-Cons-iff* [iff] :
 $\langle \text{trace-tick-swap } u = \checkmark((r, s)) \# t \longleftrightarrow u = \checkmark((s, r)) \# \text{trace-tick-swap } t \rangle$
by (metis trace-tick-swap.simps(3) trace-tick-swap-trace-tick-swap)+

lemma *snoc-ev-eq-trace-tick-swap-iff* [iff] :
 $\langle t @ [\text{ev } a] = \text{trace-tick-swap } u \longleftrightarrow u = \text{trace-tick-swap } t @ [\text{ev } a] \rangle$
and *trace-tick-swap-eq-snoc-ev-iff* [iff] :
 $\langle \text{trace-tick-swap } u = t @ [\text{ev } a] \longleftrightarrow u = \text{trace-tick-swap } t @ [\text{ev } a] \rangle$
by (metis trace-tick-swap-append trace-tick-swap.simps(1, 2) trace-tick-swap-trace-tick-swap)+

lemma *snoc-tick-eq-trace-tick-swap-iff* [iff] :
 $\langle t @ [\checkmark((r, s))] = \text{trace-tick-swap } u \longleftrightarrow u = \text{trace-tick-swap } t @ [\checkmark((s, r))] \rangle$
and *trace-tick-swap-eq-snoc-tick-iff* [iff] :
 $\langle \text{trace-tick-swap } u = t @ [\checkmark((r, s))] \longleftrightarrow u = \text{trace-tick-swap } t @ [\checkmark((s, r))] \rangle$
by (metis trace-tick-swap-append trace-tick-swap.simps(1, 3) trace-tick-swap-trace-tick-swap)+

lemma *trace-tick-swap-eq-ev-ConsE* :
 $\langle \text{trace-tick-swap } u = \text{ev } a \# t \implies (\bigwedge u'. u = \text{ev } a \# u' \implies t = \text{trace-tick-swap } u' \implies \text{thesis}) \implies \text{thesis} \rangle$
and *trace-tick-swap-eq-tick-ConsE* :
 $\langle \text{trace-tick-swap } u = \checkmark((r, s)) \# t \implies (\bigwedge u'. u = \checkmark((s, r)) \# u' \implies t = \text{trace-tick-swap } u' \implies \text{thesis}) \implies \text{thesis} \rangle$
and *trace-tick-swap-eq-snoc-evE* :
 $\langle \text{trace-tick-swap } u = t @ [\text{ev } a] \implies (\bigwedge u'. u = u' @ [\text{ev } a] \implies t = \text{trace-tick-swap } u' \implies \text{thesis}) \implies \text{thesis} \rangle$
and *trace-tick-swap-eq-snoc-tickE* :
 $\langle \text{trace-tick-swap } u = t @ [\checkmark((r, s))] \implies (\bigwedge u'. u = u' @ [\checkmark((s, r))] \implies t = \text{trace-tick-swap } u' \implies \text{thesis}) \implies \text{thesis} \rangle$
by (simp, metis trace-tick-swap-trace-tick-swap)+

lemma *trace-tick-swap-tickFree* :
 $\langle tF t \implies \text{trace-tick-swap } t = \text{map } (\text{ev} \circ \text{of-ev}) t \rangle$ **for** $t :: \langle ('a, ('r \times 's)) \text{trace}_{ptick} \rangle$
proof (induct t)
show $\langle \text{trace-tick-swap } [] = \text{map } (\text{ev} \circ \text{of-ev}) [] \rangle$ **by** simp
next
fix e **and** $t :: \langle ('a, ('r \times 's)) \text{trace}_{ptick} \rangle$
assume $\langle tF (e \# t) \rangle$ **and** $\langle tF t \implies \text{trace-tick-swap } t = \text{map } (\text{ev} \circ \text{of-ev}) t \rangle$

moreover from $\langle tF (e \# t) \rangle$ obtain a where $\langle e = ev \ a \rangle \langle tF \ t \rangle$
 by (meson is-ev-def tickFree-Cons-iff)
 ultimately show $\langle trace\text{-}tick\text{-}swap (e \# t) = map (ev \circ of\text{-}ev) (e \# t) \rangle$ by simp
 qed

lemma trace-tick-swap-front-tickFree :

$\langle trace\text{-}tick\text{-}swap \ t = (\text{ if } tF \ t \text{ then } map (ev \circ of\text{-}ev) \ t$
 $\text{ else } map (ev \circ of\text{-}ev) (butlast \ t) \ @ \ [case \ last \ t \ of \ \checkmark((r, \ s)) \Rightarrow \checkmark((s,$
 $r))]) \rangle$
 if $\langle ftF \ t \rangle$ for $t :: \langle ('a, ('r \times 's)) \ trace_{ptick} \rangle$
proof –
 show ?thesis
proof (split if-split, intro conjI impI)
 show $\langle tF \ t \Rightarrow trace\text{-}tick\text{-}swap \ t = map (ev \circ of\text{-}ev) \ t \rangle$
 by (simp add: trace-tick-swap-tickFree)
 next
 assume $\langle \neg \ tF \ t \rangle$
 with $\langle tF \ t \rangle$ obtain $t' \ r \ s$ where $\langle t = t' \ @ \ [\checkmark((r, \ s))] \rangle \langle tF \ t' \rangle$
 by (metis front-tickFree-append-iff nonTickFree-n-frontTickFree not-Cons-self2
 surj-pair)
 hence $\langle trace\text{-}tick\text{-}swap \ t = trace\text{-}tick\text{-}swap \ t' \ @ \ [\checkmark((s, \ r))] \rangle$
 by (metis trace-tick-swap-append trace-tick-swap.simps(1, 3))
 also from $\langle tF \ t' \rangle \langle t = t' \ @ \ [\checkmark((r, \ s))] \rangle$
 have $\langle trace\text{-}tick\text{-}swap \ t' = map (ev \circ of\text{-}ev) (butlast \ t) \rangle$ by (simp add: trace-tick-swap-tickFree)
 also from $\langle t = t' \ @ \ [\checkmark((r, \ s))] \rangle$
 have $\langle [\checkmark((s, \ r))] = [case \ last \ t \ of \ \checkmark((r, \ s)) \Rightarrow \checkmark((s, \ r))] \rangle$ by simp
 finally show $\langle trace\text{-}tick\text{-}swap \ t = map (ev \circ of\text{-}ev) (butlast \ t) \ @$
 $[case \ last \ t \ of \ \checkmark((r, \ s)) \Rightarrow \checkmark((s, \ r))] \rangle$.
 qed
 qed

lemma tickFree-trace-tick-swap-iff [simp] : $\langle tF (trace\text{-}tick\text{-}swap \ t) \longleftrightarrow tF \ t \rangle$
 by (metis tickFree-map-ev-comp trace-tick-swap-tickFree trace-tick-swap-trace-tick-swap)

lemma front-tickFree-trace-tick-swap-iff [simp] : $\langle ftF (trace\text{-}tick\text{-}swap \ t) \longleftrightarrow ftF \ t \rangle$
 by (metis (no-types, lifting) front-tickFree-iff-tickFree-butlast map-butlast
 tickFree-trace-tick-swap-iff trace-tick-swap-def)

Swapping a Refusal

definition refusal-tick-swap :: $\langle ('a, ('r \times 's)) \ refusal_{ptick} \Rightarrow ('a, ('s \times 'r)) \ re-$
 $fusal_{ptick} \rangle$
 where $\langle refusal\text{-}tick\text{-}swap \ X = tick\text{-}swap \ 'X \rangle$

lemma refusal-tick-swap-empty [simp] : $\langle refusal\text{-}tick\text{-}swap \ \{\} = \{\} \rangle$
 by (simp add: refusal-tick-swap-def)

lemma *refusal-tick-swap-insert* [*simp*] :
 $\langle \text{refusal-tick-swap } (\text{insert } x \ X) = \text{insert } (\text{tick-swap } x) (\text{refusal-tick-swap } X) \rangle$
by (*simp add: refusal-tick-swap-def*)

lemma *refusal-tick-swap-union* :
 $\langle \text{refusal-tick-swap } (X \cup Y) = \text{refusal-tick-swap } X \cup \text{refusal-tick-swap } Y \rangle$
by (*simp add: refusal-tick-swap-def image-Un*)

lemma *refusal-tick-swap-diff* :
 $\langle \text{refusal-tick-swap } (X - Y) = \text{refusal-tick-swap } X - \text{refusal-tick-swap } Y \rangle$
by (*simp add: refusal-tick-swap-def image-set-diff*)

lemma *refusal-tick-swap-inter* :
 $\langle \text{refusal-tick-swap } (X \cap Y) = \text{refusal-tick-swap } X \cap \text{refusal-tick-swap } Y \rangle$
by (*simp add: refusal-tick-swap-def image-Int*)

lemma *refusal-tick-swap-singl* : $\langle \text{refusal-tick-swap } \{e\} = \{\text{tick-swap } e\} \rangle$ **by** *simp*

lemma *refusal-tick-swap-comp-refusal-tick-swap* [*simp*] :
 $\langle \text{refusal-tick-swap} \circ \text{refusal-tick-swap} = \text{id} \rangle$
by (*auto simp add: refusal-tick-swap-def image-iff*)

lemma *refusal-tick-swap-refusal-tick-swap* [*simp*] :
 $\langle \text{refusal-tick-swap } (\text{refusal-tick-swap } X) = X \rangle$
by (*simp add: comp-eq-dest-lhs*)

lemma *inj-refusal-tick-swap* [*simp*] : $\langle \text{inj } \text{refusal-tick-swap} \rangle$
by (*metis injI refusal-tick-swap-refusal-tick-swap*)

lemma *surj-refusal-tick-swap* [*simp*] : $\langle \text{surj } \text{refusal-tick-swap} \rangle$
by (*metis surjI refusal-tick-swap-refusal-tick-swap*)

lemma *bij-refusal-tick-swap* [*simp*] : $\langle \text{bij } \text{refusal-tick-swap} \rangle$
by (*simp add: bij-betw-def*)

lemma *strict-mono-refusal-tick-swap* : $\langle \text{strict-mono } \text{refusal-tick-swap} \rangle$
by (*rule strict-monoI*)
(metis refusal-tick-swap-refusal-tick-swap sup.strict-order-iff refusal-tick-swap-union)

lemma *empty-eq-refusal-tick-swap-iff* [*iff*] : $\langle \{\} = \text{refusal-tick-swap } X \longleftrightarrow X = \{\} \rangle$
and *refusal-tick-swap-eq-empty-iff* [*iff*] : $\langle \text{refusal-tick-swap } X = \{\} \longleftrightarrow X = \{\} \rangle$
by (*simp-all add: refusal-tick-swap-def*)

lemma *insert-ev-eq-refusal-tick-swap-iff* [*iff*] :
 $\langle \text{insert } (\text{ev } a) \ X = \text{refusal-tick-swap } Y \longleftrightarrow Y = \text{insert } (\text{ev } a) (\text{refusal-tick-swap } X) \rangle$

and *refusal-tick-swap-eq-insert-ev-iff* [iff] :
 $\langle \text{refusal-tick-swap } Y = \text{insert } (ev \ a) \ X \longleftrightarrow Y = \text{insert } (ev \ a) \ (\text{refusal-tick-swap } X) \rangle$
by (*metis refusal-tick-swap-insert refusal-tick-swap-refusal-tick-swap tick-swap.simps(1)*) +

lemma *insert-tick-eq-refusal-tick-swap-iff* [iff] :
 $\langle \text{insert } \checkmark((r, s)) \ X = \text{refusal-tick-swap } Y \longleftrightarrow Y = \text{insert } \checkmark((s, r)) \ (\text{refusal-tick-swap } X) \rangle$
and *refusal-tick-swap-eq-insert-tick-iff* [iff] :
 $\langle \text{refusal-tick-swap } Y = \text{insert } \checkmark((r, s)) \ X \longleftrightarrow Y = \text{insert } \checkmark((s, r)) \ (\text{refusal-tick-swap } X) \rangle$
by (*metis refusal-tick-swap-insert refusal-tick-swap-refusal-tick-swap tick-swap.simps(2)*) +

lemma *refusal-tick-swap-eq-insert-evE* :
 $\langle \text{refusal-tick-swap } Y = \text{insert } (ev \ a) \ X \implies (\bigwedge Y'. Y = \text{insert } (ev \ a) \ Y' \implies X = \text{refusal-tick-swap } Y' \implies \text{thesis}) \implies \text{thesis} \rangle$
and *refusal-tick-swap-eq-insert-tickE* :
 $\langle \text{refusal-tick-swap } Y = \text{insert } \checkmark((r, s)) \ X \implies (\bigwedge Y'. Y = \text{insert } \checkmark((s, r)) \ Y' \implies X = \text{refusal-tick-swap } Y' \implies \text{thesis}) \implies \text{thesis} \rangle$
by (*simp, metis refusal-tick-swap-refusal-tick-swap*) +

lemma *refusal-tick-swap-tickFree* :
 $\langle X \subseteq \text{range } ev \implies \text{refusal-tick-swap } X = (ev \circ \text{of-ev}) \ 'X \rangle$
by (*force simp add: refusal-tick-swap-def*)

lemma *tickFree-refusal-tick-swap-iff* :
 $\langle \text{refusal-tick-swap } X \subseteq \text{range } ev \longleftrightarrow X \subseteq \text{range } ev \rangle$
by (*simp add: refusal-tick-swap-def subset-iff image-def*)
(metis tick-swap.simps(1) tick-swap-tick-swap)

The old version of interleaving of traces is not affected.

lemma *setinterleaves-imp-setinterleaves-trace-tick-swap* :
 $\langle t \text{ setinterleaves } ((u, v), S) \implies \text{trace-tick-swap } t \text{ setinterleaves } ((\text{trace-tick-swap } u, \text{trace-tick-swap } v), \text{refusal-tick-swap } S) \rangle$

proof (*induct* $\langle (u, S, v) \rangle$ *arbitrary: t u v rule: setinterleaving.induct*)
case 1 **thus** ?case **by** *simp*
next
case (2 *y v*)
from 2.prem1 **obtain** *t'* **where** $\langle y \notin S \rangle \langle t = y \# t' \rangle \langle t' \text{ setinterleaves } (([], v), S) \rangle$
by (*auto split: if-split-asm*)
from 2.hyps[*OF* $\langle y \notin S \rangle \langle t' \text{ setinterleaves } (([], v), S) \rangle$]
have $\langle \text{trace-tick-swap } t' \text{ setinterleaves } (([], \text{trace-tick-swap } v), \text{refusal-tick-swap } S) \rangle$ **by** *simp*
with $\langle y \notin S \rangle$ **show** ?case **by** (*cases y*) (*auto simp add:* $\langle t = y \# t' \rangle$ *refusal-tick-swap-def split: prod.split*)
next

```

    case (3 x u)
    from 3.premis obtain t' where  $\langle x \notin S \rangle \langle t = x \# t' \rangle \langle t' \text{ setinterleaves } ((u, []), S) \rangle$ 
    by (auto split: if-split-asm)
    from 3.hyps[OF  $\langle x \notin S \rangle \langle t' \text{ setinterleaves } ((u, []), S) \rangle]$ 
    have  $\langle \text{trace-tick-swap } t' \text{ setinterleaves } ((\text{trace-tick-swap } u, []), \text{refusal-tick-swap } S) \rangle$  by simp
    with  $\langle x \notin S \rangle$  show ?case by (cases x) (auto simp add:  $\langle t = x \# t' \rangle$  refusal-tick-swap-def split: prod.split)
next
    case (4 x u y v)
    from 4.premis
    consider (both-in) t' where  $\langle x \in S \rangle \langle y \in S \rangle \langle x = y \rangle \langle t = x \# t' \rangle \langle t' \text{ setinterleaves } ((u, v), S) \rangle$ 
    | (inR-mvL) t' where  $\langle x \notin S \rangle \langle y \in S \rangle \langle t = x \# t' \rangle \langle t' \text{ setinterleaves } ((u, y \# v), S) \rangle$ 
    | (inL-mvR) t' where  $\langle x \in S \rangle \langle y \notin S \rangle \langle t = y \# t' \rangle \langle t' \text{ setinterleaves } ((x \# u, v), S) \rangle$ 
    | (notin-mvL) t' where  $\langle x \notin S \rangle \langle y \notin S \rangle \langle t = x \# t' \rangle \langle t' \text{ setinterleaves } ((u, y \# v), S) \rangle$ 
    | (notin-mvR) t' where  $\langle x \notin S \rangle \langle y \notin S \rangle \langle t = y \# t' \rangle \langle t' \text{ setinterleaves } ((x \# u, v), S) \rangle$ 
    by (auto split: if-split-asm)
    thus ?case
    proof cases
    case both-in
    from 4.hyps(1)[OF both-in(1-3, 5)] both-in(1-3)
    show ?thesis by (cases y, auto simp add: both-in(4) refusal-tick-swap-def split: prod.split)
    next
    case inR-mvL
    have  $\langle \neg y \notin S \rangle$  by (simp add: inR-mvL(2))
    from 4.hyps(5)[OF inR-mvL(1)  $\langle \neg y \notin S \rangle$  inR-mvL(4)] inR-mvL(1, 2)
    show ?thesis by (cases x, auto simp add: inR-mvL(3) refusal-tick-swap-def SyncSingleHeadAdd image-iff split: prod.split)
    next
    case inL-mvR
    have  $* : \langle a \text{ setinterleaves } ((t, u), \text{tick-swap } S) \rangle \implies h \notin \text{tick-swap } S \implies \langle h \# a \rangle \text{ setinterleaves } ((t, h \# u), \text{tick-swap } S) \rangle$  for a t h u
    by (cases t, auto split: if-split-asm)
    from 4.hyps(2)[OF inL-mvR(1, 2, 4)] inL-mvR(1, 2)
    show ?thesis by (cases y, auto simp add: inL-mvR(3) refusal-tick-swap-def image-iff * split: prod.split)
    next
    case notin-mvL
    from 4.hyps(3)[OF notin-mvL(1, 2, 4)] notin-mvL(1, 2)
    show ?thesis by (cases y, auto simp add: notin-mvL(3) refusal-tick-swap-def split: prod.split)
    (simp-all add: inj-image-mem-iff trace-tick-swap-def)

```



```

next
  case notin-mvR
  from 4.hyps(4)[OF notin-mvR(1, 2, 4)] notin-mvR(1, 2)
  show ?thesis by (cases x, auto simp add: notin-mvR(3) refusal-tick-swap-def
split: prod.split)
    (simp-all add: inj-image-mem-iff trace-tick-swap-def)
qed
qed

```

lemma *trace-tick-swap-setinterleaves-iff* :
 $\langle \text{trace-tick-swap } t \text{ setinterleaves } ((u, v), S) \longleftrightarrow$
 $t \text{ setinterleaves } ((\text{trace-tick-swap } u, \text{trace-tick-swap } v), \text{refusal-tick-swap } S) \rangle$
by (metis refusal-tick-swap-refusal-tick-swap trace-tick-swap-trace-tick-swap
setinterleaves-imp-setinterleaves-trace-tick-swap)

Swapping a Failure

definition *failure-tick-swap* :: $\langle ('a, ('r \times 's)) \text{ failure}_{ptick} \Rightarrow ('a, ('s \times 'r)) \text{ failure}_{ptick} \rangle$
where $\langle \text{failure-tick-swap } F \equiv \text{case } F \text{ of } (t, X) \Rightarrow (\text{trace-tick-swap } t, \text{refusal-tick-swap } X) \rangle$

lemma *failure-tick-swap-empty* [simp] : $\langle \text{failure-tick-swap } ([], \{\}) = ([], \{\}) \rangle$
by (simp add: failure-tick-swap-def)

lemma *failure-tick-swap-comp-failure-tick-swap* [simp] :
 $\langle \text{failure-tick-swap} \circ \text{failure-tick-swap} = \text{id} \rangle$
by (auto simp add: failure-tick-swap-def)

lemma *failure-tick-swap-failure-tick-swap* [simp] :
 $\langle \text{failure-tick-swap } (\text{failure-tick-swap } F) = F \rangle$
by (simp add: comp-eq-dest-lhs)

lemma *inj-failure-tick-swap* [simp] : $\langle \text{inj failure-tick-swap} \rangle$
by (metis injI failure-tick-swap-failure-tick-swap)

lemma *surj-failure-tick-swap* [simp] : $\langle \text{surj failure-tick-swap} \rangle$
by (metis surjI failure-tick-swap-failure-tick-swap)

lemma *bij-failure-tick-swap* [simp] : $\langle \text{bij failure-tick-swap} \rangle$
by (simp add: bij-betw-def)

lemma *empty-eq-failure-tick-swap-iff* [iff] : $\langle ([], \{\}) = \text{failure-tick-swap } F \longleftrightarrow F = ([], \{\}) \rangle$
and *failure-tick-swap-eq-empty-iff* [iff] : $\langle \text{failure-tick-swap } F = ([], \{\}) \longleftrightarrow F = ([], \{\}) \rangle$

$\langle [], \{\} \rangle$
by (*auto simp add: failure-tick-swap-def split: prod.split*)

5.1.2 The Operator

Definition

lift-definition *TickSwap* :: $\langle ('a, 'r \times 's) \text{ process}_{ptick} \Rightarrow ('a, 's \times 'r) \text{ process}_{ptick} \rangle$
is $\langle \lambda P. (\{(t, X). \text{failure-tick-swap } (t, X) \in \mathcal{F} P\}, \{t. \text{trace-tick-swap } t \in \mathcal{D} P\}) \rangle$

— One might expect $\lambda P. (\text{failure-tick-swap } ' \mathcal{F} P, \text{trace-tick-swap } ' \mathcal{D} P)$ instead. This is equivalent, see the projections below, but easier for the following proof obligation.

proof —

show $\langle ?thesis P \rangle$ (**is** $\langle is\text{-process } (?f, ?d) \rangle$) **for** P

proof (*unfold is-process-def FAILURES-def DIVERGENCES-def fst-conv snd-conv, intro conjI impI allI*)

show $\langle [], \{\} \rangle \in ?f$ **by** (*simp add: is-processT1*)

next

show $\langle (t, X) \in ?f \implies ftF t \rangle$ **for** $t X$

by (*simp add: failure-tick-swap-def*)

(*use is-processT2 front-tickFree-trace-tick-swap-iff in blast*)

next

show $\langle (t @ u, \{\}) \in ?f \implies (t, \{\}) \in ?f \rangle$ **for** $t u$

by (*simp add: failure-tick-swap-def (metis trace-tick-swap-append is-processT3)*)

next

show $\langle (t, Y) \in ?f \wedge X \subseteq Y \implies (t, X) \in ?f \rangle$ **for** $t X Y$

by (*simp add: failure-tick-swap-def*)

(*metis is-processT4 le-iff-sup refusal-tick-swap-union*)

next

fix $t X Y$ **assume** $\langle (t, X) \in ?f \wedge (\forall e. e \in Y \longrightarrow (t @ [e], \{\}) \notin ?f) \rangle$

hence $\langle (\text{trace-tick-swap } t, \text{refusal-tick-swap } X) \in \mathcal{F} P \wedge$

$(\forall e. e \in \text{refusal-tick-swap } Y \longrightarrow (\text{trace-tick-swap } t @ [e], \{\}) \notin \mathcal{F} P) \rangle$

by (*auto simp add: failure-tick-swap-def refusal-tick-swap-def trace-tick-swap-append*)

thus $\langle (t, X \cup Y) \in ?f \rangle$

by (*simp add: failure-tick-swap-def is-processT5 refusal-tick-swap-union*)

next

show $\langle (t @ [\checkmark(s-r)], \{\}) \in ?f \implies (t, X - \{\checkmark(s-r)\}) \in ?f \rangle$ **for** $t X s-r$

by (*cases s-r (simp add: failure-tick-swap-def trace-tick-swap-append*

is-processT6 refusal-tick-swap-diff)

next

show $\langle t \in ?d \wedge tF t \wedge ftF u \implies t @ u \in ?d \rangle$ **for** $t u$

by (*simp add: trace-tick-swap-append is-processT7*)

next

show $\langle t \in ?d \implies (t, X) \in ?f \rangle$ **for** $t X$

by (*simp add: failure-tick-swap-def is-processT8*)

next

show $\langle t @ [\checkmark(s-r)] \in ?d \implies t \in ?d \rangle$ **for** $t s-r$

by (*cases s-r (auto simp add: trace-tick-swap-append intro: is-processT9)*)

qed

qed

Projections

lemma $F\text{-TickSwap}' : \langle \mathcal{F} (\text{TickSwap } P) = \{(t, X). \text{failure-tick-swap } (t, X) \in \mathcal{F} P\} \rangle$

by (*simp add: Failures.rep-eq TickSwap.rep-eq FAILURES-def*)

lemma $D\text{-TickSwap}' : \langle \mathcal{D} (\text{TickSwap } P) = \{t. \text{trace-tick-swap } t \in \mathcal{D} P\} \rangle$

by (*simp add: Divergences.rep-eq TickSwap.rep-eq DIVERGENCES-def*)

lemma $T\text{-TickSwap}' : \langle \mathcal{T} (\text{TickSwap } P) = \{t. \text{trace-tick-swap } t \in \mathcal{T} P\} \rangle$

by (*simp add: set-eq-iff F-TickSwap' failure-tick-swap-def flip: T-F-spec*)

lemmas $\text{TickSwap-projs}' = F\text{-TickSwap}' D\text{-TickSwap}' T\text{-TickSwap}'$

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

lemma $F\text{-TickSwap} : \langle \mathcal{F} (\text{TickSwap } P) = \text{failure-tick-swap } ' \mathcal{F} P \rangle$

by (*simp add: set-eq-iff F-TickSwap'*)

(*metis (no-types, lifting) failure-tick-swap-failure-tick-swap imageE image-eqI*)

lemma $D\text{-TickSwap} : \langle \mathcal{D} (\text{TickSwap } P) = \text{trace-tick-swap } ' \mathcal{D} P \rangle$

by (*simp add: set-eq-iff D-TickSwap'*)

(*metis (no-types, lifting) trace-tick-swap-trace-tick-swap imageE image-eqI*)

lemma $T\text{-TickSwap} : \langle \mathcal{T} (\text{TickSwap } P) = \text{trace-tick-swap } ' \mathcal{T} P \rangle$

by (*simp add: set-eq-iff T-TickSwap'*)

(*metis (no-types, lifting) trace-tick-swap-trace-tick-swap imageE image-eqI*)

lemmas $\text{TickSwap-projs} = F\text{-TickSwap} D\text{-TickSwap} T\text{-TickSwap}$

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

lemma $F\text{-TickSwap}'' : \langle \mathcal{F} (\text{TickSwap } P) = \{(\text{trace-tick-swap } t, \text{refusal-tick-swap } X) \mid t X. (t, X) \in \mathcal{F} P \} \rangle$

by (*auto simp add: F-TickSwap failure-tick-swap-def*)

lemma $D\text{-TickSwap}'' : \langle \mathcal{D} (\text{TickSwap } P) = \{ \text{trace-tick-swap } t \mid t. t \in \mathcal{D} P \} \rangle$

by (*auto simp add: D-TickSwap*)

lemma $T\text{-TickSwap}'' : \langle \mathcal{T} (\text{TickSwap } P) = \{ \text{trace-tick-swap } t \mid t. t \in \mathcal{T} P \} \rangle$

by (*auto simp add: T-TickSwap*)

lemmas $\text{TickSwap-projs}'' = F\text{-TickSwap}'' D\text{-TickSwap}'' T\text{-TickSwap}''$

Properties

lemma $\text{events-TickSwap} [\text{simp}] : \langle \text{events-of } (\text{TickSwap } P) = \text{events-of } P \rangle$

by (*auto simp add: events-of-def T-TickSwap trace-tick-swap-def*)

lemma $\text{ticks-TickSwap} [\text{simp}] : \langle \text{ticks-of } (\text{TickSwap } P) = \{(s, r). (r, s) \in \text{ticks-of } P\} \rangle$

by (*auto simp add: ticks-of-def T-TickSwap' trace-tick-swap-append*)

(metis trace-tick-swap-trace-tick-swap)

lemma *strict-ticks-TickSwap* [simp] :
 $\langle \text{strict-ticks-of } (\text{TickSwap } P) = \{(s, r). (r, s) \in \text{strict-ticks-of } P\} \rangle$
by (auto simp add: strict-ticks-of-def TickSwap-projs' trace-tick-swap-append)
 (metis trace-tick-swap-trace-tick-swap)

lemma *trace-tick-swap-image-setinterleaving_{Pair}* :
 $\langle \text{trace-tick-swap } ' \text{setinterleaving}_{ptick} (\lambda r s. \lfloor (r, s) \rfloor, u, A, v) =$
 $\text{setinterleaving}_{ptick} (\lambda r s. \lfloor (r, s) \rfloor, v, A, u) \rangle$
for $u :: \langle ('a, 'r) \text{trace}_{ptick} \rangle$ **and** $v :: \langle ('a, 's) \text{trace}_{ptick} \rangle$
by (rule sym, induct $\langle (\lambda r :: 'r. \lambda s :: 's. \lfloor (r, s) \rfloor, u, A, v) \rangle$
 arbitrary: $u\ v$) (simp-all, safe, auto)

lemma *trace-tick-swap-setinterleaves_{Pair}-iff* [iff] :
 $\langle \text{trace-tick-swap } t \text{setinterleaves} \checkmark_{\lambda r s. \lfloor (r, s) \rfloor} ((u, v), A) \longleftrightarrow$
 $t \text{setinterleaves} \checkmark_{\lambda r s. \lfloor (r, s) \rfloor} ((v, u), A) \rangle$
by (metis (mono-tags, lifting) image-eqI trace-tick-swap-image-setinterleaving_{Pair}
 trace-tick-swap-trace-tick-swap)

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}* : $\langle \text{tick-swap} = \text{map-event}_{ptick} \text{id prod.swap} \rangle$
proof (rule ext)
show $\langle \text{tick-swap } e = \text{map-event}_{ptick} \text{id prod.swap } e \rangle$ **for** $e :: \langle ('a, 'r \times 's) \text{event}_{ptick} \rangle$
by (cases e) (auto split: event_{ptick}.splits prod.splits)
qed

lemma *trace-tick-swap-is-map-map-event_{ptick}* :
 $\langle \text{trace-tick-swap} = \text{map } (\text{map-event}_{ptick} \text{id prod.swap}) \rangle$
by (simp add: tick-swap-is-map-event_{ptick} trace-tick-swap-def)

lemma *refusal-tick-swap-is-image-map-event_{ptick}* :
 $\langle \text{refusal-tick-swap} = (') (\text{map-event}_{ptick} \text{id prod.swap}) \rangle$
by (rule ext) (simp add: refusal-tick-swap-def tick-swap-is-map-event_{ptick})

theorem *TickSwap-is-Renaming* :
 $\langle \text{TickSwap } P = \text{Renaming } P \text{id prod.swap} \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
proof (subst Process-eq-spec-optimized, safe)
fix t **assume** $\langle t \in \mathcal{D} \ ?lhs \rangle$
with *D-imp-front-tickFree* **have** $\langle \text{ftF } t \rangle$ **by** blast
define $t1$ **where** $\langle t1 \equiv \text{trace-tick-swap } (\text{if } \text{tF } t \text{ then } t \text{ else butlast } t) \rangle$
define $t2$ **where** $\langle t2 \equiv \text{if } \text{tF } t \text{ then } [] \text{ else } [\text{last } t] \rangle$
have $\langle t = \text{map } (\text{map-event}_{ptick} \text{id prod.swap}) t1 @ t2 \rangle$
by (simp add: t1-def t2-def flip: trace-tick-swap-is-map-map-event_{ptick})
 (metis append-butlast-last-id tickFree-Nil)
moreover from $\langle \text{ftF } t \rangle$ *front-tickFree-iff-tickFree-butlast* $t1$ -def **have** $\langle \text{tF } t1 \rangle$ **by**

auto
moreover have $\langle ftF\ t2 \rangle$ **by** (*simp add: t2-def*)
moreover from $t1\text{-def}\ D\text{-TickSwap}'\ \langle ftF\ t \rangle\ \langle t \in \mathcal{D}\ ?lhs \rangle$
div-butlast-when-non-tickFree-iff **have** $\langle t1 \in \mathcal{D}\ P \rangle$ **by** *blast*
ultimately show $\langle t \in \mathcal{D}\ ?rhs \rangle$ **unfolding** *D-Renaming* **by** *blast*
next
fix t **assume** $\langle t \in \mathcal{D}\ ?rhs \rangle$
then obtain $t1\ t2$
where $\langle t = \text{map}\ (\text{map-event}_{ptick}\ id\ \text{prod.swap})\ t1\ @\ t2 \rangle\ \langle tF\ t1 \rangle\ \langle ftF\ t2 \rangle\ \langle t1 \in \mathcal{D}\ P \rangle$
unfolding *D-Renaming* **by** *blast*
thus $\langle t \in \mathcal{D}\ ?lhs \rangle$ **by** (*simp add: D-TickSwap' trace-tick-swap-append is-processT7 flip: trace-tick-swap-is-map-map-event_{ptick}*)
next
fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle$
then obtain $t'\ X'$ **where** $\langle t = \text{trace-tick-swap}\ t' \rangle\ \langle X = \text{refusal-tick-swap}\ X' \rangle$
 $\langle (t', X') \in \mathcal{F}\ P \rangle$
unfolding *F-TickSwap failure-tick-swap-def* **by** *auto*
moreover have $\langle \text{map-event}_{ptick}\ id\ \text{prod.swap} - ' \text{refusal-tick-swap}\ X' = X' \rangle$
by (*simp add: set-eq-iff*) (*metis inj-image-mem-iff inj-tick-swap refusal-tick-swap-def tick-swap-is-map-event_{ptick}*)
ultimately show $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$
by (*auto simp add: F-Renaming simp flip: trace-tick-swap-is-map-map-event_{ptick}*)
next
fix $t\ X$ **assume** *same-div* : $\langle \mathcal{D}\ ?lhs = \mathcal{D}\ ?rhs \rangle$
assume $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$
then consider $\langle t \in \mathcal{D}\ ?rhs \rangle$
| t' **where** $\langle t = \text{map}\ (\text{map-event}_{ptick}\ id\ \text{prod.swap})\ t' \rangle\ \langle (t', \text{map-event}_{ptick}\ id\ \text{prod.swap} - ' X) \in \mathcal{F}\ P \rangle$
unfolding *Renaming-projs* **by** *blast*
thus $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle$
proof cases
from *same-div D-F* **show** $\langle t \in \mathcal{D}\ ?rhs \implies (t, X) \in \mathcal{F}\ ?lhs \rangle$ **by** *blast*
next
fix t' **assume** $*$: $\langle t = \text{map}\ (\text{map-event}_{ptick}\ id\ \text{prod.swap})\ t' \rangle$
 $\langle (t', \text{map-event}_{ptick}\ id\ \text{prod.swap} - ' X) \in \mathcal{F}\ P \rangle$
from $*(1)$ **have** $\langle (t, X) = \text{failure-tick-swap}\ (t', \text{map-event}_{ptick}\ id\ \text{prod.swap} - ' X) \rangle$
by (*simp add: failure-tick-swap-def refusal-tick-swap-def trace-tick-swap-def flip: tick-swap-is-map-event_{ptick}*)
with $*(2)$ **show** $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle$ **by** (*simp add: F-TickSwap*)
qed
qed

lemma *TickSwap-TickSwap* [*simp*] : $\langle \text{TickSwap}\ (\text{TickSwap}\ P) = P \rangle$
by (*simp add: Process-eq-spec TickSwap-projs'*)

lemma *TickSwap-comp-TickSwap* [*simp*] : $\langle \text{TickSwap} \circ \text{TickSwap} = \text{id} \rangle$
by (*rule ext*) *simp*

lemma *TickSwap-eq-iff-eq-TickSwap* : $\langle \text{TickSwap } P = Q \longleftrightarrow P = \text{TickSwap } Q \rangle$
by *auto*

lemma *inj-TickSwap* [*simp*] : $\langle \text{inj } \text{TickSwap} \rangle$
by (*metis injI TickSwap-TickSwap*)

lemma *surj-TickSwap* [*simp*] : $\langle \text{surj } \text{TickSwap} \rangle$
by (*metis surjI TickSwap-TickSwap*)

lemma *bij-TickSwap* [*simp*] : $\langle \text{bij } \text{TickSwap} \rangle$
by (*simp add: bij-betw-def*)

lemma *strict-mono-TickSwap* : $\langle \text{strict-mono } \text{TickSwap} \rangle$
by (*rule strict-monoI*)
(metis D-TickSwap F-TickSwap failure-refine-def image-mono injD inj-TickSwap
nless-le failure-divergence-refine-def divergence-refine-def)

Monotonicity Properties

lemma *mono-TickSwap* : $\langle P \sqsubseteq Q \implies \text{TickSwap } P \sqsubseteq \text{TickSwap } Q \rangle$
by (*simp add: TickSwap-is-Renaming mono-Renaming*)

lemma *mono-TickSwap-FD* : $\langle P \sqsubseteq_{FD} Q \implies \text{TickSwap } P \sqsubseteq_{FD} \text{TickSwap } Q \rangle$
and *mono-TickSwap-DT* : $\langle P \sqsubseteq_{DT} Q \implies \text{TickSwap } P \sqsubseteq_{DT} \text{TickSwap } Q \rangle$
and *mono-TickSwap-F* : $\langle P \sqsubseteq_F Q \implies \text{TickSwap } P \sqsubseteq_F \text{TickSwap } Q \rangle$
and *mono-TickSwap-D* : $\langle P \sqsubseteq_D Q \implies \text{TickSwap } P \sqsubseteq_D \text{TickSwap } Q \rangle$
and *mono-TickSwap-T* : $\langle P \sqsubseteq_T Q \implies \text{TickSwap } P \sqsubseteq_T \text{TickSwap } Q \rangle$
by (*simp-all add: TickSwap-projs refine-defs image-mono*)

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* : $\langle \text{TickSwap } P \sqsubseteq \text{TickSwap } Q \longleftrightarrow P \sqsubseteq Q \rangle$
and *FD-TickSwap-iff* : $\langle \text{TickSwap } P \sqsubseteq_{FD} \text{TickSwap } Q \longleftrightarrow P \sqsubseteq_{FD} Q \rangle$
and *DT-TickSwap-iff* : $\langle \text{TickSwap } P \sqsubseteq_{DT} \text{TickSwap } Q \longleftrightarrow P \sqsubseteq_{DT} Q \rangle$
and *F-TickSwap-iff* : $\langle \text{TickSwap } P \sqsubseteq_F \text{TickSwap } Q \longleftrightarrow P \sqsubseteq_F Q \rangle$
and *D-TickSwap-iff* : $\langle \text{TickSwap } P \sqsubseteq_D \text{TickSwap } Q \longleftrightarrow P \sqsubseteq_D Q \rangle$
and *T-TickSwap-iff* : $\langle \text{TickSwap } P \sqsubseteq_T \text{TickSwap } Q \longleftrightarrow P \sqsubseteq_T Q \rangle$
by (*rule iffI; drule monos-TickSwap, simp add: monos-TickSwap*)**+**

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] : $\langle \text{cont } P \implies \text{cont } (\lambda x. \text{TickSwap } (P \ x)) \rangle$
by (simp add: *TickSwap-is-Renaming*)

Algebraic Laws

Constant Processes **lemma** *TickSwap-STOP* [simp] : $\langle \text{TickSwap } \text{STOP} = \text{STOP} \rangle$
by (simp add: *STOP-iff-T T-TickSwap T-STOP*)

lemma *TickSwap-is-STOP-iff* [simp] : $\langle \text{TickSwap } P = \text{STOP} \longleftrightarrow P = \text{STOP} \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-BOT* [simp] : $\langle \text{TickSwap } \perp = \perp \rangle$
by (simp add: *BOT-iff-Nil-D D-TickSwap D-BOT*)

lemma *TickSwap-is-BOT-iff* [simp] : $\langle \text{TickSwap } P = \perp \longleftrightarrow P = \perp \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-SKIP* [simp] : $\langle \text{TickSwap } (\text{SKIP } (r, s)) = \text{SKIP } (s, r) \rangle$
by (simp add: *TickSwap-is-Renaming*)

lemma *TickSwap-is-SKIP-iff* [simp] : $\langle \text{TickSwap } P = \text{SKIP } (r, s) \longleftrightarrow P = \text{SKIP } (s, r) \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-SKIPS* [simp] : $\langle \text{TickSwap } (\text{SKIPS } R\text{-}S) = \text{SKIPS } \{(s, r). (r, s) \in R\text{-}S\} \rangle$
by (auto simp add: *Process-eq-spec TickSwap-projs' SKIPS-projs*)
(auto simp add: *failure-tick-swap-def refusal-tick-swap-def*)

lemma *TickSwap-is-SKIPS-iff* [simp] :
 $\langle \text{TickSwap } P = \text{SKIPS } R\text{-}S \longleftrightarrow P = \text{SKIPS } \{(s, r). (r, s) \in R\text{-}S\} \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

Binary (or less) Operators **lemma** *TickSwap-Ndet* [simp] : $\langle \text{TickSwap } (P \sqcap Q) = \text{TickSwap } P \sqcap \text{TickSwap } Q \rangle$
by (simp add: *Process-eq-spec TickSwap-projs Ndet-projs image-Un*)

lemma *TickSwap-is-Ndet-iff* [simp] : $\langle \text{TickSwap } P = Q \sqcap R \longleftrightarrow P = \text{TickSwap } Q \sqcap \text{TickSwap } R \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Det* [simp] :
 $\langle \text{TickSwap } (P \sqcap Q) = \text{TickSwap } P \sqcap \text{TickSwap } Q \rangle$
by (simp add: *TickSwap-is-Renaming Renaming-Det*)

lemma *TickSwap-is-Det-iff* [simp] : $\langle \text{TickSwap } P = Q \sqcap R \longleftrightarrow P = \text{TickSwap } Q \sqcap \text{TickSwap } R \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Sliding* [simp] : $\langle \text{TickSwap } (P \triangleright Q) = \text{TickSwap } P \triangleright \text{TickSwap } Q \rangle$
by (simp add: *Sliding-def*)

lemma *TickSwap-is-Sliding-iff* [simp] : $\langle \text{TickSwap } P = Q \triangleright R \longleftrightarrow P = \text{TickSwap } Q \triangleright \text{TickSwap } R \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Sync* [simp] :
 $\langle \text{TickSwap } (P \llbracket S \rrbracket Q) = \text{TickSwap } P \llbracket S \rrbracket \text{TickSwap } Q \rangle$
by (simp add: *TickSwap-is-Renaming bij-Renaming-Sync*)

lemma *TickSwap-is-Sync-iff* [simp] :
 $\langle \text{TickSwap } P = Q \llbracket S \rrbracket R \longleftrightarrow P = \text{TickSwap } Q \llbracket S \rrbracket \text{TickSwap } R \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Seq* [simp] :
 $\langle \text{TickSwap } (P ; Q) = \text{TickSwap } P ; \text{TickSwap } Q \rangle$
by (simp add: *Renaming-Seq TickSwap-is-Renaming*)

lemma *TickSwap-is-Seq-iff* [simp] :
 $\langle \text{TickSwap } P = Q ; R \longleftrightarrow P = \text{TickSwap } Q ; \text{TickSwap } R \rangle$
by (simp add: *TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Renaming* [simp] :
 $\langle \text{TickSwap } (\text{Renaming } P f g) =$
 $\text{Renaming } (\text{TickSwap } P) f (\text{prod.swap} \circ g \circ \text{prod.swap}) \rangle$
by (simp add: *TickSwap-is-Renaming flip: Renaming-comp*)
(metis comp-apply swap-swap)

lemma *TickSwap-Renaming'* :
 $\langle \text{TickSwap } (\text{Renaming } P f g) = \text{Renaming } P f (\text{prod.swap} \circ g) \rangle$
by (simp add: *TickSwap-is-Renaming flip: Renaming-comp*)

lemma *TickSwap-is-Renaming-iff* [simp] :
 $\langle \text{TickSwap } P = \text{Renaming } Q f g \longleftrightarrow P = \text{Renaming } (\text{TickSwap } Q) f (\text{prod.swap} \circ g \circ \text{prod.swap}) \rangle$

by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Hiding* [*simp*] : $\langle \text{TickSwap } (P \setminus S) = \text{TickSwap } P \setminus S \rangle$
by (*simp add: TickSwap-is-Renaming bij-Renaming-Hiding*)

lemma *TickSwap-is-Hiding-iff* [*simp*] : $\langle \text{TickSwap } P = Q \setminus S \longleftrightarrow P = \text{TickSwap } Q \setminus S \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Interrupt* [*simp*] :
 $\langle \text{TickSwap } (P \triangle Q) = \text{TickSwap } P \triangle \text{TickSwap } Q \rangle$
by (*simp add: TickSwap-is-Renaming Renaming-Interrupt*)

lemma *TickSwap-is-Interrupt-iff* [*simp*] :
 $\langle \text{TickSwap } P = Q \triangle R \longleftrightarrow P = \text{TickSwap } Q \triangle \text{TickSwap } R \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Throw* [*simp*] :
 $\langle \text{TickSwap } (P \Theta a \in A. Q a) = \text{TickSwap } P \Theta a \in A. \text{TickSwap } (Q a) \rangle$
by (*simp add: TickSwap-is-Renaming inj-on-Renaming-Throw*)
(rule *mono-Throw-eq, metis id-apply inj-on-id inv-into-f-f*)

lemma *TickSwap-is-Throw-iff* [*simp*] :
 $\langle \text{TickSwap } P = Q \Theta a \in A. R a \longleftrightarrow P = \text{TickSwap } Q \Theta a \in A. \text{TickSwap } (R a) \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

Architectural Operators **lemma** *TickSwap-GlobalNdet* [*simp*] :
 $\langle \text{TickSwap } (\sqcap a \in A. P a) = \sqcap a \in A. \text{TickSwap } (P a) \rangle$
by (*simp add: TickSwap-is-Renaming Renaming-distrib-GlobalNdet*)

lemma *TickSwap-is-GlobalNdet-iff* [*simp*] :
 $\langle \text{TickSwap } P = \sqcap a \in A. Q a \longleftrightarrow P = \sqcap a \in A. \text{TickSwap } (Q a) \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-GlobalDet* [*simp*] :
 $\langle \text{TickSwap } (\sqcap a \in A. P a) = \sqcap a \in A. \text{TickSwap } (P a) \rangle$
by (*simp add: TickSwap-is-Renaming Renaming-distrib-GlobalDet*)

lemma *TickSwap-is-GlobalDet-iff* [*simp*] :
 $\langle \text{TickSwap } P = \sqcap a \in A. Q a \longleftrightarrow P = \sqcap a \in A. \text{TickSwap } (Q a) \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-MultiSync* [*simp*] :

$\langle \text{TickSwap } (\llbracket S \rrbracket m \in \# M. P m) = \llbracket S \rrbracket m \in \# M. \text{TickSwap } (P m) \rangle$
by (induct M rule: induct-subset-mset-empty-single) simp-all

lemma *TickSwap-is-TickSwap-MultiSync-iff* [simp] :
 $\langle \text{TickSwap } P = \llbracket S \rrbracket m \in \# M. Q m \longleftrightarrow P = \llbracket S \rrbracket m \in \# M. \text{TickSwap } (Q m) \rangle$
by (simp add: TickSwap-eq-iff-eq-TickSwap)

lemma *TickSwap-MultiSeq* [simp] :
 $\langle L \neq [] \implies \text{TickSwap } (SEQ l \in @ L. P l) = SEQ l \in @ L. \text{TickSwap } (P l) \rangle$
by (induct L rule: rev-induct, simp-all)
 (metis MultiSeq-Nil SKIP-Seq TickSwap-Seq)

lemma *TickSwap-is-MultiSeq-iff* [simp] :
 $\langle L \neq [] \implies \text{TickSwap } P = SEQ l \in @ L. Q l \longleftrightarrow P = SEQ l \in @ L. \text{TickSwap } (Q l) \rangle$
by (metis TickSwap-MultiSeq TickSwap-TickSwap)

Communications lemma *TickSwap-write0* [simp] : $\langle \text{TickSwap } (e \rightarrow P) = e \rightarrow \text{TickSwap } P \rangle$
by (simp add: TickSwap-is-Renaming Renaming-write0)

lemma *TickSwap-is-write0-iff* [simp] : $\langle \text{TickSwap } P = e \rightarrow Q \longleftrightarrow P = e \rightarrow \text{TickSwap } Q \rangle$
by (simp add: TickSwap-eq-iff-eq-TickSwap)

lemma *TickSwap-write* [simp] : $\langle \text{TickSwap } (c!e \rightarrow P) = c!e \rightarrow \text{TickSwap } P \rangle$
by (simp add: TickSwap-is-Renaming Renaming-write)

lemma *TickSwap-is-write-iff* [simp] : $\langle \text{TickSwap } P = c!e \rightarrow Q \longleftrightarrow P = c!e \rightarrow \text{TickSwap } Q \rangle$
by (simp add: TickSwap-eq-iff-eq-TickSwap)

lemma *TickSwap-Mprefix* [simp] :
 $\langle \text{TickSwap } (\Box a \in A \rightarrow P a) = \Box a \in A \rightarrow \text{TickSwap } (P a) \rangle$
by (simp add: Mprefix-GlobalDet)

lemma *TickSwap-is-Mprefix-iff* [simp] :
 $\langle \text{TickSwap } P = (\Box a \in A \rightarrow Q a) \longleftrightarrow P = \Box a \in A \rightarrow \text{TickSwap } (Q a) \rangle$
by (simp add: TickSwap-eq-iff-eq-TickSwap)

lemma *TickSwap-read* [simp] : $\langle \text{TickSwap } (c?a \in A \rightarrow P a) = c?a \in A \rightarrow \text{TickSwap } (P a) \rangle$
by (simp add: read-def comp-def)

lemma *TickSwap-is-read-iff* [simp] :

$\langle \text{TickSwap } P = c?a \in A \rightarrow Q \ a \longleftrightarrow P = c?a \in A \rightarrow \text{TickSwap } (Q \ a) \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-Mndetprefix* [*simp*] :
 $\langle \text{TickSwap } (\sqcap a \in A \rightarrow P \ a) = \sqcap a \in A \rightarrow \text{TickSwap } (P \ a) \rangle$
by (*simp add: Mndetprefix-GlobalNdet*)

lemma *TickSwap-is-Mndetprefix-iff* [*simp*] :
 $\langle \text{TickSwap } P = (\sqcap a \in A \rightarrow Q \ a) \longleftrightarrow P = \sqcap a \in A \rightarrow \text{TickSwap } (Q \ a) \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

lemma *TickSwap-ndet-write* [*simp*] : $\langle \text{TickSwap } (c!!a \in A \rightarrow P \ a) = c!!a \in A \rightarrow \text{TickSwap } (P \ a) \rangle$
by (*simp add: ndet-write-def comp-def*)

lemma *TickSwap-is-ndet-write-iff* [*simp*] :
 $\langle \text{TickSwap } P = c!!a \in A \rightarrow Q \ a \longleftrightarrow P = c!!a \in A \rightarrow \text{TickSwap } (Q \ a) \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

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* :: $\langle [(\text{'a}, \text{'r}) \text{ process}_{ptick}, \text{'a} \Rightarrow \text{'b}] \Rightarrow (\text{'b}, \text{'r}) \text{ process}_{ptick} \rangle$
where $\langle \text{RenamingEv } P \ f \equiv \text{Renaming } P \ f \text{ id} \rangle$

lemma *RenamingEv-id-unfolded* [*iff*] :
 $\langle \text{Renaming } P \ f \ (\lambda r. \ r) = \text{RenamingEv } P \ f \rangle$ **by** (*simp add: id-def*)

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

lemmas *monos-RenamingEv* = *monos-Renaming* [**where** $g = \text{id}$]

lemma *RenamingEv-SKIP* : $\langle \text{RenamingEv } (\text{SKIP } r) \ f = \text{SKIP } r \rangle$ **by** *simp*

lemma *RenamingEv-cont* :
 $\langle \text{cont } P \Longrightarrow \text{finitary } f \Longrightarrow \text{cont } (\lambda x. \text{RenamingEv } (P \ x) \ f) \rangle$ **by** *simp*

lemma *RenamingEv-Seq* :
 $\langle \text{RenamingEv } (P ; Q) f = \text{RenamingEv } P f ; \text{RenamingEv } Q f \rangle$
by (*simp add: Renaming-Seq*)

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* :: $\langle (a, r) \text{ process}_{ptick}, r \Rightarrow s \rangle \Rightarrow (a, s) \text{ process}_{ptick}$

where $\langle \text{RenamingTick } P \ g \equiv \text{Renaming } P \ id \ g \rangle$

lemma *RenamingTick-id-unfolded* [iff] :

$\langle \text{Renaming } P \ (\lambda a. a) \ g = \text{RenamingTick } P \ g \rangle$ **by** (*simp add: id-def*)

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* :

$\langle \text{cont } P \implies \text{finitary } g \implies \text{cont } (\lambda x. \text{RenamingTick } (P \ x) \ g) \rangle$ **by** *simp*

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 $(\backslash)$, see below.

lemma *RenamingTick-Throw* :

$\langle \text{RenamingTick } (P \ \Theta \ a \in A. \ Q \ a) \ g = \text{RenamingTick } P \ g \ \Theta \ a \in A. \text{RenamingTick } (Q \ a) \ g \rangle$

proof (*subst inj-on-Renaming-Throw*)

show $\langle \text{inj-on } id \ (\text{events-of } P \cup A) \rangle$ **by** *simp*

next
show $\langle \text{RenamingTick } P \ g \ \Theta \ b \in id \ ' \ A. \text{RenamingTick } (Q \ (\text{inv-into } A \ id \ b)) \ g =$
 $\text{RenamingTick } P \ g \ \Theta \ a \in A. \text{RenamingTick } (Q \ a) \ g \rangle$
by (*simp*, *rule mono-Throw-eq*)
(metis f-inv-into-f id-apply image-id)
qed

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* :
 $\langle \text{RenamingEv } (\text{RenamingTick } P \ g) \ f = \text{Renaming } P \ f \ g \rangle$
and *RenamingTick-RenamingEv-is-Renaming* :
 $\langle \text{RenamingTick } (\text{RenamingEv } P \ f) \ g = \text{Renaming } P \ f \ g \rangle$
by (*metis Renaming-comp comp-id fun.map-id*) $+$

5.2.3 Properties

lemma *isInfHidden-seqRun-imp-tickFree-seqRun* :
 $\langle \text{isInfHidden-seqRun } x \ P \ A \ t \implies tF \ (\text{seqRun } t \ x \ i) \rangle$
by (*metis event_{ptick}.disc(1) image-iff isInfHidden-seqRun-imp-tickFree tickFree-seqRun-iff*)

lemma *tickFree-map-map-event_{ptick}-is* :
 $\langle tF \ t \implies \text{map } (\text{map-event}_{ptick} \ f \ g) \ t = \text{map } (ev \circ f \circ of-ev) \ t \rangle$
by (*induct t*) (*auto simp add: is-ev-def*)

lemma *RenamingTick-Hiding* :
 $\langle \text{RenamingTick } (P \setminus A) \ g = \text{RenamingTick } P \ g \setminus A \rangle$
(is $\langle ?lhs = ?rhs \rangle$) **for** $P :: \langle ('a, 'r) \text{process}_{ptick} \rangle$

proof –
let $?RT = \langle \lambda P. \text{RenamingTick } P \ g \rangle$
let $?th-A = \langle \lambda t. \text{trace-hide } t \ (ev \ 'A) \rangle$
let $?map = \langle \text{map } (\text{map-event}_{ptick} \ id \ g) \rangle$
have $\$: \langle ?th-A \ (?map \ t) = ?map \ (?th-A \ t) \rangle$ **for** t
by (*induct t*) (*simp-all add: image-iff map-event_{ptick}-eq-ev-iff*)

have $\$ \$: \langle \text{map-event}_{ptick} \text{ id } g - ' X \cup \text{ev } ' A = \text{map-event}_{ptick} \text{ id } g - ' X \cup \text{map-event}_{ptick} \text{ id } g - ' \text{ev } ' A \rangle \text{ for } X$
by (*auto simp add: map-event_{ptick}-eq-ev-iff*)
show $\langle ?lhs = ?rhs \rangle$
proof (*rule Process-eq-optimizedI*)
fix t **assume** $\langle t \in \mathcal{D} \ ?rhs \rangle$
then obtain $t1 \ t2$ **where** $*$: $\langle t = ?map \ t1 \ @ \ t2 \rangle$
 $\langle tF \ t1 \rangle \ \langle ftF \ t2 \rangle \ \langle t1 \in \mathcal{D} \ (P \setminus A) \rangle$ **unfolding** *D-Renaming* **by** *blast*
from $*(4)$ **obtain** $u \ v$ **where** $**$: $\langle ftF \ v \rangle \ \langle tF \ u \rangle \ \langle t1 = ?th-A \ u \ @ \ v \rangle$
 $\langle u \in \mathcal{D} \ P \vee (\exists x. \text{isInfHidden-seqRun-strong } x \ P \ A \ u) \rangle$
by (*blast elim: D-Hiding-seqRunE*)
from $*(4)$ **show** $\langle t \in \mathcal{D} \ ?rhs \rangle$
proof (*elim disjE exE*)
assume $\langle u \in \mathcal{D} \ P \rangle$
with $*(2)$ **have** $\langle ?map \ u \in \mathcal{D} \ (?RT \ P) \rangle$
by (*auto simp add: D-Renaming intro: front-tickFree-Nil*)
thus $\langle t \in \mathcal{D} \ ?rhs \rangle$
by (*simp add: D-Hiding $*(1)$ $*(3)$ flip: $\$$*)
(metis $(2, 3)$ $*(2, 3)$ front-tickFree-append map-event_{ptick}-tickFree tickFree-append-iff)*
next
fix x **assume** $*** : \langle \text{isInfHidden-seqRun-strong } x \ P \ A \ u \rangle$
have $\langle \text{isInfHidden-seqRun } (\text{ev} \circ \text{of-ev} \circ x) \ (?RT \ P) \ A \ (?map \ u) \rangle$
proof (*intro allI conjI*)
fix i
have $\langle \text{seqRun } (?map \ u) \ (\text{ev} \circ \text{of-ev} \circ x) \ i = ?map \ (\text{seqRun } u \ x \ i) \rangle$
by (*simp add: seqRun-def image-iff ev-eq-map-event_{ptick}-iff*)
*(metis $***$ event_{ptick}.sel(1) imageE)*
also have $\langle ?map \ (\text{seqRun } u \ x \ i) \in \mathcal{T} \ (?RT \ P) \rangle$
unfolding *T-Renaming using $***$ Un-iff* **by** *blast*
finally show $\langle \text{seqRun } (?map \ u) \ (\text{ev} \circ \text{of-ev} \circ x) \ i \in \mathcal{T} \ (?RT \ P) \rangle$.
next
show $\langle (\text{ev} \circ \text{of-ev} \circ x) \ i \in \text{ev } ' A \rangle \text{ for } i$
by (*metis $***$ comp-apply event_{ptick}.sel(1) image-iff*)
qed
thus $\langle t \in \mathcal{D} \ ?rhs \rangle$
by (*simp (no-asm) add: D-Hiding-seqRun $*(1)$ $*(3)$ flip: $\$$*)
(metis $(2, 3)$ $*(2, 3)$ front-tickFree-append map-event_{ptick}-tickFree tickFree-append-iff)*
qed
next
fix t **assume** $\langle t \in \mathcal{D} \ ?rhs \rangle$
then obtain $u \ v$ **where** $*$: $\langle ftF \ v \rangle \ \langle tF \ u \rangle \ \langle t = ?th-A \ u \ @ \ v \rangle$
 $\langle u \in \mathcal{D} \ (?RT \ P) \vee (\exists x. \text{isInfHidden-seqRun-strong } x \ (?RT \ P) \ A \ u) \rangle$
by (*blast elim: D-Hiding-seqRunE*)
from $*(4)$ **show** $\langle t \in \mathcal{D} \ ?lhs \rangle$
proof (*elim disjE exE*)
assume $\langle u \in \mathcal{D} \ (?RT \ P) \rangle$
then obtain $u1 \ u2$ **where** $**$: $\langle u = ?map \ u1 \ @ \ u2 \rangle$

```

    < tF u1 > < ftF u2 > < u1 ∈ D P > unfolding D-Renaming by blast
from mem-D-imp-mem-D-Hiding **(4) have < ?th-A u1 ∈ D (P \ A) > .
thus < t ∈ D ?lhs >
    by (simp add: D-Renaming *(3) **(1) $)
      (metis *(1, 2) **(1, 2) Hiding-tickFree
        front-tickFree-append tickFree-append-iff)
next
fix x assume ** : < isInfHidden-seqRun-strong x (?RT P) A u >
hence < ∀ i. ∃ v. seqRun u x i = ?map v ∧ v ∈ T P >
    unfolding Renaming-projs by blast
    then obtain f where *** : < seqRun u x i = ?map (f i) > < f i ∈ T P > for i
by metis
    have < tF (f i) > for i
      by (metis isInfHidden-seqRun-imp-tickFree-seqRun
        ** ***(1) map-eventptick-tickFree)
    hence < ?map (f i) = map (ev ∘ of-ev) (f i) > for i
      by (simp add: tickFree-map-map-eventptick-is)
    from ***(1) [unfolded this]
    have < map (ev ∘ of-ev) (seqRun u x i) =
      (map (ev ∘ of-ev) (map (ev ∘ of-ev) (f i))) :: (< 'a, 'r > traceptick) > for i by
simp
    also have < map (ev ∘ of-ev) (map (ev ∘ of-ev) (f i)) = f i > for i
      using < ∧ i. tF (f i) > [of i]
      by (auto simp add: tickFree-iff-is-map-ev)
    finally have < f i = map (ev ∘ of-ev) (seqRun u x i) > for i by (rule sym)
    hence **** : < f i = seqRun (f 0) (ev ∘ of-ev ∘ x) i > for i
      by (simp add: seqRun-def ***(1))
    have < isInfHidden-seqRun (ev ∘ of-ev ∘ x) P A (f 0) >
    proof (intro allI conjI)
      show < seqRun (f 0) (ev ∘ of-ev ∘ x) i ∈ T P > for i by (metis ***(2) ****)
    next
      show < (ev ∘ of-ev ∘ x) i ∈ ev 'A > for i
        using ** [THEN spec, of i] by auto
    qed
    with < ∧ i. tF (f i) > have < ?th-A (f 0) ∈ D (P \ A) >
      by (simp add: D-Hiding-seqRun)
      (metis append.right-neutral comp-apply front-tickFree-Nil)
    moreover have < ?th-A u = ?map (?th-A (f 0)) >
      by (metis $ ***(1) seqRun-0)
    ultimately show < t ∈ D ?lhs >
      by (simp add: D-Renaming *(3))
      (use *(1) Hiding-tickFree < ∧ i. tF (f i) > in blast)
    qed
next
fix t X assume < (t, X) ∈ F ?lhs > < t ∉ D ?lhs >
then obtain t' where * : < t = ?map t' >
  < (t', map-eventptick id g - 'X) ∈ F (P \ A) >
    unfolding Renaming-projs by blast
from *(2) consider < t' ∈ D (P \ A) >

```



```

| (**) t'' where ⟨t' = ?th-A t'⟩
  ⟨(t'', map-eventptick id g - ' X ∪ ev ' A) ∈ F P⟩
  unfolding F-Hiding D-Hiding by blast
thus ⟨(t, X) ∈ F ?rhs⟩
proof cases
  assume ⟨t' ∈ D (P \ A)⟩
  hence ⟨tF t' ∨ (∃ t'' r. t' = t'' @ [✓(r)] ∧ tF t'')⟩
    by (metis D-imp-front-tickFree front-tickFree-append-iff
      nonTickFree-n-frontTickFree not-Cons-self2)
  with ⟨t' ∈ D (P \ A)⟩ ⟨t ∉ D ?lhs⟩ have False
    by (elim disjE exE conjE, simp-all add: D-Renaming *(1))
      (use front-tickFree-Nil in blast, metis front-tickFree-single is-processT9)
  thus ⟨(t, X) ∈ F ?rhs⟩ ..
next
case **
  from *(2) have ⟨(?map t'', X ∪ ev ' A) ∈ F (?RT P)⟩
    by (auto simp add: F-Renaming $$)
  thus ⟨(t, X) ∈ F ?rhs⟩ by (simp add: F-Hiding *(1) *(1)) (metis $)
qed
next
fix t X assume ⟨(t, X) ∈ F ?rhs⟩ ⟨t ∉ D ?rhs⟩
then obtain t' where * : ⟨t = ?th-A t'⟩ ⟨(t', X ∪ ev ' A) ∈ F (?RT P)⟩
  unfolding F-Hiding D-Hiding by blast
from *(2) consider ⟨t' ∈ D (?RT P)⟩
| (**) t'' where ⟨t' = ?map t''⟩
  ⟨(t'', map-eventptick id g - ' X ∪ map-eventptick id g - ' ev ' A) ∈ F P⟩
  by (auto simp add: Renaming-projs)
thus ⟨(t, X) ∈ F ?lhs⟩
proof cases
  assume ⟨t' ∈ D (?RT P)⟩
  hence ⟨tF t' ∨ (∃ t'' r. t' = t'' @ [✓(r)] ∧ tF t'')⟩
    by (metis D-imp-front-tickFree front-tickFree-append-iff
      nonTickFree-n-frontTickFree not-Cons-self2)
  with ⟨t' ∈ D (?RT P)⟩ ⟨t ∉ D ?rhs⟩ have False
    by (elim disjE exE, auto simp add: D-Hiding-seqRun *(1) image-iff
      intro: front-tickFree-single is-processT9)
  thus ⟨(t, X) ∈ F ?lhs⟩ ..
next
case **
  from *(2) have ⟨(?th-A t'', map-eventptick id g - ' X) ∈ F (P \ A)⟩
    by (auto simp add: F-Hiding $$)
  thus ⟨(t, X) ∈ F ?lhs⟩
    by (auto simp add: F-Renaming *(1) *(1) $)
qed
qed
qed

```

corollary *bij-Renaming-Hiding* :

$\langle \text{Renaming } (P \setminus S) f g = \text{Renaming } P f g \setminus f \text{ ' } S \rangle$ (is $\langle ?lhs = ?rhs \rangle$) if $\langle \text{bij } f \rangle$
 — We already have $\llbracket \text{bij } fa; \text{bij } ga \rrbracket \implies \text{Renaming } (Pa \setminus Sa) fa ga = \text{Renaming } Pa fa ga \setminus fa \text{ ' } Sa$, but the assumption $\text{bij } g$ is actually not necessary.
proof —
 have $\langle ?lhs = \text{RenamingTick } (\text{RenamingEv } (P \setminus S) f) g \rangle$
 by (simp only: *RenamingTick-RenamingEv-is-Renaming*)
 also have $\langle \dots = \text{RenamingTick } (\text{RenamingEv } P f \setminus f \text{ ' } S) g \rangle$
 by (simp only: *bij-RenamingEv-Hiding[OF $\langle \text{bij } f \rangle$]*)
 also have $\langle \dots = \text{RenamingTick } (\text{RenamingEv } P f) g \setminus f \text{ ' } S \rangle$
 by (simp only: *RenamingTick-Hiding*)
 also have $\langle \dots = ?rhs \rangle$
 by (simp only: *RenamingTick-RenamingEv-is-Renaming*)
 finally show $\langle ?lhs = ?rhs \rangle$.
qed

lemma *Renaming-is-restrictable-on-events-of-strict-ticks-of* :
 $\langle \text{Renaming } P f g = \text{Renaming } P f' g' \rangle$
 if *fun-hyps* : $\langle \bigwedge a. a \in \alpha(P) \implies f a = f' a \rangle$
 $\langle \bigwedge r. r \in \text{S}(P) \implies g r = g' r \rangle$
 for $f f' :: \langle 'a \Rightarrow 'b \rangle$ and $g g' :: \langle 'r \Rightarrow 't \rangle$
 — probably also possible to strengthen with *strict-events-of*
proof —
 have $* : \langle \text{Renaming } P f g \sqsubseteq_{FD} \text{Renaming } P f' g' \rangle$
 if *fun-hyps-bis* : $\langle \bigwedge a. a \in \alpha(P) \implies f a = f' a \rangle \langle \bigwedge r. r \in \text{S}(P) \implies g r = g' r \rangle$
 for $f f' :: \langle 'a \Rightarrow 'b \rangle$ and $g g' :: \langle 'r \Rightarrow 't \rangle$
proof —
 have $\$: \langle \text{map } (\text{map-event}_{ptick} f g) u = \text{map } (\text{map-event}_{ptick} f' g') u \rangle$
 if $\langle u \in \mathcal{T} P \rangle$ and $\langle tF u \rangle$ for u
proof —
 from $\langle u \in \mathcal{T} P \rangle$ have $\langle ev a \in \text{set } u \implies a \in \alpha(P) \rangle$ for a
 by (*meson events-of-memI*)
 with $\langle tF u \rangle$ show $\langle \text{map } (\text{map-event}_{ptick} f g) u = \text{map } (\text{map-event}_{ptick} f' g') u \rangle$
 $u \rangle$
 by (*induct u, simp-all*)
 (*metis event_{ptick}.collapse(1) event_{ptick}.simps(9) fun-hyps-bis(1)*)
qed
 have $\langle (\forall t. t \in \mathcal{D} (\text{Renaming } P f' g') \longrightarrow t \in \mathcal{D} (\text{Renaming } P f g)) \wedge$
 $(\forall t X. (t, X) \in \mathcal{F} (\text{Renaming } P f' g') \longrightarrow t \notin \mathcal{D} (\text{Renaming } P f' g') \longrightarrow$
 $(t, X) \in \mathcal{F} (\text{Renaming } P f g)) \rangle$
proof (*intro conjI allI impI*)
 fix t assume $\langle t \in \mathcal{D} (\text{Renaming } P f' g') \rangle$
 then obtain $u1 u2$ where $* : \langle t = \text{map } (\text{map-event}_{ptick} f' g') u1 @ u2 \rangle$
 $\langle tF u1 \rangle \langle tF u2 \rangle \langle u1 \in \mathcal{D} P \rangle$ **unfolding** *D-Renaming* **by** *blast*
 have $\langle \text{map } (\text{map-event}_{ptick} f' g') u1 = \text{map } (\text{map-event}_{ptick} f g) u1 \rangle$
 by (*simp add: $\langle tF u1 \rangle \$*(4) D-T$*)
 with $*$ show $\langle t \in \mathcal{D} (\text{Renaming } P f g) \rangle$
 by (*auto simp add: D-Renaming*)

```

next
fix t X assume  $\langle (t, X) \in \mathcal{F} (\text{Renaming } P f' g') \rangle \langle t \notin \mathcal{D} (\text{Renaming } P f' g') \rangle$ 
then obtain u where  $\ast : \langle t = \text{map} (\text{map-event}_{\text{ptick}} f' g') u \rangle$ 
 $\langle (u, \text{map-event}_{\text{ptick}} f' g' - 'X) \in \mathcal{F} P \rangle$ 
unfolding Renaming-projs by blast
show  $\langle (t, X) \in \mathcal{F} (\text{Renaming } P f g) \rangle$ 
proof (cases  $\langle tF u \rangle$ )
assume  $\langle tF u \rangle$ 
have  $\langle (u, \text{map-event}_{\text{ptick}} f' g' - 'X \cup \{ev\ a \mid a. a \notin \alpha(P)\} \cup$ 
 $\{\checkmark(r) \mid r. r \notin \checkmark s(P)\}) \in \mathcal{F} P \rangle$  (is  $\langle (u, ?Y) \in \mathcal{F} P \rangle$ )
by (intro is-processT5, simp-all add:  $\ast(2)$ )
(meson T-F-spec events-of-memI in-set-conv-decomp,
metis (mono-tags, lifting)  $\ast(1)$  D-Renaming F-imp-front-tickFree T-F-spec
 $\langle t \notin \mathcal{D} (\text{Renaming } P f' g') \rangle$  append-Nil2 append-T-imp-tickFree
is-processT1
is-processT9 list.simps(3) mem-Collect-eq strict-ticks-of-memI)
moreover from fun-hyps-bis
have  $\langle e \in \text{map-event}_{\text{ptick}} f g - 'X \implies e \in ?Y \rangle$  for e
by (cases e) auto
ultimately have  $\langle (u, \text{map-event}_{\text{ptick}} f g - 'X) \in \mathcal{F} P \rangle$ 
by (meson is-processT4 subset-eq)
moreover have  $\langle t = \text{map} (\text{map-event}_{\text{ptick}} f g) u \rangle$ 
by (metis  $\$ \ast$  F-T  $\langle tF u \rangle$ )
ultimately show  $\langle (t, X) \in \mathcal{F} (\text{Renaming } P f g) \rangle$ 
by (auto simp add: F-Renaming)
next
assume  $\langle \neg tF u \rangle$ 
then obtain u' r where  $\langle tF u' \rangle \langle u = u' @ [\checkmark(r)] \rangle$ 
by (metis  $\ast(2)$  F-imp-front-tickFree front-tickFree-append-iff
nonTickFree-n-frontTickFree not-Cons-self2)
from  $\ast(2)$  F-T  $\langle u = u' @ [\checkmark(r)] \rangle$  have  $\langle u' @ [\checkmark(r)] \in \mathcal{T} P \rangle$  by blast
have  $\$ \$ : \langle \text{map} (\text{map-event}_{\text{ptick}} f' g') u' = \text{map} (\text{map-event}_{\text{ptick}} f g) u' \rangle$ 
by (metis  $\$ \ast(2)$  F-T  $\langle tF u' \rangle \langle u = u' @ [\checkmark(r)] \rangle$  is-processT3-TR-append)
have  $\langle \text{map} (\text{map-event}_{\text{ptick}} f' g') u' @ [\checkmark(g' r)] \in \mathcal{T} (\text{Renaming } P f g) \rangle$ 
proof (cases  $\langle r \in \checkmark s(P) \rangle$ )
assume  $\langle r \in \checkmark s(P) \rangle$ 
hence  $\langle g' r = g r \rangle$  by (simp add: fun-hyps-bis(2))
with  $\$ \$$  show  $\langle \text{map} (\text{map-event}_{\text{ptick}} f' g') u' @ [\checkmark(g' r)] \in \mathcal{T} (\text{Renaming } P f g) \rangle$ 
by (simp add: T-Renaming) (use  $\langle u' @ [\checkmark(r)] \in \mathcal{T} P \rangle$  in auto)
next
assume  $\langle r \notin \checkmark s(P) \rangle$ 
hence  $\langle u' \in \mathcal{D} P \rangle$ 
by (metis  $\langle u' @ [\checkmark(r)] \in \mathcal{T} P \rangle$  is-processT9 strict-ticks-of-memI)
have  $\langle \text{map} (\text{map-event}_{\text{ptick}} f g) u' \in \mathcal{D} (\text{Renaming } P f g) \rangle$ 
using D-Renaming F-imp-front-tickFree  $\langle tF u' \rangle$  is-processT1 by blast
with  $\$ \$$  have  $\langle \text{map} (\text{map-event}_{\text{ptick}} f' g') u' \in \mathcal{D} (\text{Renaming } P f g) \rangle$  by
presburger
hence  $\langle \text{map} (\text{map-event}_{\text{ptick}} f' g') u' @ [\checkmark(g' r)] \in \mathcal{D} (\text{Renaming } P f g) \rangle$ 

```

by (simp add: $\langle tF \ u' \rangle$ is-processT7 map-event_{ptick}-tickFree)
 thus $\langle \text{map} (\text{map-event}_{ptick} f' g') \ u' @ [\checkmark(g' r)] \in \mathcal{T} (\text{Renaming } P f g) \rangle$
 by (simp add: D-T)
 qed
 hence $\langle (\text{map} (\text{map-event}_{ptick} f' g') \ u' @ [\checkmark(g' r)]), X \rangle \in \mathcal{F} (\text{Renaming } P f g) \rangle$
 by (simp add: tick-T-F)
 also have $\langle \text{map} (\text{map-event}_{ptick} f' g') \ u' @ [\checkmark(g' r)] = t \rangle$
 by (simp add: $\ast(1) \ \langle u = u' @ [\checkmark(r)] \rangle$)
 finally show $\langle (t, X) \in \mathcal{F} (\text{Renaming } P f g) \rangle$.
 qed
 qed
 thus $\langle \text{Renaming } P f g \sqsubseteq_{FD} \text{Renaming } P f' g' \rangle$
 by (auto simp add: refine-defs intro: is-processT8)
 qed
 show $\langle \text{Renaming } P f g = \text{Renaming } P f' g' \rangle$
 proof (rule FD-antisym)
 show $\langle \text{Renaming } P f g \sqsubseteq_{FD} \text{Renaming } P f' g' \rangle \ \langle \text{Renaming } P f' g' \sqsubseteq_{FD} \text{Renaming } P f g \rangle$
 by (simp-all add: $\ast$ fun-hyps)
 qed
 qed

corollary *Renaming-is-restrictable-on-events-of-ticks-of* :
 $\langle [\bigwedge a. a \in \alpha(P) \implies f a = f' a; \bigwedge r. r \in \checkmark s(P) \implies g r = g' r] \rangle$
 $\implies \text{Renaming } P f g = \text{Renaming } P f' g' \rangle$
 by (rule *Renaming-is-restrictable-on-events-of-strict-ticks-of*)
 (simp-all add: ticks-of-is-strict-ticks-of-or-UNIV)

corollary *RenamingEv-is-restrictable-on-events-of* :
 $\langle (\bigwedge a. a \in \alpha(P) \implies f a = f' a) \implies \text{RenamingEv } P f = \text{RenamingEv } P f' \rangle$
 by (fact *Renaming-is-restrictable-on-events-of-ticks-of*
 [of $P f f' id id$, simplified])

corollary *RenamingTick-is-restrictable-on-strict-ticks-of* :
 $\langle (\bigwedge r. r \in \checkmark s(P) \implies g r = g' r) \implies \text{RenamingTick } P g = \text{RenamingTick } P g' \rangle$
 by (fact *Renaming-is-restrictable-on-events-of-strict-ticks-of*
 [of $P id id g g'$, simplified])

corollary *RenamingTick-is-restrictable-on-ticks-of* :
 $\langle (\bigwedge r. r \in \checkmark s(P) \implies g r = g' r) \implies \text{RenamingTick } P g = \text{RenamingTick } P g' \rangle$
 by (fact *Renaming-is-restrictable-on-events-of-ticks-of*
 [of $P id id g g'$, simplified])

5.3 Renaming and Generalized Synchronization Product

lemma (in $\text{Sync}_{\text{ptick-locale}}$) $\text{inj-on-RenamingTick-Sync}_{\text{ptick}}$:

$\langle \text{RenamingTick } (P \llbracket S \rrbracket_{\checkmark} Q) \ g =$
 $\text{Sync}_{\text{ptick-locale}}.\text{Sync}_{\text{ptick}} \ (\lambda r \ s. \text{case } r \otimes \checkmark s \text{ of } [r-s] \Rightarrow [g \ r-s] \mid \Diamond \Rightarrow \Diamond) \ P \ S$
 $Q \rangle$
 (is $\langle ?lhs = ?rhs \rangle$)
 if $\text{inj-on-g} : \langle \text{inj-on } g \text{ range-tick-join} \rangle$

proof –

let $?map\text{-}evt = \langle \lambda g. \text{map } (\text{map-event}_{\text{ptick}} \text{ id } g) \rangle$
 let $?tick\text{-}join' = \langle \lambda r \ s. \text{case } r \otimes \checkmark s \text{ of } [r-s] \Rightarrow [g \ r-s] \mid \Diamond \Rightarrow \Diamond \rangle$
 interpret $\text{Sync}_{\text{ptick}}' : \text{Sync}_{\text{ptick-locale}} \ ?tick\text{-}join'$
 by (intro interpretable-inj-on-range-tick-join inj-on-g)
 — Thus $\text{Sync}_{\text{ptick}}'.\text{Sync}_{\text{ptick}} \ P \ S \ Q$ is well defined.

have $\text{inj-on-inv-into-g} :$
 $\langle \text{inj-on } (\text{inv-into range-tick-join } g) \ \text{Sync}_{\text{ptick}}'.\text{range-tick-join} \rangle$
 by (rule inj-onI, simp split: option.split-asm)
 (metis (mono-tags, lifting) f-inv-into-f image-eqI mem-Collect-eq)

from $\text{inv-into-f-f inj-on-g}$ have $\text{expanded-tick-join} :$
 $\langle (\otimes \checkmark) = (\lambda r \ s. \text{case } ?tick\text{-}join' \ r \ s \text{ of } \Diamond \Rightarrow \Diamond \mid [r-s] \Rightarrow [\text{inv-into range-tick-join}$
 $g \ r-s]) \rangle$
 by (fastforce split: split: option.split)

show $\langle ?lhs = ?rhs \rangle$

proof (rule Process-eq-optimizedI)

fix t assume $\langle t \in \mathcal{D} \ ?lhs \rangle$

then obtain $u1 \ u2$ where $*$: $\langle t = \text{map } (\text{map-event}_{\text{ptick}} \text{ id } g) \ u1 \ @ \ u2 \rangle$
 $\langle tF \ u1 \rangle \langle ftF \ u2 \rangle \langle u1 \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 unfolding $D\text{-Renaming}$ by blast

from $*(4)$ obtain $v1 \ w1 \ t\text{-}P \ t\text{-}Q$
 where $**$: $\langle u1 = v1 \ @ \ w1 \rangle \langle tF \ v1 \rangle \langle ftF \ w1 \rangle$
 $\langle v1 \ \text{setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$

unfolding $D\text{-Sync}_{\text{ptick}}$ by blast

from $\text{inj-on-map-map-event}_{\text{ptick}}\text{-setinterleaves}_{\text{ptick}}[OF \ \text{inj-on-g} \ ** (4)]$

have $\langle ?map\text{-}evt \ g \ v1 \ \text{setinterleaves}_{\checkmark ?tick\text{-}join'} ((t\text{-}P, t\text{-}Q), S) \rangle$.

moreover from $*(1-3) \ ** (1, 2)$

have $\langle t = ?map\text{-}evt \ g \ v1 \ @ \ (?map\text{-}evt \ g \ w1 \ @ \ u2) \wedge$
 $tF \ (?map\text{-}evt \ g \ v1) \wedge ftF \ (?map\text{-}evt \ g \ w1 \ @ \ u2) \rangle$
 by (simp add: front-tickFree-append-iff map-event_{ptick}-tickFree)

ultimately show $\langle t \in \mathcal{D} \ ?rhs \rangle$
 using $*(5)$ by (simp (no-asm) add: $\text{Sync}_{\text{ptick}}'.D\text{-Sync}_{\text{ptick}}$) blast

next

fix t assume $\langle t \in \mathcal{D} \ ?rhs \rangle$

then obtain $u \ v \ t\text{-}P \ t\text{-}Q$
 where $*$: $\langle t = u \ @ \ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle$
 $\langle u \ \text{setinterleaves}_{\checkmark ?tick\text{-}join'} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$

unfolding $\text{Sync}_{\text{ptick}}'.D\text{-Sync}_{\text{ptick}}$ by blast

from $\langle tF\ u \rangle$ **have** $\langle e \in \text{set } u \implies \text{map-event}_{ptick}\ id\ (g \circ \text{inv-into range-tick-join } g) \ e = e \rangle$ **for** e
by (*cases* e) (*simp-all* *add: tickFree-def disjoint-iff*)
hence $\langle t = ?\text{map-evt } g\ (?\text{map-evt } (\text{inv-into range-tick-join } g)\ u) \ @\ v \rangle$
by (*simp* *add: *(1) flip: map-event_{ptick}-comp*)
(induct u , *simp-all*)
moreover **have** $\langle tF\ (?\text{map-evt } (\text{inv-into range-tick-join } g)\ u) \rangle$
by (*simp* *add: *(2) map-event_{ptick}-tickFree*)
moreover
{
have $\langle ?\text{map-evt } (\text{inv-into range-tick-join } g)\ u =$
 $?\text{map-evt } (\text{inv-into range-tick-join } g)\ u \ @\ [] \rangle$ **by** *simp*
moreover **have** $\langle tF\ ((\text{map } (\text{map-event}_{ptick}\ id\ (\text{inv-into range-tick-join } g)))$
 $u) \rangle$
by (*simp* *add: *(2) map-event_{ptick}-tickFree*)
moreover **have** $\langle tF\ [] \rangle$ **by** *simp*
moreover **from** *Sync_{ptick}'inj-on-map-map-event_{ptick}-setinterleaves_{ptick}*
*[OF inj-on-inv-into-g *(4), folded expanded-tick-join]*
have $\langle ?\text{map-evt } (\text{inv-into range-tick-join } g)\ u$
 $\text{setinterleaves}_{\checkmark \text{tick-join}} ((t-P, t-Q), S) \rangle$.
ultimately **have** $\langle ?\text{map-evt } (\text{inv-into range-tick-join } g)\ u \in \mathcal{D}\ (P\ [S]_{\checkmark}\ Q) \rangle$
unfolding *D-Sync_{ptick}* **using** **(5)* **by** *blast*
}
ultimately **show** $\langle t \in \mathcal{D}\ ?lhs \rangle$
unfolding *D-Renaming* **using** **(3)* **by** *blast*
next
fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle$ $\langle t \notin \mathcal{D}\ ?lhs \rangle$
then **obtain** u
where $*$: $\langle t = ?\text{map-evt } g\ u \rangle$ $\langle (u, \text{map-event}_{ptick}\ id\ g - 'X) \in \mathcal{F}\ (P\ [S]_{\checkmark}\ Q) \rangle$
unfolding *Renaming-projs* **by** *blast*
with $\langle t \notin \mathcal{D}\ ?lhs \rangle$ **obtain** $t-P\ t-Q\ X-P\ X-Q$
where $**$: $\langle (t-P, X-P) \in \mathcal{F}\ P \rangle$ $\langle (t-Q, X-Q) \in \mathcal{F}\ Q \rangle$
 $\langle u\ \text{setinterleaves}_{\checkmark \text{tick-join}} ((t-P, t-Q), S) \rangle$
 $\langle \text{map-event}_{ptick}\ id\ g - 'X \subseteq \text{super-ref-Sync}_{ptick}\ (\otimes_{\checkmark})\ X-P\ S\ X-Q \rangle$
by (*auto* *simp* *add: D-Renaming Sync_{ptick}-projs*)
(metis *append.right-neutral front-tickFree-Nil map-event_{ptick}-front-tickFree*)
have $\langle t\ \text{setinterleaves}_{\checkmark ?\text{tick-join}} ((t-P, t-Q), S) \rangle$
using **(1) ***(3) inj-on-map-map-event_{ptick}-setinterleaves_{ptick} inj-on-g* **by**
blast
moreover **from** *vimage-inj-on-subset-super-ref-Sync_{ptick}-iff* *[OF inj-on-g, THEN*
*iffD1, OF ***(4)]*
have $\langle X \subseteq \text{super-ref-Sync}_{ptick}\ ?\text{tick-join}'\ X-P\ S\ X-Q \rangle$.
ultimately **show** $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$
unfolding *Sync_{ptick}'F-Sync_{ptick}* **using** **(1, 2)* **by** *fast*
next
fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$ $\langle t \notin \mathcal{D}\ ?rhs \rangle$
then **obtain** $t-P\ t-Q\ X-P\ X-Q$
where $*$: $\langle (t-P, X-P) \in \mathcal{F}\ P \rangle$ $\langle (t-Q, X-Q) \in \mathcal{F}\ Q \rangle$

$\langle t \text{ setinterleaves } \checkmark ?\text{tick-join}' ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} ?\text{tick-join}' X-P S X-Q \rangle$
unfolding $\text{Sync}_{ptick}.\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
from $\text{Sync}_{ptick}.\text{setinterleaves}_{ptick}\text{-imp-set-range-tick-join}[OF *(3)]$
have $\langle \{r-s. \checkmark(r-s) \in \text{set } t\} \subseteq \text{Sync}_{ptick}.\text{range-tick-join} \rangle$.
hence $\langle e \in \text{set } t \implies \text{map-event}_{ptick} \text{ id } (g \circ \text{inv-into range-tick-join } g) e = e \rangle$
for e
by (*cases e, auto simp add: subset-iff split: option.split-asm*)
(metis (mono-tags, lifting) inv-into-f-f mem-Collect-eq inj-on-g)
hence $\langle t = ?\text{map-evt } g (?\text{map-evt } (\text{inv-into range-tick-join } g) t) \rangle$
by (*simp add: *(1) flip: map-event_{ptick}-comp*)
(induct t, simp-all)
moreover
{ from $\text{Sync}_{ptick}.\text{inj-on-map-map-event}_{ptick}\text{-setinterleaves}_{ptick}$
 $[OF \text{inj-on-inv-into-g } *(3), \text{folded expanded-tick-join}]$
have $\langle ?\text{map-evt } (\text{inv-into range-tick-join } g) t$
 $\text{setinterleaves } \checkmark ?\text{tick-join}' ((t-P, t-Q), S) \rangle$.
moreover from $\text{vimage-inj-on-subset-super-ref-Sync}_{ptick}\text{-iff}$
 $[OF \text{inj-on-g, THEN iffD2, OF } *(4)]$
have $\langle \text{map-event}_{ptick} \text{ id } g - ' X \subseteq$
 $\text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P S X-Q \rangle$.
ultimately have $\langle (?\text{map-evt } (\text{inv-into range-tick-join } g) t,$
 $\text{map-event}_{ptick} \text{ id } g - ' X) \in \mathcal{F} (P \llbracket S \rrbracket \checkmark Q) \rangle$
by (*auto simp add: F-Sync_{ptick} *(1, 2)*)
}
ultimately show $\langle (t, X) \in \mathcal{F} ?lhs \rangle$ **unfolding** *F-Renaming* **by** *blast*
qed
qed

lemma (**in** $\text{Sync}_{ptick}\text{-locale}$) *inj-RenamingTick-Sync_{ptick}-inj-RenamingTick* :
 $\langle \text{RenamingTick } P g \llbracket S \rrbracket \checkmark \text{RenamingTick } Q h =$
 $\text{Sync}_{ptick}\text{-locale}.\text{Sync}_{ptick} (\lambda r s. g r \otimes \checkmark h s) P S Q \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
if $\langle \text{inj } g \rangle$ **and** $\langle \text{inj } h \rangle$

for $P :: \langle ('a, 'r') \text{process}_{ptick} \rangle$ **and** $Q :: \langle ('a, 's') \text{process}_{ptick} \rangle$
proof –
interpret *tjoin-interpreted* : $\text{Sync}_{ptick}\text{-locale } \langle \lambda r s. g r \otimes \checkmark h s \rangle$
by *unfold-locales (meson injD inj-tick-join <inj g> <inj h>)*
let $?map\text{-evt} = \langle \lambda g. \text{map } (\text{map-event}_{ptick} \text{ id } g) \rangle$
let $?map\text{-ev} = \langle \lambda t. \text{map } \text{ev } (\text{map of-ev } t) \rangle$
let $?RT = \text{RenamingTick}$
have $* : \langle tF t \implies ?map\text{-evt } g t = ?map\text{-ev } t \rangle$ **for** $t :: \langle ('a, 'r') \text{trace}_{ptick} \rangle$
by (*induct t (auto simp add: is-ev-def)*)
have $** : \langle tF t \implies ?map\text{-evt } h t = ?map\text{-ev } t \rangle$ **for** $t :: \langle ('a, 's') \text{trace}_{ptick} \rangle$
by (*induct t (auto simp add: is-ev-def)*)
have $*** : \langle t \text{ setinterleaves } \checkmark (\otimes \checkmark) ((?\text{map-evt } g u, ?\text{map-evt } h v), S)$
 $\longleftrightarrow t \text{ setinterleaves } \checkmark \lambda r s. g r \otimes \checkmark h s ((u, v), S) \rangle$ **for** $t u v$

by (induct $\langle \lambda r s. g r \otimes \checkmark h s, u, S, v \rangle$ arbitrary: $t u v$) (auto split: option.split)
 have **** : $\langle ?map\text{-}evt\ g\ t = ?map\text{-}evt\ g\ t' \longleftrightarrow t = t' \rangle$ for $t\ t' :: \langle ('a, 'r') \rangle$
trace_{ptick}
 by (rule iffI, induct t arbitrary: t' , auto)
 (metis event_{ptick}.inj-map id-apply inj-def $\langle inj\ g \rangle$)
 have ***** : $\langle ?map\text{-}evt\ h\ t = ?map\text{-}evt\ h\ t' \longleftrightarrow t = t' \rangle$ for $t\ t' :: \langle ('a, 's') \rangle$
trace_{ptick}
 by (rule iffI, induct t arbitrary: t' , auto)
 (metis event_{ptick}.inj-map id-apply inj-def $\langle inj\ h \rangle$)
 show $\langle lhs = rhs \rangle$
 proof (rule Process-eq-optimizedI)
 fix t assume $\langle t \in \mathcal{D}\ ?lhs \rangle$
 then obtain $u\ v\ t\text{-}P'\ t\text{-}Q'$ where $\$: \langle t = u @ v \rangle \langle tF\ u \rangle \langle ftF\ v \rangle$
 $\langle u\ setinterleaves_{\checkmark}(\otimes \checkmark) ((t\text{-}P', t\text{-}Q'), S) \rangle$
 $\langle t\text{-}P' \in \mathcal{D}\ (?RT\ P\ g) \wedge t\text{-}Q' \in \mathcal{T}\ (?RT\ Q\ h) \wedge t\text{-}Q' \notin \mathcal{D}\ (?RT\ Q\ h) \vee$
 $t\text{-}P' \in \mathcal{T}\ (?RT\ P\ g) \wedge t\text{-}P' \notin \mathcal{D}\ (?RT\ P\ g) \wedge t\text{-}Q' \in \mathcal{D}\ (?RT\ Q\ h) \vee$
 $t\text{-}P' \in \mathcal{D}\ (?RT\ P\ g) \wedge t\text{-}Q' \in \mathcal{D}\ (?RT\ Q\ h) \rangle$
 unfolding $D\text{-}Sync_{ptick}$ by blast
 from $\$(5)$ show $\langle t \in \mathcal{D}\ ?rhs \rangle$
 proof (elim disjE conjE)
 assume $\langle t\text{-}P' \in \mathcal{D}\ (?RT\ P\ g) \rangle \langle t\text{-}Q' \in \mathcal{T}\ (?RT\ Q\ h) \rangle \langle t\text{-}Q' \notin \mathcal{D}\ (?RT\ Q\ h) \rangle$
 then obtain $t\text{-}P_1\ t\text{-}P_2\ t\text{-}Q$
 where $\$ \$: \langle t\text{-}P' = ?map\text{-}evt\ g\ t\text{-}P_1 @ t\text{-}P_2 \rangle \langle tF\ t\text{-}P_1 \rangle \langle ftF\ t\text{-}P_2 \rangle \langle t\text{-}P_1 \in \mathcal{D}\ P \rangle$
 $\langle t\text{-}Q' = ?map\text{-}evt\ h\ t\text{-}Q \rangle \langle t\text{-}Q \in \mathcal{T}\ Q \rangle$ unfolding *Renaming-projs* by blast
 from $\$(4)[unfolded\ \$\$(1), THEN\ setinterleaves_{ptick}\text{-}appendL]$
 obtain $u_1\ u_2\ t\text{-}Q'_1\ t\text{-}Q'_2$ where $\$ \$ \$: \langle u = u_1 @ u_2 \rangle \langle t\text{-}Q' = t\text{-}Q'_1 @ t\text{-}Q'_2 \rangle$
 $\langle u_1\ setinterleaves_{\checkmark}(\otimes \checkmark) ((?map\text{-}evt\ g\ t\text{-}P_1, t\text{-}Q'_1), S) \rangle$ by blast
 obtain $t\text{-}Q_1\ t\text{-}Q_2$ where $\langle t\text{-}Q = t\text{-}Q_1 @ t\text{-}Q_2 \rangle \langle t\text{-}Q'_1 = ?map\text{-}evt\ h\ t\text{-}Q_1 \rangle$
 by (metis $\$ \$\$(5)\ \$ \$\$(2)\ map\text{-}eq\text{-}append\text{-}conv$)
 from $\$ \$\$(3)[unfolded\ this(2), THEN\ ***[THEN\ iffD1]]$
 have $\langle u_1\ setinterleaves_{\checkmark} \lambda r s. g r \otimes \checkmark h s ((t\text{-}P_1, t\text{-}Q_1), S) \rangle$.
 moreover from $\langle t\text{-}Q = t\text{-}Q_1 @ t\text{-}Q_2 \rangle$ is-processT3-TR-append $\$ \$\$(6)$
 have $\langle t\text{-}Q_1 \in \mathcal{T}\ Q \rangle$ by blast
 ultimately show $\langle t \in \mathcal{D}\ ?rhs \rangle$
 using $\$(1-3)\ \$\$(4)\ \$ \$\$(1)\ front\text{-}tickFree\text{-}append$
 by (auto simp add: tjoin-interpreted. $D\text{-}Sync_{ptick}$)
 next
 assume $\langle t\text{-}Q' \in \mathcal{D}\ (?RT\ Q\ h) \rangle \langle t\text{-}P' \in \mathcal{T}\ (?RT\ P\ g) \rangle \langle t\text{-}P' \notin \mathcal{D}\ (?RT\ P\ g) \rangle$
 then obtain $t\text{-}Q_1\ t\text{-}Q_2\ t\text{-}P$
 where $\$ \$: \langle t\text{-}Q' = ?map\text{-}evt\ h\ t\text{-}Q_1 @ t\text{-}Q_2 \rangle \langle tF\ t\text{-}Q_1 \rangle \langle ftF\ t\text{-}Q_2 \rangle \langle t\text{-}Q_1 \in \mathcal{D}\ Q \rangle$
 $\langle t\text{-}P' = ?map\text{-}evt\ g\ t\text{-}P \rangle \langle t\text{-}P \in \mathcal{T}\ P \rangle$ unfolding *Renaming-projs* by blast
 from $\$(4)[unfolded\ \$\$(1), THEN\ setinterleaves_{ptick}\text{-}appendR]$
 obtain $u_1\ u_2\ t\text{-}P'_1\ t\text{-}P'_2$ where $\$ \$ \$: \langle u = u_1 @ u_2 \rangle \langle t\text{-}P' = t\text{-}P'_1 @ t\text{-}P'_2 \rangle$
 $\langle u_1\ setinterleaves_{\checkmark}(\otimes \checkmark) ((t\text{-}P'_1, ?map\text{-}evt\ h\ t\text{-}Q_1), S) \rangle$ by blast
 obtain $t\text{-}P_1\ t\text{-}P_2$ where $\langle t\text{-}P = t\text{-}P_1 @ t\text{-}P_2 \rangle \langle t\text{-}P'_1 = ?map\text{-}evt\ g\ t\text{-}P_1 \rangle$
 by (metis $\$ \$\$(5)\ \$ \$\$(2)\ map\text{-}eq\text{-}append\text{-}conv$)
 from $\$ \$\$(3)[unfolded\ this(2), THEN\ ***[THEN\ iffD1]]$

have $\langle u_1 \text{ setinterleaves } \checkmark_{\lambda r s. g r} \otimes \checkmark_{h s} ((t-P_1, t-Q_1), S) \rangle$.
 moreover from $\langle t-P = t-P_1 @ t-P_2 \rangle$ is-processT3-TR-append $\$(6)$
 have $\langle t-P_1 \in \mathcal{T} P \rangle$ by blast
 ultimately show $\langle t \in \mathcal{D} ?rhs \rangle$
 using $\$(1-3) \$(4) \$(1)$ front-tickFree-append
 by (auto simp add: tjoin-interpreted.D-Sync_{ptick})
 next
 assume $\langle t-P' \in \mathcal{D} (?RT P g) \rangle \langle t-Q' \in \mathcal{D} (?RT Q h) \rangle$
 then obtain $t-P_1 \ t-P_2 \ t-Q_1 \ t-Q_2$
 where $\$: \langle t-P' = ?map-evt g t-P_1 @ t-P_2 \rangle \langle tF t-P_1 \rangle \langle ftF t-P_2 \rangle \langle t-P_1 \in \mathcal{D} P \rangle$
 $\langle t-Q' = ?map-evt h t-Q_1 @ t-Q_2 \rangle \langle tF t-Q_1 \rangle \langle ftF t-Q_2 \rangle \langle t-Q_1 \in \mathcal{D} Q \rangle$
 unfolding D-Renaming by blast
 from $\$(4)[unfolding \$(1, 5), THEN \text{setinterleaves}_{ptick}\text{-appendL}]$
 obtain $u_1 \ u_2 \ t-Q_1\text{-bis} \ t-Q_2\text{-bis}$
 where $\$: \langle u = u_1 @ u_2 \rangle \langle ?map-evt h t-Q_1 @ t-Q_2 = t-Q_1\text{-bis} @ t-Q_2\text{-bis} \rangle$
 $\langle u_1 \text{ setinterleaves } \checkmark_{(\otimes \checkmark)} ((?map-evt g t-P_1, t-Q_1\text{-bis}), S) \rangle$ by blast
 from $\$(2)$ have $\langle t-Q_1\text{-bis} = ?map-evt h (take (length t-Q_1\text{-bis}) t-Q_1) \vee$
 $t-Q_1\text{-bis} = ?map-evt h t-Q_1 @ take (length t-Q_1\text{-bis} - length t-Q_1) t-Q_2 \rangle$
 by (cases $\langle length t-Q_1\text{-bis} \leq length t-Q_1 \rangle$)
 (simp-all add: append-eq-append-conv-if take-map split: if-split-asm)
 thus $\langle t \in \mathcal{D} ?rhs \rangle$
 proof (elim disjE)
 assume $\langle t-Q_1\text{-bis} = ?map-evt h (take (length t-Q_1\text{-bis}) t-Q_1) \rangle$
 hence $\langle u_1 \text{ setinterleaves } \checkmark_{\lambda r s. g r} \otimes \checkmark_{h s} ((t-P_1, take (length t-Q_1\text{-bis}) t-Q_1), S) \rangle$
 by (metis $\$(3) **$)
 moreover have $\langle take (length t-Q_1\text{-bis}) t-Q_1 \in \mathcal{T} Q \rangle$
 by (metis $\$(8)$ D-T append-take-drop-id is-processT3-TR-append)
 ultimately show $\langle t \in \mathcal{D} ?rhs \rangle$
 using $\$(1-3) \$(4) \$(1)$ front-tickFree-append
 by (auto simp add: tjoin-interpreted.D-Sync_{ptick})
 next
 assume $\langle t-Q_1\text{-bis} = ?map-evt h t-Q_1 @ take (length t-Q_1\text{-bis} - length t-Q_1) t-Q_2 \rangle$
 with $\$(3)$
 have $\langle u_1 \text{ setinterleaves } \checkmark_{(\otimes \checkmark)} ((?map-evt g t-P_1,$
 $?map-evt h t-Q_1 @ take (length t-Q_1\text{-bis} - length t-Q_1) t-Q_2), S) \rangle$
 by simp
 from $\text{setinterleaves}_{ptick}\text{-appendR}[OF this]$ obtain $u_{11} \ u_{12} \ t-P_{11} \ t-P_{12}$
 where $\$: \langle u_1 = u_{11} @ u_{12} \rangle \langle ?map-evt g t-P_1 = t-P_{11} @ t-P_{12} \rangle$
 $\langle u_{11} \text{ setinterleaves } \checkmark_{(\otimes \checkmark)} ((t-P_{11}, ?map-evt h t-Q_1), S) \rangle$ by blast
 have $\langle t-P_{11} = ?map-evt g (take (length t-P_{11}) t-P_1) \rangle$
 by (metis $\$(2)$ append-eq-conv-conj take-map)
 hence $\langle u_{11} \text{ setinterleaves } \checkmark_{\lambda r s. g r} \otimes \checkmark_{h s} ((take (length t-P_{11}) t-P_1, t-Q_1), S) \rangle$
 by (metis $\$(3) **$)
 moreover have $\langle take (length t-P_{11}) t-P_1 \in \mathcal{T} P \rangle$

by (metis $\$(4)$ *D-T append-take-drop-id is-processT3-TR-append*)
 ultimately show $\langle t \in \mathcal{D} \text{ ?rhs} \rangle$
 by (simp add: *tjoin-interpreted.D-Sync_{ptick}*)
 (metis (no-types, lifting) $\$(1, 2, 3)$ $\$(8)$ $\$(1)$ $\$(1)$ $\$(1)$)
 append.assoc front-tickFree-append tickFree-append-iff)
 qed
 qed
 next
 fix t assume $\langle t \in \mathcal{D} \text{ ?rhs} \rangle$
 then obtain $u \ v \ t\text{-}P \ t\text{-}Q$ where $\$: \langle t = u @ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle$
 $\langle u \text{ setinterleaves}_{\lambda r \ s. \ g \ r} \otimes \checkmark \ h \ s \ ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$
 unfolding *tjoin-interpreted.D-Sync_{ptick}* by blast
 from *tickFree-setinterleaves_{ptick}-iff*[*THEN iffD1*, OF $\$(4, 2)$]
 have $\langle tF \ t\text{-}P \rangle \langle tF \ t\text{-}Q \rangle$ by *simp-all*

 with $\$(5)$ have $\langle ?\text{map-evt } g \ t\text{-}P \in \mathcal{D} \ (\text{?RT } P \ g) \wedge ?\text{map-evt } h \ t\text{-}Q \in \mathcal{T} \ (\text{?RT } Q \ h) \vee$
 $?\text{map-evt } g \ t\text{-}P \in \mathcal{T} \ (\text{?RT } P \ g) \wedge ?\text{map-evt } h \ t\text{-}Q \in \mathcal{D} \ (\text{?RT } Q \ h) \rangle$
 by (simp add: *Renaming-projs*) (metis *append.right-neutral front-tickFree-Nil*)
 moreover from $***[\text{THEN iffD2}, \text{OF } \$(4)]$
 have $\langle u \text{ setinterleaves}_{\lambda r \ s. \ g \ r} \otimes \checkmark \ ((?\text{map-evt } g \ t\text{-}P, ?\text{map-evt } h \ t\text{-}Q), S) \rangle$.
 ultimately show $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$
 using $\$(1-3)$ by (auto simp add: *D-Sync_{ptick}*)
 next
 fix $t \ X$ assume $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle \langle t \notin \mathcal{D} \text{ ?lhs} \rangle$
 then obtain $t\text{-}P' \ X\text{-}P' \ t\text{-}Q' \ X\text{-}Q'$
 where $\$: \langle (t\text{-}P', X\text{-}P') \in \mathcal{F} \ (\text{?RT } P \ g) \rangle \langle (t\text{-}Q', X\text{-}Q') \in \mathcal{F} \ (\text{?RT } Q \ h) \rangle$
 $\langle t \text{ setinterleaves}_{\lambda r \ s. \ g \ r} \otimes \checkmark \ ((t\text{-}P', t\text{-}Q'), S) \rangle \langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \otimes \checkmark \ X\text{-}P' \ S \ X\text{-}Q' \rangle$
 unfolding *Sync_{ptick}-projs* by blast
 from $\langle t \notin \mathcal{D} \text{ ?lhs} \rangle \$(1, 2)[\text{THEN F-T}] \(3)
 have $\langle t\text{-}P' \notin \mathcal{D} \ (\text{?RT } P \ g) \wedge t\text{-}Q' \notin \mathcal{D} \ (\text{?RT } Q \ h) \rangle$
 by (simp add: *D-Sync_{ptick}*) (metis *append-self-conv front-tickFree-Nil*)
 with $\$(1, 2)$ obtain $t\text{-}P \ t\text{-}Q$
 where $\$(\$: \langle t\text{-}P' = ?\text{map-evt } g \ t\text{-}P \rangle \langle (t\text{-}P, \text{map-event}_{\text{ptick}} \text{id } g - 'X\text{-}P') \in \mathcal{F} \ P \rangle$
 $\langle t\text{-}Q' = ?\text{map-evt } h \ t\text{-}Q \rangle \langle (t\text{-}Q, \text{map-event}_{\text{ptick}} \text{id } h - 'X\text{-}Q') \in \mathcal{F} \ Q \rangle$
 unfolding *Renaming-projs* by blast
 from $\$(3)[\text{unfolded } \$(1, 3), \text{THEN } ***[\text{THEN iffD1}]]$
 have $\langle t \text{ setinterleaves}_{\lambda r \ s. \ g \ r} \otimes \checkmark \ h \ s \ ((t\text{-}P, t\text{-}Q), S) \rangle$.
 moreover from $\$(4)$ *inj-tick-join*
 have $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\lambda r \ s. \ g \ r \otimes \checkmark \ h \ s)$
 $(\text{map-event}_{\text{ptick}} \text{id } g - 'X\text{-}P') \ S \ (\text{map-event}_{\text{ptick}} \text{id } h - 'X\text{-}Q') \rangle$
 by (simp add: *super-ref-Sync_{ptick}-def*, *safe*) blast
 ultimately show $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$
 using $\$(2, 4)$ by (auto simp add: *tjoin-interpreted.F-Sync_{ptick}*)
 next
 fix $t \ X$ assume $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle \langle t \notin \mathcal{D} \text{ ?rhs} \rangle$

then obtain $t-P \ t-Q \ X-P \ X-Q$
 where $\$: \langle (t-P, X-P) \in \mathcal{F} \ P \rangle \langle (t-Q, X-Q) \in \mathcal{F} \ Q \rangle$
 $\langle t \text{ setinterleaves } \checkmark_{\lambda r \ s. \ g \ r \otimes \checkmark \ h \ s} ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} (\lambda r \ s. \ g \ r \otimes \checkmark \ h \ s) \ X-P \ S \ X-Q \rangle$
 unfolding $tjoin\text{-interpreted.Sync}_{ptick}\text{-projs}$ by *blast*
 from $\langle t \notin \mathcal{D} \ ?rhs \rangle$ have $\langle t-P \notin \mathcal{D} \ P \wedge t-Q \notin \mathcal{D} \ Q \rangle$
 by (*simp add: tjoin-interpreted.D-Sync_{ptick}*)
 (*metis \$(1-3) \ F-T \ append.right-neutral \ front-tickFree-Nil*)
 hence $\$ \$: \langle t-P @ [\checkmark(r)] \in \mathcal{T} \ P \implies r \in \checkmark s(P) \rangle$
 $\langle t-Q @ [\checkmark(s)] \in \mathcal{T} \ Q \implies s \in \checkmark s(Q) \rangle$ for $r \ s$
 by (*meson is-processT9 strict-ticks-of-memI*) +
 have $\$ \$ \$: \langle ?map\text{-evt } g \ t-P @ [\checkmark(g-r)] \in \mathcal{T} \ (?RT \ P \ g) \longleftrightarrow (\exists r. \ g-r = g \ r \wedge$
 $t-P @ [\checkmark(r)] \in \mathcal{T} \ P) \rangle$ for $g-r$
 proof (*rule iffI*)
 from $\langle t-P \notin \mathcal{D} \ P \wedge t-Q \notin \mathcal{D} \ Q \rangle$ have $\langle ?map\text{-evt } g \ t-P \notin \mathcal{D} \ (?RT \ P \ g) \rangle$
 by (*simp add: D-Renaming map-eq-append-conv ******)
 (*use is-processT7 map-event_{ptick}-front-tickFree in blast*)
 hence $\langle ?map\text{-evt } g \ t-P @ [\checkmark(g-r)] \notin \mathcal{D} \ (?RT \ P \ g) \rangle$ by (*meson is-processT9*)
 moreover assume $\langle ?map\text{-evt } g \ t-P @ [\checkmark(g-r)] \in \mathcal{T} \ (?RT \ P \ g) \rangle$
 ultimately show $\langle ?map\text{-evt } g \ t-P @ [\checkmark(g-r)] \in \mathcal{T} \ (?RT \ P \ g) \implies \exists r. \ g-r =$
 $g \ r \wedge t-P @ [\checkmark(r)] \in \mathcal{T} \ P \rangle$
 by (*auto simp add: Renaming-projs append-eq-map-conv tick-eq-map-event_{ptick}-iff ******)
 next
 show $\langle \exists r. \ g-r = g \ r \wedge t-P @ [\checkmark(r)] \in \mathcal{T} \ P \implies ?map\text{-evt } g \ t-P @ [\checkmark(g-r)]$
 $\in \mathcal{T} \ (?RT \ P \ g) \rangle$
 by (*auto simp add: T-Renaming*)
 qed
 have $\$ \$ \$ \$: \langle ?map\text{-evt } h \ t-Q @ [\checkmark(h-s)] \in \mathcal{T} \ (?RT \ Q \ h) \longleftrightarrow (\exists s. \ h-s = h \ s \wedge$
 $t-Q @ [\checkmark(s)] \in \mathcal{T} \ Q) \rangle$ for $h-s$
 proof (*rule iffI*)
 from $\langle t-P \notin \mathcal{D} \ P \wedge t-Q \notin \mathcal{D} \ Q \rangle$ have $\langle ?map\text{-evt } h \ t-Q \notin \mathcal{D} \ (?RT \ Q \ h) \rangle$
 by (*simp add: D-Renaming map-eq-append-conv ******)
 (*use is-processT7 map-event_{ptick}-front-tickFree in blast*)
 hence $\langle ?map\text{-evt } h \ t-Q @ [\checkmark(h-s)] \notin \mathcal{D} \ (?RT \ Q \ h) \rangle$ by (*meson is-processT9*)
 moreover assume $\langle ?map\text{-evt } h \ t-Q @ [\checkmark(h-s)] \in \mathcal{T} \ (?RT \ Q \ h) \rangle$
 ultimately show $\langle ?map\text{-evt } h \ t-Q @ [\checkmark(h-s)] \in \mathcal{T} \ (?RT \ Q \ h) \implies \exists s. \ h-s =$
 $h \ s \wedge t-Q @ [\checkmark(s)] \in \mathcal{T} \ Q \rangle$
 by (*auto simp add: Renaming-projs append-eq-map-conv tick-eq-map-event_{ptick}-iff ******)
 next
 show $\langle \exists s. \ h-s = h \ s \wedge t-Q @ [\checkmark(s)] \in \mathcal{T} \ Q \implies ?map\text{-evt } h \ t-Q @ [\checkmark(h-s)]$
 $\in \mathcal{T} \ (?RT \ Q \ h) \rangle$
 by (*auto simp add: T-Renaming*)
 qed
 define $X-P'$ where $\langle X-P' \equiv \text{map-event}_{ptick} \text{id } g \text{ ' } X-P \cup \{\checkmark(g-r) \mid g-r. \ ?map\text{-evt}$
 $g \ t-P @ [\checkmark(g-r)] \notin \mathcal{T} \ (?RT \ P \ g)\} \rangle$
 define $X-Q'$ where $\langle X-Q' \equiv \text{map-event}_{ptick} \text{id } h \text{ ' } X-Q \cup \{\checkmark(h-s) \mid h-s. \ ?map\text{-evt}$

$h \ t\text{-}Q \ @ \ [\checkmark(h\text{-}s)] \notin \mathcal{T} \ (?\text{RT } Q \ h)\rangle$

have $\langle \text{map-event}_{ptick} \ id \ g \ -' \ (\text{map-event}_{ptick} \ id \ g \ ' \ X\text{-}P) = X\text{-}P \rangle$
 $\langle \text{map-event}_{ptick} \ id \ h \ -' \ (\text{map-event}_{ptick} \ id \ h \ ' \ X\text{-}Q) = X\text{-}Q \rangle$
by (*simp-all add: set-eq-iff image-iff*)
 $(metis \ event_{ptick}.inj\text{-}map \ injD \ inj\text{-}on\text{-}id \ \langle inj \ g \rangle,$
 $metis \ event_{ptick}.inj\text{-}map \ injD \ inj\text{-}on\text{-}id \ \langle inj \ h \rangle)$
with $\$(1, 2)$ **have** $\langle (?map\text{-}evt \ g \ t\text{-}P, \ \text{map-event}_{ptick} \ id \ g \ ' \ X\text{-}P) \in \mathcal{F} \ (?\text{RT } P$
 $g) \rangle$
 $\langle (?map\text{-}evt \ h \ t\text{-}Q, \ \text{map-event}_{ptick} \ id \ h \ ' \ X\text{-}Q) \in \mathcal{F} \ (?\text{RT } Q \ h) \rangle$
by (*auto simp add: F-Renaming*)
hence $\langle (?map\text{-}evt \ g \ t\text{-}P, \ X\text{-}P') \in \mathcal{F} \ (?\text{RT } P \ g) \rangle$
 $\langle (?map\text{-}evt \ h \ t\text{-}Q, \ X\text{-}Q') \in \mathcal{F} \ (?\text{RT } Q \ h) \rangle$
by (*auto simp add: X-P'-def X-Q'-def intro: is-processT5 F-T*)
moreover have $\langle t \ \text{setinterleaves} \ \checkmark(\otimes \checkmark) \ ((?map\text{-}evt \ g \ t\text{-}P, \ ?map\text{-}evt \ h \ t\text{-}Q), \ S) \rangle$
by (*simp add: $\$(3)$ ****)
moreover have $\langle e \in X \implies e \in \text{super-ref-Sync}_{ptick} \ (\otimes \checkmark) \ X\text{-}P' \ S \ X\text{-}Q' \rangle$ **for** e
using $\$(4)[\text{THEN set-mp, of } e]$
by (*cases e,*
 $\text{simp-all add: X-P'-def X-Q'-def super-ref-Sync}_{ptick}\text{-def image-iff}$
 $\text{ev-eq-map-event}_{ptick}\text{-iff tick-eq-map-event}_{ptick}\text{-iff} \ \$\$ \$\$ \$\$ \$\$)$
 metis
ultimately show $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$ **by** (*simp add: F-Sync_{ptick} blast*)
qed
qed

Chapter 6

Commutativity and Associativity of Synchronization

6.1 Commutativity

6.1.1 Motivation

The classical synchronization product is commutative: $P \llbracket A \rrbracket Q = Q \llbracket A \rrbracket P$ but in our generalization such a law cannot be obtained in all generality. Imagine for example that the $(\otimes \checkmark)$ parameter is actually $\lambda r s. [(r, s)]$: we easily figure out that in this case the corresponding law should be something like $P \llbracket A \rrbracket_{\checkmark Pair} Q = TickSwap (Q \llbracket A \rrbracket_{\checkmark Pair} P)$. More generally, in the **locale**, when writing $P \llbracket A \rrbracket_{\checkmark} Q$, P is of type $(\text{'a'}, \text{'r'}) process_{ptick}$ while Q is of type $(\text{'a'}, \text{'s'}) process_{ptick}$ 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 Syncptick-comm-locale =
  Syncptick-locale  $\langle (\otimes \checkmark) \rangle$  for tick-join ::  $\langle \text{'r'} \Rightarrow \text{'s'} \Rightarrow \text{'t'} option \rangle$  (infixl  $\langle \otimes \checkmark \rangle$  100)
+
fixes tick-join-rev      ::  $\langle \text{'s'} \Rightarrow \text{'r'} \Rightarrow \text{'u'} option \rangle$  (infixl  $\langle \otimes \checkmark_{rev} \rangle$  100)
  and tick-join-conv      ::  $\langle \text{'t'} \Rightarrow \text{'u'} \rangle$  ( $\langle \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \rangle$ )
  and tick-join-rev-conv ::  $\langle \text{'u'} \Rightarrow \text{'t'} \rangle$  ( $\langle \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark \rangle$ )
assumes tick-join-None-iff :
   $\langle r \otimes \checkmark s = \Diamond \longleftrightarrow s \otimes \checkmark_{rev} r = \Diamond \rangle$ 
and tick-join-Some-imp :
   $\langle r \otimes \checkmark s = [r-s] \Longrightarrow s \otimes \checkmark_{rev} r = [\otimes \checkmark \Rightarrow \otimes \checkmark_{rev} r-s] \rangle$ 
and tick-join-rev-Some-imp :
   $\langle s \otimes \checkmark_{rev} r = [s-r] \Longrightarrow r \otimes \checkmark s = [\otimes \checkmark_{rev} \Rightarrow \otimes \checkmark s-r] \rangle$ 
begin

```

There is an obvious symmetry over the variables.

sublocale *Sync_{ptick}-comm-locale-sym* :
Sync_{ptick}-comm-locale $\langle (\otimes \checkmark_{rev}) \rangle \langle (\otimes \checkmark) \rangle \langle \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark \rangle \langle \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \rangle$
proof *unfold-locale*
 show $\langle s \otimes \checkmark_{rev} r = \lfloor s-r \rfloor \Rightarrow s' \otimes \checkmark_{rev} r' = \lfloor s-r \rfloor$
 $\Rightarrow s' = s \wedge r' = r \rangle$ **for** $s \ r \ s-r \ s' \ r'$
 using *inj-tick-join tick-join-rev-Some-imp* **by** *blast*
next
 show $\langle s \otimes \checkmark_{rev} r = \Diamond \longleftrightarrow r \otimes \checkmark s = \Diamond \rangle$ **for** $s \ r$
 by (*simp add: tick-join-None-iff*)
next
 show $\langle s \otimes \checkmark_{rev} r = \lfloor s-r \rfloor \Rightarrow r \otimes \checkmark s = \lfloor \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark \ s-r \rfloor \rangle$ **for** $s \ r \ s-r$
 by (*simp add: tick-join-rev-Some-imp*)
next
 show $\langle r \otimes \checkmark s = \lfloor r-s \rfloor \Rightarrow s \otimes \checkmark_{rev} r = \lfloor \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ r-s \rfloor \rangle$ **for** $r \ s \ r-s$
 by (*simp add: tick-join-Some-imp*)
qed

notation *Sync_{ptick}-comm-locale-sym.Sync_{ptick}* $\langle \langle - \rfloor \checkmark_{rev} - \rangle \ [70, 0, 71] \ 70)$
notation *Sync_{ptick}-comm-locale-sym.Inter_{ptick}* $\langle \langle - \parallel \checkmark_{rev} - \rangle \ [72, 73] \ 72)$
notation *Sync_{ptick}-comm-locale-sym.Par_{ptick}* $\langle \langle - \parallel \checkmark_{rev} - \rangle \ [74, 75] \ 74)$

6.1.3 First Properties

lemma *tick-join-conv-image-range-tick-join* :
 $\langle \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ ' \ range-tick-join = Sync_{ptick}\text{-comm-locale-sym.range-tick-join} \rangle$
by (*simp add: set-eq-iff flip: setcompr-eq-image*)
 (*metis option.inject tick-join-Some-imp tick-join-rev-Some-imp*)

lemma *tick-join-rev-conv-comp-tick-join-conv* [*simp*] :
 $\langle r-s \in range-tick-join \Rightarrow \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark (\otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ r-s) = r-s \rangle$
using *tick-join-Some-imp tick-join-rev-Some-imp* **by** *fastforce*

lemma *inj-on-tick-join-conv* : $\langle inj-on \ \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ range-tick-join \rangle$
by (*rule inj-onI, simp*)
 (*metis option.inject tick-join-Some-imp tick-join-rev-Some-imp*)

lemma *bij-betw-tick-join-conv* :
 $\langle bij-betw \ \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ range-tick-join \ Sync_{ptick}\text{-comm-locale-sym.range-tick-join} \rangle$
proof (*rule bij-betw-imageI*)
 show $\langle inj-on \ \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ range-tick-join \rangle$
 by (*fact inj-on-tick-join-conv*)
next
 show $\langle \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} \ ' \ range-tick-join = Sync_{ptick}\text{-comm-locale-sym.range-tick-join} \rangle$
 using *tick-join-conv-image-range-tick-join* **by** *blast*
qed

lemma *map-tick-join-rev-conv-map-tick-join-conv* :
 $\langle \{r-s. \checkmark(r-s) \in \text{set } t\} \subseteq \text{range-tick-join} \implies$
 $\text{map} (\text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark_{\text{rev}} \Rightarrow \otimes \checkmark) (\text{map} (\text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark \Rightarrow \otimes \checkmark_{\text{rev}}) t)$
 $= t \rangle$
proof (*induct t*)
case *Nil* **show** *?case* **by** *simp*
next
let *?f1* = $\langle \text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark \Rightarrow \otimes \checkmark_{\text{rev}} \rangle$
let *?f2* = $\langle \text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark_{\text{rev}} \Rightarrow \otimes \checkmark \rangle$
case (*Cons e t*)
have $\langle \text{map } ?f2 (\text{map } ?f1 (e \# t)) = ?f2 (?f1 e) \# \text{map } ?f2 (\text{map } ?f1 t) \rangle$ **by** *simp*
also have $\langle ?f2 (?f1 e) = e \rangle$
proof (*cases e*)
show $\langle e = \text{ev } a \implies ?f2 (?f1 e) = e \rangle$ **for** *a* **by** *simp*
next
fix *r-s* **assume** $\langle e = \checkmark(r-s) \rangle$
with *Cons.prem*s **have** $\langle r-s \in \text{range-tick-join} \rangle$ **by** *auto*
with $\langle e = \checkmark(r-s) \rangle$ *inj-on-tick-join-conv*
show $\langle ?f2 (?f1 e) = e \rangle$ **by** *simp*
qed
also have $\langle \text{map } ?f2 (\text{map } ?f1 t) = t \rangle$
by (*rule Cons.hyps*) (*use Cons.prem*s **in** *auto*)
finally show $\langle \text{map } ?f2 (\text{map } ?f1 (e \# t)) = e \# t \rangle$.
qed
end

6.1.4 Commutativity

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

lemma *setinterleaves_{ptick}-imp-setinterleaves_{ptick}-rev* :

$\langle t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((u, v), A) \implies$
 $\text{map} (\text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark \Rightarrow \otimes \checkmark_{\text{rev}}) t$
 $\text{setinterleaves}_{\checkmark(\otimes \checkmark_{\text{rev}})} ((v, u), A) \rangle$
 — Finally not used, and probably obtainable as a corollary of $t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)}$
 $((u, v), A) \implies \text{map} (\text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark \Rightarrow \otimes \checkmark_{\text{rev}}) t \text{ setinterleaves}_{\checkmark \lambda r s. \text{case } r \otimes \checkmark s \text{ of } \Diamond \Rightarrow \Diamond \mid [r-s] \Rightarrow [\otimes \checkmark \Rightarrow \otimes \checkmark]}$
 $((u, v), A)$
proof (*induct* $\langle ((\otimes \checkmark), u, A, v) \rangle$ *arbitrary: t u v*)
case (*tick-setinterleaving_{ptick}-tick* *r u s v*)
from *tick-setinterleaving_{ptick}-tick.prem*s
obtain *r-s t'*
where $*$: $\langle r \otimes \checkmark s = [r-s] \rangle \langle t = \checkmark(r-s) \# t' \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((u, v), A) \rangle$
by (*auto split: option.split-asm*)
from *tick-setinterleaving_{ptick}-tick.hyps*[*OF* $*(1)$, *OF* $*(3)$]
have $\langle \text{map} (\text{map-event}_{\text{ptick}} \text{id} \otimes \checkmark \Rightarrow \otimes \checkmark_{\text{rev}}) t' \text{ setinterleaves}_{\checkmark(\otimes \checkmark_{\text{rev}})} ((v, u), A) \rangle$.

moreover from *tick-join-Some-imp*[*OF* $\ast(1)$]
have $\langle s \otimes \checkmark_{rev} r = \lfloor \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} r-s \rfloor \rangle$.
ultimately show $\langle \text{map} (\text{map-event}_{ptick} \text{id } \otimes \checkmark \Rightarrow \otimes \checkmark_{rev}) t$
 $\text{setinterleaves}_{\checkmark(\otimes \checkmark_{rev})} ((\checkmark(s) \# v, \checkmark(r) \# u), A) \rangle$
by (*simp add: $\ast(1, 2)$*)
qed *auto*

lemma *vimage-tick-join-rev-conv-subset-super-ref-Sync_{ptick}-iff* :
 $\langle \text{map-event}_{ptick} \text{id } \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark - 'X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark_{rev}) X-Q A X-P$
 $\longleftrightarrow X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P A X-Q \rangle$
(is $\langle ?lhs1 \subseteq ?lhs2 \longleftrightarrow X \subseteq ?rhs \rangle$)
— Same: finally not used, and probably obtainable as a corollary of (*map-event_{ptick}*
id $\otimes \checkmark_{rev} \Rightarrow \otimes \checkmark - ' ?X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark_{rev}) ?X-P ?A ?X-Q = (?X \subseteq$
 $\text{super-ref-Sync}_{ptick} (\lambda r s. \text{case } r \otimes \checkmark_{rev} s \text{ of } \Diamond \Rightarrow \Diamond \mid \lfloor r-s \rfloor \Rightarrow \lfloor \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark r-s \rfloor)$
 $?X-P ?A ?X-Q)$.
proof —
have $\ast : \langle (\lambda r s. \text{case } r \otimes \checkmark_{rev} s \text{ of } \Diamond \Rightarrow \Diamond \mid \lfloor r-s \rfloor \Rightarrow \lfloor \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark r-s \rfloor) =$
 $(\lambda s r. r \otimes \checkmark s) \rangle$
by (*intro ext, simp split: option.split*)
(metis tick-join-None-iff tick-join-rev-Some-imp)
show *?thesis*
proof (*subst Sync_{ptick}-comm-locale-sym.vimage-inj-on-subset-super-ref-Sync_{ptick}-iff*)
show $\langle \text{inj-on } \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark \text{ Sync}_{ptick}\text{-comm-locale-sym.range-tick-join} \rangle$
by (*fact Sync_{ptick}-comm-locale-sym.inj-on-tick-join-conv*)
next
show $\langle X \subseteq \text{super-ref-Sync}_{ptick}$
 $(\lambda r s. \text{case } r \otimes \checkmark_{rev} s \text{ of } \Diamond \Rightarrow \Diamond \mid \lfloor r-s \rfloor \Rightarrow \lfloor \otimes \checkmark_{rev} \Rightarrow \otimes \checkmark r-s \rfloor) X-Q A$
 $X-P$
 $\longleftrightarrow X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X-P A X-Q \rangle$
using *super-ref-Sync_{ptick}-sym* **by** (*simp add: $\ast$*) *blast*
qed
qed

In the end, the proof is quite simple: mainly a corollary of *inj-on g range-tick-join*
 $\Rightarrow \text{RenamingTick } (P \llbracket S \rrbracket_{\checkmark} Q) g = \text{Sync}_{ptick}\text{-locale.Sync}_{ptick} (\lambda r s. \text{case } r$
 $\otimes \checkmark s \text{ of } \Diamond \Rightarrow \Diamond \mid \lfloor r-s \rfloor \Rightarrow \lfloor g r-s \rfloor) P S Q$.

theorem *Sync_{ptick}-commute* :
 $\langle \text{RenamingTick } (P \llbracket A \rrbracket_{\checkmark} Q) \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} = Q \llbracket A \rrbracket_{\checkmark_{rev}} P \rangle$
proof —
from *inj-on-RenamingTick-Sync_{ptick}*[*OF inj-on-tick-join-conv*]
have $\langle \text{RenamingTick } (P \llbracket A \rrbracket_{\checkmark} Q) \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} =$
 $\text{Sync}_{ptick}\text{-locale.Sync}_{ptick}$
 $(\lambda r s. \text{case } r \otimes \checkmark s \text{ of } \Diamond \Rightarrow \Diamond \mid \lfloor r-s \rfloor \Rightarrow \lfloor \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} r-s \rfloor) P A Q \rangle$
(is $\langle - = \text{Sync}_{ptick}\text{-locale.Sync}_{ptick} ?tick\text{-join}' P A Q \rangle$) .
also have $\langle ?tick\text{-join}' = (\lambda r s. s \otimes \checkmark_{rev} r) \rangle$
by (*intro ext*)
(simp add: Sync_{ptick}-comm-locale-sym.tick-join-rev-Some-imp
tick-join-None-iff split: option.split)
finally show $\langle \text{RenamingTick } (P \llbracket A \rrbracket_{\checkmark} Q) \otimes \checkmark \Rightarrow \otimes \checkmark_{rev} = Q \llbracket A \rrbracket_{\checkmark_{rev}} P \rangle$

by (*metis Sync_{ptick}-comm-locale-sym.Sync_{ptick}-sym*)
qed

end

6.2 Associativity

6.2.1 Motivation

The classical synchronization product is associative: $P \llbracket A \rrbracket (Q \llbracket A \rrbracket R) = P \llbracket A \rrbracket Q \llbracket A \rrbracket R$ 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 Syncptick-assoc-locale =
  Syncptick1 : Syncptick-locale  $\langle (\otimes \checkmark 1) \rangle$  +
  Syncptick2 : Syncptick-locale  $\langle (\otimes \checkmark 2) \rangle$  +
  Syncptick3 : Syncptick-locale  $\langle (\otimes \checkmark 3) \rangle$  +
  Syncptick4 : Syncptick-locale  $\langle (\otimes \checkmark 4) \rangle$ 
  for tick-join1 ::  $\langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$  (infixl  $\langle \otimes \checkmark 1 \rangle$  100)
    and tick-join2 ::  $\langle 't \Rightarrow 'u \Rightarrow 'v \text{ option} \rangle$  (infixl  $\langle \otimes \checkmark 2 \rangle$  100)
    and tick-join3 ::  $\langle 'r \Rightarrow 'w \Rightarrow 'x \text{ option} \rangle$  (infixl  $\langle \otimes \checkmark 3 \rangle$  100)
    and tick-join4 ::  $\langle 's \Rightarrow 'u \Rightarrow 'w \text{ option} \rangle$  (infixl  $\langle \otimes \checkmark 4 \rangle$  100) +
  fixes tick-assoc-ren ::  $\langle 'v \Rightarrow 'x \rangle$  ( $\langle \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 \rangle$ )
    and tick-assoc-ren-conv ::  $\langle 'x \Rightarrow 'v \rangle$  ( $\langle \otimes \checkmark 3 \Rightarrow \otimes \checkmark 2 \rangle$ )
  assumes None-assms-tick-join :
     $\langle r \otimes \checkmark 1 s = \Diamond \implies s \otimes \checkmark 4 u = \Diamond \vee r \otimes \checkmark 3 [s \otimes \checkmark 4 u] = \Diamond \rangle$ 
     $\langle r \otimes \checkmark 1 s \neq \Diamond \implies [r \otimes \checkmark 1 s] \otimes \checkmark 2 u = \Diamond \implies s \otimes \checkmark 4 u = \Diamond \vee r \otimes \checkmark 3 [s \otimes \checkmark 4 u] = \Diamond \rangle$ 
     $\langle s \otimes \checkmark 4 u = \Diamond \implies r \otimes \checkmark 1 s = \Diamond \vee [r \otimes \checkmark 1 s] \otimes \checkmark 2 u = \Diamond \rangle$ 
     $\langle s \otimes \checkmark 4 u \neq \Diamond \implies r \otimes \checkmark 3 [s \otimes \checkmark 4 u] = \Diamond \implies r \otimes \checkmark 1 s = \Diamond \vee [r \otimes \checkmark 1 s] \otimes \checkmark 2 u = \Diamond \rangle$ 
  and tick-assoc-ren-hyp :
     $\langle r \otimes \checkmark 1 s = [t] \implies t \otimes \checkmark 2 u = [v] \implies [r \otimes \checkmark 3 [s \otimes \checkmark 4 u]] = \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 v \rangle$ 
  and tick-assoc-ren-conv-hyp :
     $\langle s \otimes \checkmark 4 u = [w] \implies r \otimes \checkmark 3 w = [x] \implies [[r \otimes \checkmark 1 s] \otimes \checkmark 2 u] = \otimes \checkmark 3 \Rightarrow \otimes \checkmark 2 x \rangle$ 
begin

```

There is a symmetry over the variables.

sublocale Sync_{ptick}-assoc-locale-sym :

Sync_{ptick}-assoc-locale $\langle \lambda u s. s \otimes \checkmark 4 u \rangle \langle \lambda w r. r \otimes \checkmark 3 w \rangle \langle \lambda u t. t \otimes \checkmark 2 u \rangle$

$\langle \lambda s \ r. \ r \ \otimes \checkmark 1 \ s \rangle \langle \otimes \checkmark 3 \Rightarrow \otimes \checkmark 2 \rangle \langle \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 \rangle$
by *unfold-locale*
 (fact *None-assms-tick-join tick-assoc-ren-hyp tick-assoc-ren-conv-hyp*) +
end

6.2.3 First Properties

lemma (in *Sync_{ptick}-assoc-locale*) *tick-assoc-ren-tick-assoc-ren-conv* :
 $\langle \exists r \ s \ u \ w. \ s \ \otimes \checkmark 4 \ u = \lfloor w \rfloor \wedge r \ \otimes \checkmark 3 \ w = \lfloor x \rfloor \implies$
 $\otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 \ (\otimes \checkmark 3 \Rightarrow \otimes \checkmark 2 \ x) = x \rangle$
by (*metis None-assms-tick-join(1,2) option.collapse option.distinct(1)*
option.sel tick-assoc-ren-hyp tick-assoc-ren-conv-hyp)

lemma (in *Sync_{ptick}-assoc-locale*) *tick-assoc-ren-conv-tick-assoc-ren* :
 $\langle \exists r \ s \ t \ u. \ r \ \otimes \checkmark 1 \ s = \lfloor t \rfloor \wedge t \ \otimes \checkmark 2 \ u = \lfloor v \rfloor \implies \otimes \checkmark 3 \Rightarrow \otimes \checkmark 2 \ (\otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 \ v) =$
 $v \rangle$
by (*metis Sync_{ptick}-assoc-locale-sym.tick-assoc-ren-tick-assoc-ren-conv*)

lemma (in *Sync_{ptick}-assoc-locale*) *inj-on-tick-assoc-ren* :
 $\langle \text{inj-on } \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 \ \{v. \exists r \ s \ t \ u. \ r \ \otimes \checkmark 1 \ s = \lfloor t \rfloor \wedge t \ \otimes \checkmark 2 \ u = \lfloor v \rfloor\} \rangle$
by (*rule inj-onI, simp*) (*metis tick-assoc-ren-conv-tick-assoc-ren*)

lemma (in *Sync_{ptick}-assoc-locale*) *inj-on-tick-assoc-ren-conv* :
 $\langle \text{inj-on } \otimes \checkmark 3 \Rightarrow \otimes \checkmark 2 \ \{x. \exists r \ s \ u \ w. \ s \ \otimes \checkmark 4 \ u = \lfloor w \rfloor \wedge r \ \otimes \checkmark 3 \ w = \lfloor x \rfloor\} \rangle$
by (*rule inj-onI, simp*) (*metis tick-assoc-ren-tick-assoc-ren-conv*)

6.2.4 Associativity for the Traces

lemma (in *Sync_{ptick}-assoc-locale*) *setinterleaves_{ptick}-assoc-left* :
 $\langle \llbracket t_t \ \text{setinterleaves}_{\checkmark(\otimes \checkmark 1)} ((t_r, t_s), A);$
 $t_v \ \text{setinterleaves}_{\checkmark(\otimes \checkmark 2)} ((t_t, t_u), A) \rrbracket \implies$
 $\exists t_w. \ \text{map} \ (\text{map-event}_{\text{ptick}} \ \text{id} \ \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3) \ t_v \ \text{setinterleaves}_{\checkmark(\otimes \checkmark 3)} ((t_r, t_w),$
 $A) \wedge$
 $t_w \ \text{setinterleaves}_{\checkmark(\otimes \checkmark 4)} ((t_s, t_u), A) \rangle$

proof –

let *?map* = $\langle \lambda t. \ \text{map} \ \text{ev} \ (\text{map} \ \text{of-ev} \ t) \rangle$
let *?map-event* = $\langle \lambda t. \ \text{map} \ (\text{map-event}_{\text{ptick}} \ \text{id} \ \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3) \ t \rangle$
show $\langle \llbracket t_t \ \text{setinterleaves}_{\checkmark(\otimes \checkmark 1)} ((t_r, t_s), A);$
 $t_v \ \text{setinterleaves}_{\checkmark(\otimes \checkmark 2)} ((t_t, t_u), A) \rrbracket \implies \text{?thesis} \rangle$
proof (*induct* $\langle ((\otimes \checkmark 2), t_t, A, t_u) \rangle$ *arbitrary: t_r t_s t_t t_u t_v*)
case *Nil-setinterleaving_{ptick}-Nil*
thus *?case*
by (*cases t_r; cases t_s*)
 (*auto intro: Nil-setinterleaves_{ptick} simp add: setinterleaving_{ptick}-sims*
split: if-split-asm event_{ptick}.split-asm option.split-asm)
next
case (*ev-setinterleaving_{ptick}-Nil a t_t*)
from *ev-setinterleaving_{ptick}-Nil.prem(2)*

have $\langle a \notin A \rangle \langle tF t_t \rangle \langle \text{set } t_t \cap \text{ev } A = \{\} \rangle \langle t_v = ?\text{map } (\text{ev } a \# t_t) \rangle$
and $\$: \langle ?\text{map } t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 2)} ((t_t, []), A) \rangle$
by (*auto simp add: setinterleaves_{ptick}-NilR-iff split: if-split-asm*)
from *ev-setinterleaving_{ptick}-Nil.prem*_s(1)
consider (*mv-left*) t_r' **where** $\langle t_r = \text{ev } a \# t_r' \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r', t_s), A) \rangle$
 $|$ (*mv-right*) t_s' **where** $\langle t_s = \text{ev } a \# t_s' \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r, t_s'), A) \rangle$
by (*auto simp add: $\langle a \notin A \rangle$ elim: Cons-ev-setinterleaves_{ptick}E*)
thus *?case*
proof cases
case mv-left
from *ev-setinterleaving_{ptick}-Nil.hyps*[*OF* $\langle a \notin A \rangle$ *mv-left*(2) $\$$]
obtain t_w **where** $* : \langle ?\text{map-event } (? \text{map } t_t) \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r', t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, []), A) \rangle$ **by blast**
from $*(1)$ **have** $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
by (*cases* t_w , *auto simp add: mv-left*(1) $\langle a \notin A \rangle \langle t_v = ?\text{map } (\text{ev } a \# t_t) \rangle$
setinterleaving_{ptick}-sims split: event_{ptick}.split)
with $*(2)$ **show** *?thesis* **by blast**
next
case mv-right
from *ev-setinterleaving_{ptick}-Nil.hyps*[*OF* $\langle a \notin A \rangle$ *mv-right*(2) $\$$]
obtain t_w **where** $* : \langle ?\text{map-event } (? \text{map } t_t) \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s', []), A) \rangle$ **by blast**
from $*(2)$ **have** $\langle \text{ev } a \# t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, []), A) \rangle$
by (*simp add: $\langle a \notin A \rangle \langle t_v = ?\text{map } (\text{ev } a \# t_t) \rangle$ mv-right*(1))
moreover from $*(1)$
have $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, \text{ev } a \# t_w), A) \rangle$
by (*cases* t_r , *auto simp add: $\langle a \notin A \rangle \langle t_v = ?\text{map } (\text{ev } a \# t_t) \rangle$ setinterleaving_{ptick}-sims*
split: event_{ptick}.split)
ultimately show *?thesis* **by blast**
qed
next
case (*tick-setinterleaving_{ptick}-Nil* $r t_t$)
from *tick-setinterleaving_{ptick}-Nil.prem*_s(2) **have** *False* **by simp**
thus *?case* ..
next
case (*Nil-setinterleaving_{ptick}-ev* $b t_u$)
from *Nil-setinterleaving_{ptick}-ev.prem*_s(1)[*THEN* *setinterleaves_{ptick}-imp-lengthLR-le*]
have $\langle t_r = [] \rangle \langle t_s = [] \rangle$ **by simp-all**
from *Nil-setinterleaving_{ptick}-ev.prem*_s(2)
have $\langle b \notin A \rangle \langle tF t_u \rangle \langle \text{set } t_u \cap \text{ev } A = \{\} \rangle \langle t_v = ?\text{map } (\text{ev } b \# t_u) \rangle$
and $\$: \langle ?\text{map } t_u \text{ setinterleaves}_{\checkmark(\otimes\checkmark 2)} ([], t_u), A) \rangle$
by (*auto simp add: setinterleaves_{ptick}-NilL-iff split: if-split-asm*)

from $Nil\text{-}setinterleaving_{ptick}\text{-}ev.hyps[OF \langle b \notin A \rangle Nil\text{-}setinterleaving_{ptick}\text{-}ev.prem(1)]$
 $\S$
obtain t_w **where** $\langle ?map\text{-}event \ (?map \ t_u) \ setinterleaves_{\checkmark(\otimes\checkmark 3)} \ ((t_r, t_w), A) \rangle$
 $\langle t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} \ ((t_s, t_u), A) \rangle$ **by** *blast*
hence $\langle ?map\text{-}event \ t_v \ setinterleaves_{\checkmark(\otimes\checkmark 3)} \ ((t_r, ev \ b \ \# \ t_w), A) \wedge$
 $ev \ b \ \# \ t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} \ ((t_s, ev \ b \ \# \ t_u), A) \rangle$
by (*simp add*: $\langle t_v = ?map \ (ev \ b \ \# \ t_u) \rangle \langle t_r = [] \rangle \langle t_s = [] \rangle \langle b \notin A \rangle$)
thus *?case ..*
next
case ($Nil\text{-}setinterleaving_{ptick}\text{-}tick \ r_u \ t_u$)
from $Nil\text{-}setinterleaving_{ptick}\text{-}tick.prem(2)$ **have** *False* **by** *simp*
thus *?case ..*
next
case ($ev\text{-}setinterleaving_{ptick}\text{-}ev \ a \ t_t \ b \ t_u$)
from $ev\text{-}setinterleaving_{ptick}\text{-}ev.prem(2)$
consider ($mv\text{-}both$) t_v' **where** $\langle a \in A \rangle \langle b \in A \rangle \langle a = b \rangle \langle t_v = ev \ b \ \# \ t_v' \rangle \langle t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 2)} \ ((t_t, t_u), A) \rangle$
 $| \quad (mvR\text{-}inL) \ t_v' \textbf{ where } \langle a \in A \rangle \langle b \notin A \rangle \langle t_v = ev \ b \ \# \ t_v' \rangle \langle t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 2)} \ ((ev \ a \ \# \ t_t, t_u), A) \rangle$
 $| \quad (mvL\text{-}inR) \ t_v' \textbf{ where } \langle a \notin A \rangle \langle b \in A \rangle \langle t_v = ev \ a \ \# \ t_v' \rangle \langle t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 2)} \ ((t_t, ev \ b \ \# \ t_u), A) \rangle$
 $| \quad (mvR\text{-}notin) \ t_v' \textbf{ where } \langle a \notin A \rangle \langle b \notin A \rangle \langle t_v = ev \ b \ \# \ t_v' \rangle \langle t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 2)} \ ((ev \ a \ \# \ t_t, t_u), A) \rangle$
 $| \quad (mvL\text{-}notin) \ t_v' \textbf{ where } \langle a \notin A \rangle \langle b \notin A \rangle \langle t_v = ev \ a \ \# \ t_v' \rangle \langle t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 2)} \ ((t_t, ev \ b \ \# \ t_u), A) \rangle$
by (*auto split: if-split-asm*)
thus *?case*
proof cases
case *mv-both*
from $ev\text{-}setinterleaving_{ptick}\text{-}ev.prem(1)$
obtain $t_r' \ t_s'$ **where** $\langle t_r = ev \ a \ \# \ t_r' \rangle \langle t_s = ev \ a \ \# \ t_s' \rangle$
 $\langle t_t \ setinterleaves_{\checkmark(\otimes\checkmark 1)} \ ((t_r', t_s'), A) \rangle$
by (*auto simp add*: $\langle a \in A \rangle \text{elim: } Cons\text{-}ev\text{-}setinterleaves_{ptick}E$)
from $ev\text{-}setinterleaving_{ptick}\text{-}ev.hyps(1)[OF \ mv\text{-}both(1-3) \ \langle t_t \ setinterleaves_{\checkmark(\otimes\checkmark 1)} \ ((t_r', t_s'), A) \rangle \ mv\text{-}both(5)]$
obtain t_w **where** $\langle ?map\text{-}event \ t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 3)} \ ((t_r', t_w), A) \rangle$
 $\langle t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} \ ((t_s', t_u), A) \rangle$ **by** *blast*
hence $\langle ?map\text{-}event \ t_v \ setinterleaves_{\checkmark(\otimes\checkmark 3)} \ ((t_r, ev \ a \ \# \ t_w), A) \wedge$
 $ev \ a \ \# \ t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} \ ((t_s, ev \ b \ \# \ t_u), A) \rangle$
by (*simp add*: $mv\text{-}both(2-4) \ \langle t_s = ev \ a \ \# \ t_s' \rangle \langle t_r = ev \ a \ \# \ t_r' \rangle$)
thus *?thesis ..*
next
case *mvR-inL*
from $ev\text{-}setinterleaving_{ptick}\text{-}ev.prem(1)$
obtain $t_r' \ t_s'$ **where** $\langle t_r = ev \ a \ \# \ t_r' \rangle \langle t_s = ev \ a \ \# \ t_s' \rangle$
 $\langle t_t \ setinterleaves_{\checkmark(\otimes\checkmark 1)} \ ((t_r', t_s'), A) \rangle$

by (auto simp add: $\langle a \in A \rangle$ elim: Cons-ev-setinterleaves_{ptick}E)
 from ev-setinterleaving_{ptick}-ev.hyps(2)[OF mvR-inL(1, 2) ev-setinterleaving_{ptick}-ev.prem(1)
 mvR-inL(4)]
 obtain t_w where $\langle ?\text{map-event } t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, t_u), A) \rangle$ by blast
 hence $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, \text{ev } b \# t_w), A) \wedge$
 $\text{ev } b \# t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \text{ev } b \# t_u), A) \rangle$
 by (simp add: $\langle t_r = \text{ev } a \# t_r' \rangle \langle t_s = \text{ev } a \# t_s' \rangle$ mvR-inL(1-3))
 thus ?thesis ..
 next
 case mvL-inR
 from ev-setinterleaving_{ptick}-ev.prem(1)
 consider (mv-left) t_r' where $\langle t_r = \text{ev } a \# t_r' \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r',$
 $t_s), A) \rangle$
 $| (mv\text{-right}) t_s' \text{ where } \langle t_s = \text{ev } a \# t_s' \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r, t_s'),$
 $A) \rangle$
 by (auto simp add: $\langle a \notin A \rangle$ elim: Cons-ev-setinterleaves_{ptick}E)
 thus ?thesis
 proof cases
 case mv-left
 from ev-setinterleaving_{ptick}-ev.hyps(3)[OF mvL-inR(1, 2) mv-left(2)
 mvL-inR(4)]
 obtain t_w where $\langle ?\text{map-event } t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r', t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \text{ev } b \# t_u), A) \rangle$ by blast
 hence $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \wedge$
 $t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \text{ev } b \# t_u), A) \rangle$
 by (cases t_w) (auto simp add: mvL-inR(1-3) mv-left(1)
 setinterleaving_{ptick}-simps split: event_{ptick}.split)
 thus ?thesis ..
 next
 case mv-right
 from ev-setinterleaving_{ptick}-ev.hyps(3)[OF mvL-inR(1, 2) mv-right(2)
 mvL-inR(4)]
 obtain t_w where $\langle ?\text{map-event } t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s', \text{ev } b \# t_u), A) \rangle$ by blast
 hence $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, \text{ev } a \# t_w), A) \wedge$
 $\text{ev } a \# t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \text{ev } b \# t_u), A) \rangle$
 by (cases t_r) (auto simp add: mvL-inR(1-3) mv-right(1)
 setinterleaving_{ptick}-simps split: event_{ptick}.split)
 thus ?thesis ..
 qed
 next
 case mvR-notin
 from ev-setinterleaving_{ptick}-ev.prem(1)
 consider (mv-left) t_r' where $\langle t_r = \text{ev } a \# t_r' \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r',$
 $t_s), A) \rangle$
 $| (mv\text{-right}) t_s' \text{ where } \langle t_s = \text{ev } a \# t_s' \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r, t_s'),$

$A\rangle$
by (auto simp add: $\langle a \notin A \rangle$ elim: Cons-ev-setinterleaves_{ptick} E)
thus ?thesis
proof cases
case mv-left
from mv-left(2) **have** $\langle ev\ a \# t_t\ setinterleaves_{\checkmark(\otimes\checkmark 1)}((t_r, t_s), A) \rangle$
by (cases t_s) (auto simp add: mv-left(1) mvR-notin(1)
setinterleaving_{ptick}-simps split: event_{ptick}.split)
from ev-setinterleaving_{ptick}-ev.hyps(5)[OF mvR-notin(1, 2) this mvR-notin(4)]
obtain t_w **where** $\langle ?map-event\ t_v'\ setinterleaves_{\checkmark(\otimes\checkmark 3)}((t_r, t_w), A) \rangle$
 $\langle t_w\ setinterleaves_{\checkmark(\otimes\checkmark 4)}((t_s, t_u), A) \rangle$ **by** blast
hence $\langle ?map-event\ t_v\ setinterleaves_{\checkmark(\otimes\checkmark 3)}((t_r, ev\ b \# t_w), A) \wedge$
 $ev\ b \# t_w\ setinterleaves_{\checkmark(\otimes\checkmark 4)}((t_s, ev\ b \# t_u), A) \rangle$
by (cases t_s) (auto simp add: mvR-notin(1-3) mv-left(1)
setinterleaving_{ptick}-simps split: event_{ptick}.split)
thus ?thesis ..
next
case mv-right
from mv-right(2) **have** $\langle ev\ a \# t_t\ setinterleaves_{\checkmark(\otimes\checkmark 1)}((t_r, t_s), A) \rangle$
by (cases t_r) (auto simp add: mv-right(1) mvR-notin(1)
setinterleaving_{ptick}-simps split: event_{ptick}.split)
from ev-setinterleaving_{ptick}-ev.hyps(5)[OF mvR-notin(1, 2) this mvR-notin(4)]
obtain t_w **where** $\langle ?map-event\ t_v'\ setinterleaves_{\checkmark(\otimes\checkmark 3)}((t_r, t_w), A) \rangle$
 $\langle t_w\ setinterleaves_{\checkmark(\otimes\checkmark 4)}((t_s, t_u), A) \rangle$ **by** blast
hence $\langle ?map-event\ t_v\ setinterleaves_{\checkmark(\otimes\checkmark 3)}((t_r, ev\ b \# t_w), A) \wedge$
 $ev\ b \# t_w\ setinterleaves_{\checkmark(\otimes\checkmark 4)}((t_s, ev\ b \# t_u), A) \rangle$
by (cases t_r) (auto simp add: mvR-notin(1-3) mv-right(1)
setinterleaving_{ptick}-simps split: event_{ptick}.split)
thus ?thesis ..
qed
next
case mvL-notin
from ev-setinterleaving_{ptick}-ev.premis(1)
consider (mv-left) t_r' **where** $\langle t_r = ev\ a \# t_r' \rangle \langle t_t\ setinterleaves_{\checkmark(\otimes\checkmark 1)}((t_r',$
 $t_s), A) \rangle$
 $| (mv-right)\ t_s' \textbf{ where } \langle t_s = ev\ a \# t_s' \rangle \langle t_t\ setinterleaves_{\checkmark(\otimes\checkmark 1)}((t_r, t_s'),$
 $A) \rangle$
by (auto simp add: $\langle a \notin A \rangle$ elim: Cons-ev-setinterleaves_{ptick} E)
thus ?thesis
proof cases
case mv-left
from ev-setinterleaving_{ptick}-ev.hyps(4)[OF mvL-notin(1, 2) mv-left(2)
mvL-notin(4)]
obtain t_w **where** $\langle ?map-event\ t_v'\ setinterleaves_{\checkmark(\otimes\checkmark 3)}((t_r', t_w), A) \rangle$
 $\langle t_w\ setinterleaves_{\checkmark(\otimes\checkmark 4)}((t_s, ev\ b \# t_u), A) \rangle$ **by** blast
hence $\langle ?map-event\ t_v\ setinterleaves_{\checkmark(\otimes\checkmark 3)}((t_r, t_w), A) \wedge$
 $t_w\ setinterleaves_{\checkmark(\otimes\checkmark 4)}((t_s, ev\ b \# t_u), A) \rangle$

by (cases t_w) (auto simp add: mv-left(1) mvL-notin(1, 3)
 setinterleaving_{ptick}-simps split: event_{ptick}.split)
 thus ?thesis ..
 next
 case mv-right
 from ev-setinterleaving_{ptick}-ev.hyps(4)[OF mvL-notin(1, 2) mv-right(2)
 mvL-notin(4)]
 obtain t_w where $\langle ?\text{map-event } t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s', \text{ev } b \# t_u), A) \rangle$ by blast
 hence $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, \text{ev } a \# t_w), A) \wedge$
 $\text{ev } a \# t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \text{ev } b \# t_u), A) \rangle$
 by (cases t_r) (auto simp add: mv-right(1) mvL-notin(1, 3)
 setinterleaving_{ptick}-simps split: event_{ptick}.split)
 thus ?thesis ..
 qed
 qed
 next
 case (ev-setinterleaving_{ptick}-tick a t_t s t_u)
 from ev-setinterleaving_{ptick}-tick.prem(2) obtain t_v'
 where $\langle a \notin A \rangle \langle t_v = \text{ev } a \# t_v' \rangle$
 and $\$: \langle t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 2)} ((t_t, \checkmark(s) \# t_u), A) \rangle$
 by (auto split: if-split-asm)
 from ev-setinterleaving_{ptick}-tick.prem(1)
 have $\langle t_r \neq [] \wedge \text{hd } t_r = \text{ev } a \wedge t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((tl \ t_r, t_s), A) \vee$
 $t_s \neq [] \wedge \text{hd } t_s = \text{ev } a \wedge t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r, tl \ t_s), A) \rangle$
 by (auto simp add: $\langle a \notin A \rangle$ elim: Cons-ev-setinterleaves_{ptick}E)
 thus ?case
 proof (elim disjE conjE)
 assume $\langle t_r \neq [] \rangle \langle \text{hd } t_r = \text{ev } a \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((tl \ t_r, t_s), A) \rangle$
 from ev-setinterleaving_{ptick}-tick.hyps[OF $\langle a \notin A \rangle$ this(3) \$]
 obtain t_w where $* : \langle ?\text{map-event } t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((tl \ t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \checkmark(s) \# t_u), A) \rangle$ by blast
 from *(1) have $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
 by (subst list.collapse[OF $\langle t_r \neq [] \rangle$, symmetric])
 (cases t_w , auto simp add: $\langle a \notin A \rangle \langle \text{hd } t_r = \text{ev } a \rangle \langle t_v = \text{ev } a \# t_v' \rangle$
 setinterleaving_{ptick}-simps split: event_{ptick}.split)
 with *(2) show ?case by blast
 next
 assume $\langle t_s \neq [] \rangle \langle \text{hd } t_s = \text{ev } a \rangle \langle t_t \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t_r, tl \ t_s), A) \rangle$
 from ev-setinterleaving_{ptick}-tick.hyps[OF $\langle a \notin A \rangle$ this(3) \$]
 obtain t_w where $* : \langle ?\text{map-event } t_v' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((tl \ t_s, \checkmark(s) \# t_u), A) \rangle$ by blast
 from *(2) have $\langle \text{ev } a \# t_w \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t_s, \checkmark(s) \# t_u), A) \rangle$
 by (subst list.collapse[OF $\langle t_s \neq [] \rangle$, symmetric])
 (simp add: $\langle a \notin A \rangle \langle t_v = \text{ev } a \# t_v' \rangle \langle \text{hd } t_s = \text{ev } a \rangle$)
 moreover from *(1)
 have $\langle ?\text{map-event } t_v \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t_r, \text{ev } a \# t_w), A) \rangle$

by (cases t_r , auto simp add: $\langle a \notin A \rangle \langle t_v = ev\ a \ \# \ t_v' \rangle$ setinterleaving_{ptick-simps}
 split: event_{ptick}.split)
 ultimately show ?case by blast
 qed
 next
 case (tick-setinterleaving_{ptick}-ev $r_t\ t_t\ b\ t_u$)
 from tick-setinterleaving_{ptick}-ev.prem(1)
 obtain $r_r\ r_s\ t_r'\ t_s'$ where $\langle r_r \otimes \checkmark 1\ r_s \rangle = \lfloor r_t \rfloor \rangle \langle t_r = \checkmark(r_r) \ \# \ t_r' \rangle \langle t_s = \checkmark(r_s) \ \# \ t_s' \rangle$
 by (auto elim: Cons-tick-setinterleaves_{ptick}E)
 from tick-setinterleaving_{ptick}-ev.prem(2) obtain t_v'
 where $\langle b \notin A \rangle \langle t_v = ev\ b \ \# \ t_v' \rangle$
 and $\$: \langle t_v' \text{ setinterleaves } \checkmark(\otimes \checkmark 2) ((\checkmark(r_t) \ \# \ t_t, t_u), A) \rangle$
 by (auto split: if-split-asm)
 from tick-setinterleaving_{ptick}-ev.hyps[OF $\langle b \notin A \rangle$ tick-setinterleaving_{ptick}-ev.prem(1)
 \$]
 obtain t_w where $\langle ?map\text{-}event\ t_v' \text{ setinterleaves } \checkmark(\otimes \checkmark 3) ((t_r, t_w), A) \rangle$
 $\langle t_w \text{ setinterleaves } \checkmark(\otimes \checkmark 4) ((t_s, t_u), A) \rangle$ by blast
 hence $\langle ?map\text{-}event\ t_v \text{ setinterleaves } \checkmark(\otimes \checkmark 3) ((t_r, ev\ b \ \# \ t_w), A) \wedge$
 $ev\ b \ \# \ t_w \text{ setinterleaves } \checkmark(\otimes \checkmark 4) ((t_s, ev\ b \ \# \ t_u), A) \rangle$
 by (simp add: $\langle b \notin A \rangle \langle t_v = ev\ b \ \# \ t_v' \rangle \langle t_r = \checkmark(r_r) \ \# \ t_r' \rangle \langle t_s = \checkmark(r_s) \ \# \ t_s' \rangle$)
 thus ?case ..
 next
 case (tick-setinterleaving_{ptick}-tick $r_t\ t_t\ r_u\ t_u$)
 from tick-setinterleaving_{ptick}-tick.prem(1)
 obtain $r_r\ r_s\ t_r'\ t_s'$ where $\langle r_r \otimes \checkmark 1\ r_s \rangle = \lfloor r_t \rfloor \rangle \langle t_r = \checkmark(r_r) \ \# \ t_r' \rangle \langle t_s = \checkmark(r_s) \ \# \ t_s' \rangle$
 $\langle t_t \text{ setinterleaves } \checkmark(\otimes \checkmark 1) ((t_r', t_s'), A) \rangle$
 by (auto elim: Cons-tick-setinterleaves_{ptick}E)
 from tick-setinterleaving_{ptick}-tick.prem(2)
 obtain $r_v\ t_v'$ where $\langle r_t \otimes \checkmark 2\ r_u \rangle = \lfloor r_v \rfloor \rangle \langle t_v = \checkmark(r_v) \ \# \ t_v' \rangle$
 $\langle t_v' \text{ setinterleaves } \checkmark(\otimes \checkmark 2) ((t_t, t_u), A) \rangle$
 by (auto split: option.split-asm)
 from $\langle r_r \otimes \checkmark 1\ r_s \rangle = \lfloor r_t \rfloor \rangle \langle r_t \otimes \checkmark 2\ r_u \rangle = \lfloor r_v \rfloor \rangle$ obtain r_w where $\langle r_s \otimes \checkmark 4\ r_u \rangle = \lfloor r_w \rfloor \rangle$
 by (metis None-assms-tick-join(3) not-None-eq option.sel)
 from $\langle r_s \otimes \checkmark 4\ r_u \rangle = \lfloor r_w \rfloor \rangle \langle r_r \otimes \checkmark 1\ r_s \rangle = \lfloor r_t \rfloor \rangle \langle r_t \otimes \checkmark 2\ r_u \rangle = \lfloor r_v \rfloor \rangle$
 obtain r_x where $\langle r_r \otimes \checkmark 3\ r_w \rangle = \lfloor r_x \rfloor \rangle$
 by (metis None-assms-tick-join(4) option.distinct(1) option.exhaust-sel option.sel)
 have $\langle \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3\ r_v = r_x \rangle$
 by (metis $\langle r_r \otimes \checkmark 1\ r_s \rangle = \lfloor r_t \rfloor \rangle \langle r_r \otimes \checkmark 3\ r_w \rangle = \lfloor r_x \rfloor \rangle \langle r_s \otimes \checkmark 4\ r_u \rangle = \lfloor r_w \rfloor \rangle$
 $\langle r_t \otimes \checkmark 2\ r_u \rangle = \lfloor r_v \rfloor \rangle$ tick-assoc-ren-hyp option.sel)
 from tick-setinterleaving_{ptick}-tick.hyps
 [OF $\langle r_t \otimes \checkmark 2\ r_u \rangle = \lfloor r_v \rfloor \rangle \langle t_t \text{ setinterleaves } \checkmark(\otimes \checkmark 1) ((t_r', t_s'), A) \rangle$
 $\langle t_v' \text{ setinterleaves } \checkmark(\otimes \checkmark 2) ((t_t, t_u), A) \rangle]$

obtain t_w **where** $\langle ?map\text{-}event\ t_v' \ setinterleaves_{\checkmark(\otimes\checkmark 3)} ((t_r', t_w), A) \rangle$
 $\langle t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} ((t_s', t_u), A) \rangle$ **by** *blast*
hence $\langle ?map\text{-}event\ t_v \ setinterleaves_{\checkmark(\otimes\checkmark 3)} ((t_r, \checkmark(r_w) \# t_w), A) \wedge$
 $\checkmark(r_w) \# t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} ((t_s, \checkmark(r_u) \# t_u), A) \rangle$
by (*simp add*: $\langle t_r = \checkmark(r_r) \# t_r' \rangle \langle t_s = \checkmark(r_s) \# t_s' \rangle \langle t_v = \checkmark(r_v) \# t_v' \rangle$
 $\langle r_s \otimes\checkmark 4\ r_u = [r_w] \rangle \langle r_r \otimes\checkmark 3\ r_w = [r_x] \rangle \langle \otimes\checkmark 2 \Rightarrow \otimes\checkmark 3\ r_v = r_x \rangle$)
thus *?case ..*
qed
qed

lemma (**in** *Sync_{ptick}-assoc-locale*) *setinterleaves_{ptick}-assoc-right* :
 $\langle t_w \ setinterleaves_{\checkmark(\otimes\checkmark 4)} ((t_s, t_u), A) \Rightarrow$
 $t_x \ setinterleaves_{\checkmark(\otimes\checkmark 3)} ((t_r, t_w), A) \Rightarrow$
 $\exists t_t. \ map \ (map\text{-}event_{ptick} \ id \ \otimes\checkmark 3 \Rightarrow \otimes\checkmark 2) \ t_x \ setinterleaves_{\checkmark(\otimes\checkmark 2)} ((t_t, t_u), A)$
 $\wedge$
 $t_t \ setinterleaves_{\checkmark(\otimes\checkmark 1)} ((t_r, t_s), A) \rangle$
by (*subst* (1 2) *setinterleaves_{ptick}-sym*, *subst* (*asm*) (1 2) *setinterleaves_{ptick}-sym*)
(fact Sync_{ptick}-assoc-locale-sym.setinterleaves_{ptick}-assoc-left)

6.2.5 Associativity

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

notation *Sync_{ptick}1.Sync_{ptick}* $\langle (- \llbracket - \rrbracket_{\checkmark 1} -) \rangle [70, 0, 71] \ 70)$
notation *Sync_{ptick}2.Sync_{ptick}* $\langle (- \llbracket - \rrbracket_{\checkmark 2} -) \rangle [70, 0, 71] \ 70)$
notation *Sync_{ptick}3.Sync_{ptick}* $\langle (- \llbracket - \rrbracket_{\checkmark 3} -) \rangle [70, 0, 71] \ 70)$
notation *Sync_{ptick}4.Sync_{ptick}* $\langle (- \llbracket - \rrbracket_{\checkmark 4} -) \rangle [70, 0, 71] \ 70)$

lemma *Sync_{ptick}-assoc-oneside* :
 $\langle P \llbracket S \rrbracket_{\checkmark 3} (Q \llbracket S \rrbracket_{\checkmark 4} R) \sqsubseteq_{FD} RenamingTick \ (P \llbracket S \rrbracket_{\checkmark 1} Q \llbracket S \rrbracket_{\checkmark 2} R) \otimes\checkmark 2 \Rightarrow \otimes\checkmark 3 \rangle$
(is $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$
proof –
let *?map-event* = $\langle \lambda t. \ map \ (map\text{-}event_{ptick} \ id \ \otimes\checkmark 2 \Rightarrow \otimes\checkmark 3) \ t \rangle$
let *?map-event-conv* = $\langle \lambda t. \ map \ (map\text{-}event_{ptick} \ id \ \otimes\checkmark 3 \Rightarrow \otimes\checkmark 2) \ t \rangle$
show $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$
proof (*rule failure-divergence-refine-optimizedI*)
fix t **assume** $\langle t \in \mathcal{D} \ ?rhs \rangle$
then obtain $t_1 \ t_2$ **where** $\langle t = ?map\text{-}event \ t_1 \ @ \ t_2 \rangle$
 $\langle tF \ t_1 \rangle \langle ftF \ t_2 \rangle \langle t_1 \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} Q \llbracket S \rrbracket_{\checkmark 2} R) \rangle$
unfolding *D-Renaming* **by** *blast*
from $\langle t_1 \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} Q \llbracket S \rrbracket_{\checkmark 2} R) \rangle$ **obtain** $t_{11} \ t_{12} \ t\text{-}P\text{-}Q \ t\text{-}R$
where $*$: $\langle t_1 = t_{11} \ @ \ t_{12} \rangle \langle tF \ t_{11} \rangle \langle ftF \ t_{12} \rangle$
 $\langle t_{11} \ setinterleaves_{\checkmark(\otimes\checkmark 2)} ((t\text{-}P\text{-}Q, t\text{-}R), S) \rangle$
 $\langle t\text{-}P\text{-}Q \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} Q) \wedge t\text{-}R \in \mathcal{T} \ R \vee$

$t\text{-}P\text{-}Q \in \mathcal{T} (P \llbracket S \rrbracket_{\checkmark 1} Q) \wedge t\text{-}P\text{-}Q \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark 1} Q) \wedge t\text{-}R \in \mathcal{D} R$
unfolding $\text{Sync}_{ptick2}.D\text{-}\text{Sync}_{ptick}$ **using** $D\text{-}T$ **by** *blast*
from $\ast(5)$ **show** $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$
proof (*elim disjE conjE*)
assume $\langle t\text{-}P\text{-}Q \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark 1} Q) \rangle \langle t\text{-}R \in \mathcal{T} R \rangle$
from $\langle t\text{-}P\text{-}Q \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark 1} Q) \rangle$ **obtain** $t\text{-}P\text{-}Q_1 \ t\text{-}P\text{-}Q_2 \ t\text{-}P \ t\text{-}Q$
where $\ast\ast : \langle t\text{-}P\text{-}Q = t\text{-}P\text{-}Q_1 @ t\text{-}P\text{-}Q_2 \rangle \langle tF \ t\text{-}P\text{-}Q_1 \rangle \langle ftF \ t\text{-}P\text{-}Q_2 \rangle$
 $\langle t\text{-}P\text{-}Q_1 \ \text{setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle t\text{-}P \in \mathcal{D} P \wedge t\text{-}Q \in \mathcal{T} Q \vee t\text{-}P \in \mathcal{T} P \wedge t\text{-}Q \in \mathcal{D} Q \rangle$
unfolding $\text{Sync}_{ptick1}.D\text{-}\text{Sync}_{ptick}$ **by** *blast*
from $\ast(4)[\text{unfolded } \ast\ast(1), \text{ THEN } \text{setinterleaves}_{ptick}\text{-appendL}]$
obtain $t_{111} \ t_{112} \ t\text{-}R_1 \ t\text{-}R_2$
where $\ast\ast\ast : \langle t_{111} = t_{111} @ t_{112} \rangle \langle t\text{-}R = t\text{-}R_1 @ t\text{-}R_2 \rangle$
 $\langle t_{111} \ \text{setinterleaves}_{\checkmark(\otimes\checkmark 2)} ((t\text{-}P\text{-}Q_1, t\text{-}R_1), S) \rangle$
 $\langle t_{112} \ \text{setinterleaves}_{\checkmark(\otimes\checkmark 2)} ((t\text{-}P\text{-}Q_2, t\text{-}R_2), S) \rangle$ **by** *blast*
from $\text{setinterleaves}_{ptick}\text{-assoc-left}[OF \ \ast\ast(4) \ \ast\ast\ast(3)]$
obtain $t\text{-}Q\text{-}R$ **where** $\ast\ast\ast\ast : \langle \text{?map-event } t_{111} \ \text{setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t\text{-}P,$
 $t\text{-}Q\text{-}R), S) \rangle$
 $\langle t\text{-}Q\text{-}R \ \text{setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t\text{-}Q, t\text{-}R_1), S) \rangle$ **by** *blast*
have $\langle tF \ (\text{?map-event } t_1) \rangle$
by (*simp add: $\langle tF \ t_1 \rangle \ \text{map-event}_{ptick}\text{-tickFree}$*)
moreover have $\langle ftF \ (\text{?map-event } (t_{112} @ t_{12}) @ t_2) \rangle$
by (*metis $\ast(1) \ \ast\ast\ast(1) \ \langle ftF \ t_2 \rangle \ \langle tF \ t_1 \rangle \ \text{front-tickFree-append}$*
 $\text{map-event}_{ptick}\text{-tickFree } \text{tickFree-append-iff}$)
moreover from $\ast\ast(5)$
have $\langle t\text{-}P \in \mathcal{D} P \wedge t\text{-}Q\text{-}R \in \mathcal{T} (Q \llbracket S \rrbracket_{\checkmark 4} R) \vee t\text{-}P \in \mathcal{T} P \wedge t\text{-}Q\text{-}R \in \mathcal{D} (Q$
 $\llbracket S \rrbracket_{\checkmark 4} R) \rangle$
proof (*elim disjE conjE*)
assume $\langle t\text{-}P \in \mathcal{D} P \rangle \langle t\text{-}Q \in \mathcal{T} Q \rangle$
hence $\langle t\text{-}P \in \mathcal{D} P \wedge t\text{-}Q\text{-}R \in \mathcal{T} (Q \llbracket S \rrbracket_{\checkmark 4} R) \rangle$
by (*simp add: $\text{Sync}_{ptick4}.T\text{-}\text{Sync}_{ptick}$*)
 $(\text{metis } \ast\ast\ast(2) \ \ast\ast\ast(2) \ \langle t\text{-}R \in \mathcal{T} R \rangle \ \text{is-processT3-TR-append})$
thus *?thesis* ..
next
assume $\langle t\text{-}P \in \mathcal{T} P \rangle \langle t\text{-}Q \in \mathcal{D} Q \rangle$
from $\ast\ast(2, 4)$ **have** $\langle tF \ t\text{-}Q \rangle$ **by** (*simp add: $\text{tickFree-setinterleaves}_{ptick}\text{-iff}$*)
with $\ast\ast\ast(2) \ \text{setinterleaves}_{ptick}\text{-tickFree-imp}$ **have** $\langle tF \ t\text{-}Q\text{-}R \rangle$ **by** *blast*
moreover from $\ast\ast\ast(2) \ \langle t\text{-}R \in \mathcal{T} R \rangle \ \text{is-processT3-TR-append}$ **have** $\langle t\text{-}R_1$
 $\in \mathcal{T} R \rangle$ **by** *blast*
ultimately have $\langle t\text{-}P \in \mathcal{T} P \wedge t\text{-}Q\text{-}R \in \mathcal{D} (Q \llbracket S \rrbracket_{\checkmark 4} R) \rangle$
unfolding $\text{Sync}_{ptick4}.D\text{-}\text{Sync}_{ptick}$
using $\ast\ast\ast(2) \ \langle t\text{-}P \in \mathcal{T} P \rangle \langle t\text{-}Q \in \mathcal{D} Q \rangle \ \text{front-tickFree-Nil}$ **by** *blast*
thus *?thesis* ..
qed
ultimately show $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$
using $\ast\ast\ast(1)$ **by** (*auto simp add: $\langle t = \text{?map-event } t_1 @ t_2 \rangle$*
 $\ast(1) \ \ast\ast\ast(1) \ \text{Sync}_{ptick3}.D\text{-}\text{Sync}_{ptick}$)
next

assume $\langle t-P-Q \in \mathcal{T} (P \llbracket S \rrbracket_{\checkmark 1} Q) \rangle \langle t-P-Q \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark 1} Q) \rangle \langle t-R \in \mathcal{D} R \rangle$
from $\text{this}(1, 2)$ **obtain** $t-P \ t-Q$
where $** : \langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle \langle t-P-Q \text{ setinterleaves}_{\checkmark(\otimes \checkmark 1)} ((t-P, t-Q), S) \rangle$
unfolding $\text{Sync}_{ptick1}.\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
from $\text{setinterleaves}_{ptick}\text{-assoc-left}[OF \ **(\checkmark 3) \ **(\checkmark 4)]$ **obtain** $t-Q'$
where $*** : \langle ?\text{map-event } t_{11} \text{ setinterleaves}_{\checkmark(\otimes \checkmark 3)} ((t-P, t-Q'), S) \rangle$
 $\langle t-Q' \text{ setinterleaves}_{\checkmark(\otimes \checkmark 4)} ((t-Q, t-R), S) \rangle$ **by** *blast*
from $*(2) \ **(\checkmark 2) \ *** \langle t-R \in \mathcal{D} R \rangle$ **have** $\langle t-Q' \in \mathcal{D} (Q \llbracket S \rrbracket_{\checkmark 4} R) \rangle$
by (*simp add: Sync_{ptick4}.D-Sync_{ptick}*)
(metis append.right-neutral front-tickFree-Nil
map-event_{ptick}-tickFree tickFree-setinterleaves_{ptick}-iff)
moreover **have** $\langle t = ?\text{map-event } t_{11} @ (? \text{map-event } t_{12} @ t_2) \rangle$
by (*simp add: *(1) $\langle t = ?\text{map-event } t_1 @ t_2 \rangle$*)
moreover **have** $\langle tF (? \text{map-event } t_{11}) \rangle$
by (*simp add: *(2) map-event_{ptick}-tickFree*)
moreover from $*(1) \ \langle tF t_2 \rangle \langle tF t_1 \rangle$ **have** $\langle tF (? \text{map-event } t_{12} @ t_2) \rangle$
using *front-tickFree-append map-event_{ptick}-tickFree tickFree-append-iff* **by**
blast
ultimately show $\langle t \in \mathcal{D} ?lhs \rangle$
unfolding $\text{Sync}_{ptick3}.\text{D-Sync}_{ptick}$ **using** $*(1) \ ***(\checkmark 1)$ **by** *blast*
qed
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} ?rhs \rangle \langle \mathcal{D} ?rhs \subseteq \mathcal{D} ?lhs \rangle$
then consider $\langle t \in \mathcal{D} ?rhs \rangle$
| t_1 **where** $\langle t = ?\text{map-event } t_1 \rangle \langle t \notin \mathcal{D} ?rhs \rangle$
 $\langle (t_1, \text{map-event}_{ptick} \text{ id } \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 - 'X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark 1} Q \llbracket S \rrbracket_{\checkmark 2} R) \rangle$
unfolding *Renaming-projs* **by** *blast*
thus $\langle (t, X) \in \mathcal{F} ?lhs \rangle$
proof cases
assume $\langle t \in \mathcal{D} ?rhs \rangle$
with $\langle \mathcal{D} ?rhs \subseteq \mathcal{D} ?lhs \rangle$ **have** $\langle t \in \mathcal{D} ?lhs \rangle$ **by** *blast*
thus $\langle (t, X) \in \mathcal{F} ?lhs \rangle$ **by** (*fact is-processT8*)
next
fix t_1 **assume** $* : \langle t = ?\text{map-event } t_1 \rangle \langle t \notin \mathcal{D} ?rhs \rangle$
 $\langle (t_1, \text{map-event}_{ptick} \text{ id } \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 - 'X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark 1} Q \llbracket S \rrbracket_{\checkmark 2} R) \rangle$
from $*(1) \ \langle t \notin \mathcal{D} ?rhs \rangle$ **have** $\langle t_1 \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark 1} Q \llbracket S \rrbracket_{\checkmark 2} R) \rangle$
by (*cases $\langle tF t_1 \rangle$, simp-all add: D-Renaming*)
(use front-tickFree-Nil in blast,
metis D-imp-front-tickFree front-tickFree-append-iff is-processT9
map-append map-event_{ptick}-front-tickFree
nonTickFree-n-frontTickFree non-tickFree-tick tickFree-Nil)
with $*(3)$ **obtain** $t-P-Q \ X-P-Q \ t-R \ X-R$
where $** : \langle (t-P-Q, X-P-Q) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark 1} Q) \rangle \langle (t-R, X-R) \in \mathcal{F} R \rangle$
 $\langle t_1 \text{ setinterleaves}_{\checkmark(\otimes \checkmark 2)} ((t-P-Q, t-R), S) \rangle$
 $\langle \text{map-event}_{ptick} \text{ id } \otimes \checkmark 2 \Rightarrow \otimes \checkmark 3 - 'X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark 2) \ X-P-Q \ S \ X-R \rangle$
unfolding $\text{Sync}_{ptick2}.\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
from $*(1)$ **consider** $\langle t-P-Q \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark 1} Q) \rangle$

| (fail) $t\text{-}P \ X\text{-}P \ t\text{-}Q \ X\text{-}Q$ **where** $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle \ \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle$
 $\langle t\text{-}P\text{-}Q \text{ setinterleaves}_{\checkmark(\otimes\checkmark 1)} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X\text{-}P\text{-}Q \subseteq \text{super-ref-Sync}_{ptick} (\otimes\checkmark 1) \ X\text{-}P \ S \ X\text{-}Q \rangle$
unfolding $\text{Sync}_{ptick1}.\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
thus $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$
proof *cases*
assume $\langle t\text{-}P\text{-}Q \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q) \rangle$
have $\langle t_1 \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q \llbracket S \rrbracket_{\checkmark 2} \ R) \rangle$
proof (*cases* $\langle tF \ t\text{-}P\text{-}Q \rangle$)
assume $\langle tF \ t\text{-}P\text{-}Q \rangle$
with $**(\checkmark 3)[\text{THEN setinterleaves}_{ptick}\text{-tickFree-imp}[\text{rotated}]]$ **have** $\langle tF \ t_1 \rangle$
by *simp*
with $**(\checkmark 3) \ **(\checkmark 2)[\text{THEN } F\text{-}T]$ $\langle t\text{-}P\text{-}Q \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q) \rangle$
show $\langle t_1 \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q \llbracket S \rrbracket_{\checkmark 2} \ R) \rangle$
by (*simp add: Sync_{ptick2}.D-Sync_{ptick}*)
(meson front-tickFree-Nil self-append-conv)
next
assume $\langle \neg \ tF \ t\text{-}P\text{-}Q \rangle$
then obtain $t\text{-}P\text{-}Q' \ r$ **where** $\langle tF \ t\text{-}P\text{-}Q' \rangle \ \langle t\text{-}P\text{-}Q = t\text{-}P\text{-}Q' \ @ \ [\checkmark(r)] \rangle$
by (*metis D-imp-front-tickFree* $\langle t\text{-}P\text{-}Q \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q) \rangle$ *butlast-snoc*
front-tickFree-iff-tickFree-butlast nonTickFree-n-frontTickFree)
moreover from $**(\checkmark 2, \checkmark 3) \ \langle \neg \ tF \ t\text{-}P\text{-}Q \rangle$ **obtain** $t\text{-}R' \ s$
where $\langle tF \ t\text{-}R' \rangle \ \langle t\text{-}R = t\text{-}R' \ @ \ [\checkmark(s)] \rangle$
by (*metis F-imp-front-tickFree butlast-snoc front-tickFree-iff-tickFree-butlast*
nonTickFree-n-frontTickFree setinterleaves_{ptick}-tickFree-imp)
ultimately obtain $r\text{-}s \ t_1' \text{ where } \langle t_1 = t_1' \ @ \ [\checkmark(r\text{-}s)] \rangle$
 $\langle t_1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 2)} ((t\text{-}P\text{-}Q', t\text{-}R'), S) \rangle$
using $**(\checkmark 3)$ **by** (*auto elim!: setinterleaves_{ptick}-snoc-tick-snoc-tickE*)
moreover have $\langle t\text{-}P\text{-}Q' \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q) \rangle$
by (*metis D-imp-front-tickFree* $\langle \neg \ tF \ t\text{-}P\text{-}Q \rangle \ \langle t\text{-}P\text{-}Q = t\text{-}P\text{-}Q' \ @ \ [\checkmark(r)] \rangle$
 $\langle t\text{-}P\text{-}Q \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q) \rangle$ *butlast-snoc div-butlast-when-non-tickFree-iff*)
moreover have $\langle t\text{-}R' \in \mathcal{T} \ R \rangle$
using $**(\checkmark 2) \ F\text{-}T \ \langle t\text{-}R = t\text{-}R' \ @ \ [\checkmark(s)] \rangle$ *is-processT3-TR-append* **by** *blast*
ultimately have $\langle t_1' \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q \llbracket S \rrbracket_{\checkmark 2} \ R) \rangle$
by (*simp add: Sync_{ptick2}.D-Sync_{ptick}*)
*(metis $**(\checkmark 3)$ D-imp-front-tickFree* $\langle tF \ t\text{-}R' \rangle \ \langle t\text{-}P\text{-}Q \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q) \rangle$
 $\langle t\text{-}R = t\text{-}R' \ @ \ [\checkmark(s)] \rangle$ *append.right-neutral butlast-snoc front-tickFree-charn*
front-tickFree-setinterleaves_{ptick}-iff tickFree-Nil tickFree-append-iff)
thus $\langle t_1 \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q \llbracket S \rrbracket_{\checkmark 2} \ R) \rangle$
by (*simp add: $\langle t_1 = t_1' \ @ \ [\checkmark(r\text{-}s)] \rangle$*)
*(metis $**(\checkmark 3)$ F-imp-front-tickFree* $\langle t_1 = t_1' \ @ \ [\checkmark(r\text{-}s)] \rangle$ *butlast-snoc*
div-butlast-when-non-tickFree-iff non-tickFree-tick tickFree-append-iff)
qed
with $\langle t_1 \notin \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark 1} \ Q \llbracket S \rrbracket_{\checkmark 2} \ R) \rangle$ **show** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle \dots$
next
case *fail*
from *setinterleaves_{ptick}-assoc-left[OF fail($\checkmark 3$) $**(\checkmark 3)$]* **obtain** $t\text{-}Q'$
where $*** : \langle ?\text{map-event } t_1 \text{ setinterleaves}_{\checkmark(\otimes\checkmark 3)} ((t\text{-}P, t\text{-}Q'), S) \rangle$
 $\langle t\text{-}Q' \text{ setinterleaves}_{\checkmark(\otimes\checkmark 4)} ((t\text{-}Q, t\text{-}R), S) \rangle$ **by** *blast*

```

from  $\ast(2) \ast\ast(2)$  fail(2)
have  $\langle t-Q', \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}) X-Q S X-R \rangle \in \mathcal{F} (Q \llbracket S \rrbracket_{\check{4}} R)$ 
  by (auto simp add: Syncptick4.F-Syncptick)
moreover have  $\langle t \text{ setinterleaves}_{\check{4}} (\otimes \check{3}) ((t-P, t-Q'), S) \rangle$ 
  by (simp add:  $\ast(1) \ast\ast(1)$ )
have  $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{3}) X-P S (\text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}))$ 
 $X-Q S X-R \rangle$ 
proof (rule subsetI)
  fix  $e$  assume  $\langle e \in X \rangle$ 
  show  $\langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{3}) X-P S (\text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}))$ 
 $X-Q S X-R \rangle$ 
proof (cases e)
  fix  $a$  assume  $\langle e = \text{ev } a \rangle$ 
  obtain  $a'$  where  $\langle \text{map-event}_{\text{ptick}} \text{id } \otimes \check{2} \Rightarrow \otimes \check{3} (\text{ev } a') = \text{ev } a \rangle$  by
simp
  with  $\langle e \in X \rangle$  have  $\langle \text{ev } a' \in \text{map-event}_{\text{ptick}} \text{id } \otimes \check{2} \Rightarrow \otimes \check{3} - ' X \rangle$ 
  by (simp add:  $\langle e = \text{ev } a \rangle$ )
  with  $\ast(4)[\text{THEN set-mp, OF this}]$  fail(4)
   $\langle \text{map-event}_{\text{ptick}} \text{id } \otimes \check{2} \Rightarrow \otimes \check{3} (\text{ev } a') = \text{ev } a \rangle$ 
  show  $\langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{3}) X-P S (\text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}))$ 
 $X-Q S X-R \rangle$ 
  by (auto simp add:  $\langle e = \text{ev } a \rangle$  subset-iff super-ref-Syncptick-def)
next
  fix  $r-s-t$  assume  $\langle e = \check{4}(r-s-t) \rangle$ 
  show  $\langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{3}) X-P S (\text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}))$ 
 $X-Q S X-R \rangle$ 
proof (cases  $\langle \exists r s t s-t. s \otimes \check{4} t = \lfloor s-t \rfloor \wedge r \otimes \check{3} s-t = \lfloor r-s-t \rfloor \rangle$ )
  assume  $\langle \exists r s t s-t. s \otimes \check{4} t = \lfloor s-t \rfloor \wedge r \otimes \check{3} s-t = \lfloor r-s-t \rfloor \rangle$ 
  then obtain  $r s t s-t$ 
  where  $\$ : \langle s \otimes \check{4} t = \lfloor s-t \rfloor \rangle \langle r \otimes \check{3} s-t = \lfloor r-s-t \rfloor \rangle$  by blast
  then obtain  $r' s' t' r-s'$ 
  where  $\$ \$ : \langle r' \otimes \check{1} s' = \lfloor r-s' \rfloor \rangle \langle r-s' \otimes \check{2} t' = \lfloor \otimes \check{3} \Rightarrow \otimes \check{2} r-s-t \rfloor \rangle$ 
  by (metis None-assms-tick-join(1,2) option.collapse option.discI
option.sel tick-assoc-ren-conv-hyp)
  have  $\langle \check{4}(\otimes \check{3} \Rightarrow \otimes \check{2} r-s-t) \in \text{map-event}_{\text{ptick}} \text{id } \otimes \check{2} \Rightarrow \otimes \check{3} - ' X \rangle$ 
  by (metis  $\langle e = \check{4}(r-s-t) \rangle \langle e \in X \rangle \$ \text{event}_{\text{ptick}}.\text{simps}(10)$ 
tick-assoc-ren-tick-assoc-ren-conv vimage-eq)
  from  $\ast(4)[\text{THEN set-mp, OF this}]$  fail(4)[THEN set-mp, of  $\langle \check{4}(r-s') \rangle$ ]
  show  $\langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{3}) X-P S (\text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}))$ 
 $X-Q S X-R \rangle$ 
  by (simp add:  $\langle e = \check{4}(r-s-t) \rangle$  subset-iff super-ref-Syncptick-def)
  (metis (no-types, lifting)  $\$ \$$  None-assms-tick-join(3,4))
  (Syncptick2.inj-tick-join option.collapse option.discI)
  (option.sel tick-assoc-ren-hyp tick-assoc-ren-tick-assoc-ren-conv)
next
  assume  $\langle \nexists r s t s-t. s \otimes \check{4} t = \lfloor s-t \rfloor \wedge r \otimes \check{3} s-t = \lfloor r-s-t \rfloor \rangle$ 
  thus  $\langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \check{3}) X-P S (\text{super-ref-Sync}_{\text{ptick}} (\otimes \check{4}))$ 
 $X-Q S X-R \rangle$ 
  by (simp add:  $\langle e = \check{4}(r-s-t) \rangle$  super-ref-Syncptick-def) blast

```

```

      qed
    qed
  qed
  ultimately show  $\langle t, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
    using  $*(1) *** (1) \text{ fail}(1)$  by (auto simp add:  $\text{Sync}_{ptick3}.F\text{-Sync}_{ptick}$ )
  qed
qed
qed
qed
end

```

lemma (in $\text{Sync}_{ptick}\text{-locale}$) *strict-ticks-of-Sync_{ptick}-subset* :

$$\langle \check{s}(P \llbracket S \rrbracket_{\check{\vee}} Q) \subseteq \{r-s \mid r-s \text{ } r \text{ } s. \text{ } r \otimes \check{\vee} s = \lfloor r-s \rfloor \wedge r \in \check{s}(P) \wedge s \in \check{s}(Q)\} \rangle \text{ (is } \langle - \subseteq \text{ ?S} \rangle)$$

proof (rule subsetI, elim strict-ticks-of-memE)

fix $t \text{ } r-s$ assume $\langle t @ \lfloor \check{\vee}(r-s) \rfloor \in \mathcal{T} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle \langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle$

from $\langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle$ have $\langle t @ \lfloor \check{\vee}(r-s) \rfloor \notin \mathcal{D} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle$ by (meson is-processT9)

with $\langle t @ \lfloor \check{\vee}(r-s) \rfloor \in \mathcal{T} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle$ obtain $t-P \text{ } t-Q$

where $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle \langle t @ \lfloor \check{\vee}(r-s) \rfloor \text{ setinterleaves}_{\check{\vee}(\otimes \check{\vee})} ((t-P, t-Q), S) \rangle$

unfolding $\text{Sync}_{ptick}\text{-projs}$ by blast

from this(3) show $\langle r-s \in \text{ ?S} \rangle$

proof (elim snoc-tick-setinterleaves_{ptick}E)

fix $t-P' \text{ } t-Q' \text{ } r \text{ } s$

assume $* : \langle r \otimes \check{\vee} s = \text{Some } r-s \rangle \langle t-P = t-P' @ \lfloor \check{\vee}(r) \rfloor \rangle \langle t-Q = t-Q' @ \lfloor \check{\vee}(s) \rfloor \rangle$

$\langle t \text{ setinterleaves}_{\check{\vee}(\otimes \check{\vee})} ((t-P', t-Q'), S) \rangle$

have $\langle t-P' \notin \mathcal{D} P \wedge t-Q' \notin \mathcal{D} Q \rangle$

proof (rule ccontr)

assume $\langle \neg (t-P' \notin \mathcal{D} P \wedge t-Q' \notin \mathcal{D} Q) \rangle$

with $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$ have $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle$

by (simp add: $D\text{-Sync}_{ptick} \text{ } *(2,3,4)$)

(metis $*(4)$ append.right-neutral append-T-imp-tickFree front-tickFree-Nil is-processT3-TR-append not-Cons-self2 setinterleaves_{ptick}-tickFree-imp)

with $\langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\check{\vee}} Q) \rangle$ show False ..

qed

with $*(2, 3) \langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$ have $\langle r \in \check{s}(P) \rangle \langle s \in \check{s}(Q) \rangle$

by (metis is-processT9 strict-ticks-of-memI)+

with $*(1)$ show $\langle r-s \in \text{ ?S} \rangle$ by blast

qed

qed

theorem (in $\text{Sync}_{ptick}\text{-assoc-locale}$) *Sync_{ptick}-assoc* :

$$\langle P \llbracket S \rrbracket_{\check{\vee}3} (Q \llbracket S \rrbracket_{\check{\vee}4} R) = \text{RenamingTick} (P \llbracket S \rrbracket_{\check{\vee}1} Q \llbracket S \rrbracket_{\check{\vee}2} R) \otimes \check{\vee} 2 \Rightarrow \otimes \check{\vee} 3 \rangle \text{ (is } \langle \text{ ?lhs} = \text{ ?rhs} \rangle)$$

```

proof (rule FD-antisym)
  show  $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$  by (fact Syncptick-assoc-oneside)
next
  from Syncptick-assoc-locale-sym.Syncptick-assoc-oneside[of R S Q P]
  have  $\langle \text{Sync}_{ptick2}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} R S$ 
     $(\text{Sync}_{ptick1}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} Q S P) \sqsubseteq_{FD}$ 
     $\text{RenamingTick} (\text{Sync}_{ptick3}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}$ 
     $(\text{Sync}_{ptick4}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} R S Q) S P) \otimes \check{3} \Rightarrow \otimes \check{2} \rangle$  .
  also have  $\langle \text{Sync}_{ptick1}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} Q S P = P \llbracket S \rrbracket_{\check{1}} Q \rangle$ 
    by (simp add: Syncptick1.Syncptick-sym)
  also have  $\langle \text{Sync}_{ptick2}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} R S P \text{-} Q = P \text{-} Q \llbracket S \rrbracket_{\check{2}} R \rangle$ 
for P-Q
    by (simp add: Syncptick2.Syncptick-sym)
  also have  $\langle \text{Sync}_{ptick4}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} R S Q = Q \llbracket S \rrbracket_{\check{4}} R \rangle$ 
    by (simp add: Syncptick4.Syncptick-sym)
  also have  $\langle \text{Sync}_{ptick3}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} Q \text{-} R S P = P \llbracket S \rrbracket_{\check{3}} Q \text{-} R \rangle$ 
for Q-R
    by (simp add: Syncptick3.Syncptick-sym)
  finally have  $\langle P \llbracket S \rrbracket_{\check{1}} Q \llbracket S \rrbracket_{\check{2}} R \sqsubseteq_{FD} \text{RenamingTick} ?lhs \otimes \check{3} \Rightarrow \otimes \check{2} \rangle$  .
  hence  $\langle ?rhs \sqsubseteq_{FD} \text{RenamingTick} (\text{RenamingTick} ?lhs \otimes \check{3} \Rightarrow \otimes \check{2}) \otimes \check{2} \Rightarrow \otimes \check{3} \rangle$ 
    by (fact mono-Renaming-FD)
  also have  $\langle \dots = \text{RenamingTick} ?lhs (\otimes \check{2} \Rightarrow \otimes \check{3} \circ \otimes \check{3} \Rightarrow \otimes \check{2}) \rangle$ 
    by (simp add: RenamingTick-comp)
  also have  $\langle \dots = \text{RenamingTick} ?lhs id \rangle$ 
proof (rule RenamingTick-is-restrictable-on-strict-ticks-of)
  fix r-s-t assume  $\langle r \text{-} s \text{-} t \in \check{s}(P \llbracket S \rrbracket_{\check{3}} (Q \llbracket S \rrbracket_{\check{4}} R)) \rangle$ 
  with Syncptick3.strict-ticks-of-Syncptick-subset obtain r s-t
    where  $\langle r \otimes \check{3} s \text{-} t = \lfloor r \text{-} s \text{-} t \rfloor \rangle \langle r \in \check{s}(P) \rangle \langle s \text{-} t \in \check{s}(Q \llbracket S \rrbracket_{\check{4}} R) \rangle$  by blast
  from this(3) Syncptick4.strict-ticks-of-Syncptick-subset obtain s t
    where  $\langle s \otimes \check{4} t = \lfloor s \text{-} t \rfloor \rangle \langle s \in \check{s}(Q) \rangle \langle t \in \check{s}(R) \rangle$  by blast
  from  $\langle r \otimes \check{3} s \text{-} t = \lfloor r \text{-} s \text{-} t \rfloor \rangle \langle s \otimes \check{4} t = \lfloor s \text{-} t \rfloor \rangle$ 
  show  $\langle (\otimes \check{2} \Rightarrow \otimes \check{3} \circ \otimes \check{3} \Rightarrow \otimes \check{2}) r \text{-} s \text{-} t = id r \text{-} s \text{-} t \rangle$ 
    by (auto intro!: tick-assoc-ren-tick-assoc-ren-conv)
qed
also have  $\langle \dots = ?lhs \rangle$  by simp
finally show  $\langle ?rhs \sqsubseteq_{FD} ?lhs \rangle$  .
qed

```


Chapter 7

First Laws

unbundle *option-type-syntax*

7.1 Behaviour with Constant Processes

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

lemma *Seq_{ptick}-const* [*simp*] : $\langle P ; \checkmark (\lambda r. Q) = P ; Q \rangle$
 — Very basic law.
by (*simp add: Process-eq-spec Seq_{ptick}-same-type-projs Seq-projs*)

7.1.1 The Laws of $\perp$

lemma *Seq_{ptick}-is-BOT-iff* : $\langle P ; \checkmark Q = \perp \longleftrightarrow P = \perp \vee (\exists r. [\checkmark(r)] \in \mathcal{T} P \wedge Q$
 $r = \perp) \rangle$
by (*simp add: BOT-iff-Nil-D Seq_{ptick}-projs*)

lemma *BOT-Seq_{ptick}* [*simp*] : $\langle \perp ; \checkmark P = \perp \rangle$ **by** (*simp add: Seq_{ptick}-is-BOT-iff*)

lemma (*in Sync_{ptick}-locale*) *Sync_{ptick}-is-BOT-iff* : $\langle P \llbracket S \rrbracket \checkmark Q = \perp \longleftrightarrow P = \perp$
 $\vee Q = \perp \rangle$
by (*simp add: BOT-iff-Nil-D D-Sync_{ptick}*)
(metis Nil-setinterleaves_{ptick} Nil-setinterleaving_{ptick}-Nil insertCI is-processT1-TR)

lemma (*in Sync_{ptick}-locale*) *Sync_{ptick}-BOT* [*simp*] : $\langle P \llbracket S \rrbracket \checkmark \perp = \perp \rangle$ **and** *BOT-Sync_{ptick}*
 $\llbracket S \rrbracket \checkmark Q = \perp \rangle$
by (*simp-all add: Sync_{ptick}-is-BOT-iff*)

7.1.2 The Laws of *STOP*

lemma *Seq_{ptick}-is-STOP-iff* :
 $\langle P ; \checkmark Q = \text{STOP} \longleftrightarrow \mathcal{T} P \subseteq \text{insert } \square \{[\checkmark(r)] \mid r. \text{True}\} \wedge$

$(\forall r. [\checkmark(r)] \in \mathcal{T} P \longrightarrow Q r = STOP) \rangle$ (is $\langle ?lhs \longleftrightarrow ?rhs \rangle$)

proof (intro iffI conjI subsetI allI impI)
 show $\langle ?lhs \Longrightarrow t \in \mathcal{T} P \Longrightarrow t \in insert [] \{[\checkmark(r)] \mid r. True\} \rangle$ for t
 by (simp add: STOP-iff-T Seq_{ptick}-projs set-eq-iff)
 (metis Prefix-Order.prefixI T-nonTickFree-imp-decomp append-Nil
 append-T-imp-tickFree is-processT3-TR length-0-conv length-map list.distinct(1))
 next
 show $\langle ?lhs \Longrightarrow [\checkmark(r)] \in \mathcal{T} P \Longrightarrow Q r = STOP \rangle$ for r
 by (force simp add: STOP-iff-T Seq_{ptick}-projs set-eq-iff)
 next
 show $\langle ?rhs \Longrightarrow P ; \checkmark Q = STOP \rangle$
 by (auto simp add: STOP-iff-T Seq_{ptick}-projs subset-iff)
 (metis D-T non-tickFree-tick,
 metis BOT-iff-Nil-D D-T D-BOT append-Nil event_{ptick}.distinct(1) mem-Collect-eq
 front-tickFree-single is-processT9 list.distinct(1) list.inject)
 qed

lemma Seq_{ptick}-is-STOP-iff-bis :
 $\langle P ; \checkmark Q = STOP \longleftrightarrow SKIPS \{r. [\checkmark(r)] \in \mathcal{T} P\} \sqsubseteq_{DT} P \wedge (\forall r. [\checkmark(r)] \in \mathcal{T} P \longrightarrow Q r = STOP) \rangle$
 (is $\langle ?lhs \longleftrightarrow ?rhs \rangle$)
proof (rule iffI)
 assume ?lhs
 from this[THEN arg-cong, where $f = \mathcal{D}$]
 have $\langle \mathcal{D} P = \{\} \rangle$
 by (simp add: Seq_{ptick}-projs D-STOP)
 (metis front-tickFree-Nil nonempty-divE[of P])
 with $\langle ?lhs \rangle$ show ?rhs
 by (subst (asm) Seq_{ptick}-is-STOP-iff)
 (auto simp add: refine-defs SKIPS-projs)
 next
 show $\langle ?rhs \Longrightarrow ?lhs \rangle$
 unfolding Seq_{ptick}-is-STOP-iff by (auto simp add: refine-defs SKIPS-projs)
 qed

corollary STOP-Seq_{ptick} [simp] : $\langle STOP ; \checkmark P = STOP \rangle$
 by (simp add: Seq_{ptick}-is-STOP-iff T-STOP)

lemma (in Sync_{ptick}-locale) STOP-Sync_{ptick}-STOP [simp] : $\langle STOP \llbracket S \rrbracket \checkmark STOP = STOP \rangle$
 by (simp add: STOP-iff-T T-Sync_{ptick} STOP-projs)

More powerful Laws lemma (in Sync_{ptick}-locale) Inter_{ptick}-STOP :

— Here, g is a free parameter.

$\langle P \llbracket \checkmark STOP = RenamingTick (P ; STOP) \rangle$
 $(\lambda r. the (tick-join r (g r))) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)

proof —

```

let ?f = ⟨λr. the (tick-join r (g r))⟩
have * : ⟨tF t ⟹ map (map-eventptick id ?f) t
           = map ev (map of-ev t)⟩ for t :: ⟨('a, 'r) traceptick⟩
  by (induct t, simp-all)
      (metis (no-types, lifting) eventptick.collapse(1) eventptick.simps(9) id-apply)
show ⟨?lhs = ?rhs⟩
proof (rule Process-eq-optimizedI)
  fix t assume ⟨t ∈ D ?lhs⟩
  then obtain u v t-P
    where ⟨t = u @ v⟩ ⟨tF v⟩ ⟨tF u ∨ v = []⟩
      ⟨u setinterleaves✓ tick-join ((t-P, []), { })⟩ ⟨t-P ∈ D P⟩
    unfolding D-Syncptick STOP-projs by blast
  from this(4) setinterleavesptick-NilR-iff
  have ⟨tF t-P⟩ ⟨u = map ev (map of-ev t-P)⟩ by auto

  from * ⟨tF t-P⟩ have ⟨u @ v = map (map-eventptick id ?f) t-P @ v⟩
    by (simp add: ⟨u = map ev (map of-ev t-P)⟩)
  moreover have ⟨t-P ∈ D (P ; STOP)⟩ by (simp add: D-Seq ⟨t-P ∈ D P⟩)
  ultimately show ⟨t ∈ D ?rhs⟩
    using ⟨tF v⟩ ⟨tF t-P⟩ by (auto simp add: D-Renaming ⟨t = u @ v⟩)
next
  fix t assume ⟨t ∈ D ?rhs⟩
  then obtain u v
    where $ : ⟨t = map (map-eventptick id ?f) u @ v⟩
      and ⟨tF u⟩ ⟨tF v⟩ ⟨u ∈ D P⟩ unfolding D-Renaming D-Seq D-STOP by
blast
  have ⟨tF (map (map-eventptick id ?f) u)⟩
    by (simp add: ⟨tF u⟩ map-eventptick-tickFree)
  moreover from ⟨tF u⟩
  have ⟨map (map-eventptick id ?f) u
        setinterleaves✓ tick-join ((u, []), { })⟩
  proof (induct u)
    case Nil show ?case by simp
  next
    case (Cons e u)
    obtain a where ⟨e = ev a⟩ ⟨tF u⟩ by (meson Cons.premis is-ev-def tick-
Free-Cons-iff)
    from Cons.hyps[OF ⟨tF u⟩] show ?case by (simp add: ⟨e = ev a⟩)
  qed
  ultimately show ⟨t ∈ D (P |||✓ STOP)⟩
    using ⟨tF v⟩ ⟨u ∈ D P⟩ by (auto simp add: D-Syncptick STOP-projs $)
next
  fix t X assume ⟨(t, X) ∈ F ?lhs⟩ ⟨t ∉ D ?lhs⟩
  then obtain t-P X-P X-Q
    where ⟨(t-P, X-P) ∈ F P⟩ ⟨t setinterleaves✓ tick-join ((t-P, []), { })⟩
      ⟨X ⊆ super-ref-Syncptick tick-join X-P { } X-Q⟩
    unfolding Syncptick-projs F-STOP by blast

  from ⟨X ⊆ super-ref-Syncptick tick-join X-P { } X-Q⟩

```

```

have $ : ⟨e ∈ map-eventptick id ?f - ' X ∪ range tick
    ⇒ e ∈ X-P ∪ range tick⟩ for e
  by (cases e) (auto simp add: super-ref-Syncptick-def)
from setinterleavesptick-NilR-iff
  [THEN iffD1, OF ⟨t setinterleaves✓tick-join ((t-P, []), { })⟩]
have ⟨tF t-P⟩ ⟨t = map ev (map of-ev t-P)⟩ by simp-all
have $$ : ⟨t = map (map-eventptick id ?f) t-P⟩
  by (simp add: * ⟨t = map ev (map of-ev t-P)⟩ ⟨tF t-P⟩)
show ⟨(t, X) ∈ F ?rhs⟩
proof (cases ⟨∃ r. t-P @ [✓(r)] ∈ T P⟩)
  show ⟨∃ r. t-P @ [✓(r)] ∈ T P ⇒ (t, X) ∈ F ?rhs⟩
    by (auto simp add: F-Renaming F-Seq F-STOP $$)
next
  assume ⟨∄ r. t-P @ [✓(r)] ∈ T P⟩
  hence ⟨(t-P, X-P ∪ range tick) ∈ F P⟩
    by (auto intro!: is-processT5 ⟨(t-P, X-P) ∈ F P⟩ F-T)
  with $ have ⟨(t-P, map-eventptick id ?f - ' X ∪ range tick) ∈ F P⟩
    by (meson is-processT4 subsetI)
  with ⟨tF t-P⟩ show ⟨(t, X) ∈ F ?rhs⟩ by (auto simp add: F-Renaming F-Seq
  $$)
qed
next
fix t X assume ⟨(t, X) ∈ F ?rhs⟩ ⟨t ∉ D ?rhs⟩
then obtain u where $ : ⟨t = map (map-eventptick id ?f) u⟩
  ⟨(u, map-eventptick id ?f - ' X) ∈ F (P ; STOP)⟩
  unfolding Renaming-projs by blast
from $(2) consider ⟨u ∈ D P⟩ | r where ⟨u @ [✓(r)] ∈ T P⟩
  | ⟨(u, map-eventptick id ?f - ' X ∪ range tick) ∈ F P⟩ ⟨tF u⟩
  by (auto simp add: Seq-projs F-STOP)
thus ⟨(t, X) ∈ F ?lhs⟩
proof cases
  let ?u' = ⟨if tF u then u else butlast u⟩
  let ?v' = ⟨if tF u then [] else [✓(?f (of-tick (last u)))⟩]
  assume ⟨u ∈ D P⟩
  hence ⟨?u' ∈ D P⟩ by (simp add: D-imp-front-tickFree div-butlast-when-non-tickFree-iff)
  moreover from D-imp-front-tickFree ⟨u ∈ D P⟩ front-tickFree-iff-tickFree-butlast
    have ⟨tF ?u'⟩ by auto
  moreover have ⟨ftF ?v'⟩ by simp
  moreover from ⟨tF ?u'⟩ have ⟨t = map (map-eventptick id ?f) ?u' @ ?v'⟩
    by (cases u rule: rev-cases, auto simp add: $(1) split: if-split-asm)
    (metis eventptick.collapse(2) eventptick.simps(10))
  ultimately have ⟨t ∈ D ?rhs⟩ unfolding D-Renaming D-Seq by blast
  with ⟨t ∉ D ?rhs⟩ show ⟨(t, X) ∈ F ?lhs⟩ ..
next
fix r assume ⟨u @ [✓(r)] ∈ T P⟩
hence ⟨(u, UNIV - {✓(r)}) ∈ F P⟩ by (simp add: is-processT6-TR)
moreover from $(1) * ⟨u @ [✓(r)] ∈ T P⟩ append-T-imp-tickFree
  have ⟨t setinterleaves✓tick-join ((u, []), { })⟩
  by (auto simp add: setinterleavesptick-NilR-iff)

```

moreover have $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } (UNIV - \{\checkmark(r)\}) \ \{\} \rangle$
 $UNIV \rangle$
by (*simp add: subset-iff super-ref-Sync_{ptick}-def*) (*metis event_{ptick}.exhaust*)
ultimately show $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$ **unfolding** $F\text{-Sync}_{ptick} \ F\text{-STOP}$ **by**
clarify blast
next
assume $\langle (u, \text{map-event}_{ptick} \ id \ ?f - 'X \cup \text{range tick}) \in \mathcal{F} \ P \rangle \langle tF \ u \rangle$
moreover from $\$(1) * \langle tF \ u \rangle$ **have** $\langle t \ \text{setinterleaves}_{\checkmark \text{ tick-join}} ((u, []), \{\}) \rangle$
by (*auto simp add: setinterleaves_{ptick}-NilR-iff*)
moreover have $\langle X \subseteq \text{super-ref-Sync}_{ptick} \ \text{tick-join } (\text{map-event}_{ptick} \ id \ ?f - 'X \cup \text{range tick}) \ \{\} \rangle$ $UNIV \rangle$
by (*simp add: subset-iff super-ref-Sync_{ptick}-def*) (*metis event_{ptick}.exhaust*)
ultimately show $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$ **unfolding** $F\text{-Sync}_{ptick} \ F\text{-STOP}$ **by**
clarify blast
qed
qed
qed

lemma (*in Sync_{ptick}-locale*) $STOP\text{-Inter}_{ptick} :$
 $\langle STOP \ ||_{\checkmark} \ Q = \text{RenamingTick } (Q ; STOP) \rangle$
 $(\lambda s. \text{ the } (\text{tick-join } (g \ s) \ s)) \rangle$
by (*metis Sync_{ptick}-locale-sym. Inter_{ptick}-STOP Sync_{ptick}-sym*)

lemma (*in Sync_{ptick}-locale*) $Par_{ptick}\text{-STOP} \ [simp] : \langle P \ ||_{\checkmark} \ STOP = (\text{if } P = \perp \text{ then } \perp \text{ else } STOP) \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle P = \perp \implies P \ ||_{\checkmark} \ STOP = \perp \rangle$ **by** *simp*
next
show $\langle P \neq \perp \implies P \ ||_{\checkmark} \ STOP = STOP \rangle$
by (*auto simp add: STOP-iff-T T-Sync_{ptick} STOP-projs set-eq-iff*
 $BOT\text{-iff-}Nil\text{-D setinterleaves}_{ptick}\text{-NilR-iff image-iff}$
 $(\text{metis event}_{ptick}.\text{collapse}(1) \ \text{last-in-set tickFree-butlast})+$)
qed

lemma (*in Sync_{ptick}-locale*) $STOP\text{-Par}_{ptick} \ [simp] : \langle STOP \ ||_{\checkmark} \ P = (\text{if } P = \perp \text{ then } \perp \text{ else } STOP) \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle P = \perp \implies STOP \ ||_{\checkmark} \ P = \perp \rangle$ **by** *simp*
next
show $\langle P \neq \perp \implies STOP \ ||_{\checkmark} \ P = STOP \rangle$
by (*auto simp add: STOP-iff-T T-Sync_{ptick} STOP-projs set-eq-iff*
 $BOT\text{-iff-}Nil\text{-D setinterleaves}_{ptick}\text{-NilL-iff image-iff}$
 $(\text{metis event}_{ptick}.\text{collapse}(1) \ \text{last-in-set tickFree-butlast})+$)
qed

7.1.3 The Laws of SKIP

Sequential Composition

SKIP is neutral for Seq_{ptick} !

lemma $SKIP\text{-}Seq_{ptick}$ [simp] : $\langle SKIP\ r ;\checkmark P = P\ r \rangle$
 by (simp add: Process-eq-spec Seq_{ptick}-projs ref-Seq_{ptick}-def SKIP-projs)

lemma $Seq_{ptick}\text{-}SKIP$ [simp] : $\langle P ;\checkmark SKIP = P \rangle$

proof (subst Process-eq-spec-optimized, safe)

show $\langle s \in \mathcal{D}\ (P ;\checkmark SKIP) \implies s \in \mathcal{D}\ P \rangle$
 and $\langle s \in \mathcal{D}\ P \implies s \in \mathcal{D}\ (P ;\checkmark SKIP) \rangle$ for s
 by (simp-all add: D-Seq_{ptick}-same-type D-SKIP)

next

show $\langle (s, X) \in \mathcal{F}\ (P ;\checkmark SKIP) \implies (s, X) \in \mathcal{F}\ P \rangle$ for $s\ X$
 by (auto simp add: F-Seq_{ptick}-same-type F-SKIP is-processT6-TR-notin tick-T-F
 intro : is-processT4 is-processT8)

next

show $\langle (s, X) \in \mathcal{F}\ P \implies (s, X) \in \mathcal{F}\ (P ;\checkmark SKIP) \rangle$ for $s\ X$
 by (simp add: F-Seq_{ptick}-same-type F-SKIP)
 (metis (mono-tags, opaque-lifting) F-T T-nonTickFree-imp-decomp
 append.right-neutral f-inv-into-f is-processT5-S7')

qed

lemma $SKIPS\text{-}Seq_{ptick}$ [simp] : $\langle SKIPS\ R ;\checkmark P = \sqcap r \in R. P\ r \rangle$
 by (auto simp add: Process-eq-spec GlobalNdet-projs Seq_{ptick}-projs
 STOP-projs SKIPS-projs ref-Seq_{ptick}-def)

lemma $finite\text{-}ticks\text{-}Seq_{ptick}$ [finite-ticks-simps] : $\langle \mathbb{F}_{\checkmark}(P ;\checkmark Q) \rangle$

if $\langle \mathbb{F}_{\checkmark}(P) \rangle$ and $\langle (\bigwedge r. r \in \checkmark s(P) \implies \mathbb{F}_{\checkmark}(Q\ r)) \rangle$

proof (rule finite-ticksI)

fix t assume $\langle t \in \mathcal{T}\ (P ;\checkmark Q) \rangle \langle t \notin \mathcal{D}\ (P ;\checkmark Q) \rangle$

have $\langle \{r'. t @ [\checkmark(r')] \in \mathcal{T}\ (P ;\checkmark Q)\} \subseteq$

$(\bigcup u \in \{u. \exists v\ r. t = \text{map}\ (ev \circ \text{of-ev})\ u @ v \wedge u @ [\checkmark(r)] \in \mathcal{T}\ P \wedge u \notin \mathcal{D}\ P \wedge tF\ u \wedge ftF\ v\}.$

$\bigcup r \in \{r. u @ [\checkmark(r)] \in \mathcal{T}\ P\}. \{s. \text{drop}\ (\text{length}\ u)\ t @ [\checkmark(s)] \in \mathcal{T}\ (Q\ r)\} \rangle$
 (is $\langle ?lhs \subseteq ?rhs \rangle$)

proof (rule subsetI)

fix r' assume $\langle r' \in ?lhs \rangle$

hence $\langle t @ [\checkmark(r')] \in \mathcal{T}\ (P ;\checkmark Q) \rangle$ by simp

with $\langle t \notin \mathcal{D}\ (P ;\checkmark Q) \rangle$ obtain $t'\ r\ u'$

where $\langle t @ [\checkmark(r')] = \text{map}\ (ev \circ \text{of-ev})\ t' @ u' \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T}\ P \rangle \langle t' \notin \mathcal{D}\ P \rangle \langle tF\ t' \rangle \langle u' \in \mathcal{T}\ (Q\ r) \rangle$

by (auto simp add: Seq_{ptick}-projs)

(metis non-tickFree-tick tickFree-append-iff tickFree-map-ev-comp,
 metis (no-types, opaque-lifting) T-imp-front-tickFree butlast-append but-
 last-snoc

front-tickFree-iff-tickFree-butlast non-tickFree-tick tickFree-append-iff

```

    tickFree-imp-front-tickFree tickFree-map-ev-comp,
    metis (no-types, opaque-lifting) append.assoc butlast-snoc front-tickFree-charn
non-tickFree-tick
    tickFree-Nil tickFree-append-iff tickFree-imp-front-tickFree tick-
Free-map-ev-comp)

thus  $\langle r' \in ?rhs \rangle$ 
apply auto
by (smt (verit, ccfv-SIG) Prefix-Order.same-prefix-nil T-imp-front-tickFree
append-eq-append-conv2 append-eq-append-conv-if append-eq-conv-conj
append-eq-first-pref-spec append-same-eq eventptick.disc(2) front-tickFree-dw-closed
length-map tickFree-Cons-iff tickFree-append-iff
tickFree-map-ev-comp)
qed
moreover have  $\langle finite \dots \rangle$ 
proof (rule finite-UN-I)
  show  $\langle finite \{u. \exists v r. t = map (ev \circ of-ev) u @ v \wedge u @ [\checkmark(r)] \in \mathcal{T} P \wedge u \notin \mathcal{D} P \wedge tF u \wedge ftF v\} \rangle$ 
  by (rule finite-subset[of -  $\langle \{u. u \leq map (ev \circ of-ev) (if tF t then t else butlast t)\} \rangle$ ])
    (auto simp add: append-T-imp-tickFree tickFree-map-ev-of-ev-same-type-is
prefixes-fin,
metis prefixI append-T-imp-tickFree butlast-append map-append
not-Cons-self2 tickFree-Nil tickFree-map-ev-of-ev-eq-iff)
next
  fix u assume  $\langle u \in \{u. \exists v r. t = map (ev \circ of-ev) u @ v \wedge u @ [\checkmark(r)] \in \mathcal{T} P \wedge u \notin \mathcal{D} P \wedge tF u \wedge ftF v\} \rangle$ 
  then obtain r v where  $\langle t = map (ev \circ of-ev) u @ v \rangle \langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle u \notin \mathcal{D} P \rangle \langle tF u \rangle \langle ftF v \rangle$  by blast
  show  $\langle finite (\bigcup_{r \in \{r. u @ [\checkmark(r)] \in \mathcal{T} P\}} \{s. drop (length u) t @ [\checkmark(s)] \in \mathcal{T} (Q r)\}) \rangle$ 
  proof (rule finite-UN-I)
    show  $\langle finite \{r. u @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$ 
    by (simp add:  $\langle F_{\checkmark}(P) \rangle \langle u \notin \mathcal{D} P \rangle$  finite-ticksD)
  next
    fix r assume  $\langle r \in \{r. u @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$ 
    hence  $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$  ..
    with  $\langle t \notin \mathcal{D} (P ;_{\checkmark} Q) \rangle \langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF u \rangle$  have  $\langle v \notin \mathcal{D} (Q r) \rangle$ 
    by (auto simp add:  $\langle t = map (ev \circ of-ev) u @ v \rangle Seq_{ptick-projs}$ )
    show  $\langle finite \{s. drop (length u) t @ [\checkmark(s)] \in \mathcal{T} (Q r)\} \rangle$ 
    by (simp add:  $\langle t = map (ev \circ of-ev) u @ v \rangle$ 
      (metis  $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle u \notin \mathcal{D} P \rangle \langle v \notin \mathcal{D} (Q r) \rangle$  finite-ticksD
is-processT9 strict-ticks-of-memI  $\langle (\bigwedge r. r \in \checkmark s(P) \implies F_{\checkmark}(Q r) \rangle$ ))
  qed
qed
ultimately show  $\langle finite \{r. t @ [\checkmark(r)] \in \mathcal{T} (P ;_{\checkmark} Q) \} \rangle$  by (fact finite-subset)
qed

```

lemma *finite-ticks-fun-Seq_{ptick}-bis* :
 $\langle \mathbb{F}_{\checkmark \Rightarrow}(f) \Rightarrow (\bigwedge x r. r \in \checkmark s(f x) \Rightarrow \mathbb{F}_{\checkmark}(x) \Rightarrow \mathbb{F}_{\checkmark}(g x r)) \Rightarrow \mathbb{F}_{\checkmark \Rightarrow}(\lambda x. f x ; \checkmark g x) \rangle$
by (*simp add: finite-ticks-fun-def finite-ticks-Seq_{ptick}*)

lemma *finite-ticks-fun-Seq_{ptick} [finite-ticks-fun-simps]* :
— Big approximation.
 $\langle \mathbb{F}_{\checkmark \Rightarrow}(f) \Rightarrow (\bigwedge r. r \in (\bigcup x. \checkmark s(f x)) \Rightarrow \mathbb{F}_{\checkmark \Rightarrow}(\lambda x. g x r)) \Rightarrow \mathbb{F}_{\checkmark \Rightarrow}(\lambda x. f x ; \checkmark g x) \rangle$
by (*rule finite-ticks-fun-Seq_{ptick}-bis*)
(*auto simp add: finite-ticks-fun-def*)

Synchronization Product

The generalization of the synchronization product was essentially motivated by the following theorem (in comparison to $SKIP\ r\ \llbracket A \rrbracket\ SKIP\ s = (\text{if } r = s \text{ then } SKIP\ r \text{ else } STOP)$).

theorem (*in Sync_{ptick}-locale*) *SKIP-Sync_{ptick}-SKIP [simp]* :
 $\langle SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s = (\text{case tick-join } r\ s \text{ of } \lfloor r-s \rfloor \Rightarrow SKIP\ r-s \mid \Diamond \Rightarrow STOP) \rangle$
proof (*split option.split, intro conjI impI allI*)
show $\langle \text{tick-join } r\ s = \Diamond \Rightarrow SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s = STOP \rangle$
unfolding *STOP-iff-T T-Sync_{ptick} SKIP-projs set-eq-iff*
by (*safe, simp-all, metis Nil-setinterleaving_{ptick}-Nil insertCI*)
next
show $\langle SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s = SKIP\ r-s \rangle$ **if** $\langle \text{tick-join } r\ s = \lfloor r-s \rfloor \rangle$ **for** $r-s$
proof (*rule Process-eq-optimizedI*)
show $\langle t \in \mathcal{D}\ (SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s) \Rightarrow t \in \mathcal{D}\ (SKIP\ r-s) \rangle$ **for** t
by (*simp add: D-Sync_{ptick} SKIP-projs*)
next
show $\langle t \in \mathcal{D}\ (SKIP\ r-s) \Rightarrow t \in \mathcal{D}\ (SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s) \rangle$ **for** t
by (*simp add: D-Sync_{ptick} SKIP-projs*)
next
fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F}\ (SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s) \rangle$ $\langle t \notin \mathcal{D}\ (SKIP\ r\ \llbracket A \rrbracket_{\checkmark}\ SKIP\ s) \rangle$
then obtain $t-P\ t-Q\ X-P\ X-Q$
where *fail*: $\langle (t-P, X-P) \in \mathcal{F}\ (SKIP\ r) \rangle \langle (t-Q, X-Q) \in \mathcal{F}\ (SKIP\ s) \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P, t-Q), A) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{ tick-join } X-P\ A\ X-Q \rangle$
unfolding *Sync_{ptick}-projs* **by** *blast*
from *fail(1-3)* **consider** $\langle t = [] \rangle \langle \checkmark(r) \notin X-P \rangle \langle \checkmark(s) \notin X-Q \rangle \mid \langle t = [\checkmark(r-s)] \rangle$
by (*cases t-P; cases t-Q*) (*simp-all add: F-SKIP tick-join r s = floor(r-s)*)
thus $\langle (t, X) \in \mathcal{F}\ (SKIP\ r-s) \rangle$
proof *cases*
assume $\langle t = [] \rangle \langle \checkmark(r) \notin X-P \rangle \langle \checkmark(s) \notin X-Q \rangle$
from $\langle \checkmark(r) \notin X-P \rangle \langle \checkmark(s) \notin X-Q \rangle$ *fail(4)* $\langle \text{tick-join } r\ s = \lfloor r-s \rfloor \rangle$
have $\langle \checkmark(r-s) \notin X \rangle$
by (*simp add: super-ref-Sync_{ptick}-def subset-iff*)
(*metis event_{ptick}.distinct(1) event_{ptick}.inject(2) inj-tick-join*)


```

    with  $\langle t = [] \rangle$  show  $\langle (t, X) \in \mathcal{F} (SKIP\ r\ s) \rangle$  by (simp add: F-SKIP)
  next
    show  $\langle t = [\checkmark(r-s)] \implies (t, X) \in \mathcal{F} (SKIP\ r\ s) \rangle$  by (simp add: F-SKIP)
  qed
next
  fix  $t :: \langle ('a, -) trace_{ptick} \rangle$  and  $X$  assume  $\langle (t, X) \in \mathcal{F} (SKIP\ r\ s) \rangle$ 
  then consider  $\langle t = [] \rangle \langle \checkmark(r-s) \notin X \rangle \mid \langle t = [\checkmark(r-s)] \rangle$ 
    unfolding F-SKIP by blast
  thus  $\langle (t, X) \in \mathcal{F} (SKIP\ r\ [A]_{\checkmark}\ SKIP\ s) \rangle$ 
  proof cases
    assume  $\langle t = [] \rangle \langle \checkmark(r-s) \notin X \rangle$ 
    have  $\langle ([], - \{ \checkmark(r) \}) \in \mathcal{F} (SKIP\ r) \rangle$ 
      by (simp add: F-SKIP)
    moreover have  $\langle ([], - \{ \checkmark(s) \}) \in \mathcal{F} (SKIP\ s) \rangle$ 
      by (simp add: F-SKIP)
    moreover have  $\langle t\ setinterleaves_{\checkmark tick-join} ([], []), A \rangle$ 
      by (simp add:  $\langle t = [] \rangle$ )
    moreover have  $\langle X \subseteq super-ref-Sync_{ptick}\ tick-join (- \{ \checkmark(r) \})\ A\ (- \{ \checkmark(s) \}) \rangle$ 
      using  $\langle \checkmark(r-s) \notin X \rangle \langle tick-join\ r\ s = \lfloor r-s \rfloor \rangle$ 
      by (simp add: super-ref-Sync_{ptick}-def subset-iff)
      (metis event_{ptick}.exhaust option.inject)
    ultimately show  $\langle (t, X) \in \mathcal{F} (SKIP\ r\ [A]_{\checkmark}\ SKIP\ s) \rangle$ 
      by (simp (no-asm) add: F-Sync_{ptick}) blast
  next
    assume  $\langle t = [\checkmark(r-s)] \rangle$ 
    have  $\langle ([\checkmark(r)], UNIV) \in \mathcal{F} (SKIP\ r) \rangle$ 
      by (simp add: F-SKIP)
    moreover have  $\langle ([\checkmark(s)], UNIV) \in \mathcal{F} (SKIP\ s) \rangle$ 
      by (simp add: F-SKIP)
    moreover have  $\langle t\ setinterleaves_{\checkmark tick-join} ([\checkmark(r)], [\checkmark(s)]), A \rangle$ 
      by (simp add:  $\langle t = [\checkmark(r-s)] \rangle \langle tick-join\ r\ s = \lfloor r-s \rfloor \rangle$ )
    moreover have  $\langle X \subseteq super-ref-Sync_{ptick}\ tick-join\ UNIV\ A\ UNIV \rangle$ 
      by (simp add: super-ref-Sync_{ptick}-def subset-iff)
      (metis event_{ptick}.exhaust)
    ultimately show  $\langle (t, X) \in \mathcal{F} (SKIP\ r\ [A]_{\checkmark}\ SKIP\ s) \rangle$ 
      by (simp (no-asm) add: F-Sync_{ptick}) blast
  qed
qed
qed

```

```

lemma (in Sync_{ptick}-locale) STOP-Sync_{ptick}-SKIP [simp] :  $\langle STOP\ [A]_{\checkmark}\ SKIP\ s = STOP \rangle$ 
  and SKIP-Sync_{ptick}-STOP [simp] :  $\langle SKIP\ r\ [A]_{\checkmark}\ STOP = STOP \rangle$ 
  by (force simp add: STOP-iff-T T-Sync_{ptick} STOP-projs SKIP-projs)+

```

```

lemma (in Sync_{ptick}-locale) Mprefix-Sync_{ptick}-SKIP :

```

$\langle \Box a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r =$
 $\Box a \in (A - S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)
proof (rule *Process-eq-optimizedI*)
fix t **assume** $\langle t \in \mathcal{D} \ ?lhs \rangle$
then obtain $u \ v \ t\text{-}P \ a \ t\text{-}Q$
where $*$: $\langle t = u \ @ \ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle \langle a \in A \rangle \langle t\text{-}P \in \mathcal{D} \ (P \ a) \rangle$
 $\langle u \ \text{setinterleaves}_{\checkmark}(\otimes \checkmark) ((ev \ a \ \# \ t\text{-}P, t\text{-}Q), S) \rangle \langle t\text{-}Q = [] \vee t\text{-}Q = [\checkmark(r)] \rangle$
unfolding $D\text{-}Sync_{ptick} \text{ SKIP-projs } M\text{prefix-projs}$ **by** *blast*
from $*(6, 7)$ **obtain** u'
where $\langle a \notin S \rangle \langle u = ev \ a \ \# \ u' \rangle \langle u' \ \text{setinterleaves}_{\checkmark}(\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle$
by (*auto split: if-split-asm*)
from $this(2, 3) \ *(2, 5, 7) \ \text{front-tickFree-charn}$ **have** $\langle u' \in \mathcal{D} \ (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r) \rangle$
by (*auto simp add: D-Sync_{ptick} SKIP-projs*)
with $*(1-4) \ \langle a \notin S \rangle \langle u = ev \ a \ \# \ u' \rangle$ *is-processT7* **show** $\langle t \in \mathcal{D} \ ?rhs \rangle$
by (*auto simp add: D-Mprefix*)
next
fix t **assume** $\langle t \in \mathcal{D} \ ?rhs \rangle$
then obtain $a \ t'$ **where** $\langle t = ev \ a \ \# \ t' \rangle \langle a \in A \rangle \langle a \notin S \rangle \langle t' \in \mathcal{D} \ (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r) \rangle$
unfolding $D\text{-}Mprefix$ **by** *blast*
then obtain $u \ v \ t\text{-}P \ t\text{-}Q$
where $*$: $\langle t' = u \ @ \ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle \langle t\text{-}P \in \mathcal{D} \ (P \ a) \rangle$
 $\langle u \ \text{setinterleaves}_{\checkmark}(\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle \langle t\text{-}Q = [] \vee t\text{-}Q = [\checkmark(r)] \rangle$
unfolding $D\text{-}Sync_{ptick} \text{ SKIP-projs}$ **by** *blast*
have $\langle t = (ev \ a \ \# \ u) \ @ \ v \rangle$ **by** (*simp add: *(1) $\langle t = ev \ a \ \# \ t' \rangle$*)
moreover from $*(2)$ **have** $\langle tF \ (ev \ a \ \# \ u) \rangle$ **by** *simp*
moreover from $*(5, 6)$ **have** $\langle ev \ a \ \# \ u \ \text{setinterleaves}_{\checkmark}(\otimes \checkmark) ((ev \ a \ \# \ t\text{-}P, t\text{-}Q), S) \rangle$
by (*cases t-Q (simp-all add: $\langle a \notin S \rangle \text{ Cons-setinterleaving}_{ptick}\text{-Cons}$)*)
moreover have $\langle ev \ a \ \# \ t\text{-}P \in \mathcal{D} \ (\Box a \in A \rightarrow P \ a) \rangle$
by (*simp add: *(4) D-Mprefix $\langle a \in A \rangle$*)
ultimately show $\langle t \in \mathcal{D} \ ?lhs \rangle$
unfolding $D\text{-}Sync_{ptick} \text{ SKIP-projs}$ **using** $*(3, 6)$ **by** *blast*
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle \langle t \notin \mathcal{D} \ ?lhs \rangle$
then obtain $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$
where $*$: $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ (\Box a \in A \rightarrow P \ a) \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ (\text{SKIP } r) \rangle$
 $\langle t \ \text{setinterleaves}_{\checkmark}(\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X\text{-}P \ S \ X\text{-}Q \rangle$
unfolding $Sync_{ptick}\text{-projs}$ **by** *blast*
from $*(1)$ **consider** $\langle t\text{-}P = [] \rangle \langle X\text{-}P \cap ev \ 'A = \{ \} \rangle$
 $| \ a \ t\text{-}P'$ **where** $\langle t\text{-}P = ev \ a \ \# \ t\text{-}P' \rangle \langle a \in A \rangle \langle (t\text{-}P', X\text{-}P) \in \mathcal{F} \ (P \ a) \rangle$
unfolding $F\text{-}Mprefix$ **by** *blast*
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
proof *cases*
assume $\langle t\text{-}P = [] \rangle \langle X\text{-}P \cap ev \ 'A = \{ \} \rangle$
from $\langle t\text{-}P = [] \rangle \ *(2, 3)$ **have** $\langle t = [] \rangle \langle t\text{-}Q = [] \rangle \langle \checkmark(r) \notin X\text{-}Q \rangle$

by (auto simp add: F-SKIP)
 with $\ast(4)$ show $\langle t-P = [] \implies X-P \cap \text{ev } A = \{\} \implies (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$
 by (auto simp add: F-Mprefix super-ref-Sync_{ptick}-def)
 next
 fix $a \ t-P'$ assume $\langle t-P = \text{ev } a \# t-P' \rangle \langle a \in A \rangle \langle (t-P', X-P) \in \mathcal{F} (P \ a) \rangle$
 from $\ast(2, 3)$ obtain t'
 where $\langle t = \text{ev } a \# t' \rangle \langle a \notin S \rangle \langle t' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P', t-Q), S) \rangle$
 by (auto simp add: $\langle t-P = \text{ev } a \# t-P' \rangle$ F-SKIP split: if-split-asm)
 from $\ast(2, 4) \langle (t-P', X-P) \in \mathcal{F} (P \ a) \rangle$ this(3)
 have $\langle (t', X) \in \mathcal{F} (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r) \rangle$ by (auto simp add: F-Sync_{ptick})
 thus $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$ by (simp add: $\langle t = \text{ev } a \# t' \rangle$ F-Mprefix $\langle a \in A \rangle \langle a \notin S \rangle$)
 qed
 next
 fix $t \ X$ assume $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle \langle t \notin \mathcal{D} \text{ ?rhs} \rangle$
 from $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$ consider $\langle t = [] \rangle \langle X \cap \text{ev } (A - S) = \{\} \rangle$
 | $a \ t'$ where $\langle t = \text{ev } a \# t' \rangle \langle a \in A \rangle \langle a \notin S \rangle \langle (t', X) \in \mathcal{F} (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r) \rangle$
 unfolding F-Mprefix by blast
 thus $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$
 proof cases
 assume $\langle t = [] \rangle \langle X \cap \text{ev } (A - S) = \{\} \rangle$
 from $\langle X \cap \text{ev } (A - S) = \{\} \rangle$
 have $\langle ([], \text{range tick} \cup \{\text{ev } a \mid a. \text{ev } a \in X \wedge a \notin S\}) \in \mathcal{F} (\Box a \in A \rightarrow P \ a) \rangle$
 by (auto simp add: F-Mprefix)
 moreover have $\langle ([], \text{UNIV} - \{\checkmark(r)\}) \in \mathcal{F} (\text{SKIP } r) \rangle$ by (simp add: F-SKIP)
 moreover have $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\otimes\checkmark) (\text{range tick} \cup \{\text{ev } a \mid a. \text{ev } a \in X \wedge a \notin S\}) S (\text{UNIV} - \{\checkmark(r)\}) \rangle$
 by (simp add: subset-iff super-ref-Sync_{ptick}-def) (metis event_{ptick}.exhaust)
 moreover have $\langle [] \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ([], [], S) \rangle$ by simp
 ultimately show $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ by (simp (no-asm) add: $\langle t = [] \rangle$ F-Sync_{ptick})
 blast
 next
 fix $a \ t'$ assume $\langle t = \text{ev } a \# t' \rangle \langle a \in A \rangle \langle a \notin S \rangle \langle (t', X) \in \mathcal{F} (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r) \rangle$
 from this(1, 4) $\langle t \notin \mathcal{D} \text{ ?rhs} \rangle \langle a \in A \rangle \langle a \notin S \rangle$
 obtain $t-P \ t-Q \ X-P \ X-Q$
 where $\ast : \langle (t-P, X-P) \in \mathcal{F} (P \ a) \rangle \langle (t-Q, X-Q) \in \mathcal{F} (\text{SKIP } r) \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\otimes\checkmark) X-P \ S \ X-Q \rangle$
 unfolding D-Mprefix Sync_{ptick}-projs by force
 have $\langle (\text{ev } a \# t-P, X-P) \in \mathcal{F} (\Box a \in A \rightarrow P \ a) \rangle$
 by (simp add: $\ast(1)$ F-Mprefix $\langle a \in A \rangle$)
 moreover from $\ast(2, 3)$ have $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((\text{ev } a \# t-P, t-Q), S) \rangle$
 by (auto simp add: $\langle t = \text{ev } a \# t' \rangle$ F-SKIP $\langle a \notin S \rangle$)
 ultimately show $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ unfolding F-Sync_{ptick} using $\ast(2, 4)$ by
 auto
 qed
 qed

corollary (in $\text{Sync}_{ptick}\text{-locale}$) $\text{SKIP-Sync}_{ptick}\text{-Mprefix}$:
 $\langle \text{SKIP } r \llbracket S \rrbracket_{\checkmark} \square b \in B \rightarrow Q \ b = \square b \in (B - S) \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)
by (subst (1 2) $\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}\text{-sym mono-Mprefix-eq}$)
(fact $\text{Sync}_{ptick}\text{-locale-sym.Mprefix-Sync}_{ptick}\text{-SKIP}$)

lemma (in $\text{Sync}_{ptick}\text{-locale}$) $\text{finite-ticks-Sync}_{ptick}$ [$\text{finite-ticks-simps}$] :

$\langle \mathbb{F}_{\checkmark}(P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ if $\langle \mathbb{F}_{\checkmark}(P) \rangle$ and $\langle \mathbb{F}_{\checkmark}(Q) \rangle$
proof (rule finite-ticksI)
fix t **assume** $\langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
have $\langle \{r-s. t @ [\checkmark(r-s)] \in \mathcal{T} (P \llbracket S \rrbracket_{\checkmark} Q) \} \subseteq$
 $(\bigcup (t-P, t-Q) \in \{(t-P, t-Q). t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), S)\}).$
 $\{r-s. \exists r s. r \otimes \checkmark s = \lfloor r-s \rfloor \wedge$
 $t-P @ [\checkmark(r)] \in \mathcal{T} P \wedge t-P \notin \mathcal{D} P \wedge$
 $t-Q @ [\checkmark(s)] \in \mathcal{T} Q \wedge t-Q \notin \mathcal{D} Q\} \rangle$
(is $\langle - \subseteq ?rhs \rangle$)
proof (rule subsetI)
fix $r-s$ **assume** $\langle r-s. t @ [\checkmark(r-s)] \in \mathcal{T} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
hence $\langle t @ [\checkmark(r-s)] \in \mathcal{T} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$..
moreover from $\langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **have** $\langle t @ [\checkmark(r-s)] \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (meson is-processT9)
ultimately obtain $t-P \ t-Q$ **where** $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle \langle t-P \notin \mathcal{D} P \rangle$
 $\langle t-Q \notin \mathcal{D} Q \rangle$
 $\langle t @ [\checkmark(r-s)] \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), S) \rangle$
by (simp add: $T\text{-Sync}_{ptick} \ D\text{-Sync}_{ptick}$) (use $\text{front-tickFree-Nil}$ in blast)
from this(5) show $\langle r-s \in ?rhs \rangle$
proof (elim $\text{snoc-tick-setinterleaves}_{ptick} E$)
fix $r \ s \ t-P' \ t-Q'$
assume $\text{assms} : \langle t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P', t-Q'), S) \rangle$
 $\langle r \otimes \checkmark s = \lfloor r-s \rfloor \rangle \langle t-P = t-P' @ [\checkmark(r)] \rangle \langle t-Q = t-Q' @ [\checkmark(s)] \rangle$
from $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle \langle t-P \notin \mathcal{D} P \rangle \langle t-Q \notin \mathcal{D} Q \rangle$ $\text{assms}(3, 4)$
have $\langle t-P' \notin \mathcal{D} P \rangle \langle t-Q' \notin \mathcal{D} Q \rangle$
by (meson $T\text{-imp-front-tickFree front-tickFree-append-iff is-processT7 not-Cons-self2}$) +
with $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$ assms **show** $\langle r-s \in ?rhs \rangle$ **by auto**
qed
qed
moreover have $\langle \text{finite} \dots \rangle$
proof (rule finite-UN-I, safe)
show $\langle \text{finite} \{(t-P, t-Q). t \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), S)\} \rangle$
by (fact $\text{finite-setinterleaves}_{ptick}\text{-tick-join}$)
next
fix $t-P \ t-Q$
let $?S = \langle \{r-s. \exists r s. r \otimes \checkmark s = \lfloor r-s \rfloor \wedge$
 $t-P @ [\checkmark(r)] \in \mathcal{T} P \wedge t-P \notin \mathcal{D} P \wedge$
 $t-Q @ [\checkmark(s)] \in \mathcal{T} Q \wedge t-Q \notin \mathcal{D} Q\} \rangle$
have $\langle \text{Some } ' ?S \subseteq (\lambda(r, s). r \otimes \checkmark s) '$
 $(\{r. t-P @ [\checkmark(r)] \in \mathcal{T} P \wedge t-P \notin \mathcal{D} P\} \times$

$\{s. t.Q @ [\checkmark(s)] \in \mathcal{T} Q \wedge t.Q \notin \mathcal{D} Q\}$ by *force*
moreover have $\langle \text{finite } \dots \rangle$ by *(simp add: finite-ticksD $\langle \mathbb{F}_{\checkmark}(P) \rangle \langle \mathbb{F}_{\checkmark}(Q) \rangle$)*
ultimately have $\langle \text{finite } (\text{Some } ' ?S) \rangle$ by *(fact finite-subset)*
thus $\langle \text{finite } ?S \rangle$ by *(simp add: finite-image-iff)*
qed
ultimately show $\langle \text{finite } \{r.s. t @ [\checkmark(r-s)] \in \mathcal{T} (P \llbracket S \rrbracket_{\checkmark} Q) \} \rangle$ by *(fact finite-subset)*
qed

lemma (in *Sync_{ptick}-locale*) *finite-ticks-fun-Sync_{ptick} [finite-ticks-fun-simps]* :
 $\langle \mathbb{F}_{\checkmark \Rightarrow}(f) \rangle \implies \langle \mathbb{F}_{\checkmark \Rightarrow}(g) \rangle \implies \langle \mathbb{F}_{\checkmark \Rightarrow}(\lambda x. f x \llbracket S \rrbracket_{\checkmark} g x) \rangle$
by *(simp add: finite-ticks-fun-def finite-ticks-Sync_{ptick})*

7.2 Associativity of Sequential Composition

lemma *Seq_{ptick}-assoc* : $\langle P ;_{\checkmark} (\lambda r. Q r ;_{\checkmark} R) \rangle = \langle P ;_{\checkmark} Q ;_{\checkmark} R \rangle$
for $P :: \langle ('a, 'r) \text{ process}_{\text{ptick}} \rangle$
and $Q :: \langle 'r \Rightarrow ('a, 's) \text{ process}_{\text{ptick}} \rangle$
and $R :: \langle 's \Rightarrow ('a, 't) \text{ process}_{\text{ptick}} \rangle$
proof *(rule Process-eq-optimizedI)*
fix t **assume** $\langle t \in \mathcal{D} (P ;_{\checkmark} (\lambda r. Q r ;_{\checkmark} R)) \rangle$
then consider $(D-P) t' u$ **where** $\langle t = \text{map } (ev \circ of-ev) t' @ u \rangle \langle t' \in \mathcal{D} P \rangle \langle tF t' \rangle \langle ftF u \rangle$
 $\mid (D-Q-R) t' r u$ **where** $\langle t = \text{map } (ev \circ of-ev) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle u \in \mathcal{D} (Q r ;_{\checkmark} R) \rangle$
unfolding *Seq_{ptick}-projs[*of P*]* **by** *fast*
thus $\langle t \in \mathcal{D} (P ;_{\checkmark} Q ;_{\checkmark} R) \rangle$
proof *cases*
case *D-P*
define $t'' :: \langle ('a, 's) \text{ trace}_{\text{ptick}} \rangle$ **where** $\langle t'' = \text{map } (ev \circ of-ev) t' \rangle$
from *D-P* **have** $\langle t'' \in \mathcal{D} (P ;_{\checkmark} Q) \rangle$
by *(auto simp add: t''-def Seq_{ptick}-projs intro: front-tickFree-Nil)*
moreover have $\langle tF t'' \rangle$ **by** *(simp add: t''-def)*
ultimately have $\langle \text{map } (ev \circ of-ev) t'' \in \mathcal{D} (P ;_{\checkmark} Q ;_{\checkmark} R) \rangle$
by *(simp add: Seq_{ptick}-projs[*of P ;_{\checkmark} Q*])*
(metis append.right-neutral front-tickFree-Nil)
also have $\langle \text{map } (ev \circ of-ev) t'' = \text{map } (ev \circ of-ev) t' \rangle$
by *(simp add: t''-def)*
finally have $\langle \text{map } (ev \circ of-ev) t' \in \mathcal{D} (P ;_{\checkmark} Q ;_{\checkmark} R) \rangle$.
with *D-P(1, 3, 4)* **show** $\langle t \in \mathcal{D} (P ;_{\checkmark} Q ;_{\checkmark} R) \rangle$ **by** *(simp add: is-processT7)*
next
case *D-Q-R*
from *D-Q-R(3)* **consider** $(D-Q) u' v$ **where** $\langle u = \text{map } (ev \circ of-ev) u' @ v \rangle$
 $\langle u' \in \mathcal{D} (Q r) \rangle \langle tF u' \rangle \langle ftF v \rangle$
 $\mid (D-R) u' s v$ **where** $\langle u = \text{map } (ev \circ of-ev) u' @ v \rangle \langle u' @ [\checkmark(s)] \in \mathcal{T} (Q r) \rangle$
 $\langle tF u' \rangle \langle v \in \mathcal{D} (R s) \rangle$
unfolding *Seq_{ptick}-projs* **by** *blast*
thus $\langle t \in \mathcal{D} (P ;_{\checkmark} Q ;_{\checkmark} R) \rangle$
proof *cases*

```

    case D-Q
    define t'' :: ⟨('a, 's) traceptick⟩ where ⟨t'' = map (ev ∘ of-ev) t' @ map (ev
    ∘ of-ev) u'⟩
    have ⟨t'' ∈ D (P ;✓ Q)⟩
    by (simp add: t''-def Seqptick-projs)
    (metis D-Q(2,3) D-Q-R(2) append-T-imp-tickFree list.distinct(1)
    tickFree-map-ev-of-ev-same-type-is)
    moreover have ⟨tF t'⟩ by (simp add: t''-def)
    ultimately have ⟨map (ev ∘ of-ev) t'' ∈ D (P ;✓ Q ;✓ R)⟩
    by (simp add: Seqptick-projs[of ⟨P ;✓ Q⟩])
    (metis append.right-neutral front-tickFree-Nil)
    also have ⟨map (ev ∘ of-ev) t'' = map (ev ∘ of-ev) t' @ map (ev ∘ of-ev) u'⟩
    by (simp add: t''-def)
    finally have ⟨map (ev ∘ of-ev) t' @ map (ev ∘ of-ev) u' ∈ D (P ;✓ Q ;✓ R)⟩
    .

    with D-Q(1,4) D-Q-R(1) is-processT7 show ⟨t ∈ D (P ;✓ Q ;✓ R)⟩ by
    fastforce
  next
  case D-R
  define t'' :: ⟨('a, 's) traceptick⟩ where ⟨t'' = map (ev ∘ of-ev) (map (ev ∘
  of-ev) t' @ u')⟩
  have ⟨t'' @ [✓(s)] ∈ T (P ;✓ Q)⟩
  by (simp add: t''-def Seqptick-projs[of P] del: map-map)
  (metis D-Q-R(2) D-R(2) append-T-imp-tickFree not-Cons-self2
  tickFree-map-ev-of-ev-same-type-is)
  moreover have ⟨tF t'⟩ unfolding t''-def by (blast intro: tickFree-map-ev-comp)

  ultimately have ⟨map (ev ∘ of-ev) t'' @ v ∈ D (P ;✓ Q ;✓ R)⟩
  unfolding Seqptick-projs[of ⟨P ;✓ Q⟩] using D-R(4) by blast
  also have ⟨map (ev ∘ of-ev) t'' @ v = t⟩
  by (simp add: D-Q-R(1) D-R(1) t''-def)
  finally show ⟨t ∈ D (P ;✓ Q ;✓ R)⟩ .
qed
qed
next

fix t assume ⟨t ∈ D (P ;✓ Q ;✓ R)⟩
then consider (D-P-Q) t' u where ⟨t = map (ev ∘ of-ev) t' @ u⟩ ⟨t' ∈ D (P
;✓ Q)⟩ ⟨tF t'⟩ ⟨ftF u⟩
| (D-R) t' s u where ⟨t = map (ev ∘ of-ev) t' @ u⟩ ⟨t' @ [✓(s)] ∈ T (P ;✓ Q)⟩
⟨u ∈ D (R s)⟩
  unfolding Seqptick-projs[of ⟨P ;✓ Q⟩] by blast
  thus ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
proof cases
  case D-P-Q
  from D-P-Q(2) consider (D-P) t'' u' where ⟨t' = map (ev ∘ of-ev) t'' @ u'⟩
  ⟨t'' ∈ D P⟩ ⟨tF t''⟩ ⟨ftF u'⟩
  | (D-Q) t'' r u' where ⟨t' = map (ev ∘ of-ev) t'' @ u'⟩ ⟨t'' @ [✓(r)] ∈ T P⟩
  ⟨tF t''⟩ ⟨u' ∈ D (Q r)⟩

```

```

    unfolding Seqptick-projs by blast
  thus ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
  proof cases
    case D-P
    from D-P(2, 3) have ⟨map (ev ∘ of-ev) t'' ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
      by (auto simp add: Seqptick-projs[of P] intro: front-tickFree-Nil)
    thus ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
      by (simp add: D-P-Q(1) D-P(1))
        (metis (mono-tags, lifting) D-P-Q(4) front-tickFree-append
          is-processT7 tickFree-map-ev-comp)
  next
    case D-Q
    from D-P-Q(3) D-Q(1, 4) have ⟨map (ev ∘ of-ev) u' ∈ D (Q r ;✓ R)⟩
      by (simp add: Seqptick-projs) (metis append.right-neutral front-tickFree-Nil)
    with D-Q(2, 3) have ⟨map (ev ∘ of-ev) (map (ev ∘ of-ev) t'' @ u') ∈ D (P
      ;✓ (λr. Q r ;✓ R))⟩
      by (auto simp add: Seqptick-projs[of P])
    with D-P-Q(4) is-processT7 show ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
      by (fastforce simp add: D-P-Q(1) D-Q(1))
  qed
  next
    case D-R
    then consider (T-P) t'' r u' where ⟨t' = map (ev ∘ of-ev) t'' @ u'⟩
      ⟨t'' @ [✓(r)] ∈ T P⟩ ⟨tF t''⟩ ⟨u' @ [✓(s)] ∈ T (Q r)⟩
    | (D-P) t'' u' where ⟨t' = map (ev ∘ of-ev) t'' @ u'⟩ ⟨t'' ∈ D P⟩ ⟨tF t''⟩ ⟨tF
      u'⟩
      by (auto simp add: Seqptick-projs append-eq-append-conv2 append-eq-map-conv
        Cons-eq-append-conv front-tickFree-append-iff intro: D-P T-P)
    thus ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
  proof cases
    case T-P
    from D-R(3) T-P(4) have ⟨map (ev ∘ of-ev) u' @ u ∈ D (Q r ;✓ R)⟩
      by (simp add: Seqptick-projs) (metis append-T-imp-tickFree not-Cons-self2)
    with T-P(2, 3) have ⟨map (ev ∘ of-ev) t'' @ map (ev ∘ of-ev) u' @ u ∈
      D (P ;✓ (λr. Q r ;✓ R))⟩
      by (auto simp add: Seqptick-projs[of P])
    also have ⟨map (ev ∘ of-ev) t'' @ map (ev ∘ of-ev) u' @ u = t⟩
      by (simp add: D-R(1) T-P(1))
    finally show ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩ .
  next
    case D-P
    from D-P(2, 3) have ⟨map (ev ∘ of-ev) t'' ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
      by (auto simp add: Seqptick-projs[of P] intro: front-tickFree-Nil)
    with D-R(3) show ⟨t ∈ D (P ;✓ (λr. Q r ;✓ R))⟩
      by (simp add: D-R(1) D-P(1))
        (metis (mono-tags, lifting) D-imp-front-tickFree
          front-tickFree-append is-processT7 tickFree-map-ev-comp)
  qed
  qed

```

next

```

fix  $t\ X$  assume  $\langle (t, X) \in \mathcal{F} (P ; \checkmark (\lambda r. Q\ r ; \checkmark R)) \rangle \langle t \notin \mathcal{D} (P ; \checkmark (\lambda r. Q\ r ; \checkmark R)) \rangle$ 
then consider  $(F\text{-}P)\ t'$  where  $\langle t = \text{map} (ev \circ of\text{-}ev)\ t' \rangle \langle (t', \text{ref}\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ P \rangle \langle tF\ t' \rangle$ 
   $\mid (F\text{-}Q\text{-}R)\ t'\ r\ u$  where  $\langle t = \text{map} (ev \circ of\text{-}ev)\ t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T}\ P \rangle$ 
   $\langle tF\ t' \rangle \langle (u, X) \in \mathcal{F} (Q\ r ; \checkmark R) \rangle$ 
  unfolding  $Seq_{ptick}\text{-}projs[of\ P]$  by fast
thus  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q ; \checkmark R) \rangle$ 
proof cases
  case  $F\text{-}P$ 
    from  $F\text{-}P(2)$  have  $\langle (t', \text{ref}\text{-}Seq_{ptick}(\text{ref}\text{-}Seq_{ptick}\ X)) \in \mathcal{F}\ P \rangle$ 
    by (simp add: ref-Seqptick-idem)
    with  $F\text{-}P(3)$  have  $\langle (\text{map} (ev \circ of\text{-}ev)\ t', \text{ref}\text{-}Seq_{ptick}\ X) \in \mathcal{F} (P ; \checkmark Q) \rangle$ 
    by (auto simp add: Seqptick-projs)
    thus  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q ; \checkmark R) \rangle$ 
    by (simp add: Seqptick-projs[of  $\langle P ; \checkmark Q \rangle]$  F-P(1))
    (metis map-ev-of-ev-map-ev-of-ev tickFree-map-ev-comp)
  next
    case  $F\text{-}Q\text{-}R$ 
    from  $F\text{-}Q\text{-}R(4)$ 
    consider  $(F\text{-}Q)\ u'$  where  $\langle u = \text{map} (ev \circ of\text{-}ev)\ u' \rangle$ 
     $\langle (u', \text{ref}\text{-}Seq_{ptick}\ X) \in \mathcal{F} (Q\ r) \rangle \langle tF\ u' \rangle$ 
     $\mid (F\text{-}R)\ u'\ s\ u''$  where  $\langle u = \text{map} (ev \circ of\text{-}ev)\ u' @ u'' \rangle \langle u' @ [\checkmark(s)] \in \mathcal{T} (Q\ r) \rangle \langle tF\ u' \rangle \langle (u'', X) \in \mathcal{F} (R\ s) \rangle$ 
     $\mid (D\text{-}Q)\ u'\ u''$  where  $\langle u = \text{map} (ev \circ of\text{-}ev)\ u' @ u'' \rangle \langle u' \in \mathcal{D} (Q\ r) \rangle \langle tF\ u' \rangle \langle tF\ u'' \rangle$ 
    unfolding  $Seq_{ptick}\text{-}projs$  by blast
    thus  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q ; \checkmark R) \rangle$ 
    proof cases
      case  $F\text{-}Q$ 
        from  $F\text{-}Q(2)\ F\text{-}Q\text{-}R(2, 3)$ 
        have  $\langle (\text{map} (ev \circ of\text{-}ev)\ t' @ u', \text{ref}\text{-}Seq_{ptick}\ X) \in \mathcal{F} (P ; \checkmark Q) \rangle$ 
        by (auto simp add: Seqptick-projs)
        with  $F\text{-}Q(1, 3)\ F\text{-}Q\text{-}R(1)$  show  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q ; \checkmark R) \rangle$ 
        by (simp add: Seqptick-projs[of  $\langle P ; \checkmark Q \rangle]$ )
        (metis map-append map-ev-of-ev-map-ev-of-ev tickFree-append-iff tickFree-map-ev-comp)
      next
        case  $F\text{-}R$ 
        from  $F\text{-}Q\text{-}R(2, 3)\ F\text{-}R(2)$  have  $\$ : \langle \text{map} (ev \circ of\text{-}ev)\ t' @ u' @ [\checkmark(s)] \in \mathcal{T} (P ; \checkmark Q) \rangle$ 
        by (auto simp add: Seqptick-projs)
        have  $\$ \$ : \langle tF (\text{map} (ev \circ of\text{-}ev)\ t' @ u') \rangle$  by (simp add: F-R(3))
        show  $\langle (t, X) \in \mathcal{F} (P ; \checkmark Q ; \checkmark R) \rangle$ 
        by (simp add: Seqptick-projs[of  $\langle P ; \checkmark Q \rangle]$ )
        (metis map-append[of  $\langle ev \circ of\text{-}ev \rangle \langle \text{map} (ev \circ of\text{-}ev)\ t' \rangle u]$ )
        (F-Q-R(1) F-R(1, 4) \$ \$ \$ append-eq-appendI map-ev-of-ev-map-ev-of-ev)

```



```

next
  case D-Q
  from D-Q(2) F-Q-R(2, 3) have $ :  $\langle \text{map } (ev \circ of\text{-}ev) \ t' \ @ \ u' \in \mathcal{D} \ (P ; \checkmark \ Q) \rangle$ 
    by (auto simp add: Seqptick-projs)
  have $$ :  $\langle tF \ (\text{map } (ev \circ of\text{-}ev) \ t' \ @ \ u') \rangle$  by (simp add: D-Q(3))
  have  $\langle t \in \mathcal{D} \ (P ; \checkmark \ Q ; \checkmark \ R) \rangle$ 
    by (simp add: Seqptick-projs[of  $\langle P ; \checkmark \ Q \rangle$ ])
    (metis D-Q(1, 4) F-Q-R(1) $ $$ append-assoc map-append map-ev-of-ev-map-ev-of-ev)
  thus  $\langle (t, X) \in \mathcal{F} \ (P ; \checkmark \ Q ; \checkmark \ R) \rangle$  by (fact is-processT8)
qed
qed
next
fix t X assume  $\langle (t, X) \in \mathcal{F} \ (P ; \checkmark \ Q ; \checkmark \ R) \rangle \ \langle t \notin \mathcal{D} \ (P ; \checkmark \ Q ; \checkmark \ R) \rangle$ 
then consider (F-P-Q) t' where  $\langle t = \text{map } (ev \circ of\text{-}ev) \ t' \rangle$ 
   $\langle (t', \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ (P ; \checkmark \ Q) \rangle \ \langle tF \ t' \rangle$ 
| (F-R) t' s u where  $\langle t = \text{map } (ev \circ of\text{-}ev) \ t' \ @ \ u \rangle$ 
   $\langle t' \ @ \ [\checkmark(s)] \in \mathcal{T} \ (P ; \checkmark \ Q) \rangle \ \langle tF \ t' \rangle \ \langle (u, X) \in \mathcal{F} \ (R \ s) \rangle$ 
  unfolding Seqptick-projs[of  $\langle P ; \checkmark \ Q \rangle$ ] by fast
thus  $\langle (t, X) \in \mathcal{F} \ (P ; \checkmark \ (\lambda r. \ Q \ r ; \checkmark \ R)) \rangle$ 
proof cases
  case F-P-Q
  from F-P-Q(1, 3)  $\langle t \notin \mathcal{D} \ (P ; \checkmark \ Q ; \checkmark \ R) \rangle$  have  $\langle t' \notin \mathcal{D} \ (P ; \checkmark \ Q) \rangle$ 
    by (simp add: Seqptick-projs) (metis front-tickFree-Nil self-append-conv)
  with F-P-Q(2) consider (F-P) t'' where  $\langle t' = \text{map } (ev \circ of\text{-}ev) \ t'' \rangle$ 
     $\langle (t'', \text{ref-Seq}_{ptick} \ (\text{ref-Seq}_{ptick} \ X :: ('a, 's) \text{refusal}_{ptick})) \in \mathcal{F} \ P \rangle \ \langle tF \ t'' \rangle$ 
  | (F-Q) t'' r u where  $\langle t' = \text{map } (ev \circ of\text{-}ev) \ t'' \ @ \ u \rangle \ \langle t'' \ @ \ [\checkmark(r)] \in \mathcal{T} \ P \rangle \ \langle tF$ 
   $t'' \rangle$ 
     $\langle (u, \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ (Q \ r) \rangle$ 
  unfolding Seqptick-projs by fast
  thus  $\langle (t, X) \in \mathcal{F} \ (P ; \checkmark \ (\lambda r. \ Q \ r ; \checkmark \ R)) \rangle$ 
proof cases
  case F-P
  from F-P(2, 3) have  $\langle (t'', \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ P \rangle$ 
    by (simp add: ref-Seqptick-idem)
  moreover have  $\langle t = \text{map } (ev \circ of\text{-}ev) \ t'' \rangle$ 
    by (simp add: F-P-Q(1) F-P(1))
  ultimately show  $\langle (t, X) \in \mathcal{F} \ (P ; \checkmark \ (\lambda r. \ Q \ r ; \checkmark \ R)) \rangle$ 
    by (auto simp add: Seqptick-projs[of P] F-P(3))
next
  case F-Q
  from F-P-Q(3) F-Q(1, 4) have  $\langle (\text{map } (ev \circ of\text{-}ev) \ u, X) \in \mathcal{F} \ (Q \ r ; \checkmark \ R) \rangle$ 
    by (auto simp add: Seqptick-projs)
  with F-Q(2, 3) show  $\langle (t, X) \in \mathcal{F} \ (P ; \checkmark \ (\lambda r. \ Q \ r ; \checkmark \ R)) \rangle$ 
    by (auto simp add: Seqptick-projs[of P] F-P-Q(1) F-Q(1))
qed
next
  case F-R
  from F-R(2) consider (T-P) t'' r u' where  $\langle t' = \text{map } (ev \circ of\text{-}ev) \ t'' \ @ \ u' \rangle$ 
     $\langle t'' \ @ \ [\checkmark(r)] \in \mathcal{T} \ P \rangle \ \langle tF \ t'' \rangle \ \langle u' \ @ \ [\checkmark(s)] \in \mathcal{T} \ (Q \ r) \rangle$ 

```

```

| (D-P) t'' u' where ⟨t' = map (ev ∘ of-ev) t'' @ u'⟩ ⟨t'' ∈  $\mathcal{D} P$ ⟩ ⟨tF t''⟩ ⟨tF
u'⟩
by (auto simp add: Seqptick-projs append-eq-append-conv2 Cons-eq-append-conv)
(auto simp add: append-eq-map-conv front-tickFree-append-iff intro: D-P
T-P)
thus ⟨(t, X) ∈  $\mathcal{F} (P ;_{\checkmark} (\lambda r. Q r ;_{\checkmark} R))$ ⟩
proof cases
case T-P
with F-R(3, 4) T-P(1, 4) have ⟨(map (ev ∘ of-ev) u' @ u, X) ∈  $\mathcal{F} (Q r ;_{\checkmark}$ 
R)⟩
by (auto simp add: Seqptick-projs)
with T-P(2, 3) show ⟨(t, X) ∈  $\mathcal{F} (P ;_{\checkmark} (\lambda r. Q r ;_{\checkmark} R))$ ⟩
by (auto simp add: Seqptick-projs[of P] F-R(1) T-P(1))
next
case D-P
with D-P(2,3) F-R(4) have ⟨t ∈  $\mathcal{D} (P ;_{\checkmark} (\lambda r. Q r ;_{\checkmark} R))$ ⟩
by (simp add: Seqptick-projs F-R(1) D-P(1))
(metis F-imp-front-tickFree front-tickFree-append tickFree-map-ev-comp)
thus ⟨(t, X) ∈  $\mathcal{F} (P ;_{\checkmark} (\lambda r. Q r ;_{\checkmark} R))$ ⟩ by (fact is-processT8)
qed
qed
qed

```

7.3 Distributivity of Non-Determinism

7.3.1 Sequential Composition

lemma Seq_{ptick}-distrib-GlobalNdet-left :
 ⟨P ;_✓ (λr. $\sqcap_{a \in A}. Q a r$) = (if A = {} then P ;_✓ (λr. STOP) else $\sqcap_{a \in A}. (P ;_{\checkmark} Q a)$)⟩
by simp (auto simp add: Process-eq-spec GlobalNdet-projs STOP-projs Seq_{ptick}-projs)

lemma Seq_{ptick}-distrib-GlobalNdet-right : ⟨($\sqcap_{a \in A}. P a$) ;_✓ Q = $\sqcap_{a \in A}. (P a ;_{\checkmark} Q)$ ⟩
by (simp add: Process-eq-spec GlobalNdet-projs STOP-projs Seq_{ptick}-projs)
 (safe; simp; blast) — quicker than auto proof

lemma Seq_{ptick}-distrib-Ndet-left : ⟨P ;_✓ (λr. Q r $\sqcap$ R r) = (P ;_✓ Q) $\sqcap$ (P ;_✓ R)⟩
by (fact Seq_{ptick}-distrib-GlobalNdet-left[of P ⟨{0 :: nat, 1}⟩
 ⟨λn. if n = 0 then Q else if n = 1 then R else undefined⟩,
 simplified GlobalNdet-distrib-unit-bis, simplified])

lemma *Seq_{ptick}-distrib-Ndet-right* : $\langle P \sqcap Q ;_{\checkmark} R = (P ;_{\checkmark} R) \sqcap (Q ;_{\checkmark} R) \rangle$
by (*fact Seq_{ptick}-distrib-GlobalNdet-right*[*of* $\langle \{0 :: \text{nat}, 1\} \rangle$
 $\langle \lambda n. \text{if } n = 0 \text{ then } P \text{ else if } n = 1 \text{ then } Q \text{ else undefined} \rangle R,$
simplified GlobalNdet-distrib-unit-bis, simplified])

7.3.2 Synchronization Product

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

lemma *Sync_{ptick}-distrib-GlobalNdet-left* :
 $\langle P \llbracket S \rrbracket_{\checkmark} \sqcap a \in A. Q \ a = (\text{if } A = \{\} \text{ then } P \llbracket S \rrbracket_{\checkmark} \text{ STOP else } \sqcap a \in A. (P \llbracket S \rrbracket_{\checkmark} Q \ a)) \rangle$
(is $\langle ?lhs = (\text{if } A = \{\} \text{ then } P \llbracket S \rrbracket_{\checkmark} \text{ STOP else } ?rhs) \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle A = \{\} \implies ?lhs = P \llbracket S \rrbracket_{\checkmark} \text{ STOP} \rangle$
by (*simp add: GlobalNdet.abs-eq STOP.abs-eq*)
next
show $\langle ?lhs = ?rhs \rangle$ **if** $\langle A \neq \{\} \rangle$
proof (*subst Process-eq-spec-optimized, safe*)
show $\langle t \in \mathcal{D} \ ?lhs \implies t \in \mathcal{D} \ ?rhs \rangle$ **for** t
by (*simp add: D-Sync_{ptick} GlobalNdet-projs*)
(metis $A \neq \{\}$ ex-in-conv is-processT1-TR)
next
show $\langle t \in \mathcal{D} \ ?rhs \implies t \in \mathcal{D} \ ?lhs \rangle$ **for** t
by (*simp add: GlobalNdet-projs D-Sync_{ptick}*) *blast*
next
assume *same-div* : $\langle \mathcal{D} \ ?lhs = \mathcal{D} \ ?rhs \rangle$
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$
with $\langle A \neq \{\} \rangle$ **consider** $\langle t \in \mathcal{D} \ ?lhs \rangle$
| (*fail*) $t \text{-} P \ t \text{-} Q \ X \text{-} P \ X \text{-} Q \ a$ **where**
 $\langle (t \text{-} P, X \text{-} P) \in \mathcal{F} \ P \rangle \langle a \in A \rangle \langle (t \text{-} Q, X \text{-} Q) \in \mathcal{F} \ (Q \ a) \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t \text{-} P, t \text{-} Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{tick-join } X \text{-} P \ S \ X \text{-} Q \rangle$
unfolding *Sync_{ptick}-projs F-GlobalNdet* **by** *force*
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
proof *cases*
from *same-div D-F* **show** $\langle t \in \mathcal{D} \ ?lhs \implies (t, X) \in \mathcal{F} \ ?rhs \rangle$ **by** *blast*
next
case *fail*
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$ **by** (*simp add: F-GlobalNdet F-Sync_{ptick}*) *blast*
qed
next
show $\langle (t, X) \in \mathcal{F} \ ?rhs \implies (t, X) \in \mathcal{F} \ ?lhs \rangle$ **for** $t \ X$
by (*simp add: GlobalNdet-projs F-Sync_{ptick} $\langle A \neq \{\} \rangle$*) *blast*
qed
qed

lemma *Sync_{ptick}-distrib-GlobalNdet-right* :
 $\langle \sqcap a \in A. P \ a \llbracket S \rrbracket_{\checkmark} Q = (\text{if } A = \{\} \text{ then } \text{STOP} \llbracket S \rrbracket_{\checkmark} Q \text{ else } \sqcap a \in A. (P \ a \llbracket S \rrbracket_{\checkmark} Q)) \rangle$

$Q))\rangle$
 (is $\langle ?lhs = (if\ A = \{\} \text{ then } STOP\ \llbracket S \rrbracket_{\checkmark}\ Q \text{ else } ?rhs) \rangle$)
proof (*split if-split, intro conjI impI*)
 show $\langle A = \{\} \implies ?lhs = STOP\ \llbracket S \rrbracket_{\checkmark}\ Q \rangle$
 by (*simp add: GlobalNdet.abs-eq STOP.abs-eq*)
next
 show $\langle ?lhs = ?rhs \rangle$ **if** $\langle A \neq \{\} \rangle$
proof (*subst Process-eq-spec-optimized, safe*)
 show $\langle t \in \mathcal{D}\ ?lhs \implies t \in \mathcal{D}\ ?rhs \rangle$ **for** t
 by (*simp add: D-Sync_{ptick} GlobalNdet-projs*)
 (*metis $\langle A \neq \{\} \rangle$ ex-in-conv is-processT1-TR*)
next
 show $\langle t \in \mathcal{D}\ ?rhs \implies t \in \mathcal{D}\ ?lhs \rangle$ **for** t
 by (*simp add: GlobalNdet-projs D-Sync_{ptick}*) *blast*
next
assume *same-div* : $\langle \mathcal{D}\ ?lhs = \mathcal{D}\ ?rhs \rangle$
fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle$
with $\langle A \neq \{\} \rangle$ **consider** $\langle t \in \mathcal{D}\ ?lhs \rangle$
 | (*fail*) $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q\ a$ **where**
 $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F}\ (P\ a) \rangle \langle a \in A \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \rangle$
 $\langle t\ \text{setinterleaves}_{\checkmark}^{\text{tick-join}} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}}\ \text{tick-join}\ X\text{-}P\ S\ X\text{-}Q \rangle$
unfolding *Sync_{ptick}-projs F-GlobalNdet* **by** *force*
thus $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$
proof *cases*
from *same-div D-F* **show** $\langle t \in \mathcal{D}\ ?lhs \implies (t, X) \in \mathcal{F}\ ?rhs \rangle$ **by** *blast*
next
case *fail*
thus $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$ **by** (*simp add: F-GlobalNdet F-Sync_{ptick}*) *blast*
qed
next
show $\langle (t, X) \in \mathcal{F}\ ?rhs \implies (t, X) \in \mathcal{F}\ ?lhs \rangle$ **for** $t\ X$
by (*simp add: GlobalNdet-projs F-Sync_{ptick} $\langle A \neq \{\} \rangle$*) *blast*
qed
qed

lemma (*in Sync_{ptick}-locale*) *Sync_{ptick}-GlobalNdet-cartprod*:
 $\langle (\sqcap (a, b) \in A \times B. (P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ b)) =$
 $(if\ A = \{\} \vee B = \{\} \text{ then } STOP \text{ else } (\sqcap a \in A. P\ a)\ \llbracket S \rrbracket_{\checkmark}\ (\sqcap b \in B. Q\ b)) \rangle$
by (*simp add: GlobalNdet-cartprod Sync_{ptick}-distrib-GlobalNdet-left*
Sync_{ptick}-distrib-GlobalNdet-right GlobalNdet-sets-commute[of A])

lemma *Sync_{ptick}-distrib-Ndet-left* :
 $\langle P\ \llbracket S \rrbracket_{\checkmark}\ Q \sqcap R = (P\ \llbracket S \rrbracket_{\checkmark}\ Q) \sqcap (P\ \llbracket S \rrbracket_{\checkmark}\ R) \rangle$
by (*rule trans[OF - Sync_{ptick}-distrib-GlobalNdet-left*
[of P S $\langle \{True, False\} \rangle \langle \lambda b. if\ b \text{ then } Q \text{ else } R \rangle]]]$)
(simp-all add: GlobalNdet-distrib-unit))

corollary *Sync_{ptick}-distrib-Ndet-right* :
 $\langle P \sqcap Q \llbracket S \rrbracket_{\checkmark} R = (P \llbracket S \rrbracket_{\checkmark} R) \sqcap (Q \llbracket S \rrbracket_{\checkmark} R) \rangle$
by (*rule trans*[*OF trans*[*OF - Sync_{ptick}-distrib-GlobalNdet-right*
[*of* $\langle \{ True, False \} \rangle \langle \lambda b. \text{if } b \text{ then } P \text{ else } Q \rangle S R \rangle$]])
(*simp-all add: GlobalNdet-distrib-unit*)

end

Chapter 8

Communications

8.1 Step Laws

8.1.1 Sequential Composition

lemma *Mprefix-Seq_{ptick}*: $\langle \Box a \in A \rightarrow P \ a \ ; \checkmark \ Q = \Box a \in A \rightarrow (P \ a \ ; \checkmark \ Q) \rangle$ (is
 $\langle ?lhs = ?rhs \rangle$)

proof (rule *Process-eq-optimizedI*)

show $\langle t \in \mathcal{D} \ ?lhs \implies t \in \mathcal{D} \ ?rhs \rangle$ **for** t

by (cases t , auto simp add: *Seq_{ptick}-projs Mprefix-projs image-iff Cons-eq-append-conv*)

 blast

next

show $\langle t \in \mathcal{D} \ ?rhs \implies t \in \mathcal{D} \ ?lhs \rangle$ **for** t

by (cases t , auto simp add: *Seq_{ptick}-projs Mprefix-projs image-iff Cons-eq-map-conv*
Cons-eq-append-conv)

 (metis *event_{ptick}.disc(1) event_{ptick}.sel(1) tickFree-Cons-iff*,
 metis *append-Cons event_{ptick}.discI(1) event_{ptick}.sel(1) tickFree-Cons-iff*)

next

fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle \ \langle t \notin \mathcal{D} \ ?lhs \rangle$

then consider $(F-P) \ t'$ **where** $\langle t = \text{map } (ev \circ of-ev) \ t' \rangle$

$\langle (t', \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ (\Box a \in A \rightarrow P \ a) \rangle \ \langle tF \ t' \rangle$

 | $(F-Q) \ t' \ r \ u$ **where** $\langle t = \text{map } (ev \circ of-ev) \ t' @ u \rangle \ \langle t' @ [\checkmark(r)] \in \mathcal{T} \ (\Box a \in A$
 $\rightarrow P \ a) \rangle \ \langle tF \ t' \rangle \ \langle (u, X) \in \mathcal{F} \ (Q \ r) \rangle$

unfolding *Seq_{ptick}-projs* **by** *fast*

thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$

proof *cases*

case $F-P$ **thus** $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$

by (cases t' ; simp add: *Seq_{ptick}-projs ref-Seq_{ptick}-def Mprefix-projs disjoint-iff*
 image-iff)

 (metis *IntI event_{ptick}.sel(1) rangeI*, metis *event_{ptick}.sel(1)*)

next

case $F-Q$ **thus** $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$

by (cases t) (auto simp add: *Seq_{ptick}-projs Mprefix-projs Cons-eq-append-conv*) +

 qed

next

```

fix t X assume  $\langle t, X \rangle \in \mathcal{F} \text{ ?rhs} \langle t \notin \mathcal{D} \text{ ?rhs} \rangle$ 
from  $\langle t, X \rangle \in \mathcal{F} \text{ ?rhs}$  consider  $\langle t = [] \rangle \langle X \cap \text{ev} \text{ ' } A = \{\} \rangle$ 
  | a t' where  $\langle t = \text{ev } a \# t' \rangle \langle a \in A \rangle \langle (t', X) \in \mathcal{F} (P \text{ a ; } \checkmark Q) \rangle$ 
  unfolding F-Mprefix by blast
thus  $\langle t, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
proof cases
  show  $\langle t = [] \implies X \cap \text{ev} \text{ ' } A = \{\} \implies (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
  by (auto simp add: Seqptick-projs ref-Seqptick-def F-Mprefix)
next
fix a t' assume  $\langle t = \text{ev } a \# t' \rangle \langle a \in A \rangle \langle (t', X) \in \mathcal{F} (P \text{ a ; } \checkmark Q) \rangle$ 
from  $\langle (t', X) \in \mathcal{F} (P \text{ a ; } \checkmark Q) \rangle \langle t \notin \mathcal{D} \text{ ?rhs} \rangle \langle a \in A \rangle \langle t = \text{ev } a \# t' \rangle$ 
consider (F-P) t'' where  $\langle t' = \text{map } (\text{ev} \circ \text{of-ev}) \text{ } t'' \rangle \langle (t'', \text{ref-Seq}_{\text{ptick}} X) \in \mathcal{F} (P \text{ a}) \rangle \langle tF \text{ } t'' \rangle$ 
  | (F-Q) t'' r u where  $\langle t' = \text{map } (\text{ev} \circ \text{of-ev}) \text{ } t'' @ u \rangle \langle t'' @ [\checkmark(r)] \in \mathcal{T} (P \text{ a}) \rangle \langle tF \text{ } t'' \rangle \langle (u, X) \in \mathcal{F} (Q \text{ } r) \rangle$ 
  by (auto simp add: Seqptick-projs D-Mprefix)
thus  $\langle t, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
proof cases
  case F-P thus  $\langle t, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
  by (simp add: Mprefix-projs Seqptick-projs  $\langle t = \text{ev } a \# t' \rangle$  Cons-eq-map-conv)
  (metis  $\langle a \in A \rangle \text{event}_{\text{ptick}}.\text{disc}(1) \text{event}_{\text{ptick}}.\text{sel}(1) \text{tickFree-Cons-iff}$ )
next
  case F-Q thus  $\langle t, X \rangle \in \mathcal{F} \text{ ?lhs}$ 
  by (simp add: Mprefix-projs Seqptick-projs  $\langle t = \text{ev } a \# t' \rangle$  Cons-eq-map-conv
  append-eq-Cons-conv)
  (metis (no-types, lifting)  $\langle a \in A \rangle \text{append-Cons comp-apply event}_{\text{ptick}}.\text{disc}(1) \text{event}_{\text{ptick}}.\text{sel}(1) \text{list.simps}(9) \text{tickFree-Cons-iff}$ )
qed
qed
qed

```

8.1.2 Synchronization Product

lemma (in Sync_{ptick}-locale) Mprefix-Sync_{ptick}-Mprefix-bis :

$$\begin{aligned}
& \langle \Box a \in (A \cup A') \rightarrow P \text{ a } \llbracket S \rrbracket_{\checkmark} \Box b \in (B \cup B') \rightarrow Q \text{ b} = \\
& (\Box a \in A \rightarrow (P \text{ a } \llbracket S \rrbracket_{\checkmark} \Box b \in (B \cup B') \rightarrow Q \text{ b})) \Box \\
& (\Box b \in B \rightarrow (\Box a \in (A \cup A') \rightarrow P \text{ a } \llbracket S \rrbracket_{\checkmark} Q \text{ b})) \Box \\
& (\Box x \in (A' \cap B') \rightarrow (P \text{ x } \llbracket S \rrbracket_{\checkmark} Q \text{ x})) \rangle \\
& (\text{is } \langle ?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2 = ?rhs1 \Box ?rhs2 \Box ?rhs3 \rangle) \\
& \text{if sets-assms: } \langle A \cap S = \{\} \rangle \langle A' \subseteq S \rangle \langle B \cap S = \{\} \rangle \langle B' \subseteq S \rangle
\end{aligned}$$

proof (rule Process-eq-optimizedI)

```

fix t assume  $\langle t \in \mathcal{D} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
then obtain u v t-P t-Q
  where * :  $\langle t = u @ v \rangle \langle tF \text{ } u \rangle \langle ftF \text{ } v \rangle$ 
   $\langle u \text{ setinterleaves}_{\checkmark \text{tick-join}} ((t-P, t-Q), S) \rangle$ 
   $\langle t-P \in \mathcal{D} ?lhs1 \wedge t-Q \in \mathcal{T} ?lhs2 \vee$ 
   $t-P \in \mathcal{T} ?lhs1 \wedge t-Q \in \mathcal{D} ?lhs2 \rangle$ 
  unfolding D-Syncptick by blast
from *(5) show  $\langle t \in \mathcal{D} (?rhs1 \Box ?rhs2 \Box ?rhs3) \rangle$ 

```



```

proof (elim disjE conjE)
  assume  $\langle t-P \in \mathcal{D} \text{ ?lhs1} \rangle \langle t-Q \in \mathcal{T} \text{ ?lhs2} \rangle$ 
  from  $\langle t-P \in \mathcal{D} \text{ ?lhs1} \rangle$  obtain  $a \ t-P'$ 
    where  $** : \langle a \in A \vee a \in A' \rangle \langle t-P = \text{ev } a \# t-P' \rangle \langle t-P' \in \mathcal{D} (P \ a) \rangle$ 
    unfolding D-Mprefix by blast
  from  $\langle t-Q \in \mathcal{T} \text{ ?lhs2} \rangle$  consider  $\langle t-Q = [] \rangle$ 
     $| \ b \ t-Q'$  where  $\langle b \in B \vee b \in B' \rangle \langle t-Q = \text{ev } b \# t-Q' \rangle \langle t-Q' \in \mathcal{T} (Q \ b) \rangle$ 
    unfolding T-Mprefix by blast
  thus  $\langle t \in \mathcal{D} (\text{?rhs1} \sqcap \text{?rhs2} \sqcap \text{?rhs3}) \rangle$ 
proof cases
  assume  $\langle t-Q = [] \rangle$ 
  with  $*(4)$  obtain  $u'$  where  $\langle a \notin S \rangle \langle u = \text{ev } a \# u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P', t-Q), S) \rangle$ 
    by (auto simp add: *(2) split: if-split-asm)
  moreover from  $\langle u = \text{ev } a \# u' \rangle *(2)$  have  $\langle tF \ u' \rangle$  by simp
  ultimately have  $\langle t \in \mathcal{D} \text{ ?rhs1} \rangle$ 
    using  $*(1, 3) *(1, 3) \langle t-Q \in \mathcal{T} \text{ ?lhs2} \rangle \langle A' \subseteq S \rangle$ 
    by (auto simp add: simp add: D-Mprefix D-Syncptick)
  thus  $\langle t \in \mathcal{D} (\text{?rhs1} \sqcap \text{?rhs2} \sqcap \text{?rhs3}) \rangle$  by (simp add: D-Det)
next
  fix  $b \ t-Q'$  assume  $*** : \langle b \in B \vee b \in B' \rangle \langle t-Q = \text{ev } b \# t-Q' \rangle \langle t-Q' \in \mathcal{T} (Q \ b) \rangle$ 
  from  $*(2)$  have  $\$ : \langle u = \text{ev } x \# u' \implies tF \ u' \rangle$  for  $x \ u'$  by simp
  from  $*(4)$  sets-assms  $*(1) *** (1)$ 
  consider (mv-both)  $u'$  where  $\langle a \in S \rangle \langle b = a \rangle \langle a \in A' \rangle \langle a \in B' \rangle \langle u = \text{ev } a \# u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P', t-Q'), S) \rangle$ 
     $| \ \langle \text{mv-left} \rangle \ u'$  where  $\langle a \notin S \rangle \langle a \in A \rangle \langle u = \text{ev } a \# u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P', t-Q), S) \rangle$ 
     $| \ \langle \text{mv-right} \rangle \ u'$  where  $\langle b \notin S \rangle \langle b \in B \rangle \langle u = \text{ev } b \# u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P, t-Q'), S) \rangle$ 
    by (auto simp add: *(2) *** (2) disjoint-iff split: if-split-asm)
  thus  $\langle t \in \mathcal{D} (\text{?rhs1} \sqcap \text{?rhs2} \sqcap \text{?rhs3}) \rangle$ 
proof cases
  case mv-both
    have  $\langle tF \ u' \rangle$  by (simp add: \$ mv-both(5))
    with  $*(3) *(3) *** (3) \text{ mv-both}(2, 6)$ 
    have  $\langle u' @ v \in \mathcal{D} (P \ a \llbracket S \rrbracket_{\checkmark} Q \ a) \rangle$  by (auto simp add: D-Syncptick)
    hence  $\langle t \in \mathcal{D} \text{ ?rhs3} \rangle$  by (simp add: D-Mprefix *(1) mv-both(3-5))
    thus  $\langle t \in \mathcal{D} (\text{?rhs1} \sqcap \text{?rhs2} \sqcap \text{?rhs3}) \rangle$  by (simp add: D-Det)
  next
  case mv-left
    have  $\langle tF \ u' \rangle$  by (simp add: \$ mv-left(3))
    with  $*(3) *(3) \langle t-Q \in \mathcal{T} \text{ ?lhs2} \rangle \text{ mv-left}(4)$ 
    have  $\langle u' @ v \in \mathcal{D} (P \ a \llbracket S \rrbracket_{\checkmark} \sqcap b \in (B \cup B') \rightarrow Q \ b) \rangle$  by (auto simp add: D-Syncptick)
    hence  $\langle t \in \mathcal{D} \text{ ?rhs1} \rangle$  by (simp add: D-Mprefix *(1) mv-left(2, 3))
    thus  $\langle t \in \mathcal{D} (\text{?rhs1} \sqcap \text{?rhs2} \sqcap \text{?rhs3}) \rangle$  by (simp add: D-Det)

```

```

next
  case mv-right
  have  $\langle tF\ u' \rangle$  by (simp add: $ mv-right(3))
  with  $\ast(3)\ \ast\ast(3)\ \text{mv-right}(4)\ \langle t-P \in \mathcal{D}\ ?lhs1 \rangle$ 
  have  $\langle u' @ v \in \mathcal{D}\ (\Box a \in (A \cup A') \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ b) \rangle$ 
    by (auto simp add: D-Syncptick)
  hence  $\langle t \in \mathcal{D}\ ?rhs2 \rangle$  by (simp add: D-Mprefix  $\ast(1)\ \text{mv-right}(2, 3)$ )
  thus  $\langle t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3) \rangle$  by (simp add: D-Det)
qed
qed
next
assume  $\langle t-P \in \mathcal{T}\ ?lhs1 \rangle\ \langle t-Q \in \mathcal{D}\ ?lhs2 \rangle$ 
from  $\langle t-Q \in \mathcal{D}\ ?lhs2 \rangle$  obtain  $b\ t-Q'$ 
  where  $\ast\ast : \langle b \in B \vee b \in B' \rangle\ \langle t-Q = ev\ b\ \# \ t-Q' \rangle\ \langle t-Q' \in \mathcal{D}\ (Q\ b) \rangle$ 
  unfolding D-Mprefix by blast
from  $\langle t-P \in \mathcal{T}\ ?lhs1 \rangle$  consider  $\langle t-P = [] \rangle$ 
  |  $a\ t-P'$  where  $\langle a \in A \vee a \in A' \rangle\ \langle t-P = ev\ a\ \# \ t-P' \rangle\ \langle t-P' \in \mathcal{T}\ (P\ a) \rangle$ 
  unfolding T-Mprefix by blast
thus  $\langle t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3) \rangle$ 
proof cases
  assume  $\langle t-P = [] \rangle$ 
  with  $\ast(4)$  obtain  $u'$  where  $\langle b \notin S \rangle\ \langle u = ev\ b\ \# \ u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q'), S) \rangle$ 
    by (auto simp add:  $\ast\ast(2)\ \text{split: if-split-asm}$ )
  moreover from  $\langle u = ev\ b\ \# \ u' \rangle\ \langle tF\ u \rangle$  have  $\langle tF\ u' \rangle$  by simp
  ultimately have  $\langle t \in \mathcal{D}\ ?rhs2 \rangle$ 
    using  $\ast(1, 3)\ \ast\ast(1, 3)\ \langle t-P \in \mathcal{T}\ ?lhs1 \rangle\ \langle B' \subseteq S \rangle$ 
    by (auto simp add: simp add: D-Mprefix D-Syncptick)
  thus  $\langle t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3) \rangle$  by (simp add: D-Det)
next
fix  $a\ t-P'$  assume  $\ast\ast\ast : \langle a \in A \vee a \in A' \rangle\ \langle t-P = ev\ a\ \# \ t-P' \rangle\ \langle t-P' \in \mathcal{T}\ (P\ a) \rangle$ 
from  $\langle tF\ u \rangle$  have  $\$ : \langle u = ev\ x\ \# \ u' \implies tF\ u' \rangle$  for  $x\ u'$  by simp
from  $\ast(4)\ \text{sets-assms}\ \ast\ast(1)\ \ast\ast\ast(1)$ 
consider (mv-both)  $u'$  where  $\langle a \in S \rangle\ \langle b = a \rangle\ \langle a \in A' \rangle\ \langle a \in B' \rangle$ 
   $\langle u = ev\ a\ \# \ u' \rangle\ \langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P', t-Q'), S) \rangle$ 
  | (mv-left)  $u'$  where  $\langle a \notin S \rangle\ \langle a \in A \rangle\ \langle u = ev\ a\ \# \ u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P', t-Q), S) \rangle$ 
  | (mv-right)  $u'$  where  $\langle b \notin S \rangle\ \langle b \in B \rangle\ \langle u = ev\ b\ \# \ u' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q'), S) \rangle$ 
  by (auto simp add:  $\ast\ast(2)\ \ast\ast\ast(2)\ \text{disjoint-iff split: if-split-asm}$ )
thus  $\langle t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3) \rangle$ 
proof cases
  case mv-both
  have  $\langle tF\ u' \rangle$  by (simp add: $ mv-both(5))
  with  $\ast(3)\ \ast\ast(3)\ \ast\ast\ast(3)\ \text{mv-both}(2, 6)$ 
  have  $\langle u' @ v \in \mathcal{D}\ (P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ a) \rangle$  by (auto simp add: D-Syncptick)
  hence  $\langle t \in \mathcal{D}\ ?rhs3 \rangle$  by (simp add: D-Mprefix  $\ast(1)\ \text{mv-both}(3-5)$ )
  thus  $\langle t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3) \rangle$  by (simp add: D-Det)

```

```

next
  case mv-left
  have  $\langle tF\ u' \rangle$  by (simp add: $ mv-left(3))
  with  $\ast(3)\ \ast\ast(3)\ mv-left(4)\ \langle t-Q \in \mathcal{D}\ ?lhs2 \rangle$ 
  have  $\langle u' @ v \in \mathcal{D}\ (P\ a\ \llbracket S \rrbracket_{\checkmark} \sqcap b \in (B \cup B') \rightarrow Q\ b) \rangle$  by (auto simp add:
D-Syncptick)
  hence  $\langle t \in \mathcal{D}\ ?rhs1 \rangle$  by (simp add: D-Mprefix  $\ast(1)\ mv-left(2, 3)$ )
  thus  $\langle t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  by (simp add: D-Det)
next
  case mv-right
  have  $\langle tF\ u' \rangle$  by (simp add: $ mv-right(3))
  with  $\ast(3)\ \ast\ast(3)\ \langle t-P \in \mathcal{T}\ ?lhs1 \rangle\ mv-right(4)$ 
  have  $\langle u' @ v \in \mathcal{D}\ (\sqcap a \in (A \cup A') \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ b) \rangle$ 
  by (auto simp add: D-Syncptick)
  hence  $\langle t \in \mathcal{D}\ ?rhs2 \rangle$  by (simp add: D-Mprefix  $\ast(1)\ mv-right(2, 3)$ )
  thus  $\langle t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  by (simp add: D-Det)
qed
qed
qed
next

fix t assume  $\langle t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$ 
consider  $\langle t = [] \rangle \mid r-s\ t' \text{ where } \langle t = \checkmark(r-s) \# t' \rangle \mid x\ t' \text{ where } \langle t = ev\ x \# t' \rangle$ 
  by (metis eventptick.exhaust neq-Nil-conv)
thus  $\langle t \in \mathcal{D}\ (?lhs1\ \llbracket S \rrbracket_{\checkmark}\ ?lhs2) \rangle$ 
proof cases
  assume  $\langle t = [] \rangle$ 
  with  $\langle t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  have False
  by (simp add: D-Det D-Mprefix)
  thus  $\langle t \in \mathcal{D}\ (?lhs1\ \llbracket S \rrbracket_{\checkmark}\ ?lhs2) \rangle$  ..
next
  fix r-s t' assume  $\langle t = \checkmark(r-s) \# t' \rangle$ 
  with  $\langle t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  have False
  by (simp add: D-Det D-Mprefix)
  thus  $\langle t \in \mathcal{D}\ (?lhs1\ \llbracket S \rrbracket_{\checkmark}\ ?lhs2) \rangle$  ..
next
  fix x t' assume  $\langle t = ev\ x \# t' \rangle$ 
  with  $\langle t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  consider
    (mv-left)  $\langle x \in A \rangle\ \langle t' \in \mathcal{D}\ (P\ x\ \llbracket S \rrbracket_{\checkmark}\ ?lhs2) \rangle$ 
    | (mv-right)  $\langle x \in B \rangle\ \langle t' \in \mathcal{D}\ (?lhs1\ \llbracket S \rrbracket_{\checkmark}\ Q\ x) \rangle$ 
    | (mv-both)  $\langle x \in A' \rangle\ \langle x \in B' \rangle\ \langle t' \in \mathcal{D}\ (P\ x\ \llbracket S \rrbracket_{\checkmark}\ Q\ x) \rangle$ 
  by (auto simp add: D-Det D-Mprefix)
  thus  $\langle t \in \mathcal{D}\ (?lhs1\ \llbracket S \rrbracket_{\checkmark}\ ?lhs2) \rangle$ 
proof cases
  case mv-left
  from  $\langle x \in A \rangle\ \langle A \cap S = \{\} \rangle$  have  $\langle x \notin S \rangle$  by blast
  from mv-left(2) obtain u v t-P t-Q
  where  $\ast : \langle t' = u @ v \rangle\ \langle tF\ u \rangle\ \langle ftF\ v \rangle$ 
     $\langle u\ setinterleaves_{\checkmark tick-join}\ ((t-P, t-Q), S) \rangle$ 

```

$\langle t-P \in \mathcal{D}(P x) \wedge t-Q \in \mathcal{T} \text{ ?lhs2} \vee$
 $t-P \in \mathcal{T}(P x) \wedge t-Q \in \mathcal{D} \text{ ?lhs2} \rangle$
unfolding $D\text{-Sync}_{ptick}$ **by** *blast*
have $\langle t = (ev x \# u) @ v \rangle$ **by** (*simp add: *(1) $\langle t = ev x \# t' \rangle$*)
moreover have $\langle tF (ev x \# u) \rangle$ **by** (*simp add: *(2)*)
moreover from $*(4)$ **have** $\langle ev x \# u \text{ setinterleaves}_{\checkmark tick-join} ((ev x \# t-P,$
 $t-Q), S) \rangle$
by (*cases t-Q*) (*auto simp add: $\langle x \notin S \rangle \text{ setinterleaving}_{ptick}\text{-simps split:}$*
 $event_{ptick}.split$)
moreover from $*(5)$ *mv-left(1)*
have $\langle ev x \# t-P \in \mathcal{D} \text{ ?lhs1} \wedge t-Q \in \mathcal{T} \text{ ?lhs2} \vee$
 $ev x \# t-P \in \mathcal{T} \text{ ?lhs1} \wedge t-Q \in \mathcal{D} \text{ ?lhs2} \rangle$ **by** (*simp add: Mprefix-projs*)
ultimately show $\langle t \in \mathcal{D}(\text{?lhs1 } \llbracket S \rrbracket_{\checkmark} \text{ ?lhs2}) \rangle$
using $*(3)$ **by** (*simp (no-asm) add: D-Sync_{ptick}*) *blast*
next
case *mv-right*
from $\langle x \in B \rangle \langle B \cap S = \{\} \rangle$ **have** $\langle x \notin S \rangle$ **by** *blast*
from *mv-right(2)* **obtain** $u v t-P t-Q$
where $*$: $\langle t' = u @ v \rangle \langle tF u \rangle \langle ftF v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$
 $\langle t-P \in \mathcal{D} \text{ ?lhs1} \wedge t-Q \in \mathcal{T}(Q x) \vee$
 $t-P \in \mathcal{T} \text{ ?lhs1} \wedge t-Q \in \mathcal{D}(Q x) \rangle$
unfolding $D\text{-Sync}_{ptick}$ **by** *blast*
have $\langle t = (ev x \# u) @ v \rangle$ **by** (*simp add: *(1) $\langle t = ev x \# t' \rangle$*)
moreover have $\langle tF (ev x \# u) \rangle$ **by** (*simp add: *(2)*)
moreover from $*(4)$ **have** $\langle ev x \# u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, ev x \#$
 $t-Q), S) \rangle$
by (*cases t-P*) (*auto simp add: $\langle x \notin S \rangle \text{ setinterleaving}_{ptick}\text{-simps split:}$*
 $event_{ptick}.split$)
moreover from $*(5)$ *mv-right(1)*
have $\langle t-P \in \mathcal{D} \text{ ?lhs1} \wedge ev x \# t-Q \in \mathcal{T} \text{ ?lhs2} \vee$
 $t-P \in \mathcal{T} \text{ ?lhs1} \wedge ev x \# t-Q \in \mathcal{D} \text{ ?lhs2} \rangle$ **by** (*simp add: Mprefix-projs*)
ultimately show $\langle t \in \mathcal{D}(\text{?lhs1 } \llbracket S \rrbracket_{\checkmark} \text{ ?lhs2}) \rangle$
using $*(3)$ **by** (*simp (no-asm) add: D-Sync_{ptick}*) *blast*
next
case *mv-both*
from $\langle x \in A' \rangle \langle A' \subseteq S \rangle$ **have** $\langle x \in S \rangle$ **by** *blast*
from *mv-both(3)* **obtain** $u v t-P t-Q$
where $*$: $\langle t' = u @ v \rangle \langle tF u \rangle \langle ftF v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$
 $\langle t-P \in \mathcal{D}(P x) \wedge t-Q \in \mathcal{T}(Q x) \vee$
 $t-P \in \mathcal{T}(P x) \wedge t-Q \in \mathcal{D}(Q x) \rangle$
unfolding $D\text{-Sync}_{ptick}$ **by** *blast*
have $\langle t = (ev x \# u) @ v \rangle$ **by** (*simp add: *(1) $\langle t = ev x \# t' \rangle$*)
moreover have $\langle tF (ev x \# u) \rangle$ **by** (*simp add: *(2)*)
moreover from $*(4)$ **have** $\langle ev x \# u \text{ setinterleaves}_{\checkmark tick-join} ((ev x \# t-P,$
 $ev x \# t-Q), S) \rangle$
by (*auto simp add: $\langle x \in S \rangle$*)
moreover from $*(5)$ *mv-both(1, 2)*

have $\langle ev\ x \# t\text{-}P \in \mathcal{D} \ ?lhs1 \wedge ev\ x \# t\text{-}Q \in \mathcal{T} \ ?lhs2 \vee$
 $ev\ x \# t\text{-}P \in \mathcal{T} \ ?lhs1 \wedge ev\ x \# t\text{-}Q \in \mathcal{D} \ ?lhs2 \rangle$ **by** (*simp add: Mprefix-projs*)
ultimately show $\langle t \in \mathcal{D} \ (\ ?lhs1 \llbracket S \rrbracket_{\checkmark} \ ?lhs2) \rangle$
using $\ast(\beta)$ **by** (*simp (no-asm) add: D-Sync_{ptick}*) *blast*
qed
qed
next

fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F} \ (\ ?lhs1 \llbracket S \rrbracket_{\checkmark} \ ?lhs2) \rangle \langle t \notin \mathcal{D} \ (\ ?lhs1 \llbracket S \rrbracket_{\checkmark} \ ?lhs2) \rangle$
then obtain $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$
where $fail : \langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ ?lhs1 \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ ?lhs2 \rangle$
 $\langle t\ setinterleaves_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick} \ tick\text{-}join\ X\text{-}P\ S\ X\text{-}Q \rangle$
unfolding *Sync_{ptick}-projs* **by** *blast*
consider $\langle t = [] \rangle \mid r\text{-}s\ t' \textbf{ where } \langle t = \checkmark(r\text{-}s) \# t' \rangle \mid a\ t' \textbf{ where } \langle t = ev\ a \# t' \rangle$
by (*metis event_{ptick}.exhaust neg-Nil-conv*)
thus $\langle (t, X) \in \mathcal{F} \ (\ ?rhs1 \sqcap \ ?rhs2 \sqcap \ ?rhs3) \rangle$
proof cases
assume $\langle t = [] \rangle$
with *Nil-setinterleaves_{ptick}* $fail(\beta)$ **have** $\langle t\text{-}P = [] \rangle \langle t\text{-}Q = [] \rangle$ **by** *blast+*
with $fail(1, 2)$ **have** $\langle X\text{-}P \cap ev\ ' (A \cup A') = \{\} \rangle \langle X\text{-}Q \cap ev\ ' (B \cup B') = \{\} \rangle$
by (*simp-all add: F-Mprefix*)
with $fail(4)$ $\langle A \cap S = \{\} \rangle \langle B \cap S = \{\} \rangle$ **show** $\langle (t, X) \in \mathcal{F} \ (\ ?rhs1 \sqcap \ ?rhs2 \sqcap \ ?rhs3) \rangle$
by (*simp add: \langle t = [] \rangle Det-projs Mprefix-projs super-ref-Sync_{ptick}-def*)
(use event_{ptick}.distinct(1) in blast)
next
fix $r\text{-}s\ t'$ **assume** $\langle t = \checkmark(r\text{-}s) \# t' \rangle$
hence $\langle t = [\checkmark(r\text{-}s)] \rangle$
by (*metis F-imp-front-tickFree \langle (t, X) \in \mathcal{F} \ (\ ?lhs1 \llbracket S \rrbracket_{\checkmark} \ ?lhs2) \rangle*
event_{ptick}.disc(2) front-tickFree-Cons-iff)
with $fail(\beta)$ **obtain** $r\ s$ **where** $\langle tick\text{-}join\ r\ s = Some\ r\text{-}s \rangle$
by (*auto elim: Cons-tick-setinterleaves_{ptick}E*)
from $\langle t = [\checkmark(r\text{-}s)] \rangle$ $fail(\beta)$ $\langle tick\text{-}join\ r\ s = Some\ r\text{-}s \rangle$
have $\langle t\text{-}P = [\checkmark(r)] \rangle$
by (*auto dest: inj-tick-join Nil-setinterleaves_{ptick}*
elim: Cons-tick-setinterleaves_{ptick}E)
with $fail(1)$ $\langle t = [\checkmark(r\text{-}s)] \rangle$ **have** *False* **by** (*simp add: F-Mprefix*)
thus $\langle (t, X) \in \mathcal{F} \ (\ ?rhs1 \sqcap \ ?rhs2 \sqcap \ ?rhs3) \rangle \dots$
next
fix $a\ t'$ **assume** $\langle t = ev\ a \# t' \rangle$
from $fail(1\text{--}\beta)$ *sets-assms* **consider**
(mv-left) $t\text{-}P'$ **where**
 $\langle a \notin S \rangle \langle a \in A \rangle \langle t\text{-}P = ev\ a \# t\text{-}P' \rangle \langle (t\text{-}P', X\text{-}P) \in \mathcal{F} \ (P\ a) \rangle$
 $\langle t' \ setinterleaves_{\checkmark tick\text{-}join} ((t\text{-}P', t\text{-}Q), S) \rangle$
| (mv-right) $t\text{-}Q'$ **where**
 $\langle a \notin S \rangle \langle a \in B \rangle \langle t\text{-}Q = ev\ a \# t\text{-}Q' \rangle \langle (t\text{-}Q', X\text{-}Q) \in \mathcal{F} \ (Q\ a) \rangle$
 $\langle t' \ setinterleaves_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q'), S) \rangle$
| (mv-both) $t\text{-}P'\ t\text{-}Q'$ **where**

```

    ⟨a ∈ S⟩ ⟨a ∈ A'⟩ ⟨a ∈ B'⟩ ⟨t-P = ev a # t-P'⟩ ⟨t-Q = ev a # t-Q'⟩
    ⟨(t-P', X-P) ∈ F (P a)⟩ ⟨(t-Q', X-Q) ∈ F (Q a)⟩ ⟨t' setinterleavestick-join
((t-P', t-Q'), S)⟩
    by (unfold ⟨t = ev a # t'⟩, elim Cons-ev-setinterleavesptickE)
    (simp-all add: F-Mprefix subset-iff disjoint-iff, blast+)
  thus ⟨(t, X) ∈ F (?rhs1 □ ?rhs2 □ ?rhs3)⟩
  proof cases
    case mv-left
    with fail(2, 4) have ⟨(t, X) ∈ F ?rhs1⟩
    by (subst F-Mprefix) (auto simp add: F-Syncptick ⟨t = ev a # t'⟩)
    thus ⟨(t, X) ∈ F (?rhs1 □ ?rhs2 □ ?rhs3)⟩
    by (simp add: F-Det ⟨t = ev a # t'⟩)
  next
    case mv-right
    with fail(1, 4) have ⟨(t, X) ∈ F ?rhs2⟩
    by (subst F-Mprefix) (auto simp add: F-Syncptick ⟨t = ev a # t'⟩)
    thus ⟨(t, X) ∈ F (?rhs1 □ ?rhs2 □ ?rhs3)⟩
    by (simp add: F-Det ⟨t = ev a # t'⟩)
  next
    case mv-both
    with fail(4) have ⟨(t, X) ∈ F ?rhs3⟩
    by (auto simp add: F-Mprefix F-Syncptick ⟨t = ev a # t'⟩)
    thus ⟨(t, X) ∈ F (?rhs1 □ ?rhs2 □ ?rhs3)⟩
    by (simp add: F-Det ⟨t = ev a # t'⟩)
  qed
qed
next
fix t X assume ⟨(t, X) ∈ F (?rhs1 □ ?rhs2 □ ?rhs3)⟩
  ⟨t ∉ D (?rhs1 □ ?rhs2 □ ?rhs3)⟩
consider ⟨t = []⟩ | r-s t' where ⟨t = ✓(r-s) # t'⟩ | a t' where ⟨t = ev a # t'⟩
  by (metis eventptick.exhaust neg-Nil-conv)
thus ⟨(t, X) ∈ F (?lhs1 [S]✓ ?lhs2)⟩
proof cases
  define X-P where X-P ≡ {ev a | a. ev a ∈ X ∧ a ∈ - (A ∪ A')} ∪
    {✓(r) | r-s r s. tick-join r s = Some r-s ∧ ✓(r-s) ∈ X}
  define X-Q where X-Q ≡ {ev b | b. ev b ∈ X ∧ b ∈ - (B ∪ B')} ∪
    {✓(s) | r-s r s. tick-join r s = Some r-s ∧ ✓(r-s) ∈ X}
  assume ⟨t = []⟩
  with ⟨(t, X) ∈ F (?rhs1 □ ?rhs2 □ ?rhs3)⟩
  have ⟨X ∩ ev 'A = {}⟩ ∧ X ∩ ev 'B = {} ∧ X ∩ ev '(A' ∩ B') = {}
  unfolding Det-projs F-Mprefix by auto
  with sets-assms(2, 4) have ⟨X ⊆ super-ref-Syncptick tick-join X-P S X-Q⟩
  by (simp add: super-ref-Syncptick-def X-P-def X-Q-def
    subset-iff disjoint-iff image-iff)
    (metis IntI eventptick.exhaust)
  moreover have ⟨([], X-P) ∈ F ?lhs1⟩ by (auto simp add: F-Mprefix X-P-def)
  moreover have ⟨([], X-Q) ∈ F ?lhs2⟩ by (auto simp add: F-Mprefix X-Q-def)
  ultimately show ⟨(t, X) ∈ F (?lhs1 [S]✓ ?lhs2)⟩

```

```

    by (simp add:  $\langle t = [] \rangle$   $F\text{-Sync}_{ptick}$ ) (use Nil-setinterleavingptick-Nil in blast)
next
  fix  $r\text{-}s$   $t'$  assume  $\langle t = \checkmark(r\text{-}s) \# t' \rangle$ 
  with  $\langle (t, X) \in \mathcal{F} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$ 
  have False by (simp add:  $F\text{-Det}$   $F\text{-Mprefix}$ )
  thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$  ..
next
  fix  $x$   $t'$  assume  $\langle t = \text{ev } x \# t' \rangle$ 
  with  $\langle (t, X) \in \mathcal{F} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$ 
  consider (mv-left)  $\langle x \in A \rangle \langle (t', X) \in \mathcal{F} (P \ x \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
    | (mv-right)  $\langle x \in B \rangle \langle (t', X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} Q \ x) \rangle$ 
    | (mv-both)  $\langle x \in A' \rangle \langle x \in B' \rangle \langle (t', X) \in \mathcal{F} (P \ x \llbracket S \rrbracket_{\checkmark} Q \ x) \rangle$ 
  unfolding  $F\text{-Det}$   $F\text{-Mprefix}$  by auto
  thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
proof cases
  case mv-left
  from mv-left(2) consider  $\langle t' \in \mathcal{D} (P \ x \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
  | (fail)  $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$  where
     $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} (P \ x) \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} ?lhs2 \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$ 
     $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X\text{-}P \ S \ X\text{-}Q \rangle$ 
  unfolding  $\text{Sync}_{ptick}\text{-projs}$  by blast
  thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
proof cases
  assume  $\langle t' \in \mathcal{D} (P \ x \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
  hence  $\langle t \in \mathcal{D} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$ 
  by (simp add:  $D\text{-Det}$   $D\text{-Mprefix}$   $\langle t = \text{ev } x \# t' \rangle$  mv-left(1))
  with  $\langle t \notin \mathcal{D} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  have False ..
  thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$  ..
next
  case fail
  have  $\langle (\text{ev } x \# t\text{-}P, X\text{-}P) \in \mathcal{F} ?lhs1 \rangle$ 
  by (simp add:  $F\text{-Mprefix}$  fail(1) mv-left(1))
  moreover from  $\langle t = \text{ev } x \# t' \rangle$  fail(3) mv-left(1)  $\langle A \cap S = \{\} \rangle$ 
  have  $\langle t \text{ setinterleaves}_{\checkmark tick\text{-}join} ((\text{ev } x \# t\text{-}P, t\text{-}Q), S) \rangle$ 
  by (cases  $t\text{-}Q$ ) (auto simp add: setinterleavingptick-simps split: eventptick.split)
  ultimately show  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
  using fail(2, 4) by (auto simp add:  $F\text{-Sync}_{ptick}$ )
qed
next
  case mv-right
  from mv-right(2) consider  $\langle t' \in \mathcal{D} (?lhs1 \llbracket S \rrbracket_{\checkmark} Q \ x) \rangle$ 
  | (fail)  $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$  where
     $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} ?lhs1 \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} (Q \ x) \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$ 
     $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X\text{-}P \ S \ X\text{-}Q \rangle$ 
  unfolding  $\text{Sync}_{ptick}\text{-projs}$  by blast
  thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
proof cases

```

```

    assume  $\langle t' \in \mathcal{D} (?lhs1 \llbracket S \rrbracket_{\checkmark} Q x) \rangle$ 
    hence  $\langle t \in \mathcal{D} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$ 
      by (simp add: D-Det D-Mprefix  $\langle t = ev\ x \# t' \rangle$  mv-right(1))
    with  $\langle t \notin \mathcal{D} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  have False ..
    thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$  ..
  next
  case fail
  have  $\langle (ev\ x \# t-Q, X-Q) \in \mathcal{F} ?lhs2 \rangle$ 
    by (simp add: F-Mprefix fail(2) mv-right(1))
  moreover from  $\langle t = ev\ x \# t' \rangle$  fail(3) mv-right(1)  $\langle B \cap S = \{\} \rangle$ 
  have  $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((t-P, ev\ x \# t-Q), S) \rangle$ 
  by (cases t-P) (auto simp add: setinterleavingptick-simps split: eventptick.split)
  ultimately show  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
    using fail(1, 4) by (auto simp add: F-Syncptick)
qed
next
case mv-both
from mv-both(3) consider  $\langle t' \in \mathcal{D} (P\ x \llbracket S \rrbracket_{\checkmark} Q\ x) \rangle$ 
  | (fail) t-P t-Q X-P X-Q where
     $\langle (t-P, X-P) \in \mathcal{F} (P\ x) \rangle$   $\langle (t-Q, X-Q) \in \mathcal{F} (Q\ x) \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$   $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X-P\ S\ X-Q \rangle$ 
  unfolding Syncptick-projs by blast
  thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
  proof cases
    assume  $\langle t' \in \mathcal{D} (P\ x \llbracket S \rrbracket_{\checkmark} Q\ x) \rangle$ 
    hence  $\langle t \in \mathcal{D} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$ 
      by (simp add: D-Det D-Mprefix  $\langle t = ev\ x \# t' \rangle$  mv-both(1, 2))
    with  $\langle t \notin \mathcal{D} (?rhs1 \sqcap ?rhs2 \sqcap ?rhs3) \rangle$  have False ..
    thus  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$  ..
  next
  case fail
  have  $\langle (ev\ x \# t-P, X-P) \in \mathcal{F} ?lhs1 \rangle$ 
    by (simp add: F-Mprefix fail(1) mv-both(1))
  moreover have  $\langle (ev\ x \# t-Q, X-Q) \in \mathcal{F} ?lhs2 \rangle$ 
    by (simp add: F-Mprefix fail(2) mv-both(2))
  moreover from  $\langle t = ev\ x \# t' \rangle$  fail(3) mv-both(1)  $\langle A' \subseteq S \rangle$ 
  have  $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((ev\ x \# t-P, ev\ x \# t-Q), S) \rangle$  by auto
  ultimately show  $\langle (t, X) \in \mathcal{F} (?lhs1 \llbracket S \rrbracket_{\checkmark} ?lhs2) \rangle$ 
    using fail(4) by (simp (no-asm) add: F-Syncptick) blast
qed
qed
qed
qed

```

corollary (in *Sync_{ptick}-locale*) *Mprefix-Sync_{ptick}-Mprefix*:

— This version is easier to use.

$\langle \square a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} \square b \in B \rightarrow Q\ b =$

$(\Box a \in (A - S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in (B - S) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \Box$
 $(\Box x \in (A \cap B \cap S) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} Q x)) \rangle$
by (*subst Mprefix-Sync_{ptick}-Mprefix-bis*
[of $\langle A - S \rangle S \langle A \cap S \rangle \langle B - S \rangle \langle B \cap S \rangle$, simplified Un-Diff-Int])
(simp-all add: Int-commute inf-left-commute)

corollary (in *Sync_{ptick}-locale*) *Mprefix-Sync_{ptick}-Mprefix-for-procomata*:

— Specialized version for Proc-Omata.

$\langle \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\Box a \in (A - S - B) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in (B - S - A) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \Box$
 $(\Box x \in (A \cap B - S) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b) \cap (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q x)) \Box$
 $(\Box x \in (A \cap B \cap S) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} Q x)) \rangle$

proof —

have * : $\langle \Box a \in (A - S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b) =$
 $(\Box a \in (A - S - B) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box a \in (A \cap B - S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \rangle$
by (*metis Diff-Int2 Diff-Int-distrib2 Mprefix-Un-distrib Un-Diff-Int*)
have ** : $\langle \Box b \in (B - S) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b) =$
 $(\Box b \in (B - S - A) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \Box$
 $(\Box b \in (A \cap B - S) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \rangle$
by (*metis (no-types) Int-Diff Int-commute Mprefix-Un-distrib Un-Diff-Int*)
have $\langle \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\Box a \in (A - S - B) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in (B - S - A) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \Box$
 $((\Box a \in (A \cap B - S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in (A \cap B - S) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b))) \Box$
 $(\Box x \in (A \cap B \cap S) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} Q x)) \rangle$
unfolding *Mprefix-Sync_{ptick}-Mprefix*
by (*auto simp add: ** Det-assoc intro!: arg-cong[where f = $\langle \lambda P. P \Box \rightarrow \rangle$]*)
(subst (3) Det-commute, subst Det-assoc,
*auto simp add: * Det-commute intro: arg-cong[where f = $\langle \lambda P. P \Box \rightarrow \rangle$])*
also have $\langle (\Box a \in (A \cap B - S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in (A \cap B - S) \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) =$
 $\Box x \in (A \cap B - S) \rightarrow ((P x \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b) \cap (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark}$
 $Q x)) \rangle$
by (*simp add: Mprefix-Det-Mprefix, rule mono-Mprefix-eq, simp*)
finally show ?thesis .
qed

unbundle *option-type-syntax*

8.2 Extended step Laws

8.2.1 Sequential Composition

lemma *Mndetprefix-Seq_{ptick}*: $\langle \Box a \in A \rightarrow P a ; \checkmark Q = \Box a \in A \rightarrow (P a ; \checkmark Q) \rangle$
by (*auto simp add: Mndetprefix-GlobalNdet Seq_{ptick}-distrib-GlobalNdet-right*
write0-def Mprefix-Seq_{ptick} intro: mono-GlobalNdet-eq)

8.2.2 Synchronization Product

Behaviour of SKIPS

lemma (*in Sync_{ptick}-locale*) *SKIPS-Sync_{ptick}-SKIPS* :
 $\langle \text{SKIPS } R \llbracket A \rrbracket \checkmark \text{SKIPS } S = \Box (r, s) \in R \times S. (\text{case } r \otimes \checkmark s \text{ of } \lfloor r-s \rfloor \Rightarrow \text{SKIP}$
 $r-s \mid \Diamond \Rightarrow \text{STOP}) \rangle$
by (*simp add: SKIPS-def Sync_{ptick}-distrib-GlobalNdet-left Sync_{ptick}-distrib-GlobalNdet-right*)
*(simp add: GlobalNdet-cartprod[*of* $R \ S \ \langle \lambda r s. \text{case } r \otimes \checkmark s \text{ of } \Diamond \Rightarrow \text{STOP} \mid \lfloor r-s \rfloor \Rightarrow \text{SKIP } r-s \rangle$*]
 $\Rightarrow \text{SKIP } r-s$])
*GlobalNdet-sets-commute[*of* $R \ S \ \langle \lambda r s. \text{case } r \otimes \checkmark s \text{ of } \Diamond \Rightarrow \text{STOP} \mid \lfloor r-s \rfloor \Rightarrow \text{SKIP } r-s \rangle$*]
 $\Rightarrow \text{SKIP } r-s$])

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* :
 $\langle \Box (r, s) \in R \times S. \text{SKIP } \lfloor \text{tick-join } r s \rfloor =$
 $\text{SKIPS } ((\text{the} \circ (\lambda(r, s). \text{tick-join } r s)) \text{ ' } (R \times S)) \rangle$
by (*simp add: SKIPS-def mono-GlobalNdet-eq2 split-def*)

lemma *GlobalNdet-prod-case-SKIP-STOP-is-GlobalNdet-prod-SKIP-iff* :
 $\langle \Box (r, s) \in R \times S. (\text{case } \text{tick-join } r s \text{ of } \Diamond \Rightarrow \text{STOP} \mid \lfloor r-s \rfloor \Rightarrow \text{SKIP } r-s) =$
 $\Box (r, s) \in R \times S. \text{SKIP } \lfloor \text{tick-join } r s \rfloor$
 $\longleftrightarrow (\forall r s. r \in R \longrightarrow s \in S \longrightarrow \text{tick-join } r s \neq \Diamond) \rangle$
(is $\langle ?lhs1 = ?lhs2 \longleftrightarrow ?rhs \rangle$)
proof (*rule iffI*)
show $\langle ?rhs \implies ?lhs1 = ?lhs2 \rangle$
by (*force intro: mono-GlobalNdet-eq*)
next
have $\langle \text{UNIV} \in \mathcal{R} \ ?lhs2 \longleftrightarrow R = \{\} \vee S = \{\} \rangle$
by (*simp add: Refusals-iff F-GlobalNdet F-SKIP*)
moreover have $\langle \text{UNIV} \in \mathcal{R} \ ?lhs1 \longleftrightarrow R = \{\} \vee S = \{\} \vee (\exists r s. r \in R \wedge s$
 $\in S \wedge \text{tick-join } r s = \Diamond) \rangle$
by (*auto simp add: Refusals-iff F-GlobalNdet F-SKIP F-STOP split: option.split*)
ultimately show $\langle ?lhs1 = ?lhs2 \implies ?rhs \rangle$ **by** (*metis empty-iff*)
qed

lemma (*in Sync_{ptick}-locale*) *SKIPS-Sync_{ptick}-SKIPS-bis* :
 $\langle \text{SKIPS } R \llbracket A \rrbracket \checkmark \text{SKIPS } S = \text{SKIPS } ((\text{the} \circ (\lambda(r, s). r \otimes \checkmark s)) \text{ ' } (R \times S)) \rangle$
if $\langle \bigwedge r s. r \in R \implies s \in S \implies r \otimes \checkmark s \neq \Diamond \rangle$
by (*unfold SKIPS-Sync_{ptick}-SKIPS, fold GlobalNdet-prod-SKIP-is-SKIPS*)

(simp add: SKIPS-Sync_{ptick}-SKIPS GlobalNdet-prod-case-SKIP-STOP-is-GlobalNdet-prod-SKIP-iff that)

lemma (in Sync_{ptick}-locale)

SKIPS-Sync_{ptick}-STOP [simp] : $\langle \text{SKIPS } R \llbracket A \rrbracket_{\checkmark} \text{ STOP} = \text{STOP} \rangle$
and STOP-Sync_{ptick}-SKIPS [simp] : $\langle \text{STOP } \llbracket A \rrbracket_{\checkmark} \text{ SKIPS } S = \text{STOP} \rangle$
by (fact SKIPS-Sync_{ptick}-SKIPS[**where** $S = \langle \{\} \rangle$, simplified]
 SKIPS-Sync_{ptick}-SKIPS[**where** $R = \langle \{\} \rangle$, simplified])+

Derived step Laws with Non-Determinism

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

lemma Mprefix-Inter_{ptick}-Mprefix :

$\langle \Box a \in A \rightarrow P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\Box a \in A \rightarrow (P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b)) \Box (\Box b \in B \rightarrow (\Box a \in A \rightarrow P a \parallel_{\checkmark} Q b)) \rangle$
by (fact Mprefix-Sync_{ptick}-Mprefix[**where** $S = \langle \{\} \rangle$, simplified])

lemma Mprefix-Par_{ptick}-Mprefix : $\langle \Box a \in A \rightarrow P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b = \Box x \in (A \cap B) \rightarrow (P x \parallel_{\checkmark} Q x) \rangle$

by (fact Mprefix-Sync_{ptick}-Mprefix[**where** $S = \langle \text{UNIV} \rangle$, simplified])

lemma Mprefix-Sync_{ptick}-Mprefix-subset :

$\langle [A \subseteq S; B \subseteq S] \implies \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b = \Box x \in (A \cap B) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} Q x) \rangle$
by (fact Mprefix-Sync_{ptick}-Mprefix-bis[of $\langle \{\} \rangle$ S A $\langle \{\} \rangle$ B , simplified])

lemma Mprefix-Sync_{ptick}-Mprefix-indep :

$\langle [A \cap S = \{\}; B \cap S = \{\}] \implies$
 $\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b)) \Box (\Box b \in B \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \rangle$
by (fact Mprefix-Sync_{ptick}-Mprefix-bis[of A S $\langle \{\} \rangle$ B $\langle \{\} \rangle$, simplified])

lemma Mprefix-Sync_{ptick}-Mprefix-left :

$\langle [A \cap S = \{\}; B \subseteq S] \implies \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b = \Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b) \rangle$
by (fact Mprefix-Sync_{ptick}-Mprefix-bis[of A S $\langle \{\} \rangle$ $\langle \{\} \rangle$ B , simplified])

lemma Mprefix-Sync_{ptick}-Mprefix-right :

$\langle [A \subseteq S; B \cap S = \{\}] \implies \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b = \Box b \in B \rightarrow (\Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b) \rangle$
by (fact Mprefix-Sync_{ptick}-Mprefix-bis[of $\langle \{\} \rangle$ S A B $\langle \{\} \rangle$, simplified])

lemma Mprefix-Sync_{ptick}-STOP : $\langle \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \text{ STOP} = \Box a \in (A - S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} \text{ STOP}) \rangle$

by (subst Mprefix-empty[symmetric], subst Mprefix-Sync_{ptick}-Mprefix, simp)

lemma *STOP-Sync_{ptick}-Mprefix* : $\langle \text{STOP } \llbracket S \rrbracket_{\checkmark} \square b \in B \rightarrow Q \ b = \square b \in (B - S) \rightarrow (\text{STOP } \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$
by (*subst Mprefix-empty[symmetric]*, *subst Mprefix-Sync_{ptick}-Mprefix*, *simp*)

Mixing deterministic and non deterministic prefix choices **lemma** *Mndetprefix-Sync_{ptick}-Mprefix* :

$\langle \square a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} (\square b \in B \rightarrow Q \ b) =$
 (*if* $A = \{\}$ *then* $\text{STOP } \llbracket S \rrbracket_{\checkmark} (\square b \in B \rightarrow Q \ b)$
else $\square a \in A. (\text{if } a \in S \text{ then } \text{STOP} \text{ else } (a \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} (\square b \in B \rightarrow Q \ b)))) \square$
 $(\square b \in (B - S) \rightarrow ((a \rightarrow P \ a) \ \llbracket S \rrbracket_{\checkmark} Q \ b)) \square$
(if $a \in B \cap S \text{ then } (a \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} Q \ a)) \text{ else } \text{STOP}) \rangle$
unfolding *Mndetprefix-GlobalNdet Sync_{ptick}-distrib-GlobalNdet-right*
write0-def Mprefix-Sync_{ptick}-Mprefix
by (*auto simp add: Mprefix-singl insert-Diff-if Int-insert-left*
intro: mono-GlobalNdet-eq arg-cong2[where f = $\langle (\square) \rangle$])

lemma *Mprefix-Sync_{ptick}-Mndetprefix* :

$\langle \square a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} (\square b \in B \rightarrow Q \ b) =$
 (*if* $B = \{\}$ *then* $(\square a \in A \rightarrow P \ a) \ \llbracket S \rrbracket_{\checkmark} \text{STOP}$
else $\square b \in B. (\text{if } b \in S \text{ then } \text{STOP} \text{ else } (b \rightarrow ((\square a \in A \rightarrow P \ a) \ \llbracket S \rrbracket_{\checkmark} Q \ b))) \square$
 $(\square a \in (A - S) \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} (b \rightarrow Q \ b))) \square$
(if $b \in A \cap S \text{ then } (b \rightarrow (P \ b \ \llbracket S \rrbracket_{\checkmark} Q \ b)) \text{ else } \text{STOP}) \rangle$
unfolding *Mndetprefix-GlobalNdet Sync_{ptick}-distrib-GlobalNdet-left*
write0-def Mprefix-Sync_{ptick}-Mprefix
by (*subst Det-commute*)
(auto simp add: Diff-triv Mprefix-singl Mprefix-is-STOP-iff disjoint-iff
intro!: mono-GlobalNdet-eq arg-cong2[where f = $\langle (\square) \rangle$] split: if-split-asm)

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

lemma *Mndetprefix-Sync_{ptick}-STOP* :

$\langle \square a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \text{STOP} =$
 (*if* $A \cap S = \{\}$ *then* $\square a \in A \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} \text{STOP})$
else $(\square a \in (A - S) \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} \text{STOP})) \square \text{STOP} \rangle$
(is $\langle ?lhs = (\text{if } A \cap S = \{\} \text{ then } ?rhs1 \text{ else } ?rhs2 \square \text{STOP}) \rangle$)

proof –

have $\langle \square a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \text{STOP} =$
 $\square a \in A. (\text{if } a \in S \text{ then } \text{STOP} \text{ else } (a \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} \text{STOP}))) \rangle$ *(is $\langle ?lhs = ?rhs' \rangle$)*

by (*subst Mndetprefix-Sync_{ptick}-Mprefix[where B = $\langle \{\} \rangle$, simplified]*)
(auto intro: mono-GlobalNdet-eq)

also have $\langle ?rhs' = (\text{if } A \cap S = \{\} \text{ then } ?rhs1 \text{ else } ?rhs2 \square \text{STOP}) \rangle$

proof (*split if-split, intro conjI impI*)

show $\langle A \cap S = \{\} \implies ?rhs' = ?rhs1 \rangle$

by (*auto simp add: Mndetprefix-GlobalNdet intro!: mono-GlobalNdet-eq*)

next

show $\langle ?rhs' = ?rhs2 \square \text{STOP} \rangle$ **if** $\langle A \cap S \neq \{\} \rangle$

proof (*cases $\langle A - S = \{\} \rangle$*)

show $\langle ?rhs' = ?rhs2 \square \text{STOP} \rangle$ **if** $\langle A - S = \{\} \rangle$

```

    by (simp add:  $\langle A - S = \{\} \rangle$  GlobalNdet-is-STOP-iff)
      (use  $\langle A - S = \{\} \rangle$  in blast)
  next
    show  $\langle ?rhs' = ?rhs2 \sqcap STOP \rangle$  if  $\langle A - S \neq \{\} \rangle$ 
    proof (subst Int-Diff-Un[symmetric],
      subst GlobalNdet-factorization-union
      [OF  $\langle A \cap S \neq \{\} \rangle$   $\langle A - S \neq \{\} \rangle$ , symmetric])
      have  $\langle (\sqcap a \in (A \cap S). (if\ a \in S\ then\ STOP\ else\ (a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP)))) \rangle$ 
        =  $STOP \rangle$  (is  $\langle ?fact1 = STOP \rangle$ )
      by (simp add: GlobalNdet-is-STOP-iff)
      moreover have  $\langle (\sqcap a \in (A - S). (if\ a \in S\ then\ STOP\ else\ (a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP)))) \rangle$ 
        =  $?rhs2 \rangle$  (is  $\langle ?fact2 = ?rhs2 \rangle$ )
      by (auto simp add: Mndetprefix-GlobalNdet intro: mono-GlobalNdet-eq)
      ultimately show  $\langle ?fact1 \sqcap ?fact2 = ?rhs2 \sqcap STOP \rangle$  by (metis Ndet-commute)
    qed
  qed
  qed
  finally show  $\langle ?lhs = (if\ A \cap S = \{\}\ then\ ?rhs1\ else\ ?rhs2 \sqcap STOP) \rangle$  .
  qed

```

lemma *STOP-Sync_{ptick}-Mndetprefix* :

```

 $\langle STOP \llbracket S \rrbracket_{\checkmark} (\sqcap b \in B \rightarrow Q\ b) =$ 
  ( if  $B \cap S = \{\}$  then  $\sqcap b \in B \rightarrow (STOP \llbracket S \rrbracket_{\checkmark}\ Q\ b)$ 
    else  $(\sqcap b \in (B - S) \rightarrow (STOP \llbracket S \rrbracket_{\checkmark}\ Q\ b)) \sqcap STOP \rangle$ 
  (is  $\langle ?lhs = (if\ B \cap S = \{\}\ then\ ?rhs1\ else\ ?rhs2 \sqcap STOP) \rangle$ )
proof -
  have  $\langle STOP \llbracket S \rrbracket_{\checkmark} (\sqcap b \in B \rightarrow Q\ b) =$ 
     $\sqcap b \in B. (if\ b \in S\ then\ STOP\ else\ (b \rightarrow (STOP \llbracket S \rrbracket_{\checkmark}\ Q\ b))) \rangle$  (is  $\langle ?lhs =$ 
     $?rhs' \rangle$ )
  by (subst Mprefix-Syncptick-Mndetprefix[where  $A = \langle \{\} \rangle$ , simplified])
    (auto intro: mono-GlobalNdet-eq)
  also have  $\langle ?rhs' = (if\ B \cap S = \{\}\ then\ ?rhs1\ else\ ?rhs2 \sqcap STOP) \rangle$ 
  proof (split if-split, intro conjI impI)
    show  $\langle B \cap S = \{\} \implies ?rhs' = ?rhs1 \rangle$ 
    by (auto simp add: Mndetprefix-GlobalNdet intro!: mono-GlobalNdet-eq)
  next
    show  $\langle ?rhs' = ?rhs2 \sqcap STOP \rangle$  if  $\langle B \cap S \neq \{\} \rangle$ 
    proof (cases  $\langle B - S = \{\} \rangle$ )
      show  $\langle ?rhs' = ?rhs2 \sqcap STOP \rangle$  if  $\langle B - S = \{\} \rangle$ 
      by (simp add:  $\langle B - S = \{\} \rangle$  GlobalNdet-is-STOP-iff)
      (use  $\langle B - S = \{\} \rangle$  in blast)
    next
      show  $\langle ?rhs' = ?rhs2 \sqcap STOP \rangle$  if  $\langle B - S \neq \{\} \rangle$ 
      proof (subst Int-Diff-Un[symmetric],
        subst GlobalNdet-factorization-union
        [OF  $\langle B \cap S \neq \{\} \rangle$   $\langle B - S \neq \{\} \rangle$ , symmetric])
        have  $\langle (\sqcap b \in (B \cap S). (if\ b \in S\ then\ STOP\ else\ (b \rightarrow (STOP \llbracket S \rrbracket_{\checkmark}\ Q\ b)))) \rangle$ 
          =  $STOP \rangle$  (is  $\langle ?fact1 = STOP \rangle$ )

```

by (simp add: GlobalNdet-is-STOP-iff)
 moreover have $\langle \Box b \in (B - S). (if\ b \in S\ then\ STOP\ else\ (b \rightarrow (STOP$
 $\llbracket S \rrbracket_{\checkmark} Q\ b))) \rangle$
 $= ?rhs2 \rangle$ (is $\langle ?fact2 = ?rhs2 \rangle$)
 by (auto simp add: Mndetprefix-GlobalNdet intro: mono-GlobalNdet-eq)
 ultimately show $\langle ?fact1 \Box ?fact2 = ?rhs2 \Box STOP \rangle$ by (metis Ndet-commute)
 qed
 qed
 qed
 finally show $\langle ?lhs = (if\ B \cap S = \{\} \ then\ ?rhs1\ else\ ?rhs2 \Box STOP) \rangle$.
 qed

corollary *Mndetprefix-Sync_{ptick}-Mprefix-subset* :

$\langle \Box a \in A \rightarrow P\ a \rangle \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q\ b) =$
 $(\ if\ A \subseteq B\ then\ \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a)$
 $\ else\ (\Box a \in (A \cap B) \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a)) \Box STOP \rangle$
 $(is\ \langle ?lhs = (if\ A \subseteq B\ then\ ?rhs1\ else\ ?rhs2) \rangle) \ if\ \langle A \subseteq S \rangle \langle B \subseteq S \rangle$
proof (cases $\langle A = \{\} \rangle$)
 show $\langle A = \{\} \implies ?lhs = (if\ A \subseteq B\ then\ ?rhs1\ else\ ?rhs2) \rangle$
 by (simp add: Mprefix-is-STOP-iff STOP-Sync_{ptick}-Mprefix $\langle B \subseteq S \rangle$)
 next
 from $\langle A \subseteq S \rangle$ have $* : \langle a \in A \implies a \in S \rangle$ for a by blast
 from $\langle B \subseteq S \rangle$ have $** : \langle B - S = \{\} \rangle \langle b \in B \wedge b \in S \iff b \in B \rangle$ for b by
 auto
 assume $\langle A \neq \{\} \rangle$
 have $\langle ?lhs = \Box a \in A. (if\ a \in B\ then\ (a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a))\ else\ STOP) \rangle$ (is $\langle ?lhs$
 $= ?rhs' \rangle$)
 by (auto simp add: Mndetprefix-Sync_{ptick}-Mprefix $* ** \langle A \neq \{\} \rangle$ intro: mono-GlobalNdet-eq)
 also have $\langle ?rhs' = (if\ A \subseteq B\ then\ ?rhs1\ else\ ?rhs2) \rangle$
proof (split if-split, intro conjI impI)
 show $\langle A \subseteq B \implies ?rhs' = \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a) \rangle$
 by (auto simp add: Mndetprefix-GlobalNdet intro!: mono-GlobalNdet-eq)
 next
 show $\langle ?rhs' = (\Box a \in (A \cap B) \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a)) \Box STOP \rangle$ if $\langle \neg A \subseteq B \rangle$
proof (cases $\langle A \cap B = \{\} \rangle$)
 show $\langle A \cap B = \{\} \implies ?rhs' = (\Box a \in (A \cap B) \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a)) \Box STOP \rangle$
 by (auto simp add: GlobalNdet-is-STOP-iff)
 next
 assume $\langle A \cap B \neq \{\} \rangle$
 from $\langle \neg A \subseteq B \rangle$ have $\langle A - B \neq \{\} \rangle$ by blast
 show $\langle ?rhs' = (\Box a \in (A \cap B) \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ a)) \Box STOP \rangle$
 by (auto simp add: Mndetprefix-GlobalNdet GlobalNdet-is-STOP-iff
 simp flip: GlobalNdet-factorization-union
 [OF $\langle A \cap B \neq \{\} \rangle \langle A - B \neq \{\} \rangle$, unfolded Int-Diff-Un]
 intro!: arg-cong2[where $f = \langle \Box \rangle$] mono-GlobalNdet-eq)
 qed
 qed
 finally show $\langle ?lhs = (if\ A \subseteq B\ then\ ?rhs1\ else\ ?rhs2) \rangle$ by simp

qed

corollary *Mprefix-Sync_{ptick}-Mndetprefix-subset* :

$\langle \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\text{ if } B \subseteq A \text{ then } \Box b \in B \rightarrow (P b \llbracket S \rrbracket_{\checkmark} Q b)$
 $\text{ else } (\Box b \in (A \cap B) \rightarrow (P b \llbracket S \rrbracket_{\checkmark} Q b)) \Box STOP \rangle$
 $(\text{is } \langle ?lhs = (\text{if } B \subseteq A \text{ then } ?rhs1 \text{ else } ?rhs2) \rangle \text{ if } \langle A \subseteq S \rangle \langle B \subseteq S \rangle$
proof (cases $\langle B = \{\} \rangle$)
show $\langle B = \{\} \implies ?lhs = (\text{if } B \subseteq A \text{ then } ?rhs1 \text{ else } ?rhs2) \rangle$
by (simp add: Mprefix-is-STOP-iff Mprefix-Sync_{ptick}-STOP $\langle A \subseteq S \rangle$)
next
from $\langle B \subseteq S \rangle$ **have** $*$: $\langle b \in B \implies b \in S \rangle$ **for** b **by** blast
from $\langle A \subseteq S \rangle$ **have** $**$: $\langle A - S = \{\} \rangle \langle a \in A \wedge a \in S \longleftrightarrow a \in A \rangle$ **for** a **by**
auto
assume $\langle B \neq \{\} \rangle$
have $\langle ?lhs = \Box b \in B. (\text{if } b \in A \text{ then } (b \rightarrow (P b \llbracket S \rrbracket_{\checkmark} Q b)) \text{ else } STOP) \rangle$ (is $\langle ?lhs$
 $= ?rhs' \rangle$)
by (auto simp add: Mprefix-Sync_{ptick}-Mndetprefix $** \langle B \neq \{\} \rangle$ intro: mono-GlobalNdet-eq)
also have $\langle ?rhs' = (\text{if } B \subseteq A \text{ then } ?rhs1 \text{ else } ?rhs2) \rangle$
proof (split if-split, intro conjI impI)
show $\langle B \subseteq A \implies ?rhs' = \Box b \in B \rightarrow (P b \llbracket S \rrbracket_{\checkmark} Q b) \rangle$
by (auto simp add: Mndetprefix-GlobalNdet intro!: mono-GlobalNdet-eq)
next
show $\langle ?rhs' = (\Box a \in (A \cap B) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} Q a)) \Box STOP \rangle$ **if** $\langle \neg B \subseteq A \rangle$
proof (cases $\langle A \cap B = \{\} \rangle$)
show $\langle A \cap B = \{\} \implies ?rhs' = (\Box a \in (A \cap B) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} Q a)) \Box STOP \rangle$
by (auto simp add: GlobalNdet-is-STOP-iff)
next
assume $\langle A \cap B \neq \{\} \rangle$
hence $\langle B \cap A \neq \{\} \rangle$ **by** blast
from $\langle \neg B \subseteq A \rangle$ **have** $\langle B - A \neq \{\} \rangle$ **by** blast
show $\langle ?rhs' = (\Box a \in (A \cap B) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} Q a)) \Box STOP \rangle$
by (auto simp add: Mndetprefix-GlobalNdet GlobalNdet-is-STOP-iff
simp flip: Int-commute GlobalNdet-factorization-union
[OF $\langle B \cap A \neq \{\} \rangle \langle B - A \neq \{\} \rangle$, unfolded Int-Diff-Un]
intro!: arg-cong2[where $f = \langle (\Box) \rangle$] mono-GlobalNdet-eq)
qed
qed
finally show $\langle ?lhs = (\text{if } B \subseteq A \text{ then } ?rhs1 \text{ else } ?rhs2) \rangle$ **by** simp
qed

corollary *Mndetprefix-Sync_{ptick}-Mprefix-indep* :

$\langle (\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b) =$
 $(\text{ if } A = \{\} \text{ then } \Box b \in B \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q b)$
 $\text{ else } \Box a \in A. (a \rightarrow (P a \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b))) \Box$
 $(\Box b \in B \rightarrow ((a \rightarrow P a) \llbracket S \rrbracket_{\checkmark} Q b)) \rangle$

if $\langle A \cap S = \{\} \rangle$ **and** $\langle B \cap S = \{\} \rangle$
proof (*cases* $\langle A = \{\} \rangle$)
 show $\langle A = \{\} \implies ?thesis \rangle$
 by (*simp add: Diff-triv STOP-Sync_{ptick}-Mprefix* $\langle B \cap S = \{\} \rangle$)
next
 from *that*(1) **have** $*$: $\langle a \in A \implies a \notin S \rangle$ **for** a **by** *blast*
 from *that*(2) **have** $**$: $\langle B - S = B \rangle$ **by** *blast*
 show $?thesis$ **if** $\langle A \neq \{\} \rangle$
 by (*simp add: Mndetprefix-Sync_{ptick}-Mprefix* $\langle A \neq \{\} \rangle$)
 (*rule mono-GlobalNdet-eq, simp add: * ***)
qed

corollary *Mprefix-Sync_{ptick}-Mndetprefix-indep* :
 $\langle (\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b) =$
 (*if* $B = \{\}$ *then* $\Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} STOP)$
 else $\Box b \in B. (b \rightarrow ((\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} Q b)) \Box$
 $(\Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} (b \rightarrow Q b))) \rangle$
if $\langle A \cap S = \{\} \rangle$ $\langle B \cap S = \{\} \rangle$
proof (*cases* $\langle B = \{\} \rangle$)
 show $\langle B = \{\} \implies ?thesis \rangle$
 by (*simp add: Diff-triv Mprefix-Sync_{ptick}-STOP* $\langle A \cap S = \{\} \rangle$)
next
 from *that*(2) **have** $*$: $\langle b \in B \implies b \notin S \rangle$ **for** b **by** *blast*
 from *that*(1) **have** $**$: $\langle A - S = A \rangle$ **by** *blast*
 show $?thesis$ **if** $\langle B \neq \{\} \rangle$
 by (*simp add: Mprefix-Sync_{ptick}-Mndetprefix* $\langle B \neq \{\} \rangle$)
 (*rule mono-GlobalNdet-eq, simp add: * ***)
qed

corollary *Mndetprefix-Sync_{ptick}-Mprefix-left* :
 $\langle (\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b) =$
 (*if* $A = \{\}$ *then* $STOP \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b)$
 else $\Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b)) \rangle$
if $\langle A \cap S = \{\} \rangle$ **and** $\langle B \subseteq S \rangle$
proof (*cases* $\langle A = \{\} \rangle$)
 show $\langle A = \{\} \implies ?thesis \rangle$ **by** *simp*
next
 from *that*(1) **have** $*$: $\langle a \in A \implies a \notin S \rangle$ **for** a **by** *blast*
 from *that*(2) **have** $**$: $\langle B - S = \{\} \rangle$ **by** *blast*
 show $?thesis$ **if** $\langle A \neq \{\} \rangle$
 by (*simp add: Mndetprefix-Sync_{ptick}-Mprefix* $\langle A \neq \{\} \rangle$, *unfold Mndetpre-*
fix-GlobalNdet)
 (*rule mono-GlobalNdet-eq, simp add: * ***)
qed

corollary *Mndetprefix-Sync_{ptick}-Mprefix-right* :
 $\langle (\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b) =$
 (*if* $A = \{\}$ *then* $STOP \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b)$

else $\Box b \in B \rightarrow ((\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} Q b))$
 if $\langle A \subseteq S \rangle$ and $\langle B \cap S = \{\} \rangle$
proof (cases $\langle A = \{\} \rangle$)
 show $\langle A = \{\} \implies ?thesis \rangle$ by simp
next
 from that(1) have $*$: $\langle a \in A \implies a \in S \rangle$ for a by blast
 from that(2) have $**$: $\langle B - S = B \rangle$ by blast
 show $?thesis$ if $\langle A \neq \{\} \rangle$
 by (simp add: Mndetprefix-Sync_{ptick}-Mprefix $\langle A \neq \{\} \rangle$,
 simp add: Mndetprefix-GlobalNdet Sync_{ptick}-distrib-GlobalNdet-right $\langle A \neq \{\} \rangle$
 flip: GlobalNdet-Mprefix-distr)
 (rule mono-GlobalNdet-eq, use $*$ $**$ in auto)
qed

corollary Mprefix-Sync_{ptick}-Mndetprefix-left :
 $\langle (\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b) =$
 (if $B = \{\}$ then $(\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} STOP$
 else $\Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b)) \rangle$
 if $\langle A \cap S = \{\} \rangle$ $\langle B \subseteq S \rangle$
proof (cases $\langle B = \{\} \rangle$)
 show $\langle B = \{\} \implies ?thesis \rangle$ by simp
next
 from that(2) have $*$: $\langle b \in B \implies b \in S \rangle$ for b by blast
 from that(1) have $**$: $\langle A - S = A \rangle$ by blast
 show $?thesis$ if $\langle B \neq \{\} \rangle$
 by (simp add: Mprefix-Sync_{ptick}-Mndetprefix $\langle B \neq \{\} \rangle$,
 simp add: Mndetprefix-GlobalNdet Sync_{ptick}-distrib-GlobalNdet-left $\langle B \neq \{\} \rangle$
 flip: GlobalNdet-Mprefix-distr)
 (rule mono-GlobalNdet-eq, use $*$ $**$ in auto)
qed

corollary Mprefix-Sync_{ptick}-Mndetprefix-right :
 $\langle (\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} (\Box b \in B \rightarrow Q b) =$
 (if $B = \{\}$ then $(\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} STOP$
 else $\Box b \in B \rightarrow ((\Box a \in A \rightarrow P a) \llbracket S \rrbracket_{\checkmark} Q b)) \rangle$
 if $\langle A \subseteq S \rangle$ $\langle B \cap S = \{\} \rangle$
proof (cases $\langle B = \{\} \rangle$)
 show $\langle B = \{\} \implies ?thesis \rangle$ by simp
next
 from that(2) have $*$: $\langle b \in B \implies b \notin S \rangle$ for b by blast
 from that(1) have $**$: $\langle A - S = \{\} \rangle$ by blast
 show $?thesis$ if $\langle B \neq \{\} \rangle$
 by (simp add: Mprefix-Sync_{ptick}-Mndetprefix $\langle B \neq \{\} \rangle$,
 unfold Mndetprefix-GlobalNdet)
 (rule mono-GlobalNdet-eq, simp add: $*$ $**$)
qed

corollary $Mndetprefix\text{-}Par_{ptick}\text{-}Mprefix :$

$\langle \Box a \in A \rightarrow P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\text{if } A \subseteq B \text{ then } \Box a \in A \rightarrow (P a \parallel_{\checkmark} Q a) \text{ else } (\Box a \in (A \cap B) \rightarrow (P a \parallel_{\checkmark} Q a))$
 $\Box STOP) \rangle$
by (*simp add: Mndetprefix-Sync_{ptick}-Mprefix-subset*)

corollary $Mprefix\text{-}Par_{ptick}\text{-}Mndetprefix :$

$\langle \Box a \in A \rightarrow P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\text{if } B \subseteq A \text{ then } \Box b \in B \rightarrow (P b \parallel_{\checkmark} Q b) \text{ else } (\Box b \in (A \cap B) \rightarrow (P b \parallel_{\checkmark} Q b))$
 $\Box STOP) \rangle$
by (*simp add: Mprefix-Sync_{ptick}-Mndetprefix-subset*)

corollary $Mndetprefix\text{-}Inter_{ptick}\text{-}Mprefix :$

$\langle \Box a \in A \rightarrow P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\text{if } A = \{\} \text{ then } \Box b \in B \rightarrow \text{RenamingTick } (Q b ; STOP) \text{ (}\lambda s. \text{ the (tick-join } (g$
 $s) s))$
 $\text{else } \Box a \in A. (a \rightarrow (P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in B \rightarrow (a \rightarrow P a \parallel_{\checkmark} Q b))) \rangle$
by (*simp add: Mndetprefix-Sync_{ptick}-Mprefix-indep*
 $Mprefix\text{-}Seq STOP\text{-}Inter_{ptick}[of - g])$

corollary $Mprefix\text{-}Inter_{ptick}\text{-}Mndetprefix :$

$\langle \Box a \in A \rightarrow P a \parallel_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\text{if } B = \{\} \text{ then } \Box a \in A \rightarrow \text{RenamingTick } (P a ; STOP) \text{ (}\lambda r. \text{ the (tick-join } r$
 $(g r)))$
 $\text{else } \Box b \in B. (b \rightarrow (\Box a \in A \rightarrow P a \parallel_{\checkmark} Q b)) \Box$
 $(\Box a \in A \rightarrow (P a \parallel_{\checkmark} b \rightarrow Q b))) \rangle$
by (*simp add: Mprefix-Sync_{ptick}-Mndetprefix-indep*
 $Mprefix\text{-}Seq Inter_{ptick}\text{-}STOP[of - g])$

Mixing two non deterministic prefix choices lemma $Mndetprefix\text{-}Sync_{ptick}\text{-}Mndetprefix$
 $:$

$\langle \Box a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q b =$
 $(\text{if } A = \{\} \text{ then } \text{if } B \cap S = \{\} \text{ then } \Box b \in B \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q b)$
 $\text{else } (\Box x \in (B - S) \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q x)) \Box STOP$
 $\text{else } \text{if } B = \{\} \text{ then } \text{if } A \cap S = \{\} \text{ then } \Box a \in A \rightarrow (P a \llbracket S \rrbracket_{\checkmark} STOP)$
 $\text{else } (\Box x \in (A - S) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} STOP)) \Box STOP$
 $\text{else } \Box b \in B. \Box a \in A.$
 $(\text{if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q b)) \Box$
 $(\text{if } b \in S \text{ then } STOP \text{ else } b \rightarrow (a \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \Box$
 $(\text{if } a = b \wedge b \in S \text{ then } b \rightarrow (P a \llbracket S \rrbracket_{\checkmark} Q b) \text{ else } STOP) \rangle$
 $(\text{is } \langle ?lhs = (\text{if } A = \{\} \text{ then } \text{if } B \cap S = \{\} \text{ then } ?mv\text{-}right \text{ else } ?mv\text{-}right' \Box$
 $STOP$
 $\text{else } \text{if } B = \{\} \text{ then } \text{if } A \cap S = \{\} \text{ then } ?mv\text{-}left \text{ else } ?mv\text{-}left' \Box$
 $STOP$
 $\text{else } ?huge\text{-}mess) \rangle$

proof (*split if-split, intro conjI impI*)

show $\langle A = \{\} \implies ?lhs = (\text{if } B \cap S = \{\} \text{ then } ?mv\text{-}right \text{ else } ?mv\text{-}right' \Box$

$STOP\rangle$
by (*auto simp add: STOP-Sync_{ptick}-Mndetprefix intro: mono-Mndetprefix-eq*)
next
show $\langle ?lhs = (if\ B = \{\} \text{ then } if\ A \cap S = \{\} \text{ then } ?mv\text{-}left \text{ else } ?mv\text{-}left' \sqcap STOP$
*else ?huge-mess) **if** $\langle A \neq \{\} \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle B = \{\} \implies ?lhs = (if\ A \cap S = \{\} \text{ then } ?mv\text{-}left \text{ else } ?mv\text{-}left' \sqcap STOP) \rangle$
by (*auto simp add: Mndetprefix-Sync_{ptick}-STOP intro: mono-Mndetprefix-eq*)
next
assume $\langle B \neq \{\} \rangle$
have $\langle ?lhs = \sqcap b \in B. \sqcap a \in A. (a \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} (b \rightarrow Q\ b)) \rangle$
by (*simp add: Mndetprefix-GlobalNdet $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle$*
Sync_{ptick}-distrib-GlobalNdet-left Sync_{ptick}-distrib-GlobalNdet-right)
also have $\langle \dots = ?huge\text{-}mess \rangle$
by (*auto simp add: write0-def Mprefix-Sync_{ptick}-Mprefix Diff-triv Mpre-*
fix-is-STOP-iff
*intro!: mono-GlobalNdet-eq arg-cong2[**where** $f = \langle (\square) \rangle$])*
finally show $\langle ?lhs = ?huge\text{-}mess \rangle$.
qed
qed*

lemma *Mndetprefix-Sync_{ptick}-Mndetprefix-subset* :
 $\langle \sqcap a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \sqcap b \in B \rightarrow Q\ b =$
 $(if\ \exists b. A = \{b\} \wedge B = \{b\}$
then (THE $b. B = \{b\}$) $\rightarrow (P\ (THE\ a. A = \{a\})\ \llbracket S \rrbracket_{\checkmark} Q\ (THE\ b. B = \{b\}))$
else $(\sqcap x \in (A \cap B) \rightarrow (P\ x\ \llbracket S \rrbracket_{\checkmark} Q\ x)) \sqcap STOP$
 $\rangle$
 $(is\ \langle ?lhs = (if\ ?cond \text{ then } ?rhs1 \text{ else } ?rhs2) \rangle \text{ if } \langle A \subseteq S \rangle \langle B \subseteq S \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle ?cond \implies ?lhs = ?rhs1 \rangle$
by (*elim exE, simp add: write0-def*)
(subst Mprefix-Sync_{ptick}-Mprefix-subset, use $\langle A \subseteq S \rangle$ in simp-all)
next
assume $\langle \neg ?cond \rangle$
let $?term = \langle \lambda a\ b. (b \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ b)) \rangle$
have $\langle ?lhs = \sqcap b \in B. \sqcap a \in A. (if\ a = b \text{ then } ?term\ a\ b \text{ else } STOP) \rangle$
 $(is\ \langle ?lhs = \sqcap b \in B. \sqcap a \in A. ?rhs' b\ a \rangle)$
proof (*cases $\langle A = \{\} \vee B = \{\} \rangle$*)
from $\langle A \subseteq S \rangle \langle B \subseteq S \rangle$ **show** $\langle A = \{\} \vee B = \{\} \implies ?lhs = (\sqcap b \in B. \sqcap a \in A.$
 $?rhs' b\ a) \rangle$
by (*elim disjE*) (*simp-all add: Mndetprefix-Sync_{ptick}-STOP STOP-Sync_{ptick}-Mndetprefix*
Int-absorb2 Mndetprefix-is-STOP-iff Ndet-is-STOP-iff)
next
show $\langle \neg (A = \{\} \vee B = \{\}) \implies ?lhs = (\sqcap b \in B. \sqcap a \in A. ?rhs' b\ a) \rangle$
by (*simp add: Mndetprefix-Sync_{ptick}-Mndetprefix*)
(intro mono-GlobalNdet-eq, use $\langle A \subseteq S \rangle \langle B \subseteq S \rangle$ in auto)
qed

also have $\langle (\sqcap b \in B. \sqcap a \in A. ?rhs' b\ a) = ?rhs2 \rangle$

```

proof (cases  $\langle B \cap A = \{\} \rangle$ )
  assume  $\langle B \cap A = \{\} \rangle$ 
  hence  $\langle A \cap B = \{\} \rangle$  by blast
  hence  $\langle (\sqcap b \in B. \sqcap a \in A. ?rhs' b a) = STOP \rangle$  by (auto simp add: Global-
Ndet-is-STOP-iff)
  thus  $\langle (\sqcap b \in B. \sqcap a \in A. ?rhs' b a) = ?rhs2 \rangle$  by (auto simp add:  $\langle A \cap B = \{\} \rangle$ )
next
  show  $\langle (\sqcap b \in B. \sqcap a \in A. ?rhs' b a) = ?rhs2 \rangle$  if  $\langle B \cap A \neq \{\} \rangle$ 
  proof (cases  $\langle B - A = \{\} \rangle$ )
    assume  $\langle B - A = \{\} \rangle$ 
    hence  $\langle A \cap B = B \rangle$  by blast
    have  $\langle (\sqcap a \in A. ?rhs' b a) = (if A = \{b\} then ?term b b else ?term b b \sqcap$ 
STOP)  $\rangle$ 
      (is  $\langle (\sqcap a \in A. ?rhs' b a) = ?rhs'' b \rangle$  if  $\langle b \in B \rangle$  for  $b$ )
    proof (cases  $\langle A \cap \{b\} = \{\} \rangle$ )
      from  $\langle B - A = \{\} \rangle \langle b \in B \rangle$ 
      show  $\langle A \cap \{b\} = \{\} \implies (\sqcap a \in A. ?rhs' b a) = ?rhs'' b \rangle$  by auto
    next
      show  $\langle (\sqcap a \in A. ?rhs' b a) = ?rhs'' b \rangle$  if  $\langle A \cap \{b\} \neq \{\} \rangle$ 
      proof (cases  $\langle A - \{b\} = \{\} \rangle$ )
        show  $\langle A - \{b\} = \{\} \implies (\sqcap a \in A. ?rhs' b a) = ?rhs'' b \rangle$ 
          using  $\langle A \cap \{b\} \neq \{\} \rangle$  by auto
        next
          show  $\langle \sqcap a \in A. ?rhs' b a = ?rhs'' b \rangle$  if  $\langle A - \{b\} \neq \{\} \rangle$ 
          using  $\langle A - \{b\} \neq \{\} \rangle \langle A \cap \{b\} \neq \{\} \rangle$ 
          by (auto simp add: GlobalNdet-is-STOP-iff
simp flip: GlobalNdet-factorization-union
[OF  $\langle A \cap \{b\} \neq \{\} \rangle \langle A - \{b\} \neq \{\} \rangle$ , unfolded Int-Diff-Un]
intro: arg-cong2[where  $f = \langle (\sqcap) \rangle$ ])
        qed
      qed
    hence  $\langle (\sqcap b \in B. \sqcap a \in A. ?rhs' b a) = \sqcap b \in B. ?rhs'' b \rangle$ 
      by (fact mono-GlobalNdet-eq)
    also have  $\langle (\sqcap b \in B. ?rhs'' b) = ?rhs2 \rangle$ 
    proof -
      from  $\langle \neg ?cond \rangle$  have  $\langle (\sqcap b \in B. ?rhs'' b) = \sqcap b \in B. ?term b b \sqcap STOP \rangle$ 
        by (metis Diff-eq-empty-iff Int-commute  $\langle A \cap B = B \rangle$ 
 $\langle B - A = \{\} \rangle$  subset-singleton-iff  $\langle B \cap A \neq \{\} \rangle$ )
      also have  $\langle \dots = (\sqcap b \in B. ?term b b) \sqcap STOP \rangle$ 
        by (simp add: Process-eq-spec Ndet-projs GlobalNdet-projs STOP-projs)
      finally show  $\langle (\sqcap b \in B. ?rhs'' b) = ?rhs2 \rangle$ 
        by (simp add: Mndetprefix-GlobalNdet  $\langle A \cap B = B \rangle$ )
      qed
    finally show  $\langle (\sqcap b \in B. \sqcap a \in A. ?rhs' b a) = ?rhs2 \rangle$  .
  next
    assume  $\langle B - A \neq \{\} \rangle$ 
    have  $\langle \sqcap a \in A. (if a = b then ?term a b else STOP) =$ 
      (if  $b \in A$  then if  $A = \{b\}$  then  $?term b b$  else  $(?term b b) \sqcap STOP$  else
STOP)  $\rangle$ 

```

if $\langle b \in B \rangle$ **for** b
proof (*split if-split, intro conjI impI*)
show $\langle \Box a \in A. (if\ a = b\ then\ ?term\ a\ b\ else\ STOP) =$
 $(if\ A = \{b\}\ then\ ?term\ b\ b\ else\ (?term\ b\ b) \sqcap STOP) \rangle$ **if** $\langle b \in A \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle A = \{b\} \implies \Box a \in A. (if\ a = b\ then\ ?term\ a\ b\ else\ STOP) = ?term$
 $b\ b \rangle$ **by** *simp*
next
assume $\langle A \neq \{b\} \rangle$
with $\langle b \in A \rangle$ **have** $\langle insert\ b\ A = A \rangle \langle A - \{b\} \neq \{\} \rangle$ **by** *auto*
show $\langle A \neq \{b\} \implies \Box a \in A. (if\ a = b\ then\ ?term\ a\ b\ else\ STOP) = ?term$
 $b\ b \sqcap STOP \rangle$
by (*auto simp add: GlobalNdet-is-STOP-iff intro!: arg-cong2[where f =*
 $\langle (\Box) \rangle$
 $simp\ flip: GlobalNdet-factorization-union$
 $[of\ \langle \{b\} \rangle, OF - \langle A - \{b\} \neq \{\} \rangle, simplified, unfolded\ \langle insert\ b\ A =$
 $A \rangle]$)
qed
next
show $\langle b \notin A \implies \Box a \in A. (if\ a = b\ then\ ?term\ a\ b\ else\ STOP) = STOP \rangle$
by (*auto simp add: GlobalNdet-is-STOP-iff*)
qed

hence $\langle \Box b \in B. \Box a \in A. ?rhs'\ b\ a =$
 $\Box b \in B. (if\ b \in A\ then\ if\ A = \{b\}\ then\ ?term\ b\ b\ else\ (?term\ b\ b) \sqcap$
 $STOP\ else\ STOP) \rangle$
by (*fact mono-GlobalNdet-eq*)
also from $\langle B - A \neq \{\} \rangle$ **have** $\langle \dots = (\Box b \in B. (if\ b \in A\ then\ ?term\ b\ b\ else$
 $STOP)) \sqcap STOP \rangle$
by (*simp add: Process-eq-spec GlobalNdet-projs, safe*)
 $(simp-all\ add: GlobalNdet-projs\ STOP-projs\ Ndet-projs\ split: if-split-asm,$
 $auto)$
also have $\langle \dots = ?rhs2 \rangle$
proof (*fold GlobalNdet-factorization-union*
 $[OF\ \langle B \cap A \neq \{\} \rangle \langle B - A \neq \{\} \rangle, unfolded\ Int-Diff-Un])$
have $\langle \Box b \in (B \cap A). (if\ b \in A\ then\ ?term\ b\ b\ else\ STOP) =$
 $\Box b \in (B \cap A). ?term\ b\ b \rangle$ **by** (*auto intro: mono-GlobalNdet-eq*)
moreover have $\langle \Box b \in (B - A). (if\ b \in A\ then\ ?term\ b\ b\ else\ STOP) =$
 $STOP \rangle$
by (*simp add: GlobalNdet-is-STOP-iff*)
ultimately show $\langle (\Box b \in (B \cap A). (if\ b \in A\ then\ ?term\ b\ b\ else\ STOP)) \sqcap$
 $(\Box b \in (B - A). (if\ b \in A\ then\ ?term\ b\ b\ else\ STOP)) \sqcap STOP$
 $= ?rhs2 \rangle$
by (*metis Mndetprefix-GlobalNdet Int-commute Ndet-assoc Ndet-id*)
qed
finally show $\langle (\Box b \in B. \Box a \in A. ?rhs'\ b\ a) = ?rhs2 \rangle$.
qed
qed
finally show $\langle ?lhs = ?rhs2 \rangle$.

qed

lemma *Mndetprefix-Sync_{ptick}-Mndetprefix-indep* :

$\langle A \cap S = \{\} \implies B \cap S = \{\} \implies$
 $\Box a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q\ b =$
 $(\text{ if } A = \{\} \text{ then } \Box b \in B \rightarrow (STOP\ \llbracket S \rrbracket_{\checkmark}\ Q\ b)$
 $\text{ else if } B = \{\} \text{ then } \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP)$
 $\text{ else } \Box b \in B. \Box a \in A.$
 $((a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ b \rightarrow Q\ b))) \Box$
 $((b \rightarrow (a \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ b)))) \rangle$

by (*simp add: Mndetprefix-Sync_{ptick}-STOP STOP-Sync_{ptick}-Mndetprefix*)
(auto simp add: Mndetprefix-GlobalNdet Sync_{ptick}-distrib-GlobalNdet-left
Sync_{ptick}-distrib-GlobalNdet-right disjoint-iff write0-def
Mprefix-Sync_{ptick}-Mprefix Int-assoc insert-Diff-if
intro!: mono-GlobalNdet-eq)

lemma *Mndetprefix-Sync_{ptick}-Mndetprefix-left* :

$\langle \Box a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q\ b = \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q\ b) \rangle$
 $(\text{is } \langle ?lhs = ?rhs \rangle \text{ if } \langle A \cap S = \{\} \rangle \langle B \subseteq S \rangle)$

proof –

let $?rhs' = \langle \Box b \in B. \Box a \in A. a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ b \rightarrow Q\ b) \rangle$
have $\langle ?lhs = ($
 $\text{ if } A = \{\} \text{ then if } B \cap S = \{\} \text{ then } \Box b \in B \rightarrow (STOP\ \llbracket S \rrbracket_{\checkmark}\ Q\ b)$
 $\text{ else } (\Box x \in (B - S) \rightarrow (STOP\ \llbracket S \rrbracket_{\checkmark}\ Q\ x)) \Box STOP$
 $\text{ else if } B = \{\} \text{ then if } A \cap S = \{\} \text{ then } \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP)$
 $\text{ else } (\Box x \in (A - S) \rightarrow (P\ x\ \llbracket S \rrbracket_{\checkmark}\ STOP)) \Box STOP$
 $\text{ else } \Box b \in B. \Box a \in A.$
 $(\text{if } a \in S \text{ then } STOP \text{ else } (a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ b \rightarrow Q\ b))) \Box$
 $(\text{if } b \in S \text{ then } STOP \text{ else } (b \rightarrow (a \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ b))) \Box$
 $(\text{if } a = b \wedge b \in S \text{ then } b \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ b) \text{ else } STOP) \rangle$
 $(\text{is } \langle ?lhs = (\text{if } A = \{\} \text{ then } ?rhs1 \text{ else if } B = \{\} \text{ then } ?rhs2 \text{ else } ?rhs3) \rangle)$

by (*fact Mndetprefix-Sync_{ptick}-Mndetprefix*)

also from $\langle B \subseteq S \rangle$ **have** $\langle ?rhs1 = STOP \rangle$

by (*auto simp add: Ndet-is-STOP-iff Mndetprefix-GlobalNdet GlobalNdet-is-STOP-iff*)

also from $\langle A \cap S = \{\} \rangle$ **have** $\langle ?rhs2 = \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP) \rangle$ **by**

presburger

also from $\langle A \cap S = \{\} \rangle \langle B \subseteq S \rangle$

have $\langle ?rhs3 = \Box b \in B. \Box a \in A. a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ b \rightarrow Q\ b) \rangle$

by (*intro mono-GlobalNdet-eq auto*)

finally have $\langle ?lhs = ($
 $\text{ if } A = \{\} \text{ then } STOP$
 $\text{ else if } B = \{\} \text{ then } \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP)$
 $\text{ else } ?rhs' \rangle$

moreover have $\langle B \neq \{\} \implies ?rhs' = \Box a \in A. a \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} \Box b \in B. b \rightarrow Q\ b) \rangle$
by (*subst GlobalNdet-sets-commute*)
(simp add: Sync_{ptick}-distrib-GlobalNdet-left write0-GlobalNdet)

moreover have $\langle \dots = \Box a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q\ b) \rangle$
by (*simp add: Mndetprefix-GlobalNdet*)

ultimately show $\langle ?lhs = ?rhs \rangle$ **by** *simp*

qed

end

corollary (in $\text{Sync}_{ptick}\text{-locale}$) $\text{Mndetprefix-Sync}_{ptick}\text{-Mndetprefix-right}$:
 $\langle \Box a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} \Box b \in B \rightarrow Q\ b = \Box b \in B \rightarrow (\Box a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} Q\ b) \rangle$
(is $\langle ?lhs = ?rhs \rangle$) **if** $\langle A \subseteq S \rangle \langle B \cap S = \{\} \rangle$
by (*subst* (1 2) $\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}\text{-sym}$)
(simp add: Sync_{ptick}-locale-sym.Mndetprefix-Sync_{ptick}-Mndetprefix-left that)

8.3 Read and Write Laws

8.3.1 Sequential Composition

lemma read-Seq_{ptick} : $\langle c?a \in A \rightarrow P\ a ;_{\checkmark} Q = c?a \in A \rightarrow (P\ a ;_{\checkmark} Q) \rangle$
by (*simp add: read-def Mprefix-Seq_{ptick} comp-def*)

lemma $\text{write0-Seq}_{ptick}$: $\langle a \rightarrow P ;_{\checkmark} Q = a \rightarrow (P ;_{\checkmark} Q) \rangle$
by (*simp add: write0-def Mprefix-Seq_{ptick}*)

lemma $\text{ndet-write-Seq}_{ptick}$: $\langle c!a \in A \rightarrow P\ a ;_{\checkmark} Q = c!a \in A \rightarrow (P\ a ;_{\checkmark} Q) \rangle$
by (*simp add: ndet-write-is-GlobalNdet-write0 Seq_{ptick}-distrib-GlobalNdet-right write0-Seq_{ptick}*)

lemma write-Seq_{ptick} : $\langle c!a \rightarrow P ;_{\checkmark} Q = c!a \rightarrow (P ;_{\checkmark} Q) \rangle$
by (*simp add: write0-Seq_{ptick} write-is-write0*)

8.3.2 Synchronization Product

General Laws

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

read **lemma** $\text{read-Sync}_{ptick}\text{-read}$:
 — This is the general case.
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b =$
 $(c?a \in (A - c - 'S) \rightarrow (P\ a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b)) \Box$
 $(d?b \in (B - d - 'S) \rightarrow (c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} Q\ b)) \Box$
 $(\Box x \in (c - 'A \cap d - 'B \cap S) \rightarrow (P\ (\text{inv-into } A\ c\ x) \llbracket S \rrbracket_{\checkmark} Q\ (\text{inv-into } B\ d\ x))) \rangle$
(is $\langle ?lhs = ?rhs1 \Box ?rhs2 \Box ?rhs3 \rangle$)
if $\langle c - 'A \cap S = \{\} \vee c - 'A \subseteq S \vee \text{inj-on } c\ A \rangle$
 $\langle d - 'B \cap S = \{\} \vee d - 'B \subseteq S \vee \text{inj-on } d\ B \rangle$

— 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 $\text{inv-into } (A \setminus c - ' S) c$ is equal to $\text{inv-into } A c$. This requires $A \setminus c - ' S = A$ (which is equivalent to $c - ' A \cap S = \{\}$) or $\text{inj-on } c A$. We need obviously a similar assumption for B .

proof —

have $*$: $\langle \bigwedge e X. e - ' (X - e - ' S) = e - ' X - S \rangle$ **by** *auto*

have $\langle ?lhs = (\Box a \in (c - ' A - S) \rightarrow (P (\text{inv-into } A c a) \llbracket S \rrbracket_{\checkmark} (\Box x \in d - ' B \rightarrow Q (\text{inv-into } B d x)))) \Box$
 $(\Box b \in (d - ' B - S) \rightarrow ((\Box x \in c - ' A \rightarrow P (\text{inv-into } A c x)) \llbracket S \rrbracket_{\checkmark} Q (\text{inv-into } B d b))) \Box$
 $(\Box x \in (c - ' A \cap d - ' B \cap S) \rightarrow (P (\text{inv-into } A c x) \llbracket S \rrbracket_{\checkmark} Q (\text{inv-into } B d x))) \rangle$

(is $\langle ?lhs = ?rhs1' \Box ?rhs2' \Box ?rhs3 \rangle$)

by (*simp add: read-def Mprefix-Sync_{ptick}-Mprefix comp-def*)

also from *that(1)* **have** $\langle ?rhs1' = ?rhs1 \rangle$

proof (*elim disjE*)

assume $\langle c - ' A \cap S = \{\} \rangle$

hence $\langle A - c - ' S = A \wedge c - ' A - S = c - ' A \rangle$ **by** *fast*

thus $\langle ?rhs1' = ?rhs1 \rangle$ **by** (*simp add: read-def comp-def*)

next

assume $\langle c - ' A \subseteq S \rangle$

hence $\langle A - c - ' S = \{\} \wedge c - ' A - S = \{\} \rangle$ **by** *fast*

show $\langle ?rhs1' = ?rhs1 \rangle$ **by** (*simp add: <?this>*)

next

assume $\langle \text{inj-on } c A \rangle$

hence $\langle \text{inj-on } c (A - c - ' S) \rangle$ **by** (*simp add: inj-on-diff*)

with $\langle \text{inj-on } c A \rangle$ **show** $\langle ?rhs1' = ?rhs1 \rangle$

by (*auto simp add: read-def comp-def * intro: mono-Mprefix-eq*)

qed

also from *that(2)* **have** $\langle ?rhs2' = ?rhs2 \rangle$

proof (*elim disjE*)

assume $\langle d - ' B \cap S = \{\} \rangle$

hence $\langle B - d - ' S = B \wedge d - ' B - S = d - ' B \rangle$ **by** *fast*

thus $\langle ?rhs2' = ?rhs2 \rangle$ **by** (*simp add: read-def comp-def*)

next

assume $\langle d - ' B \subseteq S \rangle$

hence $\langle B - d - ' S = \{\} \wedge d - ' B - S = \{\} \rangle$ **by** *fast*

show $\langle ?rhs2' = ?rhs2 \rangle$ **by** (*simp add: <?this>*)

next

assume $\langle \text{inj-on } d B \rangle$

hence $\langle \text{inj-on } d (B - d - ' S) \rangle$ **by** (*simp add: inj-on-diff*)

with $\langle \text{inj-on } d B \rangle$ **show** $\langle ?rhs2' = ?rhs2 \rangle$

by (*auto simp add: read-def comp-def * intro: mono-Mprefix-eq*)

qed

finally show $\langle ?lhs = ?rhs1 \Box ?rhs2 \Box ?rhs3 \rangle$.

qed

Enforce read lemma *read-Sync_{ptick}-read-forced-read-left* :

$\langle c? a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} d? b \in B \rightarrow Q b =$

$(c?a \in (A - c - 'S) \rightarrow (P a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q b)) \square$
 $(d?b \in (B - d - 'S) \rightarrow (c?a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b)) \square$
 $(c?x \in (A \cap c - ' (d ' B \cap S)) \rightarrow (P x \llbracket S \rrbracket_{\checkmark} Q x)) \rangle$
(is $\langle ?lhs = ?rhs1 \square ?rhs2 \square ?rhs3 \rangle$
if $\langle c ' A \cap S = \{\} \vee inj-on\ c\ A \rangle$
 $\langle d ' B \cap S = \{\} \vee inj-on\ d\ B \rangle$
 $\langle \bigwedge a\ b. a \in A \implies b \in B \implies c\ a = d\ b \implies d\ b \in S \implies a = b \rangle$
proof $-$
let $?rhs3' = \langle (\square x \in (c ' A \cap d ' B \cap S) \rightarrow (P (inv-into\ A\ c\ x) \llbracket S \rrbracket_{\checkmark} Q (inv-into\ B\ d\ x))) \rangle$
have $*$: $\langle c ' (A \cap c - ' (d ' B \cap S)) = c ' A \cap d ' B \cap S \rangle$ **by** *blast*
have $**$: $\langle c ' (A \cap c - ' d ' B) = c ' A \cap d ' B \rangle$ **by** *blast*
from *that(1, 2)* **consider** $\langle c ' A \cap S = \{\} \vee d ' B \cap S = \{\} \rangle$
 $| \langle inj-on\ c\ A \rangle \langle inj-on\ d\ B \rangle$ **by** *blast*
hence $\langle ?rhs3' = ?rhs3 \rangle$
proof *cases*
assume $\langle c ' A \cap S = \{\} \vee d ' B \cap S = \{\} \rangle$
hence $\langle c ' A \cap d ' B \cap S = \{\} \wedge A \cap c - ' (d ' B \cap S) = \{\} \rangle$ **by** *blast*
thus $\langle ?rhs3' = ?rhs3 \rangle$ **by** *simp*
next
assume $\langle inj-on\ c\ A \rangle \langle inj-on\ d\ B \rangle$
show $\langle ?rhs3' = ?rhs3 \rangle$
proof (*unfold read-def * comp-def,*
intro mono-Mprefix-eq arg-cong2 [**where** $f = \langle \lambda P\ Q. P \llbracket S \rrbracket_{\checkmark} Q \rangle$])
fix x **assume** $\langle x \in c ' A \cap d ' B \cap S \rangle$
moreover from $\langle inj-on\ c\ A \rangle$ *inj-on-Int*
have $\langle inj-on\ c\ A \wedge inj-on\ c\ (A \cap c - ' (d ' B \cap S)) \rangle$ **by** *blast*
ultimately show $\langle P (inv-into\ A\ c\ x) = P (inv-into\ (A \cap c - ' (d ' B \cap S))\ c\ x) \rangle$
by (*simp add: image-iff, elim conjE bexE, simp*)
next
fix x **assume** $\$: \langle x \in c ' A \cap d ' B \cap S \rangle$
then obtain $a\ b$ **where** $\$ \$: \langle x = c\ a \rangle \langle a \in A \rangle \langle x = d\ b \rangle \langle b \in B \rangle$ **by** *blast*
from $\langle inj-on\ c\ A \rangle$ *inj-on-Int* **have** $\$ \$ \$: \langle inj-on\ c\ (A \cap c - ' (d ' B \cap S)) \rangle$
by *blast*
have $\langle inv-into\ B\ d\ x = b \rangle$ **by** (*simp add: \\$ \\$ (3, 4) inj-on\ d\ B*)
also have $\langle b = a \rangle$ **by** (*metis \\$ \\$ Int-iff that(3)*)
also have $\langle a = inv-into\ (A \cap c - ' (d ' B \cap S))\ c\ x \rangle$
by (*metis \\$ \\$ (1, 2) \\$ \\$ \\$ * Int-lower1*
 $\langle inj-on\ c\ A \rangle$ *inj-on-image-mem-iff inv-into-f-eq*)
finally have $\langle inv-into\ B\ d\ x = inv-into\ (A \cap c - ' (d ' B \cap S))\ c\ x \rangle$.
thus $\langle Q (inv-into\ B\ d\ x) = Q (inv-into\ (A \cap c - ' (d ' B \cap S))\ c\ x) \rangle$ **by** *simp*
qed
qed
moreover have $\langle ?lhs = ?rhs1 \square ?rhs2 \square ?rhs3' \rangle$
using *that(1, 2)* **by** (*subst read-Sync_{ptick}-read*) *auto*
ultimately show $\langle ?lhs = ?rhs1 \square ?rhs2 \square ?rhs3 \rangle$ **by** *argo*
qed

corollary (in $\text{Sync}_{ptick}\text{-locale}$) *read-Sync_{ptick}-read-forced-read-right*:

$\langle c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \ d?b \in B \rightarrow Q\ b =$
 $(c?a \in (A - c - 'S) \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} \ d?b \in B \rightarrow Q\ b)) \square$
 $(d?b \in (B - d - 'S) \rightarrow (c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \ Q\ b)) \square$
 $(d?x \in (B \cap d - ' (c - 'A \cap S)) \rightarrow (P\ x\ \llbracket S \rrbracket_{\checkmark} \ Q\ x)) \rangle$
(is $\langle ?lhs = ?rhs1 \square ?rhs2 \square ?rhs3 \rangle$
if $\langle c - 'A \cap S = \{\} \vee \text{inj-on } c\ A \rangle$
 $\langle d - 'B \cap S = \{\} \vee \text{inj-on } d\ B \rangle$
 $\langle \bigwedge a\ b. a \in A \implies b \in B \implies c\ a = d\ b \implies d\ b \in S \implies a = b \rangle$
unfolding $\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}\text{-sym}$
by (*subst Sync_{ptick}-locale-sym.read-Sync_{ptick}-read-forced-read-left*[*OF that(2, 1)*],
metis that(3))
(auto simp add: Det-commute intro: arg-cong2[where f = $\langle \square \rangle$])

Special Cases lemma *read-Sync_{ptick}-read-subset* :

$\langle c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \ d?b \in B \rightarrow Q\ b =$
 $\square x \in (c - 'A \cap d - 'B) \rightarrow (P\ (\text{inv-into } A\ c\ x)\ \llbracket S \rrbracket_{\checkmark} \ Q\ (\text{inv-into } B\ d\ x)) \rangle$
if $\langle c - 'A \subseteq S \rangle \langle d - 'B \subseteq S \rangle$

proof –

from that have $*$: $\langle A - c - 'S = \{\} \rangle \langle B - d - 'S = \{\} \rangle$ **by** *auto*
from that(1) have $**$: $\langle c - 'A \cap d - 'B \cap S = c - 'A \cap d - 'B \rangle$ **by** *blast*
show *?thesis* **by** (*subst read-Sync_{ptick}-read*)
*(use that in <simp-all add: * **>)*

qed

lemma *read-Sync_{ptick}-read-subset-forced-read-left* :

$\langle c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \ d?b \in B \rightarrow Q\ b = c?x \in (A \cap c - 'd - 'B) \rightarrow (P\ x\ \llbracket S \rrbracket_{\checkmark} \ Q\ x) \rangle$
if $\langle c - 'A \subseteq S \rangle \langle d - 'B \subseteq S \rangle \langle \text{inj-on } c\ A \rangle \langle \text{inj-on } d\ B \rangle$
 $\langle \bigwedge a\ b. a \in A \implies b \in B \implies c\ a = d\ b \implies d\ b \in S \implies a = b \rangle$

proof –

from that have $*$: $\langle A - c - 'S = \{\} \rangle \langle B - d - 'S = \{\} \rangle$ **by** *auto*
from that(1) have $**$: $\langle A \cap (c - 'd - 'B \cap c - 'S) = A \cap c - 'd - 'B \rangle$ **by** *blast*
show *?thesis* **by** (*subst read-Sync_{ptick}-read-forced-read-left*)
*(use that(3, 4, 5) in <simp-all add: * **>)*

qed

lemma *read-Sync_{ptick}-read-subset-forced-read-right* :

$\langle c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} \ d?b \in B \rightarrow Q\ b = d?x \in (B \cap d - 'c - 'A) \rightarrow (P\ x\ \llbracket S \rrbracket_{\checkmark} \ Q\ x) \rangle$
if $\langle c - 'A \subseteq S \rangle \langle d - 'B \subseteq S \rangle \langle \text{inj-on } c\ A \rangle \langle \text{inj-on } d\ B \rangle$
 $\langle \bigwedge a\ b. a \in A \implies b \in B \implies c\ a = d\ b \implies d\ b \in S \implies a = b \rangle$

proof –

from that have $*$: $\langle B - d - 'S = \{\} \rangle \langle A - c - 'S = \{\} \rangle$ **by** *auto*
from that(1) have $**$: $\langle B \cap (d - 'c - 'A \cap d - 'S) = B \cap d - 'c - 'A \rangle$ **by** *blast*
show *?thesis* **by** (*subst read-Sync_{ptick}-read-forced-read-right*)
*(use that(3, 4, 5) in <simp-all add: * **>)*

qed

lemma *read-Sync_{ptick}-read-indep* :
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b =$
 $(c?a \in A \rightarrow (P\ a \llbracket S \rrbracket_{\checkmark} (d?b \in B \rightarrow Q\ b))) \sqcap (d?b \in B \rightarrow ((c?a \in A \rightarrow P\ a) \llbracket S \rrbracket_{\checkmark} Q\ b)) \rangle$
if $\langle c\ 'A \cap S = \{\}\rangle \langle d\ 'B \cap S = \{\}\rangle$
proof –
from *that* **have** $*$: $\langle A - c - 'S = A \rangle \langle B - d - 'S = B \rangle$ **by** *auto*
from *that*(1) **have** $**$: $\langle c\ 'A \cap d\ 'B \cap S = \{\}\rangle$ **by** *blast*
show *?thesis* **by** (*subst read-Sync_{ptick}-read*) (*use that in* $\langle \text{simp-all add: } ** \rangle$)
qed

lemma *read-Sync_{ptick}-read-left* :
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b = c?a \in A \rightarrow (P\ a \llbracket S \rrbracket_{\checkmark} (d?b \in B \rightarrow Q\ b)) \rangle$
if $\langle c\ 'A \cap S = \{\}\rangle \langle d\ 'B \subseteq S \rangle$
proof –
from *that*(1) **have** $*$: $\langle A - c - 'S = A \rangle \langle c\ 'A \cap d\ 'B \cap S = \{\}\rangle$ **by** *auto*
from *that*(2) **have** $**$: $\langle B - d - 'S = \{\}\rangle$ **by** *blast*
show *?thesis* **by** (*subst read-Sync_{ptick}-read*)
(use that in $\langle \text{simp-all add: } ** \rangle$ *)*
qed

lemma *read-Sync_{ptick}-read-right* :
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b = d?b \in B \rightarrow ((c?a \in A \rightarrow P\ a) \llbracket S \rrbracket_{\checkmark} Q\ b) \rangle$
if $\langle c\ 'A \subseteq S \rangle \langle d\ 'B \cap S = \{\}\rangle$
proof –
from *that*(2) **have** $*$: $\langle B - d - 'S = B \rangle \langle c\ 'A \cap d\ 'B \cap S = \{\}\rangle$ **by** *auto*
from *that*(1) **have** $**$: $\langle A - c - 'S = \{\}\rangle$ **by** *blast*
show *?thesis* **by** (*subst read-Sync_{ptick}-read*)
(use that in $\langle \text{simp-all add: } ** \rangle$ *)*
qed

corollary *read-Par_{ptick}-read* :
 $\langle c?a \in A \rightarrow P\ a \parallel_{\checkmark} d?b \in B \rightarrow Q\ b =$
 $\sqcap x \in (c\ 'A \cap d\ 'B) \rightarrow (P\ (inv\ into\ A\ c\ x) \parallel_{\checkmark} Q\ (inv\ into\ B\ d\ x)) \rangle$
by (*simp add: read-Sync_{ptick}-read-subset*)

corollary *read-Par_{ptick}-read-forced-read-left* :
 $\langle \llbracket inj\ on\ c\ A; inj\ on\ d\ B; \bigwedge a\ b. a \in A \implies b \in B \implies c\ a = d\ b \implies a = b \rrbracket \implies$
 $c?a \in A \rightarrow P\ a \parallel_{\checkmark} d?b \in B \rightarrow Q\ b = c?x \in (A \cap c - 'd\ 'B) \rightarrow (P\ x \parallel_{\checkmark} Q\ x) \rangle$
by (*subst read-Sync_{ptick}-read-forced-read-left*) *simp-all*

corollary *read-Par_{ptick}-read-forced-read-right* :
 $\langle \llbracket inj\ on\ c\ A; inj\ on\ d\ B; \bigwedge a\ b. a \in A \implies b \in B \implies c\ a = d\ b \implies a = b \rrbracket \implies$
 $c?a \in A \rightarrow P\ a \parallel_{\checkmark} d?b \in B \rightarrow Q\ b = d?x \in (B \cap d - 'c\ 'A) \rightarrow (P\ x \parallel_{\checkmark} Q\ x) \rangle$
by (*subst read-Sync_{ptick}-read-forced-read-right*) *simp-all*

corollary *read-Inter_{ptick}-read* :

$\langle \llbracket \text{inj-on } c \ A; \text{inj-on } d \ B; \bigwedge a \ b. a \in A \implies b \in B \implies c \ a = d \ b \implies a = b \rrbracket \implies$
 $c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b =$
 $(c?a \in A \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b)) \sqcap (c?b \in B \rightarrow (c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ Q \ b)) \rangle$
by (*simp add: read-Sync_{ptick}-read*)

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

lemma *read-Sync_{ptick}-read-forced-read-same-chan* :

$\langle c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b =$
 $(c?a \in (A - c - 'S) \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b)) \sqcap$
 $(c?b \in (B - c - 'S) \rightarrow (c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ Q \ b)) \sqcap$
 $(c?x \in (A \cap B \cap c - 'S) \rightarrow (P \ x \ \llbracket S \rrbracket_{\checkmark} \ Q \ x)) \rangle$
(is $\langle ?lhs = ?rhs1 \sqcap ?rhs2 \sqcap ?rhs3 \rangle$ **)**
if $\langle c - 'A \cap S = \{\} \vee \text{inj-on } c \ A \rangle \langle c - 'B \cap S = \{\} \vee \text{inj-on } c \ B \rangle$
 $\langle \bigwedge a \ b. a \in A \implies b \in B \implies c \ a = c \ b \implies c \ b \in S \implies a = b \rangle$

proof –

— Actually, the third assumption is equivalent to the following (we of course do not use that(3) in the proof of equivalence).

from *that(1, 2)*

have $\langle \text{inj-on } c \ ((A \cup B) \cap c - 'S) \longleftrightarrow$
 $(\forall a \ b. a \in A \longrightarrow b \in B \longrightarrow c \ a = c \ b \longrightarrow c \ b \in S \longrightarrow a = b) \rangle$

by (*elim disjE, simp-all add: inj-on-def*)

$((\text{auto})[3], \text{metis Int-iff Un-iff vimageE vimageI})$

from *that(3)* **have** $*$: $\langle A \cap (c - 'c - 'B \cap c - 'S) = A \cap B \cap c - 'S \rangle$ **by** *auto blast*

show *?thesis* **by** (*simp add: read-Sync_{ptick}-read-forced-read-left that **)

qed

lemma *read-Sync_{ptick}-read-forced-read-same-chan-weaker* :

— Easier with a stronger assumption.

$\langle \text{inj-on } c \ (A \cup B) \implies$
 $c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b =$
 $(c?a \in (A - c - 'S) \rightarrow (P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b)) \sqcap$
 $(c?b \in (B - c - 'S) \rightarrow (c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ Q \ b)) \sqcap$
 $(c?x \in (A \cap B \cap c - 'S) \rightarrow (P \ x \ \llbracket S \rrbracket_{\checkmark} \ Q \ x)) \rangle$

by (*rule read-Sync_{ptick}-read-forced-read-same-chan*)

$(\text{simp-all add: inj-on-Un, metis Un-iff inj-onD inj-on-Un})$

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.

$\langle c?a \in A \rightarrow P \ a \ \llbracket S \rrbracket_{\checkmark} \ c?b \in B \rightarrow Q \ b = c?x \in (A \cap B) \rightarrow (P \ x \ \llbracket S \rrbracket_{\checkmark} \ Q \ x) \rangle$

if $\langle c - 'A \subseteq S \rangle \langle c - 'B \subseteq S \rangle \langle \text{inj-on } c \ (A \cup B) \rangle$

proof –

from *that(3)* **have** $\langle A \cap c - 'c - 'B = A \cap B \rangle$ **by** (*auto simp add: inj-on-def*)

with *that(3)* **show** *?thesis*

by (subst read-Sync_{ptick}-read-subset-forced-read-left)
 (simp-all add: that(1, 2) inj-on-Un, meson Un-iff inj-on-contrad that(3))
 qed

read and ndet-write. lemma ndet-write-Sync_{ptick}-read :
 $\langle c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ d?b \in B \rightarrow Q\ b =$
 $(\text{ if } A = \{\} \text{ then } STOP\ \llbracket S \rrbracket_{\checkmark}\ d?b \in B \rightarrow Q\ b$
 $\text{ else } \Box a \in c\ 'A. (\text{ if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P\ (\text{inv-into } A\ c\ a)\ \llbracket S \rrbracket_{\checkmark}\ d?b \in B$
 $\rightarrow Q\ b)) \Box$
 $(\Box b \in (d\ 'B - S) \rightarrow (a \rightarrow P\ (\text{inv-into } A\ c\ a)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B\ d$
 $b))) \Box$
 $(\text{ if } a \in d\ 'B \cap S \text{ then } a \rightarrow (P\ (\text{inv-into } A\ c\ a)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B$
 $d\ a)) \text{ else } STOP)) \rangle$
 by (auto simp add: ndet-write-def read-def Mndetprefix-Sync_{ptick}-Mprefix
 intro: mono-GlobalNdet-eq arg-cong2[where f = $\langle (\Box) \rangle$])

lemma read-Sync_{ptick}-ndet-write :
 $\langle c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ d!!b \in B \rightarrow Q\ b =$
 $(\text{ if } B = \{\} \text{ then } c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ STOP$
 $\text{ else } \Box b \in d\ 'B. (\text{ if } b \in S \text{ then } STOP \text{ else } b \rightarrow (c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B$
 $d\ b))) \Box$
 $(\Box a \in (c\ 'A - S) \rightarrow (P\ (\text{inv-into } A\ c\ a)\ \llbracket S \rrbracket_{\checkmark}\ b \rightarrow Q\ (\text{inv-into } B\ d$
 $b))) \Box$
 $(\text{ if } b \in c\ 'A \cap S \text{ then } b \rightarrow (P\ (\text{inv-into } A\ c\ b)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B$
 $d\ b)) \text{ else } STOP)) \rangle$
 by (auto simp add: ndet-write-def read-def Mprefix-Sync_{ptick}-Mndetprefix
 intro: mono-GlobalNdet-eq arg-cong2[where f = $\langle (\Box) \rangle$])

lemma ndet-write-Sync_{ptick}-read-subset :
 $\langle c\ 'A \subseteq S \implies d\ 'B \subseteq S \implies$
 $c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ d?b \in B \rightarrow Q\ b =$
 $(\text{ if } c\ 'A \subseteq d\ 'B \text{ then } \Box a \in c\ 'A \rightarrow (P\ (\text{inv-into } A\ c\ a)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B\ d$
 $a)))$
 $\text{ else } (\Box a \in (c\ 'A \cap d\ 'B) \rightarrow (P\ (\text{inv-into } A\ c\ a)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B\ d\ a))) \Box$
 $STOP) \rangle$
 by (simp add: read-def ndet-write-def Mndetprefix-Sync_{ptick}-Mprefix-subset)

lemma read-Sync_{ptick}-ndet-write-subset :
 $\langle c\ 'A \subseteq S \implies d\ 'B \subseteq S \implies$
 $c?a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark}\ d!!b \in B \rightarrow Q\ b =$
 $(\text{ if } d\ 'B \subseteq c\ 'A \text{ then } \Box b \in d\ 'B \rightarrow (P\ (\text{inv-into } A\ c\ b)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B\ d$
 $b)))$
 $\text{ else } (\Box b \in (c\ 'A \cap d\ 'B) \rightarrow (P\ (\text{inv-into } A\ c\ b)\ \llbracket S \rrbracket_{\checkmark}\ Q\ (\text{inv-into } B\ d\ b))) \Box$
 $STOP) \rangle$
 by (simp add: read-def ndet-write-def Mprefix-Sync_{ptick}-Mndetprefix-subset)

— If we have the same injective channel, it's better.

lemma *ndet-write-Sync_{ptick}-read-subset-same-chan*:

$\langle c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \ c?!b \in B \rightarrow Q \ b =$
 (if $A \subseteq B$ then $c!!a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \ Q \ a)$ else $(c!!a \in (A \cap B) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \ Q \ a)) \sqcap STOP \rangle$
 if $\langle c \ ' \ A \subseteq S \rangle \langle c \ ' \ B \subseteq S \rangle \langle inj\text{-}on \ c \ (A \cup B) \rangle$
proof –
 from $\langle inj\text{-}on \ c \ (A \cup B) \rangle$ **have** $*$: $\langle c \ ' \ A \subseteq c \ ' \ B \longleftrightarrow A \subseteq B \rangle$
 by (auto simp add: inj-on-eq-iff)
 from $\langle inj\text{-}on \ c \ (A \cup B) \rangle$ **have** $**$: $\langle c \ ' \ A \cap c \ ' \ B = c \ ' \ (A \cap B) \rangle$
 by (auto simp add: inj-on-Un)
 from $\langle inj\text{-}on \ c \ (A \cup B) \rangle$ **show** ?thesis
 by (unfold ndet-write-Sync_{ptick}-read-subset[OF $\langle c \ ' \ A \subseteq S \rangle \langle c \ ' \ B \subseteq S \rangle$] * **)
 (auto simp add: ndet-write-def inj-on-Un inj-on-Int
 intro!: mono-Mndetprefix-eq arg-cong2[where $f = \langle (\sqcap) \rangle$])

qed

corollary (in Sync_{ptick}-locale) *read-Sync_{ptick}-ndet-write-subset-same-chan*:

$\langle c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \ c!!b \in B \rightarrow Q \ b =$
 (if $B \subseteq A$ then $c!!b \in B \rightarrow (P \ b \llbracket S \rrbracket_{\checkmark} \ Q \ b)$ else $(c!!b \in (A \cap B) \rightarrow (P \ b \llbracket S \rrbracket_{\checkmark} \ Q \ b)) \sqcap STOP \rangle$
 if $\langle c \ ' \ A \subseteq S \rangle \langle c \ ' \ B \subseteq S \rangle \langle inj\text{-}on \ c \ (A \cup B) \rangle$
 by (subst (1 2 3) Sync_{ptick}-locale-sym.Sync_{ptick}-sym)
 (simp add: Sync_{ptick}-locale-sym.ndet-write-Sync_{ptick}-read-subset-same-chan
 [OF that(2, 1)] Un-commute Int-commute that(3))

lemma *ndet-write-Sync_{ptick}-read-indep* :

$\langle c \ ' \ A \cap S = \{\} \implies d \ ' \ B \cap S = \{\} \implies$
 $c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \ d?!b \in B \rightarrow Q \ b =$
 (if $A = \{\}$ then $d?!b \in B \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} \ Q \ b)$
 else $\sqcap a \in c \ ' \ A. (a \rightarrow (P \ (inv\text{-}into \ A \ c \ a) \llbracket S \rrbracket_{\checkmark} \ d?!b \in B \rightarrow Q \ b)) \sqcap$
 $(d?!b \in B \rightarrow (a \rightarrow P \ (inv\text{-}into \ A \ c \ a) \llbracket S \rrbracket_{\checkmark} \ Q \ b))) \rangle$
 by (auto simp add: ndet-write-def read-def Mndetprefix-Sync_{ptick}-Mprefix-indep
 comp-def
 intro: mono-GlobalNdet-eq arg-cong2[where $f = \langle (\sqcap) \rangle$])

lemma *read-Sync_{ptick}-ndet-write-indep* :

$\langle c \ ' \ A \cap S = \{\} \implies d \ ' \ B \cap S = \{\} \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \ d!!b \in B \rightarrow Q \ b =$
 (if $B = \{\}$ then $c?a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \ STOP)$
 else $\sqcap b \in d \ ' \ B. (b \rightarrow (c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \ Q \ (inv\text{-}into \ B \ d \ b))) \sqcap$
 $(c?a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \ b \rightarrow Q \ (inv\text{-}into \ B \ d \ b))) \rangle$
 by (auto simp add: ndet-write-def read-def Mprefix-Sync_{ptick}-Mndetprefix-indep
 comp-def
 intro: mono-GlobalNdet-eq arg-cong2[where $f = \langle (\sqcap) \rangle$])

lemma *ndet-write-Sync_{ptick}-read-left* :

$\langle c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \ d?!b \in B \rightarrow Q \ b = c!!a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \ d?!b \in B \rightarrow Q \ b) \rangle$

(is $\langle ?lhs = ?rhs \rangle$) if $\langle c \text{ ' } A \cap S = \{\} \rangle \langle d \text{ ' } B \subseteq S \rangle$
proof –
 from that have $\langle ?lhs = (if\ A = \{\} \text{ then } STOP \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b \text{ else } ?rhs) \rangle$
 by (auto simp add: ndet-write-def read-def
 Mndetprefix-Sync_{ptick}-Mprefix-left comp-def
 intro: mono-GlobalNdet-eq arg-cong2[where $f = \langle (\square) \rangle$])
 also have $\langle \dots = ?rhs \rangle$
 by (simp add: read-def ndet-write-def Mprefix-is-STOP-iff
 STOP-Sync_{ptick}-Mprefix that(2))
 finally show $\langle ?lhs = ?rhs \rangle$.
qed

lemma read-Sync_{ptick}-ndet-write-left :
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b = c?a \in A \rightarrow (P\ a \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b) \rangle$
 (is $\langle ?lhs = ?rhs \rangle$) if $\langle c \text{ ' } A \cap S = \{\} \rangle \langle d \text{ ' } B \subseteq S \rangle$
proof –
 from that have $\langle ?lhs = (if\ B = \{\} \text{ then } (c?a \in A \rightarrow P\ a) \llbracket S \rrbracket_{\checkmark} STOP \text{ else } ?rhs) \rangle$
 by (auto simp add: ndet-write-def read-def
 Mprefix-Sync_{ptick}-Mndetprefix-left comp-def
 intro: mono-GlobalNdet-eq arg-cong2[where $f = \langle (\square) \rangle$])
 also have $\langle \dots = ?rhs \rangle$
 by (simp add: read-def comp-def)
 (use Mprefix-Sync_{ptick}-Mprefix-left that(1) in force)
 finally show $\langle ?lhs = ?rhs \rangle$.
qed

corollary (in Sync_{ptick}-locale) ndet-write-Sync_{ptick}-read-right :
 $\langle c!!a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b = d?b \in B \rightarrow (c!!a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} Q\ b) \rangle$
 if $\langle c \text{ ' } A \subseteq S \rangle \langle d \text{ ' } B \cap S = \{\} \rangle$
 by (subst (1 2) Sync_{ptick}-locale-sym.Sync_{ptick}-sym)
 (simp add: Sync_{ptick}-locale-sym.read-Sync_{ptick}-ndet-write-left[OF that(2, 1)])

corollary (in Sync_{ptick}-locale) read-Sync_{ptick}-ndet-write-right :
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b = d!!b \in B \rightarrow (c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} Q\ b) \rangle$
 if $\langle c \text{ ' } A \subseteq S \rangle \langle d \text{ ' } B \cap S = \{\} \rangle$
 by (subst (1 2) Sync_{ptick}-locale-sym.Sync_{ptick}-sym)
 (simp add: Sync_{ptick}-locale-sym.ndet-write-Sync_{ptick}-read-left[OF that(2, 1)])

read and write. lemma write-Sync_{ptick}-read :
 $\langle c!a \rightarrow P\ \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b =$
 (if $c\ a \in S$ then $STOP$ else $c!a \rightarrow (P\ \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q\ b)$) $\square$
 ($\square b \in (d \text{ ' } B - S) \rightarrow (c!a \rightarrow P\ \llbracket S \rrbracket_{\checkmark} Q\ (inv\ into\ B\ d\ b))$) $\square$
 (if $c\ a \in d \text{ ' } B \cap S$ then $c!a \rightarrow (P\ \llbracket S \rrbracket_{\checkmark} Q\ (inv\ into\ B\ d\ (c\ a)))$ else $STOP$)
 by (subst ndet-write-Sync_{ptick}-read[where $A = \langle \{a\} \rangle$, simplified])
 (simp add: write-is-write0 image-iff)

lemma read-Sync_{ptick}-write :
 $\langle c?a \in A \rightarrow P\ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q =$

(if $d \ b \in S$ then $STOP$ else $d!b \rightarrow (c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} Q)$) $\square$
 ($\square a \in (c \ ' \ A - S) \rightarrow (P \ (inv\text{-}into \ A \ c \ a) \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q)$) $\square$
 (if $d \ b \in c \ ' \ A \cap S$ then $d!b \rightarrow (P \ (inv\text{-}into \ A \ c \ (d \ b)) \llbracket S \rrbracket_{\checkmark} Q)$ else $STOP$)
by (subst read-Sync_{ptick}-ndet-write[**where** $B = \langle \{b\} \rangle$, simplified])
 (simp add: write-is-write0 image-iff)

lemma write-Sync_{ptick}-read-subset :

$\langle c \ a \in S \implies d \ ' \ B \subseteq S \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b =$
 (if $c \ a \in d \ ' \ B$ then $c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \ (inv\text{-}into \ B \ d \ (c \ a)))$ else $STOP$)
by (simp add: write-Sync_{ptick}-read)
 (metis Det-STOP Det-commute Diff-eq-empty-iff Mprefix-empty)

lemma read-Sync_{ptick}-write-subset :

$\langle c \ ' \ A \subseteq S \implies d \ b \in S \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q =$
 (if $d \ b \in c \ ' \ A$ then $d!b \rightarrow (P \ (inv\text{-}into \ A \ c \ (d \ b)) \llbracket S \rrbracket_{\checkmark} Q)$ else $STOP$)
by (simp add: read-Sync_{ptick}-write)
 (metis Diff-eq-empty-iff Mprefix-empty STOP-Det)

— If we have the same injective channel, it's better.

lemma write-Sync_{ptick}-read-subset-same-chan:

$\langle c \ a \in S \implies c \ ' \ B \subseteq S \implies inj\text{-}on \ c \ (insert \ a \ B) \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} c?b \in B \rightarrow Q \ b = (if \ a \in B \text{ then } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \ a) \text{ else } STOP)$
by (subst ndet-write-Sync_{ptick}-read-subset-same-chan[**where** $A = \langle \{a\} \rangle$, simplified]) simp-all

lemma read-Sync_{ptick}-write-subset-same-chan:

$\langle c \ ' \ A \subseteq S \implies c \ b \in S \implies inj\text{-}on \ c \ (insert \ b \ A) \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} c!b \rightarrow Q = (if \ b \in A \text{ then } c!b \rightarrow (P \ b \llbracket S \rrbracket_{\checkmark} Q) \text{ else } STOP)$
by (subst read-Sync_{ptick}-ndet-write-subset-same-chan[**where** $B = \langle \{b\} \rangle$, simplified]) simp-all

lemma write-Sync_{ptick}-read-indep :

$\langle c \ a \notin S \implies d \ ' \ B \cap S = \{\} \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b =$
 ($c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b)$) $\square$ ($d?b \in B \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \ b)$)
by (subst ndet-write-Sync_{ptick}-read-indep[**where** $A = \langle \{a\} \rangle$, simplified])
 (simp-all add: write-is-write0)

lemma read-Sync_{ptick}-write-indep :

$\langle c \ ' \ A \cap S = \{\} \implies d \ b \notin S \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q =$
 ($d!b \rightarrow (c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} Q)$) $\square$ ($c?a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q)$)
by (subst read-Sync_{ptick}-ndet-write-indep[**where** $B = \langle \{b\} \rangle$, simplified])
 (simp-all add: write-is-write0)

lemma *write-Sync_{ptick}-read-left* :

$\langle c \ a \notin S \implies d \ ' \ B \subseteq S \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b = c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b) \rangle$
by (*subst ndet-write-Sync_{ptick}-read-left*[**where** $A = \langle \{a\} \rangle$, *simplified*]) *simp-all*

lemma *read-Sync_{ptick}-write-left* :

$\langle c \ ' \ A \cap S = \{\} \implies d \ b \in S \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = c?a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q) \rangle$
by (*subst read-Sync_{ptick}-ndet-write-left*[**where** $B = \langle \{b\} \rangle$, *simplified*]) *simp-all*

lemma *write-Sync_{ptick}-read-right* :

$\langle c \ a \in S \implies d \ ' \ B \cap S = \{\} \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b = d?b \in B \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$
by (*subst ndet-write-Sync_{ptick}-read-right*[**where** $A = \langle \{a\} \rangle$, *simplified*]) *simp-all*

lemma *read-Sync_{ptick}-write-right* :

$\langle c \ ' \ A \subseteq S \implies d \ b \notin S \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = d!b \rightarrow (c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*subst read-Sync_{ptick}-ndet-write-right*[**where** $B = \langle \{b\} \rangle$, *simplified*]) *simp-all*

ndet-write **and** *ndet-write* **lemma** *ndet-write-Sync_{ptick}-ndet-write* :

$\langle c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b =$
 $(\text{ if } A = \{\} \text{ then } \text{ if } d \ ' \ B \cap S = \{\} \text{ then } d!!b \in B \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q \ b)$
 $\text{ else } (\Box x \in d \ ' \ (B - d - ' \ S) \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q \ (inv\text{-into } B \ d \ x)))$
 $\Box STOP$
 $\text{ else } \text{ if } B = \{\} \text{ then } \text{ if } c \ ' \ A \cap S = \{\} \text{ then } c!!a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} STOP)$
 $\text{ else } (\Box x \in c \ ' \ (A - c - ' \ S) \rightarrow (P \ (inv\text{-into } A \ c \ x) \llbracket S \rrbracket_{\checkmark}$
 $STOP)) \Box STOP$
 $\text{ else } \Box b \in d \ ' \ B. \Box a \in c \ ' \ A.$
 $(\text{ if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P \ (inv\text{-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} b \rightarrow Q \ (inv\text{-into}$
 $B \ d \ b))) \Box$
 $(\text{ if } b \in S \text{ then } STOP \text{ else } b \rightarrow (a \rightarrow P \ (inv\text{-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} Q \ (inv\text{-into}$
 $B \ d \ b))) \Box$
 $(\text{ if } a = b \wedge b \in S \text{ then } b \rightarrow (P \ (inv\text{-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} Q \ (inv\text{-into } B \ d$
 $b)) \text{ else } STOP) \rangle$

proof –

have $\langle d \ ' \ (B - d - ' \ S) = d \ ' \ B - S \rangle \langle c \ ' \ (A - c - ' \ S) = c \ ' \ A - S \rangle$ **by** *auto*
thus *?thesis*
by (*auto simp add: ndet-write-def Mndetprefix-Sync_{ptick}-Mndetprefix comp-def*
intro!: mono-GlobalNdet-eq split: if-split-asm)

qed

lemma *ndet-write-Sync_{ptick}-ndet-write-subset* :

$\langle c \ ' \ A \subseteq S \implies d \ ' \ B \subseteq S \implies$
 $c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b =$

$(\text{ if } \exists b. c \text{ ' } A = \{b\} \wedge d \text{ ' } B = \{b\}$
 $\text{ then } (THE\ b. d \text{ ' } B = \{b\}) \rightarrow (P\ (inv\text{-into}\ A\ c\ (THE\ a. c \text{ ' } A = \{a\}))\ \llbracket S \rrbracket_{\checkmark} Q$
 $(inv\text{-into}\ B\ d\ (THE\ b. d \text{ ' } B = \{b\})))$
 $\text{ else } (\Box x \in (c \text{ ' } A \cap d \text{ ' } B) \rightarrow (P\ (inv\text{-into}\ A\ c\ x)\ \llbracket S \rrbracket_{\checkmark} Q\ (inv\text{-into}\ B\ d\ x))) \sqcap$
 $STOP)$
by (*auto simp add: ndet-write-def Mndetprefix-Sync_{ptick}-Mndetprefix-subset*)

corollary *inj-on-ndet-write-Sync_{ptick}-ndet-write-subset* :

$\langle c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b =$
 $(\text{ if } \exists b. c \text{ ' } A = \{b\} \wedge d \text{ ' } B = \{b\}$
 $\text{ then } d\ (THE\ b. B = \{b\}) \rightarrow (P\ (THE\ a. A = \{a\})\ \llbracket S \rrbracket_{\checkmark} Q\ (THE\ b. B = \{b\}))$
 $\text{ else } (\Box x \in (c \text{ ' } A \cap d \text{ ' } B) \rightarrow (P\ (inv\text{-into}\ A\ c\ x)\ \llbracket S \rrbracket_{\checkmark} Q\ (inv\text{-into}\ B\ d\ x))) \sqcap$
 $STOP)\rangle$

if $\langle inj\text{-on}\ c\ A \rangle \langle inj\text{-on}\ d\ B \rangle \langle c \text{ ' } A \subseteq S \rangle \langle d \text{ ' } B \subseteq S \rangle$

proof –

from *that(1)* **have** $\langle c \text{ ' } A = \{a'\} \implies \exists! a. A = \{a\} \wedge a' = c\ a \rangle$ **for** a'

by (*fastforce elim!: inj-img-insertE*)

moreover from *that(2)* **have** $\langle d \text{ ' } B = \{b'\} \implies \exists! b. B = \{b\} \wedge b' = d\ b \rangle$ **for** b'

by (*fastforce elim!: inj-img-insertE*)

ultimately show *?thesis*

by (*auto simp add: ndet-write-Sync_{ptick}-ndet-write-subset[OF that(3, 4)] inv-into-f-eq*

intro: arg-cong2[where $f = \langle (\rightarrow) \rangle$] arg-cong2[where $f = \langle \lambda P\ Q. P\ \llbracket S \rrbracket_{\checkmark}$

Q\rangle)

qed

lemma *ndet-write-Sync_{ptick}-ndet-write-indep* :

$\langle c \text{ ' } A \cap S = \{\} \implies d \text{ ' } B \cap S = \{\} \implies$

$c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b =$

$(\text{ if } A = \{\} \text{ then } d!!b \in B \rightarrow (STOP\ \llbracket S \rrbracket_{\checkmark} Q\ b)$

$\text{ else if } B = \{\} \text{ then } c!!a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} STOP)$

$\text{ else } \Box b \in d \text{ ' } B. \Box a \in c \text{ ' } A.$

$((a \rightarrow (P\ (inv\text{-into}\ A\ c\ a)\ \llbracket S \rrbracket_{\checkmark} b \rightarrow Q\ (inv\text{-into}\ B\ d\ b)))) \sqcap$

$((b \rightarrow (a \rightarrow P\ (inv\text{-into}\ A\ c\ a)\ \llbracket S \rrbracket_{\checkmark} Q\ (inv\text{-into}\ B\ d\ b))))\rangle$

by (*auto simp add: ndet-write-Sync_{ptick}-ndet-write disjoint-iff intro!: mono-GlobalNdet-eq*)

lemma *ndet-write-Sync_{ptick}-ndet-write-left* :

$\langle c \text{ ' } A \cap S = \{\} \implies d \text{ ' } B \subseteq S \implies$

$c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b = c!!a \in A \rightarrow (P\ a\ \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b)\rangle$

by (*simp add: ndet-write-def Mndetprefix-Sync_{ptick}-Mndetprefix-left comp-def*)

lemma *ndet-write-Sync_{ptick}-ndet-write-right* :

$\langle c \text{ ' } A \subseteq S \implies d \text{ ' } B \cap S = \{\} \implies$

$c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q\ b = d!!b \in B \rightarrow (c!!a \in A \rightarrow P\ a\ \llbracket S \rrbracket_{\checkmark} Q\ b)\rangle$

by (*simp add: ndet-write-def Mndetprefix-Sync_{ptick}-Mndetprefix-right comp-def*)

ndet-write **and** *write* **lemma** *write-Sync_{ptick}-ndet-write* :

$$\begin{aligned}
& \langle c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b = \\
& \quad (\text{ if } B = \{\} \text{ then } c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} STOP \\
& \quad \text{ else } \sqcap b \in d \text{ ' } B. (\text{ if } b \in S \text{ then } STOP \text{ else } b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \\
& \quad b))) \sqcap \\
& \quad (\text{ if } c \ a \in S \text{ then } STOP \text{ else } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q \text{ (inv-into } B \ d \\
& \quad b))) \sqcap \\
& \quad (\text{ if } b = c \ a \wedge c \ a \in S \text{ then } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ (c \ a))) \\
& \text{ else } STOP)) \rangle \\
& \text{ by } (subst \text{ read-Sync}_{ptick}\text{-ndet-write}[\text{where } A = \langle \{a\} \rangle, \text{ simplified}], \\
& \quad \text{ auto simp add: write-def Mprefix-singl split: if-split-asm} \\
& \quad \text{ intro!: mono-GlobalNdet-eq arg-cong2}[\text{where } f = \langle (\square) \rangle] \text{ mono-Mprefix-eq} \\
& \quad (\text{simp add: insert-Diff-if write0-def})
\end{aligned}$$

lemma *ndet-write-Sync_{ptick}-write* :

$$\begin{aligned}
& \langle c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow Q = \\
& \quad (\text{ if } A = \{\} \text{ then } STOP \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow Q \\
& \quad \text{ else } \sqcap a \in c \text{ ' } A. (\text{ if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P \text{ (inv-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow \\
& \quad Q)) \sqcap \\
& \quad (\text{ if } d \ b \in S \text{ then } STOP \text{ else } d!!b \rightarrow (a \rightarrow P \text{ (inv-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} \\
& \quad Q)) \sqcap \\
& \quad (\text{ if } a = d \ b \wedge d \ b \in S \text{ then } d!!b \rightarrow (P \text{ (inv-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} Q) \text{ else } \\
& \quad STOP)) \rangle \\
& \text{ by } (subst \text{ ndet-write-Sync}_{ptick}\text{-read}[\text{where } B = \langle \{b\} \rangle, \text{ simplified}], \\
& \quad \text{ auto simp add: write-def Mprefix-singl split: if-split-asm} \\
& \quad \text{ intro!: mono-GlobalNdet-eq arg-cong2}[\text{where } f = \langle (\square) \rangle] \text{ mono-Mprefix-eq} \\
& \quad (\text{simp add: insert-Diff-if write0-def})
\end{aligned}$$

lemma *write-Sync_{ptick}-ndet-write-subset* :

$$\begin{aligned}
& \langle c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b = \\
& \quad (\text{ if } c \ a \notin d \text{ ' } B \text{ then } STOP \text{ else if } d \text{ ' } B = \{c \ a\} \text{ then } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ (c \ a))) \\
& \quad \text{ else } (c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ (c \ a)))) \sqcap STOP) \rangle \text{ if } \langle c \ a \in S \rangle \langle d \text{ ' } B \subseteq S \rangle \\
& \text{ proof } (subst \text{ read-Sync}_{ptick}\text{-ndet-write-subset}[\text{where } A = \langle \{a\} \rangle, \text{ simplified}]) \\
& \quad \text{ from } \langle c \ a \in S \rangle \text{ show } \langle c \ a \in S \rangle . \\
& \text{ next} \\
& \quad \text{ from } \langle d \text{ ' } B \subseteq S \rangle \text{ show } \langle d \text{ ' } B \subseteq S \rangle . \\
& \text{ next} \\
& \quad \text{ show } \langle (\text{ if } d \text{ ' } B \subseteq \{c \ a\} \text{ then } \sqcap b \in d \text{ ' } B \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ b)) \\
& \quad \text{ else } (\sqcap b \in (c \text{ ' } \{a\} \cap d \text{ ' } B) \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ b))) \sqcap STOP) = \\
& \quad (\text{ if } c \ a \notin d \text{ ' } B \text{ then } STOP \text{ else if } d \text{ ' } B = \{c \ a\} \text{ then } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \\
& \quad \text{ (inv-into } B \ d \ (c \ a))) \\
& \quad \text{ else } (c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ (c \ a)))) \sqcap STOP) \rangle \\
& \quad (\text{is } \langle ?lhs = (\text{ if } c \ a \notin d \text{ ' } B \text{ then } STOP \text{ else if } d \text{ ' } B = \{c \ a\} \text{ then } ?rhs \text{ else } ?rhs \\
& \quad \sqcap STOP) \rangle) \\
& \text{ proof } (split \text{ if-split, intro conjI impI}) \\
& \quad \text{ show } \langle c \ a \notin d \text{ ' } B \implies ?lhs = STOP \rangle
\end{aligned}$$

by (auto simp add: GlobalNdet-is-STOP-iff image-subset-iff image-iff)
 next
 show $\langle \neg c \ a \notin d \ ' \ B \implies ?lhs = (if \ d \ ' \ B = \{c \ a\} \text{ then } ?rhs \text{ else } ?rhs \sqcap STOP) \rangle$
 by (auto simp add: image-subset-iff Ndet-is-STOP-iff write-is-write0)
 qed
 qed

corollary (in $Sync_{ptick}\text{-locale}$) $ndet\text{-write}\text{-}Sync_{ptick}\text{-write-subset}$:
 $\langle (c!!a \in A \rightarrow P \ a) \llbracket S \rrbracket_{\checkmark} (d!!b \rightarrow Q) =$
 $(\text{ if } d \ b \notin c \ ' \ A \text{ then } STOP \text{ else if } c \ ' \ A = \{d \ b\} \text{ then } d!!b \rightarrow (P \ (inv\text{-into } A \ c \ (d \ b)) \llbracket S \rrbracket_{\checkmark} Q)$
 $\text{ else } (d!!b \rightarrow (P \ (inv\text{-into } A \ c \ (d \ b)) \llbracket S \rrbracket_{\checkmark} Q)) \sqcap STOP \rangle \text{ if } \langle c \ ' \ A \subseteq S \rangle \langle d \ b \in S \rangle$
 by (subst (1 2 3) $Sync_{ptick}\text{-locale-sym}.$ $Sync_{ptick}\text{-sym}$)
 (simp add: $Sync_{ptick}\text{-locale-sym}.$ $write\text{-}Sync_{ptick}\text{-ndet-write-subset}$ that)

lemma $write\text{-}Sync_{ptick}\text{-ndet-write-indep}$:
 $\langle c \ a \notin S \implies d \ ' \ B \cap S = \{\} \implies$
 $c!!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b =$
 $(\text{ if } B = \{\} \text{ then } c!!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} STOP)$
 $\text{ else } \sqcap b \in d \ ' \ B. (c!!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q \ (inv\text{-into } B \ d \ b))) \sqcap$
 $(b \rightarrow (c!!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \ (inv\text{-into } B \ d \ b)))) \rangle$
 by (subst $ndet\text{-write}\text{-}Sync_{ptick}\text{-ndet-write-indep}$ [**where** $A = \langle \{a\} \rangle$, *simplified*])
 (auto simp add: $write\text{-is-write0}$ intro: $mono\text{-GlobalNdet-eq}$)

lemma $ndet\text{-write}\text{-}Sync_{ptick}\text{-write-indep}$:
 $\langle c \ ' \ A \cap S = \{\} \implies d \ b \notin S \implies$
 $c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow Q =$
 $(\text{ if } A = \{\} \text{ then } d!!b \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q)$
 $\text{ else } \sqcap a \in c \ ' \ A. (a \rightarrow (P \ (inv\text{-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow Q)) \sqcap$
 $(d!!b \rightarrow (a \rightarrow P \ (inv\text{-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} Q))) \rangle$
 by (subst $ndet\text{-write}\text{-}Sync_{ptick}\text{-ndet-write-indep}$ [**where** $B = \langle \{b\} \rangle$, *simplified*])
 (auto simp add: $write\text{-is-write0}$ intro: $mono\text{-GlobalNdet-eq}$)

lemma $write\text{-}Sync_{ptick}\text{-ndet-write-left}$:
 $\langle c \ a \notin S \implies d \ ' \ B \subseteq S \implies c!!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b = c!!a \rightarrow (P \llbracket S \rrbracket_{\checkmark}$
 $d!!b \in B \rightarrow Q \ b) \rangle$
 by (subst $ndet\text{-write}\text{-}Sync_{ptick}\text{-ndet-write-left}$ [**where** $A = \langle \{a\} \rangle$, *simplified*]) *simp-all*

lemma $ndet\text{-write}\text{-}Sync_{ptick}\text{-write-left}$:
 $\langle c \ ' \ A \cap S = \{\} \implies d \ b \in S \implies c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow Q = c!!a \in A \rightarrow (P$
 $a \llbracket S \rrbracket_{\checkmark} d!!b \rightarrow Q) \rangle$
 by (subst $ndet\text{-write}\text{-}Sync_{ptick}\text{-ndet-write-left}$ [**where** $B = \langle \{b\} \rangle$, *simplified*]) *simp-all*

lemma $write\text{-}Sync_{ptick}\text{-ndet-write-right}$:
 $\langle c \ a \in S \implies d \ ' \ B \cap S = \{\} \implies c!!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b = d!!b \in B \rightarrow$

$(c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$
by (*subst ndet-write-Sync_{ptick}-ndet-write-right*[**where** $A = \langle \{a\} \rangle$, *simplified*])
simp-all

lemma *ndet-write-Sync_{ptick}-write-right* :
 $\langle c \ ' \ A \subseteq S \implies d \ b \notin S \implies c!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = d!b \rightarrow (c!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*subst ndet-write-Sync_{ptick}-ndet-write-right*[**where** $B = \langle \{b\} \rangle$, *simplified*])
simp-all

write and write **lemma** *write-Sync_{ptick}-write* :
 $\langle c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q =$
 (if $d \ b \in S$ then *STOP* else $d!b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)$) $\square$
 (if $c \ a \in S$ then *STOP* else $c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q)$) $\square$
 (if $c \ a = d \ b \wedge d \ b \in S$ then $c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)$ else *STOP*) $\rangle$
by (*subst read-Sync_{ptick}-read*[**where** $A = \langle \{a\} \rangle$ **and** $B = \langle \{b\} \rangle$, *simplified*])
 (*simp add: write-def insert-Diff-if Det-commute Int-insert-right*)

lemma *write-Inter_{ptick}-write* :
 $\langle c!a \rightarrow P \ |||_{\checkmark} d!b \rightarrow Q = (c!a \rightarrow (P \ |||_{\checkmark} d!b \rightarrow Q)) \square (d!b \rightarrow (c!a \rightarrow P \ |||_{\checkmark} Q)) \rangle$
by (*simp add: write-Sync_{ptick}-write Det-commute*)

lemma *write-Par_{ptick}-write* :
 $\langle c!a \rightarrow P \ ||_{\checkmark} d!b \rightarrow Q = (if \ c \ a = d \ b \text{ then } c!a \rightarrow (P \ ||_{\checkmark} Q) \text{ else } STOP) \rangle$
by (*simp add: write-Sync_{ptick}-write*)

lemma *write-Sync_{ptick}-write-subset* :
 $\langle c \ a \in S \implies d \ b \in S \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = (if \ c \ a = d \ b \text{ then } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q) \text{ else } STOP) \rangle$
by (*simp add: write-Sync_{ptick}-write*)

lemma *write-Sync_{ptick}-write-indep* :
 $\langle c \ a \notin S \implies d \ b \notin S \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = (c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q)) \square (d!b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)) \rangle$
by (*simp add: Det-commute write-Sync_{ptick}-write*)

lemma *write-Sync_{ptick}-write-left* :
 $\langle c \ a \notin S \implies d \ b \in S \implies c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q) \rangle$
by (*auto simp add: write-Sync_{ptick}-write*)

lemma *write-Sync_{ptick}-write-right* :
 $\langle c \ a \in S \implies d \ b \notin S \implies c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = d!b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*auto simp add: write-Sync_{ptick}-write*)

read and $(\rightarrow)$. **lemma** *write0-Sync_{ptick}-read* :
 $\langle a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b =$
 $(\text{if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b)) \square$
 $(\square b \in (d \text{ ' } B - S) \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \ (\text{inv-into } B \ d \ b))) \square$
 $(\text{if } a \in d \text{ ' } B \cap S \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \ (\text{inv-into } B \ d \ a)) \text{ else } STOP) \rangle$
by (*simp add: write-Sync_{ptick}-read*[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *read-Sync_{ptick}-write0* :
 $\langle c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 $(\text{if } b \in S \text{ then } STOP \text{ else } b \rightarrow (c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} Q)) \square$
 $(\square a \in (c \text{ ' } A - S) \rightarrow (P \ (\text{inv-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)) \square$
 $(\text{if } b \in c \text{ ' } A \cap S \text{ then } b \rightarrow (P \ (\text{inv-into } A \ c \ b) \llbracket S \rrbracket_{\checkmark} Q) \text{ else } STOP) \rangle$
by (*simp add: read-Sync_{ptick}-write*[**where** $d = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-read-subset* :
 $\langle a \in S \implies d \text{ ' } B \subseteq S \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b =$
 $(\text{if } a \in d \text{ ' } B \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \ (\text{inv-into } B \ d \ a)) \text{ else } STOP) \rangle$
by (*simp add: write-Sync_{ptick}-read-subset*[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *read-Sync_{ptick}-write0-subset* :
 $\langle c \text{ ' } A \subseteq S \implies b \in S \implies$
 $c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 $(\text{if } b \in c \text{ ' } A \text{ then } b \rightarrow (P \ (\text{inv-into } A \ c \ b) \llbracket S \rrbracket_{\checkmark} Q) \text{ else } STOP) \rangle$
by (*simp add: read-Sync_{ptick}-write-subset*[**where** $d = \langle \lambda x. x \rangle$, *unfolded write-is-write0*])

lemma *write0-Sync_{ptick}-read-subset-same-chan*:
 $\langle a \in S \implies B \subseteq S \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} id?b \in B \rightarrow Q \ b = (\text{if } a \in B \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \ a) \text{ else } STOP) \rangle$
by (*simp add: write-Sync_{ptick}-read-subset-same-chan*
[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *read-Sync_{ptick}-write0-subset-same-chan*:
 $\langle A \subseteq S \implies b \in S \implies$
 $id?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (\text{if } b \in A \text{ then } b \rightarrow (P \ b \llbracket S \rrbracket_{\checkmark} Q) \text{ else } STOP) \rangle$
by (*simp add: read-Sync_{ptick}-write-subset-same-chan*
[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-read-indep* :
 $\langle a \notin S \implies d \text{ ' } B \cap S = \{\} \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b =$
 $(a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b)) \square (d?b \in B \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \ b)) \rangle$
by (*simp add: write-Sync_{ptick}-read-indep*[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *read-Sync_{ptick}-write0-indep* :

$\langle c \text{ ' } A \cap S = \{\} \implies b \notin S \implies$
 $c?a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 $(b \rightarrow (c?a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} Q)) \square (c?a \in A \rightarrow (P \text{ } a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)) \rangle$
by (*simp add: read-Sync_{ptick}-write-indep[where d = id, unfolded write-is-write0, simplified]*)

lemma *write0-Sync_{ptick}-read-left* :
 $\langle a \notin S \implies d \text{ ' } B \subseteq S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \text{ } b = a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d?b \in B$
 $\rightarrow Q \text{ } b) \rangle$
by (*simp add: write-Sync_{ptick}-read-left[where c = id, unfolded write-is-write0, simplified]*)

lemma *read-Sync_{ptick}-write0-left* :
 $\langle c \text{ ' } A \cap S = \{\} \implies b \in S \implies c?a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = c?a \in A \rightarrow (P \text{ } a$
 $\llbracket S \rrbracket_{\checkmark} b \rightarrow Q) \rangle$
by (*simp add: read-Sync_{ptick}-write-left[where d = id, unfolded write-is-write0, simplified]*)

lemma *write0-Sync_{ptick}-read-right* :
 $\langle a \in S \implies d \text{ ' } B \cap S = \{\} \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \text{ } b = d?b \in B \rightarrow (a \rightarrow$
 $P \llbracket S \rrbracket_{\checkmark} Q \text{ } b) \rangle$
by (*simp add: write-Sync_{ptick}-read-right[where c = id, unfolded write-is-write0, simplified]*)

lemma *read-Sync_{ptick}-write0-right* :
 $\langle c \text{ ' } A \subseteq S \implies b \notin S \implies c?a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = b \rightarrow (c?a \in A \rightarrow P \text{ } a$
 $\llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: read-Sync_{ptick}-write-right[where d = id, unfolded write-is-write0, simplified]*)

ndet-write **and** ($\rightarrow$) **lemma** *write0-Sync_{ptick}-ndet-write* :
 $\langle a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \text{ } b =$
 $(\text{ if } B = \{\} \text{ then } a \rightarrow P \llbracket S \rrbracket_{\checkmark} STOP$
 $\text{ else } \sqcap b \in d \text{ ' } B. (\text{ if } b \in S \text{ then } STOP \text{ else } b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q (\text{inv-into } B \text{ } d$
 $b))) \square$
 $(\text{ if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q (\text{inv-into } B \text{ } d \text{ } b))) \square$
 $(\text{ if } b = a \wedge a \in S \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q (\text{inv-into } B \text{ } d \text{ } a)) \text{ else}$
 $STOP) \rangle$
by (*simp add: write-Sync_{ptick}-ndet-write[where c = $\langle \lambda x. x \rangle$, unfolded write-is-write0, simplified]*)

lemma *ndet-write-Sync_{ptick}-write0* :
 $\langle c!!a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 $(\text{ if } A = \{\} \text{ then } STOP \llbracket S \rrbracket_{\checkmark} b \rightarrow Q$
 $\text{ else } \sqcap a \in c \text{ ' } A. (\text{ if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P (\text{inv-into } A \text{ } c \text{ } a) \llbracket S \rrbracket_{\checkmark} b \rightarrow$
 $Q)) \square$
 $(\text{ if } b \in S \text{ then } STOP \text{ else } b \rightarrow (a \rightarrow P (\text{inv-into } A \text{ } c \text{ } a) \llbracket S \rrbracket_{\checkmark} Q)) \square$
 $(\text{ if } a = b \wedge b \in S \text{ then } b \rightarrow (P (\text{inv-into } A \text{ } c \text{ } a) \llbracket S \rrbracket_{\checkmark} Q) \text{ else}$
 $STOP) \rangle$

by (*simp add: ndet-write-Sync_{ptick}-write*[**where** $d = \langle \lambda x. x \rangle$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-ndet-write-subset* :

$\langle a \in S \implies d \text{ ' } B \subseteq S \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b =$
 $(\text{ if } a \notin d \text{ ' } B \text{ then } STOP \text{ else if } d \text{ ' } B = \{a\} \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B$
 $d \ a))$
 $\text{ else } (a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ a))) \sqcap STOP \rangle$
by (*simp add: write-Sync_{ptick}-ndet-write-subset*[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *ndet-write-Sync_{ptick}-write0-subset* :

$\langle c \text{ ' } A \subseteq S \implies b \in S \implies$
 $c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 $(\text{ if } b \notin c \text{ ' } A \text{ then } STOP \text{ else if } c \text{ ' } A = \{b\} \text{ then } b \rightarrow (P \text{ (inv-into } A \ c \ b) \llbracket S \rrbracket_{\checkmark}$
 $Q)$
 $\text{ else } (b \rightarrow (P \text{ (inv-into } A \ c \ b) \llbracket S \rrbracket_{\checkmark} Q)) \sqcap STOP \rangle$
by (*simp add: ndet-write-Sync_{ptick}-write-subset*[**where** $d = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-ndet-write-indep* :

$\langle a \notin S \implies d \text{ ' } B \cap S = \{\} \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b =$
 $(\text{ if } B = \{\} \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} STOP)$
 $\text{ else } \sqcap b \in d \text{ ' } B. (a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q \text{ (inv-into } B \ d \ b))) \sqcap$
 $(b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \text{ (inv-into } B \ d \ b)))) \rangle$
by (*simp add: write-Sync_{ptick}-ndet-write-indep*[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *ndet-write-Sync_{ptick}-write0-indep* :

$\langle c \text{ ' } A \cap S = \{\} \implies b \notin S \implies$
 $c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 $(\text{ if } A = \{\} \text{ then } b \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q)$
 $\text{ else } \sqcap a \in c \text{ ' } A. (a \rightarrow (P \text{ (inv-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)) \sqcap$
 $(b \rightarrow (a \rightarrow P \text{ (inv-into } A \ c \ a) \llbracket S \rrbracket_{\checkmark} Q)))) \rangle$
by (*simp add: ndet-write-Sync_{ptick}-write-indep*[**where** $d = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-ndet-write-left* :

$\langle a \notin S \implies d \text{ ' } B \subseteq S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b = a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!!b \in B$
 $\rightarrow Q \ b) \rangle$
by (*simp add: write-Sync_{ptick}-ndet-write-left*[**where** $c = id$, *unfolded write-is-write0, simplified*])

lemma *ndet-write-Sync_{ptick}-write0-left* :

$\langle c \text{ ' } A \cap S = \{\} \implies b \in S \implies c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = c!!a \in A \rightarrow (P \ a$
 $\llbracket S \rrbracket_{\checkmark} b \rightarrow Q) \rangle$
by (*simp add: ndet-write-Sync_{ptick}-write-left*[**where** $d = id$, *unfolded write-is-write0,*

simplified)

lemma *write-Sync_{ptick}-ndet-write0-right* :

$\langle a \in S \implies d \text{ ' } B \cap S = \{\} \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \text{ } b = d!!b \in B \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q \text{ } b) \rangle$

by (*simp add: write-Sync_{ptick}-ndet-write-right* [**where** $c = id$, *unfolded write-is-write0*, *simplified*])

lemma *ndet-write-Sync_{ptick}-write0-right* :

$\langle c \text{ ' } A \subseteq S \implies b \notin S \implies c!!a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = b \rightarrow (c!!a \in A \rightarrow P \text{ } a \llbracket S \rrbracket_{\checkmark} Q) \rangle$

by (*simp add: ndet-write-Sync_{ptick}-write-right* [**where** $d = id$, *unfolded write-is-write0*, *simplified*])

($\rightarrow$) **and** ($\rightarrow$) **lemma** *write0-Sync_{ptick}-write0* :

$\langle a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
 (if $b \in S$ then *STOP* else $b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)$) $\square$
 (if $a \in S$ then *STOP* else $a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)$) $\square$
 (if $a = b \wedge b \in S$ then $a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)$ else *STOP*) $\rangle$

by (*simp add: write-Sync_{ptick}-write* [**where** $c = id$ **and** $d = id$, *unfolded write-is-write0*, *simplified*])

lemma *write0-Sync_{ptick}-write0-bis* :

$\langle (a \rightarrow P) \llbracket S \rrbracket_{\checkmark} (b \rightarrow Q) =$
 (if $a \in S$
 then if $b \in S$
 then if $a = b$
 then $a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)$
 else *STOP*
 else $(b \rightarrow ((a \rightarrow P) \llbracket S \rrbracket_{\checkmark} Q))$
 else if $b \in S$
 then $a \rightarrow (P \llbracket S \rrbracket_{\checkmark} (b \rightarrow Q))$
 else $(a \rightarrow (P \llbracket S \rrbracket_{\checkmark} (b \rightarrow Q))) \square (b \rightarrow ((a \rightarrow P) \llbracket S \rrbracket_{\checkmark} Q)) \rangle$

by (*cases* $\langle a \in S \rangle$; *cases* $\langle b \in S \rangle$) (*auto simp add: write0-Sync_{ptick}-write0 Det-commute*)

lemma *write0-Inter_{ptick}-write0* :

$\langle a \rightarrow P \parallel_{\checkmark} b \rightarrow Q = (a \rightarrow (P \parallel_{\checkmark} b \rightarrow Q)) \square (b \rightarrow (a \rightarrow P \parallel_{\checkmark} Q)) \rangle$

by (*simp add: write0-Sync_{ptick}-write0 Det-commute*)

lemma *write0-Par_{ptick}-write0* :

$\langle a \rightarrow P \parallel_{\checkmark} b \rightarrow Q = (\text{if } a = b \text{ then } a \rightarrow (P \parallel_{\checkmark} Q) \text{ else } \text{STOP}) \rangle$

by (*simp add: write0-Sync_{ptick}-write0*)

lemma *write0-Sync_{ptick}-write0-subset* :

$\langle a \in S \implies b \in S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (\text{if } a = b \text{ then } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q) \text{ else } \text{STOP}) \rangle$

by (*simp add: write-Sync_{ptick}-write-subset*[**where** $c = id$ **and** $d = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-write0-indep* :

$\langle a \notin S \implies b \notin S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)) \sqcap (b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)) \rangle$

by (*simp add: write-Sync_{ptick}-write-indep*[**where** $c = id$ **and** $d = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-write0-left* :

$\langle a \notin S \implies b \in S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q) \rangle$

by (*simp add: write-Sync_{ptick}-write-left*[**where** $c = id$ **and** $d = id$, *unfolded write-is-write0, simplified*])

lemma *write0-Sync_{ptick}-write0-right* :

$\langle a \in S \implies b \notin S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q) \rangle$

by (*simp add: write-Sync_{ptick}-write-right*[**where** $c = id$ **and** $d = id$, *unfolded write-is-write0, simplified*])

write and ($\rightarrow$) **lemma** *write0-Sync_{ptick}-write* :

$\langle a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q =$
(if $d b \in S$ *then* $STOP$ *else* $d!b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)) \sqcap$
(if $a \in S$ *then* $STOP$ *else* $a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q)) \sqcap$
(if $a = d b \wedge d b \in S$ *then* $a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)$ *else* $STOP) \rangle$

by (*simp add: write0-Sync_{ptick}-write0 write-is-write0*)

lemma *write-Sync_{ptick}-write0* :

$\langle c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q =$
(if $b \in S$ *then* $STOP$ *else* $b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)) \sqcap$
(if $c a \in S$ *then* $STOP$ *else* $c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)) \sqcap$
(if $c a = b \wedge b \in S$ *then* $c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)$ *else* $STOP) \rangle$

by (*simp add: write0-Sync_{ptick}-write0 write-is-write0*)

lemma *write0-Sync_{ptick}-write-subset* :

$\langle a \in S \implies d b \in S \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = (if\ a = d\ b\ then\ a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)\ else\ STOP) \rangle$

by (*simp add: write0-Sync_{ptick}-write*)

lemma *write-Sync_{ptick}-write0-subset* :

$\langle c\ a \in S \implies b \in S \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (if\ c\ a = b\ then\ c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} Q)\ else\ STOP) \rangle$

by (*simp add: write-Sync_{ptick}-write0*)

lemma *write0-Sync_{ptick}-write-indep* :

$\langle a \notin S \implies d b \notin S \implies$
 $a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = (a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q)) \sqcap (d!b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)) \rangle$

by (*simp add: Det-commute write0-Sync_{ptick}-write*)

lemma *write-Sync_{ptick}-write0-indep* :

$\langle c \ a \notin S \implies b \notin S \implies$
 $c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q)) \sqcap (b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q)) \rangle$
by (*simp add: Det-commute write-Sync_{ptick}-write0*)

lemma *write0-Sync_{ptick}-write-left* :

$\langle a \notin S \implies d \ b \in S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = a \rightarrow (P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q) \rangle$
by (*simp add: write0-Sync_{ptick}-write0-left write-is-write0*)

lemma *write-Sync_{ptick}-write0-left* :

$\langle c \ a \notin S \implies b \in S \implies c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q) \rangle$
by (*simp add: write0-Sync_{ptick}-write0-left write-is-write0*)

lemma *write0-Sync_{ptick}-write-right* :

$\langle a \in S \implies d \ b \notin S \implies a \rightarrow P \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = d!b \rightarrow (a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: write0-Sync_{ptick}-write0-right write-is-write0*)

lemma *write-Sync_{ptick}-write0-right* :

$\langle c \ a \in S \implies b \notin S \implies c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = b \rightarrow (c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: write0-Sync_{ptick}-write0-right write-is-write0*)

Synchronization with *SKIP* and *STOP*

SKIP Without injectivity, the result is a trivial corollary of $read \ c \ A \ P \equiv Mprefix \ (c \ ' \ A) \ (P \circ inv\text{-}into \ A \ c)$ and $Mprefix \ A \ P \llbracket S \rrbracket_{\checkmark} SKIP \ r = \sqcap_{a \in (A \setminus S)} (P \ a \llbracket S \rrbracket_{\checkmark} SKIP \ r)$.

lemma *read-Sync_{ptick}-SKIP* :

$\langle c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} SKIP \ r = c?a \in (A - c - ' \ S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} SKIP \ r) \rangle$ **if**
 $\langle inj\text{-}on \ c \ A \rangle$

proof –

have $\langle c \ ' \ (A - c - ' \ S) = c \ ' \ A - S \rangle$ **by** *blast*

show $\langle c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} SKIP \ r = c?a \in (A - c - ' \ S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} SKIP \ r) \rangle$

by (*auto simp add: read-def Mprefix-Sync_{ptick}-SKIP* $\langle ?this \rangle$ *inj-on-diff* $\langle inj\text{-}on \ c \ A \rangle$

intro: mono-Mprefix-eq)

qed

lemma *SKIP-Sync_{ptick}-read* :

$\langle SKIP \ r \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b = d?b \in (B - d - ' \ S) \rightarrow (SKIP \ r \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$ **if**
 $\langle inj\text{-}on \ d \ B \rangle$

proof –

have $\langle d \ ' \ (B - d - ' \ S) = d \ ' \ B - S \rangle$ **by** *blast*

show $\langle SKIP \ r \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b = d?b \in (B - d - ' \ S) \rightarrow (SKIP \ r \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$

by (*auto simp add: read-def SKIP-Sync_{ptick}-Mprefix* $\langle ?this \rangle$ *inj-on-diff* $\langle inj\text{-}on \ d \ B \rangle$

intro: mono-Mprefix-eq)

qed

corollary *write-Sync_{ptick}-SKIP* :

$\langle c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} \text{ SKIP } s = (\text{if } c \ a \in S \text{ then } STOP \text{ else } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} \text{ SKIP } s)) \rangle$
and *SKIP-Sync_{ptick}-write* :
 $\langle \text{SKIP } r \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = (\text{if } d \ b \in S \text{ then } STOP \text{ else } d!b \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q)) \rangle$
by (*simp-all add: write-def Mprefix-Sync_{ptick}-SKIP SKIP-Sync_{ptick}-Mprefix Diff-triv*)

corollary *write0-Sync_{ptick}-SKIP* :

$\langle a \rightarrow P \llbracket S \rrbracket_{\checkmark} \text{ SKIP } s = (\text{if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} \text{ SKIP } s)) \rangle$
and *SKIP-Sync_{ptick}-write0* :
 $\langle \text{SKIP } r \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (\text{if } b \in S \text{ then } STOP \text{ else } b \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q)) \rangle$
by (*simp-all add: write0-def Mprefix-Sync_{ptick}-SKIP SKIP-Sync_{ptick}-Mprefix Diff-triv*)

lemma *ndet-write-Sync_{ptick}-SKIP* :

$\langle c!!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r =$
 $(\text{if } c \ ' \ A \cap S = \{\} \text{ then } c!!a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r)$
 $\text{else } (c!!a \in (A - c \ - \ ' \ S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{ SKIP } r)) \sqcap STOP) \rangle$
 $(\text{is } \langle ?lhs = (\text{if } - \text{ then } ?rhs1 \text{ else } ?rhs2 \sqcap STOP) \rangle \text{ if } \langle \text{inj-on } c \ A \rangle$

proof (*split if-split, intro conjI impI*)

assume $\langle c \ ' \ A \cap S = \{\} \rangle$

hence $\langle A - c \ - \ ' \ S = A \rangle$ **by** *blast*

from $\langle c \ ' \ A \cap S = \{\} \rangle$ **show** $\langle ?lhs = ?rhs1 \rangle$

by (*auto simp add: $\langle ?this \rangle$ ndet-write-is-GlobalNdet-write0 disjoint-iff*
Sync_{ptick}-distrib-GlobalNdet-right write0-Sync_{ptick}-SKIP
intro!: mono-GlobalNdet-eq split: if-split-asm)

next

show $\langle ?lhs = ?rhs2 \sqcap STOP \rangle$ **if** $\langle c \ ' \ A \cap S \neq \{\} \rangle$

proof (*cases $\langle c \ ' \ A - S = \{\} \rangle$*)

assume $\langle c \ ' \ A - S = \{\} \rangle$

hence $\langle A - c \ - \ ' \ S = \{\} \rangle$ **by** *blast*

from $\langle c \ ' \ A - S = \{\} \rangle$ **show** $\langle ?lhs = ?rhs2 \sqcap STOP \rangle$

by (*auto simp add: ndet-write-is-GlobalNdet-write0 GlobalNdet-is-STOP-iff*
 $\langle ?this \rangle$ *Sync_{ptick}-distrib-GlobalNdet-right write0-Sync_{ptick}-SKIP*)

next

have $\langle c \ ' \ (A - c \ - \ ' \ S) = c \ ' \ A - S \rangle$ **by** *blast*

show $\langle ?lhs = ?rhs2 \sqcap STOP \rangle$ **if** $\langle c \ ' \ A - S \neq \{\} \rangle$

by (*subst Ndet-commute, unfold ndet-write-is-GlobalNdet-write0 Sync_{ptick}-distrib-GlobalNdet-right*)
 $(\text{auto simp add: GlobalNdet-is-STOP-iff write0-Sync_{ptick}-SKIP}$
 $\langle ?this \rangle \langle \text{inj-on } c \ A \rangle \text{ inj-on-diff}$
simp flip: GlobalNdet-factorization-union
 $[OF \langle c \ ' \ A \cap S \neq \{\} \rangle \langle c \ ' \ A - S \neq \{\} \rangle, \text{ unfolded Int-Diff-Un}]$
intro!: arg-cong2[where $f = \langle (\sqcap) \rangle$] mono-GlobalNdet-eq)

qed

qed

corollary (in $\text{Sync}_{ptick}\text{-locale}$) $\text{SKIP-Sync}_{ptick}\text{-ndet-write}$:
 $\langle \text{inj-on } d \ B \implies \text{SKIP } r \llbracket S \rrbracket_{\checkmark} d!!b \in B \rightarrow Q \ b =$
 $(\text{ if } d \text{ ' } B \cap S = \{\} \text{ then } d!!b \in B \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q \ b)$
 $\text{ else } (d!!b \in (B - d - \text{' } S) \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q \ b)) \sqcap \text{STOP} \rangle$
by (subst (1 2 3) $\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}\text{-sym}$)
 $(\text{simp add: Sync}_{ptick}\text{-locale-sym.ndet-write-Sync}_{ptick}\text{-SKIP})$

corollary (in $\text{Sync}_{ptick}\text{-locale}$) $\text{Mndetprefix-Sync}_{ptick}\text{-SKIP}$:
 $\langle \sqcap a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \text{SKIP } r =$
 $(\text{ if } A \cap S = \{\} \text{ then } \sqcap a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{SKIP } r)$
 $\text{ else } (\sqcap a \in (A - S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{SKIP } r)) \sqcap \text{STOP} \rangle$
using $\text{ndet-write-Sync}_{ptick}\text{-SKIP}[\text{of id } A \ P \ S \ r]$
by (simp add: $\text{ndet-write-id-is-Mndetprefix}$)

corollary (in $\text{Sync}_{ptick}\text{-locale}$) $\text{Sync}_{ptick}\text{-SKIP-Mndetprefix}$:
 $\langle \text{SKIP } r \llbracket S \rrbracket_{\checkmark} \sqcap b \in B \rightarrow Q \ b =$
 $(\text{ if } B \cap S = \{\} \text{ then } \sqcap b \in B \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q \ b)$
 $\text{ else } (\sqcap b \in (B - S) \rightarrow (\text{SKIP } r \llbracket S \rrbracket_{\checkmark} Q \ b)) \sqcap \text{STOP} \rangle$
by (subst (1 2 3) $\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}\text{-sym}$)
 $(\text{simp add: Sync}_{ptick}\text{-locale-sym.Mndetprefix-Sync}_{ptick}\text{-SKIP})$

STOP Without injectivity, the result is a trivial corollary of $\text{read } c \ A \ P \equiv \text{Mprefix } (c \text{ ' } A) (P \circ \text{inv-into } A \ c)$ and $\text{Mprefix } A \ P \llbracket S \rrbracket_{\checkmark} \text{SKIP } r = \sqcap a \in (A \setminus S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{SKIP } r)$.

lemma $\text{read-Sync}_{ptick}\text{-STOP}$:
 $\langle c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \text{STOP} = c?a \in (A - c - \text{' } S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{STOP}) \rangle$ **if** $\langle \text{inj-on } c \ A \rangle$

proof –

have $\langle c \text{ ' } (A - c - \text{' } S) = c \text{ ' } A - S \rangle$ **by** blast
show $\langle c?a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} \text{STOP} = c?a \in (A - c - \text{' } S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} \text{STOP}) \rangle$
by (auto simp add: $\langle ?this \rangle \text{read-def Mprefix-Sync}_{ptick}\text{-STOP inj-on-diff } \langle \text{inj-on } c \ A \rangle$

intro: mono-Mprefix-eq)

qed

lemma $\text{STOP-Sync}_{ptick}\text{-read}$:
 $\langle \text{STOP } \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b = d?b \in (B - d - \text{' } S) \rightarrow (\text{STOP } \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$ **if** $\langle \text{inj-on } d \ B \rangle$

proof –

have $\langle d \text{ ' } (B - d - \text{' } S) = d \text{ ' } B - S \rangle$ **by** blast
show $\langle \text{STOP } \llbracket S \rrbracket_{\checkmark} d?b \in B \rightarrow Q \ b = d?b \in (B - d - \text{' } S) \rightarrow (\text{STOP } \llbracket S \rrbracket_{\checkmark} Q \ b) \rangle$
by (auto simp add: $\langle ?this \rangle \text{read-def STOP-Sync}_{ptick}\text{-Mprefix inj-on-diff } \langle \text{inj-on } d \ B \rangle$

intro: mono-Mprefix-eq)

qed

corollary *write-Sync_{ptick}-STOP* :
 $\langle c!a \rightarrow P \llbracket S \rrbracket_{\checkmark} STOP = (\text{if } c \ a \in S \text{ then } STOP \text{ else } c!a \rightarrow (P \llbracket S \rrbracket_{\checkmark} STOP)) \rangle$
and *STOP-Sync_{ptick}-write* :
 $\langle STOP \llbracket S \rrbracket_{\checkmark} d!b \rightarrow Q = (\text{if } d \ b \in S \text{ then } STOP \text{ else } d!b \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q)) \rangle$
by (*simp-all add: write-def Mprefix-Sync_{ptick}-STOP STOP-Sync_{ptick}-Mprefix Diff-triv*)

corollary *write0-Sync_{ptick}-STOP* :
 $\langle a \rightarrow P \llbracket S \rrbracket_{\checkmark} STOP = (\text{if } a \in S \text{ then } STOP \text{ else } a \rightarrow (P \llbracket S \rrbracket_{\checkmark} STOP)) \rangle$
and *STOP-Sync_{ptick}-write0* :
 $\langle STOP \llbracket S \rrbracket_{\checkmark} b \rightarrow Q = (\text{if } b \in S \text{ then } STOP \text{ else } b \rightarrow (STOP \llbracket S \rrbracket_{\checkmark} Q)) \rangle$
by (*simp-all add: write0-def Mprefix-Sync_{ptick}-STOP STOP-Sync_{ptick}-Mprefix Diff-triv*)

lemma *ndet-write-Sync_{ptick}-STOP* :
 $\langle c!a \in A \rightarrow P \ a \llbracket S \rrbracket_{\checkmark} STOP =$
 $(\text{if } c \ ' \ A \cap S = \{\} \text{ then } c!a \in A \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} STOP)$
 $\text{else } (c!a \in (A - c \ - \ ' \ S) \rightarrow (P \ a \llbracket S \rrbracket_{\checkmark} STOP)) \sqcap STOP) \rangle$
 $(\text{is } \langle ?lhs = (\text{if } - \text{ then } ?rhs1 \text{ else } ?rhs2 \sqcap STOP) \rangle \text{ if } \langle \text{inj-on } c \ A \rangle$
proof (*split if-split, intro conjI impI*)
assume $\langle c \ ' \ A \cap S = \{\} \rangle$
hence $\langle A - c \ - \ ' \ S = A \rangle$ **by** *blast*
from $\langle c \ ' \ A \cap S = \{\} \rangle$ **show** $\langle ?lhs = ?rhs1 \rangle$
by (*auto simp add: \langle ?this \rangle ndet-write-is-GlobalNdet-write0 disjoint-iff*
Sync_{ptick}-distrib-GlobalNdet-right write0-Sync_{ptick}-STOP
intro!: mono-GlobalNdet-eq split: if-split-asm)

next
show $\langle ?lhs = ?rhs2 \sqcap STOP \rangle$ **if** $\langle c \ ' \ A \cap S \neq \{\} \rangle$
proof (*cases* $\langle c \ ' \ A - S = \{\} \rangle$)
assume $\langle c \ ' \ A - S = \{\} \rangle$
hence $\langle A - c \ - \ ' \ S = \{\} \rangle$ **by** *blast*
from $\langle c \ ' \ A - S = \{\} \rangle$ **show** $\langle ?lhs = ?rhs2 \sqcap STOP \rangle$
by (*auto simp add: ndet-write-is-GlobalNdet-write0 GlobalNdet-is-STOP-iff*
 $\langle ?this \rangle \text{ Sync}_{ptick}\text{-distrib-GlobalNdet-right write0-Sync}_{ptick}\text{-STOP}$)

next
have $\langle c \ ' \ (A - c \ - \ ' \ S) = c \ ' \ A - S \rangle$ **by** *blast*
show $\langle ?lhs = ?rhs2 \sqcap STOP \rangle$ **if** $\langle c \ ' \ A - S \neq \{\} \rangle$
by (*subst Ndet-commute, unfold ndet-write-is-GlobalNdet-write0 Sync_{ptick}-distrib-GlobalNdet-right*)
 $(\text{auto simp add: GlobalNdet-is-STOP-iff write0-Sync}_{ptick}\text{-STOP}$
 $\langle ?this \rangle \langle \text{inj-on } c \ A \rangle \text{ inj-on-diff}$
simp flip: GlobalNdet-factorization-union
 $[OF \langle c \ ' \ A \cap S \neq \{\} \rangle \langle c \ ' \ A - S \neq \{\} \rangle, \text{unfolded Int-Diff-Un}]$
intro!: arg-cong2[where f = \langle (\sqcap) \rangle] mono-GlobalNdet-eq)

qed
qed

corollary (*in* $\text{Sync}_{ptick}\text{-locale}$) $\text{STOP}\text{-Sync}_{ptick}\text{-ndet-write}$:

$\langle \text{inj-on } d \ B \implies \text{STOP } \llbracket S \rrbracket_{\checkmark} \ d!!b \in B \rightarrow Q \ b =$
 $(\text{ if } d \text{ ' } B \cap S = \{\} \text{ then } d!!b \in B \rightarrow (\text{STOP } \llbracket S \rrbracket_{\checkmark} \ Q \ b)$
 $\text{ else } (d!!b \in (B - d - \text{' } S) \rightarrow (\text{STOP } \llbracket S \rrbracket_{\checkmark} \ Q \ b)) \sqcap \text{STOP} \rangle$
by (*subst* (1 2 3) $\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick}\text{-sym}$)
 $(\text{simp add: } \text{Sync}_{ptick}\text{-locale-sym.ndet-write-Sync}_{ptick}\text{-STOP})$

end

Chapter 9

Operational Semantics Laws

9.1 Behaviour of *initials*

9.1.1 *TickSwap*

lemma *initials-TickSwap* :

$\langle (TickSwap\ P)^0 = (\text{ if } P = \perp \text{ then } UNIV$
 $\quad \text{ else } \{ev\ a \mid a. ev\ a \in P^0\} \cup \{\checkmark((s, r)) \mid r\ s. \checkmark((r, s)) \in P^0\} \rangle$
by (*auto simp add: TickSwap-is-Renaming initials-Renaming image-iff*
 $\text{ map-event}_{ptick}\text{-eq-tick-iff map-event}_{ptick}\text{-eq-ev-iff}$
 $\text{ tick-eq-map-event}_{ptick}\text{-iff ev-eq-map-event}_{ptick}\text{-iff}$
 $(metis\ tick\text{-swap.elims})$)

9.1.2 Sequential Composition

lemma *initials-Seq_{ptick}* :

$\langle (P ; \checkmark\ Q)^0 = (\text{ if } P = \perp \text{ then } UNIV$
 $\quad \text{ else } \{ev\ a \mid a. ev\ a \in P^0\} \cup (\bigcup_{r \in \{r. \checkmark(r) \in P^0\}} (Q\ r)^0) \rangle$
 $(\text{ is } \langle - = (if - then - else ?rhs) \rangle)$

proof (*split if-split, intro conjI impI*)

show $\langle P = \perp \implies (P ; \checkmark\ Q)^0 = UNIV \rangle$ **by** *simp*

next

show $\langle (P ; \checkmark\ Q)^0 = \{ev\ a \mid a. ev\ a \in P^0\} \cup (\bigcup_{r \in \{r. \checkmark(r) \in P^0\}} (Q\ r)^0) \rangle$ **if** $\langle P \neq \perp \rangle$

proof (*intro subset-antisym subsetI*)

fix e **assume** $\langle e \in (P ; \checkmark\ Q)^0 \rangle$

from *event_{ptick}.exhaust* **consider** a **where** $\langle e = ev\ a \rangle \mid r$ **where** $\langle e = \checkmark(r) \rangle$

by *blast*

thus $\langle e \in ?rhs \rangle$

proof *cases*

from $\langle e \in (P ; \checkmark\ Q)^0 \rangle \langle P \neq \perp \rangle$ **show** $\langle e = ev\ a \implies e \in ?rhs \rangle$ **for** a

by (*auto simp add: image-iff initials-def Seq_{ptick}-projs Cons-eq-append-conv*
 $BOT\text{-iff-}Nil\text{-}D$ *intro: D-T dest: initials-memD*)

(*use initials-memI in* $\langle blast\ dest: initials\text{-}memD \rangle$)

next

from $\langle e \in (P ; \checkmark\ Q)^0 \rangle \langle P \neq \perp \rangle$ **show** $\langle e = \checkmark(r) \implies e \in ?rhs \rangle$ **for** r

```

      by (auto simp add: image-iff initials-def Seqptick-projs BOT-iff-Nil-D
Cons-eq-append-conv)
    qed
  next
    show  $\langle e \in ?rhs \implies e \in (P ; \checkmark Q)^0 \rangle$  for  $e$ 
      by (simp add: initials-def Seqptick-projs, elim disjE exE conjE)
      (fastforce, metis list.simps(8) self-append-conv2 tickFree-Nil)
    qed
  qed

```

9.1.3 Synchronization Product

```

lemma (in Syncptick-locale) initials-Syncptick :
   $\langle (P \llbracket S \rrbracket_{\checkmark} Q)^0 =$ 
    (if  $P = \perp \vee Q = \perp$  then UNIV else
       $\{ev\ a \mid a \in S \wedge ev\ a \in P^0 \wedge ev\ a \in Q^0 \vee a \notin S \wedge (ev\ a \in P^0 \vee ev\ a \in Q^0)\}$ 
     $\cup$ 
     $\{\checkmark(r-s) \mid r-s\ r\ s.\ tick-join\ r\ s = Some\ r-s \wedge \checkmark(r) \in P^0 \wedge \checkmark(s) \in Q^0\} \rangle$ 
  (is  $\langle (P \llbracket S \rrbracket_{\checkmark} Q)^0 = (if\ P = \perp \vee Q = \perp\ then\ UNIV\ else\ ?rhs-ev \cup ?rhs-tick) \rangle$ )
proof (split if-split, intro conjI impI)
  show  $\langle P = \perp \vee Q = \perp \implies (P \llbracket S \rrbracket_{\checkmark} Q)^0 = UNIV \rangle$ 
    by (metis Syncptick-is-BOT-iff initials-BOT)
  next
    show  $\langle (P \llbracket S \rrbracket_{\checkmark} Q)^0 = ?rhs-ev \cup ?rhs-tick \rangle$  if non-BOT :  $\langle \neg (P = \perp \vee Q = \perp) \rangle$ 
  proof (intro subset-antisym subsetI)
    show  $\langle e \in ?rhs-ev \cup ?rhs-tick \implies e \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$  for  $e$ 
  proof (elim UnE)
    assume  $\langle e \in ?rhs-ev \rangle$ 
    then consider  $a$  where  $\langle e = ev\ a \rangle \langle a \in S \rangle \langle ev\ a \in P^0 \rangle \langle ev\ a \in Q^0 \rangle$ 
      |  $a$  where  $\langle e = ev\ a \rangle \langle a \notin S \rangle \langle ev\ a \in P^0 \vee ev\ a \in Q^0 \rangle$  by blast
    thus  $\langle e \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$ 
  proof cases
    fix  $a$  assume  $\langle e = ev\ a \rangle \langle a \in S \rangle \langle ev\ a \in P^0 \rangle \langle ev\ a \in Q^0 \rangle$ 
    have  $* : \langle [ev\ a] \ setinterleaves_{\checkmark tick-join} ([ev\ a], [ev\ a]), S \rangle$ 
      by (simp add:  $\langle a \in S \rangle$ )
    from  $\langle ev\ a \in P^0 \rangle \langle ev\ a \in Q^0 \rangle$  show  $\langle e \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$ 
      by (simp add:  $\langle e = ev\ a \rangle$  initials-def T-Syncptick) (use * in blast)
  next
    fix  $a$  assume  $\langle e = ev\ a \rangle \langle a \notin S \rangle \langle ev\ a \in P^0 \vee ev\ a \in Q^0 \rangle$ 
    from  $\langle ev\ a \in P^0 \vee ev\ a \in Q^0 \rangle$  show  $\langle e \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$ 
  proof (elim disjE)
    assume  $\langle ev\ a \in P^0 \rangle$ 
    have  $* : \langle [ev\ a] \ setinterleaves_{\checkmark tick-join} ([ev\ a], []), S \rangle$ 
      by (simp add:  $\langle a \notin S \rangle$ )
    from  $\langle ev\ a \in P^0 \rangle$  show  $\langle e \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$ 
  by (simp add:  $\langle e = ev\ a \rangle$  initials-def T-Syncptick) (meson * is-processT1-TR)
  next
    assume  $\langle ev\ a \in Q^0 \rangle$ 
    have  $* : \langle [ev\ a] \ setinterleaves_{\checkmark tick-join} ([], [ev\ a]), S \rangle$ 

```

```

      by (simp add: ⟨a ∉ S⟩)
      from ⟨ev a ∈ Q0⟩ show ⟨e ∈ (P [S]✓ Q)0⟩
    by (simp add: ⟨e = ev a⟩ initials-def T-Syncptick) (meson * is-processT1-TR)
  qed
qed
next
  assume ⟨e ∈ ?rhs-tick⟩
  then obtain r-s r s where ⟨tick-join r s = Some r-s⟩
    ⟨e = ✓(r-s)⟩ ⟨✓(r) ∈ P0⟩ ⟨✓(s) ∈ Q0⟩ by blast
  have * : ⟨[✓(r-s)] setinterleaves✓tick-join ([✓(r)], [✓(s)]), S⟩
    by (simp add: ⟨tick-join r s = Some r-s⟩)
  from ⟨✓(r) ∈ P0⟩ ⟨✓(s) ∈ Q0⟩ show ⟨e ∈ (P [S]✓ Q)0⟩
    by (simp add: ⟨e = ✓(r-s)⟩ initials-def T-Syncptick) (use * in blast)
  qed
next
  fix e assume ⟨e ∈ (P [S]✓ Q)0⟩
  then consider t-P t-Q where ⟨t-P ∈ T P⟩ ⟨t-Q ∈ T Q⟩
    ⟨[e] setinterleaves✓tick-join ((t-P, t-Q), S)⟩
  | (div) t u t-P t-Q
  where ⟨[e] = t @ u⟩ ⟨ftF u⟩ ⟨tF t ∨ u = []⟩
    ⟨t setinterleaves✓tick-join ((t-P, t-Q), S)⟩
    ⟨t-P ∈ D P ∧ t-Q ∈ T Q ∨ t-P ∈ T P ∧ t-Q ∈ D Q⟩
  unfolding initials-def T-Syncptick by blast
  thus ⟨e ∈ ?rhs-ev ∪ ?rhs-tick⟩
proof cases
  show ⟨t-P ∈ T P ⟹ t-Q ∈ T Q ⟹
    [e] setinterleaves✓tick-join ((t-P, t-Q), S) ⟹
    e ∈ ?rhs-ev ∪ ?rhs-tick⟩ for t-P t-Q
  by (cases e; cases t-P; cases t-Q)
    (auto simp add: initials-def setinterleavingptick-simps
      split: if-split-asm option.split-asm eventptick.splits
      dest!: Nil-setinterleavesptick)
next
  case div
  have ⟨t ≠ []⟩ by (metis BOT-iff-Nil-D Nil-setinterleavesptick div(4,5) non-BOT)
  hence ⟨t = [e] ∧ u = []⟩
    by (metis append-Cons append-Nil div(1) list.inject neq-Nil-conv)
  with div(4, 5) non-BOT show ⟨e ∈ ?rhs-ev ∪ ?rhs-tick⟩
    by (cases e; cases t-P; cases t-Q)
      (auto simp add: initials-def setinterleavingptick-simps BOT-iff-Nil-D
        split: if-split-asm eventptick.splits option.split-asm
        dest!: Nil-setinterleavesptick intro: D-T)
  qed
qed
qed

```

9.2 Laws of After

9.2.1 Sequential Composition

locale *AfterDuplicated-same-events* = *AfterDuplicated* Ψ_α Ψ_β
for $\Psi_\alpha :: \langle ('a, 'r) \text{ process}_{ptick} \Rightarrow 'a \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
and $\Psi_\beta :: \langle ('a, 's) \text{ process}_{ptick} \Rightarrow 'a \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$
begin

notation *After* _{α} .*After* (**infixl** $\langle \text{after}_\alpha \rangle$ 86)

notation *After* _{β} .*After* (**infixl** $\langle \text{after}_\beta \rangle$ 86)

lemma *not-skippable-or-not-initialR-After-Seq_{ptick}*:

$\langle (P ; \checkmark Q) \text{ after}_\beta a = (\text{if } \text{ev } a \in P^0 \text{ then } P \text{ after}_\alpha a ; \checkmark Q \text{ else } \Psi_\beta (P ; \checkmark Q) a) \rangle$
if $\langle \text{range tick} \cap P^0 = \{\} \vee (\forall r. \checkmark(r) \in P^0 \longrightarrow \text{ev } a \notin (Q r)^0) \rangle$

proof (*cases* $\langle P = \perp \rangle$)

show $\langle P = \perp \implies (P ; \checkmark Q) \text{ after}_\beta a =$
 $(\text{if } \text{ev } a \in P^0 \text{ then } P \text{ after}_\alpha a ; \checkmark Q \text{ else } \Psi_\beta (P ; \checkmark Q) a) \rangle$
by (*simp add: After.After-BOT*)

next

note *denot-projs* = *After.After-projs Seq_{ptick}-projs*

assume *non-BOT*: $\langle P \neq \perp \rangle$

with that have $\$: \langle \text{ev } a \in (P ; \checkmark Q)^0 \longleftrightarrow \text{ev } a \in P^0 \rangle$

by (*auto simp add: initials-Seq_{ptick}*)

show $\langle (P ; \checkmark Q) \text{ after}_\beta a = (\text{if } \text{ev } a \in P^0 \text{ then } P \text{ after}_\alpha a ; \checkmark Q \text{ else } \Psi_\beta (P ; \checkmark Q) a) \rangle$

proof (*split if-split, intro conjI impI*)

from $\$$ **show** $\langle \text{ev } a \notin P^0 \implies (P ; \checkmark Q) \text{ after}_\beta a = \Psi_\beta (P ; \checkmark Q) a \rangle$

by (*simp add: After _{β} .not-initial-After*)

next

assume *initial-P* : $\langle \text{ev } a \in P^0 \rangle$

show $\langle (P ; \checkmark Q) \text{ after}_\beta a = P \text{ after}_\alpha a ; \checkmark Q \rangle$

proof (*rule Process-eq-optimizedI*)

fix t **assume** $\langle t \in \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$

then consider $(D-P) t' u$ **where** $\langle \text{ev } a \# t = \text{map } (\text{ev} \circ \text{of-ev}) t' @ u \rangle \langle t' \in \mathcal{D} P \rangle \langle tF t' \rangle \langle ftF u \rangle$

$| (D-Q) t' r u$ **where** $\langle \text{ev } a \# t = \text{map } (\text{ev} \circ \text{of-ev}) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF t' \rangle \langle u \in \mathcal{D} (Q r) \rangle$

by (*auto simp add: denot-projs \$ initial-P*)

thus $\langle t \in \mathcal{D} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$

proof *cases*

case *D-P* **with** *non-BOT initial-P* **show** $\langle t \in \mathcal{D} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$

by (*cases t'*) (*auto simp add: BOT-iff-Nil-D denot-projs*)

next

case *D-Q* **with** *initial-P* **show** $\langle t \in \mathcal{D} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$

by (*cases t'*, *simp-all add: BOT-iff-Nil-D denot-projs*)

(*metis D-T disjoint-iff initials-memI rangeI that, blast*)

qed

next

```

fix  $t$  assume  $\langle t \in \mathcal{D} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$ 
then consider  $(D-P) \ t' \ u$  where  $\langle t = \text{map} (ev \circ of-ev) \ t' @ u \rangle \langle ev \ a \ \# \ t' \rangle$ 
 $\in \mathcal{D} \ P \rangle \langle tF \ t' \rangle \langle ftF \ u \rangle$ 
  |  $(D-Q) \ t' \ r \ u$  where  $\langle t = \text{map} (ev \circ of-ev) \ t' @ u \rangle \langle ev \ a \ \# \ t' @ [\checkmark(r)] \rangle$ 
 $\in \mathcal{T} \ P \rangle \langle tF \ t' \rangle \langle u \in \mathcal{D} (Q \ r) \rangle$ 
    by  $(\text{auto simp add: denot-projs initial-P})$ 
    thus  $\langle t \in \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
    proof cases
      case  $D-P$  thus  $\langle t \in \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
      by  $(\text{simp add: denot-projs } \$ \text{ initial-P Cons-eq-append-conv Cons-eq-map-conv})$ 
         $(\text{metis event}_{ptick}.disc(1) \text{ event}_{ptick}.sel(1) \text{ tickFree-Cons-iff})$ 
      next
        case  $D-Q$  thus  $\langle t \in \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
        by  $(\text{simp add: denot-projs } \$ \text{ initial-P Cons-eq-append-conv Cons-eq-map-conv})$ 
           $(\text{metis append-Cons event}_{ptick}.sel(1) \text{ is-ev-def tickFree-Cons-iff})$ 
        qed
      next
        fix  $t \ X$  assume  $\langle (t, X) \in \mathcal{F} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle \langle t \notin \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
        then consider  $(F-P) \ t'$  where  $\langle ev \ a \ \# \ t = \text{map} (ev \circ of-ev) \ t' \rangle$ 
 $\langle (t', \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ P \rangle \langle tF \ t' \rangle$ 
        |  $(F-Q) \ t' \ r \ u$  where  $\langle ev \ a \ \# \ t = \text{map} (ev \circ of-ev) \ t' @ u \rangle \langle t' @ [\checkmark(r)] \rangle \in \mathcal{T}$ 
 $P \rangle$ 
           $\langle tF \ t' \rangle \langle (u, X) \in \mathcal{F} (Q \ r) \rangle$ 
          by  $(\text{auto simp add: denot-projs } \$ \text{ initial-P})$ 
             $(\text{metis (mono-tags, lifting) comp-apply list.simps(9) tickFree-Cons-iff})$ 
          thus  $\langle (t, X) \in \mathcal{F} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$ 
          proof cases
            case  $F-P$  thus  $\langle (t, X) \in \mathcal{F} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$ 
            by  $(\text{auto simp add: denot-projs initial-P})$ 
            next
              case  $F-Q$  thus  $\langle (t, X) \in \mathcal{F} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$ 
              by  $(\text{cases } t', \text{ auto simp add: denot-projs initial-P intro: initials-memI})$ 
                 $(\text{metis F-T Int-iff empty-iff initials-memI rangeI that})$ 
              qed
            next
              fix  $t \ X$  assume  $\langle (t, X) \in \mathcal{F} (P \text{ after}_\alpha a ; \checkmark Q) \rangle \langle t \notin \mathcal{D} (P \text{ after}_\alpha a ; \checkmark Q) \rangle$ 
              then consider  $(F-P) \ t'$  where  $\langle t = \text{map} (ev \circ of-ev) \ t' \rangle$ 
 $\langle (ev \ a \ \# \ t', \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ P \rangle \langle tF \ t' \rangle$ 
              |  $(F-Q) \ t' \ r \ u$  where  $\langle t = \text{map} (ev \circ of-ev) \ t' @ u \rangle \langle ev \ a \ \# \ t' @ [\checkmark(r)] \rangle \in \mathcal{T}$ 
 $P \rangle$ 
                 $\langle tF \ t' \rangle \langle (u, X) \in \mathcal{F} (Q \ r) \rangle$ 
                by  $(\text{auto simp add: denot-projs initial-P})$ 
                thus  $\langle (t, X) \in \mathcal{F} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
                proof cases
                  case  $F-P$  thus  $\langle (t, X) \in \mathcal{F} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
                  by  $(\text{simp add: denot-projs } \$ \text{ initial-P Cons-eq-map-conv})$ 
                     $(\text{metis event}_{ptick}.disc(1) \text{ event}_{ptick}.sel(1) \text{ tickFree-Cons-iff})$ 
                  next

```

```

      case F-Q thus  $\langle (t, X) \in \mathcal{F} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
    by (simp add: denot-projs $ initial-P Cons-eq-append-conv Cons-eq-map-conv)
      (metis append-Cons eventptick.disc(1) eventptick.sel(1) tickFree-Cons-iff)
  qed
qed
qed
qed

```

lemma *skippable-not-initialL-After-Seq_{ptick}*:

```

 $\langle (P ; \checkmark Q) \text{ after}_\beta a = ( \text{ if } (\exists r. \checkmark(r) \in P^0 \wedge \text{ev } a \in (Q r)^0)$ 
  then  $\bigcap_{r \in \{r. \checkmark(r) \in P^0 \wedge \text{ev } a \in (Q r)^0\}} Q r \text{ after}_\beta a$ 
  else  $\Psi_\beta (P ; \checkmark Q) a \rangle$ 
(is  $\langle (P ; \checkmark Q) \text{ after}_\beta a = (\text{if } ?\text{prem then } ?\text{rhs else } -) \rangle$ ) if  $\langle \text{ev } a \notin P^0 \rangle$ 
proof –
  note denot-projs = After.After-projs Seqptick-projs GlobalNdet-projs
  from initials-BOT  $\langle \text{ev } a \notin P^0 \rangle$  have non-BOT :  $\langle P \neq \perp \rangle$  by blast
  with  $\langle \text{ev } a \notin P^0 \rangle$  have $ :  $\langle \text{ev } a \in (P ; \checkmark Q)^0 \longleftrightarrow ?\text{prem} \rangle$ 
    by (auto simp add: initials-Seqptick)
  show  $\langle (P ; \checkmark Q) \text{ after}_\beta a = (\text{if } ?\text{prem then } ?\text{rhs else } \Psi_\beta (P ; \checkmark Q) a) \rangle$ 
proof (split if-split, intro conjI impI)
  show  $\langle \neg ?\text{prem} \implies (P ; \checkmark Q) \text{ after}_\beta a = \Psi_\beta (P ; \checkmark Q) a \rangle$ 
    by (rule Afterβ.not-initial-After, use $ in blast)
  next
  show  $\langle (P ; \checkmark Q) \text{ after}_\beta a = ?\text{rhs} \rangle$  if ?prem
proof (rule Process-eq-optimizedI)
  fix t assume  $\langle t \in \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 
  then consider (D-P)  $t' u$  where  $\langle \text{ev } a \# t = \text{map } (\text{ev} \circ \text{of-ev}) t' @ u \rangle \langle t' \in \mathcal{D} P \rangle$ 
 $\langle tF t' \rangle \langle ftF u \rangle$ 
    | (D-Q)  $t' r u$  where  $\langle \text{ev } a \# t = \text{map } (\text{ev} \circ \text{of-ev}) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle$ 
 $\langle tF t' \rangle \langle u \in \mathcal{D} (Q r) \rangle$ 
    by (auto simp add: denot-projs  $\langle ?\text{prem} \rangle$  $  $\langle \text{ev } a \notin P^0 \rangle$ )
  thus  $\langle t \in \mathcal{D} ?\text{rhs} \rangle$ 
proof cases
  case D-P with non-BOT  $\langle \text{ev } a \notin P^0 \rangle$  show  $\langle t \in \mathcal{D} ?\text{rhs} \rangle$ 
  by (simp add: denot-projs Cons-eq-append-conv Cons-eq-map-conv BOT-iff-Nil-D)
    (metis D-T eventptick.collapse(1) initials-memI tickFree-Cons-iff)
  next
  case D-Q with  $\langle \text{ev } a \notin P^0 \rangle$  show  $\langle t \in \mathcal{D} ?\text{rhs} \rangle$ 
  by (simp add: denot-projs Cons-eq-append-conv Cons-eq-map-conv)
    (metis D-T append-Nil eventptick.collapse(1) initials-memI
      is-processT3-TR-append tickFree-Cons-iff)
  qed
  next
  from  $\langle ?\text{prem} \rangle$  show  $\langle t \in \mathcal{D} ?\text{rhs} \implies t \in \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$  for t
  by (simp add: denot-projs $ Cons-eq-append-conv)
    (metis append-Nil initials-memD tickFree-Nil)
  next
  fix t X assume  $\langle (t, X) \in \mathcal{F} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle \langle t \notin \mathcal{D} ((P ; \checkmark Q) \text{ after}_\beta a) \rangle$ 

```

$a\rangle$
then consider $(F-P) \ t'$ **where** $\langle ev \ a \ \# \ t = map \ (ev \circ \ of-ev) \ t' \rangle$
 $\langle (t', \ ref-Seq_{ptick} \ X) \in \mathcal{F} \ P \rangle \ \langle tF \ t' \rangle$
 $| \ (F-Q) \ t' \ r \ u$ **where** $\langle ev \ a \ \# \ t = map \ (ev \circ \ of-ev) \ t' \ @ \ u \rangle$
 $\langle t' \ @ \ [\checkmark(r)] \in \mathcal{T} \ P \rangle \ \langle tF \ t' \rangle \ \langle (u, X) \in \mathcal{F} \ (Q \ r) \rangle$
by $(simp \ add: \ denot-projs \ \$ \ \langle ?prem \rangle) \ \text{metis}$
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
proof cases
case $F-P$ **with** $\langle ev \ a \notin P^0 \rangle$ **show** $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
by $(cases \ t', \ simp-all \ add: \ denot-projs \ \langle ?prem \rangle)$
 $(use \ F-T \ initials-memI \ in \ blast)$
next
case $F-Q$ **with** $\langle ev \ a \notin P^0 \rangle \ \langle ?prem \rangle$ **show** $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
by $(cases \ t', \ auto \ simp \ add: \ denot-projs \ intro: \ initials-memI)$
 $(metis \ F-T \ initials-memI)$
qed
next
from $\langle ?prem \rangle$ **show** $\langle (t, X) \in \mathcal{F} \ ?rhs \implies t \notin \mathcal{D} \ ?rhs \implies$
 $(t, X) \in \mathcal{F} \ ((P ; \checkmark \ Q) \ after_{\beta} \ a) \rangle$ **for** $t \ X$
by $(simp \ add: \ denot-projs \ \$ \ Cons-eq-append-conv \ Cons-eq-map-conv \ split:$
 $if-split-asm)$
 $(metis \ initials-memD \ self-append-conv2 \ tickFree-Nil)$
qed
qed
qed

lemma *skippable-initialL-initialR-After-Seq_{ptick}:*
 $\langle (P ; \checkmark \ Q) \ after_{\beta} \ a = (P \ after_{\alpha} \ a ; \checkmark \ Q) \sqcap (\sqcap r \in \{r. \checkmark(r) \in P^0 \wedge ev \ a \in (Q \ r)^0\}. \ Q \ r \ after_{\beta} \ a) \rangle$
(is $\langle (P ; \checkmark \ Q) \ after_{\beta} \ a = (P \ after_{\alpha} \ a ; \checkmark \ Q) \sqcap ?rhs \rangle$
if $assms : \langle \exists r. \checkmark(r) \in P^0 \wedge ev \ a \in (Q \ r)^0 \rangle \ \langle ev \ a \in P^0 \rangle$
proof $(cases \ \langle P = \perp \rangle)$
show $\langle P = \perp \implies (P ; \checkmark \ Q) \ after_{\beta} \ a = (P \ after_{\alpha} \ a ; \checkmark \ Q) \sqcap ?rhs \rangle$
by $(simp \ add: \ After.After-BOT)$
next
note $denot-projs = After.After-projs \ Seq_{ptick}-projs \ GlobalNdet-projs \ Ndet-projs$
show $\langle (P ; \checkmark \ Q) \ after_{\beta} \ a = (P \ after_{\alpha} \ a ; \checkmark \ Q) \sqcap ?rhs \rangle$ **if** $\langle P \neq \perp \rangle$
proof $(rule \ Process-eq-optimizedI)$
fix t **assume** $\langle t \in \mathcal{D} \ ((P ; \checkmark \ Q) \ after_{\beta} \ a) \rangle$
then consider $(D-P) \ t' \ u$ **where** $\langle ev \ a \ \# \ t = map \ (ev \circ \ of-ev) \ t' \ @ \ u \rangle \ \langle t' \in$
 $\mathcal{D} \ P \rangle \ \langle tF \ t' \rangle \ \langle tF \ u \rangle$
 $| \ (D-Q) \ t' \ r \ u$ **where** $\langle ev \ a \ \# \ t = map \ (ev \circ \ of-ev) \ t' \ @ \ u \rangle \ \langle t' \ @ \ [\checkmark(r)] \in \mathcal{T}$
 $P \rangle \ \langle tF \ t' \rangle \ \langle u \in \mathcal{D} \ (Q \ r) \rangle$
by $(auto \ simp \ add: \ denot-projs \ initials-Seq_{ptick} \ assms \ split: \ if-split-asm)$
thus $\langle t \in \mathcal{D} \ ((P \ after_{\alpha} \ a ; \checkmark \ Q) \sqcap ?rhs) \rangle$
proof cases
case $D-P$ **with** $\langle P \neq \perp \rangle$ **show** $\langle t \in \mathcal{D} \ ((P \ after_{\alpha} \ a ; \checkmark \ Q) \sqcap ?rhs) \rangle$

```

    by (auto simp add: denot-projs assms Cons-eq-append-conv BOT-iff-Nil-D)
  next
    case D-Q thus  $\langle t \in \mathcal{D} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \rangle$ 
    by (auto simp add: denot-projs assms Cons-eq-append-conv)
      (meson D-T initials-memI, blast)
  qed
next
  from  $\langle P \neq \perp \rangle$  show  $\langle t \in \mathcal{D} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \implies t \in \mathcal{D} ((P ;\checkmark Q) \text{ after}_\beta a) \rangle$  for  $t$ 
  by (auto simp add: denot-projs assms initials-Seqptick Cons-eq-append-conv Cons-eq-map-conv)
    (metis eventptick.disc(1) eventptick.sel(1) tickFree-Cons-iff,
     metis Cons-eq-appendI append-T-imp-tickFree eventptick.sel(1) list.distinct(1),
     metis append-Nil initials-memD tickFree-Nil)
next
  fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} ((P ;\checkmark Q) \text{ after}_\beta a) \rangle \langle t \notin \mathcal{D} ((P ;\checkmark Q) \text{ after}_\beta a) \rangle$ 
  then consider (F-P)  $t'$  where  $\langle ev a \# t = \text{map } (ev \circ of-ev) t' \rangle$ 
     $\langle (t', \text{ref-Seq}_{ptick} X) \in \mathcal{F} P \rangle \langle tF t' \rangle$ 
  | (F-Q)  $t' r u$  where  $\langle ev a \# t = \text{map } (ev \circ of-ev) t' @ u \rangle$ 
     $\langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF t' \rangle \langle (u, X) \in \mathcal{F} (Q r) \rangle$ 
  by (simp add: denot-projs assms initials-Seqptick split: if-split-asm) meson+
  thus  $\langle (t, X) \in \mathcal{F} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \rangle$ 
  proof cases
    case F-P thus  $\langle (t, X) \in \mathcal{F} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \rangle$ 
    by (auto simp add: denot-projs assms)
  next
    case F-Q with assms show  $\langle (t, X) \in \mathcal{F} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \rangle$ 
    by (cases  $t'$ , simp-all add: denot-projs)
      (meson F-T initials-memI, blast)
  qed
next
  fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \rangle \langle t \notin \mathcal{D} ((P \text{ after}_\alpha a ;\checkmark Q) \sqcap ?rhs) \rangle$ 
  hence  $\langle (t, X) \in \mathcal{F} (P \text{ after}_\alpha a ;\checkmark Q) \wedge t \notin \mathcal{D} (P \text{ after}_\alpha a ;\checkmark Q) \vee$ 
     $(t, X) \in \mathcal{F} ?rhs \wedge t \notin \mathcal{D} ?rhs \rangle$  by (simp add: Ndet-projs)
  thus  $\langle (t, X) \in \mathcal{F} ((P ;\checkmark Q) \text{ after}_\beta a) \rangle$ 
  proof (elim disjE conjE)
    show  $\langle (t, X) \in \mathcal{F} (P \text{ after}_\alpha a ;\checkmark Q) \implies t \notin \mathcal{D} (P \text{ after}_\alpha a ;\checkmark Q) \implies$ 
       $(t, X) \in \mathcal{F} ((P ;\checkmark Q) \text{ after}_\beta a) \rangle$ 
    by (simp add: denot-projs assms initials-Seqptick  $\langle P \neq \perp \rangle$  Cons-eq-append-conv Cons-eq-map-conv)
      (metis (no-types, lifting) Cons-eq-appendI eventptick.sel(1) is-ev-def tickFree-Cons-iff)
  next
    from assms show  $\langle (t, X) \in \mathcal{F} ?rhs \implies t \notin \mathcal{D} ?rhs \implies (t, X) \in \mathcal{F} ((P ;\checkmark Q) \text{ after}_\beta a) \rangle$ 
    by (simp add: denot-projs initials-Seqptick  $\langle P \neq \perp \rangle$  Cons-eq-append-conv Cons-eq-map-conv split: if-split-asm)
      (metis append-Nil initials-memD tickFree-Nil)

```


qed
qed
qed

lemma *not-initialL-not-initialR-After-Seq_{ptick}*:
 $\langle ev\ a \notin P^0 \implies (\bigwedge r. \checkmark(r) \in P^0 \implies ev\ a \notin (Q\ r)^0) \implies$
 $(P ; \checkmark Q) \text{ after}_\beta a = \Psi_\beta (P ; \checkmark Q) a \rangle$
by (*meson skippable-not-initialL-After-Seq_{ptick}*)

lemma *After-Seq_{ptick}*:
 $\langle (P ; \checkmark Q) \text{ after}_\beta a =$
 $(\text{ if } \forall r. \checkmark(r) \in P^0 \longrightarrow ev\ a \notin (Q\ r)^0$
 $\text{ then if } ev\ a \in P^0 \text{ then } P \text{ after}_\alpha a ; \checkmark Q \text{ else } \Psi_\beta (P ; \checkmark Q) a$
 $\text{ else if } ev\ a \in P^0$
 $\text{ then } (P \text{ after}_\alpha a ; \checkmark Q) \sqcap (\bigwedge r \in \{r. \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}. Q\ r \text{ after}_\beta$
 $a)$
 $\text{ else } \bigwedge r \in \{r. \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}. Q\ r \text{ after}_\beta a) \rangle$
by (*simp add: not-skippable-or-not-initialR-After-Seq_{ptick}*
skippable-initialL-initialR-After-Seq_{ptick} skippable-not-initialL-After-Seq_{ptick})

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* +
 $After_{lhs} : After\ \Psi_{lhs} + After_{rhs} : After\ \Psi_{rhs} + After_{ptick} : After\ \Psi_{ptick}$
for *tick-join* :: $\langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$
and $\Psi_{lhs} :: \langle [('a, 'r) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
and $\Psi_{rhs} :: \langle [('a, 's) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$
and $\Psi_{ptick} :: \langle [('a, 't) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 't) \text{ process}_{ptick} \rangle$
begin

notation $After_{lhs}.After$ (**infixl** $\langle after_{lhs} \rangle$ 86)
notation $After_{rhs}.After$ (**infixl** $\langle after_{rhs} \rangle$ 86)
notation $After_{ptick}.After$ (**infixl** $\langle after_{ptick} \rangle$ 86)

sublocale *After-Sync_{ptick}-locale-sym* :
 $After-Sync_{ptick}\text{-locale } \langle \lambda s\ r. \text{ tick-join } r\ s \rangle \Psi_{rhs} \Psi_{lhs} \Psi_{ptick}$
by *unfold-locales*

lemma *initialL-not-initialR-not-in-After-Sync_{ptick}*:
 $\langle (P \llbracket S \rrbracket \checkmark Q) \text{ after}_{ptick} a = P \text{ after}_{lhs} a \llbracket S \rrbracket \checkmark Q \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)
if *initial-hyps*: $\langle ev\ a \in P^0 \rangle \langle ev\ a \notin Q^0 \rangle$ **and** *notin*: $\langle a \notin S \rangle$
proof (*cases* $\langle P = \perp \vee Q = \perp \rangle$)

```

show  $\langle P = \perp \vee Q = \perp \implies ?lhs = ?rhs \rangle$ 
  by (elim disjE) (simp-all add: Afterptick.After-BOT Afterlhs.After-BOT)
next
from initial-hyps and notin have init :  $\langle ev\ a \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$ 
  by (simp add: initials-Syncptick)
show  $\langle ?lhs = ?rhs \rangle$  if non-BOT :  $\langle \neg (P = \perp \vee Q = \perp) \rangle$ 
proof (rule Process-eq-optimizedI)
  fix t assume  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
  with init have  $\langle ev\ a \# t \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$  by (simp add: Afterptick.D-After)
  then obtain u v t-P t-Q
    where * :  $\langle ev\ a \# t = u @ v \rangle \langle tF\ u \rangle \langle ftF\ v \rangle$ 
     $\langle u\ setinterleaves_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$ 
     $\langle t-P \in \mathcal{D}\ P \wedge t-Q \in \mathcal{T}\ Q \vee t-P \in \mathcal{T}\ P \wedge t-Q \in \mathcal{D}\ Q \rangle$ 
  unfolding D-Syncptick by blast
  from *(4, 5) non-BOT have  $\langle u \neq [] \rangle$ 
    by (auto simp add: BOT-iff-Nil-D dest: Nil-setinterleavesptick)
  with *(1) obtain u' where  $\langle u = ev\ a \# u' \rangle \langle t = u' @ v \rangle$ 
    by (cases u) simp-all
  from *(4, 5) initial-hyps(2) obtain t-P'
    where  $\langle t-P = ev\ a \# t-P' \rangle \langle u' setinterleaves_{\checkmark tick-join} ((t-P', t-Q), S) \rangle$ 
    by (cases t-P; cases t-Q)
    (auto simp add: setinterleavingptick-simps  $\langle u = ev\ a \# u' \rangle$ 
      split: eventptick.splits if-split-asm option.split-asm
      intro: D-T initials-memI)
  moreover from  $\langle t-P = ev\ a \# t-P' \rangle$  *(2, 5)
  have  $\langle t-P' \in \mathcal{D}\ (P\ after_{lhs}\ a) \wedge t-Q \in \mathcal{T}\ Q \vee$ 
     $t-P' \in \mathcal{T}\ (P\ after_{lhs}\ a) \wedge t-Q \in \mathcal{D}\ Q \rangle$ 
    by (simp add: Afterlhs.After-projs initial-hyps(1))
  moreover from *(2) have  $\langle tF\ u' \rangle$  by (simp add:  $\langle u = ev\ a \# u' \rangle$ )
  ultimately show  $\langle t \in \mathcal{D}\ ?rhs \rangle$ 
    using *(3) by (auto simp add:  $\langle t = u' @ v \rangle$  D-Syncptick)
next
fix t assume  $\langle t \in \mathcal{D}\ ?rhs \rangle$ 
then obtain u v t-P t-Q
  where * :  $\langle t = u @ v \rangle \langle tF\ u \rangle \langle ftF\ v \rangle$ 
   $\langle u setinterleaves_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$ 
   $\langle t-P \in \mathcal{D}\ (P\ after_{lhs}\ a) \wedge t-Q \in \mathcal{T}\ Q \vee$ 
   $t-P \in \mathcal{T}\ (P\ after_{lhs}\ a) \wedge t-Q \in \mathcal{D}\ Q \rangle$ 
  unfolding D-Syncptick by blast
  have  $\langle ev\ a \# t = (ev\ a \# u) @ v \rangle$  by (simp add: *(1))
  moreover from *(2) have  $\langle tF\ (ev\ a \# u) \rangle$  by simp
  moreover from *(4)
  have  $\langle ev\ a \# u setinterleaves_{\checkmark tick-join} ((ev\ a \# t-P, t-Q), S) \rangle$ 
    by (metis notin rev.simps(2) rev-setinterleavesptick-rev-rev-iff
      setinterleavesptick-snoc-notinL)
  moreover from *(5) have  $\langle ev\ a \# t-P \in \mathcal{D}\ P \wedge t-Q \in \mathcal{T}\ Q \vee$ 
     $ev\ a \# t-P \in \mathcal{T}\ P \wedge t-Q \in \mathcal{D}\ Q \rangle$ 
    by (simp add: Afterlhs.After-projs initial-hyps(1))
  ultimately have  $\langle ev\ a \# t \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 

```

```

    using *(3) unfolding  $D\text{-Sync}_{ptick}$  by blast
  thus  $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$ 
    by (simp add:  $\text{After}_{ptick}.D\text{-After init}$ )
next
  fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
     $\langle t \notin \mathcal{D} \text{ ?lhs} \rangle$ 
  hence  $\langle (ev\ a \ \# \ t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle \langle ev\ a \ \# \ t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
    by (simp-all add:  $\text{After}_{ptick}.After\text{-projs init}$ )
  then obtain  $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$ 
    where  $*$  :  $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \rangle$ 
       $\langle ev\ a \ \# \ t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t\text{-}P, t\text{-}Q), S) \rangle$ 
       $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X\text{-}P\ S\ X\text{-}Q \rangle$ 
    unfolding  $\text{Sync}_{ptick}\text{-projs}$  by blast
  from *(2, 3) initial-hyps(2) obtain  $t\text{-}P'$ 
    where  $\langle t\text{-}P = ev\ a \ \# \ t\text{-}P' \rangle \langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t\text{-}P', t\text{-}Q), S) \rangle$ 
    by (metis  $\text{Cons-ev-setinterleaves}_{ptick} E\ F\text{-}T\ \text{initials-memI}$ )
  moreover from *(1) have  $\langle (t\text{-}P', X\text{-}P) \in \mathcal{F} (P \text{ after}_{lhs}\ a) \rangle$ 
    by (simp add:  $\langle t\text{-}P = ev\ a \ \# \ t\text{-}P' \rangle \text{After}_{lhs}.F\text{-After initial-hyps}(1)$ )
  ultimately show  $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
    using *(2, 4) by (auto simp add:  $\text{Sync}_{ptick}\text{-projs}$ )
next
  fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
     $\langle t \notin \mathcal{D} \text{ ?rhs} \rangle$ 
  then obtain  $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$ 
    where  $*$  :  $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} (P \text{ after}_{lhs}\ a) \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \rangle$ 
       $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t\text{-}P, t\text{-}Q), S) \rangle$ 
       $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X\text{-}P\ S\ X\text{-}Q \rangle$ 
    unfolding  $\text{Sync}_{ptick}\text{-projs}$  by blast
  from *(1) have  $\langle (ev\ a \ \# \ t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \rangle$ 
    by (simp add:  $\text{After}_{lhs}.F\text{-After initial-hyps}(1)$ )
  moreover from *(3)
  have  $\langle ev\ a \ \# \ t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((ev\ a \ \# \ t\text{-}P, t\text{-}Q), S) \rangle$ 
    by (metis  $\text{notin rev.simps}(2)$   $\text{rev-setinterleaves}_{ptick}\text{-rev-rev-iff}$ 
       $\text{setinterleaves}_{ptick}\text{-snoc-notinL}$ )
  ultimately have  $\langle (ev\ a \ \# \ t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
    using *(2, 4) by (auto simp add:  $F\text{-Sync}_{ptick}$ )
  thus  $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$  by (simp add:  $\text{After}_{ptick}.F\text{-After init}$ )
qed
qed

```

lemma (in $\text{After-Sync}_{ptick}\text{-locale}$) $\text{not-initialL-initialR-not-in-After-Sync}_{ptick}$:
 $\langle (P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{ptick}\ a = P \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{rhs}\ a \rangle$ (is $\langle \text{?lhs} = \text{?rhs} \rangle$)
if initial-hyps : $\langle ev\ a \notin P^0 \rangle \langle ev\ a \in Q^0 \rangle$ **and** notin : $\langle a \notin S \rangle$
using $\text{After-Sync}_{ptick}\text{-locale-sym.initialL-not-initialR-not-in-After-Sync}_{ptick}$
 $[OF\ \langle ev\ a \in Q^0 \rangle \langle ev\ a \notin P^0 \rangle \langle a \notin S \rangle]$
by (simp add: $\text{Sync}_{ptick}\text{-sym}$)

lemma *not-initialL-in-After-Sync_{ptick}*:

$\langle \text{ev } a \notin P^0 \implies a \in S \implies$
 $(P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{\text{ptick}} a = (\text{if } Q = \perp \text{ then } \perp \text{ else } \Psi_{\text{ptick}} (P \llbracket S \rrbracket_{\checkmark} Q) a) \rangle$
by (*simp add: After_{ptick}.After-BOT, rule impI*)
(subst After_{ptick}.not-initial-After, auto simp add: initials-Sync_{ptick})

lemma *not-initialR-in-After-Sync_{ptick}*:

$\langle \text{ev } a \notin Q^0 \implies a \in S \implies$
 $(P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{\text{ptick}} a = (\text{if } P = \perp \text{ then } \perp \text{ else } \Psi_{\text{ptick}} (P \llbracket S \rrbracket_{\checkmark} Q) a) \rangle$
by (*simp add: After_{ptick}.After-BOT, rule impI*)
(subst After_{ptick}.not-initial-After, auto simp add: initials-Sync_{ptick})

lemma *initialL-initialR-in-After-Sync_{ptick}*:

$\langle (P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{\text{ptick}} a = P \text{ after}_{\text{lhs}} a \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{\text{rhs}} a \rangle$ (**is** $\langle ?\text{lhs} = ?\text{rhs} \rangle$)
if *initial-hyps*: $\langle \text{ev } a \in P^0 \rangle \langle \text{ev } a \in Q^0 \rangle$ **and** *inside*: $\langle a \in S \rangle$
proof (*cases* $\langle P = \perp \vee Q = \perp \rangle$)
show $\langle P = \perp \vee Q = \perp \implies ?\text{lhs} = ?\text{rhs} \rangle$
by (*elim disjE*) (*simp-all add: After.After-BOT*)
next
from *initial-hyps inside* **have** *init* : $\langle \text{ev } a \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$
by (*simp add: initials-Sync_{ptick}*)
show $\langle ?\text{lhs} = ?\text{rhs} \rangle$ **if** *non-BOT* : $\langle \neg (P = \perp \vee Q = \perp) \rangle$
proof (*rule Process-eq-optimizedI*)
fix *t* **assume** $\langle t \in \mathcal{D} ((P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{\text{ptick}} a) \rangle$
hence $\langle \text{ev } a \# t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **by** (*simp add: After_{ptick}.D-After init*)
then obtain *u v t-P t-Q*
where $\ast : \langle \text{ev } a \# t = u @ v \rangle \langle tF u \rangle \langle tF v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P, t-Q), S) \rangle$
 $\langle t-P \in \mathcal{D} P \wedge t-Q \in \mathcal{T} Q \vee t-P \in \mathcal{T} P \wedge t-Q \in \mathcal{D} Q \rangle$
unfolding *D-Sync_{ptick}* **by** *blast*
from $\ast(4, 5)$ *non-BOT* **have** $\langle u \neq [] \rangle$
by (*auto simp add: BOT-iff-Nil-D dest: Nil-setinterleaves_{ptick}*)
with $\ast(1)$ **obtain** *u'* **where** $\langle u = \text{ev } a \# u' \rangle \langle t = u' @ v \rangle$
by (*cases u*) *simp-all*
from $\ast(4)$ *inside initial-hyps*(1, 2) $\langle u = \text{ev } a \# u' \rangle$
obtain *t-P' t-Q'* **where** $\langle t-P = \text{ev } a \# t-P' \rangle \langle t-Q = \text{ev } a \# t-Q' \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-P', t-Q'), S) \rangle$
by (*metis (no-types, opaque-lifting) Cons-ev-setinterleaves_{ptick}E*)
moreover from $\ast(2)$ **have** $\langle tF u' \rangle$ **by** (*simp add: $\langle u = \text{ev } a \# u' \rangle$*)
moreover from $\ast(5)$
have $\langle t-P' \in \mathcal{D} (P \text{ after}_{\text{lhs}} a) \wedge t-Q' \in \mathcal{T} (Q \text{ after}_{\text{rhs}} a) \vee$
 $t-P' \in \mathcal{T} (P \text{ after}_{\text{lhs}} a) \wedge t-Q' \in \mathcal{D} (Q \text{ after}_{\text{rhs}} a) \rangle$
by (*simp add: $\langle t-P = \text{ev } a \# t-P' \rangle \langle t-Q = \text{ev } a \# t-Q' \rangle$*)
After.After-projs initial-hyps)
ultimately show $\langle t \in \mathcal{D} ?\text{rhs} \rangle$
using $\ast(3)$ **by** (*auto simp add: $\langle t = u' @ v \rangle$ D-Sync_{ptick}*)
next
fix *t* **assume** $\langle t \in \mathcal{D} ?\text{rhs} \rangle$

then obtain $u \ v \ t\text{-}P \ t\text{-}Q$
where $*$: $\langle t = u \ @ \ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle t\text{-}P \in \mathcal{D} (P \text{ after}_{lhs} a) \wedge t\text{-}Q \in \mathcal{T} (Q \text{ after}_{rhs} a) \vee$
 $t\text{-}P \in \mathcal{T} (P \text{ after}_{lhs} a) \wedge t\text{-}Q \in \mathcal{D} (Q \text{ after}_{rhs} a) \rangle$
unfolding $D\text{-}Sync_{ptick}$ **by** *blast*
from $*(1)$ **have** $\langle ev \ a \ \# \ t = (ev \ a \ \# \ u) \ @ \ v \rangle$ **by** *simp*
moreover from $*(2)$ **have** $\langle tF \ (ev \ a \ \# \ u) \rangle$ **by** *simp*
moreover from $*(4)$ **have** $\langle ev \ a \ \# \ u \text{ setinterleaves}_{\checkmark tick\text{-}join}$
 $((ev \ a \ \# \ t\text{-}P, ev \ a \ \# \ t\text{-}Q), S) \rangle$
by (*simp add: inside*)
moreover from $*(5)$ **have** $\langle ev \ a \ \# \ t\text{-}P \in \mathcal{D} \ P \wedge ev \ a \ \# \ t\text{-}Q \in \mathcal{T} \ Q \vee$
 $ev \ a \ \# \ t\text{-}P \in \mathcal{T} \ P \wedge ev \ a \ \# \ t\text{-}Q \in \mathcal{D} \ Q \rangle$
by (*simp add: After.After-projs initial-hyps*)
ultimately have $\langle ev \ a \ \# \ t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
unfolding $D\text{-}Sync_{ptick}$ **using** $*(3)$ **by** *blast*
thus $\langle t \in \mathcal{D} \ ?lhs \rangle$ **by** (*simp add: After_{ptick}.D-After init*)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle \langle t \notin \mathcal{D} \ ?lhs \rangle$
hence $\langle (ev \ a \ \# \ t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle \langle ev \ a \ \# \ t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*simp-all add: After.After-projs init*)
then obtain $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$
where $*$: $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle$
 $\langle ev \ a \ \# \ t \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X\text{-}P \ S \ X\text{-}Q \rangle$
unfolding $Sync_{ptick}\text{-}projs$ **by** *blast*
from $*(3)$ **obtain** $t\text{-}P' \ t\text{-}Q'$
where $\langle t\text{-}P = ev \ a \ \# \ t\text{-}P' \rangle \langle t\text{-}Q = ev \ a \ \# \ t\text{-}Q' \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P', t\text{-}Q'), S) \rangle$
by (*metis (no-types) Cons-ev-setinterleaves_{ptick} E inside*)
moreover from $*(1, 2)$ **have** $\langle (t\text{-}P', X\text{-}P) \in \mathcal{F} (P \text{ after}_{lhs} a) \rangle$
 $\langle (t\text{-}Q', X\text{-}Q) \in \mathcal{F} (Q \text{ after}_{rhs} a) \rangle$
by (*simp-all add: $\langle t\text{-}P = ev \ a \ \# \ t\text{-}P' \rangle \langle t\text{-}Q = ev \ a \ \# \ t\text{-}Q' \rangle$*
 $After.F\text{-}After \text{ initial-hyps}$)
ultimately show $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
by (*auto simp add: F-Sync_{ptick} intro!: *(4)*)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle \langle t \notin \mathcal{D} \ ?rhs \rangle$
then obtain $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$
where $*$: $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} (P \text{ after}_{lhs} a) \rangle$
 $\langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} (Q \text{ after}_{rhs} a) \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{ tick-join } X\text{-}P \ S \ X\text{-}Q \rangle$
unfolding $Sync_{ptick}\text{-}projs$ **by** *blast*
from $*(1, 2)$ **have** $\langle (ev \ a \ \# \ t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle$
 $\langle (ev \ a \ \# \ t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle$
by (*simp-all add: After.F-After initial-hyps*)
moreover from $*(3)$ **have** $\langle ev \ a \ \# \ t \text{ setinterleaves}_{\checkmark tick\text{-}join}$
 $((ev \ a \ \# \ t\text{-}P, ev \ a \ \# \ t\text{-}Q), S) \rangle$

```

    by (simp add: inside)
  ultimately have  $\langle \text{ev } a \# t, X \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q)$ 
    using  $\ast(4)$  by (simp (no-asm) add:  $F\text{-Sync}_{ptick}$ ) blast
  thus  $\langle t, X \rangle \in \mathcal{F} \text{ ?lhs}$  by (simp add:  $\text{After}_{ptick}.F\text{-After init}$ )
qed
qed

```

lemma *initialL-initialR-not-in-After-Sync_{ptick}*:

```

 $\langle (P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{ptick} a = (P \text{ after}_{lhs} a \llbracket S \rrbracket_{\checkmark} Q) \sqcap (P \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{rhs} a) \rangle$ 
(is  $\langle \text{?lhs} = \text{?rhs1} \sqcap \text{?rhs2} \rangle$ )
if initial-hyps:  $\langle \text{ev } a \in P^0 \rangle \langle \text{ev } a \in Q^0 \rangle$  and notin:  $\langle a \notin S \rangle$ 
proof (cases  $\langle P = \perp \vee Q = \perp \rangle$ )
  show  $\langle P = \perp \vee Q = \perp \implies \text{?lhs} = \text{?rhs1} \sqcap \text{?rhs2} \rangle$ 
    by (elim disjE) (simp-all add:  $\text{After.After-BOT}$ )
next
  from initial-hyps(1) notin have init :  $\langle \text{ev } a \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$ 
    by (simp add: initials-Syncptick)
  show  $\langle \text{?lhs} = \text{?rhs1} \sqcap \text{?rhs2} \rangle$  if non-BOT :  $\langle \neg (P = \perp \vee Q = \perp) \rangle$ 
  proof (rule Process-eq-optimizedI)
    fix t assume  $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$ 
    hence  $\langle \text{ev } a \# t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
      by (simp add:  $\text{After}_{ptick}.D\text{-After init}$ )
    then obtain u v t-P t-Q
      where  $\ast : \langle \text{ev } a \# t = u @ v \rangle \langle tF u \rangle \langle tF v \rangle$ 
       $\langle u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$ 
       $\langle t-P \in \mathcal{D} P \wedge t-Q \in \mathcal{T} Q \vee t-P \in \mathcal{T} P \wedge t-Q \in \mathcal{D} Q \rangle$ 
      unfolding  $D\text{-Sync}_{ptick}$  by blast
    from  $\ast(4, 5)$  non-BOT have  $\langle u \neq [] \rangle$ 
      by (auto simp add: BOT-iff-Nil-D dest: Nil-setinterleavesptick)
    with  $\ast(1)$  obtain u' where  $\langle u = \text{ev } a \# u' \rangle \langle t = u' @ v \rangle$ 
      by (cases u) simp-all
    from  $\ast(2)$  have  $\langle tF u' \rangle$  by (simp add:  $\langle u = \text{ev } a \# u' \rangle$ )
    from  $\ast(4)$  notin  $\langle u = \text{ev } a \# u' \rangle$ 
    consider t-P' where  $\langle t-P = \text{ev } a \# t-P' \rangle$ 
       $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P', t-Q), S) \rangle$ 
    | t-Q' where  $\langle t-Q = \text{ev } a \# t-Q' \rangle$ 
       $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q'), S) \rangle$ 
      by (metis (no-types) Cons-ev-setinterleavesptick E)
    thus  $\langle t \in \mathcal{D} (\text{?rhs1} \sqcap \text{?rhs2}) \rangle$ 
  proof cases
    fix t-P' assume  $\$ : \langle t-P = \text{ev } a \# t-P' \rangle$ 
       $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P', t-Q), S) \rangle$ 
    from  $\ast(5)$  have  $\langle t-P' \in \mathcal{D} (P \text{ after}_{lhs} a) \wedge t-Q \in \mathcal{T} Q \vee$ 
       $t-P' \in \mathcal{T} (P \text{ after}_{lhs} a) \wedge t-Q \in \mathcal{D} Q \rangle$ 
      by (simp add:  $\$(1)$   $\text{After}_{lhs}.After\text{-projs initial-hyps}(1)$ )
    with  $\$(2) \ast(3) \langle tF u' \rangle$  have  $\langle t \in \mathcal{D} \text{ ?rhs1} \rangle$ 
      by (auto simp add:  $\langle t = u' @ v \rangle D\text{-Sync}_{ptick}$ )
  end
end

```

```

    thus  $\langle t \in \mathcal{D} (?rhs1 \sqcap ?rhs2) \rangle$  by (simp add: D-Ndet)
  next
    fix  $t-Q'$  assume  $\$ : \langle t-Q = ev\ a \ \# \ t-Q' \rangle$ 
     $\langle u' \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q'), S) \rangle$ 
    from  $*(5)$  have  $\langle t-P \in \mathcal{D}\ P \wedge t-Q' \in \mathcal{T}\ (Q \text{ after}_{rhs}\ a) \vee$ 
       $t-P \in \mathcal{T}\ P \wedge t-Q' \in \mathcal{D}\ (Q \text{ after}_{rhs}\ a) \rangle$ 
    by (simp add:  $\$(1)$  Afterrhs.After-projs initial-hyps(2))
    with  $\$(2) \ * (3) \ \langle tF\ u' \rangle$  have  $\langle t \in \mathcal{D}\ ?rhs2 \rangle$ 
    by (auto simp add:  $\langle t = u' @ v \rangle$  D-Syncptick)
    thus  $\langle t \in \mathcal{D} (?rhs1 \sqcap ?rhs2) \rangle$  by (simp add: D-Ndet)
  qed
next
  fix  $t$  assume  $\langle t \in \mathcal{D} (?rhs1 \sqcap ?rhs2) \rangle$ 
  then consider  $\langle t \in \mathcal{D}\ ?rhs1 \rangle \mid \langle t \in \mathcal{D}\ ?rhs2 \rangle$ 
  by (auto simp add: D-Ndet)
  hence  $\langle ev\ a \ \# \ t \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  proof cases
    assume  $\langle t \in \mathcal{D}\ ?rhs1 \rangle$ 
    then obtain  $u\ v\ t-P\ t-Q$ 
    where  $\$ : \langle t = u @ v \rangle \langle tF\ u \rangle \langle ftF\ v \rangle$ 
     $\langle u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$ 
     $\langle t-P \in \mathcal{D}\ (P \text{ after}_{lhs}\ a) \wedge t-Q \in \mathcal{T}\ Q \vee$ 
       $t-P \in \mathcal{T}\ (P \text{ after}_{lhs}\ a) \wedge t-Q \in \mathcal{D}\ Q \rangle$ 
    unfolding D-Syncptick by blast
    from  $*(1)$  have  $\langle ev\ a \ \# \ t = (ev\ a \ \# \ u) @ v \rangle$  by simp
    moreover from  $*(2)$  have  $\langle tF\ (ev\ a \ \# \ u) \rangle$  by simp
    moreover from  $*(4)$ 
    have  $\langle ev\ a \ \# \ u \text{ setinterleaves}_{\checkmark tick-join} ((ev\ a \ \# \ t-P, t-Q), S) \rangle$ 
    by (metis notin rev.simps(2) rev-setinterleavesptick-rev-rev-iff
      setinterleavesptick-snoc-notinL)
    moreover from  $*(5)$  have  $\langle ev\ a \ \# \ t-P \in \mathcal{D}\ P \wedge t-Q \in \mathcal{T}\ Q \vee$ 
       $ev\ a \ \# \ t-P \in \mathcal{T}\ P \wedge t-Q \in \mathcal{D}\ Q \rangle$ 
    by (simp add: Afterlhs.After-projs initial-hyps(1))
    ultimately show  $\langle ev\ a \ \# \ t \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
    using  $*(3)$  unfolding D-Syncptick by blast
  next
    assume  $\langle t \in \mathcal{D}\ ?rhs2 \rangle$ 
    then obtain  $u\ v\ t-P\ t-Q$ 
    where  $\$ : \langle t = u @ v \rangle \langle tF\ u \rangle \langle ftF\ v \rangle$ 
     $\langle u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$ 
     $\langle t-P \in \mathcal{D}\ P \wedge t-Q \in \mathcal{T}\ (Q \text{ after}_{rhs}\ a) \vee$ 
       $t-P \in \mathcal{T}\ P \wedge t-Q \in \mathcal{D}\ (Q \text{ after}_{rhs}\ a) \rangle$ 
    unfolding D-Syncptick by blast
    from  $*(1)$  have  $\langle ev\ a \ \# \ t = (ev\ a \ \# \ u) @ v \rangle$  by simp
    moreover from  $*(2)$  have  $\langle tF\ (ev\ a \ \# \ u) \rangle$  by simp
    moreover from  $*(4)$  have  $\langle ev\ a \ \# \ u \text{ setinterleaves}_{\checkmark tick-join} ((t-P, ev\ a \ \#$ 
 $t-Q), S) \rangle$ 
    by (metis notin rev.simps(2) rev-setinterleavesptick-rev-rev-iff
      setinterleavesptick-snoc-notinR)
  end

```

```

moreover from  $\ast(5)$  have  $\langle t-P \in \mathcal{D} P \wedge \text{ev } a \# t-Q \in \mathcal{T} Q \vee$ 
 $t-P \in \mathcal{T} P \wedge \text{ev } a \# t-Q \in \mathcal{D} Q \rangle$ 
  by (simp add: Afterrhs.After-projs initial-hyps(2))
ultimately show  $\langle \text{ev } a \# t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  using  $\ast(3)$  unfolding D-Syncptick by blast
qed
thus  $\langle t \in \mathcal{D} ?lhs \rangle$  by (simp add: Afterptick.D-After init)
next
fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} ?lhs \rangle$   $\langle t \notin \mathcal{D} ?lhs \rangle$ 
hence  $\langle (\text{ev } a \# t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$   $\langle \text{ev } a \# t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  by (simp-all add: Afterptick.After-projs init)
then obtain  $t-P \ t-Q \ X-P \ X-Q$ 
  where  $\ast : \langle (t-P, X-P) \in \mathcal{F} P \rangle \langle (t-Q, X-Q) \in \mathcal{F} Q \rangle$ 
 $\langle \text{ev } a \# t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t-P, t-Q), S) \rangle$ 
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{tick-join } X-P \ S \ X-Q \rangle$ 
  unfolding Syncptick-projs by blast
from  $\ast(3)$  notin
consider  $t-P'$  where  $\langle t-P = \text{ev } a \# t-P' \rangle$ 
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t-P', t-Q), S) \rangle$ 
 $| \ t-Q'$  where  $\langle t-Q = \text{ev } a \# t-Q' \rangle$ 
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t-P, t-Q'), S) \rangle$ 
  by (metis (no-types) Cons-ev-setinterleavesptick E)
thus  $\langle (t, X) \in \mathcal{F} (?rhs1 \sqcap ?rhs2) \rangle$ 
proof cases
  fix  $t-P'$  assume  $\$ : \langle t-P = \text{ev } a \# t-P' \rangle$ 
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t-P', t-Q), S) \rangle$ 
  from  $\ast(1)$  have  $\langle (t-P', X-P) \in \mathcal{F} (P \text{ after}_{lhs} a) \rangle$ 
  by (simp add: $(1) Afterlhs.F-After initial-hyps(1))
  with  $\$(2) \ast(2, 4)$  have  $\langle (t, X) \in \mathcal{F} ?rhs1 \rangle$ 
  by (auto simp add: F-Syncptick)
  thus  $\langle (t, X) \in \mathcal{F} (?rhs1 \sqcap ?rhs2) \rangle$  by (simp add: F-Ndet)
next
  fix  $t-Q'$  assume  $\$ : \langle t-Q = \text{ev } a \# t-Q' \rangle$ 
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t-P, t-Q'), S) \rangle$ 
  from  $\ast(2)$  have  $\langle (t-Q', X-Q) \in \mathcal{F} (Q \text{ after}_{rhs} a) \rangle$ 
  by (simp add: $(1) Afterrhs.F-After initial-hyps(2))
  with  $\$(2) \ast(1, 4)$  have  $\langle (t, X) \in \mathcal{F} ?rhs2 \rangle$ 
  by (auto simp add: F-Syncptick)
  thus  $\langle (t, X) \in \mathcal{F} (?rhs1 \sqcap ?rhs2) \rangle$  by (simp add: F-Ndet)
qed
next
fix  $t X$  assume  $\langle (t, X) \in \mathcal{F} (?rhs1 \sqcap ?rhs2) \rangle$   $\langle t \notin \mathcal{D} (?rhs1 \sqcap ?rhs2) \rangle$ 
then consider  $\langle (t, X) \in \mathcal{F} ?rhs1 \rangle$   $\langle t \notin \mathcal{D} ?rhs1 \rangle$ 
 $| \ \langle (t, X) \in \mathcal{F} ?rhs2 \rangle$   $\langle t \notin \mathcal{D} ?rhs2 \rangle$ 
  by (auto simp add: Ndet-projs)
hence  $\langle (\text{ev } a \# t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
proof cases
  assume  $\langle (t, X) \in \mathcal{F} ?rhs1 \rangle$   $\langle t \notin \mathcal{D} ?rhs1 \rangle$ 
  then obtain  $t-P \ t-Q \ X-P \ X-Q$ 

```


where $*$: $\langle (t-P, X-P) \in \mathcal{F} (P \text{ after}_{lhs} a) \rangle \langle (t-Q, X-Q) \in \mathcal{F} Q \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} tick-join X-P S X-Q \rangle$
unfolding $\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
from $*(1)$ **have** $\langle (ev a \# t-P, X-P) \in \mathcal{F} P \rangle$
by (*simp add: After_{lhs}.F-After initial-hyps(1)*)
moreover from $*(3)$
have $\langle ev a \# t \text{ setinterleaves}_{\checkmark tick-join} ((ev a \# t-P, t-Q), S) \rangle$
by (*metis notin rev.simps(2) rev-setinterleaves_{ptick}-rev-rev-iff setinterleaves_{ptick}-snoc-notinL*)
ultimately show $\langle (ev a \# t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*auto simp add: F-Sync_{ptick} intro!: *(2, 4)*)
next
assume $\langle (t, X) \in \mathcal{F} ?rhs2 \rangle \langle t \notin \mathcal{D} ?rhs2 \rangle$
then obtain $t-P \ t-Q \ X-P \ X-Q$
where $*$: $\langle (t-P, X-P) \in \mathcal{F} P \rangle \langle (t-Q, X-Q) \in \mathcal{F} (Q \text{ after}_{rhs} a) \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark tick-join} ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} tick-join X-P S X-Q \rangle$
unfolding $\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
from $*(2)$ **have** $\langle (ev a \# t-Q, X-Q) \in \mathcal{F} Q \rangle$
by (*simp add: After_{rhs}.F-After initial-hyps(2)*)
moreover from $*(3)$
have $\langle ev a \# t \text{ setinterleaves}_{\checkmark tick-join} ((t-P, ev a \# t-Q), S) \rangle$
by (*metis notin rev.simps(2) rev-setinterleaves_{ptick}-rev-rev-iff setinterleaves_{ptick}-snoc-notinR*)
ultimately show $\langle (ev a \# t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*auto simp add: F-Sync_{ptick} intro!: *(1, 4)*)
qed
thus $\langle (t, X) \in \mathcal{F} ?lhs \rangle$ **by** (*simp add: After_{ptick}.F-After init*)
qed
qed

lemma *not-initialL-not-initialR-After-Sync_{ptick}* :
 $\langle ev a \notin P^0 \implies ev a \notin Q^0 \implies (P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{ptick} a = \Psi_{ptick} (P \llbracket S \rrbracket_{\checkmark} Q) a \rangle$
by (*subst After_{ptick}.not-initial-After*) (*auto simp add: initials-Sync_{ptick}*)

Finally, the monster theorem !

theorem *After-Sync_{ptick}*:
 $\langle (P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{ptick} a =$
 $(\text{ if } P = \perp \vee Q = \perp \text{ then } \perp$
 $\text{ else if } ev a \in P^0 \wedge ev a \in Q^0$
 $\text{ then if } a \in S \text{ then } P \text{ after}_{lhs} a \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{rhs} a$
 $\text{ else } (P \text{ after}_{lhs} a \llbracket S \rrbracket_{\checkmark} Q) \sqcap (P \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{rhs} a)$
 $\text{ else if } ev a \in P^0 \wedge a \notin S \text{ then } P \text{ after}_{lhs} a \llbracket S \rrbracket_{\checkmark} Q$
 $\text{ else if } ev a \in Q^0 \wedge a \notin S \text{ then } P \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{rhs} a$
 $\text{ else } \Psi_{ptick} (P \llbracket S \rrbracket_{\checkmark} Q) a \rangle$
by (*auto simp add: After_{ptick}.After-BOT initialL-not-initialR-not-in-After-Sync_{ptick}*)

$initialL-initialR-in-After-Sync_{ptick}$ $initialL-initialR-not-in-After-Sync_{ptick}$
 $not-initialL-initialR-not-in-After-Sync_{ptick}$ $not-initialL-not-initialR-After-Sync_{ptick}$
 $not-initialR-in-After-Sync_{ptick}$ $not-initialL-in-After-Sync_{ptick}$

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_α Ω_α Ψ_β Ω_β
for $\Psi_\alpha :: \langle ['a, 'r] process_{ptick}, 'a \rangle \Rightarrow ('a, 'r) process_{ptick}$
and $\Omega_\alpha :: \langle ['a, 'r] process_{ptick}, 'r \rangle \Rightarrow ('a, 'r) process_{ptick}$
and $\Psi_\beta :: \langle ['a, 's] process_{ptick}, 'a \rangle \Rightarrow ('a, 's) process_{ptick}$
and $\Omega_\beta :: \langle ['a, 's] process_{ptick}, 's \rangle \Rightarrow ('a, 's) process_{ptick}$

sublocale *AfterExtDuplicated-same-events* $\subseteq$ *AfterDuplicated-same-events* .
 — Recovering *AfterDuplicated-same-events.After-Seq_{ptick}*

context *AfterExtDuplicated-same-events*
begin

notation *After_{tick} α .After* (**infixl** $\langle after_\alpha \rangle$ 86)
notation *After_{tick} β .After* (**infixl** $\langle after_\beta \rangle$ 86)
notation *After_{tick} α .After_{tick}* (**infixl** $\langle after_{\checkmark\alpha} \rangle$ 86)
notation *After_{tick} β .After_{tick}* (**infixl** $\langle after_{\checkmark\beta} \rangle$ 86)

lemma *After_{tick}-Seq_{ptick}* :

$\langle (P ;_{\checkmark} Q) after_{\checkmark\beta} e =$
 (case e of $\checkmark(r) \Rightarrow \Omega_\beta (P ;_{\checkmark} Q) r$
 $\mid ev\ a \Rightarrow$
 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 $\Psi_\beta (P ;_{\checkmark} Q) a$
 else if $ev\ a \in P^0$
 then $(P after_{\checkmark\alpha} ev\ a ;_{\checkmark} Q) \sqcap$
 $(\sqcap r \in \{r. \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}. Q\ r after_{\checkmark\beta} ev\ a)$
 else $\sqcap r \in \{r. \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}. Q\ r after_{\checkmark\beta} ev\ a \rangle$

by (auto simp add: After_{tick} β .After_{tick}-def After_{tick} α .After_{tick}-def After-Seq_{ptick}
split: event_{ptick}.split)

end

Synchronization Product

locale AfterExt-Sync_{ptick}-locale = Sync_{ptick}-locale tick-join +

AfterExt_{lhs} : AfterExt Ψ_{lhs} Ω_{lhs} +

AfterExt_{rhs} : AfterExt Ψ_{rhs} Ω_{rhs} +

AfterExt_{ptick} : AfterExt Ψ_{ptick} Ω_{ptick}

for tick-join :: $\langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$

and $\Psi_{lhs} :: \langle [('a, 'r) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$

and $\Omega_{lhs} :: \langle [('a, 'r) \text{ process}_{ptick}, 'r] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$

and $\Psi_{rhs} :: \langle [('a, 's) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$

and $\Omega_{rhs} :: \langle [('a, 's) \text{ process}_{ptick}, 's] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$

and $\Psi_{ptick} :: \langle [('a, 't) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 't) \text{ process}_{ptick} \rangle$

and $\Omega_{ptick} :: \langle [('a, 't) \text{ process}_{ptick}, 't] \Rightarrow ('a, 't) \text{ process}_{ptick} \rangle$

begin

sublocale After-Sync_{ptick}-locale tick-join Ψ_{lhs} Ψ_{rhs} Ψ_{ptick} by unfold-locales

sublocale AfterExt-Sync_{ptick}-locale-sym :

AfterExt-Sync_{ptick}-locale $\langle \lambda s r. \text{ tick-join } r s \rangle \Psi_{rhs}$ Ω_{rhs} Ψ_{lhs} Ω_{lhs} Ψ_{ptick} Ω_{ptick}

by unfold-locales

notation AfterExt_{lhs}.After_{tick} (infixl $\langle \text{after}_{\checkmark lhs} \rangle$ 86)

notation AfterExt_{rhs}.After_{tick} (infixl $\langle \text{after}_{\checkmark rhs} \rangle$ 86)

notation AfterExt_{ptick}.After_{tick} (infixl $\langle \text{after}_{\checkmark ptick} \rangle$ 86)

theorem After_{tick}-Sync_{ptick}:

$\langle (P \llbracket S \rrbracket_{\checkmark} Q) \text{ after}_{\checkmark ptick} e =$

(case e of $\checkmark(r-s) \Rightarrow \Omega_{ptick} (P \llbracket S \rrbracket_{\checkmark} Q) r-s$

| $ev a \Rightarrow$

if $P = \perp \vee Q = \perp$ then $\perp$

else if $ev a \in P^0 \wedge ev a \in Q^0$

then if $a \in S$ then $P \text{ after}_{\checkmark lhs} ev a \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{\checkmark rhs} ev a$

else $(P \text{ after}_{\checkmark lhs} ev a \llbracket S \rrbracket_{\checkmark} Q) \sqcap (P \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{\checkmark rhs} ev a)$

else if $ev a \in P^0 \wedge a \notin S$ then $P \text{ after}_{\checkmark lhs} ev a \llbracket S \rrbracket_{\checkmark} Q$

else if $ev a \in Q^0 \wedge a \notin S$ then $P \llbracket S \rrbracket_{\checkmark} Q \text{ after}_{\checkmark rhs} ev a$

else $\Psi_{ptick} (P \llbracket S \rrbracket_{\checkmark} Q) a \rangle$

by (cases e) (simp-all add: AfterExt.After_{tick}-def After-Sync_{ptick})

end

9.3.3 Generic Operational Semantics as Locales

Sequential Composition

locale *OpSemTransitionsDuplicated-same-events* =
OpSemTransitionsDuplicated Ψ_α Ω_α τ -trans $_\alpha$ Ψ_β Ω_β τ -trans $_\beta$
for $\Psi_\alpha :: \langle [(\text{'a', 'r'}) \text{ process}_{ptick}, \text{'a'}] \Rightarrow (\text{'a', 'r'}) \text{ process}_{ptick} \rangle$
and $\Omega_\alpha :: \langle [(\text{'a', 'r'}) \text{ process}_{ptick}, \text{'r'}] \Rightarrow (\text{'a', 'r'}) \text{ process}_{ptick} \rangle$
and τ -trans $_\alpha :: \langle [(\text{'a', 'r'}) \text{ process}_{ptick}, (\text{'a', 'r'}) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl** $\langle_\alpha \rightsquigarrow_\tau \rangle$ 50)
and $\Psi_\beta :: \langle [(\text{'a', 's'}) \text{ process}_{ptick}, \text{'a'}] \Rightarrow (\text{'a', 's'}) \text{ process}_{ptick} \rangle$
and $\Omega_\beta :: \langle [(\text{'a', 's'}) \text{ process}_{ptick}, \text{'s'}] \Rightarrow (\text{'a', 's'}) \text{ process}_{ptick} \rangle$
and τ -trans $_\beta :: \langle [(\text{'a', 's'}) \text{ process}_{ptick}, (\text{'a', 's'}) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl** $\langle_\beta \rightsquigarrow_\tau \rangle$ 50)
sublocale *OpSemTransitionsDuplicated-same-events* $\subseteq$ *AfterExtDuplicated-same-events*
by *unfold-locales*
context *OpSemTransitionsDuplicated-same-events* **begin**
notation *OpSemTransitions* $_\alpha$.ev-trans $(\langle - \rangle_\alpha \rightsquigarrow_- \rightarrow [50, 3, 51] 50)$
notation *OpSemTransitions* $_\alpha$.tick-trans $(\langle - \rangle_\alpha \rightsquigarrow_\checkmark_- \rightarrow [50, 3, 51] 50)$
notation *OpSemTransitions* $_\beta$.ev-trans $(\langle - \rangle_\beta \rightsquigarrow_- \rightarrow [50, 3, 51] 50)$
notation *OpSemTransitions* $_\beta$.tick-trans $(\langle - \rangle_\beta \rightsquigarrow_\checkmark_- \rightarrow [50, 3, 51] 50)$
lemma τ -trans-Seq $_{ptick}R$: $\langle P ;_\checkmark Q \rangle_\beta \rightsquigarrow_\tau Q'$ **if** $\langle P \rangle_\alpha \rightsquigarrow_\checkmark_\tau P'$ **and** $\langle Q \rangle_r \beta \rightsquigarrow_\tau Q'$
proof –
from *that(1)* **have** $\langle P \rangle \sqsubseteq_{FD} SKIP \ r$
by (*meson OpSemTransitions* $_\alpha$.exists-tick-trans-is-initial-tick initial-tick-iff-FD-SKIP)
with *FD-iff-eq-Ndet* **have** $\langle P \rangle = P \sqcap SKIP \ r$..
hence $\langle P ;_\checkmark Q \rangle = (P ;_\checkmark Q) \sqcap (SKIP \ r ;_\checkmark Q)$
by (*metis Seq* $_{ptick}$ -distrib-Ndet-right)
also have $\langle \dots \rangle = (P ;_\checkmark Q) \sqcap Q \ r$ **by** *simp*
also have $\langle \dots \rangle_\beta \rightsquigarrow_\tau Q \ r$ **by** (*simp add: OpSemTransitions* $_\beta$. τ -trans-NdetR)
finally show $\langle P ;_\checkmark Q \rangle_\beta \rightsquigarrow_\tau Q'$ **by** (*rule OpSemTransitions* $_\beta$. τ -trans-transitivity)
(fact that(2))
qed
lemma $\langle \checkmark(r) \rangle \in P^0 \implies Q \ r \ \beta \rightsquigarrow_e Q' \implies P ;_\checkmark Q \ \beta \rightsquigarrow_e Q'$ **for** $P :: \langle (\text{'a', 'r'}) \text{ process}_{ptick} \rangle$
by (*meson OpSemTransitions* $_\beta$. τ -trans-eq *OpSemTransitions* $_\beta$. τ -trans-ev-trans τ -trans-Seq $_{ptick}R$ *OpSemTransitions* $_\alpha$.exists-tick-trans-is-initial-tick)
end

locale *OpSemTransitionsSeq* $_{ptick}$ =
OpSemTransitionsDuplicated-same-events Ψ_α Ω_α τ -trans $_\alpha$ Ψ_β Ω_β τ -trans $_\beta$

for $\Psi_\alpha :: \langle [('a, 'r) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
and $\Omega_\alpha :: \langle [('a, 'r) \text{ process}_{ptick}, 'r] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
and $\tau\text{-trans}_\alpha :: \langle [('a, 'r) \text{ process}_{ptick}, ('a, 'r) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl**
 $\langle \alpha \rightsquigarrow_\tau \rangle$ 50)
and $\Psi_\beta :: \langle [('a, 's) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$
and $\Omega_\beta :: \langle [('a, 's) \text{ process}_{ptick}, 's] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$
and $\tau\text{-trans}_\beta :: \langle [('a, 's) \text{ process}_{ptick}, ('a, 's) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl** $\langle \beta \rightsquigarrow_\tau \rangle$
50) +
assumes $\tau\text{-trans-Seq}_{ptick} L : \langle P \alpha \rightsquigarrow_\tau P' \Longrightarrow P ; \checkmark Q \beta \rightsquigarrow_\tau P' ; \checkmark Q \rangle$
begin

lemma $ev\text{-trans-Seq}_{ptick} L : \langle P \alpha \rightsquigarrow_e P' \Longrightarrow P ; \checkmark Q \beta \rightsquigarrow_e P' ; \checkmark Q \rangle$
by (*cases* $\langle P = \perp \rangle$, *solves* $\langle \text{simp add: OpSemTransitions}_\beta.BOT\text{-ev-trans-anything} \rangle$)
(auto simp add: OpSemTransitions}_\beta.ev-trans-def After_{tick-Seq}_{ptick} initials-Seq}_{ptick}
OpSemTransitions}_\alpha.ev-trans-def
intro: OpSemTransitions}_\beta.\tau\text{-trans-NdetL OpSemTransitions}_\beta.\tau\text{-trans-transitivity}
 $\tau\text{-trans-Seq}_{ptick} L$)

lemmas $Seq_{ptick}\text{-OpSem-rules} = \tau\text{-trans-Seq}_{ptick} L \text{ ev-trans-Seq}_{ptick} L \tau\text{-trans-Seq}_{ptick} R$

end

Synchronization Product

locale $OpSemTransitions\text{-Sync}_{ptick}\text{-locale} = Sync_{ptick}\text{-locale} \langle (\otimes \checkmark) \rangle +$
 $OpSemTransitions_{lhs} : OpSemTransitions \Psi_{lhs} \Omega_{lhs} \langle (lhs \rightsquigarrow_\tau) \rangle +$
 $OpSemTransitions_{rhs} : OpSemTransitions \Psi_{rhs} \Omega_{rhs} \langle (rhs \rightsquigarrow_\tau) \rangle +$
 $OpSemTransitions_{ptick} : OpSemTransitions \Psi_{ptick} \Omega_{ptick} \langle (ptick \rightsquigarrow_\tau) \rangle$
for $tick\text{-join} :: \langle 'r \Rightarrow 's \Rightarrow 't \text{ option} \rangle$ (**infixl** $\langle \otimes \checkmark \rangle$ 100)
and $\Psi_{lhs} :: \langle [('a, 'r) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
and $\Omega_{lhs} :: \langle [('a, 'r) \text{ process}_{ptick}, 'r] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
and $\tau\text{-trans}_{lhs} :: \langle [('a, 'r) \text{ process}_{ptick}, ('a, 'r) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl**
 $\langle lhs \rightsquigarrow_\tau \rangle$ 50)
and $\Psi_{rhs} :: \langle [('a, 's) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$
and $\Omega_{rhs} :: \langle [('a, 's) \text{ process}_{ptick}, 's] \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$
and $\tau\text{-trans}_{rhs} :: \langle [('a, 's) \text{ process}_{ptick}, ('a, 's) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl**
 $\langle rhs \rightsquigarrow_\tau \rangle$ 50)
and $\Psi_{ptick} :: \langle [('a, 't) \text{ process}_{ptick}, 'a] \Rightarrow ('a, 't) \text{ process}_{ptick} \rangle$
and $\Omega_{ptick} :: \langle [('a, 't) \text{ process}_{ptick}, 't] \Rightarrow ('a, 't) \text{ process}_{ptick} \rangle$
and $\tau\text{-trans}_{ptick} :: \langle [('a, 't) \text{ process}_{ptick}, ('a, 't) \text{ process}_{ptick}] \Rightarrow \text{bool} \rangle$ (**infixl**
 $\langle ptick \rightsquigarrow_\tau \rangle$ 50) +
assumes $\tau\text{-trans-Sync}_{ptick} L : \langle P lhs \rightsquigarrow_\tau P' \Longrightarrow P \llbracket A \rrbracket \checkmark Q ptick \rightsquigarrow_\tau P' \llbracket A \rrbracket \checkmark Q \rangle$
and $\tau\text{-trans-Sync}_{ptick} R : \langle Q rhs \rightsquigarrow_\tau Q' \Longrightarrow P \llbracket A \rrbracket \checkmark Q ptick \rightsquigarrow_\tau P \llbracket A \rrbracket \checkmark Q' \rangle$
begin

sublocale $AfterExt\text{-Sync}_{ptick}\text{-locale}$ **by** *unfold-locales*

sublocale *OpSemTransitions-Sync_{ptick}-locale-sym* :
OpSemTransitions-Sync_{ptick}-locale
 $\langle \lambda s \ r. \ r \otimes \checkmark \ s \rangle \Psi_{rhs} \ \Omega_{rhs} \ \langle (rhs \rightsquigarrow \tau) \rangle \Psi_{lhs} \ \Omega_{lhs} \ \langle (lhs \rightsquigarrow \tau) \rangle \Psi_{ptick} \ \Omega_{ptick} \ \langle (ptick \rightsquigarrow \tau) \rangle$
proof *unfold-locale*
 show $\langle P \rightsquigarrow_{rhs} P' \rangle \implies P \llbracket A \rrbracket_{\checkmark}^{sym} Q \rightsquigarrow_{ptick} P' \llbracket A \rrbracket_{\checkmark}^{sym} Q$ for $P \ P' \ A \ Q$
 by (*simp add: Sync_{ptick}-sym τ -trans-Sync_{ptick}R*)
next
 show $\langle Q \rightsquigarrow_{lhs} Q' \rangle \implies P \llbracket A \rrbracket_{\checkmark}^{sym} Q \rightsquigarrow_{ptick} P \llbracket A \rrbracket_{\checkmark}^{sym} Q'$ for $Q \ Q' \ A \ P$
 by (*simp add: Sync_{ptick}-sym τ -trans-Sync_{ptick}L*)
qed

notation *OpSemTransitions_{lhs}.ev-trans* $\langle _ \rightsquigarrow_{lhs} _ \rangle [50, 3, 51] \ 50$
notation *OpSemTransitions_{lhs}.tick-trans* $\langle _ \rightsquigarrow_{lhs} _ \rangle_{\checkmark} [50, 3, 51] \ 50$
notation *OpSemTransitions_{rhs}.ev-trans* $\langle _ \rightsquigarrow_{rhs} _ \rangle [50, 3, 51] \ 50$
notation *OpSemTransitions_{rhs}.tick-trans* $\langle _ \rightsquigarrow_{rhs} _ \rangle_{\checkmark} [50, 3, 51] \ 50$
notation *OpSemTransitions_{ptick}.ev-trans* $\langle _ \rightsquigarrow_{ptick} _ \rangle [50, 3, 51] \ 50$
notation *OpSemTransitions_{ptick}.tick-trans* $\langle _ \rightsquigarrow_{ptick} _ \rangle_{\checkmark} [50, 3, 51] \ 50$

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 :
 $\langle P \llbracket A \rrbracket_{\checkmark} Q \rightsquigarrow_{ptick} SKIP \ r \llbracket A \rrbracket_{\checkmark} Q \rangle$ if $\langle P \rightsquigarrow_{lhs} _ \rangle_{\checkmark} P'$
proof –
 from *that have* $\langle P \sqsubseteq_{FD} SKIP \ r \rangle$
 by (*simp add: OpSemTransitions_{lhs}.tick-trans-def initial-tick-iff-FD-SKIP*)
 with *FD-iff-eq-Ndet* **have** $\langle P = P \sqcap SKIP \ r \rangle$..
 hence $\langle P \llbracket A \rrbracket_{\checkmark} Q = (P \llbracket A \rrbracket_{\checkmark} Q) \sqcap (SKIP \ r \llbracket A \rrbracket_{\checkmark} Q) \rangle$
 by (*metis Sync_{ptick}-distrib-Ndet-right*)
 also **have** $\langle \dots \rightsquigarrow_{ptick} SKIP \ r \llbracket A \rrbracket_{\checkmark} Q \rangle$
 by (*fact OpSemTransitions_{ptick}. τ -trans-NdetR*)
 finally **show** $\langle P \llbracket A \rrbracket_{\checkmark} Q \rightsquigarrow_{ptick} SKIP \ r \llbracket A \rrbracket_{\checkmark} Q \rangle$.
qed

lemma τ -trans-SKIP-Sync_{ptick}R :
 $\langle P \llbracket A \rrbracket_{\checkmark} Q \rightsquigarrow_{ptick} P \llbracket A \rrbracket_{\checkmark} SKIP \ s \rangle$ if $\langle Q \rightsquigarrow_{rhs} _ \rangle_{\checkmark} Q'$
proof –
 from *that have* $\langle Q \sqsubseteq_{FD} SKIP \ s \rangle$
 by (*simp add: OpSemTransitions_{rhs}.tick-trans-def initial-tick-iff-FD-SKIP*)
 with *FD-iff-eq-Ndet* **have** $\langle Q = Q \sqcap SKIP \ s \rangle$..
 hence $\langle P \llbracket A \rrbracket_{\checkmark} Q = (P \llbracket A \rrbracket_{\checkmark} Q) \sqcap (P \llbracket A \rrbracket_{\checkmark} SKIP \ s) \rangle$
 by (*metis Sync_{ptick}-distrib-Ndet-left*)
 also **have** $\langle \dots \rightsquigarrow_{ptick} P \llbracket A \rrbracket_{\checkmark} SKIP \ s \rangle$
 by (*fact OpSemTransitions_{ptick}. τ -trans-NdetR*)
 finally **show** $\langle P \llbracket A \rrbracket_{\checkmark} Q \rightsquigarrow_{ptick} P \llbracket A \rrbracket_{\checkmark} SKIP \ s \rangle$.
qed

lemma *tick-trans-SKIP-Sync_{ptick}-SKIP*:

$\langle r \otimes \checkmark s = \text{Some } r\text{-}s \implies \text{SKIP } r \llbracket A \rrbracket \checkmark \text{SKIP } s \text{ }_{ptick} \rightsquigarrow \checkmark r\text{-}s \Omega_{ptick} (\text{SKIP } r\text{-}s) \text{ }_{r\text{-}s} \rangle$
by (*simp add: OpSemTransitions_{ptick}.SKIP-trans-tick- Ω -SKIP*)

lemma *ev-trans-Sync_{ptick}L* :
 $\langle a \notin A \implies P \text{ }_{lhs} \rightsquigarrow_a P' \implies P \llbracket A \rrbracket \checkmark Q \text{ }_{ptick} \rightsquigarrow_a P' \llbracket A \rrbracket \checkmark Q \rangle$
by (*auto simp add: OpSemTransitions_{ptick}.ev-trans-def After_{tick}-Sync_{ptick} initials-Sync_{ptick} OpSemTransitions_{lhs}.ev-trans-def intro: OpSemTransitions_{ptick}.BOT- τ -trans-anything OpSemTransitions_{ptick}. τ -trans-NdetL OpSemTransitions_{ptick}. τ -trans-transitivity τ -trans-Sync_{ptick}L*)

lemma *ev-trans-Sync_{ptick}R* :
 $\langle a \notin A \implies Q \text{ }_{rhs} \rightsquigarrow_a Q' \implies P \llbracket A \rrbracket \checkmark Q \text{ }_{ptick} \rightsquigarrow_a P \llbracket A \rrbracket \checkmark Q' \rangle$
by (*auto simp add: OpSemTransitions_{ptick}.ev-trans-def After_{tick}-Sync_{ptick} initials-Sync_{ptick} OpSemTransitions_{rhs}.ev-trans-def intro: OpSemTransitions_{ptick}.BOT- τ -trans-anything OpSemTransitions_{ptick}. τ -trans-NdetR OpSemTransitions_{ptick}. τ -trans-transitivity τ -trans-Sync_{ptick}R*)

lemma *ev-trans-Sync_{ptick}LR* :
 $\langle a \in A \implies P \text{ }_{lhs} \rightsquigarrow_a P' \implies Q \text{ }_{rhs} \rightsquigarrow_a Q' \implies P \llbracket A \rrbracket \checkmark Q \text{ }_{ptick} \rightsquigarrow_a P' \llbracket A \rrbracket \checkmark Q' \rangle$
by (*auto simp add: OpSemTransitions_{ptick}.ev-trans-def OpSemTransitions_{lhs}.ev-trans-def OpSemTransitions_{rhs}.ev-trans-def After_{tick}-Sync_{ptick} initials-Sync_{ptick} intro: OpSemTransitions_{ptick}.BOT- τ -trans-anything OpSemTransitions_{ptick}. τ -trans-transitivity τ -trans-Sync_{ptick}L τ -trans-Sync_{ptick}R*)

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*

$\langle \lambda r s. \text{if } r = s \text{ then } [r] \text{ else } \Diamond \rangle$
 $\langle \lambda s r. \text{if } s = r \text{ then } [s] \text{ else } \Diamond \rangle$ *id id*
by *unfold-locales (auto split: if-split-asm)*

notation *Sync_{Classic}.Sync_{ptick}* $\langle (- \llbracket \checkmark_{Classic} - \rrbracket) \rangle$ [70, 0, 71] 70)

notation *Sync_{Classic}.Inter_{ptick}* $\langle (- \parallel \checkmark_{Classic} -) \rangle$ [72, 73] 72)

notation *Sync_{Classic}.Par_{ptick}* $\langle (- \parallel \checkmark_{Classic} -) \rangle$ [74, 75] 74)

10.1.2 Product Type

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

$\langle \lambda r s. [(r, s)] \rangle \langle \lambda s r. [(s, r)] \rangle \text{prod.swap prod.swap}$
by *unfold-locales (auto split: if-split-asm)*

notation $\text{Sync}_{\text{Pair}}.\text{Sync}_{\text{ptick}}$ $(\langle (- \llbracket - \rrbracket \checkmark_{\text{Pair}} -) \rangle [70, 0, 71] 70)$
notation $\text{Sync}_{\text{Pair}}.\text{Inter}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Pair}} -) \rangle [72, 73] 72)$
notation $\text{Sync}_{\text{Pair}}.\text{Par}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Pair}} -) \rangle [74, 75] 74)$

10.1.3 List Type

Pair

interpretation $\text{Sync}_{\text{Pairlist}} : \text{Sync}_{\text{ptick}}\text{-comm-locale}$
 $\langle \lambda r s. [[r, s]] \rangle \langle \lambda s r. [[s, r]] \rangle$
 $\langle \lambda rs. [rs ! \text{Suc } 0, rs ! 0] \rangle \langle \lambda rs. [rs ! \text{Suc } 0, rs ! 0] \rangle$
by *unfold-locales (auto intro: inj-onI)*

notation $\text{Sync}_{\text{Pairlist}}.\text{Sync}_{\text{ptick}}$ $(\langle (- \llbracket - \rrbracket \checkmark_{\text{Pairlist}} -) \rangle [70, 0, 71] 70)$
notation $\text{Sync}_{\text{Pairlist}}.\text{Inter}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Pairlist}} -) \rangle [72, 73] 72)$
notation $\text{Sync}_{\text{Pairlist}}.\text{Par}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Pairlist}} -) \rangle [74, 75] 74)$

Right List

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

interpretation $\text{Sync}_{\text{Rlist}} : \text{Sync}_{\text{ptick}}\text{-comm-locale}$
 $\langle \lambda r s. [r \# s] \rangle \langle \lambda s r. [s @ [r]] \rangle$
 $\langle \text{rotate1} \rangle \langle \lambda rs. \text{if } rs = [] \text{ then } [] \text{ else last } rs \# \text{butlast } rs \rangle$
 $\text{— } \lambda rs. \text{last } rs \# \text{butlast } rs \text{ is not injective.}$
by *unfold-locales (auto intro: inj-onI)*

notation $\text{Sync}_{\text{Rlist}}.\text{Sync}_{\text{ptick}}$ $(\langle (- \llbracket - \rrbracket \checkmark_{\text{Rlist}} -) \rangle [70, 0, 71] 70)$
notation $\text{Sync}_{\text{Rlist}}.\text{Inter}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Rlist}} -) \rangle [72, 73] 72)$
notation $\text{Sync}_{\text{Rlist}}.\text{Par}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Rlist}} -) \rangle [74, 75] 74)$

Left List

Here, we want to have one process of type $(\text{'a'}, \text{'r list'}) \text{process}_{\text{ptick}}$ on the left hand side, and one of type $(\text{'a'}, \text{'r'}) \text{process}_{\text{ptick}}$ 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 $\text{Sync}_{\text{Rlist}}.\text{Sync}_{\text{ptick}}\text{-comm-locale-sym}.\text{Sync}_{\text{ptick}}$ $(\langle (- \llbracket - \rrbracket \checkmark_{\text{Llist}} -) \rangle [70, 0, 71] 70)$
notation $\text{Sync}_{\text{Rlist}}.\text{Sync}_{\text{ptick}}\text{-comm-locale-sym}.\text{Inter}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Llist}} -) \rangle [72, 73] 72)$
notation $\text{Sync}_{\text{Rlist}}.\text{Sync}_{\text{ptick}}\text{-comm-locale-sym}.\text{Par}_{\text{ptick}}$ $(\langle (- \parallel \checkmark_{\text{Llist}} -) \rangle [74, 75] 74)$

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*

$\langle \lambda r s. \text{if length } r = \text{lenL then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda s r. \text{if length } r = \text{lenL then } [s @ r] \text{ else } \Diamond \rangle$
 $\langle \lambda rs. \text{drop lenL } rs @ \text{take lenL } rs \rangle$
 $\langle \lambda rs. \text{rev (take lenL (rev rs)) } @ \text{rev (drop lenL (rev rs))} \rangle$
by *unfold-locales (auto split: if-split-asm)*

end

abbreviation *Sync_{ListslenL}-syntax* ::

$\langle [(\text{'a, 'r list}) \text{ process}_{\text{ptick}}, \text{nat, 'a set, ('a, 'r list) process}_{\text{ptick}}]$
 $\Rightarrow (\text{'a, 'r list}) \text{ process}_{\text{ptick}} \rangle \langle (- \text{-(} \llbracket - \rrbracket \checkmark_{\text{ListslenL}} \text{)} -) \rangle [70, 0, 0, 71] 70 \rangle$
where $\langle P \text{ lenL } \llbracket A \rrbracket \checkmark_{\text{ListslenL}} Q \equiv \text{Sync}_{\text{ListslenL}}.\text{Sync}_{\text{ptick}} \text{ lenL } P A Q \rangle$

abbreviation *Inter_{ListslenL}-syntax* ::

$\langle [(\text{'a, 'r list}) \text{ process}_{\text{ptick}}, \text{nat, ('a, 'r list) process}_{\text{ptick}}]$
 $\Rightarrow (\text{'a, 'r list}) \text{ process}_{\text{ptick}} \rangle \langle (- \text{-(} \llbracket \llbracket - \rrbracket \checkmark_{\text{ListslenL}} \rrbracket -) \rangle [72, 0, 73] 72 \rangle$
where $\langle P \text{ lenL } \llbracket \llbracket - \rrbracket \checkmark_{\text{ListslenL}} Q \equiv \text{Sync}_{\text{ListslenL}}.\text{Inter}_{\text{ptick}} \text{ lenL } P Q \rangle$

abbreviation *Par_{ListslenL}-syntax* ::

$\langle [(\text{'a, 'r list}) \text{ process}_{\text{ptick}}, \text{nat, ('a, 'r list) process}_{\text{ptick}}]$
 $\Rightarrow (\text{'a, 'r list}) \text{ process}_{\text{ptick}} \rangle \langle (- \text{-(} \llbracket \checkmark_{\text{ListslenL}} \rrbracket -) \rangle [74, 0, 75] 75 \rangle$
where $\langle P \text{ lenL } \llbracket \checkmark_{\text{ListslenL}} Q \equiv \text{Sync}_{\text{ListslenL}}.\text{Par}_{\text{ptick}} \text{ lenL } P Q \rangle$

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* ::

$\langle [(\text{'a, 'r list}) \text{ process}_{\text{ptick}}, \text{nat, 'a set, ('a, 'r list) process}_{\text{ptick}}]$
 $\Rightarrow (\text{'a, 'r list}) \text{ process}_{\text{ptick}} \rangle \langle (- \text{-(} \llbracket - \rrbracket \checkmark_{\text{ListslenR}} \text{)} -) \rangle [70, 0, 0, 71] 70 \rangle$
where $\langle P \text{ lenL } \llbracket A \rrbracket \checkmark_{\text{ListslenR}} Q \equiv \text{Sync}_{\text{ListslenL}}.\text{Sync}_{\text{ptick-comm-locale-sym}}.\text{Sync}_{\text{ptick}} \text{ lenL } P A Q \rangle$

abbreviation *Inter_{ListslenR}-syntax* ::

$\langle [(\text{'a, 'r list}) \text{ process}_{\text{ptick}}, \text{nat, ('a, 'r list) process}_{\text{ptick}}]$
 $\Rightarrow (\text{'a, 'r list}) \text{ process}_{\text{ptick}} \rangle \langle (- \text{-(} \llbracket \llbracket - \rrbracket \checkmark_{\text{ListslenR}} \rrbracket -) \rangle [72, 0, 73] 72 \rangle$
where $\langle P \text{ lenL } \llbracket \llbracket - \rrbracket \checkmark_{\text{ListslenR}} Q \equiv \text{Sync}_{\text{ListslenL}}.\text{Sync}_{\text{ptick-comm-locale-sym}}.\text{Inter}_{\text{ptick}} \text{ lenL } P Q \rangle$

abbreviation $Par_{ListslenR}\text{-syntax} ::$
 $\langle [(('a, 'r \text{ list}) \text{ process}_{ptick}, nat, ('a, 'r \text{ list}) \text{ process}_{ptick})]$
 $\Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle (\langle (- \text{ } (||\checkmark_{ListslenR}) - \text{ }) \rangle [74, 0, 75] 75)$
where $\langle P \text{ }_{lenL} ||\checkmark_{ListslenR} Q \equiv Sync_{ListslenL}.Sync_{ptick}\text{-locale-sym}.Par_{ptick}$
 $lenL P Q \rangle$

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

global-interpretation $Sync_{Lists} : Sync_{ptick}\text{-comm-locale}$
 $\langle \lambda r s. \text{ if length } r = lenL \wedge \text{ length } s = lenR \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda s r. \text{ if length } s = lenR \wedge \text{ length } r = lenL \text{ then } [s @ r] \text{ else } \Diamond \rangle$
 $\langle \lambda rs. \text{ drop } lenL rs @ \text{ take } lenL rs \rangle$
 $\langle \lambda rs. \text{ drop } lenR rs @ \text{ take } lenR rs \rangle$
by $unfold\text{-locales}$ ($auto \text{ split: if-split-asm}$)

end

abbreviation $Sync_{Lists}\text{-syntax} ::$
 $\langle [(('a, 'r \text{ list}) \text{ process}_{ptick}, nat, 'a \text{ set}, nat, ('a, 'r \text{ list}) \text{ process}_{ptick})]$
 $\Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle (\langle (- \text{ } (||\checkmark) - \text{ }) \rangle [70, 0, 0, 0, 71] 70)$
where $\langle P \text{ }_{lenL} [A] \checkmark_{lenR} Q \equiv Sync_{Lists}.Sync_{ptick} \text{ }_{lenL} \text{ }_{lenR} P A Q \rangle$

abbreviation $Inter_{Lists}\text{-syntax} ::$
 $\langle [(('a, 'r \text{ list}) \text{ process}_{ptick}, nat, nat, ('a, 'r \text{ list}) \text{ process}_{ptick})]$
 $\Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle (\langle (- \text{ } (||\checkmark) - \text{ }) \rangle [72, 0, 0, 73] 72)$
where $\langle P \text{ }_{lenL} ||\checkmark_{lenR} Q \equiv Sync_{Lists}.Inter_{ptick} \text{ }_{lenL} \text{ }_{lenR} P Q \rangle$

abbreviation $Par_{Lists}\text{-syntax} ::$
 $\langle [(('a, 'r \text{ list}) \text{ process}_{ptick}, nat, nat, ('a, 'r \text{ list}) \text{ process}_{ptick})]$
 $\Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle (\langle (- \text{ } (||\checkmark) - \text{ }) \rangle [74, 0, 0, 75] 75)$
where $\langle P \text{ }_{lenL} ||\checkmark_{lenR} Q \equiv Sync_{Lists}.Par_{ptick} \text{ }_{lenL} \text{ }_{lenR} P Q \rangle$

10.2 Associativities

10.2.1 Classical Version

lemma $Sync_{Classic}\text{-assoc} :$
 $\langle P [S] \checkmark_{Classic} (Q [S] \checkmark_{Classic} R) = P [S] \checkmark_{Classic} Q [S] \checkmark_{Classic} R \rangle$
proof –
let $?f = \langle \lambda r s. \text{ if } r = s \text{ then } [r] \text{ else } \Diamond \rangle$
interpret $* : Sync_{ptick}\text{-assoc-locale } ?f ?f ?f ?f \text{ id id}$
by ($unfold\text{-locales}$) ($auto \text{ split: if-split-asm}$)
show $?thesis$ **by** ($\text{fact } *.Sync_{ptick}\text{-assoc}[\text{simplified Renaming-id}]$)
qed

10.2.2 Product Type

lemma $Sync_{Pair}\text{-assoc} :$

$\langle P \llbracket S \rrbracket_{\checkmark Pair} (Q \llbracket S \rrbracket_{\checkmark Pair} R) = RenamingTick (P \llbracket S \rrbracket_{\checkmark Pair} Q \llbracket S \rrbracket_{\checkmark Pair} R)$
 $(\lambda((r, s), t). (r, s, t)) \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale* $\langle \lambda r s. \lfloor (r, s) \rfloor \rangle \langle \lambda r s. \lfloor (r, s) \rfloor \rangle \langle \lambda r s. \lfloor (r, s) \rfloor \rangle$
 $\langle \lambda r s. \lfloor (r, s) \rfloor \rangle \langle \lambda((r, s), t). (r, s, t) \rangle \langle \lambda(r, s, t). ((r, s), t) \rangle$
by *unfold-locale auto*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc*)
qed

10.2.3 List Type

lemma *Sync_{Rlist}-Sync_{Pairlist}-assoc* :
 $\langle P \llbracket S \rrbracket_{\checkmark Rlist} (Q \llbracket S \rrbracket_{\checkmark Pairlist} R) = (P \llbracket S \rrbracket_{\checkmark Pairlist} Q) \llbracket S \rrbracket_{\checkmark Llist} R \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale* $\langle \lambda r s. \lfloor [r, s] \rfloor \rangle \langle \lambda r s. \lfloor r @ [s] \rfloor \rangle$
 $\langle \lambda r s. \lfloor r \# s \rfloor \rangle \langle \lambda r s. \lfloor [r, s] \rfloor \rangle id id$
by *unfold-locale auto*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc*[*unfolded Renaming-id*])
qed

lemma *Sync_{Rlist}-Sync_{Llist}-assoc* :
 $\langle P \llbracket S \rrbracket_{\checkmark Rlist} (Q \llbracket S \rrbracket_{\checkmark Llist} R) = (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \llbracket S \rrbracket_{\checkmark Llist} R \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale* $\langle \lambda r s. \lfloor r \# s \rfloor \rangle \langle \lambda r s. \lfloor r @ [s] \rfloor \rangle$
 $\langle \lambda r s. \lfloor r \# s \rfloor \rangle \langle \lambda r s. \lfloor r @ [s] \rfloor \rangle id id$
by *unfold-locale auto*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc*[*unfolded Renaming-id*])
qed

lemma *Sync_{Rlist}-Sync_{ListslenL}-assoc* :
 $\langle P \llbracket S \rrbracket_{\checkmark Rlist} (Q \text{ len } Q \llbracket S \rrbracket_{\checkmark ListslenL} R) = (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \text{ Suc len } Q \llbracket S \rrbracket_{\checkmark ListslenL} R \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale*
 $\langle \lambda r s. \lfloor r \# s \rfloor \rangle$
 $\langle \lambda r s. \text{if length } r = \text{Suc len } Q \text{ then } \lfloor r @ s \rfloor \text{ else } \Diamond \rangle$
 $\langle \lambda r s. \lfloor r \# s \rfloor \rangle$
 $\langle \lambda r s. \text{if length } r = \text{len } Q \text{ then } \lfloor r @ s \rfloor \text{ else } \Diamond \rangle id id$
by *unfold-locale (auto split: if-split-asm)*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc*[*unfolded Renaming-id*])
qed

lemma *Sync_{ListslenR}-Sync_{Llist}-assoc* :
 $\langle P \text{ Suc len } Q \llbracket S \rrbracket_{\checkmark ListslenR} (Q \llbracket S \rrbracket_{\checkmark Llist} R) = (P \text{ len } Q \llbracket S \rrbracket_{\checkmark ListslenR} Q) \llbracket S \rrbracket_{\checkmark Llist} R \rangle$
proof –

interpret * : *Sync_{ptick}-assoc-locale*
 $\langle \lambda r s. \text{if length } s = \text{len}Q \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. [r @ [s]] \rangle$
 $\langle \lambda r s. \text{if length } s = \text{Suc len}Q \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. [r @ [s]] \rangle \text{ id id}$
by *unfold-locales (auto split: if-split-asm)*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc[unfolded Renaming-id]*)
qed

lemma *Sync_{Lists}-assoc* :
 $\langle P \text{ len}P \llbracket S \rrbracket \checkmark \text{len}Q + \text{len}R \ (Q \text{ len}Q \llbracket S \rrbracket \checkmark \text{len}R \ R) =$
 $P \text{ len}P \llbracket S \rrbracket \checkmark \text{len}Q \ Q \text{ len}P + \text{len}Q \llbracket S \rrbracket \checkmark \text{len}R \ R \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale*
 $\langle \lambda r s. \text{if length } r = \text{len}P \wedge \text{length } s = \text{len}Q \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. \text{if length } r = \text{len}P + \text{len}Q \wedge \text{length } s = \text{len}R \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. \text{if length } r = \text{len}P \wedge \text{length } s = \text{len}Q + \text{len}R \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. \text{if length } r = \text{len}Q \wedge \text{length } s = \text{len}R \text{ then } [r @ s] \text{ else } \Diamond \rangle \text{ id id}$
by *unfold-locales (auto split: if-split-asm)*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc[unfolded Renaming-id]*)
qed

lemma *Sync_{Rlist}-Sync_{Rlist}-assoc* :
 $\langle P \llbracket S \rrbracket \checkmark \text{Rlist} \ (Q \llbracket S \rrbracket \checkmark \text{Rlist} \ R) = (P \llbracket S \rrbracket \checkmark \text{Pairlist} \ Q) \text{ Suc } (\text{Suc } 0) \llbracket S \rrbracket \checkmark \text{Listslen}L \ R \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale*
 $\langle \lambda r s. [[r, s]] \rangle$
 $\langle \lambda r s. \text{if length } r = \text{Suc } (\text{Suc } 0) \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. [r \# s] \rangle$
 $\langle \lambda r s. [r \# s] \rangle \text{ id id}$
by *unfold-locales (auto split: if-split-asm)*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc[unfolded Renaming-id]*)
qed

lemma *Sync_{ListslenR}-Sync_{Pairlist}-assoc* :
 $\langle P \text{ Suc } (\text{Suc } 0) \llbracket S \rrbracket \checkmark \text{Listslen}R \ (Q \llbracket S \rrbracket \checkmark \text{Pairlist} \ R) = (P \llbracket S \rrbracket \checkmark \text{Llist} \ Q) \llbracket S \rrbracket \checkmark \text{Llist} \ R \rangle$
proof –
interpret * : *Sync_{ptick}-assoc-locale*
 $\langle \lambda r s. [r @ [s]] \rangle$
 $\langle \lambda r s. [r @ [s]] \rangle$
 $\langle \lambda r s. \text{if length } s = \text{Suc } (\text{Suc } 0) \text{ then } [r @ s] \text{ else } \Diamond \rangle$
 $\langle \lambda r s. [[r, s]] \rangle \text{ id id}$
by *unfold-locales (auto split: if-split-asm)*
show ?thesis **by** (fact *.*Sync_{ptick}-assoc[unfolded Renaming-id]*)
qed

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 SyncClassic-is-Sync :  $\langle P \llbracket A \rrbracket_{\checkmark \text{Classic}} Q = P \llbracket A \rrbracket Q \rangle$ 
proof (rule Process-eq-optimizedI)
  show  $\langle t \in \mathcal{D} (P \llbracket A \rrbracket Q) \implies t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark \text{Classic}} Q) \rangle$  for  $t$ 
    by (simp add: D-Sync SyncClassic.D-Syncptick'
      flip: setinterleaves-is-setinterleavesptick
      (metis setinterleaving-sym)
    next
      show  $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark \text{Classic}} Q) \implies t \in \mathcal{D} (P \llbracket A \rrbracket Q) \rangle$  for  $t$ 
        by (simp add: D-Sync SyncClassic.D-Syncptick
          flip: setinterleaves-is-setinterleavesptick
          (metis setinterleaving-sym)
        next
          fix  $t \ X$  assume  $\langle (t, X) \in \mathcal{F} (P \llbracket A \rrbracket Q) \rangle$   $\langle t \notin \mathcal{D} (P \llbracket A \rrbracket Q) \rangle$ 
          then obtain  $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$ 
            where  $*$  :  $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} Q \rangle$ 
               $\langle t \text{ setinterleaves } ((t\text{-}P, t\text{-}Q), \text{range tick} \cup \text{ev } 'A) \rangle$ 
               $\langle X = (X\text{-}P \cup X\text{-}Q) \cap (\text{range tick} \cup \text{ev } 'A) \cup X\text{-}P \cap X\text{-}Q \rangle$ 
            unfolding Sync-projs by blast
            from  $*(4)$  have  $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\lambda r \ s. \text{if } r = s \text{ then } \lfloor r \rfloor \text{ else } \Diamond) X\text{-}P$ 
 $A \ X\text{-}Q \rangle$ 
              by (auto simp add: super-ref-Syncptick-def subset-iff)
              (metis eventptick.exhaust)
            with  $*(1-3)$  show  $\langle (t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\checkmark \text{Classic}} Q) \rangle$ 
              by (auto simp add: SyncClassic.F-Syncptick setinterleaves-is-setinterleavesptick)
            next
              fix  $t \ X$  assume  $\langle (t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\checkmark \text{Classic}} Q) \rangle$   $\langle t \notin \mathcal{D} (P \llbracket A \rrbracket_{\checkmark \text{Classic}} Q) \rangle$ 
              then obtain  $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$ 
                where  $*$  :  $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} Q \rangle$ 
                   $\langle t \text{ setinterleaves } \checkmark_{\lambda r \ s. \text{if } r = s \text{ then } \lfloor r \rfloor \text{ else } \Diamond} ((t\text{-}P, t\text{-}Q), A) \rangle$ 
                   $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} (\lambda r \ s. \text{if } r = s \text{ then } \lfloor r \rfloor \text{ else } \Diamond) X\text{-}P \ A \ X\text{-}Q \rangle$ 
                unfolding SyncClassic.Syncptick-projs by blast
                from  $*(1-3)$  have  $\langle (t, (X\text{-}P \cup X\text{-}Q) \cap (\text{range tick} \cup \text{ev } 'A) \cup X\text{-}P \cap X\text{-}Q) \in$ 
 $\mathcal{F} (P \llbracket A \rrbracket Q) \rangle$ 
                  by (simp add: F-Sync setinterleaves-is-setinterleavesptick) blast
                moreover from  $*(4)$  have  $\langle X \subseteq (X\text{-}P \cup X\text{-}Q) \cap (\text{range tick} \cup \text{ev } 'A) \cup X\text{-}P$ 
 $\cap X\text{-}Q \rangle$ 
                  by (auto simp add: super-ref-Syncptick-def subset-iff split: if-split-asm)
                ultimately show  $\langle (t, X) \in \mathcal{F} (P \llbracket A \rrbracket Q) \rangle$  by (meson is-processT4)
              qed

```

10.3.2 Other Properties

lemma $\langle \text{Sync}_{Lists}.\text{Sync}_{ptick}\text{-locale-sym}.\text{Sync}_{ptick} \text{ lenL lenR } Q \text{ } A \text{ } P = P \text{ lenL } \llbracket A \rrbracket_{\checkmark \text{ lenR}} Q \rangle$
by (*simp add: Sync_{Lists}.Sync_{ptick}-locale-sym.Sync_{ptick}-sym*)

corollary *TickSwap-Sync_{Pair}* [simp] : $\langle \text{TickSwap } (P \llbracket S \rrbracket_{\checkmark \text{ Pair}} Q) = Q \llbracket S \rrbracket_{\checkmark \text{ Pair}} P \rangle$
by (*simp add: Sync_{Pair}.Sync_{ptick}-commute TickSwap-is-Renaming*)

lemma *TickSwap-is-Sync_{Pair}-iff* [simp] :
 $\langle \text{TickSwap } P = Q \llbracket S \rrbracket_{\checkmark \text{ Pair}} R \longleftrightarrow P = R \llbracket S \rrbracket_{\checkmark \text{ Pair}} Q \rangle$
by (*simp add: TickSwap-eq-iff-eq-TickSwap*)

corollary *Sync_{Classic}-commute* : $\langle P \llbracket S \rrbracket_{\checkmark \text{ Classic}} Q = Q \llbracket S \rrbracket_{\checkmark \text{ Classic}} P \rangle$
by (*fact Sync_{Classic}.Sync_{ptick}-commute[simplified]*)

lemma $\langle \text{RenamingTick } (P \text{ lenL } \llbracket S \rrbracket_{\checkmark \text{ lenR}} Q) (\lambda r.s. \text{ drop lenL } r\text{-s } @ \text{ take lenL } r\text{-s})$
 $=$
 $Q \text{ lenR } \llbracket S \rrbracket_{\checkmark \text{ lenL}} P \rangle$
by (*fact Sync_{Lists}.Sync_{ptick}-commute*)

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* ::
 $\langle \text{nat} \Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \Rightarrow \text{bool} \rangle (\langle \text{length}_{\checkmark} \rangle \cdot ('(-)))$
where $\langle \text{length}_{\checkmark n}(P) \equiv \forall rs \in \checkmark s(P). \text{ length } rs = n \rangle$

We might imagine $\forall rs \in \checkmark s(P). \text{ length } rs = n$ 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* : $\langle (\bigwedge rs. rs \in \checkmark s(P) \Longrightarrow \text{ length } rs = n) \Longrightarrow \text{ length}_{\checkmark n}(P) \rangle$
by (*simp add: is-ticks-length-def*)

lemma *is-ticks-lengthD* : $\langle \text{ length}_{\checkmark n}(P) \Longrightarrow rs \in \checkmark s(P) \Longrightarrow \text{ length } rs = n \rangle$
by (*simp add: is-ticks-length-def*)

lemma *is-ticks-length-unique* :
 — Not suitable for simplifier.
 $\langle \text{length}_{\checkmark n}(P) \longleftrightarrow \checkmark s(P) = \{\} \vee (\forall m. \text{length}_{\checkmark m}(P) \longleftrightarrow m = n) \rangle$
by (*auto simp add: is-ticks-length-def*)

lemma *empty-strict-ticks-of-imp-is-ticks-length* :
 $\langle \checkmark s(P) = \{\} \implies \text{length}_{\checkmark n}(P) \rangle$
using *is-ticks-length-unique* **by** *blast*

lemma *nonempty-strict-ticks-of-imp-is-ticks-length-unique* :
 $\langle \checkmark s(P) \neq \{\} \implies \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark m}(P) \implies m = n \rangle$
using *is-ticks-length-unique* **by** *blast*

Behaviour

named-theorems *is-ticks-length-simp*
named-theorems *is-ticks-length-intro*

Constant Processes **lemma** *is-ticks-length-STOP* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(\text{STOP}) \rangle$ **by** (*simp add: empty-strict-ticks-of-imp-is-ticks-length*)

lemma *is-ticks-length-BOT* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(\perp) \rangle$ **by** (*simp add: empty-strict-ticks-of-imp-is-ticks-length*)

lemma *is-ticks-length-SKIP-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(\text{SKIP } rs) \longleftrightarrow \text{length } rs = n \rangle$
by (*simp add: is-ticks-length-def*)

lemma *is-ticks-length-SKIPS-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(\text{SKIPS } R) \longleftrightarrow (\forall rs \in R. \text{length } rs = n) \rangle$
by (*simp add: is-ticks-length-def strict-ticks-of-def SKIPS-projs*)

Binary (or less) Operators **lemma** *is-ticks-length-Ndet* [*is-ticks-length-intro*] :

$\langle \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P \sqcap Q) \rangle$
by (*simp add: is-ticks-length-def*)
 (*meson Un-iff strict-ticks-of-Ndet-subset subset-iff*)

lemma *is-ticks-length-Det* [*is-ticks-length-intro*] :
 $\langle \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P \sqcap Q) \rangle$
by (*simp add: is-ticks-length-def*)
 (*meson Un-iff strict-ticks-of-Det-subset subset-iff*)

lemma *is-ticks-length-Sliding* [*is-ticks-length-intro*] :
 $\langle \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P \triangleright Q) \rangle$
by (*simp add: is-ticks-length-def*)
 (*meson Un-iff strict-ticks-of-Sliding-subset subset-iff*)

lemma *is-ticks-length-Sync* [*is-ticks-length-intro*] :
 $\langle \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P \llbracket S \rrbracket Q) \rangle$
by (*simp add: is-ticks-length-def*)
(*meson Int-iff strict-ticks-of-Sync-subset subset-iff*)

lemma *is-ticks-length-Seq* [*is-ticks-length-intro*] :
 $\langle \text{non-terminating } P \vee \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P ; Q) \rangle$
proof (*elim disjE*)
show $\langle \text{non-terminating } P \implies \text{length}_{\checkmark n}(P ; Q) \rangle$
by (*metis is-ticks-length-def non-terminating-Seq non-terminating-is-right*
non-tickFree-tick strict-ticks-of-memE tickFree-append-iff)
next
from *strict-ticks-of-Seq-subset*[*of P Q*]
show $\langle \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P ; Q) \rangle$
by (*auto simp add: is-ticks-length-def split: if-split-asm*)
qed

lemma *is-ticks-length-Hiding* [*is-ticks-length-intro*] :
 $\langle \text{length}_{\checkmark n}(P \setminus S) \rangle \text{ if } \langle \text{length}_{\checkmark n}(P) \rangle$
proof (*rule is-ticks-lengthI*)
fix *rs* **assume** $\langle \text{rs} \in \checkmark \text{s}(P \setminus S) \rangle$
then obtain *t t'* **where** $\langle t = t' @ [\checkmark(rs)] \rangle \langle t \in \mathcal{T}(P \setminus S) \rangle \langle t \notin \mathcal{D}(P \setminus S) \rangle$
by (*metis is-processT9 strict-ticks-of-memE*)
from *this*(2, 3) **obtain** *u* **where** $\langle t = \text{trace-hide } u \text{ (ev ' } S) \rangle \langle u \in \mathcal{T} P \rangle$
unfolding *T-Hiding D-Hiding using F-T* **by** *fast*
from *this*(1) *this*(2)[*THEN T-imp-front-tickFree*] **obtain** *u'* **where** $\langle u = u' @ [\checkmark(rs)] \rangle$
by (*cases u rule: rev-cases, simp-all add: \langle t = t' @ [\checkmark(rs)] \rangle split: if-split-asm*)
(*metis Hiding-tickFree front-tickFree-nonempty-append-imp list.distinct(1)*
non-tickFree-tick tickFree-append-iff)
from $\langle t \notin \mathcal{D}(P \setminus S) \rangle$ *mem-D-imp-mem-D-Hiding*[*of u P S*]
have $\langle u \notin \mathcal{D} P \rangle$ **unfolding** $\langle t = \text{trace-hide } u \text{ (ev ' } S) \rangle$ **by** *blast*
with $\langle u \in \mathcal{T} P \rangle \langle u = u' @ [\checkmark(rs)] \rangle$ **have** $\langle \text{rs} \in \checkmark \text{s}(P) \rangle$
by (*simp add: strict-ticks-of-memI*)
with that show $\langle \text{length } \text{rs} = n \rangle$ **by** (*simp add: is-ticks-lengthD*)
qed

lemma *is-ticks-length-Interrupt* [*is-ticks-length-intro*] :
 $\langle \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark n}(Q) \implies \text{length}_{\checkmark n}(P \triangle Q) \rangle$
by (*simp add: is-ticks-length-def*)
(*meson Un-iff strict-ticks-of-Interrupt-subset subsetD*)

— Missing lemma from HOL-CSPM

lemma *strict-ticks-Throw-subset* :
 $\langle \checkmark \text{s}(P \ominus a \in A. Q a) \subseteq \checkmark \text{s}(P) \cup (\bigcup a \in A \cap \alpha(P). \checkmark \text{s}(Q a)) \rangle$

proof (*rule subsetI*)
fix r **assume** $\langle r \in \check{\mathbf{s}}(P \Theta a \in A. Q a) \rangle$
then obtain t **where** $\langle t @ [\check{\mathbf{v}}(r)] \in \mathcal{T} (P \Theta a \in A. Q a) \rangle \langle t @ [\check{\mathbf{v}}(r)] \notin \mathcal{D} (P \Theta a \in A. Q a) \rangle$
by (*meson is-processT9 strict-ticks-of-memE*)
then consider $\langle t @ [\check{\mathbf{v}}(r)] \in \mathcal{T} P \rangle \langle t @ [\check{\mathbf{v}}(r)] \notin \mathcal{D} P \rangle$
 $| t1 \ a \ t2$ **where** $\langle t @ [\check{\mathbf{v}}(r)] = t1 @ ev \ a \ \# \ t2 \rangle \langle t1 @ [ev \ a] \in \mathcal{T} P \rangle$
 $\langle a \in A \rangle \langle t2 \in \mathcal{T} (Q a) \rangle \langle t2 \notin \mathcal{D} (Q a) \rangle$
by (*simp add: Throw-projs*)
(metis (no-types, lifting) append-T-imp-tickFree
front-tickFree-single is-processT9 not-Cons-self2)
thus $\langle r \in \check{\mathbf{s}}(P) \cup (\bigcup a \in A \cap \alpha(P). \check{\mathbf{s}}(Q a)) \rangle$
proof cases
show $\langle t @ [\check{\mathbf{v}}(r)] \in \mathcal{T} P \implies t @ [\check{\mathbf{v}}(r)] \notin \mathcal{D} P \implies r \in \check{\mathbf{s}}(P) \cup (\bigcup a \in A \cap \alpha(P). \check{\mathbf{s}}(Q a)) \rangle$
by (*simp add: strict-ticks-of-memI*)
next
show $\langle [t @ [\check{\mathbf{v}}(r)] = t1 @ ev \ a \ \# \ t2; t1 @ [ev \ a] \in \mathcal{T} P; a \in A; t2 \in \mathcal{T} (Q a); t2 \notin \mathcal{D} (Q a)] \implies r \in \check{\mathbf{s}}(P) \cup (\bigcup a \in A \cap \alpha(P). \check{\mathbf{s}}(Q a)) \rangle$ **for** $t1 \ a \ t2$
by (*cases t2 rule: rev-cases, simp-all*)
(meson IntI events-of-memI in-set-conv-decomp strict-ticks-of-memI)
qed
qed

lemma *is-ticks-length-Throw [is-ticks-length-intro]* :
 $\langle length_{\check{\mathbf{n}}}(P \Theta a \in A. Q a) \rangle$
if $\langle length_{\check{\mathbf{n}}}(P) \rangle \langle \bigwedge a. a \in \alpha(P) \implies length_{\check{\mathbf{n}}}(Q a) \rangle$
proof –
from *that have* $\langle \forall rs \in \check{\mathbf{s}}(P) \cup (\bigcup a \in A \cap \alpha(P). \check{\mathbf{s}}(Q a)). length \ rs = n \rangle$
by (*auto simp add: is-ticks-length-def*)
with *strict-ticks-Throw-subset* **show** $\langle length_{\check{\mathbf{n}}}(P \Theta a \in A. Q a) \rangle$
unfolding *is-ticks-length-def* **by** *fast*
qed

lemma *is-ticks-length-Renaming [is-ticks-length-intro]* :
 $\langle length_{\check{\mathbf{n}}}(\text{Renaming } P \ f \ g) \rangle$ **if** $\langle \bigwedge r. r \in \check{\mathbf{s}}(P) \implies length \ (g \ r) = n \rangle$
proof (*rule is-ticks-lengthI*)
fix rs **assume** $\langle rs \in \check{\mathbf{s}}(\text{Renaming } P \ f \ g) \rangle$
then obtain t **where** $\langle t @ [\check{\mathbf{v}}(rs)] \in \mathcal{T} (\text{Renaming } P \ f \ g) \rangle$
 $\langle t @ [\check{\mathbf{v}}(rs)] \notin \mathcal{D} (\text{Renaming } P \ f \ g) \rangle$
by (*meson is-processT9 strict-ticks-of-memE*)
then obtain u **where** $\langle * : \langle t @ [\check{\mathbf{v}}(rs)] = map \ (map-event_{ptick} \ f \ g) \ u \rangle$ **and** $\langle u \in \mathcal{T} P \rangle$
by (*auto simp add: Renaming-projs*)
from *this(1)* $\langle u \in \mathcal{T} P \rangle$ *append-T-imp-tickFree* **obtain** $u' \ r$
where $\langle rs = g \ r \rangle \langle u = u' @ [\check{\mathbf{v}}(r)] \rangle \langle tF \ u' \rangle$
by (*cases u rule: rev-cases*) (*auto simp add: tick-eq-map-event_{ptick}-iff*)
from $\langle * : \langle t @ [\check{\mathbf{v}}(rs)] \notin \mathcal{D} (\text{Renaming } P \ f \ g) \rangle$ *this(2, 3)* *front-tickFree-Cons-iff*

have $\langle u' \notin \mathcal{D} P \rangle$ **by** (*auto simp add: D-Renaming*)
moreover from $\langle u \in \mathcal{T} P \rangle$ **have** $\langle u' @ [\checkmark(r)] \in \mathcal{T} P \rangle$
by (*simp add: $\langle u = u' @ [\checkmark(r)] \rangle$*)
ultimately have $\langle r \in \checkmark s(P) \rangle$ **by** (*meson is-processT9 strict-ticks-of-memI*)
with that have $\langle \text{length } (g \ r) = n \rangle$ **by** *blast*
thus $\langle \text{length } rs = n \rangle$ **by** (*simp add: $\langle rs = g \ r \rangle$*)
qed

Architectural Operators **lemma** *is-ticks-length-GlobalNdet* [*is-ticks-length-intro*]

$\langle (\bigwedge a. a \in A \implies \text{length}_{\checkmark n}(P \ a)) \implies \text{length}_{\checkmark n}(\bigwedge a \in A. P \ a) \rangle$
by (*simp add: is-ticks-length-def*)
(metis (no-types, lifting) UN-E strict-ticks-of-GlobalNdet-subset subsetD)

lemma *is-ticks-length-GlobalDet* [*is-ticks-length-intro*] :
 $\langle (\bigwedge a. a \in A \implies \text{length}_{\checkmark n}(P \ a)) \implies \text{length}_{\checkmark n}(\bigwedge a \in A. P \ a) \rangle$
by (*simp add: is-ticks-length-def*)
(metis (no-types, lifting) UN-E strict-ticks-of-GlobalDet-subset subsetD)

lemma *is-ticks-length-MultiSync* [*is-ticks-length-intro*] :
 $\langle (\bigwedge m. m \in \text{set-mset } M \implies \text{length}_{\checkmark n}(P \ m)) \implies \text{length}_{\checkmark n}(\llbracket S \rrbracket \ m \in \# \ M. P \ m) \rangle$
by (*induct M rule: induct-subset-mset-empty-single*)
(simp-all add: is-ticks-length-STOP is-ticks-length-Sync)

lemma *is-ticks-length-MultiSeq* [*is-ticks-length-intro*] :
 $\langle L \neq [] \implies \text{length}_{\checkmark n}(P \ (\text{last } L)) \implies \text{length}_{\checkmark n}(SEQ \ l \in @ \ L. P \ l) \rangle$
by (*induct L rule: rev-induct*)
(simp-all add: is-ticks-length-Seq)

Communications **lemma** *is-ticks-length-write0-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(e \rightarrow P) \longleftrightarrow \text{length}_{\checkmark n}(P) \rangle$
by (*simp add: is-ticks-length-def strict-ticks-of-write0*)

lemma *is-ticks-length-write-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(c!e \rightarrow P) \longleftrightarrow \text{length}_{\checkmark n}(P) \rangle$
by (*simp add: is-ticks-length-def strict-ticks-of-write*)

lemma *is-ticks-length-Mprefix-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(\Box a \in A \rightarrow P \ a) = (\forall a \in A. \text{length}_{\checkmark n}(P \ a)) \rangle$
by (*auto simp add: is-ticks-length-def strict-ticks-of-Mprefix*)

lemma *is-ticks-length-read-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(c?a \in A \rightarrow P \ a) = (\forall b \in c \text{ ' } A. \text{length}_{\checkmark n}(P \ (\text{inv-into } A \ c \ b))) \rangle$
by (*simp add: read-def is-ticks-length-Mprefix-iff*)

corollary $\langle \text{inj-on } c \ A \implies \text{length}_{\checkmark n}(c?a \in A \rightarrow P \ a) = (\forall a \in A. \text{length}_{\checkmark n}(P \ a)) \rangle$
by (*simp add: is-ticks-length-read-iff*)

lemma *is-ticks-length-Mndetprefix-iff* [*is-ticks-length-simp*] :

$\langle \text{length}_{\checkmark n}(\sqcap a \in A \rightarrow P a) = (\forall a \in A. \text{length}_{\checkmark n}(P a)) \rangle$
by (*auto simp add: is-ticks-length-def strict-ticks-of-Mndetprefix*)

lemma *is-ticks-length-ndet-write-iff* [*is-ticks-length-simp*] :
 $\langle \text{length}_{\checkmark n}(c!!a \in A \rightarrow P a) = (\forall b \in c. \text{length}_{\checkmark n}(P(\text{inv-into } A \ c \ b))) \rangle$
by (*simp add: ndet-write-def is-ticks-length-Mndetprefix-iff*)

corollary $\langle \text{inj-on } c \ A \implies \text{length}_{\checkmark n}(c!!a \in A \rightarrow P a) = (\forall a \in A. \text{length}_{\checkmark n}(P a)) \rangle$
by (*simp add: is-ticks-length-ndet-write-iff*)

Generalizations lemma *strict-ticks-of-Seq_{ptick}-subset* : $\langle \checkmark s(P ;_{\checkmark} Q) \subseteq \bigcup \{ \checkmark s(Q \ r) \mid r. r \in \checkmark s(P) \} \rangle$
by (*auto simp add: Seq_{ptick}-projs append-eq-map-conv elim!: strict-ticks-of-memE*)
(*metis tickFree-Nil non-tickFree-tick tickFree-map-ev-comp*
front-tickFree-charn tickFree-append-iff tickFree-append-iff
last-snoc[of map (ev o of-ev) - @ -] last-snoc[of - <checkmark(-)>]
butlast-snoc[of map (ev o of-ev) - @ -] butlast-snoc[of - <checkmark(-)>]
append.assoc[of map (ev o of-ev) - - [-]] tickFree-imp-front-tickFree
T-imp-front-tickFree is-processT9 strict-ticks-of-memI,
metis butlast-append butlast-snoc front-tickFree-iff-tickFree-butlast non-tickFree-tick
tickFree-append-iff tickFree-imp-front-tickFree tickFree-map-ev-comp)

lemma *non-terminating-Seq_{ptick}* :
 $\langle P ;_{\checkmark} Q = \text{RenamingTick } P \ g \rangle$ **if** $\langle \text{non-terminating } P \rangle$
proof –
from $\langle \text{non-terminating } P \rangle$ **have** $\mathcal{L} : \langle \mathcal{D} \ P = \{ \} \rangle$ $\langle t @ [\checkmark(r)] \notin \mathcal{T} \ P \rangle$ **for** $t \ r$
by (*force simp add: non-terminating-is-right nonterminating-implies-div-free*) +
show $\langle P ;_{\checkmark} Q = \text{RenamingTick } P \ g \rangle$
proof (*rule Process-eq-optimizedI*)
show $\langle t \in \mathcal{D} \ (P ;_{\checkmark} Q) \implies t \in \mathcal{D} \ (\text{RenamingTick } P \ g) \rangle$
and $\langle t \in \mathcal{D} \ (\text{RenamingTick } P \ g) \implies t \in \mathcal{D} \ (P ;_{\checkmark} Q) \rangle$ **for** t
by (*simp-all add: Seq_{ptick}-projs Renaming-projs $\mathcal{L}$*)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ (P ;_{\checkmark} Q) \rangle$
then obtain t' **where** $*$: $\langle t = \text{map} \ (ev \circ of-ev) \ t' \rangle$
 $\langle (t', \text{ref-Seq}_{ptick} \ X) \in \mathcal{F} \ P \rangle$ $\langle tF \ t' \rangle$
by (*auto simp add: Seq_{ptick}-projs Renaming-projs $\mathcal{L}$*)
have $\$$: $\langle t = \text{map} \ (\text{map-event}_{ptick} \ id \ g) \ t' \rangle$
by (*simp add: *(1, 3) tickFree-map-map-event_{ptick}-is*)
have $\$ \$$: $\langle \text{map-event}_{ptick} \ id \ g - ' X \subseteq \text{ref-Seq}_{ptick} \ X \rangle$
by (*simp add: subset-iff ref-Seq_{ptick}-def image-iff*)
(*metis Int-iff event_{ptick}.exhaust event_{ptick}.sel(1) event_{ptick}.sims(9) id-apply*
rangeI)
show $\langle (t, X) \in \mathcal{F} \ (\text{RenamingTick } P \ g) \rangle$
by (*simp add: Renaming-projs*) (*metis \$ \\$ \$ *(2) is-processT4*)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ (\text{RenamingTick } P \ g) \rangle$
then obtain t' **where** $*$: $\langle t = \text{map} \ (\text{map-event}_{ptick} \ id \ g) \ t' \rangle$

$\langle (t', \text{map-event}_{ptick} \text{ id } g - 'X) \in \mathcal{F} P \rangle$
by (*auto simp add: Renaming-projs* $\mathcal{L}$)
from $*(2)$ *F-T non-terminating-is-right* $\langle \text{non-terminating } P \rangle$ **have** $\langle tF t' \rangle$ **by**
blast
have $\langle (t', \text{map-event}_{ptick} \text{ id } g - 'X \cup \text{range tick}) \in \mathcal{F} P \rangle$
by (*rule is-processT5[OF *(2)]*) (*use* $\mathcal{L}(2)$ *F-T in blast*)
moreover have $\langle \text{ref-Seq}_{ptick} X \subseteq \text{map-event}_{ptick} \text{ id } g - 'X \cup \text{range tick} \rangle$
by (*auto simp add: ref-Seq_{ptick}-def*)
ultimately have $\langle (t', \text{ref-Seq}_{ptick} X) \in \mathcal{F} P \rangle$
by (*metis is-processT4*)
moreover have $\langle t = \text{map } (ev \circ \text{of-ev}) t' \rangle$
by (*simp add: *(1) $\langle tF t' \rangle$ tickFree-map-map-event_{ptick}-is*)
ultimately show $\langle (t, X) \in \mathcal{F} (P ;_{\checkmark} Q) \rangle$
by (*auto simp add: Seq_{ptick}-projs $\langle tF t' \rangle$*)
qed
qed

lemma *is-ticks-length-Seq_{ptick} [is-ticks-length-intro]* :
 $\langle \text{non-terminating } P \vee (\forall r \in \check{\mathbf{s}}(P). \text{length}_{\check{\mathbf{n}}}(Q r)) \implies \text{length}_{\check{\mathbf{n}}}(P ;_{\checkmark} Q) \rangle$
proof (*elim disjE*)
assume $\langle \text{non-terminating } P \rangle$
hence $\langle \check{\mathbf{s}}(P) = \{\} \rangle$
by (*metis (full-types) non-terminating-Seq strict-ticks-of-BOT*
strict-ticks-of-Seq-subset subset-empty)
show $\langle \text{non-terminating } P \implies \text{length}_{\check{\mathbf{n}}}(P ;_{\checkmark} Q) \rangle$
by (*subst non-terminating-Seq_{ptick}, assumption*)
(rule is-ticks-length-Renaming, simp add: is-ticks-length-Renaming $\langle \check{\mathbf{s}}(P) = \{\} \rangle$)
next
from *strict-ticks-of-Seq_{ptick}-subset[of P Q]*
show $\langle \forall r \in \check{\mathbf{s}}(P). \text{length}_{\check{\mathbf{n}}}(Q r) \implies \text{length}_{\check{\mathbf{n}}}(P ;_{\checkmark} Q) \rangle$
by (*auto simp add: is-ticks-length-def*)
qed

lemma *is-ticks-length-Sync_{ptick}* :
 $\langle \text{length}_{\check{\mathbf{n}}}(\text{Sync}_{ptick}\text{-locale}.\text{Sync}_{ptick} \text{ tick-join } P A Q) \rangle$
— We cannot work directly inside the locale since in this context the types of ticks *t* cannot be set to *r* list.
if $\langle \text{Sync}_{ptick}\text{-locale tick-join} \rangle$
and $\langle \bigwedge r s. r \in \check{\mathbf{s}}(P) \implies s \in \check{\mathbf{s}}(Q) \implies$
case tick-join r s of $\Diamond \Rightarrow \text{True} \mid [r-s] \Rightarrow \text{length } r-s = n$
proof —
interpret *Sync_{ptick}-locale tick-join*
by (*fact $\langle \text{Sync}_{ptick}\text{-locale tick-join} \rangle$*)
show $\langle \text{length}_{\check{\mathbf{n}}}(P \llbracket A \rrbracket_{\checkmark} Q) \rangle$

proof (*rule is-ticks-lengthI*)
fix rs **assume** $\langle rs \in \check{s}(P \llbracket A \rrbracket_{\check{Q}}) \rangle$
then obtain t **where** $\langle t @ [\check{r}(rs)] \in \mathcal{T}(P \llbracket A \rrbracket_{\check{Q}}) \rangle \langle t @ [\check{r}(rs)] \notin \mathcal{D}(P \llbracket A \rrbracket_{\check{Q}}) \rangle$
by (*meson is-processT9 strict-ticks-of-memE*)
then obtain $t-P$ $t-Q$ **where** $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$
and $*$: $\langle t @ [\check{r}(rs)] \text{ setinterleaves}_{\check{r} \text{ tick-join}} ((t-P, t-Q), A) \rangle$
unfolding $\text{Sync}_{\text{ptick-projs}}$ **by** *blast*
with $\langle t @ [\check{r}(rs)] \notin \mathcal{D}(P \llbracket A \rrbracket_{\check{Q}}) \rangle$ **have** $\langle t-P \notin \mathcal{D} P \rangle \langle t-Q \notin \mathcal{D} Q \rangle$
by (*simp add: D-Sync_{ptick}'*, *use front-tickFree-Nil in blast*) +
from $*$ **obtain** r s $t-P'$ $t-Q'$
where $\langle \text{tick-join } r \ s = \lfloor rs \rfloor \rangle \langle t-P = t-P' @ [\check{r}(r)] \rangle \langle t-Q = t-Q' @ [\check{r}(s)] \rangle$
by (*blast elim: snoc-tick-setinterleaves_{ptick}E*)
from $\text{this}(2, 3)$ $\langle t-P \notin \mathcal{D} P \rangle \langle t-Q \notin \mathcal{D} Q \rangle \langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$
have $\langle r \in \check{s}(P) \rangle \langle s \in \check{s}(Q) \rangle$ **by** (*metis strict-ticks-of-memI*) +
from $\text{that}(2)[\text{OF this, unfolded } \langle \text{tick-join } r \ s = \lfloor rs \rfloor \rangle]$ **show** $\langle \text{length } rs = n \rangle$ **by**
simp
qed
qed

lemma *is-ticks-length-One-RenamingTick-singl* [*is-ticks-length-simp*] :

$\langle \text{length}_{\check{r} \text{ Suc } 0}(\text{RenamingTick } P \ (\lambda r. [r])) \rangle$

by (*simp add: is-ticks-length-Renaming*)

lemma *is-ticks-length-Two-Sync_{Pairlist}* [*is-ticks-length-simp*] :

$\langle \text{length}_{\check{r} \text{ Suc } 0}(P \llbracket S \rrbracket_{\check{r} \text{ Pairlist}} Q) \rangle$

by (*simp add: is-ticks-length-Sync_{ptick}[OF Sync_{Pairlist}.Sync_{ptick}-locale-axioms]*)

lemma *is-ticks-length-Suc-Sync_{Rlist}* [*is-ticks-length-intro*] :

$\langle \text{length}_{\check{r} n}(Q) \implies \text{length}_{\check{r} \text{ Suc } n}(P \llbracket S \rrbracket_{\check{r} \text{ Rlist}} Q) \rangle$

by (*rule is-ticks-length-Sync_{ptick}[OF Sync_{Rlist}.Sync_{ptick}-locale-axioms]*)
(simp add: is-ticks-lengthD)

The equivalence is false.

lemma *False if* $\langle \bigwedge P \ Q \ n. \text{length}_{\check{r} \text{ Suc } n}(P \llbracket S \rrbracket_{\check{r} \text{ Rlist}} Q) \implies \text{length}_{\check{r} n}(Q) \rangle$

using *that[of 0 STOP <SKIP [undefined]>]*

by (*simp add: is-ticks-length-STOP is-ticks-length-SKIP-iff*)

lemma *is-ticks-length-Suc-Sync_{Llist}* [*is-ticks-length-intro*] :

$\langle \text{length}_{\check{r} n}(P) \implies \text{length}_{\check{r} \text{ Suc } n}(P \llbracket S \rrbracket_{\check{r} \text{ Llist}} Q) \rangle$

by (*rule is-ticks-length-Sync_{ptick}*
[OF Sync_{Rlist}.Sync_{ptick}-comm-locale-sym.Sync_{ptick}-locale-axioms])
(simp add: is-ticks-lengthD)

lemma *is-ticks-length-sum-Sync_{ListLenL}* [*is-ticks-length-intro*] :

$\langle \text{length}_{\checkmark}^m(Q) \implies \text{length}_{\checkmark}^n + m(P \ n \llbracket S \rrbracket_{\checkmark}^{\text{ListLenL}} Q) \rangle$
by (rule *is-ticks-length-Sync_{ptick}* [OF *Sync_{ListLenL}.Sync_{ptick}-locale-axioms*])
 (simp add: *is-ticks-lengthD*)

lemma *is-ticks-length-sum-Sync_{ListLenR} [is-ticks-length-intro]* :
 $\langle \text{length}_{\checkmark}^n(P) \implies \text{length}_{\checkmark}^n + m(P \ m \llbracket S \rrbracket_{\checkmark}^{\text{ListLenR}} Q) \rangle$
by (rule *is-ticks-length-Sync_{ptick}*
 [OF *Sync_{ListLenL}.Sync_{ptick}-comm-locale-sym.Sync_{ptick}-locale-axioms*])
 (simp add: *is-ticks-lengthD*)

lemma *is-ticks-length-sum-Sync_{Lists} [is-ticks-length-intro]* :
 $\langle \text{length}_{\checkmark}^n + m(P \ n \llbracket S \rrbracket_{\checkmark}^m Q) \rangle$
by (rule *is-ticks-length-Sync_{ptick}* [OF *Sync_{Lists}.Sync_{ptick}-locale-axioms*]) simp

10.4.2 Conversions

lemma *Sync_{PairList}-to-Sync_{RList}* :
 $\langle P \llbracket S \rrbracket_{\checkmark}^{\text{PairList}} Q = P \llbracket S \rrbracket_{\checkmark}^{\text{RList}} \text{RenamingTick } Q \ (\lambda s. [s]) \rangle$
by (rule *Sync_{RList}.inj-RenamingTick-Sync_{ptick}-inj-RenamingTick*
 [of *id* $\langle \lambda s. [s] \rangle$, *simplified*, *symmetric*])
 (auto intro: *inj-onI*)

lemma *Sync_{PairList}-to-Sync_{LList}* :
 $\langle P \llbracket S \rrbracket_{\checkmark}^{\text{PairList}} Q = \text{RenamingTick } P \ (\lambda r. [r]) \llbracket S \rrbracket_{\checkmark}^{\text{LList}} Q \rangle$
by (rule *Sync_{RList}.Sync_{ptick}-comm-locale-sym.inj-RenamingTick-Sync_{ptick}-inj-RenamingTick*
 [of $\langle \lambda r. [r] \rangle$ *id*, *simplified*, *symmetric*])
 (auto intro: *inj-onI*)

lemma *Sync_{RList}-to-Sync_{ListLenL}* :
 $\langle P \llbracket S \rrbracket_{\checkmark}^{\text{RList}} Q = \text{RenamingTick } P \ (\lambda r. [r]) \text{Suc } 0 \llbracket S \rrbracket_{\checkmark}^{\text{ListLenL}} Q \rangle$
by (rule *Sync_{ListLenL}.inj-RenamingTick-Sync_{ptick}-inj-RenamingTick*
 [of $\langle \lambda r. [r] \rangle$ *id* $\langle \text{Suc } 0 \rangle$, *simplified*, *symmetric*])
 (auto intro: *inj-onI*)

lemma *Sync_{LList}-to-Sync_{ListLenR}* :
 $\langle P \llbracket S \rrbracket_{\checkmark}^{\text{LList}} Q = P \ \text{Suc } 0 \llbracket S \rrbracket_{\checkmark}^{\text{ListLenR}} \text{RenamingTick } Q \ (\lambda s. [s]) \rangle$
by (rule *Sync_{ListLenL}.Sync_{ptick}-comm-locale-sym.inj-RenamingTick-Sync_{ptick}-inj-RenamingTick*
 [of *id* $\langle \lambda s. [s] \rangle$ $\langle \text{Suc } 0 \rangle$, *simplified*, *symmetric*])
 (auto intro: *inj-onI*)

lemma *Sync_{ListLenL}-to-Sync_{Lists}* :
 $\langle \text{length}_{\checkmark}^m(Q) \implies P \ n \llbracket S \rrbracket_{\checkmark}^{\text{ListLenL}} Q = P \ n \llbracket S \rrbracket_{\checkmark}^m Q \rangle$
by (auto intro!: *Sync_{Lists}.Sync_{ptick}-same-tick-join-on-strict-ticks-of*
Sync_{ListLenL}.Sync_{ptick}-locale-axioms
dest: is-ticks-lengthD)

lemma *Sync_{ListLenR}-to-Sync_{Lists}* :

$\langle \text{length}_{\checkmark n}(P) \implies P \llbracket S \rrbracket_{\checkmark \text{Listslen} R} Q = P \llbracket S \rrbracket_{\checkmark m} Q \rangle$
by (*auto intro!*: *SyncLists.Syncptick-same-tick-join-on-strict-ticks-of*
SyncListslenL.Syncptick-comm-locale-sym.Syncptick-locale-axioms
dest: is-ticks-lengthD)

corollary *SyncListslenL-is-SyncListslenR* :
 $\langle \text{length}_{\checkmark n}(P) \implies \text{length}_{\checkmark m}(Q) \implies P \llbracket S \rrbracket_{\checkmark \text{Listslen} L} Q = P \llbracket S \rrbracket_{\checkmark \text{Listslen} R} Q \rangle$
by (*simp add: SyncListslenL-to-SyncLists SyncListslenR-to-SyncLists*)

corollary *SyncPairlist-to-SyncListslenL* :
 $\langle P \llbracket S \rrbracket_{\checkmark \text{Pairlist}} Q = \text{RenamingTick } P (\lambda r. [r]) \text{ Suc } 0 \llbracket S \rrbracket_{\checkmark \text{Listslen} L} \text{RenamingTick } Q (\lambda s. [s]) \rangle$
by (*simp add: SyncPairlist-to-SyncRlist SyncRlist-to-SyncListslenL*)

corollary *SyncPairlist-to-SyncListslenR* :
 $\langle P \llbracket S \rrbracket_{\checkmark \text{Pairlist}} Q = \text{RenamingTick } P (\lambda r. [r]) \text{ Suc } 0 \llbracket S \rrbracket_{\checkmark \text{Listslen} R} \text{RenamingTick } Q (\lambda s. [s]) \rangle$
by (*simp add: SyncLlist-to-SyncListslenR SyncPairlist-to-SyncLlist*)

corollary *SyncRlist-to-SyncLists* :
 $\langle \text{length}_{\checkmark m}(Q) \implies P \llbracket S \rrbracket_{\checkmark Rlist} Q = \text{RenamingTick } P (\lambda r. [r]) \text{ Suc } 0 \llbracket S \rrbracket_{\checkmark m} Q \rangle$
by (*simp add: SyncListslenL-to-SyncLists SyncRlist-to-SyncListslenL*)

corollary *SyncLlist-to-SyncLists* :
 $\langle \text{length}_{\checkmark n}(P) \implies P \llbracket S \rrbracket_{\checkmark Llist} Q = P \llbracket S \rrbracket_{\checkmark \text{Suc } 0} \text{RenamingTick } Q (\lambda s. [s]) \rangle$
by (*simp add: SyncListslenR-to-SyncLists SyncLlist-to-SyncListslenR*)

corollary *SyncPairlist-to-SyncLists* :
 $\langle P \llbracket S \rrbracket_{\checkmark \text{Pairlist}} Q = \text{RenamingTick } P (\lambda r. [r]) \text{ Suc } 0 \llbracket S \rrbracket_{\checkmark \text{Suc } 0} \text{RenamingTick } Q (\lambda s. [s]) \rangle$
by (*simp add: SyncLlist-to-SyncLists SyncPairlist-to-SyncLlist*
is-ticks-length-One-RenamingTick-singl)

lemma *SyncPair-to-SyncPairlist* :
 $\langle \text{RenamingTick } (P \llbracket S \rrbracket_{\checkmark \text{Pair}} Q) (\lambda (r, s). [r, s]) = P \llbracket S \rrbracket_{\checkmark \text{Pairlist}} Q \rangle$
by (*rule SyncPair.inj-on-RenamingTick-Syncptick*
[of $\langle \lambda (r, s). [r, s] \rangle$, simplified]
(auto intro: inj-onI))

lemma *SyncPairlist-to-SyncPair* :
 $\langle \text{RenamingTick } (P \llbracket S \rrbracket_{\checkmark \text{Pairlist}} Q) (\lambda rs. (rs ! 0, rs ! \text{Suc } 0)) = P \llbracket S \rrbracket_{\checkmark \text{Pair}} Q \rangle$
by (*rule SyncPairlist.inj-on-RenamingTick-Syncptick*
[of $\langle \lambda rs. (rs ! 0, rs ! \text{Suc } 0) \rangle$, simplified]
(auto intro: inj-onI))

lemma *Sync_{Pair}-to-Sync_{List}* :

⟨*RenamingTick* ($P \llbracket S \rrbracket_{\checkmark Pair} Q$) ($\lambda(r, s). r \# s$) = $P \llbracket S \rrbracket_{\checkmark Rlist} Q$ ⟩
by (*rule Sync_{Pair}.inj-on-RenamingTick-Sync_{ptick}*
 [of ⟨ $\lambda(r, s). r \# s$ ⟩, *simplified*])
 (*auto intro: inj-onI*)

lemma *Sync_{Pair}-to-Sync_{List}* :

⟨*RenamingTick* ($P \llbracket S \rrbracket_{\checkmark Pair} Q$) ($\lambda(r, s). r @ [s]$) = $P \llbracket S \rrbracket_{\checkmark Llist} Q$ ⟩
by (*rule Sync_{Pair}.inj-on-RenamingTick-Sync_{ptick}*
 [of ⟨ $\lambda(r, s). r @ [s]$ ⟩, *simplified*])
 (*auto intro: inj-onI*)

lemma *Sync_{Pair}-to-Sync_{ListslenL}* :

⟨*RenamingTick* ($P \llbracket S \rrbracket_{\checkmark Pair} Q$) ($\lambda(r, s). r @ s$) = $P \llbracket S \rrbracket_{\checkmark ListslenL} Q$ ⟩
 (**is** ⟨ $?lhs = ?rhs$ ⟩) **if** ⟨*length_n*(P)⟩

proof –

let $?g = \langle \lambda rs. (take\ n\ rs, drop\ n\ rs) \rangle$

let $?g' = \langle \lambda(r, s). r @ s \rangle$

let $?RT = RenamingTick$

have ⟨ $?RT\ ?lhs\ ?g = ?RT\ ?rhs\ ?g$ ⟩

proof (*subst Sync_{ListslenL}.inj-on-RenamingTick-Sync_{ptick}*)

show ⟨*inj-on* $?g$ (*Sync_{ListslenL}.range-tick-join* n)⟩

by (*rule inj-onI*) (*auto split: if-split-asm*)

next

have ⟨ $?RT\ (?RT\ (P \llbracket S \rrbracket_{\checkmark Pair} Q)\ ?g')\ ?g = P \llbracket S \rrbracket_{\checkmark Pair} Q$ ⟩

proof (*fold RenamingTick-comp*, *subst* (2) *RenamingTick-id*[of ⟨ $P \llbracket S \rrbracket_{\checkmark Pair} Q$ ⟩, *symmetric*])

show ⟨ $?RT\ (P \llbracket S \rrbracket_{\checkmark Pair} Q)\ (?g \circ ?g') = ?RT\ (P \llbracket S \rrbracket_{\checkmark Pair} Q)\ id$ ⟩

proof (*rule RenamingTick-is-restrictable-on-strict-ticks-of*)

from that show ⟨ $rs \in \checkmark s(P \llbracket S \rrbracket_{\checkmark Pair} Q) \implies (?g \circ (\lambda(x, y). x @ y))\ rs = id\ rs$ ⟩ **for** rs

by (*auto dest!: set-mp[OF Sync_{Pair}.strict-ticks-of-Sync_{ptick}-subset]*
is-ticks-lengthD)

qed

qed

also have ⟨ $P \llbracket S \rrbracket_{\checkmark Pair} Q =$

Sync_{ptick}-locale.Sync_{ptick}

($\lambda r\ s. case\ if\ length\ r = n\ then\ [r @ s]\ else\ \Diamond$

of $\Diamond \Rightarrow \Diamond \mid [rs] \Rightarrow [?g\ rs])\ P\ S\ Q$ ⟩ (**is** ⟨ $- = ?rhs'$ ⟩)

by (*rule Sync_{Pair}.Sync_{ptick}-same-tick-join-on-strict-ticks-of[symmetric]*, *unfold-locales*)

(*use* ⟨*length_n*(P)⟩ **in** *auto split: if-split-asm dest: is-ticks-lengthD*)

finally show ⟨ $?RT\ ?lhs\ ?g = ?rhs'$ ⟩ .

qed

hence ⟨ $?RT\ (?RT\ ?lhs\ ?g)\ ?g' = ?RT\ (?RT\ ?rhs\ ?g)\ ?g'$ ⟩ **by** *simp*

also from ⟨*length_n*(P)⟩ **have** ⟨ $?RT\ (?RT\ ?lhs\ ?g)\ ?g' = ?lhs$ ⟩

by (*auto simp flip: RenamingTick-comp intro!: RenamingTick-is-restrictable-on-strict-ticks-of*

$dest! : set\text{-}mp[OF\ Sync_{Pair}.strict\text{-}ticks\text{-}of\text{-}Sync_{ptick}\text{-}subset] \text{ is-ticks-length} D)$
also from $\langle length_{\checkmark n}(P) \rangle$ **have** $\langle ?RT\ (?RT\ ?rhs\ ?g)\ ?g' = ?rhs \rangle$
by $(fold\ RenamingTick\text{-}comp, subst\ (2)\ RenamingTick\text{-}id[of\ ?rhs, symmetric])$
 $(auto\ simp\ del: RenamingTick\text{-}id\ intro! : RenamingTick\text{-}is\text{-}restrictable\text{-}on\text{-}strict\text{-}ticks\text{-}of$
 $dest! : set\text{-}mp[OF\ Sync_{Pair}.strict\text{-}ticks\text{-}of\text{-}Sync_{ptick}\text{-}subset] \text{ is-ticks-length} D)$
finally show $\langle ?lhs = ?rhs \rangle$.
qed

corollary $Sync_{Pair}\text{-}to\text{-}Sync_{ListslenR} :$
 $\langle RenamingTick\ (P\ \llbracket S \rrbracket_{\checkmark Pair}\ Q)\ (\lambda(r, s). r @ s) = P\ n\llbracket S \rrbracket_{\checkmark ListslenR}\ Q \rangle$
 $(is\ \langle ?lhs = ?rhs \rangle)$ **if** $\langle length_{\checkmark n}(Q) \rangle$
proof –
let $?RT = RenamingTick$
have $\langle ?RT\ (P\ \llbracket S \rrbracket_{\checkmark Pair}\ Q)\ (\lambda(x, y). x @ y) =$
 $?RT\ (Q\ \llbracket S \rrbracket_{\checkmark Pair}\ P)\ ((\lambda(x, y). x @ y) \circ prod.swap) \rangle$
by $(simp\ add: RenamingTick\text{-}comp\ subst\ Sync_{Pair}.Sync_{ptick}\text{-}commute)$
also from $\langle length_{\checkmark n}(Q) \rangle$
have $\langle \dots = ?RT\ (Q\ \llbracket S \rrbracket_{\checkmark Pair}\ P)\ ((\lambda rs. drop\ n\ rs @ take\ n\ rs) \circ (\lambda(x, y). x @$
 $y)) \rangle$
by $(auto\ intro! : RenamingTick\text{-}is\text{-}restrictable\text{-}on\text{-}strict\text{-}ticks\text{-}of$
 $dest! : set\text{-}mp[OF\ Sync_{Pair}.strict\text{-}ticks\text{-}of\text{-}Sync_{ptick}\text{-}subset] \text{ is-ticks-length} D)$
also have $\langle \dots = ?RT\ (Q\ n\llbracket S \rrbracket_{\checkmark ListslenL}\ P)\ (\lambda rs. drop\ n\ rs @ take\ n\ rs) \rangle$
by $(simp\ add: RenamingTick\text{-}comp\ Sync_{Pair}\text{-}to\text{-}Sync_{ListslenL}[OF\ \langle length_{\checkmark n}(Q) \rangle])$
also have $\langle \dots = P\ n\llbracket S \rrbracket_{\checkmark ListslenR}\ Q \rangle$
by $(fact\ Sync_{ListslenL}.Sync_{ptick}\text{-}commute)$
finally show $\langle ?lhs = ?rhs \rangle$.
qed

corollary $Sync_{Pair}\text{-}to\text{-}Sync_{Lists} :$
 $\langle RenamingTick\ (P\ \llbracket S \rrbracket_{\checkmark Pair}\ Q)\ (\lambda(r, s). r @ s) = P\ n\llbracket S \rrbracket_{\checkmark m}\ Q \rangle$
 $(is\ \langle ?lhs = ?rhs \rangle)$ **if** $\langle length_{\checkmark n}(P) \rangle$ **and** $\langle length_{\checkmark m}(Q) \rangle$
by $(subst\ Sync_{Pair}\text{-}to\text{-}Sync_{ListslenL}[OF\ \langle length_{\checkmark n}(P) \rangle])$
 $(rule\ Sync_{ListslenL}\text{-}to\text{-}Sync_{Lists}[OF\ \langle length_{\checkmark m}(Q) \rangle])$

10.5 First Laws

corollary $Inter_{Classic}\text{-}STOP\ [simp] :$
 $\langle P\ |||_{\checkmark Classic}\ STOP = P ; STOP \rangle$
by $(simp\ add: Sync_{Classic}.Inter_{ptick}\text{-}STOP[of\ P\ id])$

corollary $Inter_{Pair}\text{-}STOP :$
 $\langle P\ |||_{\checkmark Pair}\ STOP = RenamingTick\ (P ; STOP)\ (\lambda r. (r, g\ r)) \rangle$
by $(simp\ add: Sync_{Pair}.Inter_{ptick}\text{-}STOP[of\ P\ g])$

corollary $Inter_{Pairlist}\text{-}STOP :$
 $\langle P\ |||_{\checkmark Pairlist}\ STOP = RenamingTick\ (P ; STOP)\ (\lambda r. [r, g\ r]) \rangle$
by $(simp\ add: Sync_{Pairlist}.Inter_{ptick}\text{-}STOP[of\ P\ g])$

corollary $Inter_{Rlist}\text{-}STOP :$

$\langle P \parallel \checkmark_{Rlist} STOP = RenamingTick (P ; STOP) (\lambda r. r \# g r) \rangle$
by (*simp add: Sync_{Rlist}.Inter_{ptick}-STOP[*of* $P g$]*)

corollary $Inter_{Llist}\text{-}STOP :$

$\langle P \parallel \checkmark_{Llist} STOP = RenamingTick (P ; STOP) (\lambda r. r @ [g r]) \rangle$
by (*simp add: Sync_{Rlist}.Sync_{ptick}-comm-locale-sym.Inter_{ptick}-STOP[*of* $P g$]*)

corollary $Inter_{ListslenL}\text{-}STOP :$

$\langle P \ n \parallel \checkmark_{ListslenL} STOP =$
 $RenamingTick (P ; STOP) (\lambda r. \text{if length } r = n \text{ then } r @ g r \text{ else undefined}) \rangle$
by (*auto simp add: Sync_{ListslenL}.Inter_{ptick}-STOP[*of* $n P g$] option.the-def*
intro!: arg-cong[where $f = \langle RenamingTick (P ; STOP) \rangle$])

corollary $Inter_{ListslenR}\text{-}STOP :$

$\langle P \ n \parallel \checkmark_{ListslenR} STOP =$
 $RenamingTick (P ; STOP) (\lambda r. \text{if length } (g r) = n \text{ then } r @ g r \text{ else undefined}) \rangle$
by (*auto simp add: Sync_{ListslenL}.Sync_{ptick}-comm-locale-sym.Inter_{ptick}-STOP[*of* $n P g$] option.the-def*
intro!: arg-cong[where $f = \langle RenamingTick (P ; STOP) \rangle$])

corollary $Inter_{Lists}\text{-}STOP :$

$\langle P \ n \parallel \checkmark_m STOP =$
 $RenamingTick (P ; STOP) (\lambda r. \text{if length } r = n \wedge \text{length } (g r) = m \text{ then } r @ g r \text{ else undefined}) \rangle$
by (*auto simp add: Sync_{Lists}.Inter_{ptick}-STOP[*of* $n m P g$] option.the-def*
intro!: arg-cong[where $f = \langle RenamingTick (P ; STOP) \rangle$])

corollary $STOP\text{-}Inter_{Classic} [simp] :$

$\langle STOP \parallel \checkmark_{Classic} Q = Q ; STOP \rangle$
by (*simp add: Sync_{Classic}.STOP-Inter_{ptick}[*of* $Q id$]*)

corollary $STOP\text{-}Inter_{Pair} :$

$\langle STOP \parallel \checkmark_{Pair} Q = RenamingTick (Q ; STOP) (\lambda s. (g s, s)) \rangle$
by (*simp add: Sync_{Pair}.STOP-Inter_{ptick}[*of* $Q g$]*)

corollary $STOP\text{-}Inter_{Pairlist} :$

$\langle STOP \parallel \checkmark_{Pairlist} Q = RenamingTick (Q ; STOP) (\lambda s. [g s, s]) \rangle$
by (*simp add: Sync_{Pairlist}.STOP-Inter_{ptick}[*of* $Q g$]*)

corollary $STOP\text{-}Inter_{Rlist} :$

$\langle STOP \parallel \checkmark_{Rlist} Q = RenamingTick (Q ; STOP) (\lambda s. g s \# s) \rangle$
by (*simp add: Sync_{Rlist}.STOP-Inter_{ptick}[*of* $Q g$]*)

corollary $STOP\text{-}Inter_{Llist} :$

$\langle STOP \parallel \checkmark_{Llist} Q = RenamingTick (Q ; STOP) (\lambda s. g s @ [s]) \rangle$
by (*simp add: Sync_{Rlist}.Sync_{ptick}-comm-locale-sym.STOP-Inter_{ptick}[*of* $Q g$]*)

corollary $STOP\text{-}Inter_{ListslenL}$:

$\langle STOP\ n ||| \checkmark_{ListslenL}\ Q =$
 $RenamingTick\ (Q ; STOP)\ (\lambda r. \text{ if length } (g\ r) = n \text{ then } g\ r @ r \text{ else undefined}) \rangle$
by $(auto\ simp\ add: Sync_{ListslenL}.STOP\text{-}Inter_{ptick}[of\ n\ Q\ g]\ option.the-def$
 $intro!: arg-cong[\text{where } f = \langle RenamingTick\ (Q ; STOP) \rangle])$

corollary $STOP\text{-}Inter_{ListslenR}$:

$\langle STOP\ n ||| \checkmark_{ListslenR}\ Q =$
 $RenamingTick\ (Q ; STOP)\ (\lambda r. \text{ if length } r = n \text{ then } g\ r @ r \text{ else undefined}) \rangle$
by $(auto\ simp\ add: Sync_{ListslenL}.Sync_{ptick-comm-locale-sym}.STOP\text{-}Inter_{ptick}[of\ n\ Q\ g]\ option.the-def$
 $intro!: arg-cong[\text{where } f = \langle RenamingTick\ (Q ; STOP) \rangle])$

corollary $STOP\text{-}Inter_{Lists}$:

$\langle STOP\ n ||| \checkmark_m\ Q =$
 $RenamingTick\ (Q ; STOP)\ (\lambda r. \text{ if length } (g\ r) = n \wedge \text{ length } r = m \text{ then } g\ r @$
 $r \text{ else undefined}) \rangle$
by $(auto\ simp\ add: Sync_{Lists}.STOP\text{-}Inter_{ptick}[of\ n\ m\ Q\ g]\ option.the-def$
 $intro!: arg-cong[\text{where } f = \langle RenamingTick\ (Q ; STOP) \rangle])$

corollary $SKIP\text{-}Sync_{Classic}\text{-}SKIP$:

$\langle SKIP\ r \llbracket A \rrbracket \checkmark_{Classic}\ SKIP\ s =$
 $(\text{if } r = s \text{ then } SKIP\ r \text{ else } STOP) \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{Pair}\text{-}SKIP$:

$\langle SKIP\ r \llbracket A \rrbracket \checkmark_{Pair}\ SKIP\ s = SKIP\ (r, s) \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{Pairlist}\text{-}SKIP$:

$\langle SKIP\ r \llbracket A \rrbracket \checkmark_{Pairlist}\ SKIP\ s = SKIP\ [r, s] \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{Rlist}\text{-}SKIP$:

$\langle SKIP\ r \llbracket A \rrbracket \checkmark_{Rlist}\ SKIP\ s = SKIP\ (r \# s) \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{Llist}\text{-}SKIP$:

$\langle SKIP\ r \llbracket A \rrbracket \checkmark_{Llist}\ SKIP\ s = SKIP\ (r @ [s]) \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{ListslenL}\text{-}SKIP$:

$\langle SKIP\ r\ n \llbracket A \rrbracket \checkmark_{ListslenL}\ SKIP\ s =$
 $(\text{if length } r = n \text{ then } SKIP\ (r @ s) \text{ else } STOP) \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{ListslenR}\text{-}SKIP$:

$\langle SKIP\ r\ n \llbracket A \rrbracket \checkmark_{ListslenR}\ SKIP\ s =$
 $(\text{if length } s = n \text{ then } SKIP\ (r @ s) \text{ else } STOP) \rangle$ **by** $simp$

corollary $SKIP\text{-}Sync_{Lists}\text{-}SKIP$:

$\langle SKIP\ r\ n \llbracket A \rrbracket \checkmark_m\ SKIP\ s =$

$(\text{if length } r = n \wedge \text{length } s = m \text{ then SKIP } (r @ s) \text{ else STOP}) \rangle \text{ by simp}$

10.6 Operational Laws

10.6.1 Classical Version

locale *After-Sync_{Classic}-locale* = *After-Sync_{ptick}-locale* $\langle \lambda r s. \text{if } r = s \text{ then } \lfloor r \rfloor \text{ else } \Diamond \rangle$
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \llbracket S \rrbracket_{\checkmark_{Classic}} Q \rangle \text{ by (fact refl)}$

end

locale *AfterExt-Sync_{Classic}-locale* =
AfterExt-Sync_{ptick}-locale $\langle \lambda r s. \text{if } r = s \text{ then } \lfloor r \rfloor \text{ else } \Diamond \rangle$

sublocale *AfterExt-Sync_{Classic}-locale* $\subseteq$ *After-Sync_{Classic}-locale*
by *unfold-locale*s

locale *OpSemTransitions-Sync_{Classic}-locale* =
OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. \text{if } r = s \text{ then } \lfloor r \rfloor \text{ else } \Diamond \rangle$

sublocale *OpSemTransitions-Sync_{Classic}-locale* $\subseteq$ *AfterExt-Sync_{Classic}-locale*
by *unfold-locale*s

10.6.2 Product Type

locale *After-Sync_{Pair}-locale* = *After-Sync_{ptick}-locale* $\langle \lambda r s. \lfloor (r, s) \rfloor \rangle$
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \llbracket S \rrbracket_{\checkmark_{Pair}} Q \rangle \text{ by (fact refl)}$

end

locale *AfterExt-Sync_{Pair}-locale* =
AfterExt-Sync_{ptick}-locale $\langle \lambda r s. \lfloor (r, s) \rfloor \rangle$

sublocale *AfterExt-Sync_{Pair}-locale* $\subseteq$ *After-Sync_{Pair}-locale*
by *unfold-locale*s

locale *OpSemTransitions-Sync_{Pair}-locale* =
OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. \lfloor (r, s) \rfloor \rangle$

sublocale *OpSemTransitions-Sync_{Pair}-locale* $\subseteq$ *AfterExt-Sync_{Pair}-locale*
by *unfold-locale*s

10.6.3 List Type

Pair

locale *After-Sync_{Pairlist}-locale* = *After-Sync_{ptick}-locale* $\langle \lambda r s. \llbracket r, s \rrbracket \rangle$
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \llbracket S \rrbracket_{\checkmark Pairlist} Q \rangle$ **by** (*fact refl*)

end

locale *AfterExt-Sync_{Pairlist}-locale* =
AfterExt-Sync_{ptick}-locale $\langle \lambda r s. \llbracket r, s \rrbracket \rangle$

sublocale *AfterExt-Sync_{Pairlist}-locale* $\subseteq$ *After-Sync_{Pairlist}-locale*
by *unfold-locales*

locale *OpSemTransitions-Sync_{Pairlist}-locale* =
OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. \llbracket r, s \rrbracket \rangle$

sublocale *OpSemTransitions-Sync_{Pairlist}-locale* $\subseteq$ *AfterExt-Sync_{Pairlist}-locale*
by *unfold-locales*

Right List

locale *After-Sync_{Rlist}-locale* = *After-Sync_{ptick}-locale* $\langle \lambda r s. \lfloor r \# s \rfloor \rangle$
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \llbracket S \rrbracket_{\checkmark Rlist} Q \rangle$ **by** (*fact refl*)

end

locale *AfterExt-Sync_{Rlist}-locale* =
AfterExt-Sync_{ptick}-locale $\langle \lambda r s. \lfloor r \# s \rfloor \rangle$

sublocale *AfterExt-Sync_{Rlist}-locale* $\subseteq$ *After-Sync_{Rlist}-locale*
by *unfold-locales*

locale *OpSemTransitions-Sync_{Rlist}-locale* =
OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. \lfloor r \# s \rfloor \rangle$

sublocale *OpSemTransitions-Sync_{Rlist}-locale* $\subseteq$ *AfterExt-Sync_{Rlist}-locale*
by *unfold-locales*

Left List

locale *After-Sync_{Llist}-locale* = *After-Sync_{ptick}-locale* $\langle \lambda r s. \lfloor r @ [s] \rfloor \rangle$
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \llbracket S \rrbracket_{\checkmark List} Q \rangle$ **by** (*fact refl*)

end

locale *AfterExt-Sync_{List}-locale* =
AfterExt-Sync_{ptick}-locale $\langle \lambda r s. [r @ [s]] \rangle$

sublocale *AfterExt-Sync_{List}-locale* $\subseteq$ *After-Sync_{List}-locale*
by *unfold-locales*

locale *OpSemTransitions-Sync_{List}-locale* =
OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. [r @ [s]] \rangle$

sublocale *OpSemTransitions-Sync_{List}-locale* $\subseteq$ *AfterExt-Sync_{List}-locale*
by *unfold-locales*

Arbitrary Lists

Control on left side **locale** *After-Sync_{ListslenL}-locale* =
After-Sync_{ptick}-locale $\langle \lambda r s. \text{if length } r = \text{lenL then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL* :: nat
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \llbracket S \rrbracket_{\checkmark ListslenL} Q \rangle$ **by** (*fact refl*)

end

locale *AfterExt-Sync_{ListslenL}-locale* =
AfterExt-Sync_{ptick}-locale $\langle \lambda r s. \text{if length } r = \text{lenL then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL* :: nat

sublocale *AfterExt-Sync_{ListslenL}-locale* $\subseteq$ *After-Sync_{ListslenL}-locale*
by *unfold-locales*

locale *OpSemTransitions-Sync_{ListslenL}-locale* =
OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. \text{if length } r = \text{lenL then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL* :: nat

sublocale *OpSemTransitions-Sync_{ListslenL}-locale* $\subseteq$ *AfterExt-Sync_{ListslenL}-locale*
by *unfold-locales*

Control on right side **locale** *After-Sync_{ListslenR}-locale* =
After-Sync_{ptick}-locale $\langle \lambda r s. \text{if length } s = \text{lenR then } [r @ s] \text{ else } \Diamond \rangle$
for *lenR* :: nat
begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \text{ lenR} \llbracket S \rrbracket_{\checkmark \text{ListslenR}} Q \rangle$ **by** (*fact refl*)

end

locale *AfterExt-Sync_{ListslenR}-locale* =

AfterExt-Sync_{ptick}-locale $\langle \lambda r s. \text{if length } s = \text{lenR} \text{ then } [r @ s] \text{ else } \Diamond \rangle$
for *lenR* :: nat

sublocale *AfterExt-Sync_{ListslenR}-locale* $\subseteq$ *After-Sync_{ListslenR}-locale*
by *unfold-locales*

locale *OpSemTransitions-Sync_{ListslenR}-locale* =

OpSemTransitions-Sync_{ptick}-locale $\langle \lambda r s. \text{if length } r = \text{lenL} \text{ then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL* :: nat

sublocale *OpSemTransitions-Sync_{ListslenR}-locale* $\subseteq$ *AfterExt-Sync_{ListslenR}-locale*
by *unfold-locales*

Control on both sides **locale** *After-Sync_{Lists}-locale* =

After-Sync_{ptick}-locale
 $\langle \lambda r s. \text{if length } r = \text{lenL} \wedge \text{length } s = \text{lenR} \text{ then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL lenR* :: nat

begin

— Just checking...

lemma $\langle \text{Sync}_{ptick} P S Q = P \text{ lenL} \llbracket S \rrbracket_{\checkmark \text{lenR}} Q \rangle$ **by** (*fact refl*)

end

locale *AfterExt-Sync_{Lists}-locale* =

AfterExt-Sync_{ptick}-locale
 $\langle \lambda r s. \text{if length } r = \text{lenL} \wedge \text{length } s = \text{lenR} \text{ then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL lenR* :: nat

sublocale *AfterExt-Sync_{Lists}-locale* $\subseteq$ *After-Sync_{Lists}-locale*
by *unfold-locales*

locale *OpSemTransitions-Sync_{Lists}-locale* =

OpSemTransitions-Sync_{ptick}-locale
 $\langle \lambda r s. \text{if length } r = \text{lenL} \wedge \text{length } s = \text{lenR} \text{ then } [r @ s] \text{ else } \Diamond \rangle$
for *lenL lenR* :: nat

sublocale *OpSemTransitions-Sync_{Lists}-locale* $\subseteq$ *AfterExt-Sync_{Lists}-locale*
by *unfold-locales*

Chapter 11

Architectural Versions

11.1 Sequential Composition

11.1.1 Definition

fun $MultiSeq_{ptick} :: \langle [b \text{ list}, 'b \Rightarrow 'r \Rightarrow ('a, 'r) \text{ process}_{ptick}, 'r] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
where $MultiSeq_{ptick}\text{-Nil} : \langle MultiSeq_{ptick} [] \ P = SKIP \rangle$
 $| \ MultiSeq_{ptick}\text{-Cons} : \langle MultiSeq_{ptick} (l \# L) \ P = (\lambda r. P \ l \ r ; \checkmark MultiSeq_{ptick} \ L \ P) \rangle$

syntax $\text{-}MultiSeq_{ptick} ::$
 $\langle [pttrn, 'b \text{ list}, 'b \Rightarrow 'r \Rightarrow ('a, 'r) \text{ process}_{ptick}, 'r] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
 $(\langle (3SEQ_{\checkmark} \text{-} \in @ \text{-} / \text{-}) \rangle [78, 78, 77] \ 77)$

syntax-consts $\text{-}MultiSeq_{ptick} \equiv MultiSeq_{ptick}$

translations $SEQ_{\checkmark} \ p \in @ \ L. \ P \equiv CONST \ MultiSeq_{ptick} \ L \ (\lambda p. \ P)$

11.1.2 First Properties

lemma $\langle SEQ_{\checkmark} \ p \in @ \ []. \ P \ p = SKIP \rangle$
and $\langle SEQ_{\checkmark} \ p \in @ \ [a]. \ P \ p = (\lambda r. \ P \ a \ r) \rangle$
and $\langle SEQ_{\checkmark} \ p \in @ \ [a, b]. \ P \ p = (\lambda r. \ P \ a \ r ; \checkmark P \ b) \rangle$
and $\langle SEQ_{\checkmark} \ p \in @ \ [a, b, c]. \ P \ p = (\lambda r. \ P \ a \ r ; \checkmark P \ b ; \checkmark P \ c) \rangle$
by $(simp\text{-}all \ add: Seq_{ptick}\text{-}assoc)$

lemma $\langle SEQ_{\checkmark} \ p \in @ \ [1::int .. 3]. \ P \ p = (\lambda r. \ P \ 1 \ r ; \checkmark P \ 2 ; \checkmark P \ 3) \rangle$
by $(simp \ add: upto.simps \ Seq_{ptick}\text{-}assoc)$

lemma $\langle (SEQ_{\checkmark} \ p \in @ \ []. \ P \ p) = SKIP \rangle$ **by** $(fact \ MultiSeq_{ptick}\text{-}Nil)$

lemma $\langle (SEQ_{\checkmark} \ l \in @ \ (a \# L). \ P \ l) = (\lambda r. \ P \ a \ r ; \checkmark SEQ_{\checkmark} \ l \in @ \ L. \ P \ l) \rangle$ **by** $(fact \ MultiSeq_{ptick}\text{-}Cons)$

lemma *MultiSeq_{ptick}-singl* [simp] : $\langle \text{SEQ}_{\checkmark} l \in @ [a]. P l = P a \rangle$ **by** *simp*

lemma *MultiSeq_{ptick}-snoc* : $\langle \text{SEQ}_{\checkmark} l \in @ (L @ [a]). P l = (\lambda r. (\text{SEQ}_{\checkmark} l \in @ L. P l) r ;_{\checkmark} P a) \rangle$
by (*induct L*) (*simp-all add: Seq_{ptick}-assoc*)

lemma *mono-MultiSeq_{ptick}-eq*:
 $\langle (\bigwedge l. l \in \text{set } L \implies P l = Q l) \implies \text{SEQ}_{\checkmark} l \in @ L. P l = \text{SEQ}_{\checkmark} l \in @ L. Q l \rangle$
by (*induct L*) *fastforce+*

lemma *MultiSeq_{ptick}-const* [simp] :
 $\langle (\text{SEQ}_{\checkmark} l \in @ L. (\lambda r. P l)) =$
 $(\text{if } L = [] \text{ then SKIP else } (\lambda r. \text{SEQ } l \in @ L. P l)) \rangle$
by (*induct L rule: rev-induct*) (*auto simp add: MultiSeq_{ptick}-snoc*)

11.1.3 Behaviour with binary version

lemma *MultiSeq_{ptick}-append*:
 $\langle \text{SEQ}_{\checkmark} l \in @ (L1 @ L2). P l = (\lambda r. (\text{SEQ}_{\checkmark} l \in @ L1. P l) r ;_{\checkmark} \text{SEQ}_{\checkmark} l \in @ L2. P l) \rangle$
by (*induct L1 rule: list.induct, simp-all, metis Seq_{ptick}-assoc*)

11.1.4 Other Properties

lemma *MultiSeq_{ptick}-SKIP-neutral*:
 $\langle P a = \text{SKIP} \implies \text{SEQ}_{\checkmark} l \in @ (L1 @ [a] @ L2). P l = \text{SEQ}_{\checkmark} l \in @ (L1 @ L2). P l \rangle$
by (*simp add: MultiSeq_{ptick}-append*)

lemma *MultiSeq_{ptick}-BOT-absorb*:
 $\langle P a = \perp \implies \text{SEQ}_{\checkmark} l \in @ (L1 @ [a] @ L2). P l = (\lambda r. (\text{SEQ}_{\checkmark} l \in @ L1. P l) r ;_{\checkmark} \perp) \rangle$
by (*simp add: MultiSeq_{ptick}-append lambda-strict*)

lemma *MultiSeq_{ptick}-STOP-absorb*:
 $\langle P a = (\lambda r. \text{STOP}) \implies \text{SEQ}_{\checkmark} l \in @ (L1 @ [a] @ L2). P l =$
 $(\lambda r. (\text{SEQ}_{\checkmark} l \in @ L1. P l) r ; \text{STOP}) \rangle$
by (*simp add: MultiSeq_{ptick}-append*)

lemma *is-ticks-length-MultiSeq_{ptick}* [is-ticks-length-intro] :
 $\langle \text{length}_{\checkmark n}((\text{SEQ}_{\checkmark} l \in @ L. P l) r) \rangle$
if $\langle L \neq [] \rangle$ **and** $\langle \bigwedge r'. r' \in \checkmark s((\text{SEQ}_{\checkmark} l \in @ (\text{butlast } L). P l) r) \implies \text{length}_{\checkmark n}(P (\text{last } L) r') \rangle$

proof –

from *that(1)* **obtain** $l L'$ **where** $\langle L = L' @ [l] \rangle$
by (*cases L rule: rev-cases*) *auto*

with *that*(2) **have** $\langle r' \in \check{s}((SEQ_{\check{}} l \in @ L'. P l) r) \implies length_{\check{}}(P l r') \rangle$ **for**
 r' **by** *simp*
thus *?thesis*
by (*auto simp add: $\langle L = L' @ [l] \rangle MultiSeq_{ptick}$ -snoc intro: is-ticks-length-Seq_{ptick}*)
qed

11.1.5 Behaviour with injectivity

lemma *inj-on-mapping-over-MultiSeq_{ptick}*:
 $\langle inj-on f (set L) \implies$
 $SEQ_{\check{}} l \in @ L. P l = SEQ_{\check{}} l \in @ map f L. P (inv-into (set L) f l) \rangle$
proof (*induct L*)
show $\langle inj-on f (set []) \implies MultiSeq_{ptick} [] P =$
 $SEQ_{\check{}} x \in @ map f []. P (inv-into (set []) f x) \rangle$ **by** *simp*
next
case (*Cons a L*)
show *?case*
proof (*rule ext*)
fix r
have $\langle (SEQ_{\check{}} l \in @ (a \# L). P l) r = P a r ; \check{ } SEQ_{\check{}} l \in @ L. P l \rangle$ **by** *simp*
also have $\langle SEQ_{\check{}} l \in @ L. P l = SEQ_{\check{}} l \in @ map f L. P (inv-into (set L) f l) \rangle$
using *Cons.hyps Cons.premis* **by** *auto*
also have $\langle \dots = SEQ_{\check{}} l \in @ map f L. P (inv-into (set (a \# L)) f l) \rangle$
using *Cons.premis* **by** (*auto intro!: mono-MultiSeq_{ptick}-eq*)
finally show $\langle (SEQ_{\check{}} l \in @ (a \# L). P l) r =$
 $(SEQ_{\check{}} l \in @ map f (a \# L). P (inv-into (set (a \# L)) f l)) r \rangle$
using *Cons.premis* **by** *auto*
qed
qed

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* $(P \llbracket A \rrbracket_{\check{}} Q) \otimes \check{\Rightarrow} \otimes \check{\Rightarrow}_{rev} = Q \llbracket A \rrbracket_{\check{\Rightarrow}_{rev}} P$). We therefore define the architectural version on a list.

fun *MultiSync_{ptick}* ::
 $\langle 'a \text{ set}, 'b \text{ list}, 'b \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle \Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle$
where $\langle MultiSync_{ptick} S [] P = STOP \rangle$

$$\begin{array}{|l} \langle \text{MultiSync}_{ptick} S [l] P = \text{RenamingTick} (P l) (\lambda r. [r]) \rangle \\ \langle \text{MultiSync}_{ptick} S (l \# m \# L) P = P l \llbracket S \rrbracket_{\checkmark Rlist} \text{MultiSync}_{ptick} S (m \# L) P \rangle \end{array}$$

syntax $\text{-MultiSync}_{ptick} ::$
 $\langle [pttrn, 'a \text{ set}, 'b \text{ list}, ('a, 'r) \text{ process}_{ptick}] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
 $\langle (\exists [-]_{\checkmark} - \in @ - / -) \rangle [78, 78, 78, 77] \ 77$
syntax-consts $\text{-MultiSync}_{ptick} \equiv \text{MultiSync}_{ptick}$
translations $\llbracket S \rrbracket_{\checkmark} l \in @ L. P \Rightarrow \text{CONST MultiSync}_{ptick} S L (\lambda l. P)$

Special case of $\text{MultiSync}_{ptick} S P$ when $S = \{\}$.

abbreviation $\text{MultiInter}_{ptick} ::$
 $\langle ['b \text{ list}, 'b \Rightarrow ('a, 'r) \text{ process}_{ptick}] \Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle$
where $\langle \text{MultiInter}_{ptick} L P \equiv \text{MultiSync}_{ptick} \{\} L P \rangle$

syntax $\text{-MultiInter}_{ptick} ::$
 $\langle [pttrn, 'b \text{ list}, ('a, 'r) \text{ process}_{ptick}] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
 $\langle (\exists [||]_{\checkmark} - \in @ - / -) \rangle [78, 78, 77] \ 77$
syntax-consts $\text{-MultiInter}_{ptick} \equiv \text{MultiInter}_{ptick}$
translations $||_{\checkmark} l \in @ L. P \Rightarrow \text{CONST MultiInter}_{ptick} L (\lambda l. P)$

Special case of $\text{MultiSync}_{ptick} S P$ when $S = \text{UNIV}$.

abbreviation $\text{MultiPar}_{ptick} ::$
 $\langle ['b \text{ list}, 'b \Rightarrow ('a, 'r) \text{ process}_{ptick}] \Rightarrow ('a, 'r \text{ list}) \text{ process}_{ptick} \rangle$
where $\langle \text{MultiPar}_{ptick} L P \equiv \text{MultiSync}_{ptick} \text{UNIV} L P \rangle$

syntax $\text{-MultiPar}_{ptick} ::$
 $\langle [pttrn, 'b \text{ list}, ('a, 'r) \text{ process}_{ptick}] \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
 $\langle (\exists [||]_{\checkmark} - \in @ - / -) \rangle [78, 78, 77] \ 77$
syntax-consts $\text{-MultiPar}_{ptick} \equiv \text{MultiPar}_{ptick}$
translations $||_{\checkmark} l \in @ L. P \Rightarrow \text{CONST MultiPar}_{ptick} L (\lambda l. P)$

11.2.2 First properties

lemma $\text{is-ticks-length-MultiSync}_{ptick} [\text{is-ticks-length-intro}] :$
 $\langle \text{length}_{\checkmark} \text{length} L (\llbracket S \rrbracket_{\checkmark} l \in @ L. P l) \rangle$
by (induct L rule: *induct-list012*)
(simp-all add: is-ticks-length-STOP is-ticks-length-Renaming is-ticks-length-Suc-Sync_{Rlist})

lemma $\text{MultiSync}_{ptick}\text{-Cons} :$
 $\langle \llbracket S \rrbracket_{\checkmark} m \in @ (l \# L). P m =$
 $(\text{ if } L = [] \text{ then } \text{RenamingTick} (P l) (\lambda r. [r])$
 $\text{ else } P l \llbracket S \rrbracket_{\checkmark Rlist} \llbracket S \rrbracket_{\checkmark} m \in @ L. P m) \rangle$
by (cases L) *simp-all*

lemma *mono-MultiSync_{ptick}-eq* :
 $\langle (\bigwedge l. l \in \text{set } L \implies P \ l = Q \ l) \implies \llbracket S \rrbracket_{\checkmark} l \in @ L. P \ l = \llbracket S \rrbracket_{\checkmark} l \in @ L. Q \ l \rangle$
by (*induct L rule: induct-list012*) *simp-all*

lemma *mono-MultiSync_{ptick}-eq2*:
 $\langle (\bigwedge l. l \in \text{set } L \implies P \ (f \ l) = Q \ l) \implies \llbracket S \rrbracket_{\checkmark} l \in @ \text{map } f \ L. P \ l = \llbracket S \rrbracket_{\checkmark} l \in @ L. Q \ l \rangle$
by (*induct L rule: induct-list012*) *simp-all*

— Some tests

lemma $\langle (\llbracket S \rrbracket_{\checkmark} l \in @ [] . P \ l) = \text{STOP} \rangle$
and $\langle (\llbracket S \rrbracket_{\checkmark} l \in @ [a] . P \ l) = \text{RenamingTick } (P \ a) \ (\lambda r. [r]) \rangle$
and $\langle (\llbracket S \rrbracket_{\checkmark} l \in @ [a, b] . P \ l) = P \ a \llbracket S \rrbracket_{\checkmark \text{Rlist}} \text{RenamingTick } (P \ b) \ (\lambda r. [r]) \rangle$
and $\langle (\llbracket S \rrbracket_{\checkmark} l \in @ [a, b, c] . P \ l) = P \ a \llbracket S \rrbracket_{\checkmark \text{Rlist}} (P \ b \llbracket S \rrbracket_{\checkmark \text{Rlist}} \text{RenamingTick } (P \ c) \ (\lambda r. [r])) \rangle$
by *simp-all*

11.2.3 Properties

lemma *MultiSync_{ptick}-is-BOT-iff*:
 $\langle \llbracket S \rrbracket_{\checkmark} l \in @ L. P \ l = \perp \longleftrightarrow (\exists l \in \text{set } L. P \ l = \perp) \rangle$
by (*induct L rule: induct-list012*)
(simp-all add: Renaming-is-BOT-iff Sync_{Rlist}.Sync_{ptick}-is-BOT-iff)

lemma *MultiSync_{ptick}-BOT-absorb*:
 $\langle l \in \text{set } L \implies P \ l = \perp \implies \llbracket S \rrbracket_{\checkmark} l \in @ L. P \ l = \perp \rangle$
using *MultiSync_{ptick}-is-BOT-iff* **by** *blast*

lemma *MultiSync_{ptick}-SKIP-id* :
 $\langle \llbracket S \rrbracket_{\checkmark} r \in @ L. \text{SKIP } r = (\text{if } L = [] \text{ then } \text{STOP} \text{ else } \text{SKIP } L) \rangle$
by (*induct L rule: induct-list012*) *simp-all*

11.2.4 Behaviour with binary version

lemma *MultiSync_{ptick}-append* :
 $\langle L1 \neq [] \implies L2 \neq [] \implies$
 $\llbracket S \rrbracket_{\checkmark} l \in @ (L1 \ @ \ L2) . P \ l =$
 $\llbracket S \rrbracket_{\checkmark} l \in @ L1 . P \ l \ \text{length } L1 \llbracket S \rrbracket_{\checkmark \text{length } L2} \llbracket S \rrbracket_{\checkmark} l \in @ L2 . P \ l \rangle$
proof (*induct L1 rule: list-nonempty-induct*)
case (*single l*) **thus** ?*case*
by (*simp add: is-ticks-length-MultiSync_{ptick} MultiSync_{ptick}-Cons flip: Sync_{Rlist}-to-Sync_{Lists}*)
next
let ?*RT* = $\langle \lambda P. \text{RenamingTick } P \ (\lambda r. [r]) \rangle$
case (*cons l L1*)

have $\langle \llbracket S \rrbracket_{\checkmark} l \in @ ((l \# L1) @ L2). P l = P l \llbracket S \rrbracket_{\checkmark Rlist} \llbracket S \rrbracket_{\checkmark} l \in @ (L1 @ L2). P l \rangle$
 by (simp add: MultiSync_{ptick}-Cons $\langle L1 \neq [] \rangle$)
 also have $\langle \dots = ?RT (P l) Suc 0 \llbracket S \rrbracket_{\checkmark length (L1 @ L2)} \llbracket S \rrbracket_{\checkmark} l \in @ (L1 @ L2). P l \rangle$
 P l
 by (intro Sync_{Rlist}-to-Sync_{Lists} is-ticks-length-MultiSync_{ptick})
 also have $\langle \dots = ?RT (P l) Suc 0 \llbracket S \rrbracket_{\checkmark length L1 + length L2} \llbracket S \rrbracket_{\checkmark} l \in @ L1. P l \text{ length } L1 \llbracket S \rrbracket_{\checkmark length L2} \llbracket S \rrbracket_{\checkmark} l \in @ L2. P l \rangle$
 using cons.hyps(2) cons.prem by simp
 also have $\langle \dots = ?RT (P l) Suc 0 \llbracket S \rrbracket_{\checkmark length L1} \llbracket S \rrbracket_{\checkmark} l \in @ L1. P l \text{ Suc } 0 + length L1 \llbracket S \rrbracket_{\checkmark length L2} \llbracket S \rrbracket_{\checkmark} l \in @ L2. P l \rangle$
 by (simp add: Sync_{Lists}-assoc)
 also have $\langle ?RT (P l) Suc 0 \llbracket S \rrbracket_{\checkmark length L1} \llbracket S \rrbracket_{\checkmark} l \in @ L1. P l = P l \llbracket S \rrbracket_{\checkmark Rlist} \llbracket S \rrbracket_{\checkmark} l \in @ L1. P l \rangle$
 by (intro Sync_{Rlist}-to-Sync_{Lists} [symmetric] is-ticks-length-MultiSync_{ptick})
 also have $\langle \dots = \llbracket S \rrbracket_{\checkmark} l \in @ (l \# L1). P l \rangle$
 by (simp add: MultiSync_{ptick}-Cons $\langle L1 \neq [] \rangle$)
 finally show ?case by simp
 qed

11.2.5 Behaviour with injectivity

lemma *inj-on-mapping-over-MultiSync_{ptick}*:

$\langle inj\text{-on } f \text{ (set } L) \implies$
 $\llbracket S \rrbracket_{\checkmark} l \in @ L. P l = \llbracket S \rrbracket_{\checkmark} l \in @ \text{map } f L. P \text{ (inv-into (set } L) f l) \rangle$
proof (induct L rule: induct-list012)
 case (3 l' l'' L)
 have $\langle \llbracket S \rrbracket_{\checkmark} l \in @ (l' \# l'' \# L). P l = P l' \llbracket S \rrbracket_{\checkmark Rlist} \llbracket S \rrbracket_{\checkmark} l \in @ (l'' \# L). P l \text{ by simp} \rangle$
 also have $\langle \llbracket S \rrbracket_{\checkmark} l \in @ (l'' \# L). P l = \llbracket S \rrbracket_{\checkmark} l \in @ \text{map } f (l'' \# L). P \text{ (inv-into (set (l'' \# L)) } f l) \rangle$
 by (metis 3.hyps(2) 3.prem inj-on-insert list.simps(15))
 also have $\langle \dots = \llbracket S \rrbracket_{\checkmark} l \in @ \text{map } f (l' \# L). P \text{ (inv-into (set (l' \# l'' \# L)) } f l) \rangle$
 using 3.prem by (auto intro!: mono-MultiSync_{ptick}-eq)
 also have $\langle P l' = P \text{ (inv-into (set (l' \# l'' \# L)) } f (f l')) \rangle$
 using 3.prem by auto
 finally show ?case by simp
 qed simp-all

11.2.6 Permuting the Sequence

A particular Case

lemma *MultiSync_{ptick}-snoc* :

$\langle \llbracket S \rrbracket_{\checkmark} m \in @ (L @ [l]). P m =$
 $(\text{if } L = [] \text{ then RenamingTick } (P l) (\lambda r. [r])$
 $\text{else } \llbracket S \rrbracket_{\checkmark} m \in @ L. P m \llbracket S \rrbracket_{\checkmark Llist} P l) \rangle$
 by (simp add: MultiSync_{ptick}-append)

(metis (lifting) ext SyncList-to-SyncLists is-ticks-length-MultiSyncptick)

At the beginning, we wanted to prove the following property.

```

theorem MultiSyncptick-rev :
  ⟨[[S]]✓ l ∈@ (rev L). P l = RenamingTick ([[S]]✓ l ∈@ L. P l) rev⟩
proof (induct L)
  case Nil show ?case by simp
next
  let ?RT = ⟨RenamingTick⟩
  case (Cons l L)
  show ?case
  proof (cases ⟨L = []⟩)
    show ⟨L = [] ⟹ ?case⟩ by (simp add: comp-def flip: Renaming-comp id-def)
  next
    assume ⟨L ≠ []⟩
    have ⟨[[S]]✓ m ∈@ (rev (l # L)). P m = [[S]]✓ m ∈@ (rev L). P m [[S]]✓ Llist
P l⟩
      by (simp add: MultiSyncptick-snoc ⟨L ≠ []⟩)
    also have ⟨... = ?RT ([[S]]✓ m ∈@ L. P m) rev [[S]]✓ Llist P l⟩
      by (simp only: Cons.hyps)
    also have ⟨... = ?RT ([[S]]✓ m ∈@ L. P m) rev [[S]]✓ Llist ?RT (P l) id⟩ by
simp
    also have ⟨... = Syncptick-locale.Syncptick (λr s. Some (rev r @ [s]))
([[S]]✓ m ∈@ L. P m) S (P l)⟩
      by (subst SyncRlist.Syncptick-comm-locale-sym.inj-RenamingTick-Syncptick-inj-RenamingTick)
simp-all
    also have ⟨... = ?RT (P l [[S]]✓ Rlist [[S]]✓ m ∈@ L. P m) rev⟩
    proof (subst SyncRlist.inj-on-RenamingTick-Syncptick)
      show ⟨inj-on rev SyncRlist.range-tick-join⟩ by simp
    next
      show ⟨Syncptick-locale.Syncptick (λr s. Some (rev r @ [s])) ([[S]]✓ m ∈@ L.
P m) S (P l) =
      Syncptick-locale.Syncptick
      (λr s. case Some (r # s) of None ⇒ None | Some r-s ⇒ Some (rev r-s))
      (P l) S (MultiSyncptick S L P)⟩
      by (subst Syncptick-locale.Syncptick-sym, simp-all)
      (unfold-locale, blast)
    qed
    also have ⟨P l [[S]]✓ Rlist [[S]]✓ m ∈@ L. P m = [[S]]✓ m ∈@ (l # L). P m⟩
      by (simp add: MultiSyncptick-Cons ⟨L ≠ []⟩)
    finally show ?case .
  qed
qed

```

This has just been established for $\text{rev } L$, 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* :

$\langle i < \text{length } xs \implies j < \text{length } xs \implies$
 $\text{permute-list } (\text{Transposition.transpose } i \ j) \ xs = xs[i := xs!j, j := xs!i] \rangle$
by (*auto simp add: permute-list-def transpose-def intro: nth-equalityI*)

lemma *inj-on-permute-list-transpose* :

$\langle i < n \implies j < n \implies \text{inj-on } (\text{permute-list } (\text{Transposition.transpose } i \ j)) \ \{xs. n \leq \text{length } xs\} \rangle$
by (*auto intro!: inj-onI simp add: permute-list-transpose-eq-list-update*)
(metis length-list-update nth-equalityI nth-list-update-eq nth-list-update-neq order-less-le-trans)

lemma *rev-permute-list-transpose* :

$\langle i < \text{length } L \implies j < \text{length } L \implies$
 $\text{rev } (\text{permute-list } (\text{Transposition.transpose } i \ j) \ L) =$
 $\text{permute-list } (\text{Transposition.transpose } (\text{length } L - \text{Suc } i) \ (\text{length } L - \text{Suc } j)) \ (\text{rev } L) \rangle$
by (*simp add: permute-list-transpose-eq-list-update rev-nth rev-update*)

lemma *permute-list-transpose-rev* :

$\langle i < \text{length } L \implies j < \text{length } L \implies$
 $\text{permute-list } (\text{Transposition.transpose } i \ j) \ (\text{rev } L) =$
 $\text{rev } (\text{permute-list } (\text{Transposition.transpose } (\text{length } L - \text{Suc } i) \ (\text{length } L - \text{Suc } j)) \ L) \rangle$
by (*simp add: permute-list-transpose-eq-list-update rev-nth rev-update*)

lemma *tickFree-map-map-event_{ptick}-id-eq* :

$\langle tF \ t \implies \text{map } (\text{map-event}_{ptick} \ \text{id } g) \ t = t \rangle$
and *tickFree-mem-T-RenamingTick-iff-mem-T* :
 $\langle tF \ t \implies t \in \mathcal{T} \ (\text{RenamingTick } P \ g) \longleftrightarrow t \in \mathcal{T} \ P \rangle$
and *tickFree-mem-D-RenamingTick-iff-mem-D* :
 $\langle tF \ t \implies t \in \mathcal{D} \ (\text{RenamingTick } P \ g) \longleftrightarrow t \in \mathcal{D} \ P \rangle$
for $P :: \langle ('a, 'r) \text{ process}_{ptick} \rangle$ **and** $g :: \langle 'r \Rightarrow 'r \rangle$

— Necessarily here, antecedents and images for g share the same type.

proof —

show $*$: $\langle tF \ t \implies \text{map } (\text{map-event}_{ptick} \ \text{id } g) \ t = t \rangle$ **for** $t :: \langle ('a, 'r) \text{ trace}_{ptick} \rangle$
by (*induct t*) (*auto simp add: is-ev-def*)
show $\langle tF \ t \implies t \in \mathcal{T} \ (\text{RenamingTick } P \ g) \longleftrightarrow t \in \mathcal{T} \ P \rangle$
by (*auto simp add: T-Renaming * map-event_{ptick}-tickFree D-T is-processT7*)
show $\langle tF \ t \implies t \in \mathcal{D} \ (\text{RenamingTick } P \ g) \longleftrightarrow t \in \mathcal{D} \ P \rangle$
by (*auto simp add: D-Renaming * is-processT7*)
*(metis * front-tickFree-Nil self-append-conv)*

qed

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 syn-

chronization $P \llbracket S \rrbracket_{\checkmark Rlist} Q$.

lemma *Sync_{Rlist}-RenamingTick-permute-list-transpose* :

$\langle P \llbracket S \rrbracket_{\checkmark Rlist} RenamingTick Q (permute-list (Transposition.transpose i j)) =$
 $RenamingTick (P \llbracket S \rrbracket_{\checkmark Rlist} Q) (permute-list (Transposition.transpose (Suc i)$
 $(Suc j))) \rangle$
 $(is \langle ?lhs = ?rhs \rangle) \text{ if } \langle i < n \rangle \langle j < n \rangle \langle \bigwedge rs. rs \in \checkmark s(Q) \implies n \leq length rs \rangle$

proof –

let $? \tau = Transposition.transpose$
let $?pl-\tau = \langle \lambda i j. permute-list (? \tau i j) \rangle$
let $?fun-evt = \langle \lambda i j. map-event_{ptick} id (?pl-\tau i j) \rangle$
let $?map-evt = \langle \lambda i j. map (?fun-evt i j) \rangle$
let $?RT = \langle \lambda i j P. RenamingTick P (?pl-\tau i j) \rangle$
let $?tj = \langle \lambda r s. [r \# s] \rangle$
note $map-event_{ptick}-eq-iff s [simp] =$
 $ev-eq-map-event_{ptick}-iff tick-eq-map-event_{ptick}-iff$
 $map-event_{ptick}-eq-ev-iff map-event_{ptick}-eq-tick-iff$
have $length-ge-eq-pl-\tau-imp-eq : \langle r = r' \rangle$
if $\langle n \leq length r \rangle$ **and** $\langle ?pl-\tau i j r = ?pl-\tau i j r' \rangle$ **for** $r r' :: \langle 'r list \rangle$

proof –

from $\langle n \leq length r \rangle$ **have** $\langle n \leq length (?pl-\tau i j r) \rangle$ **by** *simp*
with $\langle ?pl-\tau i j r = ?pl-\tau i j r' \rangle$ **have** $\langle n \leq length r' \rangle$ **by** *simp*
have $\langle inj-on (?pl-\tau i j) \{r. n \leq length r\} \rangle$
by (*simp add: $\langle i < n \rangle \langle j < n \rangle inj-on-permute-list-transpose$*)
with $\langle n \leq length r \rangle \langle n \leq length r' \rangle \langle ?pl-\tau i j r = ?pl-\tau i j r' \rangle$
show $\langle r = r' \rangle$ **by** (*auto dest: inj-onD*)

qed

have $pl-\tau-pl-\tau : \langle n \leq length r \implies ?pl-\tau i j (?pl-\tau i j r) = r \rangle$ **for** $r :: \langle 'r list \rangle$
by (*subst permute-list-compose[symmetric]*)
 $(metis lessThan-iff order-less-le-trans permutes-swap-id \langle i < n \rangle \langle j < n \rangle, simp)$

have *fun-evt-fun-evt* :

$\langle (case e of ev a \Rightarrow True \mid \checkmark(r) \Rightarrow n \leq length r) \implies ?fun-evt i j (?fun-evt i j$
 $e) = e \rangle$

for $e :: \langle ('a, 'r list) event_{ptick} \rangle$
by (*cases e*) (*simp-all add: pl-\tau-pl-\tau*)

have *map-evt-map-evt* :

$\langle (\bigwedge e. e \in set t \implies case e of ev a \Rightarrow True \mid \checkmark(r) \Rightarrow n \leq length r) \implies ?map-evt i j (?map-evt i j t) = t \rangle$ **for** $t :: \langle ('a, 'r list) trace_{ptick} \rangle$
by (*induct t*) (*simp-all add: fun-evt-fun-evt*)

from $\langle i < n \rangle \langle j < n \rangle$ **have** *pl-\tau-Cons* :

$\langle n \leq length s \implies ?pl-\tau (Suc i) (Suc j) (r \# s) = r \# ?pl-\tau i j s \rangle$ **for** r **and** s
 $:: \langle 'r list \rangle$

by (*simp add: list-update-append1 nth-append-left permute-list-transpose-eq-list-update*)

show $\langle ?lhs = ?rhs \rangle$

proof (*rule Process-eq-optimizedI*)

fix t **assume** $\langle t \in \mathcal{D} \ ?lhs \rangle$

then obtain $u v t-P t-Q$ **where** $*$: $\langle t = u @ v \rangle \langle tF u \rangle \langle ftF v \rangle$

$\langle u \text{ setinterleaves}_{?tj} ((t-P, t-Q), S) \rangle$

$\langle t-P \in \mathcal{D} P \wedge t-Q \in \mathcal{T} (?RT i j Q) \vee t-P \in \mathcal{T} P \wedge t-Q \in \mathcal{D} (?RT i j Q) \rangle$

unfolding *Sync_{Rlist}.D-Sync_{ptick}* **by** *blast*

```

    from tickFree-setinterleavesptick-iff[THEN iffD1, OF *(4, 2)] have ⟨tF t-Q⟩
  ..
  with *(5) have ⟨t-P ∈ D P ∧ t-Q ∈ T Q ∨ t-P ∈ T P ∧ t-Q ∈ D Q⟩
  by (simp add: tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-D)
  with *(1-4) have ⟨t ∈ D (P [S]✓Rlist Q)⟩
  by (auto simp add: SyncRlist.D-Syncptick)
  thus ⟨t ∈ D ?rhs⟩
  by (meson D-imp-front-tickFree div-butlast-when-non-tickFree-iff
    front-tickFree-iff-tickFree-butlast tickFree-mem-D-RenamingTick-iff-mem-D)
next
  fix t assume ⟨t ∈ D ?rhs⟩
  then obtain t1 t2
    where * : ⟨t = ?map-evt (Suc i) (Suc j) t1 @ t2⟩ ⟨tF t1⟩ ⟨ftF t2⟩ ⟨t1 ∈ D
(P [S]✓Rlist Q)⟩
    by (auto simp add: D-Renaming)
  from *(1, 2) have ⟨t = t1 @ t2⟩
  by (simp add: tickFree-map-map-eventptick-id-eq)
  from *(4) obtain u1 u2 t-P t-Q where ** : ⟨t1 = u1 @ u2⟩ ⟨tF u1⟩ ⟨ftF u2⟩
    ⟨u1 setinterleaves✓?tj ((t-P, t-Q), S)⟩
    ⟨t-P ∈ D P ∧ t-Q ∈ T Q ∨ t-P ∈ T P ∧ t-Q ∈ D Q⟩
  unfolding SyncRlist.D-Syncptick by blast
  from tickFree-setinterleavesptick-iff[THEN iffD1, OF ***(4, 2)] have ⟨tF t-Q⟩
  ..
  with ***(5) have ⟨t-P ∈ D P ∧ t-Q ∈ T (?RT i j Q) ∨ t-P ∈ T P ∧ t-Q ∈ D
(?RT i j Q)⟩
  by (simp-all add: tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-D)
  with ***(1-4) *(2, 3) ⟨t = t1 @ t2⟩ show ⟨t ∈ D ?lhs⟩
  by (auto simp add: SyncRlist.D-Syncptick intro: front-tickFree-append)
next
  fix t X assume ⟨(t, X) ∈ F ?rhs⟩ ⟨t ∉ D ?rhs⟩ ⟨t ∉ D ?lhs⟩
  then obtain t' where * : ⟨t = ?map-evt (Suc i) (Suc j) t'⟩
    ⟨(t', ?fun-evt (Suc i) (Suc j) - ' X) ∈ F (P [S]✓Rlist Q)⟩
  unfolding Renaming-projs by blast
  from *(2) ⟨t ∉ D ?rhs⟩ have ⟨t' ∉ D (P [S]✓Rlist Q)⟩
  by (metis (no-types, lifting) *(1) D-imp-front-tickFree div-butlast-when-non-tickFree-iff
    front-tickFree-iff-tickFree-butlast map-butlast map-eventptick-tickFree
    tickFree-map-map-eventptick-id-eq tickFree-mem-D-RenamingTick-iff-mem-D)
  with *(2) obtain t-P t-Q X-P X-Q
    where ** : ⟨(t-P, X-P) ∈ F P⟩ ⟨(t-Q, X-Q) ∈ F Q⟩
    ⟨t' setinterleaves✓?tj ((t-P, t-Q), S)⟩
    ⟨?fun-evt (Suc i) (Suc j) - ' X ⊆ super-ref-Syncptick ?tj X-P S X-Q⟩
  unfolding SyncRlist.Syncptick-projs by force
  from *(2) consider ⟨tF t'⟩ | t'' rs where ⟨tF t''⟩ ⟨t' = t'' @ [✓(rs)]⟩
  by (metis (lifting) F-T F-imp-front-tickFree T-nonTickFree-imp-decomp
    butlast-snoc front-tickFree-iff-tickFree-butlast)
  thus ⟨(t, X) ∈ F ?lhs⟩
proof cases
  assume ⟨tF t'⟩
  have ⟨?map-evt (Suc i) (Suc j) t' = t'⟩

```

by (simp add: $\langle tF\ t' \rangle$ tickFree-map-map-event_{ptick-id-eq})
 have $\langle ?map\text{-}evt\ i\ j\ t\text{-}Q = t\text{-}Q \rangle$
 using $**(\beta)$ $\langle tF\ t' \rangle$ tickFree-map-map-event_{ptick-id-eq} tickFree-setinterleaves_{ptick-iff}
 by blast
 define $X\text{-}Q'$ where $\langle X\text{-}Q' \equiv X\text{-}Q \cap (\text{range}\ ev \cup \{\checkmark(r) \mid r. n \leq \text{length}\ } r\} \rangle$
 define X' where $\langle X' \equiv X \cap (\text{range}\ ev \cup \{\checkmark(rs) \mid rs. \text{Suc}\ } n \leq \text{length}\ } rs\} \rangle$
 have $\langle X\text{-}Q' \subseteq X\text{-}Q \rangle$ unfolding $X\text{-}Q'\text{-}def$ by blast
 with $**(\beta)$ is-processT4 have $\langle (t\text{-}Q, X\text{-}Q') \in \mathcal{F}\ Q \rangle$ by blast
 moreover have $\langle ?fun\text{-}evt\ i\ j - ' (?fun\text{-}evt\ i\ j\ ' X\text{-}Q') = X\text{-}Q' \rangle$
 by (auto simp add: $X\text{-}Q'\text{-}def$)
 (use length-ge-eq-pl- τ -imp-eq in blast)+
 moreover have $\langle ?map\text{-}evt\ i\ j\ t\text{-}Q = t\text{-}Q \rangle$ by fact
 ultimately have $\langle (t\text{-}Q, ?fun\text{-}evt\ i\ j\ ' X\text{-}Q') \in \mathcal{F}\ (?RT\ i\ j\ Q) \rangle$
 by (auto simp add: F-Renaming)
 moreover have $\langle e \in X' \implies e \in \text{super-ref-Sync}_{ptick}\ ?tj\ X\text{-}P\ S\ (?fun\text{-}evt\ i\ j\ ' X\text{-}Q') \rangle$ for e
 using $**(\beta)$ [THEN set-mp, of $\langle ?fun\text{-}evt\ (\text{Suc}\ i)\ (\text{Suc}\ j)\ e \rangle$]
 unfolding $X'\text{-}def\ X\text{-}Q'\text{-}def\ \text{super-ref-Sync}_{ptick}\text{-}def$
 by (auto simp add: image-iff pl- τ -Cons) (use pl- τ -pl- τ in force)
 ultimately have $\langle (t, X') \in \mathcal{F}\ ?lhs \rangle$
 by (simp add: Sync_{Rlist}.F-Sync_{ptick})
 (metis $*(1)$ $**(\beta)$ $*(1, \beta)$ $\langle ?map\text{-}evt\ (\text{Suc}\ i)\ (\text{Suc}\ j)\ t' = t' \rangle$ subsetI)
 moreover have $\langle \text{Suc}\ n \leq \text{length}\ (rs) \rangle$ if $\langle t @ [\checkmark(rs)] \in \mathcal{T}\ ?lhs \rangle$ for rs
 proof -
 from $\langle t \notin \mathcal{D}\ ?lhs \rangle$ have $\langle t @ [\checkmark(rs)] \notin \mathcal{D}\ ?lhs \rangle$
 by (meson is-processT9)
 with $\langle t @ [\checkmark(rs)] \in \mathcal{T}\ ?lhs \rangle$
 obtain $t\text{-}P''\ t\text{-}Q''$ where $\mathcal{L} : \langle t\text{-}P'' \in \mathcal{T}\ P \rangle \langle t\text{-}Q'' \in \mathcal{T}\ (?RT\ i\ j\ Q) \rangle$
 $\langle t @ [\checkmark(rs)]\ \text{setinterleaves}_{\checkmark\lambda r\ s. [r \# s]} ((t\text{-}P'', t\text{-}Q''), S) \rangle$
 unfolding Sync_{Rlist}.Sync_{ptick}-projs by blast
 from $\mathcal{L}$ obtain $t\text{-}P''' t\text{-}Q''' r\ s$
 where $\mathcal{L}\mathcal{L} : \langle rs = r \# s \rangle \langle t\text{-}P'' = t\text{-}P''' @ [\checkmark(r)] \rangle \langle t\text{-}Q'' = t\text{-}Q''' @ [\checkmark(s)] \rangle$
 $\langle t\ \text{setinterleaves}_{\checkmark\lambda r\ s. [r \# s]} ((t\text{-}P''', t\text{-}Q'''), S) \rangle$
 by (auto elim: snoc-tick-setinterleaves_{ptick}E)
 have $\langle tF\ t\text{-}Q''' \rangle$ using $\mathcal{L}(2)\ \mathcal{L}\mathcal{L}(3)$ append-T-imp-tickFree by blast
 from $\langle t \notin \mathcal{D}\ ?lhs \rangle\ \mathcal{L}(1)\ \mathcal{L}\mathcal{L}(2, 4)$ have $\langle t\text{-}Q''' \notin \mathcal{D}\ (?RT\ i\ j\ Q) \rangle$
 by (simp add: Sync_{Rlist}.D-Sync_{ptick})
 (use front-tickFree-Nil is-processT3-TR-append in blast)
 with $\mathcal{L}(2)$ obtain $t\text{-}Q''''$
 where $\langle ?map\text{-}evt\ i\ j\ t\text{-}Q'''' = t\text{-}Q''' @ [\checkmark(s)] \rangle \langle t\text{-}Q'''' \in \mathcal{T}\ Q \rangle$
 by (simp add: Renaming-projs)
 (metis $\mathcal{L}\mathcal{L}(3)$ $\langle t\text{-}Q''' \notin \mathcal{D}\ (?RT\ i\ j\ Q) \rangle$ is-processT7 is-processT9
 tickFree-map-map-event_{ptick-id-eq} tickFree-mem-D-RenamingTick-iff-mem-D)
 then obtain s' where $\mathcal{L}\mathcal{L}\mathcal{L} : \langle s = ?pl\text{-}\tau\ i\ j\ s' \rangle \langle t\text{-}Q''' @ [\checkmark(s')] \in \mathcal{T}\ Q \rangle$
 by (auto simp add: map-eq-append-conv Cons-eq-map-conv
 append-T-imp-tickFree tickFree-map-map-event_{ptick-id-eq})
 have $\langle s' \in \checkmark s(Q) \rangle$
 by (meson $\mathcal{L}\mathcal{L}\mathcal{L}(2)$ $\langle tF\ t\text{-}Q''' \rangle \langle t\text{-}Q''' \notin \mathcal{D}\ (?RT\ i\ j\ Q) \rangle$ is-processT9
 strict-ticks-of-memI tickFree-mem-D-RenamingTick-iff-mem-D)

```

    with  $\langle \bigwedge rs. rs \in \check{\mathbf{s}}(Q) \implies n \leq \text{length } rs \rangle$  have  $\langle n \leq \text{length } s' \rangle$  .
    with  $\langle s = ?pl\text{-}\tau \ i \ j \ s' \rangle$  have  $\langle n \leq \text{length } s \rangle$  by simp
    with  $\langle rs = r \# s \rangle$  show  $\langle \text{Suc } n \leq \text{length } rs \rangle$  by simp
qed
ultimately have  $\langle (t, X' \cup (X \cap \{\check{\mathbf{s}}(rs) \mid rs. \neg \text{Suc } n \leq \text{length } rs\})) \in \mathcal{F} \ ?lhs \rangle$ 
  using is-processT5-S7' by blast
  also have  $\langle X' \cup (X \cap \{\check{\mathbf{s}}(rs) \mid rs. \neg \text{Suc } n \leq \text{length } rs\}) = X \rangle$ 
    by (simp add: set-eq-iff X'-def image-iff) (meson eventptick.exhaust)
  finally show  $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$  .
next
fix  $t''$   $rs$  assume  $\langle tF \ t'' \rangle \ \langle t' = t'' @ [\check{\mathbf{s}}(rs)] \rangle$ 
from  $**(3)$  obtain  $t\text{-}P' \ t\text{-}Q' \ r \ s$ 
  where  $*** : \langle r \# s = rs \rangle$ 
     $\langle t'' \text{ setinterleaves}_{\check{\mathbf{s}} \ ?tj} ((t\text{-}P', t\text{-}Q'), S) \rangle$ 
     $\langle t\text{-}P = t\text{-}P' @ [\check{\mathbf{s}}(r)] \rangle \ \langle t\text{-}Q = t\text{-}Q' @ [\check{\mathbf{s}}(s)] \rangle$ 
  by (auto elim: snoc-tick-setinterleavesptickE
    simp add:  $\langle t' = t'' @ [\check{\mathbf{s}}(rs)] \rangle$  split: if-split-asm)
have  $\langle n \leq \text{length } s \rangle$ 
proof -
  from  $**(1)[\text{THEN } F\text{-}T] \ ** (3) \ \langle t' \notin \mathcal{D} (P \llbracket S \rrbracket_{\check{\mathbf{s}} Rlist} Q) \rangle$  have  $\langle t\text{-}Q \notin \mathcal{D} \ Q \rangle$ 
    by (simp add: SyncRlist.Syncptick-projs')
    (use front-tickFree-Nil in blast)
  with  $\langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle [\text{THEN } F\text{-}T]$  have  $\langle s \in \check{\mathbf{s}}(Q) \rangle$ 
    by (simp add:  $*** (4)$  strict-ticks-of-memI)
  with  $\langle \bigwedge rs. rs \in \check{\mathbf{s}}(Q) \implies n \leq \text{length } rs \rangle$  show  $\langle n \leq \text{length } s \rangle$  .
qed
from  $\langle t'' \text{ setinterleaves}_{\check{\mathbf{s}} \ ?tj} ((t\text{-}P', t\text{-}Q'), S) \rangle$ 
have  $\langle ?map\text{-}evt \ (\text{Suc } i) \ (\text{Suc } j) \ t'' \text{ setinterleaves}_{\check{\mathbf{s}} \ ?tj} ((t\text{-}P', ?map\text{-}evt \ i \ j \ t\text{-}Q'), S) \rangle$ 
  by (metis (no-types, lifting)  $\langle tF \ t'' \rangle$  tickFree-map-map-eventptick-id-eq
    tickFree-setinterleavesptick-iff)
from setinterleavesptick-snoc-tick
  [OF this, of  $r \ \langle ?pl\text{-}\tau \ i \ j \ s \rangle \ \langle ?pl\text{-}\tau \ (\text{Suc } i) \ (\text{Suc } j) \ rs \rangle \ \langle n \leq \text{length } s \rangle$ ]
have  $\langle ?map\text{-}evt \ (\text{Suc } i) \ (\text{Suc } j) \ t' \text{ setinterleaves}_{\check{\mathbf{s}} \ ?tj} ((t\text{-}P, ?map\text{-}evt \ i \ j \ t\text{-}Q), S) \rangle$ 
  by (simp add:  $*** (1, 3, 4) \ \langle t' = t'' @ [\check{\mathbf{s}}(rs)] \rangle$ ) (metis  $*** (1) \ pl\text{-}\tau\text{-Cons}$ )
moreover from  $**(2)[\text{THEN } F\text{-}T]$  have  $\langle (?map\text{-}evt \ i \ j \ t\text{-}Q, UNIV) \in \mathcal{F} \ (\text{?RT } i \ j \ Q) \rangle$ 
  by (simp add:  $*** (4)$ , intro tick-T-F) (auto simp add: T-Renaming)
moreover have  $\langle (t\text{-}P, UNIV) \in \mathcal{F} \ P \rangle$ 
  by (metis  $*** (1) \ *** (3) \ F\text{-}T \ tick\text{-}T\text{-}F$ )
moreover have  $\langle e \in X \implies e \in \text{super-ref-Sync}_{ptick} \ ?tj \ UNIV \ S \ UNIV \rangle$  for
 $e$ 
  using  $*** (4)[\text{THEN } \text{set-mp, of } \langle ?fun\text{-}evt \ (\text{Suc } i) \ (\text{Suc } j) \ e \rangle]$ 
  by (cases e) (auto simp add: super-ref-Syncptick-def)
ultimately show  $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$ 
  using  $*(1)$  by (simp add: SyncRlist.F-Syncptick) blast
qed
next

```

fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle \langle t \notin \mathcal{D} \ ?lhs \rangle$
then obtain $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$
where $*$: $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ (\text{?RT } i\ j\ Q) \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark} \text{?tj} ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{ptick} \text{?tj } X\text{-}P\ S\ X\text{-}Q \rangle$
unfolding $\text{Sync}_{Rlist}.\text{Sync}_{ptick}\text{-projs}$ **by** *force*
from $*(1, 3) \langle t \notin \mathcal{D} \ ?lhs \rangle F\text{-}T\ \text{front-tickFree-}\text{Nil}$
have $\langle t\text{-}Q \notin \mathcal{D} \ (\text{?RT } i\ j\ Q) \rangle$ **unfolding** $\text{Sync}_{Rlist}.\text{D-Sync}_{ptick}'$ **by** *blast*
with $*(2)$ **obtain** $t\text{-}Q'$ **where** $**$: $\langle t\text{-}Q = \text{?map-evt } i\ j\ t\text{-}Q' \rangle \langle (t\text{-}Q', \text{?fun-evt } i\ j - 'X\text{-}Q) \in \mathcal{F} \ Q \rangle$
unfolding *Renaming-projs* **by** *blast*
define $X\text{-}Q'$ **where** $\langle X\text{-}Q' \equiv X\text{-}Q \cap (\text{range } ev \cup \{\checkmark(rs) \mid rs. n \leq \text{length } rs\}) \rangle$
define X' **where** $\langle X' \equiv X \cap (\text{range } ev \cup \{\checkmark(rs) \mid rs. \text{Suc } n \leq \text{length } rs\}) \rangle$

from $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle [THEN\ F\text{-}T]$ **consider** $\langle tF\ t \rangle \mid t' \text{ } rs$ **where** $\langle tF\ t' \rangle \langle t = t' @ [\checkmark(rs)] \rangle$
using *T-nonTickFree-imp-decomp append-T-imp-tickFree* **by** *blast*
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
proof cases
assume $\langle tF\ t \rangle$
hence $\langle \text{?map-evt } (\text{Suc } i) (\text{Suc } j) \ t = t \rangle$
by (*simp add: tickFree-map-map-event_{ptick}-id-eq*)
have $\langle \text{?map-evt } i\ j\ t\text{-}Q' = t\text{-}Q' \rangle$
using $*(3) \text{?}(1) \langle tF\ t \rangle \text{map-event}_{ptick}\text{-tickFree tickFree-map-map-event}_{ptick}\text{-id-eq}$
 $\text{tickFree-setinterleaves}_{ptick}\text{-iff}$ **by** *blast*
have $\langle (t\text{-}Q, \text{?fun-evt } i\ j - 'X\text{-}Q) \in \mathcal{F} \ Q \rangle$
by (*simp add: ** (1, 2) \langle \text{?map-evt } i\ j\ t\text{-}Q' = t\text{-}Q' \rangle*)
hence $\langle (t\text{-}Q, \text{?fun-evt } i\ j - 'X\text{-}Q') \in \mathcal{F} \ Q \rangle$
by (*simp add: X\text{-}Q'\text{-def is-processT4*)
moreover have $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle$ **by** (*fact *(1)*)
moreover have $\langle t \text{ setinterleaves}_{\checkmark} \text{?tj} ((t\text{-}P, t\text{-}Q), S) \rangle$ **by** (*fact *(3)*)
moreover have $\langle e \in \text{?fun-evt } (\text{Suc } i) (\text{Suc } j) - 'X' \implies$
 $e \in \text{super-ref-Sync}_{ptick} \text{?tj } X\text{-}P\ S\ (\text{?fun-evt } i\ j - 'X\text{-}Q) \rangle$ **for** e
using *set-mp[OF *(4), of \langle \text{?fun-evt } (\text{Suc } i) (\text{Suc } j) \ e \rangle]*
by (*auto simp add: super-ref-Sync_{ptick}\text{-def } X'\text{-def pl-}\tau\text{-Cons}*)
ultimately have $\langle (t, \text{?fun-evt } (\text{Suc } i) (\text{Suc } j) - 'X') \in \mathcal{F} \ (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \rangle$
by (*unfold Sync_{Rlist}.F-Sync_{ptick}, clarify*)
 $(metis ** (1, 2) \langle \text{?map-evt } i\ j\ t\text{-}Q' = t\text{-}Q' \rangle \text{subsetI})$
moreover have $\langle \text{Suc } n \leq \text{length } (rs) \rangle$ **if** $\langle t @ [\checkmark(rs)] \in \mathcal{T} \ (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \rangle$
for rs
proof –
from $\langle t \notin \mathcal{D} \ ?lhs \rangle$ **have** $\langle t \notin \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \rangle$
by (*simp add: Sync_{Rlist}.D-Sync_{ptick}*)
 $(metis \text{tickFree-mem-D-RenamingTick-iff-mem-D}$
 $\text{tickFree-mem-T-RenamingTick-iff-mem-T tickFree-setinterleaves}_{ptick}\text{-iff})$
hence $\langle t @ [\checkmark(rs)] \notin \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \rangle$
by (*meson is-processT9*)
with $\langle t @ [\checkmark(rs)] \in \mathcal{T} \ (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \rangle$
obtain $t\text{-}P''\ t\text{-}Q''$ **where** $\mathcal{L} : \langle t\text{-}P'' \in \mathcal{T} \ P \rangle \langle t\text{-}Q'' \in \mathcal{T} \ Q \rangle$

$\langle t @ [\checkmark(rs)] \text{ setinterleaves}_{\checkmark \lambda r s. [r \# s]} ((t-P'', t-Q''), S) \rangle$
unfolding $\text{Sync}_{Rlist}.\text{Sync}_{ptick}\text{-projs}$ **by** *blast*
from $\mathcal{L}$ **obtain** $t-P''' \ t-Q''' \ r \ s$
where $\mathcal{L} \mathcal{L} : \langle rs = r \# s \rangle \langle t-P'' = t-P''' @ [\checkmark(r)] \rangle \langle t-Q'' = t-Q''' @ [\checkmark(s)] \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark \lambda r s. [r \# s]} ((t-P''', t-Q'''), S) \rangle$
by (*auto elim: snoc-tick-setinterleaves_{ptick}E*)
from $\langle t \notin \mathcal{D} (P \llbracket S \rrbracket_{\checkmark Rlist} Q) \rangle$ **have** $\langle t-Q'' \notin \mathcal{D} Q \rangle$
by (*simp add: Sync_{Rlist}.D-Sync_{ptick}*)
(metis $\mathcal{L}(1)$ $\mathcal{L} \mathcal{L}(2-4)$ append.right-neutral
front-tickFree-Nil is-processT3-TR-append is-processT9)
have $\langle s \in \checkmark s(Q) \rangle$
by (*metis $\mathcal{L}(2)$ $\mathcal{L} \mathcal{L}(3)$ $\langle t-Q'' \notin \mathcal{D} Q \rangle$ strict-ticks-of-memI*)
with $\langle \bigwedge rs. rs \in \checkmark s(Q) \implies n \leq \text{length } rs \rangle$ **have** $\langle n \leq \text{length } s \rangle$.
thus $\langle \text{Suc } n \leq \text{length } rs \rangle$ **by** (*simp add: $\langle rs = r \# s \rangle$*)
qed
ultimately have $\langle (t, ?\text{fun-evt } (Suc \ i) \ (Suc \ j) - 'X' \cup$
 $?\text{fun-evt } (Suc \ i) \ (Suc \ j) - '$
 $(X \cap \{\checkmark(rs) \mid rs. \neg \text{Suc } n \leq \text{length } rs\}) \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark Rlist}$
 $Q) \rangle$
using *is-processT5-S7'* **by** *force*
also have $\langle ?\text{fun-evt } (Suc \ i) \ (Suc \ j) - 'X' \cup$
 $?\text{fun-evt } (Suc \ i) \ (Suc \ j) - '(X \cap \{\checkmark(rs) \mid rs. \neg \text{Suc } n \leq \text{length } rs\}) =$
 $?\text{fun-evt } (Suc \ i) \ (Suc \ j) - 'X \rangle$
by (*auto simp add: X'-def image-iff*) (*metis event_{ptick}.exhaust*)
finally show $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
using $\langle ?\text{map-evt } (Suc \ i) \ (Suc \ j) \ t = t \rangle$ **by** (*auto simp add: F-Renaming*)
next
fix $t' \ rs$ **assume** $\langle tF \ t' \rangle \langle t = t' @ [\checkmark(rs)] \rangle$
from $*(3)$ **obtain** $t-P'' \ t-Q'' \ r \ s$
where $*** : \langle rs = r \# s \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark ?tj} ((t-P'', t-Q''), S) \rangle$
 $\langle t-P = t-P'' @ [\checkmark(r)] \rangle \langle t-Q = t-Q'' @ [\checkmark(s)] \rangle$
 $\langle tF \ t-P'' \rangle \langle tF \ t-Q'' \rangle$
by (*auto elim!: snoc-tick-setinterleaves_{ptick}E*
simp add: $\langle t = t' @ [\checkmark(rs)] \rangle$ split: if-split-asm)
(metis $\langle tF \ t' \rangle$ tickFree-setinterleaves_{ptick}-iff)
have $\langle t-Q'' \notin \mathcal{D} Q \rangle$
by (*metis $*(2)$ $***(4)$ F-imp-front-tickFree $\langle t-Q \notin \mathcal{D} (?RT \ i \ j \ Q) \rangle$*
front-tickFree-append-iff is-processT7 non-tickFree-tick
tickFree-Nil tickFree-mem-D-RenamingTick-iff-mem-D)
from $*(1) \ *** (4)$ **obtain** s' **where** $\langle s = ?pl-\tau \ i \ j \ s' \rangle$
by (*auto simp add: $*(2)$ append-eq-map-conv Cons-eq-map-conv*)
with $*(1) \ ** (2) [THEN \ F-T] \ *** (4)$ **have** $\langle t-Q'' @ [\checkmark(s')] \in \mathcal{T} \ Q \rangle$
by (*simp add: $*** (4)$ append-eq-map-conv Cons-eq-map-conv*)
*(metis $*** (6)$ $\langle t-Q'' \notin \mathcal{D} Q \rangle$ is-processT9 length-permute-list map-event_{ptick}-tickFree*
pl-\tau-pl-\tau strict-ticks-of-memI that(3) tickFree-map-map-event_{ptick}-id-eq)
with $\langle t-Q'' \notin \mathcal{D} Q \rangle$ **have** $\langle s' \in \checkmark s(Q) \rangle$
by (*metis is-processT9 strict-ticks-of-memI*)
with $\langle \bigwedge rs. rs \in \checkmark s(Q) \implies n \leq \text{length } rs \rangle$ **have** $\langle n \leq \text{length } s' \rangle$.

with $\langle s = ?pl\text{-}\tau \ i \ j \ s' \rangle$ **have** $\langle n \leq \text{length } s \rangle$ **by** *simp*
hence $\langle \text{Suc } n \leq \text{length } rs \rangle$ **by** (*simp add: $\langle rs = r \# s \rangle$*)

have $\langle ?map\text{-}evt \ (Suc \ i) \ (Suc \ j) \ t \ \text{setinterleaves}_{?tj} \ ((t\text{-}P, \ ?map\text{-}evt \ i \ j \ t\text{-}Q),$
 $S) \rangle$
by (*simp add: $*** (1-3, 4, 6) \ \langle t = t' @ [\checkmark(rs)] \rangle \ \langle n \leq \text{length } s \rangle \ \langle tF \ t' \rangle$*)
pl- τ -Cons
setinterleaves_{ptick}-snoc-tick tickFree-map-map-event_{ptick}-id-eq
moreover **have** $\langle t\text{-}P \in \mathcal{T} \ P \rangle$ **using** $*(1) \ F\text{-}T$ **by** *blast*
moreover **from** $*(2)[\text{THEN } F\text{-}T] \ \langle n \leq \text{length } s \rangle$ **have** $\langle ?map\text{-}evt \ i \ j \ t\text{-}Q \in$
 $\mathcal{T} \ Q \rangle$
by (*auto simp add: T-Renaming $*** (4, 6) \ \text{append-eq-map-conv Cons-eq-map-conv}$*
tickFree-map-map-event_{ptick}-id-eq pl- τ -pl- τ)
(metis append-T-imp-tickFree not-Cons-self2 tickFree-map-map-event_{ptick}-id-eq,
metis $\langle t\text{-}Q'' \notin \mathcal{D} \ Q \rangle \ \text{is-process } T7 \ \text{is-process } T9$)
ultimately **have** $\langle ?map\text{-}evt \ (Suc \ i) \ (Suc \ j) \ t \in \mathcal{T} \ (P \llbracket S \rrbracket_{\checkmark Rlist} \ Q) \rangle$
by (*auto simp add: Sync_{Rlist}.T-Sync_{ptick}*)
with $\langle n \leq \text{length } s \rangle$ **have** $\langle t \in \mathcal{T} \ ?rhs \rangle$
by (*auto simp add: T-Renaming $\langle t = t' @ [\checkmark(rs)] \rangle \ \text{append-eq-map-conv}$*
Cons-eq-map-conv)
*(metis $*** (1) \ \langle tF \ t' \rangle \ \text{length-permute-list pl-}\tau\text{-Cons pl-}\tau\text{-pl-}\tau \ \text{tickFree-map-map-event}_{ptick}\text{-id-eq}$)*
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$ **by** (*simp add: $\langle t = t' @ [\checkmark(rs)] \rangle \ \text{tick-T-F}$*)
qed
qed
qed

lemma *RenamingTick-permute-list-transpose-Sync_{ListstlenL}* :
 $\langle \text{RenamingTick } P \ (\text{permute-list} \ (\text{Transposition.transpose } i \ j)) \ n \llbracket S \rrbracket_{\checkmark ListstlenL} \ Q$
 $=$
 $\text{RenamingTick} \ (P \ n \llbracket S \rrbracket_{\checkmark ListstlenL} \ Q) \ (\text{permute-list} \ (\text{Transposition.transpose } i \ j)) \rangle$
 $(\text{is } \langle ?lhs = ?rhs \rangle) \text{ if } \langle i < n \rangle \ \langle j < n \rangle \text{ for } P :: \langle ('a, 'r \text{ list}) \ \text{process}_{ptick} \rangle$
proof –
let $?pl = \langle \text{permute-list} \ (\text{Transposition.transpose } i \ j) \rangle$
let $?fun\text{-}evt = \langle \text{map-event}_{ptick} \ id \ (\text{permute-list} \ (\text{Transposition.transpose } i \ j)) \rangle$
let $?map\text{-}evt = \langle \text{map} \ ?fun\text{-}evt \rangle$
and $?RT = \langle \lambda P. \ \text{RenamingTick } P \ (\text{permute-list} \ (\text{Transposition.transpose } i \ j)) \rangle$
and $?tj = \langle \lambda r \ s. \ \text{if } \text{length } r = n \text{ then } [r @ s] \ \text{else } \Diamond \rangle$
note $\text{map-event}_{ptick}\text{-eq-iffs} \ [simp] =$
 $\text{ev-eq-map-event}_{ptick}\text{-iff tick-eq-map-event}_{ptick}\text{-iff map-event}_{ptick}\text{-eq-ev-iff map-event}_{ptick}\text{-eq-tick-iff}$
have $\text{length-eq-pl-imp} : \langle r = r' \rangle \text{ if } \langle n \leq \text{length } r \rangle \text{ and } \langle ?pl \ r = ?pl \ r' \rangle \text{ for } r \ r'$
 $:: \langle 'r \text{ list} \rangle$
proof –
from $\langle n \leq \text{length } r \rangle$ **have** $\langle n \leq \text{length } (?pl \ r) \rangle$ **by** *simp*
with $\langle ?pl \ r = ?pl \ r' \rangle$ **have** $\langle n \leq \text{length } r' \rangle$ **by** *simp*
have $\langle \text{inj-on } ?pl \ \{r. \ n \leq \text{length } r\} \rangle$
by (*simp add: $\langle i < n \rangle \ \langle j < n \rangle \ \text{inj-on-permute-list-transpose}$*)

```

with  $\langle n \leq \text{length } r \rangle \langle n \leq \text{length } r' \rangle \langle ?pl \ r = ?pl \ r' \rangle$ 
show  $\langle r = r' \rangle$  by (auto dest: inj-onD)
qed
have  $pl\text{-}pl : \langle n \leq \text{length } r \implies ?pl \ ( ?pl \ r) = r \rangle$  for  $r :: \langle 'r \text{ list} \rangle$ 
by (subst permute-list-compose[symmetric])
      (metis lessThan-iff order-less-le-trans permutes-swap-id  $\langle i < n \rangle \langle j < n \rangle$ , simp)
have fun-evt-fun-evt :
   $\langle \text{case } e \text{ of } ev \ a \Rightarrow \text{True} \mid \checkmark(r) \Rightarrow n \leq \text{length } r \rangle \implies ?fun\text{-}evt \ ( ?fun\text{-}evt \ e) = e \rangle$ 
for  $e :: \langle ('a, 'r \text{ list}) \text{ event}_{ptick} \rangle$ 
by (cases e) (simp-all add: pl-pl)
have map-evt-map-evt :
   $\langle (\bigwedge e. e \in \text{set } t \implies \text{case } e \text{ of } ev \ a \Rightarrow \text{True} \mid \checkmark(r) \Rightarrow n \leq \text{length } r) \implies$ 
     $\implies ?map\text{-}evt \ ( ?map\text{-}evt \ t) = t \rangle$  for  $t :: \langle ('a, 'r \text{ list}) \text{ trace}_{ptick} \rangle$ 
by (induct t) (simp-all add: fun-evt-fun-evt)
from  $\langle i < n \rangle \langle j < n \rangle$  have pl-append :
   $\langle n \leq \text{length } r \implies ?pl \ (r @ r') = ?pl \ r @ r' \rangle$  for  $r \ r' :: \langle 'r \text{ list} \rangle$ 
by (simp add: list-update-append1 nth-append-left permute-list-transpose-eq-list-update)

show  $\langle ?lhs = ?rhs \rangle$ 
proof (rule Process-eq-optimizedI)
  fix  $t$  assume  $\langle t \in \mathcal{D} \ ?lhs \rangle$ 
  then obtain  $u \ v \ t\text{-}P \ t\text{-}Q$  where  $*$  :  $\langle t = u @ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle$ 
     $\langle u \text{ setinterleaves}_{?tj} ((t\text{-}P, t\text{-}Q), S) \rangle$ 
     $\langle t\text{-}P \in \mathcal{D} \ ( ?RT \ P) \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ ( ?RT \ P) \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$ 
    unfolding SyncListslenL.D-Syncptick by blast
from tickFree-setinterleavesptick-iff [THEN iffD1, OF  $*(4, 2)$ ] have  $\langle tF \ t\text{-}P \rangle$ 
..
with  $*(5)$  have  $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$ 
by (simp-all add: tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-
with  $*(1-4)$  have  $\langle t \in \mathcal{D} \ (P \ n \llbracket S \rrbracket_{\checkmark ListslenL} \ Q) \rangle$ 
by (auto simp add: SyncListslenL.D-Syncptick)
thus  $\langle t \in \mathcal{D} \ ?rhs \rangle$ 
by (meson D-imp-front-tickFree div-butlast-when-non-tickFree-iff
      front-tickFree-iff-tickFree-butlast tickFree-mem-D-RenamingTick-iff-mem-D)
next
  fix  $t$  assume  $\langle t \in \mathcal{D} \ ?rhs \rangle$ 
  then obtain  $t1 \ t2$ 
    where  $*$  :  $\langle t = ?map\text{-}evt \ t1 @ t2 \rangle \langle tF \ t1 \rangle \langle ftF \ t2 \rangle \langle t1 \in \mathcal{D} \ (P \ n \llbracket S \rrbracket_{\checkmark ListslenL} \ Q) \rangle$ 
by (auto simp add: D-Renaming)
from  $*(1, 2)$  have  $\langle t = t1 @ t2 \rangle$ 
by (simp add: tickFree-map-map-eventptick-id-eq)
from  $*(4)$  obtain  $u1 \ u2 \ t\text{-}P \ t\text{-}Q$  where  $**$  :  $\langle t1 = u1 @ u2 \rangle \langle tF \ u1 \rangle \langle ftF \ u2 \rangle$ 
   $\langle u1 \text{ setinterleaves}_{?tj} ((t\text{-}P, t\text{-}Q), S) \rangle$ 
   $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \rangle$ 
unfolding SyncListslenL.D-Syncptick by blast
from tickFree-setinterleavesptick-iff [THEN iffD1, OF  $** (4, 2)$ ] have  $\langle tF \ t\text{-}P \rangle$ 
..
with  $**(5)$  have  $\langle t\text{-}P \in \mathcal{D} \ ( ?RT \ P) \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ ( ?RT \ P) \wedge t\text{-}Q$ 

```

$\in \mathcal{D} \ Q\rangle$
by (*simp-all add: tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-D*)
with $*(1-4) \ *(2, 3) \ \langle t = t1 \ @ \ t2 \rangle$ **show** $\langle t \in \mathcal{D} \ ?lhs \rangle$
by (*auto simp add: SyncListslenL.D-Syncptick intro: front-tickFree-append*)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle \ \langle t \notin \mathcal{D} \ ?lhs \rangle$
then obtain $t-P \ t-Q \ X-P \ X-Q$
where $*$: $\langle (t-P, X-P) \in \mathcal{F} \ (?RT \ P) \rangle \ \langle (t-Q, X-Q) \in \mathcal{F} \ Q \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark} \ ?tj \ ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Syncptick} \ ?tj \ X-P \ S \ X-Q \rangle$
unfolding *SyncListslenL.Syncptick-projs* **by** *force*
from $*(2, 3) \ \langle t \notin \mathcal{D} \ ?lhs \rangle \ F-T \ \text{front-tickFree-Nil}$
have $\langle t-P \notin \mathcal{D} \ (?RT \ P) \rangle$ **unfolding** *SyncListslenL.D-Syncptick'* **by** *blast*
with $*(1)$ **obtain** $t-P'$ **where** $**$: $\langle t-P = ?map\text{-}evt \ t-P' \rangle \ \langle (t-P', ?fun\text{-}evt \ -' \ X-P) \in \mathcal{F} \ P \rangle$
unfolding *Renaming-projs* **by** *blast*
from $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle [THEN \ F-T]$ **consider** $\langle tF \ t \rangle \mid t' \ rs$ **where** $\langle tF \ t' \rangle \ \langle t = t' \ @ \ [\checkmark(rs)] \rangle$
using *T-nonTickFree-imp-decomp append-T-imp-tickFree* **by** *blast*
thus $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
proof *cases*
assume $\langle tF \ t \rangle$
hence $\langle ?map\text{-}evt \ t = t \rangle$
by (*simp add: tickFree-map-map-eventptick-id-eq*)
have $\langle ?map\text{-}evt \ t-P' = t-P' \rangle$
using $*(3) \ ** (1) \ \langle tF \ t \rangle \ \text{map-event}_{ptick}\text{-tickFree} \ \text{tickFree-map-map-event}_{ptick}\text{-id-eq}$
 $\text{tickFree-setinterleaves}_{ptick}\text{-iff}$ **by** *blast*
have $\langle (t-P, ?fun\text{-}evt \ -' \ X-P) \in \mathcal{F} \ P \rangle$
by (*simp add: ** (1, 2) \ \langle ?map\text{-}evt \ t-P' = t-P' \rangle*)
moreover **have** $\langle (t-Q, X-Q) \in \mathcal{F} \ Q \rangle$ **by** (*fact * (2)*)
moreover **have** $\langle t \text{ setinterleaves}_{\checkmark} \ ?tj \ ((t-P, t-Q), S) \rangle$ **by** (*fact * (3)*)
moreover **have** $\langle e \in ?fun\text{-}evt \ -' \ X \implies e \in \text{super-ref-Syncptick} \ ?tj \ (?fun\text{-}evt \ -' \ X-P) \ S \ X-Q \rangle$ **for** e
using $*(4) [THEN \ \text{set-mp, of } \langle ?fun\text{-}evt \ e \rangle]$
by (*cases e, auto simp add: super-ref-Syncptick-def split: if-split-asm*)
 $(metis \ \text{append-eq-append-conv} \ \text{dual-order.refl} \ \text{length-permute-list} \ \text{pl-append},$
 $\text{metis} \ \text{append-eq-append-conv} \ \text{dual-order.refl} \ \text{length-permute-list} \ \text{pl-append},$
 $\text{metis} \ \text{dual-order.refl} \ \text{length-permute-list} \ \text{pl-append})$
ultimately **have** $\langle (t, ?fun\text{-}evt \ -' \ X) \in \mathcal{F} \ (P \ n \llbracket S \rrbracket_{\checkmark} \text{ListslenL} \ Q) \rangle$
by (*simp add: SyncListslenL.F-Syncptick*) *blast*
with $\langle ?map\text{-}evt \ t = t \rangle$ **show** $\langle (t, X) \in \mathcal{F} \ ?rhs \rangle$
by (*auto simp add: F-Renaming*)
next
fix $t' \ rs$ **assume** $\langle tF \ t' \rangle \ \langle t = t' \ @ \ [\checkmark(rs)] \rangle$
from $*(3)$ **obtain** $t-P'' \ t-Q'' \ r \ s$
where $***$: $\langle \text{length} \ r = n \rangle \ \langle r \ @ \ s = rs \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark} \ ?tj \ ((t-P'', t-Q''), S) \rangle$
 $\langle t-P = t-P'' \ @ \ [\checkmark(r)] \rangle \ \langle t-Q = t-Q'' \ @ \ [\checkmark(s)] \rangle$
by (*auto elim: snoc-tick-setinterleavesptickE*)

$\text{simp add: } \langle t = t' @ [\checkmark(rs)] \rangle \text{ split: if-split-asm}$
have $\langle ?pl\ r @\ s = ?pl\ rs \rangle$ **using** $*** (1, 2)$ **pl-append** **by** *force*
from $\langle t' \text{ setinterleaves}_{\checkmark ?tj} ((t-P'', t-Q'), S) \rangle$
have $\langle ?map\text{-}evt\ t' \text{ setinterleaves}_{\checkmark ?tj} ((?map\text{-}evt\ t-P'', t-Q'), S) \rangle$
by (*metis* (*no-types*, *lifting*) $\langle tF\ t' \rangle \text{ tickFree-map-map-event}_{ptick\text{-}id\text{-}eq}$
 $\text{tickFree-setinterleaves}_{ptick\text{-}iff}$)
have $\langle \text{case } e \text{ of } ev\ a \Rightarrow \text{True} \mid \checkmark(r) \Rightarrow n \leq \text{length } r \rangle$ **if** $\langle e \in \text{set } t-P' \rangle$ **for** e
proof –
have $\langle tF\ t-P'' \rangle$
using $*** (3)$ $\langle tF\ t' \rangle \text{ tickFree-setinterleaves}_{ptick\text{-}iff}$ **by** *blast*
hence $\langle e \in \text{set } t-P'' \Rightarrow \text{is-ev } e \rangle$ **for** e
by (*metis* *in-set-conv-decomp* tickFree-Cons-iff $\text{tickFree-append-iff}$)
moreover from $\langle e \in \text{set } t-P' \rangle$ **have** $\langle ?fun\text{-}evt\ e \in \text{set } t-P \rangle$
by (*simp add: ***(1))
ultimately show $\langle \text{case } e \text{ of } ev\ a \Rightarrow \text{True} \mid \checkmark(r) \Rightarrow n \leq \text{length } r \rangle$
using $*** (1)$ **by** (*cases* e , *auto simp add: ****(4)) (*metis* $\text{event}_{ptick\text{-}disc}$ (2))
qed
with *arg-cong*[*OF* $*** (1)$, **where** $f = ?map\text{-}evt$] *map-evt-map-evt*
have $\langle ?map\text{-}evt\ t-P = t-P' \rangle$ **by** *presburger*
with $**$ **have** $\langle ?map\text{-}evt\ t-P \in \mathcal{T}\ P \rangle$ **by** (*simp add: F-T*)
moreover have $\langle t-Q \in \mathcal{T}\ Q \rangle$ **using** $*(2)$ *F-T* **by** *blast*
moreover from $\langle ?map\text{-}evt\ t' \text{ setinterleaves}_{\checkmark ?tj} ((?map\text{-}evt\ t-P'', t-Q'), S) \rangle$
have $\langle ?map\text{-}evt\ t \text{ setinterleaves}_{\checkmark ?tj} ((?map\text{-}evt\ t-P, t-Q), S) \rangle$
by (*simp add:* $\langle t = t' @ [\checkmark(rs)] \rangle$ $*** (1, 4, 5)$
 $\langle ?pl\ r @\ s = ?pl\ rs \rangle \text{ setinterleaves}_{ptick\text{-}snoc\text{-}tick}$)
ultimately have $\langle ?map\text{-}evt\ t \in \mathcal{T}\ (P\ n \llbracket S \rrbracket_{\checkmark ListslenL}\ Q) \rangle$
unfolding $\text{Sync}_{ListslenL}.T\text{-Sync}_{ptick}$ **by** *blast*
hence $\langle ?map\text{-}evt\ (?map\text{-}evt\ t) \in \mathcal{T}\ ?rhs \rangle$
by (*auto simp add: T-Renaming*)
also have $\langle ?map\text{-}evt\ (?map\text{-}evt\ t) = t \rangle$
by (*simp add:* $\langle t = t' @ [\checkmark(rs)] \rangle$
 $(\text{metis } *** (1, 2) \langle tF\ t' \rangle \text{ dual-order.refl length-permute-list}$
 $\text{list.map-comp pl-append pl-pl tickFree-map-map-event}_{ptick\text{-}id\text{-}eq})$
finally show $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$ **by** (*simp add:* $\langle t = t' @ [\checkmark(rs)] \rangle \text{ tick-T-F}$)
qed
next
fix $t\ X$ **assume** $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle \langle t \notin \mathcal{D}\ ?rhs \rangle \langle t \notin \mathcal{D}\ ?lhs \rangle$
then obtain t' **where** $*$: $\langle t = ?map\text{-}evt\ t' \rangle$
 $\langle (t', ?fun\text{-}evt - 'X) \in \mathcal{F}\ (P\ n \llbracket S \rrbracket_{\checkmark ListslenL}\ Q) \rangle$
unfolding *Renaming-projs* **by** *blast*
from $*(2)$ $\langle t \notin \mathcal{D}\ ?rhs \rangle$ **have** $\langle t' \notin \mathcal{D}\ (P\ n \llbracket S \rrbracket_{\checkmark ListslenL}\ Q) \rangle$
by (*metis* (*no-types*, *lifting*) $*(1)$ *D-imp-front-tickFree div-butlast-when-non-tickFree-iff*
 $\text{front-tickFree-iff-tickFree-butlast map-butlast map-event}_{ptick\text{-}tickFree}$
 $\text{tickFree-map-map-event}_{ptick\text{-}id\text{-}eq} \text{tickFree-mem-D-RenamingTick-iff-mem-D}$)
with $*(2)$ **obtain** $t-P\ t-Q\ X-P\ X-Q$
where $**$: $\langle (t-P, X-P) \in \mathcal{F}\ P \rangle \langle (t-Q, X-Q) \in \mathcal{F}\ Q \rangle$
 $\langle t' \text{ setinterleaves}_{\checkmark ?tj} ((t-P, t-Q), S) \rangle$
 $\langle ?fun\text{-}evt - 'X \subseteq \text{super-ref-Sync}_{ptick}\ ?tj\ X-P\ S\ X-Q \rangle$
unfolding $\text{Sync}_{ListslenL}.Sync_{ptick}\text{-projs}$ **by** *force*

from $\ast(2)$ **consider** $\langle tF\ t' \rangle \mid t''\ rs$ **where** $\langle tF\ t'' \rangle \langle t' = t'' @ [\checkmark(rs)] \rangle$
by (*metis* (*lifting*) *F-T F-imp-front-tickFree T-nonTickFree-imp-decomp*
butlast-snoc front-tickFree-iff-tickFree-butlast)
thus $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$
proof cases
assume $\langle tF\ t' \rangle$
have $\langle ?map\text{-}evt\ t'\ setinterleaves_{\checkmark} ?tj\ ((?map\text{-}evt\ t-P, t-Q), S) \rangle$
by (*metis* (*lifting*) $\ast(3)$ $\langle tF\ t' \rangle\ tickFree\text{-}map\text{-}map\text{-}event_{ptick}\text{-}id\text{-}eq$
tickFree-setinterleaves_{ptick}\text{-}iff)

define $X-P'$ **where** $\langle X-P' \equiv X-P \cap (range\ ev \cup \{\checkmark(r) \mid r.\ length\ r = n\}) \rangle$
define X' **where** $\langle X' \equiv X \cap (range\ ev \cup \{\checkmark(rs) \mid rs.\ n \leq length\ rs\}) \rangle$
have $\langle X-P' \subseteq X-P \rangle$ **unfolding** $X-P'\text{-def}$ **by** *blast*
with $\ast(1)$ *is-processT4* **have** $\langle (t-P, X-P') \in \mathcal{F}\ P \rangle$ **by** *blast*
moreover have $\langle ?fun\text{-}evt - ' (?fun\text{-}evt\ ' X-P') = X-P' \rangle$
by (*auto simp add: X-P'\text{-def}*) (*use length-eq-pl-imp in blast*) +
moreover have $\langle ?map\text{-}evt\ t-P = t-P \rangle$
using $\ast(3)$ $\langle tF\ t' \rangle\ tickFree\text{-}map\text{-}map\text{-}event_{ptick}\text{-}id\text{-}eq\ tickFree\text{-}setinterleaves_{ptick}\text{-}iff$
by *blast*
ultimately have $\langle (t-P, ?fun\text{-}evt\ ' X-P') \in \mathcal{F}\ (?RT\ P) \rangle$
by (*auto simp add: F-Renaming*)
moreover have $\langle e \in X' \implies e \in super\text{-}ref\text{-}Sync_{ptick}\ ?tj\ (?fun\text{-}evt\ ' X-P')\ S$
 $X-Q \rangle$ **if** $\langle e \in X' \rangle$ **for** e
using $\ast(4)$ [*THEN set-mp, of* $\langle ?fun\text{-}evt\ e \rangle$ *fun-evt-fun-evt[of e]*
unfolding X'\text{-def} X-P'\text{-def} super-ref-Sync_{ptick}\text{-def}
by (*auto simp add: image-iff pl-append split: if-split-asm*)
(metis (*mono-tags, lifting*) *Int-iff Un-iff length-permute-list mem-Collect-eq,*
blast, metis length-permute-list)
ultimately have $\langle (t, X') \in \mathcal{F} \ ?lhs \rangle$
by (*simp add: Sync_{ListLenL.F-Sync_{ptick}}*)
(metis (*lifting*) $\ast(1)$ $\ast(2, 3)$ $\langle tF\ t' \rangle\ subsetI\ tickFree\text{-}map\text{-}map\text{-}event_{ptick}\text{-}id\text{-}eq$)
moreover from $\langle t \notin \mathcal{D} \ ?lhs \rangle$ **have** $\langle t @ [\checkmark(rs)] \in \mathcal{T} \ ?lhs \implies n \leq length$
 $(rs) \rangle$ **for** rs
by (*auto simp add: Sync_{ListLenL.Sync_{ptick}\text{-}projs*
elim!: snoc-tick-setinterleaves_{ptick}E split: if-split-asm)
(metis (*no-types, lifting*) *append.assoc butlast-snoc front-tickFree-charn*
non-tickFree-tick tickFree-Nil tickFree-append-iff tickFree-imp-front-tickFree) +
ultimately have $\langle (t, X' \cup (X \cap \{\checkmark(rs) \mid rs.\ \neg n \leq length\ rs\})) \in \mathcal{F} \ ?lhs \rangle$
using *is-processT5-S7'* **by** *fastforce*
also have $\langle X' \cup (X \cap \{\checkmark(rs) \mid rs.\ \neg n \leq length\ rs\}) = X \rangle$
by (*simp add: set-eq-iff X'\text{-def} image-iff*) (*meson event_{ptick}.exhaust*)
finally show $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$.
next
fix $t''\ rs$ **assume** $\langle tF\ t'' \rangle \langle t' = t'' @ [\checkmark(rs)] \rangle$
from $\ast(3)$ **obtain** $t-P'\ t-Q'\ r\ s$
where $\ast\ast : \langle length\ r = n \rangle \langle r @ s = rs \rangle$
 $\langle t''\ setinterleaves_{\checkmark} ?tj\ ((t-P', t-Q'), S) \rangle$
 $\langle t-P = t-P' @ [\checkmark(r)] \rangle \langle t-Q = t-Q' @ [\checkmark(s)] \rangle$
by (*auto elim: snoc-tick-setinterleaves_{ptick}E*)

```

      simp add: ⟨t' = t'' @ [✓(rs)]⟩ split: if-split-asm)
have ⟨?pl r @ s = ?pl rs⟩
  using ***(1, 2) pl-append by force
from ⟨t'' setinterleaves✓?tj ((t-P', t-Q'), S)⟩
have ⟨?map-evt t'' setinterleaves✓?tj ((?map-evt t-P', t-Q'), S)⟩
  by (metis (no-types, lifting) ⟨tF t''⟩ tickFree-map-map-eventptick-id-eq
    tickFree-setinterleavesptick-iff)
from setinterleavesptick-snoc-tick[OF this, of ⟨?pl r⟩ s ⟨?pl rs⟩]
have ⟨?map-evt t' setinterleaves✓?tj ((?map-evt t-P, t-Q), S)⟩
  by (simp add: ***(1, 4, 5) ⟨t' = t'' @ [✓(rs)]⟩ ⟨?pl r @ s = ?pl rs⟩)
moreover from *(1)[THEN F-T] have ⟨(?map-evt t-P, UNIV) ∈ F (?RT
P)⟩
  by (simp add: ***(4), intro tick-T-F) (auto simp add: T-Renaming)
moreover have ⟨(t-Q, UNIV) ∈ F Q⟩
  by (metis *(2) *(5) F-T tick-T-F)
moreover have ⟨e ∈ X ⟹ e ∈ super-ref-Syncptick ?tj UNIV S UNIV⟩ for
e
  using *(4)[THEN set-mp, of ⟨?fun-evt e⟩]
  by (cases e) (auto simp add: super-ref-Syncptick-def)
ultimately show ⟨(t, X) ∈ F ?lhs⟩
  using *(1) by (simp add: SyncListslenL.F-Syncptick) blast
qed
qed
qed

```

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 –

```

let ?RT = RenamingTick and ?MS = ⟨λL. [S]✓ l ∈@ L. P l⟩
let ?RS = ⟨λL. [S]✓ l ∈@ L. P l⟩
let ?τ = ⟨Transposition.transpose⟩
let ?pl-τ = ⟨λi j. permute-list (?τ i j)⟩
have custom-nat-induct [case-names 0 1 2 Suc] :
  ⟨thesis 0 ⟹ thesis 1 ⟹ thesis 2 ⟹
    (∧n. 2 ≤ n ⟹ (∧k. k ≤ n ⟹ thesis k) ⟹ thesis (Suc n)) ⟹ thesis n⟩
for thesis n

```

by (metis One-nat-def Suc-1 less-2-cases linorder-not-le strong-nat-induct)

```

have * : ⟨i ≤ j ⟹ i < length L ⟹ j < length L ⟹
  ?RS (?pl-τ i j L) = ?RT (?RS L) (?pl-τ i j)⟩ for i j

```

proof (induct ⟨length L⟩ arbitrary: i j L rule: custom-nat-induct)

case 0 thus ?case by simp

next

case 1 thus ?case by (cases L) simp-all

next

case 2

```

from 2.hyps 2.premis(1, 3) consider ⟨i = j⟩ | ⟨i = 0⟩ ⟨j = 1⟩ by linarith
thus ?case
proof cases
  show ⟨i = j ⟹ ?case⟩ by simp
next
  let ?g = ⟨λrs. if rs = [] then [] else last rs # butlast rs⟩
  assume ⟨i = 0⟩ ⟨j = 1⟩
  moreover obtain l1 l2 where ⟨L = [l1, l2]⟩
    by (metis 2.hyps One-nat-def Suc-1 diff-Suc-1' length-tl lessI
      list.exhaust-sel nat-less-le order.refl take0 take-all-iff)
  ultimately have ⟨?MS (?pl-τ i j L) = P l2 ⟦S⟧✓Rlist ?RT (P l1) (λr. [r])⟩
    by (simp add: permute-list-transpose-eq-list-update)
  also have ⟨... = ?RT (?RT (P l1) (λr. [r]) ⟦S⟧✓Rlist P l2) ?g⟩
    by (simp add: SyncRlist.Syncptick-comm-locale-sym.Syncptick-commute)
  also have ⟨... = ?RT (?MS L) ?g⟩
    by (simp add: ⟨L = [l1, l2]⟩ MultiSyncptick-snoc[of - ⟨l1⟩, simplified])
  also have ⟨... = ?RT (?MS L) (?pl-τ i j)⟩
    by (rule RenamingTick-is-restrictable-on-strict-ticks-of)
  fix rs assume ⟨rs ∈ ✓s(?MS L)⟩
  from is-ticks-lengthD is-ticks-length-MultiSyncptick this
  have ⟨length rs = length L⟩ .
  thus ⟨?g rs = ?pl-τ i j rs⟩
    by (cases rs; cases ⟨tl rs⟩)
      (simp-all add: ⟨L = [l1, l2]⟩ ⟨i = 0⟩ ⟨j = 1⟩
        permute-list-transpose-eq-list-update)
qed
finally show ?case .
qed
next
  case (Suc n)
  show ?case
  proof (cases ⟨i = j⟩)
    show ⟨i = j ⟹ ?case⟩
      by simp (metis RenamingTick-id eq-id-iff permute-list-id)
  next
    assume ⟨i ≠ j⟩ hence ⟨i < j⟩
      by (simp add: Suc.premis(1) nat-less-le)

    { fix i j L l0 l1 and L' :: ⟨'b list⟩
      assume ⟨i ≠ 0⟩ ⟨i < j⟩ ⟨i < length L⟩ ⟨j < length L⟩ ⟨Suc n = length L⟩
      ⟨L = l0 # l1 # L'⟩
      with ⟨i < length L⟩ ⟨j < length L⟩ ⟨i < j⟩ ⟨i ≠ 0⟩
      have * : ⟨i - 1 < j - 1⟩ ⟨i - 1 < length (l1 # L')⟩
        ⟨j - 1 < length (l1 # L')⟩ by auto
      have ⟨?pl-τ i j L = l0 # ?pl-τ (i - 1) (j - 1) (l1 # L')⟩
      proof (subst (1 2) permute-list-transpose-eq-list-update)
        show ⟨i - 1 < length (l1 # L')⟩ ⟨j - 1 < length (l1 # L')⟩
          ⟨i < length L⟩ ⟨j < length L⟩
          by (fact *(2, 3) ⟨i < length L⟩ ⟨j < length L⟩)+
    }

```

```

next
  from *  $\langle i \neq 0 \rangle$ 
  show  $\langle L[i := L ! j, j := L ! i] =$ 
     $l0 \# (l1 \# L')[i - 1 := (l1 \# L') ! (j - 1), j - 1 := (l1 \# L') ! (i$ 
 $- 1)] \rangle$ 
    by (cases  $i$ ; cases  $j$ ) (simp-all add:  $\langle L = l0 \# l1 \# L' \rangle$  nat.case-eq-if)
  qed
  hence  $\langle ?MS (?pl\text{-}\tau \ i \ j \ L) = P \ l0 \llbracket S \rrbracket_{\checkmark Rlist} ?MS (?pl\text{-}\tau \ (i - 1) \ (j - 1) \ (l1$ 
 $\# L')) \rangle$ 
    by (simp add: MultiSyncptick-Cons)
    (metis Zero-not-Suc length-Cons length-permute-list list.size(3))
  also have  $\langle ?MS (?pl\text{-}\tau \ (i - 1) \ (j - 1) \ (l1 \# L')) =$ 
 $?RT (?MS (l1 \# L')) (?pl\text{-}\tau \ (i - 1) \ (j - 1)) \rangle$ 
    by (subst Suc.hyps)
    (use *  $\langle Suc \ n = length \ L \rangle \langle L = l0 \# l1 \# L' \rangle$  in simp-all)
  also have  $\langle P \ l0 \llbracket S \rrbracket_{\checkmark Rlist} ?RT (?MS (l1 \# L')) (?pl\text{-}\tau \ (i - 1) \ (j - 1)) =$ 
 $?RT (P \ l0 \llbracket S \rrbracket_{\checkmark Rlist} ?MS (l1 \# L')) (?pl\text{-}\tau \ (Suc \ (i - 1)) \ (Suc \ (j$ 
 $- 1))) \rangle$ 
    by (rule SyncRlist-RenamingTick-permute-list-transpose[OF  $*(2, 3)$ ])
    (metis is-ticks-lengthD is-ticks-length-MultiSyncptick order-le-less)
  also have  $\langle (Suc \ (i - 1)) = i \rangle$  using  $\langle i \neq 0 \rangle$  by simp
  also have  $\langle (Suc \ (j - 1)) = j \rangle$  using  $*(1)$  by linarith
  also have  $\langle P \ l0 \llbracket S \rrbracket_{\checkmark Rlist} ?MS (l1 \# L') = ?MS \ L \rangle$ 
    by (simp add:  $\langle L = l0 \# l1 \# L' \rangle$ )
  finally have  $\langle ?MS (?pl\text{-}\tau \ i \ j \ L) = ?RT (?MS \ L) (?pl\text{-}\tau \ i \ j) \rangle$  .
} note  $\mathcal{L} = this$ 

consider  $\langle i \neq 0 \rangle \mid \langle j \neq n \rangle \mid \langle i = 0 \rangle \mid \langle j = n \rangle$  by argo
thus ?case
proof cases
  assume  $\langle i \neq 0 \rangle$ 
  from Suc.hyps(1, 3) obtain  $l0 \ l1 \ L'$  where  $\langle L = l0 \# l1 \# L' \rangle$ 
  by (cases  $L$ ; cases  $\langle tl \ L \rangle$ ) simp-all
  from  $\mathcal{L} \ \langle i \neq 0 \rangle \ \langle i < j \rangle$  Suc.prem(2, 3) Suc.hyps(3) this show ?case .
next
  assume  $\langle j \neq n \rangle$ 
  from Suc.hyps(1, 3) obtain  $l0 \ l1 \ L'$  where  $\langle L = L' @ [l1] @ [l0] \rangle$ 
  by (cases  $L$  rule: rev-cases; cases  $\langle butlast \ L \rangle$  rule: rev-cases) simp-all
  hence  $\langle rev \ L = l0 \# l1 \# rev \ L' \rangle$  by simp
  have  $\langle Suc \ n = length \ (rev \ L) \rangle$  by (simp add: Suc.hyps(3))

  have  $\langle ?MS (?pl\text{-}\tau \ i \ j \ L) = ?MS (rev \ (?pl\text{-}\tau \ (length \ L - Suc \ i) \ (length \ L -$ 
 $Suc \ j)) \ (rev \ L)) \rangle$ 
    by (subst rev-rev-ident[of  $L$ , symmetric], subst permute-list-transpose-rev)
    (simp-all add: Suc.prem(2, 3))
  also have  $\langle \dots = ?RT (?MS (?pl\text{-}\tau \ (length \ L - Suc \ i) \ (length \ L - Suc \ j)$ 
 $(rev \ L))) \ rev \rangle$ 
    by (fact MultiSyncptick-rev)
  also have  $\langle ?pl\text{-}\tau \ (length \ L - Suc \ i) \ (length \ L - Suc \ j) =$ 

```



```

      ?pl- $\tau$  (length L - Suc j) (length L - Suc i)
    by (simp add: transpose-commute)
  also have  $\langle ?MS \text{ (?pl-}\tau \text{ (length L - Suc j) (length L - Suc i) (rev L))} =$ 
     $?RT \text{ (?MS (rev L)) (?pl-}\tau \text{ (length L - Suc j) (length L - Suc i))} \rangle$ 
    by (rule  $\mathcal{L}$ ) (use Suc.hyps(3) Suc.premis(3)  $\langle j \neq n \rangle \langle i < j \rangle$ 
       $\langle rev L = l0 \# l1 \# rev L' \rangle$  in auto)
  also have  $\langle ?MS \text{ (rev L)} = ?RT \text{ (?MS L) rev} \rangle$ 
    by (fact MultiSyncptick-rev)
  also have  $\langle ?RT \text{ (?RT (?RT (?MS L) rev) (?pl-}\tau \text{ (length L - Suc j) (length$ 
     $L - Suc i))) rev} =$ 
     $?RT \text{ (?MS L) (rev } \circ \text{ (?pl-}\tau \text{ (length L - Suc j) (length L - Suc i))$ 
     $\circ rev) \rangle$ 
    by (simp add: RenamingTick-comp)
  also have  $\langle \dots = ?RT \text{ (?MS L) (?pl-}\tau \text{ j i)} \rangle$ 
  proof (rule RenamingTick-is-restrictable-on-strict-ticks-of)
    fix rs assume  $\langle rs \in \check{s} (?MS L) \rangle$ 
    hence  $\langle length \text{ rs} = length \text{ L} \rangle$ 
      using is-ticks-lengthD is-ticks-length-MultiSyncptick by blast
    thus  $\langle (rev \circ (?pl-}\tau \text{ (length L - Suc j) (length L - Suc i)) \circ rev) \text{ rs} =$ 
     $?pl-}\tau \text{ j i rs} \rangle$ 
      by (unfold comp-def, subst rev-permute-list-transpose)
      (use Suc.premis(2, 3) in auto)
  qed
  also have  $\langle \dots = ?RT \text{ (?MS L) (?pl-}\tau \text{ i j)} \rangle$ 
    by (simp add: transpose-commute)
  finally show ?case .
next
let ?g1 =  $\langle \lambda rs. \text{ if } rs = [] \text{ then } [] \text{ else last rs } \# \text{ butlast rs} \rangle$ 
let ?g2 =  $\langle \lambda rs. \text{ drop (Suc (Suc 0)) rs } @ \text{ take (Suc (Suc 0)) rs} \rangle$ 
let ?g3 =  $\langle \lambda rs. \text{ case rs of } r \# s \Rightarrow r \# (\text{if } s = [] \text{ then } [] \text{ else last } s \# \text{ butlast}$ 
 $s) \rangle$ 
let ?tj =  $\langle \lambda r s. [id \text{ r } \# (\text{if } s = [] \text{ then } [] \text{ else last } s \# \text{ butlast } s)] \rangle$ 
assume  $\langle i = 0 \rangle \langle j = n \rangle$ 
from Suc.hyps(1, 3) obtain l0 l1 L'
  where  $\langle L = l0 \# L' @ [l1] \rangle \langle L' \neq [] \rangle$ 
  by (cases L; cases  $\langle tl \text{ L} \rangle$  rule: rev-cases; force)
have  $\langle ?pl-}\tau \text{ i j L} = l1 \# L' @ [l0] \rangle$ 
  by (subst permute-list-transpose-eq-list-update)
  (use Suc.premis(3) Suc.hyps(3)  $\langle L = l0 \# L' @ [l1] \rangle$ 
    in  $\langle auto \text{ simp add: } \langle i = 0 \rangle \langle j = n \rangle \rangle$ )
hence  $\langle ?MS \text{ (?pl-}\tau \text{ i j L)} = P \text{ l1 } \llbracket S \rrbracket_{\check{R}list} (?MS \text{ L'} \llbracket S \rrbracket_{\check{L}list} P \text{ l0}) \rangle$ 
  by (simp add: MultiSyncptick-Cons MultiSyncptick-snoc  $\langle L' \neq [] \rangle$ )
also have  $\langle \dots = ?RT \text{ (?MS L'} \llbracket S \rrbracket_{\check{L}list} P \text{ l0 } \llbracket S \rrbracket_{\check{L}list} P \text{ l1}) ?g1 \rangle$ 
  by (simp only: SyncRlist.Syncptick-comm-locale-sym.Syncptick-commute)
also have  $\langle ?MS \text{ L'} \llbracket S \rrbracket_{\check{L}list} P \text{ l0 } \llbracket S \rrbracket_{\check{L}list} P \text{ l1} =$ 
   $(?MS \text{ L'} \text{ Suc (Suc 0)} \llbracket S \rrbracket_{\check{ListslenR}} (P \text{ l0 } \llbracket S \rrbracket_{\check{Pairlist}} P \text{ l1})) \rangle$ 
  by (simp only: SyncListslenR-SyncPairlist-assoc)
also have  $\langle \dots = ?RT \text{ (P l0 } \llbracket S \rrbracket_{\check{Pairlist}} P \text{ l1 } \text{ Suc (Suc 0)} \llbracket S \rrbracket_{\check{ListslenL}}$ 
 $?MS \text{ L'}) ?g2 \rangle$ 

```

```

    by (simp only: SyncListslenL.Syncptick-commute)
  also have ⟨P l0 [S]✓Pairlist P l1 Suc (Suc 0) [S]✓ListslenL ?MS L' =
    P l0 [S]✓Rlist (P l1 [S]✓Rlist ?MS L')⟩
    by (simp only: SyncRlist-SyncRlist-assoc)
  also have ⟨... = P l0 [S]✓Rlist ?RT (?MS L' [S]✓Llist P l1) ?g1⟩
    by (simp only: SyncRlist.Syncptick-comm-locale-sym.Syncptick-commute)
  also have ⟨... = ?RT (P l0) id [S]✓Rlist ?RT (?MS L' [S]✓Llist P l1)
    ?g1⟩ by simp
  also have ⟨... = Syncptick-locale.Syncptick ?tj (P l0) S (?MS L' [S]✓Llist
    P l1)⟩
  proof (rule SyncRlist.inj-RenamingTick-Syncptick-inj-RenamingTick)
    show ⟨inj id⟩ ⟨inj (λrs. if rs = [] then [] else last rs # butlast rs)⟩
      by (auto intro!: injI split: if-split-asm)
      (metis append-butlast-last-id)
  qed
  also have ⟨... = ?RT (P l0 [S]✓Rlist (?MS L' [S]✓Llist P l1)) ?g3⟩
    by (subst SyncRlist.inj-on-RenamingTick-Syncptick)
    (auto intro!: inj-onI split: if-split-asm, metis append-butlast-last-id)
  also have ⟨P l0 [S]✓Rlist (?MS L' [S]✓Llist P l1) = ?MS L⟩
  by (simp add: ⟨L = l0 # L' @ [l1]⟩ MultiSyncptick-Cons MultiSyncptick-snoc
    ⟨L' ≠ []⟩)
  also have ⟨?RT (?RT (?RT (?MS L) ?g3) ?g2) ?g1 = ?RT (?MS L) (?g1
    o ?g2 o ?g3)⟩
    by (simp only: RenamingTick-comp)
  also have ⟨... = ?RT (?MS L) (?pl-τ i j)⟩
  proof (rule RenamingTick-is-restrictable-on-strict-ticks-of)
    fix rs assume ⟨rs ∈ ✓s(?MS L)⟩
    hence ⟨length rs = length L⟩
      using is-ticks-lengthD is-ticks-length-MultiSyncptick by blast
    obtain n' where ⟨n = Suc (Suc n')⟩
      by (metis One-nat-def Suc.hyps(1) Suc-1 Suc-n-not-le-n ⟨i = 0⟩ ⟨i ≠ j⟩
        ⟨j = n⟩ nat.exhaust-sel)
    with Suc.hyps(1, 3) ⟨length rs = length L⟩
    obtain r0 r1 r2 rs' where ⟨rs = r0 # r1 # r2 # rs'⟩ ⟨n' = length rs'⟩
      by (cases rs; cases ⟨tl rs⟩; cases ⟨tl (tl rs)⟩) simp-all
    show ⟨(?g1 o ?g2 o ?g3) rs = ?pl-τ i j rs⟩
    proof (subst permute-list-transpose-eq-list-update)
      show ⟨i < length rs⟩ ⟨j < length rs⟩
        by (simp-all add: Suc.premis(2, 3) ⟨length rs = length L⟩)
    next
      show ⟨(?g1 o ?g2 o ?g3) rs = rs[i := rs ! j, j := rs ! i]⟩
        by (simp add: ⟨i = 0⟩ ⟨j = n⟩ ⟨rs = r0 # r1 # r2 # rs'⟩ ⟨n' = length
          rs'⟩
          ⟨n = Suc (Suc n')⟩ butlast-append nat.case-eq-if)
      (metis One-nat-def append-butlast-last-id diff-Suc-1' last-conv-nth
        length-0-conv length-butlast list-update-length nat.collapse)
    qed
  qed
  finally show ?case .

```

```

      qed
    qed
  qed

  consider  $\langle i \leq j \rangle \mid \langle j \leq i \rangle$  by linarith
  thus  $\langle ?RS (?pl\text{-}\tau \ i \ j \ L) = ?RT (?RS \ L) (?pl\text{-}\tau \ i \ j) \rangle$  if  $\langle i < \text{length } L \rangle \langle j < \text{length } L \rangle$ 
  proof cases
    from that show  $\langle i \leq j \implies ?RS (?pl\text{-}\tau \ i \ j \ L) = ?RT (?RS \ L) (?pl\text{-}\tau \ i \ j) \rangle$  by
    (fact *)
  next
    from that show  $\langle j \leq i \implies ?RS (?pl\text{-}\tau \ i \ j \ L) = ?RT (?RS \ L) (?pl\text{-}\tau \ i \ j) \rangle$ 
    by (subst (1 2) transpose-commute) (rule *)
  qed
qed

```

Finally, the proof of the general version relies on the fact that a permutation can be written as finite product of transpositions.

```

theorem MultiSyncptick-permute-list :
   $\langle \llbracket S \rrbracket_{\checkmark} l \in @ \text{permute-list } f \ L. \ P \ l =$ 
     $\text{RenamingTick } (\llbracket S \rrbracket_{\checkmark} l \in @ \ L. \ P \ l) \ (\text{permute-list } f) \rangle$ 
  if  $f\text{-permutes} : \langle f \text{ permutes } \{..<\text{length } L\} \rangle$ 
  using finite-lessThan f-permutes
proof (induct f rule: permutes-rev-induct)
  case id show ?case by (simp flip: id-def)
next
  let  $?RT = \text{RenamingTick}$  and  $?pl = \text{permute-list}$  and  $? \tau = \text{Transposition.transpose}$ 
  case (swap i j f)
  have  $\langle ? \tau \ i \ j \text{ permutes } \{..<\text{length } L\} \rangle$ 
    by (meson permutes-swap-id swap.hyps(1, 2))
  hence  $\langle \llbracket S \rrbracket_{\checkmark} l \in @ \ ?pl \ (f \circ ? \tau \ i \ j) \ L. \ P \ l =$ 
     $\llbracket S \rrbracket_{\checkmark} l \in @ \ (?pl \ (? \tau \ i \ j) \ (?pl \ f \ L)). \ P \ l \rangle$ 
    by (simp add: permute-list-compose)

  also have  $\langle \dots = ?RT (\llbracket S \rrbracket_{\checkmark} l \in @ \ (?pl \ f \ L). \ P \ l) \ (?pl \ (? \tau \ i \ j)) \rangle$ 
    by (metis MultiSyncptick-permute-list-transpose atLeast0LessThan
      atLeastLessThan-iff length-permute-list swap.hyps(1,2))
  also have  $\langle \dots = ?RT (?RT (\llbracket S \rrbracket_{\checkmark} l \in @ \ L. \ P \ l) \ (?pl \ f)) \ (?pl \ (? \tau \ i \ j)) \rangle$ 
    unfolding swap.hyps(4) ..
  also have  $\langle \dots = ?RT (\llbracket S \rrbracket_{\checkmark} l \in @ \ L. \ P \ l) \ (?pl \ (? \tau \ i \ j) \circ ?pl \ f) \rangle$ 
    by (simp flip: Renaming-comp)
  also have  $\langle \dots = ?RT (\llbracket S \rrbracket_{\checkmark} l \in @ \ L. \ P \ l) \ (?pl \ (f \circ (? \tau \ i \ j))) \rangle$ 
proof (rule RenamingTick-is-restrictable-on-strict-ticks-of)
  fix rs assume  $\langle rs \in \checkmark s(\llbracket S \rrbracket_{\checkmark} l \in @ \ L. \ P \ l) \rangle$ 
  from is-ticks-lengthD is-ticks-length-MultiSyncptick this
  have  $\langle \text{length } rs = \text{length } L \rangle$  .
  with  $\langle ? \tau \ i \ j \text{ permutes } \{..<\text{length } L\} \rangle$ 
  show  $\langle (?pl \ (? \tau \ i \ j) \circ ?pl \ f) \ rs = ?pl \ (f \circ ? \tau \ i \ j) \ rs \rangle$ 
    by (simp add: permute-list-compose)

```

qed
finally show *?case* .
qed

Chapter 12

Events and Ticks

12.1 Preliminaries

lemma *strict-events-of-memE-optimized-tickFree* :

$\langle (\bigwedge t. t \in \mathcal{T} P \implies t \notin \mathcal{D} P \implies ev\ a \in set\ t \implies tF\ t \implies thesis) \implies thesis \rangle$ **if** $\langle a \in \alpha(P) \rangle$

proof –

from $\langle a \in \alpha(P) \rangle$ **obtain** t **where** $\langle t \in \mathcal{T} P \rangle \langle t \notin \mathcal{D} P \rangle \langle ev\ a \in set\ t \rangle$
 by (*meson strict-events-of-memE*)
 have $\langle (if\ tF\ t\ then\ t\ else\ butlast\ t) \in \mathcal{T} P \rangle$
 by *simp* (*metis* $\langle t \in \mathcal{T} P \rangle$ *append-butlast-last-id is-processT3-TR-append tick-Free-Nil*)
 moreover have $\langle (if\ tF\ t\ then\ t\ else\ butlast\ t) \notin \mathcal{D} P \rangle$
 using *T-imp-front-tickFree* $\langle t \in \mathcal{T} P \rangle \langle t \notin \mathcal{D} P \rangle$ *div-butlast-when-non-tickFree-iff*
by *blast*
 moreover from *T-nonTickFree-imp-decomp* $\langle ev\ a \in set\ t \rangle \langle t \in \mathcal{T} P \rangle$
 have $\langle ev\ a \in set\ (if\ tF\ t\ then\ t\ else\ butlast\ t) \rangle$ **by** *force*
 moreover from *T-imp-front-tickFree* $\langle t \in \mathcal{T} P \rangle$ *front-tickFree-iff-tickFree-butlast*
 have $\langle tF\ (if\ tF\ t\ then\ t\ else\ butlast\ t) \rangle$ **by** (*metis* (*full-types*))
 ultimately show $\langle (\bigwedge t. t \in \mathcal{T} P \implies t \notin \mathcal{D} P \implies ev\ a \in set\ t \implies tF\ t \implies thesis) \implies thesis \rangle$ **by** *blast*
qed

lemma *events-of-memE-optimized-tickFree* :

$\langle (\bigwedge t. t \in \mathcal{T} P \implies ev\ a \in set\ t \implies tF\ t \implies thesis) \implies thesis \rangle$ **if** $\langle a \in \alpha(P) \rangle$

proof –

from $\langle a \in \alpha(P) \rangle$ **obtain** t **where** $\langle t \in \mathcal{T} P \rangle \langle ev\ a \in set\ t \rangle$
 by (*meson events-of-memE*)
 have $\langle (if\ tF\ t\ then\ t\ else\ butlast\ t) \in \mathcal{T} P \rangle$
 by *simp* (*metis* $\langle t \in \mathcal{T} P \rangle$ *append-butlast-last-id is-processT3-TR-append tick-Free-Nil*)
 moreover from *T-nonTickFree-imp-decomp* $\langle ev\ a \in set\ t \rangle \langle t \in \mathcal{T} P \rangle$
 have $\langle ev\ a \in set\ (if\ tF\ t\ then\ t\ else\ butlast\ t) \rangle$ **by** *force*
 moreover from *T-imp-front-tickFree* $\langle t \in \mathcal{T} P \rangle$ *front-tickFree-iff-tickFree-butlast*
 have $\langle tF\ (if\ tF\ t\ then\ t\ else\ butlast\ t) \rangle$ **by** (*metis* (*full-types*))

ultimately show $\langle (\bigwedge t. t \in \mathcal{T} P \implies ev\ a \in set\ t \implies tF\ t \implies thesis) \implies thesis \rangle$
 by *blast*
 qed

12.2 Sequential Composition

12.2.1 Events

lemma *events-of-Seq_{ptick}* : $\langle \alpha(P ; \checkmark Q) = \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$
proof (*intro subset-antisym subsetI*)
 show $\langle a \in \alpha(P ; \checkmark Q) \implies a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$ for a
proof (*elim events-of-memE*)
 fix t assume $\langle t \in \mathcal{T} (P ; \checkmark Q) \rangle$ $\langle ev\ a \in set\ t \rangle$
 from *this*(1) consider $(T-P)\ t'$ where $\langle t = map\ (ev \circ of-ev)\ t' \rangle$ $\langle t' \in \mathcal{T}\ P \rangle$ $\langle tF\ t' \rangle$
 | $(T-Q)\ t'\ r\ u$ where $\langle t = map\ (ev \circ of-ev)\ t' @ u \rangle$ $\langle t' @ [\checkmark(r)] \in \mathcal{T}\ P \rangle$ $\langle tF\ t' \rangle$ $\langle u \in \mathcal{T}\ (Q\ r) \rangle$
 | $(D-P)\ t'\ u$ where $\langle t = map\ (ev \circ of-ev)\ t' @ u \rangle$ $\langle t' \in \mathcal{D}\ P \rangle$ $\langle tF\ t' \rangle$ $\langle ftF\ u \rangle$
 unfolding *Seq_{ptick}-projs* by *blast*
 thus $\langle a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$
proof *cases*
 case *T-P*
 from *T-P*(1, 3) $\langle ev\ a \in set\ t \rangle$ have $\langle ev\ a \in set\ t' \rangle$
 by (*meson tickFree-map-ev-of-ev-eq-imp-ev-mem-iff*)
 with *T-P*(2) have $\langle a \in \alpha(P) \rangle$ by (*rule events-of-memI*)
 thus $\langle a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$ by *simp*
 next
 case *T-Q*
 have $\langle r \in \checkmark s(P) \vee \mathcal{D}\ P \neq \{\} \rangle$
 by (*metis T-Q*(2) *empty-iff strict-ticks-of-memI*)
 thus $\langle a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$
proof (*elim disjE*)
 from *T-Q*
 show $\langle r \in \checkmark s(P) \implies a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$
 by *simp* (*metis Un-iff* $\langle ev\ a \in set\ t \rangle$ *events-of-memI* *set-append* *tickFree-map-ev-of-ev-eq-imp-ev-mem-iff*)
 next
 assume $\langle \mathcal{D}\ P \neq \{\} \rangle$
 hence $\langle \alpha(P) = UNIV \rangle$ by (*simp add: events-of-is-strict-events-of-or-UNIV*)
 thus $\langle a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$ by *simp*
 qed
 next
 case *D-P*
 have $\langle \alpha(P) = UNIV \rangle$
 by (*metis D-P*(2) *empty-iff events-of-is-strict-events-of-or-UNIV*)
 thus $\langle a \in \alpha(P) \cup (\bigcup r \in \checkmark s(P). \alpha(Q\ r)) \rangle$ by *simp*
 qed
 qed
 next

show $\langle a \in \alpha(P) \cup (\bigcup r \in \check{\mathbf{s}}(P). \alpha(Q\ r)) \implies a \in \alpha(P ; \check{\mathbf{J}}\ Q) \rangle$ **for** a
proof (*elim UnE UnionE events-of-memE, safe*)
fix t **assume** $\langle t \in \mathcal{T}\ P \rangle \langle ev\ a \in set\ t \rangle$
then obtain t' **where** $\langle t' \in \mathcal{T}\ P \rangle \langle tF\ t' \rangle \langle ev\ a \in set\ t' \rangle$
by (*cases t rule: rev-cases, simp-all*)
(metis prefixI $\langle ev\ a \in set\ t \rangle$ append-T-imp-tickFree event_{ptick}.disc(1)
is-processT3-TR
not-Cons-self2 tickFree-Cons-iff tickFree-Nil tickFree-append-iff)
thus $\langle a \in \alpha(P ; \check{\mathbf{J}}\ Q) \rangle$ **by** (*auto simp add: Seq_{ptick}-projs rev-image-eqI intro!*
events-of-memI)
next
fix $a\ r$ **assume** $\langle a \in \alpha(Q\ r) \rangle \langle r \in \check{\mathbf{s}}(P) \rangle$
from $\langle r \in \check{\mathbf{s}}(P) \rangle$ **obtain** t **where** $\langle t @ [\check{\mathbf{J}}(r)] \in \mathcal{T}\ P \rangle$ **by** (*meson strict-ticks-of-memD*)
moreover from $\langle a \in \alpha(Q\ r) \rangle$ **obtain** u
where $\langle u \in \mathcal{T}\ (Q\ r) \rangle \langle ev\ a \in set\ u \rangle$ **by** (*meson events-of-memD*)
ultimately have $\langle map\ (ev \circ of-ev)\ t @ u \in \mathcal{T}\ (P ; \check{\mathbf{J}}\ Q) \rangle \langle ev\ a \in set\ (map\ (ev$
 $\circ of-ev)\ t @ u) \rangle$
by (*auto simp add: Seq_{ptick}-projs (metis append-T-imp-tickFree not-Cons-self2)*
thus $\langle a \in \alpha(P ; \check{\mathbf{J}}\ Q) \rangle$ **by** (*simp add: events-of-memI*)
qed
qed

— Big approximation.

lemma *events-of-Seq_{ptick}-subset* : $\langle \alpha(P ; \check{\mathbf{J}}\ Q) \subseteq \alpha(P) \cup (\bigcup r. \alpha(Q\ r)) \rangle$
by (*auto simp add: events-of-Seq_{ptick}*)

— Big approximation.

corollary *events-of-Seq-subset* : $\langle \alpha(P ; Q) \subseteq \alpha(P) \cup \alpha(Q) \rangle$
by (*simp add: events-of-Seq*)

lemma *strict-events-of-Seq_{ptick}-subset* : $\langle \alpha(P ; \check{\mathbf{J}}\ Q) \subseteq \alpha(P) \cup (\bigcup r \in \check{\mathbf{s}}(P). \alpha(Q\ r)) \rangle$

proof (*rule subsetI*)
show $\langle a \in \alpha(P ; \check{\mathbf{J}}\ Q) \implies a \in \alpha(P) \cup (\bigcup r \in \check{\mathbf{s}}(P). \alpha(Q\ r)) \rangle$ **for** a
proof (*elim strict-events-of-memE*)
fix t **assume** $\langle t \in \mathcal{T}\ (P ; \check{\mathbf{J}}\ Q) \rangle \langle t \notin \mathcal{D}\ (P ; \check{\mathbf{J}}\ Q) \rangle \langle ev\ a \in set\ t \rangle$
from *this(1, 2)* **consider** (T - P) t' **where** $\langle t = map\ (ev \circ of-ev)\ t' \rangle \langle t' \in \mathcal{T}\ P \rangle \langle t' \notin \mathcal{D}\ P \rangle \langle tF\ t' \rangle$
| (T - Q) $t'\ r\ u$ **where** $\langle t = map\ (ev \circ of-ev)\ t' @ u \rangle \langle t' @ [\check{\mathbf{J}}(r)] \in \mathcal{T}\ P \rangle \langle t' \notin \mathcal{D}\ P \rangle \langle tF\ t' \rangle \langle u \in \mathcal{T}\ (Q\ r) \rangle \langle u \notin \mathcal{D}\ (Q\ r) \rangle$
by (*auto simp add: Seq_{ptick}-projs (metis T-imp-front-tickFree)*)
thus $\langle a \in \alpha(P) \cup (\bigcup r \in \check{\mathbf{s}}(P). \alpha(Q\ r)) \rangle$
proof cases
case T - P
have $\langle ev\ a \in set\ t' \rangle$
by (*metis T-P(1, 4) $\langle ev\ a \in set\ t \rangle$ tickFree-map-ev-of-ev-eq-imp-ev-mem-iff*)
have $\langle a \in \alpha(P) \rangle$

```

      by (meson T-P(2, 3) ⟨ev a ∈ set t'⟩ strict-events-of-memI)
    thus ⟨a ∈ α(P) ∪ (⋃ r ∈ √s(P).α(Q r))⟩ by simp
  next
  case T-Q
  have ⟨r ∈ √s(P)⟩ by (meson T-Q(2, 3) is-processT9 strict-ticks-of-memI)
  thus ⟨a ∈ α(P) ∪ (⋃ r ∈ √s(P).α(Q r))⟩
    by simp (metis T-Q UnE ⟨ev a ∈ set t'⟩ is-processT3-TR-append set-append
      strict-events-of-memI tickFree-map-ev-of-ev-eq-imp-ev-mem-iff)
  qed
qed
qed

```

12.2.2 Ticks

```

lemma ticks-of-Seqptick :
  ⟨√s(P ;√ Q) = (if D P = {} then (⋃ r ∈ √s(P). √s(Q r)) else UNIV)⟩
proof (split if-split, intro conjI impI)
  show ⟨D P ≠ {} ⟹ √s(P ;√ Q) = UNIV⟩
    by (simp add: Seqptick-projs ticks-of-is-strict-ticks-of-or-UNIV)
      (metis front-tickFree-Nil nonempty-divE)
  next
  show ⟨D P = {} ⟹ √s(P ;√ Q) = (⋃ r ∈ √s(P). √s(Q r))⟩ if ⟨D P = {}⟩
  proof (intro subset-antisym subsetI)
    from ⟨D P = {}⟩ ticks-of-memI[of - - ⟨Q -⟩]
    show ⟨s ∈ √s(P ;√ Q) ⟹ s ∈ (⋃ r ∈ √s(P). √s(Q r))⟩ for s
      by (auto simp add: Seqptick-projs strict-ticks-of-def append-eq-map-conv
        append-eq-append-conv2 Cons-eq-append-conv elim!: ticks-of-memE)
        (blast, metis append-Nil)
  next
  show ⟨s ∈ (⋃ r ∈ √s(P). √s(Q r)) ⟹ s ∈ √s(P ;√ Q)⟩ for s
    by (auto simp add: Seqptick-projs ticks-of-def elim!: strict-ticks-of-memE)
      (meson append.assoc append-T-imp-tickFree not-Cons-self2)
  qed
qed

```

```

lemma ⟨√s(P ;√ Q) ⊆ ⋃ {√s(Q r) | r. r ∈ √s(P)}⟩
  — Already proven earlier in the construction.
  by (fact strict-ticks-of-Seqptick-subset)

```

12.3 Synchronization Product

12.3.1 Events

```

lemma (in Syncptick-locale) events-of-Syncptick-subset : ⟨α(P [S]√ Q) ⊆ α(P) ∪
α(Q)⟩
  by (subst events-of-def, simp add: T-Syncptick subset-iff)
    (metis UNIV-I empty-iff events-of-is-strict-events-of-or-UNIV
      events-of-memI setinterleavesptick-preserves-ev-notin-set)

```


lemma (in *Sync_{ptick}-locale*) *events-of-Inter_{ptick}*: $\langle \alpha(P \parallel_{\checkmark} Q) = \alpha(P) \cup \alpha(Q) \rangle$
proof (rule *subset-antisym*[*OF events-of-Sync_{ptick}-subset*])
show $\langle \alpha(P) \cup \alpha(Q) \subseteq \alpha(P \parallel_{\checkmark} Q) \rangle$
proof (rule *subsetI*, *elim UnE*)
fix *a* **assume** $\langle a \in \alpha(P) \rangle$
then obtain *t-P* **where** $\langle tF\ t-P \rangle \langle ev\ a \in set\ t-P \rangle \langle t-P \in \mathcal{T}\ P \rangle$
by (*meson events-of-memE-optimized-tickFree*)
have $\langle map\ ev\ (map\ of-ev\ t-P)\ setinterleaves_{\checkmark tick-join}\ ((t-P, []), \{\}) \rangle$
by (*simp add: tF t-P setinterleaves_{ptick}-NilR-iff*)
hence $\langle map\ ev\ (map\ of-ev\ t-P) \in \mathcal{T}\ (P \parallel_{\checkmark} Q) \rangle$
by (*simp add: T-Sync_{ptick}*) (*metis t-P ∈ T P is-processT1-TR*)
moreover from $\langle ev\ a \in set\ t-P \rangle$ **have** $\langle ev\ a \in set\ (map\ ev\ (map\ of-ev\ t-P)) \rangle$
by force
ultimately show $\langle a \in \alpha(P \parallel_{\checkmark} Q) \rangle$ **by** (*metis events-of-memI*)
next
fix *a* **assume** $\langle a \in \alpha(Q) \rangle$
then obtain *t-Q* **where** $\langle tF\ t-Q \rangle \langle ev\ a \in set\ t-Q \rangle \langle t-Q \in \mathcal{T}\ Q \rangle$
by (*meson events-of-memE-optimized-tickFree*)
have $\langle map\ ev\ (map\ of-ev\ t-Q)\ setinterleaves_{\checkmark tick-join}\ (([], t-Q), \{\}) \rangle$
by (*simp add: tF t-Q setinterleaves_{ptick}-NilL-iff*)
hence $\langle map\ ev\ (map\ of-ev\ t-Q) \in \mathcal{T}\ (P \parallel_{\checkmark} Q) \rangle$
by (*simp add: T-Sync_{ptick}*) (*metis t-Q ∈ T Q is-processT1-TR*)
moreover from $\langle ev\ a \in set\ t-Q \rangle$ **have** $\langle ev\ a \in set\ (map\ ev\ (map\ of-ev\ t-Q)) \rangle$
by force
ultimately show $\langle a \in \alpha(P \parallel_{\checkmark} Q) \rangle$ **by** (*metis events-of-memI*)
qed
qed

lemma (in *Sync_{ptick}-locale*) *strict-events-of-Sync_{ptick}-subset* :
 $\langle \alpha(P \llbracket S \rrbracket_{\checkmark} Q) \subseteq \alpha(P) \cup \alpha(Q) \rangle$
proof (rule *subsetI*)
fix *a* **assume** $\langle a \in \alpha(P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
then obtain *t* **where** $\langle t \in \mathcal{T}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle \langle ev\ a \in set\ t \rangle \langle tF\ t \rangle \langle t \notin \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
by (*blast elim: strict-events-of-memE-optimized-tickFree*)
from $\langle t \in \mathcal{T}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle \langle t \notin \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
obtain *t-P t-Q* **where** $\langle t-P \in \mathcal{T}\ P \rangle \langle t-Q \in \mathcal{T}\ Q \rangle$
and *setinter* : $\langle t\ setinterleaves_{\checkmark tick-join}\ ((t-P, t-Q), S) \rangle$
unfolding *Sync_{ptick}-projs* **by** *blast*
from *this*(3) *setinterleaves_{ptick}-preserves-ev-notin-set* $\langle ev\ a \in set\ t \rangle$
have $\langle ev\ a \in set\ t-P \vee ev\ a \in set\ t-Q \rangle$ **by** *metis*
moreover have $\langle t-P \notin \mathcal{D}\ P \wedge t-Q \notin \mathcal{D}\ Q \rangle$
proof (rule *ccontr*)
assume $\langle \neg (t-P \notin \mathcal{D}\ P \wedge t-Q \notin \mathcal{D}\ Q) \rangle$
with $\langle t-P \in \mathcal{T}\ P \rangle \langle t-Q \in \mathcal{T}\ Q \rangle \langle tF\ t \rangle$ *front-tickFree-Nil setinter*
have $\langle t \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **unfolding** *D-Sync_{ptick}* **by** *blast*
with $\langle t \notin \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **show** *False* ..

qed
 ultimately show $\langle a \in \alpha(P) \cup \alpha(Q) \rangle$
 by (meson UnCI $\langle t \cdot P \in \mathcal{T} P \rangle \langle t \cdot Q \in \mathcal{T} Q \rangle$ strict-events-of-memI)
 qed

12.3.2 Ticks

lemma (in *Sync_{ptick}-locale*)
 $\langle \check{s}(P \llbracket S \rrbracket_{\check{s}} Q) \subseteq \{r \cdot s \mid r \cdot s \text{ } r \text{ } s. r \otimes \check{s} s = \text{Some } r \cdot s \wedge r \in \check{s}(P) \wedge s \in \check{s}(Q)\} \rangle$
 — Already proven earlier in the construction.
 by (fact strict-ticks-of-Sync_{ptick}-subset)

lemma (in *Sync_{ptick}-locale*) ticks-of-no-div-Sync_{ptick}-subset :
 $\langle \mathcal{D} (P \llbracket S \rrbracket_{\check{s}} Q) = \{\} \implies \check{s}(P \llbracket S \rrbracket_{\check{s}} Q) \subseteq \{r \cdot s \mid r \cdot s \text{ } r \text{ } s. \text{tick-join } r \text{ } s = \text{Some } r \cdot s \wedge r \in \check{s}(P) \wedge s \in \check{s}(Q)\} \rangle$
 using strict-ticks-of-Sync_{ptick}-subset
 by (simp add: ticks-of-is-strict-ticks-of-or-UNIV subset-iff) blast

12.4 Architectural Operators

12.4.1 Events

lemma events-of-MultiSeq-subset :

$\langle \alpha(\text{SEQ } l \in @ L. P \text{ } l) \subseteq (\bigcup l \in \text{set } L. \bigcup r. \alpha(P \text{ } l)) \rangle$
 by (induct L rule: rev-induct)
 (auto intro!: subset-trans[OF events-of-Seq-subset])

lemma events-of-MultiSeq_{ptick}-subset :
 $\langle \alpha((\text{SEQ}_{\check{s}} l \in @ L. P \text{ } l) \text{ } r) \subseteq (\bigcup l \in \text{set } L. \bigcup r. \alpha(P \text{ } l \text{ } r)) \rangle$
 by (induct L arbitrary: r)
 (auto intro!: subset-trans[OF events-of-Seq_{ptick}-subset])

lemma strict-events-of-MultiSeq-subset :

$\langle \alpha(\text{SEQ } l \in @ L. P \text{ } l) \subseteq (\bigcup l \in \text{set } L. \bigcup r. \alpha(P \text{ } l)) \rangle$
 by (induct L rule: rev-induct)
 (auto intro!: subset-trans[OF strict-events-of-Seq-subseteq]
 split: if-split-asm)

lemma strict-events-of-MultiSeq_{ptick}-subset :
 $\langle \alpha((\text{SEQ}_{\check{s}} l \in @ L. P \text{ } l) \text{ } r) \subseteq (\bigcup l \in \text{set } L. \bigcup r. \alpha(P \text{ } l \text{ } r)) \rangle$
 by (induct L arbitrary: r, simp)
 (auto intro!: subset-trans[OF strict-events-of-Seq_{ptick}-subset])

lemma events-of-MultiSync_{ptick}-subset :

$\langle \alpha(\llbracket S \rrbracket_{\checkmark} l \in @ L. P l) \subseteq (\bigcup l \in \text{set } L. \alpha(P l)) \rangle$
by (*induct L rule: induct-list012, simp-all*)
 (*metis eq-id-iff events-of-Renaming order.order-iff-strict*
image-id events-of-is-strict-events-of-or-UNIV,
use Sync_{Rlist}.events-of-Sync_{ptick}-subset in fastforce)

lemma *events-of-MultiInter_{ptick}* :
 $\langle \alpha(\llbracket S \rrbracket_{\checkmark} l \in @ L. P l) = (\bigcup l \in \text{set } L. \alpha(P l)) \rangle$
by (*induct L rule: induct-list012,*
simp-all add: Sync_{Rlist}.events-of-Inter_{ptick})
 (*metis events-of-Renaming events-of-is-strict-events-of-or-UNIV id-apply image-id*)

lemma *strict-events-of-MultiSync_{ptick}-subset* :
 $\langle \alpha(\llbracket S \rrbracket_{\checkmark} l \in @ L. P l) \subseteq (\bigcup l \in \text{set } L. \alpha(P l)) \rangle$
by (*induct L rule: induct-list012, simp-all add: strict-events-of-inj-on-Renaming*)
 (*use Sync_{Rlist}.strict-events-of-Sync_{ptick}-subset in fastforce*)

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* :
 $\langle \checkmark s((SEQ_{\checkmark} l \in @ L. P l) r) \subseteq (\text{if } L = [] \text{ then } \{r\} \text{ else } (\bigcup r. \checkmark s(P (\text{last } L) r))) \rangle$
proof (*induct L arbitrary: r*)
case Nil show ?case by simp
next
case (Cons l L)
have $\langle (SEQ_{\checkmark} m \in @ (l \# L). P m) r = P l r ; \checkmark SEQ_{\checkmark} l \in @ L. P l \rangle$ **by simp**
also have $\langle \checkmark s(\dots) \subseteq \bigcup \{ \checkmark s((SEQ_{\checkmark} l \in @ L. P l) r') \mid r'. r' \in \checkmark s(P l r) \} \rangle$
by (*fact strict-ticks-of-Seq_{ptick}-subset*)
also have $\langle \dots \subseteq \bigcup \{ \text{if } L = [] \text{ then } \{r'\} \text{ else } \bigcup r. \checkmark s(P (\text{last } L) r) \mid r'. r' \in \checkmark s(P l r) \} \rangle$
using Cons.hyps by (blast intro: Union-subsetI)
also have $\langle \dots \subseteq (\text{if } l \# L = [] \text{ then } \{r\} \text{ else } \bigcup r. \checkmark s(P (\text{last } (l \# L)) r)) \rangle$ **by**
auto
finally show ?case .
qed

lemma *strict-ticks-of-MultiSeq-subset* :
 $\langle \checkmark s(SEQ l \in @ L. P l) \subseteq (\text{if } L = [] \text{ then } \{\text{undefined}\} \text{ else } (\bigcup r. \checkmark s(P (\text{last } L)))) \rangle$
using *strict-ticks-of-MultiSeq_{ptick}-subset* [*of L $\lambda l r. P l$*]
unfolding *MultiSeq_{ptick}-const* **by auto**

lemma *strict-ticks-of-MultiSync_{ptick}-subset* :
 $\langle \checkmark s(\llbracket S \rrbracket_{\checkmark} l \in @ L. P l) \subseteq \{ l. \text{length } l = \text{length } L \wedge (\forall i < \text{length } L. l ! i \in \checkmark s(P (L ! i))) \} \rangle$

```

proof (induct L rule: induct-list012)
  case 1 show ?case by simp
next
  case (2 l0) show ?case
    by (auto intro!: subset-trans[OF strict-ticks-of-RenamingTick-subset])
next
  case (3 l0 l1 L)
    have  $\langle \llbracket S \rrbracket_{\checkmark} l \in @ (l0 \# l1 \# L). P \ l = P \ l0 \ \llbracket S \rrbracket_{\checkmark Rlist} \llbracket S \rrbracket_{\checkmark} l \in @ (l1 \# L). P \ l \rangle$  by simp
    also have  $\langle \checkmark s(\dots) \subseteq \{r \# s \mid r \ s. \ r \in \checkmark s(P \ l0) \wedge s \in \checkmark s(\llbracket S \rrbracket_{\checkmark} l \in @ (l1 \# L). P \ l)\} \rangle$ 
      by (rule subset-trans[OF SyncRlist.strict-ticks-of-Syncptick-subset]) blast
    also have  $\langle \dots \subseteq \{r \# s \mid r \ s. \ r \in \checkmark s(P \ l0) \wedge s \in \{l. \ length \ l = length \ (l1 \# L) \wedge (\forall i < length \ (l1 \# L). \ l \ ! \ i \in \checkmark s(P \ ((l1 \# L) \ ! \ i)))\} \rangle$ 
      using 3.hyps(2) by blast
    also have  $\langle \dots = \{l. \ length \ l = length \ (l0 \# l1 \# L) \wedge (\forall i < length \ (l0 \# l1 \# L). \ l \ ! \ i \in \checkmark s(P \ ((l0 \# l1 \# L) \ ! \ i)))\} \rangle$ 
      (is  $\langle ?S1 = ?S2 \rangle$ )
    proof (unfold set-eq-iff, intro allI)
      show  $\langle l \in ?S1 \longleftrightarrow l \in ?S2 \rangle$  for l
      by (cases l, auto, metis less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc)
    qed
  finally show ?case .
qed

```

Chapter 13

Continuity Rules

13.1 Sequential Composition

13.1.1 Monotonicity

lemma *tickFree-mem-min-elems-D* : $\langle t \in \text{min-elems } (\mathcal{D} P) \implies tF t \rangle$
by (*metis D-imp-front-tickFree Prefix-Order.prefixI append-self-conv elem-min-elems*
is-processT9 min-elems-no nonTickFree-n-frontTickFree not-Cons-self2)

lemma *mono-Seq_{ptick}* : $\langle P ; \checkmark R \sqsubseteq Q ; \checkmark S \rangle$ **if** $\langle P \sqsubseteq Q \rangle$ **and** $\langle R \sqsubseteq S \rangle$
for $P Q :: \langle 'a, 'r \rangle \text{ process}_{ptick} \rangle$ **and** $R S :: \langle 'r \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$

proof –

let $?S = \langle \lambda P R. \text{map } (ev \circ of-ev) \text{ `min-elems } (\mathcal{D} P) \cup$
 $\{ \text{map } (ev \circ of-ev) t @ u \mid t r u. t @ [\checkmark(r)] \in \mathcal{T} P \wedge t \notin \mathcal{D} P \wedge$
 $tF t \wedge u \in \text{min-elems } (\mathcal{D} (R r)) \} \rangle$
{ fix P **and** $R :: \langle 'r \Rightarrow ('a, 's) \text{ process}_{ptick} \rangle$ **and** t
assume $\langle t \in \text{min-elems } (\mathcal{D} (P ; \checkmark R)) \rangle$
hence $*$: $\langle t \in \mathcal{D} (P ; \checkmark R) \rangle$ **and** $**$: $\langle \bigwedge t'. t' \in \mathcal{D} (P ; \checkmark R) \implies \neg t' < t \rangle$
by (*simp-all add: min-elems-def*)
from $*$ **consider** $(D-P) t' u$ **where** $\langle t = \text{map } (ev \circ of-ev) t' @ u \rangle \langle t' \in \mathcal{D} P \rangle$
 $\langle tF t' \rangle \langle ftF u \rangle$
 $\mid (D-R) t' r u$ **where** $\langle t = \text{map } (ev \circ of-ev) t' @ u \rangle \langle t' @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle t' \notin \mathcal{D} P \rangle \langle tF t' \rangle \langle u \in \mathcal{D} (R r) \rangle$
by (*simp add: Seq_{ptick}-projs*) (*metis D-imp-front-tickFree*)
hence $\langle t \in ?S P R \rangle$
proof cases
case $D-P$
from $D-P(1-3) **[\text{of } \langle \text{map } (ev \circ of-ev) t' \rangle]$ **have** $\langle u = [] \rangle$
by (*simp add: Seq_{ptick}-projs*)
 $(\text{metis strict-prefixI' append.right-neutral front-tickFree-Nil neq-Nil-conv})$
have $\langle t' \in \text{min-elems } (\mathcal{D} P) \rangle$
proof (*rule ccontr*)
assume $\langle t' \notin \text{min-elems } (\mathcal{D} P) \rangle$
with $D-P(2)$ **obtain** t'' **where** $\langle t'' \in \mathcal{D} P \rangle \langle t'' < t' \rangle$ **unfolding** *min-elems-def*

```

by fast
  with D-P(1, 3) **[of ⟨map (ev ∘ of-ev) t'⟩] show False
  by (auto simp add: Seqptick-projs ⟨u = []⟩)
  (metis (no-types, lifting) strict-prefixE' strict-prefix-simps(2) front-tickFree-Nil
    less-append list.simps(9) map-append self-append-conv tickFree-append-iff)
qed
thus ⟨t ∈ ?S P R⟩ by (simp add: D-P(1) ⟨u = []⟩)
next
case D-R
have ⟨u ∈ min-elems (D (R r))⟩
proof (rule ccontr)
  assume ⟨u ∉ min-elems (D (R r))⟩
  with D-R(5) obtain u' where ⟨u' ∈ D (R r)⟩ ⟨u' < u⟩ unfolding
min-elems-def by fast
  with D-R(1, 2, 4) **[of ⟨map (ev ∘ of-ev) t' @ u'⟩] show False
  by (simp add: Seqptick-projs) (use less-append in blast)
qed
with D-R(1-4) show ⟨t ∈ ?S P R⟩ by auto
qed
} note $ = this

show ⟨P ;✓ R ⊆ Q ;✓ S⟩
proof (rule below-trans)
  show ⟨P ;✓ R ⊆ Q ;✓ R⟩
  proof (unfold le-approx-def, safe)
    from le-approx1[OF ⟨P ⊆ Q⟩] le-approx-lemma-T[OF ⟨P ⊆ Q⟩]
    show ⟨t ∈ D (Q ;✓ R) ⟹ t ∈ D (P ;✓ R)⟩ for t
    unfolding Seqptick-projs by blast
  next
  from le-approx2[OF ⟨P ⊆ Q⟩] le-approx2T[OF ⟨P ⊆ Q⟩]
  show ⟨t ∉ D (P ;✓ R) ⟹ X ∈ ℛa (P ;✓ R) t ⟹ X ∈ ℛa (Q ;✓ R) t⟩ for
t X
    by (simp add: Seqptick-projs Refusals-after-def)
    (metis F-imp-front-tickFree append.right-neutral front-tickFree-Nil is-processT9)
  next
  from le-approx2[OF ⟨P ⊆ Q⟩] le-approx2T[OF ⟨P ⊆ Q⟩] le-approx1[OF ⟨P
⊆ Q⟩]
  show ⟨t ∉ D (P ;✓ R) ⟹ X ∈ ℛa (Q ;✓ R) t ⟹ X ∈ ℛa (P ;✓ R) t⟩ for
t X
    by (simp add: subset-iff Seqptick-projs Refusals-after-def)
    (metis D-T is-processT8)
  next
  show ⟨t ∈ min-elems (D (P ;✓ R)) ⟹ t ∈ ℱ (Q ;✓ R)⟩ for t
  proof (rule set-mp[OF - $])
    from le-approx2T[OF ⟨P ⊆ Q⟩] le-approx3[OF ⟨P ⊆ Q⟩] show ⟨?S P R ⊆
ℱ (Q ;✓ R)⟩
    by (simp add: subset-iff Seqptick-projs)
    (meson D-T elem-min-elems image-iff is-processT9 tickFree-mem-min-elems-D)

```

```

    qed
  qed
next
  show  $\langle Q ; \checkmark R \sqsubseteq Q ; \checkmark S \rangle$ 
  proof (unfold le-approx-def, safe)
    from le-approx1[OF fun-belowD[OF  $\langle R \sqsubseteq S \rangle$ ]]
    show  $\langle t \in \mathcal{D} (Q ; \checkmark S) \implies t \in \mathcal{D} (Q ; \checkmark R) \rangle$  for  $t$ 
      unfolding Seqptick-projs by blast
  next
    from proc-ord2a[OF fun-belowD[OF  $\langle R \sqsubseteq S \rangle$ ]]
    show  $\langle t \notin \mathcal{D} (Q ; \checkmark R) \implies X \in \mathcal{R}_a (Q ; \checkmark R) \ t \implies X \in \mathcal{R}_a (Q ; \checkmark S) \ t \rangle$ 
       $\langle t \notin \mathcal{D} (Q ; \checkmark R) \implies X \in \mathcal{R}_a (Q ; \checkmark S) \ t \implies X \in \mathcal{R}_a (Q ; \checkmark R) \ t \rangle$  for  $t \ X$ 
      by (simp add: Seqptick-projs Refusals-after-def, metis)+
  next
    show  $\langle t \in \text{min-elems } (\mathcal{D} (Q ; \checkmark R)) \implies t \in \mathcal{T} (Q ; \checkmark S) \rangle$  for  $t$ 
    proof (rule set-mp[OF - $])
      from le-approx3[OF fun-belowD[OF  $\langle R \sqsubseteq S \rangle$ ]] show  $\langle ?S \ Q \ R \subseteq \mathcal{T} (Q ; \checkmark$ 
S)  $\rangle$ 
      by (simp add: subset-iff Seqptick-projs)
      (meson D-T elem-min-elems image-iff tickFree-mem-min-elems-D)
    qed
  qed
qed
qed
qed

```

13.1.2 Preliminaries

context begin

private lemma *chain-Seq_{ptick}-left*: $\langle \text{chain } Y \implies \text{chain } (\lambda i. Y \ i ; \checkmark S) \rangle$
 by (simp add: mono-Seq_{ptick} po-class.chain-def)

private lemma *chain-Seq_{ptick}-right*: $\langle \text{chain } Y \implies \text{chain } (\lambda i. S ; \checkmark Y \ i) \rangle$
 by (simp add: mono-Seq_{ptick} po-class.chain-def)

private lemma *cont-left-prem-Seq_{ptick}* :
 $\langle (\bigsqcup i. Y \ i) ; \checkmark S = (\bigsqcup i. Y \ i ; \checkmark S) \rangle$ (is $\langle ?lhs = ?rhs \rangle$) if $\langle \text{chain } Y \rangle$
 — We have to add this hypothesis in the generalization.
proof (rule Process-eq-optimizedI)
 show $\langle t \in \mathcal{D} \ ?lhs \implies t \in \mathcal{D} \ ?rhs \rangle$ for t
 by (simp add: Seq_{ptick}-projs limproc-is-the lub ch2ch-fun $\langle \text{chain } Y \rangle$ lub-fun
chain-Seq_{ptick}-left LUB-projs) blast
 next
 have $\langle t \in \mathcal{D} \ ?lhs \rangle$ if $\langle t \in \mathcal{D} \ ?rhs \rangle$ and $\langle tF \ t \rangle$ for t
proof (cases $\langle \text{map } (ev \circ of-ev) \ t \in \mathcal{D} (\bigsqcup i. Y \ i) \rangle$)
 show $\langle \text{map } (ev \circ of-ev) \ t \in \mathcal{D} (\bigsqcup i. Y \ i) \implies t \in \mathcal{D} \ ?lhs \rangle$
 by (simp add: Seq_{ptick}-projs)
 (metis append.right-neutral front-tickFree-Nil $\langle tF \ t \rangle$ tickFree-map-ev-comp)

```

tickFree-map-ev-of-ev-eq-iff)
next
  define T1 and T2
    where  $\langle T1\ i \equiv \{t1. \exists t2. t = \text{map}\ (ev \circ of-ev)\ t1\ @\ t2 \wedge t1 \in \mathcal{D}\ (Y\ i) \wedge tF\ t1 \wedge ftF\ t2\} \rangle$ 
    and  $\langle T2\ i \equiv \{t1. \exists t2\ r. t = \text{map}\ (ev \circ of-ev)\ t1\ @\ t2 \wedge t1\ @\ [\checkmark(r)] \in \mathcal{T}\ (Y\ i) \wedge tF\ t1 \wedge t2 \in \mathcal{D}\ (S\ r)\} \rangle$  for  $i$ 
    assume  $\langle \text{map}\ (ev \circ of-ev)\ t \notin \mathcal{D}\ (\bigsqcup i. Y\ i) \rangle$ 
    with  $\langle t \in \mathcal{D}\ ?rhs \rangle$  have  $\langle T1\ i \cup T2\ i \neq \{\} \rangle$  for  $i$ 
    by (simp add: T1-def T2-def limproc-is-thelub chain-Seqptick-left  $\langle \text{chain}\ Y \rangle$ 
        LUB-projs Seqptick-projs) fast
    moreover have  $\langle \text{finite}\ (T1\ 0 \cup T2\ 0) \rangle$ 
    unfolding T1-def T2-def
    by (rule finite-subset[of -  $\langle \{u. u \leq \text{map}\ (ev \circ of-ev)\ t\} \rangle$ ])
        (use tickFree-map-ev-of-ev-eq-iff in  $\langle \text{force}\ \text{simp}\ \text{add:}\ \text{prefixes-fin} \rangle$ ) +
    moreover have  $\langle T1\ (Suc\ i) \cup T2\ (Suc\ i) \subseteq T1\ i \cup T2\ i \rangle$  for  $i$ 
    unfolding T1-def T2-def by (intro allI subsetI; simp)
        (metis (no-types, lifting)  $\langle \text{chain}\ Y \rangle$  po-class.chainE le-approx-lemma-T
le-approx1
        subsetD[of  $\langle \mathcal{D}\ (Y\ (Suc\ i)) \rangle$   $\langle \mathcal{D}\ (Y\ i) \rangle$ ] subsetD[of  $\langle \mathcal{T}\ (Y\ (Suc\ i)) \rangle$   $\langle \mathcal{T}\ (Y\ i) \rangle$ ]
         $\langle -\ @\ [\checkmark(-)] \rangle$ )
    ultimately have  $\langle (\bigcap i. T1\ i \cup T2\ i) \neq \{\} \rangle$  by (rule Inter-nonempty-finite-chained-sets)
    then obtain t1 where  $*$  :  $\langle \forall i. t1 \in T1\ i \cup T2\ i \rangle$  by auto
    then obtain t2 where  $**$  :  $\langle t = \text{map}\ (ev \circ of-ev)\ t1\ @\ t2 \rangle$   $\langle tF\ t1 \rangle$   $\langle ftF\ t2 \rangle$ 
    by (auto simp add: T1-def T2-def dest: D-imp-front-tickFree)
    show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
    proof (cases  $\langle \forall i. t1 \in \mathcal{D}\ (Y\ i) \rangle$ )
    from  $**$  show  $\langle \forall i. t1 \in \mathcal{D}\ (Y\ i) \implies t \in \mathcal{D}\ ?lhs \rangle$ 
    by (auto simp add: Seqptick-projs limproc-is-thelub  $\langle \text{chain}\ Y \rangle$  LUB-projs)
    next
    assume  $\langle \neg (\forall i. t1 \in \mathcal{D}\ (Y\ i)) \rangle$ 
    then obtain j where  $***$  :  $\langle j \leq i \implies t1 \notin \mathcal{D}\ (Y\ i) \rangle$  for  $i$ 
    by (meson  $\langle \text{chain}\ Y \rangle$  in-mono le-approx-def po-class.chain-mono)
    hence  $\langle j \leq i \implies t1 \notin T1\ i \rangle$  for  $i$  by (simp add: T1-def)
    with  $*$  have  $\langle j \leq i \implies t1 \in T2\ i \rangle$  for  $i$  by blast
    then obtain r where  $\langle t1\ @\ [\checkmark(r)] \in \mathcal{T}\ (Y\ j) \rangle$   $\langle t2 \in \mathcal{D}\ (S\ r) \rangle$ 
    unfolding T2-def by (auto simp add:  $**(1)$ )
    from this(1)  $\langle \text{chain}\ Y \rangle$   $***$  have  $\langle j \leq i \implies t1\ @\ [\checkmark(r)] \in \mathcal{T}\ (Y\ i) \rangle$  for  $i$ 
    by (metis eq-imp-le is-processT9 le-approx2T po-class.chain-mono)
    hence  $\langle t1\ @\ [\checkmark(r)] \in \mathcal{T}\ (\bigsqcup i. Y\ i) \rangle$ 
    by (meson  $***$   $\langle \text{chain}\ Y \rangle$  dual-order.refl is-processT9 is-ub-thelub le-approx2T)
    with  $\langle t2 \in \mathcal{D}\ (S\ r) \rangle$   $**(1, 2)$  show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
    by (auto simp add: Seqptick-projs)
    qed
  qed
  thus  $\langle t \in \mathcal{D}\ ?rhs \implies t \in \mathcal{D}\ ?lhs \rangle$  for  $t$ 
  by (meson D-imp-front-tickFree div-butlast-when-non-tickFree-iff front-tickFree-iff-tickFree-butlast)
next
  show  $\langle (t, X) \in \mathcal{F}\ ?lhs \implies (t, X) \in \mathcal{F}\ ?rhs \rangle$  for  $t\ X$ 

```


by (simp add: Seq_{ptick}-projs limproc-is-thelub ch2ch-fun ⟨chain Y⟩ lub-fun
 chain-Seq_{ptick}-left LUB-projs) blast
 next
 fix t X assume ⟨(t, X) ∈ $\mathcal{F}$?rhs⟩ ⟨t ∉ $\mathcal{D}$?rhs⟩
 from ⟨t ∉ $\mathcal{D}$?rhs⟩ obtain j where ⟨t ∉ $\mathcal{D}$ (Y j ; $\checkmark$ S)⟩
 by (auto simp add: limproc-is-thelub chain-Seq_{ptick}-left ⟨chain Y⟩ LUB-projs)
 moreover from ⟨(t, X) ∈ $\mathcal{F}$?rhs⟩ have ⟨(t, X) ∈ $\mathcal{F}$ (Y j ; $\checkmark$ S)⟩
 by (simp add: limproc-is-thelub chain-Seq_{ptick}-left ⟨chain Y⟩ F-LUB)
 ultimately show ⟨(t, X) ∈ $\mathcal{F}$?lhs⟩
 by (fact le-approx2[OF mono-Seq_{ptick}[OF is-ub-thelub[OF ⟨chain Y⟩] below-refl],
 THEN iffD2])
 qed

lemma ⟨finite R $\implies$ chain Y $\implies$ $\sqcap r \in R$. $(\sqcup i. Y i r) = (\sqcup i. \sqcap r \in R. Y i r)$ ⟩
 by (subst cont2contlubE[of ⟨GlobalNdet R⟩, symmetric])
 (simp-all add: lub-fun)

lemma infinite-GlobalNdet-not-cont :

— This is a counter example.

defines Y-def : ⟨Y $\equiv \lambda i r :: \text{nat. if } r \leq i \text{ then STOP else } \perp :: (\text{nat}, \text{nat})$ ⟩
 process_{ptick}⟩
 shows ⟨chain Y⟩ ⟨ $\sqcap r \in \text{UNIV. } (\sqcup i. Y i r) \neq (\sqcup i. \sqcap r \in \text{UNIV. } Y i r)$ ⟩
 proof —
 show * : ⟨chain Y⟩ unfolding Y-def by (auto intro!: chainI fun-belowI)
 have ** : ⟨chain $(\lambda i. Y i r)$ ⟩ for r by (simp add: ⟨chain Y⟩ ch2ch-fun)

 have ⟨ $(\sqcup i. Y i) = (\lambda r. \text{STOP})$ ⟩
 by (rule ext, simp add: STOP-iff-T lub-fun limproc-is-thelub T-LUB * **)
 (auto simp add: Y-def T-STOP split: if-split-asm)
 hence \$: ⟨ $\sqcap r \in \text{UNIV. } (\sqcup i. Y i r) = \text{STOP}$ ⟩
 by (simp add: GlobalNdet-is-STOP-iff * lub-fun)

 have ⟨ $\sqcap r \in \text{UNIV. } Y i r = \perp$ ⟩ for i
 by (simp add: BOT-iff-Nil-D D-GlobalNdet Y-def D-BOT)
 (use Suc-n-not-le-n in blast)
 hence \$\$: ⟨ $(\sqcup i. \sqcap r \in \text{UNIV. } Y i r) = \perp$ ⟩ by simp

 from \$ \$\$ show ⟨ $\sqcap r \in \text{UNIV. } (\sqcup i. Y i r) \neq (\sqcup i. \sqcap r \in \text{UNIV. } Y i r)$ ⟩ by simp
 qed

The same counter-example works for Seq_{ptick}.

lemma infinite-Seq_{ptick}-not-cont :

— This is a counter example.

defines P-def : ⟨P $\equiv \text{SKIPS UNIV} :: (\text{nat}, \text{nat})$ process_{ptick}⟩
 and Y-def : ⟨Y $\equiv \lambda i r :: \text{nat. if } r \leq i \text{ then STOP else } \perp :: (\text{nat}, \text{nat})$ process_{ptick}⟩
 shows ⟨chain Y⟩ ⟨P ; $\checkmark$ $(\sqcup i. Y i) \neq (\sqcup i. P ;\checkmark Y i)$ ⟩

proof –

show $\ast : \langle \text{chain } Y \rangle$ **unfolding** $Y\text{-def}$ **by** $(\text{auto intro!} : \text{chainI fun-belowI})$
have $\langle P ; \bigvee (\bigsqcup i. Y i) = \sqcap r \in \text{UNIV}. (\bigsqcup i. Y i r) \rangle$
by $(\text{simp add} : P\text{-def} \ast \text{lub-fun})$
also have $\langle \dots \neq (\bigsqcup i. \sqcap r \in \text{UNIV}. Y i r) \rangle$
unfolding $P\text{-def } Y\text{-def}$ **by** $(\text{fact infinite-GlobalNdet-not-cont}(2))$
also have $\langle (\bigsqcup i. \sqcap r \in \text{UNIV}. Y i r) = (\bigsqcup i. P ; \bigvee Y i) \rangle$
by $(\text{simp add} : P\text{-def})$
finally show $\langle P ; \bigvee (\bigsqcup i. Y i) \neq (\bigsqcup i. P ; \bigvee Y i) \rangle$.

qed

We must therefore find a condition under which $\text{Seq}_{\text{ptick}}$ is continuous.

private lemma *cont-right-prem-Seq_{ptick}* :

$\langle S ; \bigvee (\bigsqcup i. Y i) = (\bigsqcup i. S ; \bigvee Y i) \rangle$ (is $\langle ?lhs = ?rhs \rangle$) **if** $\langle \text{chain } Y \rangle$ **and** $\langle \mathbf{F}_{\bigvee}(S) \rangle$

— We have to add this hypothesis in the generalization.

proof (rule *Process-eq-optimizedI*)

show $\langle t \in \mathcal{D} \text{ ?lhs} \implies t \in \mathcal{D} \text{ ?rhs} \rangle$ **for** t

by $(\text{simp add} : \text{Seq}_{\text{ptick}}\text{-projs } \text{limproc-is-the lub } \text{ch2ch-fun } \langle \text{chain } Y \rangle \text{ lub-fun } \text{chain-Seq}_{\text{ptick}}\text{-right } D\text{-LUB})$ **blast**

next

have $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$ **if** $\langle t \in \mathcal{D} \text{ ?rhs} \rangle$ **and** $\langle tF t \rangle$ **for** t

proof (cases $\langle \text{map } (ev \circ of\text{-}ev) t \in \mathcal{D} S \rangle$)

show $\langle \text{map } (ev \circ of\text{-}ev) t \in \mathcal{D} S \implies t \in \mathcal{D} \text{ ?lhs} \rangle$

by $(\text{simp add} : \text{Seq}_{\text{ptick}}\text{-projs})$

$(\text{metis } \text{append.right-neutral } \text{front-tickFree-Nil } \langle tF t \rangle \text{ tickFree-map-ev-comp } \text{tickFree-map-ev-of-ev-eq-iff})$

next

define T **where** $\langle T i \equiv \{t1. \exists t2 r. t = \text{map } (ev \circ of\text{-}ev) t1 @ t2 \wedge t1 @ \bigvee(r) \in \mathcal{T} S \wedge tF t1 \wedge t2 \in \mathcal{D} (Y i r)\} \rangle$ **for** i

assume $\langle \text{map } (ev \circ of\text{-}ev) t \notin \mathcal{D} S \rangle$

with $\langle t \in \mathcal{D} \text{ ?rhs} \rangle$ **have** $\langle T i \neq \{\} \rangle$ **for** i

by $(\text{fastforce simp add} : T\text{-def } \text{limproc-is-the lub } \text{chain-Seq}_{\text{ptick}}\text{-right } \langle \text{chain } Y \rangle \text{ } D\text{-LUB } \text{Seq}_{\text{ptick}}\text{-projs } \text{is-processT7 } \text{tickFree-map-ev-of-ev-same-type-is})$

moreover have $\langle \text{finite } (T 0) \rangle$

unfolding $T\text{-def}$

by $(\text{rule } \text{finite-subset}[of - \langle \{u. u \leq \text{map } (ev \circ of\text{-}ev) t\} \rangle])$

$(\text{use } \text{tickFree-map-ev-of-ev-eq-iff} \text{ in } \langle \text{force simp add} : \text{prefixes-fin} \rangle) +$

moreover have $\langle T (Suc i) \subseteq T i \rangle$ **for** i

unfolding $T\text{-def}$ **by** $(\text{intro allI } Un\text{-mono } \text{subsetI} ; \text{simp})$

$(\text{metis } \langle \text{chain } Y \rangle \text{ fun-below-iff } \text{subset-iff}[of \langle \mathcal{D} (Y (Suc i) -) \rangle \langle \mathcal{D} (Y i -) \rangle])$

$\text{po-class.chainE le-approx1})$

ultimately have $\langle (\bigcap i. T i) \neq \{\} \rangle$ **by** (rule *Inter-nonempty-finite-chained-sets*)

then obtain $t1$ **where** $\langle \forall i. t1 \in T i \rangle$ **by** *auto*

then obtain $t2$ **where** $\ast : \langle t = \text{map } (ev \circ of\text{-}ev) t1 @ t2 \rangle$

$\langle tF t1 \rangle \langle \forall i. \exists r. t1 @ \bigvee(r) \in \mathcal{T} S \wedge t2 \in \mathcal{D} (Y i r) \rangle$

by $(\text{simp add} : T\text{-def})$ **blast**

have $\langle t1 \in \mathcal{T} S \rangle$ **by** $(\text{meson } \ast(3) \text{ prefixI } \text{is-processT3-TR})$

from $\ast(1, 2) \langle \text{map } (ev \circ of\text{-}ev) t \notin \mathcal{D} S \rangle$

have $\langle t1 \notin \mathcal{D} S \rangle$ **using** $\text{is-processT7 } \text{tickFree-map-ev-of-ev-eq-iff}$ **by** *fastforce*

```

define U where  $\langle U\ i \equiv \{r. t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S \wedge t2 \in \mathcal{D}\ (Y\ i\ r)\} \rangle$  for i
from  $\ast(\beta)$  have  $\langle U\ i \neq \{\} \rangle$  for i by (simp add: U-def)
moreover have  $\langle finite\ (U\ 0) \rangle$ 
proof (rule finite-subset[of -  $\langle \{r. t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S\} \rangle$ ])
  show  $\langle U\ 0 \subseteq \{r. t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S\} \rangle$  unfolding U-def by blast
next
  show  $\langle finite\ \{r. t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S\} \rangle$ 
  by (simp add:  $\langle \mathbb{F}_{\checkmark}(S) \rangle$   $\langle t1 \notin \mathcal{D}\ S \rangle$  finite-ticksD)
qed
moreover have  $\langle U\ (Suc\ i) \subseteq U\ i \rangle$  for i
  by (simp add: U-def subset-iff)
  (meson fun-below-iff in-mono le-approx1 chainE  $\langle chain\ Y \rangle$ )
ultimately have  $\langle (\bigcap i. U\ i) \neq \{\} \rangle$  by (rule Inter-nonempty-finite-chained-sets)
then obtain r where  $\ast\ast : \langle \forall i. r \in U\ i \rangle$  by auto
with  $\ast$  show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
  by (simp add: Seqptick-projs U-def  $\langle chain\ Y \rangle$  ch2ch-fun limproc-is-thelub
D-LUB lub-fun) blast
qed
thus  $\langle t \in \mathcal{D}\ ?rhs \implies t \in \mathcal{D}\ ?lhs \rangle$  for t
  by (meson D-imp-front-tickFree div-butlast-when-non-tickFree-iff front-tickFree-iff-tickFree-butlast)
next
  show  $\langle (t, X) \in \mathcal{F}\ ?lhs \implies (t, X) \in \mathcal{F}\ ?rhs \rangle$  for t X
  by (simp add: Seqptick-projs limproc-is-thelub ch2ch-fun  $\langle chain\ Y \rangle$  lub-fun
chain-Seqptick-right F-LUB) blast
next
  fix t X assume  $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$   $\langle t \notin \mathcal{D}\ ?rhs \rangle$ 
  from  $\langle t \notin \mathcal{D}\ ?rhs \rangle$  obtain j where  $\langle t \notin \mathcal{D}\ (S\ ;\checkmark\ Y\ j) \rangle$ 
  by (auto simp add: limproc-is-thelub chain-Seqptick-right  $\langle chain\ Y \rangle$  D-LUB)
  moreover from  $\langle (t, X) \in \mathcal{F}\ ?rhs \rangle$  have  $\langle (t, X) \in \mathcal{F}\ (S\ ;\checkmark\ Y\ j) \rangle$ 
  by (simp add: limproc-is-thelub chain-Seqptick-right  $\langle chain\ Y \rangle$  F-LUB)
  ultimately show  $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle$ 
  by (fact le-approx2[OF mono-Seqptick[OF below-refl is-ub-thelub[OF  $\langle chain\ Y \rangle$ ]], THEN iffD2])
qed

```

13.1.3 Continuity

We then spent a lot of time trying to prove the continuity under the assumption of *finite-ticks-fun*.

```

lemma Seqptick-cont [simp] :  $\langle cont\ (\lambda x. f\ x\ ;\checkmark\ g\ x) \rangle$ 
  if  $\langle cont\ f \rangle$  and  $\langle cont\ g \rangle$  and  $\langle \mathbb{F}_{\checkmark}(f) \rangle$ 
  for  $g :: \langle - \Rightarrow - \Rightarrow ('a, 's)\ process_{ptick} \rangle$ 
proof (rule cont-apply[where  $f = \langle \lambda x\ y. f\ x\ ;\checkmark\ y \rangle$ ])
  show  $\langle cont\ g \rangle$  by (fact  $\langle cont\ g \rangle$ )
next
  show  $\langle cont\ (\lambda x. f\ x\ ;\checkmark\ y) \rangle$  for  $y :: \langle - \Rightarrow ('a, 's)\ process_{ptick} \rangle$ 
  proof (rule contI2)
    show  $\langle monofun\ (\lambda x. f\ x\ ;\checkmark\ y) \rangle$  by (simp add: cont2monofunE mono-Seqptick
monofunI  $\langle cont\ f \rangle$ )

```

```

next
  show  $\langle \text{chain } Y \implies f (\bigsqcup i. Y i) ;_{\checkmark} y \sqsubseteq (\bigsqcup i. f (Y i) ;_{\checkmark} y) \rangle$  for  $Y$ 
  by (simp add: ch2ch-cont cont2conthubE cont-left-prem-Seqptick  $\langle \text{cont } f \rangle$ )
qed
next
  show  $\langle \text{cont } (\lambda y :: - \Rightarrow ('a, 's) \text{ process}_{\text{ptick}}. f x ;_{\checkmark} y) \rangle$  for  $x$ 
  proof (rule contI2)
    show  $\langle \text{monofun } ((;_{\checkmark}) (f x)) \rangle$  by (simp add: mono-Seqptick monofunI)
  next
    show  $\langle \text{chain } Y \implies f x ;_{\checkmark} (\bigsqcup i. Y i) \sqsubseteq (\bigsqcup i. f x ;_{\checkmark} Y i) \rangle$ 
    for  $Y :: \langle - \Rightarrow - \Rightarrow ('a, 's) \text{ process}_{\text{ptick}} \rangle$ 
    oops
    — Unfortunately here, we cannot use cont-right-prem-Seqptick since there is
    no reason for  $\mathbb{F}_{\checkmark}(x)$  to hold. Actually, we can find a counter example.

```

We could therefore only prove the weaker following version.

```

lemma Seqptick-cont [simp] :  $\langle \text{cont } (\lambda x. f x ;_{\checkmark} g x) \rangle$ 
  if  $\langle \text{cont } f \rangle$  and  $\langle \text{cont } g \rangle$  and  $\langle \bigwedge x. \mathbb{F}_{\checkmark}(f x) \rangle$ 
  for  $g :: \langle - \Rightarrow - \Rightarrow ('a, 's) \text{ process}_{\text{ptick}} \rangle$ 
proof (rule cont-apply[where  $f = \langle \lambda x y. f x ;_{\checkmark} y \rangle$ ])
  show  $\langle \text{cont } g \rangle$  by (fact  $\langle \text{cont } g \rangle$ )
next
  show  $\langle \text{cont } (\lambda x. f x ;_{\checkmark} y) \rangle$  for  $y :: \langle - \Rightarrow ('a, 's) \text{ process}_{\text{ptick}} \rangle$ 
  proof (rule contI2)
    show  $\langle \text{monofun } (\lambda x. f x ;_{\checkmark} y) \rangle$  by (simp add: cont2monofunE mono-Seqptick monofunI  $\langle \text{cont } f \rangle$ )
  next
    show  $\langle \text{chain } Y \implies f (\bigsqcup i. Y i) ;_{\checkmark} y \sqsubseteq (\bigsqcup i. f (Y i) ;_{\checkmark} y) \rangle$  for  $Y$ 
    by (simp add: ch2ch-cont cont2conthubE cont-left-prem-Seqptick  $\langle \text{cont } f \rangle$ )
  qed
next
  show  $\langle \text{cont } (\lambda y :: - \Rightarrow ('a, 's) \text{ process}_{\text{ptick}}. f x ;_{\checkmark} y) \rangle$  for  $x$ 
  proof (rule contI2)
    show  $\langle \text{monofun } ((;_{\checkmark}) (f x)) \rangle$  by (simp add: mono-Seqptick monofunI)
  next
    show  $\langle \text{chain } Y \implies f x ;_{\checkmark} (\bigsqcup i. Y i) \sqsubseteq (\bigsqcup i. f x ;_{\checkmark} Y i) \rangle$ 
    for  $Y :: \langle - \Rightarrow - \Rightarrow ('a, 's) \text{ process}_{\text{ptick}} \rangle$ 
    by (simp add: cont-right-prem-Seqptick  $\langle \bigwedge x. \mathbb{F}_{\checkmark}(f x) \rangle$ )
  qed
qed
end

```

```

corollary  $\langle \text{cont } f \implies \text{cont } g \implies \text{cont } (\lambda x. f x ;_{\checkmark} g x) \rangle$ 
  for  $f :: \langle 'b :: \text{cpo} \Rightarrow ('a, 'r :: \text{finite}) \text{ process}_{\text{ptick}} \rangle$ 
  by (simp add: finite-ticks-simps(5))

```

```

lemma MultiSeqptick-cont[simp]:
  ⟨[ $\bigwedge l. l \in \text{set } L \implies \text{cont } (f\ l); \bigwedge l\ r\ x. l \in \text{set } (\text{butlast } L) \implies \mathbb{F}_{\checkmark}(f\ l\ x\ r)$ ]]
     $\implies \text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ L. f\ l\ x)\ r)$ ⟩
proof (induct L arbitrary: r)
  show ⟨ $\bigwedge r. \text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ []. f\ l\ x)\ r)$ ⟩ by simp
next
  case (Cons l0 L)
  show ⟨ $\text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ (l0 \# L). f\ l\ x)\ r)$ ⟩
  proof (cases ⟨ $L = []$ ⟩)
    show ⟨ $L = [] \implies \text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ (l0 \# L). f\ l\ x)\ r)$ ⟩
    by (simp add: Cons.premis(1) cont2cont-fun)
  next
  show ⟨ $\text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ (l0 \# L). f\ l\ x)\ r)$ ⟩ if ⟨ $L \neq []$ ⟩
  proof (subst MultiSeqptick-Cons, intro cont2cont-lambda Seqptick-cont)
    show ⟨ $\text{cont } (\lambda x. f\ l0\ x\ r)$ ⟩ by (simp add: Cons.premis(1) cont2cont-fun)
  next
  have ⟨ $\text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ L. f\ l\ x))$ ⟩
  by (rule cont2cont-lambda, rule Cons.hyps)
    (simp-all add: Cons.premis(1, 2) ⟨ $L \neq []$ ⟩)
  thus ⟨ $\text{cont } (\lambda x. (\text{SEQ}_{\checkmark}\ l \in @\ L. f\ l\ x)\ y)$ ⟩ for y
  by (fact cont2cont-fun)
  next
  show ⟨ $\mathbb{F}_{\checkmark}(f\ l0\ x\ r)$ ⟩ for x by (simp add: Cons.premis(2) that)
  qed
qed
qed

```

13.2 Synchronization Product

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

13.2.1 Monotonicity

```

lemma mono-Syncptick : ⟨ $P \llbracket A \rrbracket_{\checkmark} Q \sqsubseteq P' \llbracket A \rrbracket_{\checkmark} Q'$ ⟩ if ⟨ $P \sqsubseteq P'$ ⟩ and ⟨ $Q \sqsubseteq Q'$ ⟩
proof (unfold le-approx-def Refusals-after-def, safe)
  from le-approx1[OF ⟨ $P \sqsubseteq P'$ ⟩] le-approx-lemma-T[OF ⟨ $P \sqsubseteq P'$ ⟩]
    le-approx1[OF ⟨ $Q \sqsubseteq Q'$ ⟩] le-approx-lemma-T[OF ⟨ $Q \sqsubseteq Q'$ ⟩]
  show ⟨ $t \in \mathcal{D} (P' \llbracket A \rrbracket_{\checkmark} Q') \implies t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q)$ ⟩ for t
  by (simp add: D-Syncptick) fast
next
  from le-approx2[OF ⟨ $P \sqsubseteq P'$ ⟩] le-approx2[OF ⟨ $Q \sqsubseteq Q'$ ⟩]
  show ⟨ $t \notin \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \implies (t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\checkmark} Q) \implies$ 

```

$(t, X) \in \mathcal{F} (P' \llbracket A \rrbracket_{\checkmark} Q') \rangle$ **for** $t X$
by (*simp add: Sync_{ptick}-projs', elim disjE*)
(metis F-T front-tickFree-Nil self-append-conv, metis)
next
from *le-approx-lemma-F*[*OF* $\langle P \sqsubseteq P' \rangle$] *le-approx-lemma-F*[*OF* $\langle Q \sqsubseteq Q' \rangle$]
le-approx1[*OF* $\langle P \sqsubseteq P' \rangle$] *le-approx-lemma-T*[*OF* $\langle P \sqsubseteq P' \rangle$]
le-approx1[*OF* $\langle Q \sqsubseteq Q' \rangle$] *le-approx-lemma-T*[*OF* $\langle Q \sqsubseteq Q' \rangle$]
show $\langle t \notin \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \implies (t, X) \in \mathcal{F} (P' \llbracket A \rrbracket_{\checkmark} Q') \implies$
 $(t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$ **for** $t X$
by (*simp add: Sync_{ptick}-projs subset-iff, elim disjE*) *metis+*
next
fix t **assume** $\langle t \in \min\text{-elems} (\mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q)) \rangle$
hence $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$ **by** (*fact elem-min-elems*)
then obtain $u v t\text{-}P t\text{-}Q$
where $*$: $\langle t = u @ v \rangle \langle tF u \rangle \langle ftF v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P, t\text{-}Q), A) \rangle$
 $\langle t\text{-}P \in \mathcal{D} P \wedge t\text{-}Q \in \mathcal{T} Q \vee t\text{-}P \in \mathcal{T} P \wedge t\text{-}Q \in \mathcal{D} Q \rangle$
unfolding *D-Sync_{ptick}* **by** *blast*
have $\langle v = [] \rangle$
proof (*rule ccontr*)
assume $\langle v \neq [] \rangle$
with $*(1)$ **have** $\langle u < t \rangle$ **by** (*simp add: dual-order.strict-iff-not*)
moreover from $*(2,4,5)$ **have** $\langle u \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: D-Sync_{ptick}*) (*use front-tickFree-Nil in blast*)
ultimately show *False*
using $\langle t \in \min\text{-elems} (\mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q)) \rangle$ *min-elems-no order-less-imp-le* **by**
blast
qed

have $\langle t\text{-}P \in \min\text{-elems} (\mathcal{D} P) \rangle$ **if** $\langle t\text{-}P \in \mathcal{D} P \rangle$
proof (*rule ccontr*)
assume $\langle t\text{-}P \notin \min\text{-elems} (\mathcal{D} P) \rangle$
with $\langle t\text{-}P \in \mathcal{D} P \rangle$ **obtain** $t\text{-}P'$ **where** $\langle t\text{-}P' < t\text{-}P \rangle \langle t\text{-}P' \in \mathcal{D} P \rangle$
by (*metis antisym-conv2 elem-min-elems min-elems5*)
from *setinterleaves_{ptick}-less-prefixL*[*OF* $*(4) \langle t\text{-}P' < t\text{-}P \rangle$]
obtain $u' t\text{-}Q'$
where $\$$: $\langle u' < u \rangle \langle t\text{-}Q' \leq t\text{-}Q \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark tick\text{-}join} ((t\text{-}P', t\text{-}Q'), A) \rangle$ **by** *blast*
from $*(5)$ *D-T* **have** $\langle t\text{-}Q \in \mathcal{T} Q \rangle$ **by** *blast*
with $*(2,3) \langle t\text{-}P' \in \mathcal{D} P \rangle$ **have** $\langle u' \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: D-Sync_{ptick}*)
(metis append.right-neutral front-tickFree-Nil is-processT3-TR)
moreover from $\langle u' < u \rangle$ **have** $\langle u' < t \rangle$
by (*simp add: *(1) (meson Prefix-Order.prefixI dual-order.strict-trans1)*)
ultimately show *False*
using $\langle t \in \min\text{-elems} (\mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q)) \rangle$ *min-elems-no nless-le* **by** *blast*
qed
with $*(5)$ **have** $\langle t\text{-}P \in \mathcal{T} P' \rangle$
by (*meson in-mono le-approx2T le-approx3* $\langle P \sqsubseteq P' \rangle$)

have $\langle t.Q \in \text{min-elems } (\mathcal{D} \ Q) \rangle$ **if** $\langle t.Q \in \mathcal{D} \ Q \rangle$
proof (rule *ccontr*)
assume $\langle t.Q \notin \text{min-elems } (\mathcal{D} \ Q) \rangle$
with $\langle t.Q \in \mathcal{D} \ Q \rangle$ **obtain** $t.Q'$ **where** $\langle t.Q' < t.Q \rangle \ \langle t.Q' \in \mathcal{D} \ Q \rangle$
by (*metis antisym-conv2 elem-min-elems min-elems5*)
from *setinterleaves_{ptick}-less-prefixR*[*OF* $\ast(4) \ \langle t.Q' < t.Q \rangle$]
obtain $u' \ t.P'$
where $\$: \langle u' < u \rangle \ \langle t.P' \leq t.P \rangle$
 $\langle u' \text{ setinterleaves}_{\check{\text{tick-join}}} ((t.P', t.Q'), A) \rangle$ **by** *blast*
from $\ast(5) \ D.T$ **have** $\langle t.P \in \mathcal{T} \ P \rangle$ **by** *blast*
with $\$(2,3) \ \langle t.Q' \in \mathcal{D} \ Q \rangle$ **have** $\langle u' \in \mathcal{D} \ (P \llbracket A \rrbracket_{\check{\text{tick}}}) \rangle$
by (*simp add: D-Sync_{ptick}*)
(*metis append.right-neutral front-tickFree-Nil is-processT3-TR*)
moreover from $\langle u' < u \rangle$ **have** $\langle u' < t \rangle$
by (*simp add: $\ast(1)$ (meson Prefix-Order.prefixI dual-order.strict-trans1)*)
ultimately show *False*
using $\langle t \in \text{min-elems } (\mathcal{D} \ (P \llbracket A \rrbracket_{\check{\text{tick}}})) \rangle$ *min-elems-no nless-le* **by** *blast*
qed
with $\ast(5)$ **have** $\langle t.Q \in \mathcal{T} \ Q' \rangle$
by (*meson in-mono le-approx2T le-approx3 $\langle Q \sqsubseteq Q' \rangle$*)
from $\langle t.P \in \mathcal{T} \ P' \rangle \ \langle t.Q \in \mathcal{T} \ Q' \rangle \ \ast(4)$ **show** $\langle t \in \mathcal{T} \ (P' \llbracket A \rrbracket_{\check{\text{tick}}}) \rangle$
by (*auto simp add: $\ast(1) \ \langle v = [] \rangle \ T\text{-Sync}_{\text{ptick}}$*)
qed

13.2.2 Preliminaries

lemma *chain-Sync_{ptick}-left* : $\langle \text{chain } Y \implies \text{chain } (\lambda i. Y \ i \llbracket A \rrbracket_{\check{\text{tick}}}) \rangle$
and *chain-Sync_{ptick}-right* : $\langle \text{chain } Z \implies \text{chain } (\lambda i. P \llbracket A \rrbracket_{\check{\text{tick}}} \ Z \ i) \rangle$
by (*simp-all add: chain-def mono-Sync_{ptick}*)

lemma *cont-left-prem-Sync_{ptick}* :
 $\langle (\bigsqcup i. Y \ i) \llbracket A \rrbracket_{\check{\text{tick}}} Q = (\bigsqcup i. Y \ i \llbracket A \rrbracket_{\check{\text{tick}}} Q) \rangle$ **if** *chain*: $\langle \text{chain } Y \rangle$
proof (rule *Process-eq-optimizedI*)
show $\langle t \in \mathcal{D} \ ((\bigsqcup i. Y \ i) \llbracket A \rrbracket_{\check{\text{tick}}} Q) \implies t \in \mathcal{D} \ (\bigsqcup i. Y \ i \llbracket A \rrbracket_{\check{\text{tick}}} Q) \rangle$ **for** t
by (*simp add: limproc-is-thelub chain chain-Sync_{ptick}-left D-Sync_{ptick} D-LUB T-LUB*) *blast*
next
show $\langle (t, X) \in \mathcal{F} \ ((\bigsqcup i. Y \ i) \llbracket A \rrbracket_{\check{\text{tick}}} Q) \implies (t, X) \in \mathcal{F} \ (\bigsqcup i. Y \ i \llbracket A \rrbracket_{\check{\text{tick}}} Q) \rangle$ **for** $t \ X$
by (*simp add: limproc-is-thelub chain chain-Sync_{ptick}-left F-Sync_{ptick} D-LUB T-LUB F-LUB*) *blast*
next
fix t
assume $\langle t \in \mathcal{D} \ (\bigsqcup i. Y \ i \llbracket A \rrbracket_{\check{\text{tick}}} Q) \rangle$
define S
where $\langle S \ i \equiv \{(t.Y, t.Q, u). \exists v. tF \ u \wedge ftF \ v \wedge t = u @ v \wedge$

$u \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-Y, t-Q), A) \wedge$
 $(t-Y \in \mathcal{D} (Y i) \wedge t-Q \in \mathcal{T} Q \vee t-Y \in \mathcal{T} (Y i) \wedge$

$t-Q \in \mathcal{D} Q)\rangle \text{ for } i$
from $\langle t \in \mathcal{D} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$ **have** $\langle S i \neq \{\} \rangle$ **for** i
by (*simp add: S-def limproc-is-thelub chain chain-Sync_{ptick}-left D-Sync_{ptick} D-LUB*) *fast*
moreover have $\langle \text{finite } (S 0) \rangle$
by (*rule finite-subset[OF - finite-setinterleaves_{ptick}-tick-join-Sync_{ptick}]*)
(auto simp add: S-def)
moreover from *le-approx1[OF po-class.chainE[OF chain]] D-T*
le-approx2T[OF po-class.chainE[OF chain]]
have $\langle S (Suc i) \subseteq S i \rangle$ **for** i **by** (*simp add: S-def*) *blast*
ultimately have $\langle (\bigcap i. S i) \neq \{\} \rangle$ **by** (*rule Inter-nonempty-finite-chained-sets*)
then obtain $t-Y \ t-Q \ u$ **where** $\langle (t-Y, t-Q, u) \in (\bigcap i. S i) \rangle$ **by** *auto*
hence $\langle tF \ u \wedge ftF \ (\text{drop } (\text{length } u) \ t) \wedge$
 $t = u @ \text{drop } (\text{length } u) \ t \wedge u \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((t-Y, t-Q), A) \wedge$
 $(\forall i. t-Y \in \mathcal{D} (Y i)) \wedge t-Q \in \mathcal{T} Q \vee (\forall i. t-Y \in \mathcal{T} (Y i)) \wedge t-Q \in \mathcal{D} Q \rangle$
by (*auto simp add: S-def*) (*meson chain-lemma le-approx1 le-approx-lemma-T subsetD chain*)
show $\langle t \in \mathcal{D} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: limproc-is-thelub chain D-Sync_{ptick} T-LUB D-LUB*)
(use <?this> in blast)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$ $\langle t \notin \mathcal{D} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$
have $\langle Y i \sqsubseteq (\bigsqcup i. Y i) \rangle$ **for** i **by** (*simp add: is-ub-thelub <chain Y>*)
moreover from $\langle t \notin \mathcal{D} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$ **obtain** j **where** $\langle t \notin \mathcal{D} (Y j \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*auto simp add: limproc-is-thelub chain-Sync_{ptick}-left <chain Y> D-LUB*)
moreover from $\langle (t, X) \in \mathcal{F} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$ **have** $\langle (t, X) \in \mathcal{F} (Y j \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: limproc-is-thelub chain-Sync_{ptick}-left <chain Y> F-LUB*)
ultimately show $\langle (t, X) \in \mathcal{F} (\bigsqcup i. Y i \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*metis (mono-tags, lifting) below-refl le-approx2 mono-Sync_{ptick}*)
qed

lemma (*in Sync_{ptick}-locale*) *cont-right-prem-Sync_{ptick}* :
 $\langle P \llbracket A \rrbracket_{\checkmark} (\bigsqcup i. Z i) = (\bigsqcup i. P \llbracket A \rrbracket_{\checkmark} Z i) \rangle$ **if** $\langle \text{chain } Z \rangle$
by (*subst (1 2) Sync_{ptick}-locale-sym.Sync_{ptick}-sym*)
(simp add: Sync_{ptick}-locale-sym.cont-left-prem-Sync_{ptick}[OF <chain Z>])

13.2.3 Continuity

lemma *Sync_{ptick}-cont[simp]*: $\langle \text{cont } (\lambda x. f \ x \llbracket A \rrbracket_{\checkmark} \ g \ x) \rangle$ **if** $\langle \text{cont } f \rangle \ \langle \text{cont } g \rangle$
proof (*rule cont-apply[where f = <λx y. y [A]_✓ g x>]*)
from $\langle \text{cont } f \rangle$ **show** $\langle \text{cont } f \rangle$.
next
show $\langle \text{cont } (\lambda y. y \llbracket A \rrbracket_{\checkmark} \ g \ x) \rangle$ **for** x
proof (*rule contI2*)


```

  show ⟨monofun (λy. y ⟦A⟧✓ g x)⟩ by (simp add: monofunI mono-Syncptick)
next
  show ⟨chain Y ⇒ (⊔ i. Y i) ⟦A⟧✓ g x ⊆ (⊔ i. Y i ⟦A⟧✓ g x)⟩ for Y
    by (simp add: cont-left-prem-Syncptick)
qed
next
  show ⟨cont (λx. y ⟦A⟧✓ g x)⟩ for y
  proof (rule cont-compose[of ⟨λx. y ⟦A⟧✓ x⟩])
    show ⟨cont (λx. y ⟦A⟧✓ x)⟩
    proof (rule contI2)
      show ⟨monofun (Syncptick y A)⟩ by (simp add: monofunI mono-Syncptick)
    next
      show ⟨chain Z ⇒ y ⟦A⟧✓ (⊔ i. Z i) ⊆ (⊔ i. y ⟦A⟧✓ Z i)⟩ for Z
        by (simp add: cont-right-prem-Syncptick)
    qed
  next
    from ⟨cont g⟩ show ⟨cont g⟩ .
  qed
qed
end

```

lemma *MultiSync_{ptick}-cont* [simp] :

⟨(⋀ l. l ∈ set L ⇒ cont (P l)) ⇒ cont (λx. ⟦S⟧✓ l ∈@ L. P l x)⟩

by (induct L rule: induct-list012)

(auto intro: RenamingTick-cont inj-imp-finitary injI)

Chapter 14

Monotonicity Properties

14.0.1 Sequential Composition

lemma *mono-Seq_{ptick}-FD* : $\langle P \sqsubseteq_{FD} P' \implies (\bigwedge r. Q \ r \sqsubseteq_{FD} Q' \ r) \implies P ; \checkmark Q \sqsubseteq_{FD} P' ; \checkmark Q' \rangle$

proof (*rule trans-FD*[*of* - $\langle P' ; \checkmark Q' \rangle$])

show $\langle P \sqsubseteq_{FD} P' \implies (\bigwedge r. Q \ r \sqsubseteq_{FD} Q' \ r) \implies P ; \checkmark Q \sqsubseteq_{FD} P' ; \checkmark Q' \rangle$

unfolding *refine-defs Seq_{ptick}-projs*

by (*auto simp add: subset-iff T-F-spec[symmetric]*)

next

show $\langle P \sqsubseteq_{FD} P' \implies (\bigwedge r. Q \ r \sqsubseteq_{FD} Q' \ r) \implies P' ; \checkmark Q \sqsubseteq_{FD} P' ; \checkmark Q' \rangle$

unfolding *less-eq-process_{ptick}-def Seq_{ptick}-projs*

by (*simp add: subset-iff T-F-spec[symmetric]*) *metis*

qed

lemma *mono-Seq_{ptick}-DT* : $\langle P \sqsubseteq_{DT} P' \implies (\bigwedge r. Q \ r \sqsubseteq_{DT} Q' \ r) \implies P ; \checkmark Q \sqsubseteq_{DT} P' ; \checkmark Q' \rangle$

proof (*rule trans-DT*[*of* - $\langle P' ; \checkmark Q' \rangle$])

show $\langle P ; \checkmark Q \sqsubseteq_{DT} P' ; \checkmark Q' \rangle$ **if** $\langle P \sqsubseteq_{DT} P' \rangle$

proof (*rule trace-divergence-refine-optimizedI*)

from $\langle P \sqsubseteq_{DT} P' \rangle$ **show** $\langle s \in \mathcal{D} (P' ; \checkmark Q) \implies s \in \mathcal{D} (P ; \checkmark Q) \rangle$ **for** s

by (*auto simp add: refine-defs Seq_{ptick}-projs*)

next

from $\langle P \sqsubseteq_{DT} P' \rangle$ **show** $\langle s \in \mathcal{T} (P' ; \checkmark Q) \implies s \in \mathcal{T} (P ; \checkmark Q) \rangle$ **for** s

by (*auto simp add: Seq_{ptick}-projs refine-defs*)

qed

next

show $\langle (\bigwedge r. Q \ r \sqsubseteq_{DT} Q' \ r) \implies P' ; \checkmark Q \sqsubseteq_{DT} P' ; \checkmark Q' \rangle$

by (*simp add: refine-defs Seq_{ptick}-projs*) *blast*

qed

lemma *mono-Seq_{ptick}-F-right* : $\langle (\bigwedge r. Q \ r \sqsubseteq_F Q' \ r) \implies P ; \checkmark Q \sqsubseteq_F P ; \checkmark Q' \rangle$

by (*auto simp add: failure-refine-def Seq_{ptick}-projs*) *blast*

lemma *mono-Seq_{ptick}-D-right* : $\langle (\bigwedge r. Q \ r \sqsubseteq_D Q' \ r) \implies P ; \checkmark Q \sqsubseteq_D P ; \checkmark Q' \rangle$

by (simp add: divergence-refine-def Seq_{ptick}-projs) blast

lemma mono-Seq_{ptick}-T-right : $\langle \bigwedge r. Q \ r \sqsubseteq_T Q' \ r \rangle \implies P \ ;\checkmark Q \sqsubseteq_T P \ ;\checkmark Q' \rangle$
 by (simp add: trace-refine-def Seq_{ptick}-projs) blast

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} :
 $\langle \bigwedge x \ r. x \in \text{set } L \implies P \ x \ r \sqsubseteq Q \ x \ r \rangle \implies$
 $(SEQ_{\checkmark} \ l \in @ \ L. \ P \ l) \ r \sqsubseteq (SEQ_{\checkmark} \ l \in @ \ L. \ Q \ l) \ r \rangle$
 by (induct L arbitrary: r, simp-all add: fun-belowI mono-Seq_{ptick})

lemma mono-MultiSeq_{ptick}-FD :
 $\langle \bigwedge x \ r. x \in \text{set } L \implies P \ x \ r \sqsubseteq_{FD} Q \ x \ r \rangle \implies$
 $(SEQ_{\checkmark} \ l \in @ \ L. \ P \ l) \ r \sqsubseteq_{FD} (SEQ_{\checkmark} \ l \in @ \ L. \ Q \ l) \ r \rangle$
and mono-MultiSeq_{ptick}-DT :
 $\langle \bigwedge x \ r. x \in \text{set } L \implies P \ x \ r \sqsubseteq_{DT} Q \ x \ r \rangle \implies$
 $(SEQ_{\checkmark} \ l \in @ \ L. \ P \ l) \ r \sqsubseteq_{DT} (SEQ_{\checkmark} \ l \in @ \ L. \ Q \ l) \ r \rangle$
 by (induct L arbitrary: r, simp-all add: monos-Seq_{ptick})

lemmas monos-MultiSeq_{ptick} =
 mono-MultiSeq_{ptick} mono-MultiSeq_{ptick}-FD mono-MultiSeq_{ptick}-DT

14.0.3 Synchronization Product

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

lemma mono-Sync_{ptick}-DT :
 $\langle P \sqsubseteq_{DT} P' \rangle \implies \langle Q \sqsubseteq_{DT} Q' \rangle \implies \langle P \llbracket A \rrbracket_{\checkmark} Q \sqsubseteq_{DT} P' \llbracket A \rrbracket_{\checkmark} Q' \rangle$
 by (simp add: refine-defs T-Sync_{ptick} D-Sync_{ptick}) blast

lemma mono-Sync_{ptick}-FD : $\langle P \llbracket A \rrbracket_{\checkmark} Q \sqsubseteq_{FD} P' \llbracket A \rrbracket_{\checkmark} Q' \rangle$
 if $\langle P \sqsubseteq_{FD} P' \rangle$ **and** $\langle Q \sqsubseteq_{FD} Q' \rangle$

proof –

from $\langle P \sqsubseteq_{FD} P' \rangle \ \langle Q \sqsubseteq_{FD} Q' \rangle$ **have** $\langle P \sqsubseteq_{DT} P' \rangle \ \langle Q \sqsubseteq_{DT} Q' \rangle$

by (simp-all add: le-ref2T refine-defs)

with mono-Sync_{ptick}-DT **have** $\langle P \llbracket A \rrbracket_{\checkmark} Q \sqsubseteq_{DT} P' \llbracket A \rrbracket_{\checkmark} Q' \rangle$ **by** blast

hence * : $\langle P \llbracket A \rrbracket_{\checkmark} Q \sqsubseteq_D P' \llbracket A \rrbracket_{\checkmark} Q' \rangle$ **by** (simp add: leDT-imp-leD)

show $\langle P \llbracket A \rrbracket_{\checkmark} Q \sqsubseteq_{FD} P' \llbracket A \rrbracket_{\checkmark} Q' \rangle$

proof (rule leF-leD-imp-leFD[OF - *],

unfold failure-refine-def, safe)

fix $t \ X$ **assume** $\langle t, X \rangle \in \mathcal{F} (P' \llbracket A \rrbracket_{\checkmark} Q')$

then consider $\langle t \in \mathcal{D} (P' \llbracket A \rrbracket_{\checkmark} Q') \rangle$

| (fail) $t \cdot P \ t \cdot Q \ X \cdot P \ X \cdot Q$

where $\langle t \cdot P, X \cdot P \rangle \in \mathcal{F} P' \ \langle t \cdot Q, X \cdot Q \rangle \in \mathcal{F} Q'$

```

    < t setinterleaves✓ tick-join ((t-P, t-Q), A) >
    < X ⊆ super-ref-Syncptick tick-join X-P A X-Q >
  unfolding Syncptick-projs by blast
thus < (t, X) ∈  $\mathcal{F}$  (P  $\llbracket A \rrbracket_{\checkmark}$  Q) >
proof cases
  show < t ∈  $\mathcal{D}$  (P'  $\llbracket A \rrbracket_{\checkmark}$  Q') ⇒ (t, X) ∈  $\mathcal{F}$  (P  $\llbracket A \rrbracket_{\checkmark}$  Q) >
    using * D-F unfolding divergence-refine-def by blast
next
case fail
from fail(1, 2) < P ⊆FD P' > < Q ⊆FD Q' >
have < (t-P, X-P) ∈  $\mathcal{F}$  P > < (t-Q, X-Q) ∈  $\mathcal{F}$  Q >
  unfolding refine-defs by auto
with fail(3, 4) show < (t, X) ∈  $\mathcal{F}$  (P  $\llbracket A \rrbracket_{\checkmark}$  Q) >
  by (auto simp add: F-Syncptick)
qed
qed
qed

```

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} :
 < (⋀ l. l ∈ set L ⇒ P l ⊆ Q l) ⇒ $\llbracket S \rrbracket_{\checkmark} l \in @ L. P l \subseteq \llbracket S \rrbracket_{\checkmark} l \in @ L. Q l$ >
 by (induct L rule: induct-list012)
 (simp-all add: Sync_{Rlist}.mono-Sync_{ptick} mono-Renaming)

lemma mono-MultiSync_{ptick}-FD :
 < (⋀ l. l ∈ set L ⇒ P l ⊆_{FD} Q l) ⇒ $\llbracket S \rrbracket_{\checkmark} l \in @ L. P l \subseteq_{FD} \llbracket S \rrbracket_{\checkmark} l \in @ L. Q l$ >
 by (induct L rule: induct-list012)
 (simp-all add: Sync_{Rlist}.mono-Sync_{ptick}-FD mono-Renaming-FD)

lemma mono-MultiSync_{ptick}-DT :
 < (⋀ l. l ∈ set L ⇒ P l ⊆_{DT} Q l) ⇒ $\llbracket S \rrbracket_{\checkmark} l \in @ L. P l \subseteq_{DT} \llbracket S \rrbracket_{\checkmark} l \in @ L. Q l$ >
 by (induct L rule: induct-list012)
 (simp-all add: Sync_{Rlist}.mono-Sync_{ptick}-DT mono-Renaming-DT)

lemmas monos-MultiSync_{ptick} =
 mono-MultiSync_{ptick} mono-MultiSync_{ptick}-FD mono-MultiSync_{ptick}-DT

Chapter 15

Non Destructiveness Rules

15.1 Synchronization Product

15.1.1 Refinement

lemma (in *Sync_{ptick}-locale*) *restriction-process_{ptick}-Sync_{ptick}-FD-div-oneside* :

assumes $\langle tF\ u \rangle \langle ftF\ v \rangle \langle t-P \in \mathcal{D}\ (P \downarrow n) \rangle \langle t-Q \in \mathcal{T}\ (Q \downarrow n) \rangle$
 $\langle u\ \text{setinterleaves}_{\checkmark tick-join}\ ((t-P, t-Q), A) \rangle$

shows $\langle u\ @\ v \in \mathcal{D}\ (P\ \llbracket A \rrbracket_{\checkmark}\ Q \downarrow n) \rangle$

proof (*insert assms(3, 4), elim D-restriction-process_{ptick}E T-restriction-process_{ptick}E*)
from *assms(1, 2, 5)* **show** $\langle t-P \in \mathcal{D}\ P \implies t-Q \in \mathcal{T}\ Q \implies u\ @\ v \in \mathcal{D}\ (P\ \llbracket A \rrbracket_{\checkmark}\ Q \downarrow n) \rangle$

by (*auto simp add: D-restriction-process_{ptick} D-Sync_{ptick}*)

next

fix $t-Q'\ t-Q''$

assume $*$: $\langle t-P \in \mathcal{D}\ P \rangle \langle \text{length } t-P \leq n \rangle \langle t-Q = t-Q' @ t-Q'' \rangle$
 $\langle t-Q' \in \mathcal{T}\ Q \rangle \langle \text{length } t-Q' = n \rangle \langle tF\ t-Q' \rangle \langle ftF\ t-Q'' \rangle$

from $\langle t-Q = t-Q' @ t-Q'' \rangle$ **have** $\langle t-Q' \leq t-Q \rangle$ **by** *simp*

from *setinterleaves_{ptick}-le-prefixR[OF assms(5) this]*

obtain $t-P'\ t-P''\ u'\ u''$

where $**$: $\langle u = u' @ u'' \rangle \langle t-P = t-P' @ t-P'' \rangle$
 $\langle u'\ \text{setinterleaves}_{\checkmark tick-join}\ ((t-P', t-Q'), A) \rangle$

by (*meson Prefix-Order.prefixE*)

from *assms(1)* $\langle u = u' @ u'' \rangle$ **have** $\langle tF\ u' \rangle$ **by** *auto*

moreover from $*(1,4)\ ***(2,3)$ **have** $\langle u' \in \mathcal{T}\ (P\ \llbracket A \rrbracket_{\checkmark}\ Q) \rangle$

by (*simp add: T-Sync_{ptick}*) (*metis D-T is-processT3-TR-append*)

moreover have $\langle \text{length } t-Q' \leq \text{length } u' \rangle$

using $*(3)$ *setinterleaves_{ptick}-imp-lengthLR-le* **by** *blast*

ultimately have $\langle u' \in \mathcal{D}\ (P\ \llbracket A \rrbracket_{\checkmark}\ Q \downarrow n) \rangle$

by (*metis *(5) D-restriction-process_{ptick}I nless-le*)

with $*(1)$ *assms(1, 2)* **show** $\langle u\ @\ v \in \mathcal{D}\ (P\ \llbracket A \rrbracket_{\checkmark}\ Q \downarrow n) \rangle$

by (metis is-processT7 tickFree-append-iff tickFree-imp-front-tickFree)
 next
 fix $t-P' t-P''$
 assume $*$: $\langle t-P = t-P' @ t-P'' \rangle \langle t-P' \in \mathcal{T} P \rangle \langle \text{length } t-P' = n \rangle$
 $\langle tF t-P' \rangle \langle ftF t-P'' \rangle \langle t-Q \in \mathcal{T} Q \rangle \langle \text{length } t-Q \leq n \rangle$
 from $\langle t-P = t-P' @ t-P'' \rangle$ have $\langle t-P' \leq t-P \rangle$ by simp
 from *setinterleaves_{ptick}-le-prefixL*[OF *assms*(5) *this*]
 obtain $t-Q' t-Q'' u' u''$
 where $**$: $\langle u = u' @ u'' \rangle \langle t-Q = t-Q' @ t-Q'' \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((t-P', t-Q'), A) \rangle$
 by (meson *Prefix-Order.prefixE*)
 from *assms*(1) $\langle u = u' @ u'' \rangle$ have $\langle tF u' \rangle$ by auto
 moreover from $*(2,6) ** (2,3)$ have $\langle u' \in \mathcal{T} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
 by (simp add: *T-Sync_{ptick}*) (metis is-processT3-TR-append)
 moreover have $\langle \text{length } t-P' \leq \text{length } u' \rangle$
 using $*(3)$ *setinterleaves_{ptick}-imp-lengthLR-le* by blast
 ultimately have $\langle u' \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 by (metis $*(3)$ *D-restriction-process_{ptick}I nless-le*)
 with $*(1)$ *assms*(1, 2) show $\langle u @ v \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 by (metis is-processT7 tickFree-append-iff tickFree-imp-front-tickFree)
 next
 fix $t-P' t-P'' t-Q' t-Q''$
 assume $\$$: $\langle t-P = t-P' @ t-P'' \rangle \langle t-P' \in \mathcal{T} P \rangle \langle \text{length } t-P' = n \rangle$
 $\langle tF t-P' \rangle \langle ftF t-P'' \rangle \langle t-Q = t-Q' @ t-Q'' \rangle \langle t-Q' \in \mathcal{T} Q \rangle$
 $\langle \text{length } t-Q' = n \rangle \langle tF t-Q' \rangle \langle ftF t-Q'' \rangle$
 from $\$(1, 6)$ have $\langle t-P' \leq t-P \rangle \langle t-Q' \leq t-Q \rangle$ by simp-all
 from *setinterleaves_{ptick}-le-prefixLR*[OF *assms*(5) *this*]
 show $\langle u @ v \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 proof (elim disjE conjE exE)
 fix $u' t-Q'''$ assume $\$ \$$: $\langle u' \leq u \rangle \langle t-Q''' \leq t-Q' \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((t-P', t-Q'''), A) \rangle$
 from $\$(7) \$\$(2)$ is-processT3-TR have $\langle t-Q''' \in \mathcal{T} Q \rangle$ by blast
 with $\$ \$\$(3) \langle t-P' \in \mathcal{T} P \rangle$ have $\langle u' \in \mathcal{T} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
 by (auto simp add: *T-Sync_{ptick}*)
 moreover have $\langle n \leq \text{length } u' \rangle$
 using $\$(3) \$\$(3)$ *setinterleaves_{ptick}-imp-lengthLR-le* by blast
 ultimately have $\langle u' \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 by (metis $\$(1)$ *D-restriction-process_{ptick}I Prefix-Order.prefixE*
assms(1) *nless-le tickFree-append-iff*)
 thus $\langle u @ v \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 by (metis $\$(1)$ *Prefix-Order.prefixE assms*(1,2) is-processT7
tickFree-append-iff tickFree-imp-front-tickFree)
 next
 fix $u' t-P'''$ assume $\$ \$$: $\langle u' \leq u \rangle \langle t-P''' \leq t-P' \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark \text{tick-join}} ((t-P''', t-Q'), A) \rangle$
 from $\$(2) \$\$(2)$ is-processT3-TR have $\langle t-P''' \in \mathcal{T} P \rangle$ by blast
 with $\$ \$\$(3) \langle t-Q' \in \mathcal{T} Q \rangle$ have $\langle u' \in \mathcal{T} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
 by (auto simp add: *T-Sync_{ptick}*)
 moreover have $\langle n \leq \text{length } u' \rangle$

using $\$(8) \(3) *setinterleaves_{ptick}-imp-lengthLR-le* by *blast*
 ultimately have $\langle u' \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 by (*metis* $\$(1)$ *D-restriction-process_{ptick}I Prefix-Order.prefixE*
assms(1) nless-le tickFree-append-iff)
 thus $\langle u @ v \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$
 by (*metis* $\$(1)$ *Prefix-Order.prefixE assms(1,2) is-processT7*
tickFree-append-iff tickFree-imp-front-tickFree)
 qed
 qed

lemma (in *Sync_{ptick}-locale*) *restriction-process_{ptick}-Sync_{ptick}-FD* :
 $\langle P \llbracket A \rrbracket_{\checkmark} Q \downarrow n \sqsubseteq_{FD} (P \downarrow n) \llbracket A \rrbracket_{\checkmark} (Q \downarrow n) \rangle$ (is $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$)
proof (*unfold refine-defs, safe*)
 show $\langle t \in \mathcal{D} ?rhs \implies t \in \mathcal{D} ?lhs \rangle$ for t
 by (*unfold D-Sync_{ptick}, safe*)
 (*solves* $\langle \text{simp add: restriction-process_{ptick}-Sync_{ptick}-FD-div-oneside} \rangle$,
metis Sync_{ptick}-locale-sym.restriction-process_{ptick}-Sync_{ptick}-FD-div-oneside
Sync_{ptick}-sym setinterleaves_{ptick}-sym)
 thus $\langle (t, X) \in \mathcal{F} ((P \downarrow n) \llbracket A \rrbracket_{\checkmark} (Q \downarrow n)) \implies (t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$ for
 $t X$
 by (*meson is-processT8 le-approx2 mono-Sync_{ptick} restriction-process_{ptick}-approx-self*)
 qed

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}*
 :
 $\langle P \llbracket A \rrbracket_{\checkmark} Q \downarrow n = (P \downarrow n) \llbracket A \rrbracket_{\checkmark} (Q \downarrow n) \rangle$ (is $\langle ?lhs = ?rhs \rangle$)
 if $\langle \alpha(P) \subseteq A \vee \alpha(Q) \subseteq A \rangle$
proof (*rule FD-antisym*)
 show $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$ by (*fact restriction-process_{ptick}-Sync_{ptick}-FD*)
 next
 have *div* : $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \implies t \in \mathcal{D} ?rhs \rangle$ for t
 by (*auto simp add: D-Sync_{ptick} restriction-process_{ptick}-projs*)
 { fix $t u v$ assume $\langle t = u @ v \rangle \langle u \in \mathcal{T} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle \langle \text{length } u = n \rangle \langle tF u \rangle \langle ftF v \rangle$
 from *this(2)* consider $\langle u \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
 | $t-P \ t-Q$ where $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark} \text{tick-join} ((t-P, t-Q), A) \rangle$
 unfolding *Sync_{ptick}-projs* by *blast*
 hence $\langle t \in \mathcal{D} ?rhs \rangle$
proof *cases*
 show $\langle u \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \implies t \in \mathcal{D} ?rhs \rangle$
 by (*simp add:* $\langle ftF v \rangle \langle t = u @ v \rangle \langle tF u \rangle$ *div is-processT7*)
 next
 fix $t-P \ t-Q$ assume $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$

```

    and setinter :  $\langle u \text{ setinterleaves}_{\text{tick-join}} ((t-P, t-Q), A) \rangle$ 
  consider  $\langle t-P \in \mathcal{D} P \vee t-Q \in \mathcal{D} Q \rangle \mid \langle t-P \notin \mathcal{D} P \rangle \langle t-Q \notin \mathcal{D} Q \rangle$  by blast
  thus  $\langle t \in \mathcal{D} ?rhs \rangle$ 
  proof cases
    assume  $\langle t-P \in \mathcal{D} P \vee t-Q \in \mathcal{D} Q \rangle$ 
    with  $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$  setinter  $\langle ftF v \rangle \langle t = u @ v \rangle \langle tF u \rangle$ 
    have  $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\text{tick-join}} Q) \rangle$  by (auto simp add: D-Syncptick)
    thus  $\langle t \in \mathcal{D} ?rhs \rangle$  by (fact div)
  next
    assume  $\langle t-P \notin \mathcal{D} P \rangle \langle t-Q \notin \mathcal{D} Q \rangle$ 
    with  $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$   $\langle \alpha(P) \subseteq A \vee \alpha(Q) \subseteq A \rangle$ 
    have  $\langle \{a. \text{ev } a \in \text{set } t-P\} \subseteq A \vee \{a. \text{ev } a \in \text{set } t-Q\} \subseteq A \rangle$ 
    by (auto dest: subsetD intro; strict-events-of-memI)
    with setinterleavesptick-subsetL[OF  $\langle tF u \rangle$  - setinter]
    setinterleavesptick-subsetR[OF  $\langle tF u \rangle$  - setinter]
    have  $\langle u = \text{map ev } (\text{map of-ev } t-P) \vee u = \text{map ev } (\text{map of-ev } t-Q) \rangle$  by blast
    with  $\langle \text{length } u = n \rangle$  have  $\langle \text{length } t-P = n \vee \text{length } t-Q = n \rangle$  by auto
    moreover from  $\langle tF u \rangle$  tickFree-setinterleavesptick-iff[OF setinter]
    have  $\langle tF t-P \rangle \langle tF t-Q \rangle$  by simp-all
    ultimately have  $\langle t-P \in \mathcal{D} (P \downarrow n) \vee t-Q \in \mathcal{D} (Q \downarrow n) \rangle$ 
    using  $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$  by (metis D-restriction-processptickI)
    moreover from  $\langle t-P \in \mathcal{T} P \rangle \langle t-Q \in \mathcal{T} Q \rangle$ 
    have  $\langle t-P \in \mathcal{T} (P \downarrow n) \rangle \langle t-Q \in \mathcal{T} (Q \downarrow n) \rangle$ 
    by (simp-all add: T-restriction-processptickI)
    ultimately show  $\langle t \in \mathcal{D} ?rhs \rangle$ 
    using  $\langle ftF v \rangle \langle t = u @ v \rangle \langle tF u \rangle$  setinter by (auto simp add: D-Syncptick)
  qed
  qed
} note * = this

show  $\langle ?rhs \sqsubseteq_{FD} ?lhs \rangle$ 
proof (unfold refine-defs, safe)
  show  $\langle t \in \mathcal{D} ?lhs \implies t \in \mathcal{D} ?rhs \rangle$  for t
  proof (elim D-restriction-processptickE)
    show  $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\text{tick-join}} Q) \implies t \in \mathcal{D} ?rhs \rangle$  by (fact div)
  next
    show  $\langle \llbracket t = u @ v; u \in \mathcal{T} (P \llbracket A \rrbracket_{\text{tick-join}} Q); \text{length } u = n; tF u; ftF v \rrbracket \implies t \in \mathcal{D} ?rhs \rangle$  for u v by (fact *)
  qed
next
show  $\langle (t, X) \in \mathcal{F} ?lhs \implies (t, X) \in \mathcal{F} ?rhs \rangle$  for t X
proof (elim F-restriction-processptickE)
  assume  $\langle (t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\text{tick-join}} Q) \rangle$ 
  then consider  $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\text{tick-join}} Q) \rangle$ 
  | (fail) t-P t-Q X-P X-Q where  $\langle (t-P, X-P) \in \mathcal{F} P \rangle \langle (t-Q, X-Q) \in \mathcal{F} Q \rangle$ 
  |  $\langle t \text{ setinterleaves}_{\text{tick-join}} ((t-P, t-Q), A) \rangle$ 
  |  $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{ tick-join } X-P A X-Q \rangle$ 
  unfolding Syncptick-projs by blast
  thus  $\langle (t, X) \in \mathcal{F} ?rhs \rangle$ 

```

```

proof cases
  from div D-F show  $\langle t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q) \implies (t, X) \in \mathcal{F} \text{ ?}rhs \rangle$  by blast
next
  case fail
  thus  $\langle (t, X) \in \mathcal{F} \text{ ?}rhs \rangle$ 
    by (auto simp add: F-Syncptick F-restriction-processptick)
  qed
next
  show  $\langle [t = u @ v; u \in \mathcal{T} (P \llbracket A \rrbracket_{\checkmark} Q); \text{length } u = n; tF \text{ } u; ftF \text{ } v] \implies (t, X) \in \mathcal{F} \text{ ?}rhs \rangle$  for  $u \text{ } v$  by (simp add: * is-processT8)
  qed
qed
qed

```

corollary *restriction-process_{ptick}-MultiSync_{ptick}-FD :*
 $\langle \llbracket A \rrbracket_{\checkmark} l \in @ L. P \downarrow n \sqsubseteq_{FD} \llbracket A \rrbracket_{\checkmark} l \in @ L. (P \downarrow n) \rangle$
proof (*induct L rule: induct-list012*)
show $\langle \llbracket A \rrbracket_{\checkmark} l \in @ []. P \downarrow n \sqsubseteq_{FD} \llbracket A \rrbracket_{\checkmark} l \in @ []. (P \downarrow n) \rangle$ **by** *simp*
next
show $\langle \llbracket A \rrbracket_{\checkmark} l \in @ [l1]. P \downarrow n \sqsubseteq_{FD} \llbracket A \rrbracket_{\checkmark} l \in @ [l1]. (P \downarrow n) \rangle$ **for** $l1$
by (*simp add: restriction-process_{ptick}-Renaming*)
next
fix $l1 \text{ } l2 \text{ } L$
assume *hyp* : $\langle \llbracket A \rrbracket_{\checkmark} l \in @ (l2 \# L). P \downarrow n \sqsubseteq_{FD} \llbracket A \rrbracket_{\checkmark} l \in @ (l2 \# L). (P \downarrow n) \rangle$
show $\langle \llbracket A \rrbracket_{\checkmark} l \in @ (l1 \# l2 \# L). P \downarrow n \sqsubseteq_{FD} \llbracket A \rrbracket_{\checkmark} l \in @ (l1 \# l2 \# L). (P \downarrow n) \rangle$
by *simp*
 (*fact trans-FD[OF Sync_{Rlist}.restriction-process_{ptick}-Sync_{ptick}-FD Sync_{Rlist}.mono-Sync_{ptick}-FD[OF idem-FD hyp]]*)
qed

The generalization of the lemma $\alpha(P) \subseteq A \vee \alpha(Q) \subseteq A \implies P \llbracket A \rrbracket_{\checkmark} Q \downarrow n = (P \downarrow n) \llbracket A \rrbracket_{\checkmark} (Q \downarrow n)$ 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} :*
 $\langle \llbracket A \rrbracket_{\checkmark} l \in @ L. P \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \llbracket A \rrbracket_{\checkmark} l \in @ L. (P \downarrow n)) \rangle$
 — *if* $n = 0$ *then* $\perp$ *else* - is necessary because we can have $L = []$.
if $\langle \bigwedge l. l \in \text{set } (tl \text{ } L) \implies \alpha(P \downarrow l) \subseteq A \rangle$
proof (*split if-split, intro conjI impI*)
show $\langle n = 0 \implies \llbracket A \rrbracket_{\checkmark} l \in @ L. P \downarrow n = \perp \rangle$ **by** *simp*
next
from *that* **show** $\langle \llbracket A \rrbracket_{\checkmark} l \in @ L. P \downarrow n = \llbracket A \rrbracket_{\checkmark} l \in @ L. (P \downarrow n) \rangle$ **if** $\langle n \neq 0 \rangle$
proof (*induct L rule: induct-list012*)
case 1 **show** *?case* **by** (*simp add: $\langle n \neq 0 \rangle$*)
next

case (2 l1) show ?case by (simp add: restriction-process_{ptick}-Renaming)
 next
 case (3 l1 l2 L)
 from 3.premis have * : $\langle \alpha(\text{MultiSync}_{\text{ptick}} A (l2 \# L) P) \subseteq A \rangle$
 by (intro subset-trans[OF strict-events-of-MultiSync_{ptick}-subset]) auto
 have $\langle \llbracket A \rrbracket_{\checkmark} l \in @ (l1 \# l2 \# L). P \downarrow n =$
 $P \downarrow l1 \llbracket A \rrbracket_{\checkmark} Rlist \llbracket A \rrbracket_{\checkmark} l \in @ (l2 \# L). P \downarrow n \rangle$ by simp
 also have $\langle \dots = (P \downarrow l1 \downarrow n) \llbracket A \rrbracket_{\checkmark} Rlist (\llbracket A \rrbracket_{\checkmark} l \in @ (l2 \# L). P \downarrow n) \rangle$
 by (simp add: Sync_{Rlist}.strict-events-of-subset-restriction-process_{ptick}-Sync_{ptick}
 *)
 also have $\langle \dots = (P \downarrow l1 \downarrow n) \llbracket A \rrbracket_{\checkmark} Rlist \llbracket A \rrbracket_{\checkmark} l \in @ (l2 \# L). (P \downarrow n) \rangle$
 using 3.hyps(2) 3.premis by auto
 also have $\langle \dots = \llbracket A \rrbracket_{\checkmark} l \in @ (l1 \# l2 \# L). (P \downarrow n) \rangle$ by simp
 finally show ?case .
 qed
 qed

corollary (in Sync_{ptick}-locale) restriction-process_{ptick}-Par_{ptick} :
 $\langle P \parallel_{\checkmark} Q \downarrow n = (P \downarrow n) \parallel_{\checkmark} (Q \downarrow n) \rangle$
 by (simp add: strict-events-of-subset-restriction-process_{ptick}-Sync_{ptick})

corollary restriction-process_{ptick}-MultiPar_{ptick} :
 $\langle \parallel_{\checkmark} l \in @ L. P \downarrow l \downarrow n = (\text{if } n = 0 \text{ then } \perp \text{ else } \parallel_{\checkmark} l \in @ L. (P \downarrow l \downarrow n)) \rangle$
 by (simp add: strict-events-of-subset-restriction-process_{ptick}-MultiSync_{ptick})

15.1.2 Non Destructiveness

lemma (in Sync_{ptick}-locale) Sync_{ptick}-non-destructive :
 $\langle \text{non-destructive } (\lambda(P, Q). P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
proof (rule order-non-destructiveI, clarify)
 fix P P' :: $\langle ('a, 'r) \text{ process}_{\text{ptick}} \rangle$ and Q Q' :: $\langle ('a, 's) \text{ process}_{\text{ptick}} \rangle$ and n
 assume $\langle (P, Q) \downarrow n = (P', Q') \downarrow n \rangle$
 hence $\langle P \downarrow n = P' \downarrow n \rangle \langle Q \downarrow n = Q' \downarrow n \rangle$
 by (simp-all add: restriction-prod-def)
 show $\langle P \llbracket A \rrbracket_{\checkmark} Q \downarrow n \sqsubseteq_{FD} P' \llbracket A \rrbracket_{\checkmark} Q' \downarrow n \rangle$
proof (rule leFD-restriction-process_{ptick}I)
 show $\langle t \in \mathcal{D} (P' \llbracket A \rrbracket_{\checkmark} Q') \implies t \in \mathcal{D} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$ for t
 by (metis (no-types, lifting) $\langle P \downarrow n = P' \downarrow n \rangle \langle Q \downarrow n = Q' \downarrow n \rangle$ in-mono
 le-ref1 mono-Sync_{ptick}-FD
 restriction-process_{ptick}-FD-self restriction-process_{ptick}-Sync_{ptick}-FD)
 next
 show $\langle (t, X) \in \mathcal{F} (P' \llbracket A \rrbracket_{\checkmark} Q') \implies (t, X) \in \mathcal{F} (P \llbracket A \rrbracket_{\checkmark} Q \downarrow n) \rangle$ for t X
 by (metis (no-types, lifting) $\langle P \downarrow n = P' \downarrow n \rangle \langle Q \downarrow n = Q' \downarrow n \rangle$ le-ref2
 mono-Sync_{ptick}-FD
 restriction-process_{ptick}-FD-self restriction-process_{ptick}-Sync_{ptick}-FD
 subsetD)
 qed
 qed

15.1.3 Setup

lemma (in $\text{Sync}_{ptick}\text{-locale}$) $\text{Sync}_{ptick}\text{-restriction-shift-process}_{ptick}$
 $[\text{restriction-shift-process}_{ptick}\text{-simpset}, \text{simp}] :$
 $\langle \text{non-destructive } f \implies \text{non-destructive } g \implies \text{non-destructive } (\lambda x. f\ x \llbracket S \rrbracket_{\checkmark} g\ x) \rangle$
 $\langle \text{constructive } f \implies \text{constructive } g \implies \text{constructive } (\lambda x. f\ x \llbracket S \rrbracket_{\checkmark} g\ x) \rangle$
by (fact $\text{non-destructive-comp-non-destructive}$
 $[\text{OF } \text{Sync}_{ptick}\text{-non-destructive non-destructive-prod-codomain}, \text{simplified}]$)
(fact $\text{non-destructive-comp-constructive}$
 $[\text{OF } \text{Sync}_{ptick}\text{-non-destructive constructive-prod-codomain}, \text{simplified}]$)

lemma $\text{MultiSync}_{ptick}\text{-restriction-shift-process}_{ptick}$
 $[\text{restriction-shift-process}_{ptick}\text{-simpset}, \text{simp}] :$
 $\langle (\bigwedge l. l \in \text{set } L \implies \text{non-destructive } (f\ l)) \implies \text{non-destructive } (\lambda x. \llbracket S \rrbracket_{\checkmark} l \in @ L. f\ l\ x) \rangle$
 $\langle (\bigwedge l. l \in \text{set } L \implies \text{constructive } (f\ l)) \implies \text{constructive } (\lambda x. \llbracket S \rrbracket_{\checkmark} l \in @ L. f\ l\ x) \rangle$
by (induct L rule: induct-list012 ; simp) +

corollary $\text{MultiSync}_{ptick}\text{-non-destructive} : \langle \text{non-destructive } (\lambda P. \llbracket S \rrbracket_{\checkmark} l \in @ L. P\ l) \rangle$
by (rule $\text{MultiSync}_{ptick}\text{-restriction-shift-process}_{ptick}(1)[\text{of } L \langle \lambda m\ x. x\ m \rangle]$) simp

Chapter 16

Other Laws

declare $[[metis-instantiate]]$

16.1 Laws of Renaming

16.1.1 Renaming and Sequential Composition

lemma *FD-Renaming-Seq_{ptick}* :

$$\langle \text{Renaming } P \text{ } f \text{ } g ; \checkmark (\lambda g-r. \sqcap r \in \{r \in \checkmark \mathbf{s}(P). g-r = g \text{ } r\}. \text{Renaming } (Q \text{ } r) \text{ } f \text{ } g') \rangle$$

$$\sqsubseteq_{FD} \text{Renaming } (P ; \checkmark Q) \text{ } f \text{ } g' \rangle \text{ (is } \langle ?lhs \sqsubseteq_{FD} ?rhs \rangle)$$

proof (*rule failure-divergence-refine-optimizedI*)

fix s **assume** $\langle s \in \mathcal{D} \text{ } ?rhs \rangle$

then obtain $s1 \text{ } s2$ **where** $*$: $\langle s = \text{map } (\text{map-event}_{ptick} \text{ } f \text{ } g') \text{ } s1 \text{ } @ \text{ } s2 \rangle \langle tF \text{ } s1 \rangle$

$$\langle tF \text{ } s2 \rangle \langle s1 \in \mathcal{D} (P ; \checkmark Q) \rangle$$

unfolding *D-Renaming* **by** *blast*

from $*(4)$ **consider** $(D-P) \text{ } t1 \text{ } t2$ **where** $\langle s1 = \text{map } (ev \circ of-ev) \text{ } t1 \text{ } @ \text{ } t2 \rangle \langle t1 \in \mathcal{D} \text{ } P \rangle \langle tF \text{ } t1 \rangle \langle tF \text{ } t2 \rangle$

$$| (D-Q) \text{ } t1 \text{ } r \text{ } t2 \text{ } \text{where } \langle s1 = \text{map } (ev \circ of-ev) \text{ } t1 \text{ } @ \text{ } t2 \rangle \langle t1 \text{ } @ \text{ } [\checkmark(r)] \in \mathcal{T} \text{ } P \rangle$$

$$\langle t1 \notin \mathcal{D} \text{ } P \rangle \langle tF \text{ } t1 \rangle \langle t2 \in \mathcal{D} (Q \text{ } r) \rangle$$

by (*simp add: Seq_{ptick}-projs*) (*metis D-imp-front-tickFree*)

thus $\langle s \in \mathcal{D} \text{ } ?lhs \rangle$

proof *cases*

case *D-P*

from $D-P(2, 3)$ **have** $\langle \text{map } (\text{map-event}_{ptick} \text{ } f \text{ } g) \text{ } t1 \in \mathcal{D} (\text{Renaming } P \text{ } f \text{ } g) \rangle$

by (*auto simp add: D-Renaming intro: front-tickFree-Nil*)

hence $\langle \text{map } (ev \circ of-ev) (\text{map } (\text{map-event}_{ptick} \text{ } f \text{ } g) \text{ } t1) \in \mathcal{D} \text{ } ?lhs \rangle$

unfolding *Seq_{ptick}-projs*

by (*metis (mono-tags, lifting) front-tickFree-Nil D-P(3)*)

$$\text{map-event}_{ptick}\text{-tickFree append.right-neutral mem-Collect-eq Un-iff})$$

also have $\langle \text{map } (ev \circ of-ev) (\text{map } (\text{map-event}_{ptick} \text{ } f \text{ } g) \text{ } t1) =$

$$\text{map } (\text{map-event}_{ptick} \text{ } f \text{ } g') (\text{map } (ev \circ of-ev) \text{ } t1) \rangle$$

by (*simp add: tF t1 tickFree-map-map-event_{ptick}-is*)

finally show $\langle s \in \mathcal{D} \text{ } ?lhs \rangle$

by (auto simp add: $\ast(1)$ $D\text{-}P(1)$ intro!: is-processT7)
 (metis list.map-comp map-event_{ptick}-tickFree tickFree-map-ev-comp,
 use $\ast(2, 3)$ $D\text{-}P(1)$ front-tickFree-append map-event_{ptick}-tickFree tick-
 Free-append-iff in blast)
 next
 case $D\text{-}Q$
 from $\ast(2)$ $D\text{-}Q(1, 5)$ have $\langle \text{map } (\text{map-event}_{\text{ptick}} f g') t2 \in \mathcal{D} \text{ (Renaming } (Q \text{ } r) f g') \rangle$
 by (auto simp add: $D\text{-}Renaming$ intro: front-tickFree-Nil)
 hence $\langle \text{map } (\text{map-event}_{\text{ptick}} f g') t2 \in \mathcal{D} \text{ } (\sqcap r' \in \{r' \in \check{\mathbf{s}}(P). g \text{ } r = g \text{ } r'\}.$
 $Renaming (Q \text{ } r') f g') \rangle$
 by (simp add: $D\text{-}GlobalNdet$)
 (metis $D\text{-}Q(2, 3)$ is-processT9 strict-ticks-of-memI)
 moreover from $D\text{-}Q(2)$ have $\langle \text{map } (\text{map-event}_{\text{ptick}} f g) t1 @ [\check{\mathbf{s}}(g \text{ } r)] \in \mathcal{T}$
 (Renaming $P \text{ } f g) \rangle$
 by (auto simp add: $T\text{-}Renaming$)
 moreover have $\langle tF (\text{map } (\text{map-event}_{\text{ptick}} f g) t1) \rangle$
 by (simp add: $D\text{-}Q(4)$ map-event_{ptick}-tickFree)
 ultimately have $\langle \text{map } (ev \circ of\text{-}ev) (\text{map } (\text{map-event}_{\text{ptick}} f g) t1) @$
 $\text{map } (\text{map-event}_{\text{ptick}} f g') t2 \in \mathcal{D} \text{ } ?lhs \rangle$
 unfolding Seq_{ptick}-projs by blast
 with $\ast(2, 3)$ have $\langle \text{map } (ev \circ of\text{-}ev) (\text{map } (\text{map-event}_{\text{ptick}} f g) t1) @$
 $\text{map } (\text{map-event}_{\text{ptick}} f g') t2 @ s2 \in \mathcal{D} \text{ } ?lhs \rangle$
 by (auto simp add: $D\text{-}Q(1)$ comp-assoc map-event_{ptick}-tickFree
 intro!: is-processT7[of $\langle - @ - \rangle$, simplified])
 also from $D\text{-}Q(4)$ have $\langle \text{map } (ev \circ of\text{-}ev) (\text{map } (\text{map-event}_{\text{ptick}} f g) t1) @$
 $\text{map } (\text{map-event}_{\text{ptick}} f g') t2 @ s2 = s \rangle$
 by (simp add: $\ast(1)$ $D\text{-}Q(1)$)
 (metis event_{ptick}.map-sel(1) in-set-conv-decomp tickFree-Cons-iff tick-
 Free-append-iff)
 finally show $\langle s \in \mathcal{D} \text{ } ?lhs \rangle$.
 qed
 next
 assume subset-div : $\langle \mathcal{D} \text{ } ?rhs \subseteq \mathcal{D} \text{ } ?lhs \rangle$
 fix $s \text{ } X$ assume $\langle (s, X) \in \mathcal{F} \text{ } ?rhs \rangle$
 then consider $\langle s \in \mathcal{D} \text{ } ?rhs \rangle$
 | (fail) $s1$ where $\langle s = \text{map } (\text{map-event}_{\text{ptick}} f g') s1 \rangle$
 $\langle (s1, \text{map-event}_{\text{ptick}} f g' - 'X) \in \mathcal{F} (P ; \check{\mathbf{s}} Q) \rangle \langle s1 \notin \mathcal{D} (P ; \check{\mathbf{s}} Q) \rangle$
 by (simp add: Renaming-projs)
 (metis (no-types, opaque-lifting) front-tickFree-Nil front-tickFree-iff-tickFree-butlast
 front-tickFree-Cons-iff[of $\langle \text{last } s \rangle \langle [] \rangle$] map-butlast[of $\langle \text{map-event}_{\text{ptick}} f g' \rangle$]
 map-is-Nil-conv[of $\langle \text{map-event}_{\text{ptick}} f g' \rangle \langle [] \rangle$] map-is-Nil-conv[of $\langle \text{map-event}_{\text{ptick}}$
 $f g' \rangle$]
 append-self-conv[of $\langle \text{map } (\text{map-event}_{\text{ptick}} f g') - \rangle \langle [] \rangle$] F-imp-front-tickFree
 snoc-eq-iff-butlast[of $\langle \text{butlast } s \rangle \langle \text{last } s \rangle s$]
 div-butlast-when-non-tickFree-iff non-tickFree-imp-not-Nil)
 thus $\langle (s, X) \in \mathcal{F} \text{ } ?lhs \rangle$
 proof cases
 from subset-div $D\text{-}F$ show $\langle s \in \mathcal{D} \text{ } ?rhs \implies (s, X) \in \mathcal{F} \text{ } ?lhs \rangle$ by blast


```

next
  case fail
  from fail(2, 3)
  consider (F-P) t1 where  $\langle s1 = \text{map } (ev \circ \text{of-ev}) \ t1 \rangle \langle (t1, \text{ref-Seq}_{ptick} \text{ (map-event}_{ptick} \ f \ g' - 'X)) \in \mathcal{F} \ P \rangle \langle tF \ t1 \rangle$ 
    | (F-Q) t1 r t2 where  $\langle s1 = \text{map } (ev \circ \text{of-ev}) \ t1 \ @ \ t2 \rangle \langle t1 \ @ \ [\checkmark(r)] \in \mathcal{T} \ P \rangle$ 
     $\langle tF \ t1 \rangle \langle t1 \notin \mathcal{D} \ P \rangle \langle (t2, \text{map-event}_{ptick} \ f \ g' - 'X) \in \mathcal{F} \ (Q \ r) \rangle$ 
    by (simp add: Seqptick-projs) (metis F-imp-front-tickFree)
    thus  $\langle (s, X) \in \mathcal{F} \ ?lhs \rangle$ 
  proof cases
    case F-P
    have  $\langle \text{map-event}_{ptick} \ f \ g - '(\text{ref-Seq}_{ptick} \ X) = \text{ref-Seq}_{ptick} \ (\text{map-event}_{ptick} \ f \ g' - 'X) \rangle$  for X
    proof (rule set-eqI)
      show  $\langle e \in \text{map-event}_{ptick} \ f \ g - '(\text{ref-Seq}_{ptick} \ X) \longleftrightarrow$ 
         $e \in \text{ref-Seq}_{ptick} \ (\text{map-event}_{ptick} \ f \ g' - 'X) \rangle$  for e
      by (cases e, auto simp add: ref-Seqptick-def image-iff)
        (metis Int-iff eventptick.sel(1) eventptick.sims(9) rangeI vimage-eq,
          metis IntI UNIV-I eventptick.sel(1) image-eqI)
    qed
    with F-P(2) have  $\langle (\text{map } (\text{map-event}_{ptick} \ f \ g) \ t1, \text{ref-Seq}_{ptick} \ X) \in \mathcal{F}$ 
      (Renaming P f g)  $\rangle$ 
    by (auto simp add: F-Renaming)
    with F-P(3) have  $\langle (\text{map } (ev \circ \text{of-ev}) \ (\text{map } (\text{map-event}_{ptick} \ f \ g) \ t1), X) \in$ 
       $\mathcal{F} \ ?lhs \rangle$ 
    by (fastforce simp add: Seqptick-projs map-eventptick-tickFree)
    also have  $\langle \text{map } (ev \circ \text{of-ev}) \ (\text{map } (\text{map-event}_{ptick} \ f \ g) \ t1) = s \rangle$ 
    by (simp add: fail(1) F-P(1))
        (metis F-P(3) eventptick.map-sel(1) in-set-conv-decomp tickFree-Cons-iff
          tickFree-append-iff)
    finally show  $\langle (s, X) \in \mathcal{F} \ ?lhs \rangle$  .
  next
  case F-Q
  with F-Q(4) have  $\langle (\text{map } (\text{map-event}_{ptick} \ f \ g') \ t2, X) \in \mathcal{F} \ (\text{Renaming } (Q \ r)$ 
     $f \ g') \rangle$ 
  by (auto simp add: F-Renaming)
  hence  $\langle (\text{map } (\text{map-event}_{ptick} \ f \ g') \ t2, X) \in$ 
     $\mathcal{F} \ (\sqcap r' \in \{r' \in \check{\mathbf{s}}(P). \ g \ r = g \ r'\}. \text{Renaming } (Q \ r') \ f \ g') \rangle$ 
  by (simp add: F-GlobalNdet)
    (metis F-Q(2, 4) is-processT9 strict-ticks-of-memI)
  moreover from F-Q(2) have  $\langle \text{map } (\text{map-event}_{ptick} \ f \ g) \ t1 \ @ \ [\checkmark(g \ r)] \in \mathcal{T}$ 
    (Renaming P f g)  $\rangle$ 
  by (auto simp add: T-Renaming)
  moreover have  $\langle tF \ (\text{map } (\text{map-event}_{ptick} \ f \ g) \ t1) \rangle$ 
  by (simp add: F-Q(3) map-eventptick-tickFree)
  ultimately have  $\langle (\text{map } (ev \circ \text{of-ev}) \ (\text{map } (\text{map-event}_{ptick} \ f \ g) \ t1) \ @$ 
     $\text{map } (\text{map-event}_{ptick} \ f \ g') \ t2, X) \in \mathcal{F} \ ?lhs \rangle$ 
  unfolding Seqptick-projs by fast
  also have  $\langle \text{map } (ev \circ \text{of-ev}) \ (\text{map } (\text{map-event}_{ptick} \ f \ g) \ t1) \ @ \ \text{map } (\text{map-event}_{ptick}$ 

```

$f g'$ $t2 = s$
by (*simp add: fail(1) F-Q(1)*)
(metis F-Q(3) event_{ptick}.map-sel(1) in-set-conv-decomp
tickFree-Cons-iff tickFree-append-iff)
finally show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$.
qed
qed
qed

lemma *inj-on-Renaming-Seq_{ptick}* :
 $\langle \text{Renaming } (P ; \checkmark Q) f g' =$
 $\text{Renaming } P f g ; \checkmark (\lambda g-r. \text{Renaming } (Q (THE r. r \in \checkmark s(P) \wedge g-r = g r)) f g') \rangle$
 $(\text{is } \langle ?lhs = ?rhs \rangle) \text{ if } \langle \text{inj-on } g \checkmark s(P) \rangle$
 — This assumption is necessary, otherwise we cannot know which tick triggered
 Q .
proof (*rule FD-antisym*)
show $\langle ?lhs \sqsubseteq_{FD} ?rhs \rangle$
proof (*rule failure-divergence-refine-optimizedI*)
fix s **assume** $\langle s \in \mathcal{D} \text{ ?rhs} \rangle$
then consider $(D-P) s1 s2$ **where** $\langle s = \text{map } (ev \circ of-ev) s1 @ s2 \rangle \langle s1 \in \mathcal{D}$
 $(\text{Renaming } P f g) \rangle \langle tF s1 \rangle \langle ftF s2 \rangle$
 $| (D-Q) s1 g-r s2$ **where** $\langle s = \text{map } (ev \circ of-ev) s1 @ s2 \rangle \langle s1 @ [\checkmark(g-r)] \in \mathcal{T}$
 $(\text{Renaming } P f g) \rangle$
 $\langle s1 \notin \mathcal{D} (\text{Renaming } P f g) \rangle \langle tF s1 \rangle \langle s2 \in \mathcal{D} (\text{Renaming } (Q (THE r. r \in$
 $\checkmark s(P) \wedge g-r = g r)) f g') \rangle$
by (*simp add: Seq_{ptick}-projs*) (*use D-imp-front-tickFree in blast*)
thus $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$
proof cases
case $D-P$
from $D-P(2)$ **obtain** $t1 t2$
where $*$: $\langle s1 = \text{map } (\text{map-event}_{ptick} f g) t1 @ t2 \rangle \langle tF t1 \rangle \langle ftF t2 \rangle \langle t1 \in$
 $\mathcal{D} P \rangle$
unfolding $D\text{-Renaming}$ **by** *blast*
from $*(2, 4)$ **have** $\langle \text{map } (ev \circ of-ev) t1 \in \mathcal{D} (P ; \checkmark Q) \rangle$
by (*auto simp add: Seq_{ptick}-projs intro: front-tickFree-Nil*)
hence $\langle \text{map } (\text{map-event}_{ptick} f g') (\text{map } (ev \circ of-ev) t1) \in \mathcal{D} \text{ ?lhs} \rangle$
unfolding $D\text{-Renaming mem-Collect-eq}$
by (*metis (mono-tags, lifting) front-tickFree-Nil tickFree-map-ev-comp ap-*
pend.right-neutral)
also have $\langle \text{map } (\text{map-event}_{ptick} f g') (\text{map } (ev \circ of-ev) t1) =$
 $\text{map } (ev \circ of-ev) (\text{map } (\text{map-event}_{ptick} f g) t1) \rangle$
by *simp (metis *(2) event_{ptick}.map-sel(1) in-set-conv-decomp tickFree-Cons-iff*
tickFree-append-iff)
finally show $\langle s \in \mathcal{D} \text{ ?lhs} \rangle$
by (*auto simp add: D-P(1, 4) *(1) front-tickFree-append comp-assoc intro!:*
is-processT7)
next

case $D-Q$
have $\langle s1 @ [\checkmark(g-r)] \notin \mathcal{D} (\text{Renaming } P f g) \rangle$ **by** (*meson* $D-Q(3)$ *is-processT9*)
with $D-Q(2-4)$ **obtain** $t1\ r$
where $*$: $\langle g-r = g\ r \rangle \langle r \in \checkmark s(P) \rangle \langle s1 = \text{map} (\text{map-event}_{ptick} f g) t1 \rangle \langle t1 @ [\checkmark(r)] \in \mathcal{T} P \rangle$
by (*auto simp add: Renaming-projs append-eq-map-conv tick-eq-map-event_{ptick}-iff*)
(metis append-Nil2 front-tickFree-Nil is-processT9 map-event_{ptick}-tickFree strict-ticks-of-memI)
from $*(1, 2) \langle \text{inj-on } g \checkmark s(P) \rangle$ **have** $\langle (THE\ r.\ r \in \checkmark s(P) \wedge g-r = g\ r) = r \rangle$
by (*auto dest: inj-onD*)
with $D-Q(5)$ **have** $\langle s2 \in \mathcal{D} (\text{Renaming } (Q\ r) f g') \rangle$ **by** *simp*
then obtain $t2\ t3$
where $**$: $\langle s2 = \text{map} (\text{map-event}_{ptick} f g') t2 @ t3 \rangle \langle tF\ t2 \rangle \langle ftF\ t3 \rangle \langle t2 \in \mathcal{D} (Q\ r) \rangle$
unfolding $D\text{-Renaming}$ **by** *blast*
from $*(4) ** (4)$ **have** $\langle \text{map} (ev \circ of-ev) t1 @ t2 \in \mathcal{D} (P ; \checkmark Q) \rangle$
by (*simp add: Seq_{ptick}-projs (metis append-T-imp-tickFree not-Cons-self)*)
with $**(2)$ **have** $\langle \text{map} (\text{map-event}_{ptick} f g') (\text{map} (ev \circ of-ev) t1 @ t2) \in \mathcal{D} ?lhs \rangle$
unfolding $D\text{-Renaming mem-Collect-eq}$
by (*metis append.right-neutral front-tickFree-Nil tickFree-append-iff tickFree-map-ev-comp*)
moreover have $\langle \text{map} (\text{map-event}_{ptick} f g') (\text{map} (ev \circ of-ev) t1 @ t2) @ t3 = s \rangle$
by (*simp add: D-Q(1) *(3) *(1)*)
*(metis *(3) D-Q(4) event_{ptick}.map-sel(1) in-set-conv-decomp map-event_{ptick}-tickFree tickFree-Cons-iff tickFree-append-iff)*
ultimately show $\langle s \in \mathcal{D} ?lhs \rangle$
by (*auto simp add: *(3) intro!: is-processT7[of $\langle - @ - \rangle$, simplified]*)
*(use *(1) D-Q(1) in force, use *(2) map-event_{ptick}-tickFree in blast)*
qed
next
assume *subset-div* : $\langle \mathcal{D} ?rhs \subseteq \mathcal{D} ?lhs \rangle$
fix $s\ X$ **assume** $\langle (s, X) \in \mathcal{F} ?rhs \rangle$
then consider $\langle s \in \mathcal{D} ?rhs \rangle$
| $(F-P)\ s1$ **where** $\langle s = \text{map} (ev \circ of-ev) s1 \rangle \langle (s1, \text{ref-Seq}_{ptick}\ X) \in \mathcal{F} (\text{Renaming } P f g) \rangle \langle s1 \notin \mathcal{D} (\text{Renaming } P f g) \rangle \langle tF\ s1 \rangle$
| $(F-Q)\ s1\ g-r\ s2$ **where** $\langle s = \text{map} (ev \circ of-ev) s1 @ s2 \rangle \langle s1 @ [\checkmark(g-r)] \in \mathcal{T} (\text{Renaming } P f g) \rangle$
 $\langle s1 \notin \mathcal{D} (\text{Renaming } P f g) \rangle \langle tF\ s1 \rangle \langle (s2, X) \in \mathcal{F} (\text{Renaming } (Q\ (THE\ r.\ r \in \checkmark s(P) \wedge g-r = g\ r)) f g') \rangle$
 $\langle s2 \notin \mathcal{D} (\text{Renaming } (Q\ (THE\ r.\ r \in \checkmark s(P) \wedge g-r = g\ r)) f g') \rangle$
by (*simp add: Seq_{ptick}-projs*)
(metis (no-types, lifting) F-imp-front-tickFree front-tickFree-chaon self-append-conv)
thus $\langle (s, X) \in \mathcal{F} ?lhs \rangle$
proof cases
from *subset-div D-F* **show** $\langle s \in \mathcal{D} ?rhs \implies (s, X) \in \mathcal{F} ?lhs \rangle$ **by** *blast*
next
case $F-P$

from $F\text{-}P(2, 3)$ **obtain** $t1$
where $*$: $\langle s1 = \text{map} (\text{map-event}_{\text{ptick}} f g) t1 \rangle \langle (t1, \text{map-event}_{\text{ptick}} f g - ' \text{ref-Seq}_{\text{ptick}} X) \in \mathcal{F} P \rangle$
unfolding *Renaming-projs* **by** *blast*
have $\langle \text{map-event}_{\text{ptick}} f g - ' (\text{ref-Seq}_{\text{ptick}} X) = \text{ref-Seq}_{\text{ptick}} (\text{map-event}_{\text{ptick}} f g' - ' X) \rangle$ **for** X
proof (*rule set-eqI*)
show $\langle e \in \text{map-event}_{\text{ptick}} f g - ' (\text{ref-Seq}_{\text{ptick}} X) \longleftrightarrow e \in \text{ref-Seq}_{\text{ptick}} (\text{map-event}_{\text{ptick}} f g' - ' X) \rangle$ **for** e
by (*cases e, auto simp add: ref-Seq_{ptick}-def image-iff*)
(metis Int-iff event_{ptick}.sel(1) event_{ptick}.sims(9) rangeI vimage-eq, metis IntI UNIV-I event_{ptick}.sel(1) image-eqI)
qed
with $*(2)$ **have** $\langle (t1, \text{ref-Seq}_{\text{ptick}} (\text{map-event}_{\text{ptick}} f g' - ' X)) \in \mathcal{F} P \rangle$ **by**
simp
hence $\langle (\text{map} (ev \circ \text{of-ev}) t1, \text{map-event}_{\text{ptick}} f g' - ' X) \in \mathcal{F} (P ; \checkmark Q) \rangle$
by (*simp add: Seq_{ptick}-projs*)
*(metis *(1) F-P(4) map-event_{ptick}-tickFree)*
hence $\langle (\text{map} (\text{map-event}_{\text{ptick}} f g') (\text{map} (ev \circ \text{of-ev}) t1), X) \in \mathcal{F} ?\text{lhs} \rangle$
unfolding *F-Renaming* **by** *blast*
also have $\langle \text{map} (\text{map-event}_{\text{ptick}} f g') (\text{map} (ev \circ \text{of-ev}) t1) = s \rangle$
by (*simp add: F-P(1) *(1)*)
*(metis *(1) F-P(4) event_{ptick}.map-sel(1) in-set-conv-decomp map-event_{ptick}-tickFree tickFree-Cons-iff tickFree-append-iff)*
finally show $\langle (s, X) \in \mathcal{F} ?\text{lhs} \rangle$.
next
case $F\text{-}Q$
have $\langle s1 @ [\checkmark(g-r)] \notin \mathcal{D} (\text{Renaming } P f g) \rangle$ **by** (*meson F-Q(3) is-processT9*)
with $F\text{-}Q(2-4)$ **obtain** $t1 r$
where $*$: $\langle g-r = g r \rangle \langle r \in \checkmark s(P) \rangle \langle s1 = \text{map} (\text{map-event}_{\text{ptick}} f g) t1 \rangle \langle t1 @ [\checkmark(r)] \in \mathcal{T} P \rangle$
by (*auto simp add: Renaming-projs append-eq-map-conv tick-eq-map-event_{ptick}-iff*)
(metis append-Nil2 front-tickFree-Nil is-processT9 map-event_{ptick}-tickFree strict-ticks-of-memI)
from $*(1, 2)$ $\langle \text{inj-on } g \checkmark s(P) \rangle$ **have** $\langle (\text{THE } r. r \in \checkmark s(P) \wedge g-r = g r) = r \rangle$
by (*auto dest: inj-onD*)
with $F\text{-}Q(5, 6)$ **have** $\langle (s2, X) \in \mathcal{F} (\text{Renaming } (Q r) f g') \rangle$
 $\langle s2 \notin \mathcal{D} (\text{Renaming } (Q r) f g') \rangle$ **by** *simp-all*
then obtain $t2$ **where** $**$: $\langle s2 = \text{map} (\text{map-event}_{\text{ptick}} f g') t2 \rangle \langle (t2, \text{map-event}_{\text{ptick}} f g' - ' X) \in \mathcal{F} (Q r) \rangle$
unfolding *Renaming-projs* **by** *blast*
from $*(4) ** (2)$ **have** $\langle (\text{map} (ev \circ \text{of-ev}) t1 @ t2, \text{map-event}_{\text{ptick}} f g' - ' X) \in \mathcal{F} (P ; \checkmark Q) \rangle$
by (*simp add: Seq_{ptick}-projs*) *(metis append-T-imp-tickFree not-Cons-self2)*
hence $\langle (\text{map} (\text{map-event}_{\text{ptick}} f g') (\text{map} (ev \circ \text{of-ev}) t1 @ t2), X) \in \mathcal{F} ?\text{lhs} \rangle$
unfolding *F-Renaming* **by** *blast*
also have $\langle \text{map} (\text{map-event}_{\text{ptick}} f g') (\text{map} (ev \circ \text{of-ev}) t1 @ t2) = s \rangle$
by (*simp add: F-Q(1) *(3) ** (1)*)
*(metis *(3) F-Q(4) event_{ptick}.map-sel(1) in-set-conv-decomp)*

$\text{map-event}_{\text{ptick-tickFree tickFree-Cons-iff tickFree-append-iff}}$
finally show $\langle (s, X) \in \mathcal{F} \text{ ?lhs} \rangle$.
qed
qed
next

have $\langle \text{?rhs} = \text{Renaming } P \text{ f } g ;_{\checkmark} (\lambda g\text{-}r. \sqcap r \in \{r \in \check{\mathbf{s}}(P). g\text{-}r = g \text{ } r\}. \text{Renaming } (Q \text{ } r) \text{ f } g') \rangle$
proof (*rule mono-Seq_{ptick}-eq*)
show $\langle \text{Renaming } P \text{ f } g = \text{Renaming } P \text{ f } g \rangle$..
next
fix $g\text{-}r$ **assume** $\langle g\text{-}r \in \check{\mathbf{s}}(\text{Renaming } P \text{ f } g) \rangle$
then obtain $s \text{ } s1$ **where** $\langle s @ [\check{\mathbf{J}}(g\text{-}r)] = \text{map } (\text{map-event}_{\text{ptick}} \text{ f } g) \text{ } s1 \rangle \langle s1 \in \mathcal{T} \text{ } P \rangle \langle s1 \notin \mathcal{D} \text{ } P \rangle$
by (*simp add: strict-ticks-of-def Renaming-projs*)
(metis (no-types, opaque-lifting) T-imp-front-tickFree append-Nil2 butlast-snoc div-butlast-when-non-tickFree-iff front-tickFree-Nil front-tickFree-iff-tickFree-butlast front-tickFree-single map-butlast)
from this(1) obtain $s1' \text{ } r$ **where** $\langle g\text{-}r = g \text{ } r \rangle \langle s1 = s1' @ [\check{\mathbf{J}}(r)] \rangle$
by (*cases s1 rule: rev-cases*) (*auto simp add: tick-eq-map-event_{ptick}-iff*)
with $\langle s1 \in \mathcal{T} \text{ } P \rangle \langle s1 \notin \mathcal{D} \text{ } P \rangle$ **have** $\langle s1' @ [\check{\mathbf{J}}(r)] \in \mathcal{T} \text{ } P \rangle \langle s1' @ [\check{\mathbf{J}}(r)] \notin \mathcal{D} \text{ } P \rangle$ **by** *simp-all*
hence $\langle r \in \check{\mathbf{s}}(P) \rangle$ **unfolding** *strict-ticks-of-def* **by** *blast*
have $\langle \{r \in \check{\mathbf{s}}(P). g\text{-}r = g \text{ } r\} = \{r\} \rangle$
by (*auto simp add: $\langle r \in \check{\mathbf{s}}(P) \rangle \langle g\text{-}r = g \text{ } r \rangle$ intro: inj-onD[OF $\langle \text{inj-on } g \check{\mathbf{s}}(P) \rangle$]*)
moreover have $\langle (\text{THE } r. r \in \check{\mathbf{s}}(P) \wedge g\text{-}r = g \text{ } r) = r \rangle$
using *calculation* **by** *blast*
ultimately have $\langle Q (\text{THE } r. r \in \check{\mathbf{s}}(P) \wedge g\text{-}r = g \text{ } r) = \text{GlobalNdet } \{r \in \check{\mathbf{s}}(P). g\text{-}r = g \text{ } r\} \text{ } Q \rangle$ **by** *simp*
thus $\langle \text{Renaming } (Q (\text{THE } r. r \in \check{\mathbf{s}}(P) \wedge g\text{-}r = g \text{ } r)) \text{ f } g' = (\sqcap r \in \{r \in \check{\mathbf{s}}(P). g\text{-}r = g \text{ } r\}. \text{Renaming } (Q \text{ } r) \text{ f } g') \rangle$
by (*simp flip: Renaming-distrib-GlobalNdet*)
qed
thus $\langle \text{?rhs} \sqsubseteq_{FD} \text{?lhs} \rangle$ **by** (*simp add: FD-Renaming-Seq_{ptick}*)
qed

When $'r$ is set on *unit*, we recover the version that we had before the generalization.

lemma $\langle \text{Renaming } (P ;_{\checkmark} Q) \text{ f } g = \text{Renaming } P \text{ f } g ;_{\checkmark} (\lambda r. \text{Renaming } (Q \text{ } ()) \text{ f } g) \rangle$
by (*subst inj-on-Renaming-Seq_{ptick}[where $g = g$]]*) (*auto intro: inj-onI*)

lemma *TickSwap-Seq_{ptick} [simp]* :
 $\langle \text{TickSwap } (P ;_{\checkmark} Q) = \text{TickSwap } P ;_{\checkmark} (\lambda(s, r). \text{TickSwap } (Q \text{ } (r, s))) \rangle$ (**is** $\langle \text{?lhs} = \text{?rhs} \rangle$)
proof –
have $\langle \text{?lhs} = \text{Renaming } (P ;_{\checkmark} Q) \text{ id prod.swap} \rangle$ **by** (*simp add: TickSwap-is-Renaming*)
also have $\langle \dots = \text{Renaming } P \text{ id prod.swap} ;_{\checkmark} \rangle$

$(\lambda s.r. \text{ Renaming } (Q \text{ (THE } r.s. \text{ } r.s \in \text{strict-ticks-of } P \wedge$
 $\text{ } s.r = \text{prod.swap } r.s)) \text{ id prod.swap}))$
 (is $\langle - = - ; \checkmark ?rhs \rangle$) by (simp add: inj-on-Renaming-Seq_{ptick})
 also have $\langle \dots = ?rhs \rangle$
 proof (rule mono-Seq_{ptick}-eq, unfold TickSwap-is-Renaming)
 show $\langle \text{Renaming } P \text{ id prod.swap} = \text{Renaming } P \text{ id prod.swap} \rangle \dots$
 next
 fix $s.r$ assume $\langle s.r \in \text{strict-ticks-of } (\text{Renaming } P \text{ id prod.swap}) \rangle$
 then obtain $r \ s$ where $\langle (r, s) \in \text{strict-ticks-of } P \rangle \langle s.r = (s, r) \rangle$
 by (auto simp flip: TickSwap-is-Renaming)
 hence $\langle (\text{THE } r.s. \text{ } r.s \in \text{strict-ticks-of } P \wedge s.r = \text{prod.swap } r.s) = (r, s) \rangle$ by
 auto
 thus $\langle \text{Renaming } (Q \text{ (THE } r.s. \text{ } r.s \in \text{strict-ticks-of } P \wedge s.r = \text{prod.swap } r.s))$
 $\text{ id prod.swap} =$
 $(\text{case } s.r \text{ of } (s, r) \Rightarrow \text{Renaming } (Q \text{ (} r, s)) \text{ id prod.swap}))$
 by (simp add: $\langle s.r = (s, r) \rangle$)
 qed
 finally show $\langle ?lhs = ?rhs \rangle$.
 qed

lemma TickSwap-is-Seq_{ptick}-iff [simp] :
 $\langle \text{TickSwap } P = Q ; \checkmark R \longleftrightarrow P = \text{TickSwap } Q ; \checkmark (\lambda(r, s). \text{TickSwap } (R \text{ (} s, r))) \rangle$
 by (simp add: TickSwap-eq-iff-eq-TickSwap)

16.1.2 Renaming and Synchronization Product

theorem (in Sync_{ptick}-locale) inj-RenamingEv-Sync_{ptick} :
 $\langle \text{RenamingEv } (P \llbracket S \rrbracket \checkmark Q) f = \text{RenamingEv } P f \llbracket f \text{ ' } S \rrbracket \checkmark \text{RenamingEv } Q f \rangle$
 (is $\langle ?lhs = ?rhs \rangle$) if $\langle \text{inj } f \rangle$
proof –
 let $?fun = \langle \text{map-event}_{\text{ptick}} f \text{ id} \rangle$
 let $?map = \langle \text{map } ?fun \rangle$
 let $?R = \langle \lambda P. \text{RenamingEv } P f \rangle$
 show $\langle ?lhs = ?rhs \rangle$
proof (rule Process-eq-optimizedI)
 fix t assume $\langle t \in \mathcal{D} \text{ } ?lhs \rangle$
 then obtain $t1 \ t2$ where $* : \langle t = ?map \ t1 \ @ \ t2 \rangle$
 $\langle tF \ t1 \rangle \langle ftF \ t2 \rangle \langle t1 \in \mathcal{D} \text{ (} P \llbracket S \rrbracket \checkmark Q) \rangle$ unfolding D-Renaming by blast
from $*(4)$ obtain $u \ v \ t-P \ t-Q$ where $** : \langle t1 = u \ @ \ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((t-P, t-Q), S) \rangle$
 $\langle t-P \in \mathcal{D} \ P \wedge t-Q \in \mathcal{T} \ Q \vee t-P \in \mathcal{T} \ P \wedge t-Q \in \mathcal{D} \ Q \rangle$
 unfolding D-Sync_{ptick} by blast
from setinterleaves_{ptick}-inj-map-map-event_{ptick}-iff-weak [THEN iffD2, OF $\langle \text{inj } f \rangle \text{ } ** (4) \text{ }]$
 have $\langle ?map \ u \text{ setinterleaves}_{\checkmark(\otimes \checkmark)} ((?map \ t-P, ?map \ t-Q), f \text{ ' } S) \rangle$.
moreover from $*(5)$ have $\langle ?map \ t-P \in \mathcal{D} \text{ (} ?R \ P) \wedge ?map \ t-Q \in \mathcal{T} \text{ (} ?R \ Q) \rangle$
 $\vee$
 $\langle ?map \ t-P \in \mathcal{T} \text{ (} ?R \ P) \wedge ?map \ t-Q \in \mathcal{D} \text{ (} ?R \ Q) \rangle$
 by (auto simp add: Renaming-projs dest: D-T)

$(metis \text{ **}(2,4) \text{ append-self-conv front-tickFree-map-tick-iff}$
 $\text{ list.map-disc-iff tickFree-setinterleaves}_{ptick}\text{-iff}) +$
moreover have $\langle t = ?map\ u\ @\ (?map\ v\ @\ t2) \rangle$ **by** $(simp\ add: \text{ **}(1) \text{ **}(1))$
moreover have $\langle tF\ (?map\ u) \rangle$ **by** $(simp\ add: \text{ **}(2) \text{ map-event}_{ptick}\text{-tickFree})$
moreover from $\text{ **}(2,3) \text{ **}(1)$ **have** $\langle ftF\ (?map\ v\ @\ t2) \rangle$
using $\text{ front-tickFree-append map-event}_{ptick}\text{-tickFree tickFree-append-iff}$ **by**
 blast
ultimately show $\langle t \in \mathcal{D}\ ?rhs \rangle$ **unfolding** $D\text{-Sync}_{ptick}$ **by** blast
next
fix t **assume** $\langle t \in \mathcal{D}\ ?rhs \rangle$
then obtain $u\ v\ t\text{-}P\ t\text{-}Q$ **where** $*$: $\langle t = u\ @\ v \rangle \langle tF\ u \rangle \langle ftF\ v \rangle$
 $\langle u\ \text{setinterleaves}_{\checkmark}(\otimes\checkmark) ((t\text{-}P, t\text{-}Q), f\ 'S) \rangle$
 $\langle t\text{-}P \in \mathcal{D}\ (?R\ P) \wedge t\text{-}Q \in \mathcal{T}\ (?R\ Q) \vee t\text{-}P \in \mathcal{T}\ (?R\ P) \wedge t\text{-}Q \in \mathcal{D}\ (?R\ Q) \rangle$
unfolding $D\text{-Sync}_{ptick}$ **by** blast
from $\text{ **}(5)$ **show** $\langle t \in \mathcal{D}\ ?lhs \rangle$
proof $(elim\ disjE\ conjE)$
assume $\langle t\text{-}P \in \mathcal{D}\ (?R\ P) \rangle \langle t\text{-}Q \in \mathcal{T}\ (?R\ Q) \rangle$
from $\langle t\text{-}P \in \mathcal{D}\ (?R\ P) \rangle$ **obtain** $t\text{-}P1\ t\text{-}P2$
where $*$: $\langle t\text{-}P = ?map\ t\text{-}P1\ @\ t\text{-}P2 \rangle \langle tF\ t\text{-}P1 \rangle \langle ftF\ t\text{-}P2 \rangle \langle t\text{-}P1 \in \mathcal{D}\ P \rangle$
unfolding $D\text{-Renaming}$ **by** blast
from $\langle t\text{-}Q \in \mathcal{T}\ (?R\ Q) \rangle$ **consider** $(T\text{-}Q)\ t\text{-}Q1$ **where** $\langle t\text{-}Q = ?map\ t\text{-}Q1 \rangle$
 $\langle t\text{-}Q1 \in \mathcal{T}\ Q \rangle$
 $| (D\text{-}Q)\ t\text{-}Q1\ t\text{-}Q2$ **where** $\langle t\text{-}Q = ?map\ t\text{-}Q1\ @\ t\text{-}Q2 \rangle \langle tF\ t\text{-}Q1 \rangle \langle ftF\ t\text{-}Q2 \rangle$
 $\langle t\text{-}Q1 \in \mathcal{D}\ Q \rangle$
unfolding $T\text{-Renaming}$ **by** blast
thus $\langle t \in \mathcal{D}\ ?lhs \rangle$
proof cases
case $T\text{-}Q$
from $\text{ **}(4)[\text{unfolded } \text{ **}(1)\ T\text{-}Q(1), \text{ THEN } \text{setinterleaves}_{ptick}\text{-appendL}]$
obtain $u1\ u2\ t\text{-}Q11\ t\text{-}Q12$ **where** $***$: $\langle u = u1\ @\ u2 \rangle \langle ?map\ t\text{-}Q1 = t\text{-}Q11$
 $@\ t\text{-}Q12 \rangle$
 $\langle u1\ \text{setinterleaves}_{\checkmark}(\otimes\checkmark) ((?map\ t\text{-}P1, t\text{-}Q11), f\ 'S) \rangle$ **by** blast
obtain $t\text{-}Q11'$ **where** $\langle t\text{-}Q11' \leq t\text{-}Q1 \rangle \langle t\text{-}Q11 = ?map\ t\text{-}Q11' \rangle$
by $(metis \text{ ***}(2) \text{ map-eq-append-conv Prefix-Order.prefixI})$
from $\text{setinterleaves}_{ptick}\text{-inj-map-map-event}_{ptick}\text{-iff-strong}$
 $[\text{ THEN } \text{ iffD1}, \text{ OF } \langle \text{inj } f \rangle \text{ ***}(3)[\text{unfolded this}]]$
obtain $u1'$ **where** $****$: $\langle u1 = ?map\ u1' \rangle$
 $\langle u1'\ \text{setinterleaves}_{\checkmark}(\otimes\checkmark) ((t\text{-}P1, t\text{-}Q11'), S) \rangle$ **by** blast
from $\text{ **}(2) \text{ ***}(1) \text{ ****}(1) \text{ map-event}_{ptick}\text{-tickFree}$
 $T\text{-}Q(2) \langle t\text{-}Q11' \leq t\text{-}Q1 \rangle \text{ is-process } T3\text{-}TR$
have $\langle u1' = u1' @ [] \rangle \langle tF\ u1' \rangle \langle ftF\ [] \rangle \langle t\text{-}Q11' \in \mathcal{T}\ Q \rangle$ **by** simp-all blast
with $****(2) \text{ **}(4)$ **have** $\langle u1' \in \mathcal{D}\ (P\ \llbracket S \rrbracket_{\checkmark}\ Q) \rangle$
unfolding $D\text{-Sync}_{ptick}$ **by** blast
moreover have $\langle t = ?map\ u1' @ (u2 @ v) \rangle$ **by** $(simp\ add: \text{ **}(1) \text{ ***}(1))$
 $****(1))$
moreover have $\langle ftF\ (u2 @ v) \rangle$
using $\text{ **}(2,3) \text{ ***}(1) \text{ front-tickFree-append tickFree-append-iff}$ **by** blast
ultimately show $\langle t \in \mathcal{D}\ ?lhs \rangle$ **unfolding** $D\text{-Renaming}$ **using** $\langle tF\ u1' \rangle$ **by**
 blast

```

next
case D-Q
have  $\langle ?map\ t-P1 \leq t-P \rangle \langle ?map\ t-Q1 \leq t-Q \rangle$ 
  by (simp-all add:  $**(1)$  D-Q(1))
from setinterleavesptick-le-prefixLR[OF  $*(4)$  this] show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
proof (elim disjE exE conjE)
  fix  $u1\ t-Q1'$  assume  $*** : \langle u1 \leq u \rangle \langle t-Q1' \leq ?map\ t-Q1 \rangle$ 
     $\langle u1\ setinterleaves_{\checkmark}(\otimes\checkmark) ((?map\ t-P1, t-Q1'), f\ 'S) \rangle$ 
  obtain  $u2$  where  $\langle u = u1 @ u2 \rangle$  using  $***(1)$  prefixE by blast
  obtain  $t-Q1''$  where  $\langle t-Q1' = ?map\ t-Q1'' \rangle \langle t-Q1'' \leq t-Q1 \rangle$ 
    by (metis  $***(2)$  prefixE prefixI map-eq-append-conv)
  from setinterleavesptick-inj-map-map-eventptick-iff-strong
    [THEN iffD1, OF  $\langle inj\ f \rangle *** (3)[unfolded\ \langle t-Q1' = ?map\ t-Q1'' \rangle]$ ]
  obtain  $u1'$  where  $**** : \langle u1 = ?map\ u1' \rangle$ 
     $\langle u1'\ setinterleaves_{\checkmark}(\otimes\checkmark) ((t-P1, t-Q1''), S) \rangle$  by blast
  have  $\langle u1' = u1' @ [] \rangle \langle ftF\ [] \rangle$  by simp-all
  moreover from  $**(2)$   $****(2)$  setinterleavesptick-tickFree-imp
  have  $\langle tF\ u1' \rangle$  by blast
  moreover from D-Q(4) D-T  $\langle t-Q1'' \leq t-Q1 \rangle$  is-processT3-TR
  have  $\langle t-Q1'' \in \mathcal{T}\ Q \rangle$  by blast
  ultimately have  $\langle u1' \in \mathcal{D}\ (P\ [S]_{\checkmark}\ Q) \rangle$ 
    unfolding D-Syncptick using  $\langle t-P1 \in \mathcal{D}\ P \rangle ****(2)$  by blast
  moreover from  $*(1-3)$   $****(1)$ 
  have  $\langle t = ?map\ u1' @ (u2 @ v) \rangle \langle ftF\ (u2 @ v) \rangle$ 
    by (auto simp add:  $\langle u = u1 @ u2 \rangle$  front-tickFree-append)
  ultimately show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
    unfolding D-Renaming using  $\langle tF\ u1' \rangle$  by blast
next
fix  $u1\ t-P1'$  assume  $*** : \langle u1 \leq u \rangle \langle t-P1' \leq ?map\ t-P1 \rangle$ 
   $\langle u1\ setinterleaves_{\checkmark}(\otimes\checkmark) ((t-P1', ?map\ t-Q1), f\ 'S) \rangle$ 
  obtain  $u2$  where  $\langle u = u1 @ u2 \rangle$  using  $***(1)$  prefixE by blast
  obtain  $t-P1''$  where  $\langle t-P1' = ?map\ t-P1'' \rangle \langle t-P1'' \leq t-P1 \rangle$ 
    by (metis  $***(2)$  prefixE prefixI map-eq-append-conv)
  from setinterleavesptick-inj-map-map-eventptick-iff-strong
    [THEN iffD1, OF  $\langle inj\ f \rangle *** (3)[unfolded\ \langle t-P1' = ?map\ t-P1'' \rangle]$ ]
  obtain  $u1'$  where  $**** : \langle u1 = ?map\ u1' \rangle$ 
     $\langle u1'\ setinterleaves_{\checkmark}(\otimes\checkmark) ((t-P1'', t-Q1), S) \rangle$  by blast
  have  $\langle u1' = u1' @ [] \rangle \langle ftF\ [] \rangle$  by simp-all
  moreover from D-Q(2)  $****(2)$  setinterleavesptick-tickFree-imp
  have  $\langle tF\ u1' \rangle$  by blast
  moreover from  $*(4)$  D-T  $\langle t-P1'' \leq t-P1 \rangle$  is-processT3-TR
  have  $\langle t-P1'' \in \mathcal{T}\ P \rangle$  by blast
  ultimately have  $\langle u1' \in \mathcal{D}\ (P\ [S]_{\checkmark}\ Q) \rangle$ 
    unfolding D-Syncptick using  $\langle t-Q1 \in \mathcal{D}\ Q \rangle ****(2)$  by blast
  moreover from  $*(1-3)$   $****(1)$ 
  have  $\langle t = ?map\ u1' @ (u2 @ v) \rangle \langle ftF\ (u2 @ v) \rangle$ 
    by (auto simp add:  $\langle u = u1 @ u2 \rangle$  front-tickFree-append)
  ultimately show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
    unfolding D-Renaming using  $\langle tF\ u1' \rangle$  by blast

```



```

qed
qed
next
  assume  $\langle t-Q \in \mathcal{D} \ (\?R \ Q) \rangle \ \langle t-P \in \mathcal{T} \ (\?R \ P) \rangle$ 
  from  $\langle t-Q \in \mathcal{D} \ (\?R \ Q) \rangle$  obtain  $t-Q1 \ t-Q2$ 
    where  $\ast : \langle t-Q = ?map \ t-Q1 \ @ \ t-Q2 \rangle \ \langle tF \ t-Q1 \rangle \ \langle tF \ t-Q2 \rangle \ \langle t-Q1 \in \mathcal{D} \ Q \rangle$ 
    unfolding D-Renaming by blast
    from  $\langle t-P \in \mathcal{T} \ (\?R \ P) \rangle$  consider  $(T-P) \ t-P1$  where  $\langle t-P = ?map \ t-P1 \rangle$ 
 $\langle t-P1 \in \mathcal{T} \ P \rangle$ 
    |  $(D-P) \ t-P1 \ t-P2$  where  $\langle t-P = ?map \ t-P1 \ @ \ t-P2 \rangle \ \langle tF \ t-P1 \rangle \ \langle tF \ t-P2 \rangle$ 
 $\langle t-P1 \in \mathcal{D} \ P \rangle$ 
    unfolding T-Renaming by blast
  thus  $\langle t \in \mathcal{D} \ ?lhs \rangle$ 
  proof cases
    case T-P
      from  $\ast(4)[unfolding \ \ast(1) \ T-P(1), \ THEN \ setinterleaves_{ptick}\text{-appendR}]$ 
      obtain  $u1 \ u2 \ t-P11 \ t-P12$  where  $\ast\ast : \langle u = u1 \ @ \ u2 \rangle \ \langle ?map \ t-P1 = t-P11$ 
 $@ \ t-P12 \rangle$ 
         $\langle u1 \ setinterleaves_{\checkmark}(\otimes\checkmark) ((t-P11, \ ?map \ t-Q1), \ f \ ' \ S) \rangle$  by blast
      obtain  $t-P11'$  where  $\langle t-P11' \leq t-P1 \rangle \ \langle t-P11 = ?map \ t-P11' \rangle$ 
        by (metis  $\ast\ast(2)$  map-eq-append-conv Prefix-Order.prefixI)
      from setinterleavesptick-inj-map-map-eventptick-iff-strong
        [THEN iffD1, OF  $\langle inj \ f \rangle \ \ast\ast(3)[unfolding \ this]$ ]
      obtain  $u1'$  where  $\ast\ast\ast : \langle u1 = ?map \ u1' \rangle$ 
         $\langle u1' \ setinterleaves_{\checkmark}(\otimes\checkmark) ((t-P11', \ t-Q1), \ S) \rangle$  by blast
      from  $\ast(2) \ \ast\ast(1) \ \ast\ast\ast(1)$  map-eventptick-tickFree
         $T-P(2) \ \langle t-P11' \leq t-P1 \rangle$  is-processT3-TR
      have  $\langle u1' = u1' @ [] \rangle \ \langle tF \ u1' \rangle \ \langle tF \ [] \rangle \ \langle t-P11' \in \mathcal{T} \ P \rangle$  by simp-all blast
      with  $\ast\ast\ast(2) \ \ast(4)$  have  $\langle u1' \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
        unfolding D-Syncptick by blast
      moreover have  $\langle t = ?map \ u1' @ (u2 @ v) \rangle$  by (simp add:  $\ast(1) \ \ast\ast(1)$ 
 $\ast\ast\ast(1))$ 
      moreover have  $\langle tF \ (u2 @ v) \rangle$ 
        using  $\ast(2,3) \ \ast\ast(1)$  front-tickFree-append tickFree-append-iff by blast
      ultimately show  $\langle t \in \mathcal{D} \ ?lhs \rangle$  unfolding D-Renaming using  $\langle tF \ u1' \rangle$  by
blast
  next
    case D-P
      have  $\langle ?map \ t-P1 \leq t-P \rangle \ \langle ?map \ t-Q1 \leq t-Q \rangle$ 
        by (simp-all add:  $\ast(1) \ D-P(1)$ )
      from setinterleavesptick-le-prefixLR[OF  $\ast(4)$  this] show  $\langle t \in \mathcal{D} \ ?lhs \rangle$ 
      proof (elim disjE exE conjE)
        fix  $u1 \ t-Q1'$  assume  $\ast\ast : \langle u1 \leq u \rangle \ \langle t-Q1' \leq ?map \ t-Q1 \rangle$ 
           $\langle u1 \ setinterleaves_{\checkmark}(\otimes\checkmark) ((?map \ t-P1, \ t-Q1'), \ f \ ' \ S) \rangle$ 
        obtain  $u2$  where  $\langle u = u1 @ u2 \rangle$  using  $\ast\ast(1)$  prefixE by blast
        obtain  $t-Q1''$  where  $\langle t-Q1' = ?map \ t-Q1'' \rangle \ \langle t-Q1'' \leq t-Q1 \rangle$ 
          by (metis  $\ast\ast(2)$  prefixE prefixI map-eq-append-conv)
        from setinterleavesptick-inj-map-map-eventptick-iff-strong
          [THEN iffD1, OF  $\langle inj \ f \rangle \ \ast\ast(3)[unfolding \ \langle t-Q1' = ?map \ t-Q1'' \rangle]$ ]

```

```

obtain  $u1'$  where  $**** : \langle u1 = ?map\ u1' \rangle$ 
   $\langle u1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P1, t-Q1''), S) \rangle$  by blast
have  $\langle u1' = u1' @ [] \rangle \langle ftF [] \rangle$  by simp-all
moreover from  $D-P(2)$   $****(2)$  setinterleavesptick-tickFree-imp
have  $\langle tF\ u1' \rangle$  by blast
moreover from  $*(4)$   $D-T$   $\langle t-Q1'' \leq t-Q1 \rangle$  is-processT3-TR
have  $\langle t-Q1'' \in \mathcal{T}\ Q \rangle$  by blast
ultimately have  $\langle u1' \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  unfolding  $D-Sync_{ptick}$  using  $\langle t-P1 \in \mathcal{D}\ P \rangle$   $****(2)$  by blast
moreover from  $*(1-3)$   $****(1)$ 
have  $\langle t = ?map\ u1' @ (u2 @ v) \rangle \langle ftF\ (u2 @ v) \rangle$ 
  by (auto simp add:  $\langle u = u1 @ u2 \rangle$  front-tickFree-append)
ultimately show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
  unfolding  $D-Renaming$  using  $\langle tF\ u1' \rangle$  by blast
next
fix  $u1\ t-P1'$  assume  $*** : \langle u1 \leq u \rangle \langle t-P1' \leq ?map\ t-P1 \rangle$ 
   $\langle u1 \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P1', ?map\ t-Q1), f\ 'S) \rangle$ 
obtain  $u2$  where  $\langle u = u1 @ u2 \rangle$  using  $***(1)$  prefixE by blast
obtain  $t-P1''$  where  $\langle t-P1' = ?map\ t-P1'' \rangle \langle t-P1'' \leq t-P1 \rangle$ 
  by (metis  $***(2)$  prefixE prefixI map-eq-append-conv)
from setinterleavesptick-inj-map-map-eventptick-iff-strong
   $[THEN\ iffD1, OF\ \langle inj\ f \rangle\ ***(3)[unfolding\ \langle t-P1' = ?map\ t-P1'' \rangle]]$ 
obtain  $u1'$  where  $**** : \langle u1 = ?map\ u1' \rangle$ 
   $\langle u1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P1'', t-Q1), S) \rangle$  by blast
have  $\langle u1' = u1' @ [] \rangle \langle ftF [] \rangle$  by simp-all
moreover from  $*(2)$   $****(2)$  setinterleavesptick-tickFree-imp
have  $\langle tF\ u1' \rangle$  by blast
moreover from  $D-P(4)$   $D-T$   $\langle t-P1'' \leq t-P1 \rangle$  is-processT3-TR
have  $\langle t-P1'' \in \mathcal{T}\ P \rangle$  by blast
ultimately have  $\langle u1' \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  unfolding  $D-Sync_{ptick}$  using  $\langle t-Q1 \in \mathcal{D}\ Q \rangle$   $****(2)$  by blast
moreover from  $*(1-3)$   $****(1)$ 
have  $\langle t = ?map\ u1' @ (u2 @ v) \rangle \langle ftF\ (u2 @ v) \rangle$ 
  by (auto simp add:  $\langle u = u1 @ u2 \rangle$  front-tickFree-append)
ultimately show  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
  unfolding  $D-Renaming$  using  $\langle tF\ u1' \rangle$  by blast
qed
qed
qed
next
fix  $t\ X$  assume  $\langle (t, X) \in \mathcal{F}\ ?lhs \rangle \langle t \notin \mathcal{D}\ ?lhs \rangle$ 
then obtain  $t'$  where  $\langle t = ?map\ t' \rangle$ 
  and  $*$  :  $\langle (t', ?fun - 'X) \in \mathcal{F}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  unfolding  $Renaming-projs$  by blast
have  $\langle t' \notin \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
proof (rule notI)
  assume  $\langle t' \in \mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
  hence  $\langle t \in \mathcal{D}\ ?lhs \rangle$ 
  by (simp add:  $\langle t = ?map\ t' \rangle$  D-Syncptick D-Renaming)

```

```

    (metis (no-types) append-Nil2 front-tickFree-Nil map-append map-eventptick-front-tickFree)
  with  $\langle t \notin \mathcal{D} \text{ ?lhs} \rangle$  show False ..
qed
with * obtain  $t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$ 
  where ** :  $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} \ P \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} \ Q \rangle$ 
     $\langle t' \text{ setinterleaves}_{\checkmark}((t\text{-}P, t\text{-}Q), S) \rangle$ 
     $\langle ?fun - ' X \subseteq \text{super-ref-Sync}_{ptick}(\otimes \checkmark) X\text{-}P \ S \ X\text{-}Q \rangle$ 
  unfolding Syncptick-projs by fast
from ** (2, 3) F-T  $\langle t' \notin \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$  append-Nil2 front-tickFree-Nil
have  $\langle t\text{-}P \notin \mathcal{D} \ P \rangle$  unfolding D-Syncptick' by blast
from ** (1, 3) F-T  $\langle t' \notin \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$  append-Nil2 front-tickFree-Nil
have  $\langle t\text{-}Q \notin \mathcal{D} \ Q \rangle$  unfolding D-Syncptick' by blast
have *** :  $\langle ?fun - ' ?fun - ' X\text{-}P = X\text{-}P \rangle \langle ?fun - ' ?fun - ' X\text{-}Q = X\text{-}Q \rangle$ 
  by (simp add: set-eq-iff image-iff,
      metis (mono-tags, opaque-lifting) eventptick.inj-map-strong id-apply injD
     $\langle inj \ f \rangle$ )
from ** (1) have  $\langle (?map \ t\text{-}P, ?fun - ' X\text{-}P) \in \mathcal{F} \ (?R \ P) \rangle$ 
  by (subst (asm) *** (1) [symmetric]) (auto simp add: F-Renaming)
moreover {
  fix  $a$  assume  $\langle ?map \ t\text{-}P @ [ev \ a] \in \mathcal{T} \ (?R \ P) \rangle \langle a \notin \text{range } f \rangle$ 
  then consider  $t\text{-}P1$  where  $\langle ?map \ t\text{-}P @ [ev \ a] = ?map \ t\text{-}P1 \rangle \langle t\text{-}P1 \in \mathcal{T} \ P \rangle$ 
    |  $t\text{-}P1 \ t\text{-}P2$  where  $\langle ?map \ t\text{-}P @ [ev \ a] = ?map \ t\text{-}P1 @ t\text{-}P2 \rangle \langle tF \ t\text{-}P1 \rangle$ 
   $\langle t\text{-}P1 \in \mathcal{D} \ P \rangle$ 
  unfolding T-Renaming by blast
  hence False
  proof cases
    from  $\langle a \notin \text{range } f \rangle$  show  $\langle ?map \ t\text{-}P @ [ev \ a] = ?map \ t\text{-}P1 \implies \text{False} \rangle$  for
   $t\text{-}P1$ 
    by (auto simp add: append-eq-map-conv ev-eq-map-eventptick-iff)
  next
    fix  $t\text{-}P1 \ t\text{-}P2$  assume  $\langle ?map \ t\text{-}P @ [ev \ a] = ?map \ t\text{-}P1 @ t\text{-}P2 \rangle \langle tF \ t\text{-}P1 \rangle$ 
   $\langle t\text{-}P1 \in \mathcal{D} \ P \rangle$ 
    from this (1)  $\langle a \notin \text{range } f \rangle$  have  $\langle t\text{-}P1 \leq t\text{-}P \rangle$ 
    by (cases  $t\text{-}P2$  rule: rev-cases, auto simp add: append-eq-map-conv
      ev-eq-map-eventptick-iff)
    (metis prefixI eventptick.inj-map inj-map-eq-map inj-on-id map-eq-append-conv
     $\langle inj \ f \rangle$ )
    with  $\langle t\text{-}P1 \in \mathcal{D} \ P \rangle$  have  $\langle t\text{-}P \in \mathcal{D} \ P \rangle$ 
    by (metis ** (1) F-imp-front-tickFree prefixE  $\langle tF \ t\text{-}P1 \rangle$  front-tickFree-append-iff
      is-processT7 tickFree-Nil tickFree-imp-front-tickFree)
    with  $\langle t\text{-}P \notin \mathcal{D} \ P \rangle$  show False ..
  qed
}
ultimately have $ :  $\langle (?map \ t\text{-}P, ?fun - ' X\text{-}P \cup \{ev \ a \mid a. a \notin \text{range } f\}) \in \mathcal{F} \$ 
 $(?R \ P) \rangle$ 
  using is-processT5-S7' by blast
from ** (2) have  $\langle (?map \ t\text{-}Q, ?fun - ' X\text{-}Q) \in \mathcal{F} \ (?R \ Q) \rangle$ 
  by (subst (asm) *** (2) [symmetric]) (auto simp add: F-Renaming)
moreover {

```

```

    fix a assume ⟨?map t-Q @ [ev a] ∈ T (?R Q)⟩ ⟨a ∉ range f⟩
    then consider t-Q1 where ⟨?map t-Q @ [ev a] = ?map t-Q1⟩ ⟨t-Q1 ∈ T
Q⟩
    | t-Q1 t-Q2 where ⟨?map t-Q @ [ev a] = ?map t-Q1 @ t-Q2⟩ ⟨tF t-Q1⟩
⟨t-Q1 ∈ D Q⟩
    unfolding T-Renaming by blast
    hence False
    proof cases
    from ⟨a ∉ range f⟩ show ⟨?map t-Q @ [ev a] = ?map t-Q1 ⟹ False⟩ for
t-Q1
    by (auto simp add: append-eq-map-conv ev-eq-map-eventptick-iff)
    next
    fix t-Q1 t-Q2 assume ⟨?map t-Q @ [ev a] = ?map t-Q1 @ t-Q2⟩ ⟨tF t-Q1⟩
⟨t-Q1 ∈ D Q⟩
    from this(1) ⟨a ∉ range f⟩ have ⟨t-Q1 ≤ t-Q⟩
    by (cases t-Q2 rule: rev-cases, auto simp add: append-eq-map-conv
ev-eq-map-eventptick-iff)
    (metis prefixI eventptick.inj-map inj-map-eq-map inj-on-id map-eq-append-conv
⟨inj f⟩)
    with ⟨t-Q1 ∈ D Q⟩ have ⟨t-Q ∈ D Q⟩
    by (metis *(2) F-imp-front-tickFree prefixE ⟨tF t-Q1⟩ front-tickFree-append-iff
is-processT7 tickFree-Nil tickFree-imp-front-tickFree)
    with ⟨t-Q ∉ D Q⟩ show False ..
    qed
  }
  ultimately have $$ : ⟨(?map t-Q, ?fun ‘ X-Q ∪ {ev a | a. a ∉ range f}) ∈ F
(?R Q)⟩
  using is-processT5-S7' by blast
  from *(3) have $$$ : ⟨t setinterleaves✓(⊗✓) ((?map t-P, ?map t-Q), f ‘ S)⟩
  by (simp add: ⟨t = ?map t'⟩ setinterleavesptick-inj-map-map-eventptick-iff-weak
⟨inj f⟩)
  have ⟨e ∈ X ⟹ e ∈ super-ref-Syncptick (⊗✓) (?fun ‘ X-P ∪ {ev a | a. a ∉
range f}) (f ‘ S)⟩
  ( ?fun ‘ X-Q ∪ {ev a | a. a ∉ range f}) by blast
  using *(4)[THEN set-mp, of ⟨map-eventptick (inv f) id e⟩] ⟨inj f⟩
  unfolding super-ref-Syncptick-def
  by (cases e, simp-all add: image-iff tick-eq-map-eventptick-iff) force
  hence $$$ : ⟨X ⊆ super-ref-Syncptick (⊗✓) (?fun ‘ X-P ∪ {ev a | a. a ∉ range
f}) (f ‘ S)⟩
  ( ?fun ‘ X-Q ∪ {ev a | a. a ∉ range f}) by blast
  from $ $$$ $$$ show ⟨(t, X) ∈ F ?rhs⟩ unfolding F-Syncptick by fast
next
fix t X assume ⟨(t, X) ∈ F ?rhs⟩ ⟨t ∉ D ?rhs⟩
then obtain t-P t-Q X-P X-Q
  where * : ⟨(t-P, X-P) ∈ F (?R P)⟩ ⟨(t-Q, X-Q) ∈ F (?R Q)⟩
  ⟨t setinterleaves✓(⊗✓) ((t-P, t-Q), f ‘ S)⟩
  ⟨X ⊆ super-ref-Syncptick (⊗✓) X-P (f ‘ S) X-Q⟩
  unfolding Syncptick-projs by blast
  have ⟨¬ (t-P ∈ D (?R P) ∨ t-Q ∈ D (?R Q))⟩

```

```

proof (rule notI)
  assume  $\langle t-P \in \mathcal{D} \ (\ ?R \ P) \vee t-Q \in \mathcal{D} \ (\ ?R \ Q) \rangle$ 
  hence  $\langle t \in \mathcal{D} \ ?rhs \rangle$ 
  by (simp add: D-Syncptick')
    (metis  $\ast(1-3)$  F-T append-Nil2 front-tickFree-Nil)
  with  $\langle t \notin \mathcal{D} \ ?rhs \rangle$  show False ..
qed
with  $\ast(1, 2)$  obtain  $t-P' \ t-Q'$ 
  where  $\ast\ast : \langle t-P = ?map \ t-P' \rangle \langle (t-P', ?fun - 'X-P) \in \mathcal{F} \ P \rangle$ 
     $\langle t-Q = ?map \ t-Q' \rangle \langle (t-Q', ?fun - 'X-Q) \in \mathcal{F} \ Q \rangle$ 
  unfolding Renaming-projs by blast
from setinterleavesptick-inj-map-map-eventptick-iff-strong
  [THEN iffD1, OF  $\langle inj \ f \rangle \ast(3)[unfolding \ \ast(1, 3)]$ ] obtain  $t'$ 
  where  $\ast\ast\ast : \langle t = ?map \ t' \rangle \langle t' \ setinterleaves_{\checkmark(\otimes\checkmark)} ((t-P', t-Q'), S) \rangle$  by blast
  have  $\langle e \in ?fun - 'X \implies e \in super-ref-Sync_{ptick} (\otimes\checkmark) (?fun - 'X-P) \ S \ (\ ?fun - 'X-Q) \rangle$  for  $e$ 
    using  $\ast(4)[THEN \ set-mp, \ of \ \langle ?fun \ e \rangle]$ 
    by (cases  $e$ ) (auto simp add: super-ref-Syncptick-def dest: injD[OF  $\langle inj \ f \rangle$ ])
  hence  $\langle ?fun - 'X \subseteq super-ref-Sync_{ptick} (\otimes\checkmark) (?fun - 'X-P) \ S \ (\ ?fun - 'X-Q) \rangle$ 
by blast
  with  $\ast\ast(2, 4) \ \ast\ast\ast(2)$  have  $\langle (t', ?fun - 'X) \in \mathcal{F} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ 
    unfolding F-Syncptick by auto
  thus  $\langle (t, X) \in \mathcal{F} \ ?lhs \rangle$  by (auto simp add:  $\ast\ast\ast(1)$  F-Renaming)
qed
qed

```

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

```

define  $Q :: \langle nat \Rightarrow ('a, 'r) \ process_{ptick} \rangle$ 
  where  $\langle Q \ r \equiv (((\rightarrow) \ undefined) \ \widetilde{\sim} \ r) \ STOP \rangle$  for  $r$ 
have  $\langle SKIPS \ UNIV \setminus \{undefined\} = (SKIPS \ UNIV :: ('a, nat) \ process_{ptick}) \rangle$ 
  by (simp add: Hiding-SKIPS)
moreover have  $\langle Q \ r \setminus \{undefined\} = STOP \rangle$  for  $r$ 
  by (induct  $r$ ) (simp-all add: Q-def Hiding-write0-non-disjoint)
ultimately have  $\ast : \langle (SKIPS \ UNIV \setminus \{undefined\}) ;_{\checkmark} (\lambda r. Q \ r \setminus \{undefined\}) \rangle$ 
   $= STOP \rangle$ 
  by (simp only: SKIPS-Segptick) simp

```

have $\langle SKIPS \ UNIV ;_{\checkmark} Q = \sqcap r \in UNIV. Q \ r \rangle$ **by** simp

moreover have $\langle [] \in \mathcal{D} \ (\dots \setminus \{undefined\}) \rangle$

proof (rule D-Hiding-seqRunI)

show $\langle ftF \ [] \ \langle tF \ [] \ \langle [] = trace-hide \ [] \ (ev \ ' \{undefined\}) \ @ \ [] \rangle \rangle$ **by** simp-all

```

next
{ fix r
  have ⟨replicate r (ev undefined) ∈ T (Q r)⟩
    by (induct r) (simp-all add: Q-def T-write0)
  also have ⟨replicate r (ev undefined) = map (λi. ev undefined) [0..i] ∈ T (Q r)⟩ for i by blast
thus ⟨[] ∈ D (⊓ r ∈ UNIV. Q r) ∨ (∃ x. isInfHidden-seqRun x (⊓ r ∈ UNIV. Q
r) {undefined} [])⟩
  by (auto simp add: T-GlobalNdet)
qed
ultimately have **: ⟨(SKIPS UNIV ;✓ Q) \ {undefined} = ⊥⟩
  by (simp add: BOT-iff-Nil-D)

have ⟨(SKIPS UNIV \ {undefined}) ;✓ (λr. Q r \ {undefined}) ≠ (SKIPS UNIV
;✓ Q) \ {undefined}⟩
  unfolding * ** by simp

hence ⟨∃ P (Q :: nat ⇒ ('a, 'r) processptick) S.
  (P \ S) ;✓ (λr. Q r \ S) ≠ (P ;✓ Q) \ S⟩ by blast

```

end

In general, only one refinement is holding.

```

theorem Hiding-Seq-FD-Seq-Hiding :
  ⟨(P ;✓ Q) \ S ⊆FD (P \ S) ;✓ (λr. Q r \ S)⟩ (is ⟨?lhs ⊆FD ?rhs⟩)
proof (rule failure-divergence-refine-optimizedI)
  let ?th = ⟨λt. trace-hide t (ev ' S)⟩ and ?map = ⟨λt. map (ev ∘ of-ev) t⟩
  fix t assume ⟨t ∈ D ?rhs⟩
  with D-imp-front-tickFree is-processT9
  consider (D-P) u v where ⟨t = ?map u @ v⟩ ⟨u ∈ D (P \ S)⟩ ⟨tF u⟩ ⟨ftF v⟩
    | (D-Q) u r v where ⟨t = ?map u @ v⟩ ⟨u @ [✓(r)] ∈ T (P \ S)⟩
      ⟨u @ [✓(r)] ∉ D (P \ S)⟩ ⟨tF u⟩ ⟨v ∈ D (Q r \ S)⟩
    by (fastforce simp add: Seqptick-projs)
  thus ⟨t ∈ D ?lhs⟩
proof cases
  case D-P
  from D-P(2) obtain u' v' x where * : ⟨u = ?th u' @ v'⟩ ⟨tF u'⟩ ⟨ftF v'⟩
    ⟨u' ∈ D P ∨ isInfHidden-seqRun x P S u'⟩
    by (blast elim: D-Hiding-seqRunE)
  from *(4) have ⟨?th (?map u') ∈ D ?lhs⟩
  proof (elim disjE)
    assume ⟨u' ∈ D P⟩
    with *(2) have ⟨?map u' ∈ D (P ;✓ Q)⟩
      by (auto simp add: Seqptick-projs intro: front-tickFree-Nil)
    with mem-D-imp-mem-D-Hiding show ⟨?th (?map u') ∈ D ?lhs⟩ .
  next

```

```

assume  $\langle \text{isInfHidden-seqRun } x \ P \ S \ u' \rangle$ 
from  $\text{this isInfHidden-seqRun-imp-tickFree-seqRun}[OF \ \text{this}]$ 
have  $\langle \text{isInfHidden-seqRun } (ev \circ of\text{-}ev \circ x) \ (P ; \checkmark \ Q) \ S \ (\text{?map } u') \rangle$ 
  by ( $\text{simp add: Seq}_{ptick}\text{-projs image-iff}$ )
  ( $\text{metis event}_{ptick}.\text{sel}(1) \ \text{list.map-comp map-append seqRun-def}$ )
thus  $\langle \text{?th } (\text{?map } u') \in \mathcal{D} \ \text{?lhs} \rangle$ 
  by ( $\text{simp add: D-Hiding-seqRun}$ )
  ( $\text{metis (no-types) append.right-neutral comp-apply}$ 
     $\text{front-tickFree-Nil tickFree-map-ev-comp}$ )
qed
also have  $\langle \text{?th } (\text{?map } u') = \text{?map } (\text{?th } u') \rangle$ 
  by ( $\text{fact tickFree-trace-hide-map-ev-comp-of-ev}[OF \ \langle tF \ u' \rangle]$ )
finally show  $\langle t \in \mathcal{D} \ \text{?lhs} \rangle$ 
  by ( $\text{simp add: D-P}(1, 4) \ * (1) \ \text{front-tickFree-append is-process} T \gamma$ )
next
case  $D\text{-}Q$ 
from  $D\text{-}Q(2, 3)$  obtain  $u'$  where  $\langle u \ @ \ [\checkmark(r)] = \text{?th } u' \rangle \langle (u', ev \text{ ' } S) \in \mathcal{F} \ P \rangle$ 
  unfolding  $T\text{-Hiding } D\text{-Hiding}$  by  $\text{fast}$ 
then obtain  $u'$  where  $\langle u = \text{?th } u' \rangle \langle (u' \ @ \ [\checkmark(r)], ev \text{ ' } S) \in \mathcal{F} \ P \rangle$ 
  by ( $\text{cases } u' \ \text{rule: rev-cases, simp-all split: if-split-asm}$ )
  ( $\text{metis F-imp-front-tickFree Hiding-tickFree butlast-snoc}$ 
     $\text{front-tickFree-iff-tickFree-butlast non-tickFree-tick tickFree-append-iff}$ )
from  $D\text{-}Q(5)$  obtain  $v' \ w' \ x$  where  $\ast : \langle v = \text{?th } v' \ @ \ w' \rangle \langle tF \ v' \rangle \langle ftF \ w' \rangle$ 
   $\langle v' \in \mathcal{D} \ (Q \ r) \vee \text{isInfHidden-seqRun } x \ (Q \ r) \ S \ v' \rangle$ 
  by ( $\text{blast elim: D-Hiding-seqRunE}$ )
from  $\ast(4)$  have  $\langle \text{?th } (\text{?map } u' \ @ \ v') \in \mathcal{D} \ \text{?lhs} \rangle$ 
proof ( $\text{elim disjE}$ )
  assume  $\langle v' \in \mathcal{D} \ (Q \ r) \rangle$ 
  hence  $\langle \text{?map } u' \ @ \ v' \in \mathcal{D} \ (P ; \checkmark \ Q) \rangle$ 
  by ( $\text{simp add: Seq}_{ptick}\text{-projs}$ )
  ( $\text{metis F-T } \langle (u' \ @ \ [\checkmark(r)], ev \text{ ' } S) \in \mathcal{F} \ P \rangle \ \text{append-T-imp-tickFree}$ 
     $\text{not-Cons-self}$ )
  with  $\text{mem-D-imp-mem-D-Hiding}$  show  $\langle \text{?th } (\text{?map } u' \ @ \ v') \in \mathcal{D} \ \text{?lhs} \rangle$  .
next
assume  $\langle \text{isInfHidden-seqRun } x \ (Q \ r) \ S \ v' \rangle$ 
hence  $\langle \text{isInfHidden-seqRun } x \ (P ; \checkmark \ Q) \ S \ (\text{?map } u' \ @ \ v') \rangle$ 
  by ( $\text{simp add: seqRun-def image-iff Seq}_{ptick}\text{-projs}$ )
  ( $\text{metis F-T } \langle (u' \ @ \ [\checkmark(r)], ev \text{ ' } S) \in \mathcal{F} \ P \rangle \ \text{append-T-imp-tickFree list.discI}$ )
thus  $\langle \text{?th } (\text{?map } u' \ @ \ v') \in \mathcal{D} \ \text{?lhs} \rangle$ 
  by ( $\text{simp add: D-Hiding-seqRun}$ )
  ( $\text{metis append.right-neutral filter-append}$ 
     $\text{front-tickFree-Nil isInfHidden-seqRun-imp-tickFree}$ )
qed
also have  $\langle \text{?th } (\text{?map } u' \ @ \ v') = \text{?map } (\text{?th } u') \ @ \ \text{?th } v' \rangle$ 
using  $D\text{-}Q(4) \ \text{Hiding-tickFree } \langle u = \text{?th } u' \rangle \ \text{tickFree-trace-hide-map-ev-comp-of-ev}$ 
by  $\text{auto}$ 
finally have  $\langle \text{?map } (\text{?th } u') \ @ \ \text{?th } v' \in \mathcal{D} \ \text{?lhs} \rangle$  .
moreover have  $\langle tF \ (\text{?map } (\text{?th } u') \ @ \ \text{?th } v') \rangle$ 
  by ( $\text{simp add: } \ast(2) \ \text{Hiding-tickFree}$ )

```

```

ultimately show  $\langle t \in \mathcal{D} \text{ ?lhs} \rangle$ 
  unfolding  $\ast(1)$   $D\text{-}Q(1)$   $\langle u = \text{?th } u' \rangle$  using  $\langle \text{ftF } w' \rangle$ 
  by (metis append.assoc is-processT7)
qed
next

assume subset-div :  $\langle \mathcal{D} \text{ ?rhs} \subseteq \mathcal{D} \text{ ?lhs} \rangle$ 
let  $\text{?th} = \langle \lambda t. \text{trace-hide } t \text{ (ev ' } S) \rangle$  and  $\text{?map} = \langle \lambda t. \text{map (ev } \circ \text{ of-ev) } t \rangle$ 
fix  $t$   $X$  assume  $\langle (t, X) \in \mathcal{F} \text{ ?rhs} \rangle$ 
then consider (div)  $\langle t \in \mathcal{D} \text{ ?rhs} \rangle$ 
  |  $(F\text{-}P) \text{ } u$  where  $\langle t = \text{?map } u \rangle \langle (u, \text{ref-Seq}_{\text{ptick}} X) \in \mathcal{F} (P \setminus S) \rangle \langle u \notin \mathcal{D} (P \setminus S) \rangle \langle \text{tF } u \rangle$ 
  |  $(F\text{-}Q) \text{ } u \text{ } r \text{ } v$  where  $\langle t = \text{?map } u @ v \rangle \langle u @ [\checkmark(r)] \in \mathcal{T} (P \setminus S) \rangle \langle u @ [\checkmark(r)] \notin \mathcal{D} (P \setminus S) \rangle$ 
     $\langle \text{tF } u \rangle \langle (v, X) \in \mathcal{F} (Q \text{ } r \setminus S) \rangle \langle v \notin \mathcal{D} (Q \text{ } r \setminus S) \rangle$ 
    by (simp add: Seqptick-projs)
    (metis F-T T-imp-front-tickFree front-tickFree-Nil is-processT9 self-append-conv)
  thus  $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
proof cases
  case div with subset-div show  $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: in-mono is-processT8)
next
case F-P
  from  $F\text{-}P(2, 3)$  obtain  $u'$  where  $\ast : \langle u = \text{?th } u' \rangle \langle (u', \text{ref-Seq}_{\text{ptick}} X \cup \text{ev ' } S) \in \mathcal{F} P \rangle$ 
  unfolding F-Hiding D-Hiding by fast
  have  $\langle \text{tF } u' \rangle$  using  $\ast(1)$   $F\text{-}P(4)$  Hiding-tickFree by blast
  have  $\$ : \langle \text{ref-Seq}_{\text{ptick}} (X \cup \text{ev ' } S) = \text{ref-Seq}_{\text{ptick}} X \cup \text{ev ' } S \rangle$ 
    by (auto simp add: image-iff ref-Seqptick-def)
    (metis Int-iff Un-iff eventptick.sel(1) image-eqI rangeI)
  from  $\langle \text{tF } u' \rangle \ast(2)$  have  $\langle (\text{?map } u', X \cup \text{ev ' } S) \in \mathcal{F} (P ; \checkmark Q) \rangle$ 
    by (auto simp add: Seqptick-projs  $\$$ )
  thus  $\langle (t, X) \in \mathcal{F} \text{ ?lhs} \rangle$ 
    by (simp add: F-Hiding)
    (metis  $\ast(1)$   $F\text{-}P(1)$   $\langle \text{tF } u' \rangle$  tickFree-trace-hide-map-ev-comp-of-ev)
next
case F-Q
  from  $F\text{-}Q(2, 3)$  obtain  $u'$  where  $\langle u @ [\checkmark(r)] = \text{?th } u' \rangle \langle (u', \text{ev ' } S) \in \mathcal{F} P \rangle$ 
  unfolding T-Hiding D-Hiding by fast
  then obtain  $u'$  where  $\ast : \langle u = \text{?th } u' \rangle \langle (u' @ [\checkmark(r)], \text{ev ' } S) \in \mathcal{F} P \rangle$ 
    by (cases u' rule: rev-cases, simp-all split: if-split-asm)
    (metis F-imp-front-tickFree Hiding-tickFree butlast-snoc
      front-tickFree-iff-tickFree-butlast non-tickFree-tick tickFree-append-iff)
  from  $F\text{-}Q(5, 6)$  obtain  $v'$  where  $\ast\ast : \langle v = \text{?th } v' \rangle \langle (v', X \cup \text{ev ' } S) \in \mathcal{F} (Q \text{ } r) \rangle$ 
  unfolding F-Hiding D-Hiding by blast
  have  $\langle (\text{?map } u' @ v', X \cup \text{ev ' } S) \in \mathcal{F} (P ; \checkmark Q) \rangle$ 
    by (simp add: Seqptick-projs)
    (metis  $\ast(2) \ast\ast(2)$  F-T append-T-imp-tickFree list.distinct(1))

```



```

with F-Q(4) show ⟨(t, X) ∈ F ?lhs⟩
by (simp add: F-Hiding F-Q(1) *(1) **(1))
  (metis Hiding-tickFree filter-append tickFree-trace-hide-map-ev-comp-of-ev)
qed
qed

```

16.4 Hiding and Synchronization Product

```

lemma setinterleavesptick-imp-superset-ev :
  ⟨t setinterleaves✓tick-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 (induct t arbitrary: u v)
  case Nil thus ?case by (auto dest: Nil-setinterleavesptick)
next
  case (Cons e t)
  from Cons.premis consider (mv-left) a u' where ⟨e = ev a⟩ ⟨u = ev a # u'⟩
  | (mv-right) a v' where ⟨e = ev a⟩ ⟨v = ev a # v'⟩
  | (mv-both-ev) a u' v' where ⟨e = ev a⟩ ⟨u = ev a # u'⟩ ⟨v = ev a # v'⟩
  | (mv-both-tick) r s u' v' where ⟨u = ✓(r) # u'⟩ ⟨v = ✓(s) # v'⟩
  | (mv-both-tick) r s u' v' where ⟨u = ✓(r) # u'⟩ ⟨v = ✓(s) # v'⟩
  | (mv-both-tick) r s u' v' where ⟨u = ✓(r) # u'⟩ ⟨v = ✓(s) # v'⟩
  | (mv-both-tick) r s u' v' where ⟨u = ✓(r) # u'⟩ ⟨v = ✓(s) # v'⟩
  | (mv-both-tick) r s u' v' where ⟨u = ✓(r) # u'⟩ ⟨v = ✓(s) # v'⟩
  by (cases e) (auto elim: Cons-ev-setinterleavesptickE Cons-tick-setinterleavesptickE)
  thus ?case by cases (auto dest!: Cons.hyps)
qed

```

```

lemma (in Syncptick-locale) disjoint-isInfHidden-seqRunL-to-Syncptick :
  assumes ⟨A ∩ S = {}⟩ and ⟨isInfHidden-seqRun x P A t-P⟩
  and ⟨t-Q ∈ T Q⟩ and ⟨t setinterleaves✓(⊗✓) ((t-P, t-Q), S)⟩
  shows ⟨isInfHidden-seqRun (ev ∘ of-ev ∘ x) (P [S]✓ Q) A t⟩
proof -
  have tF-x : ⟨tF (map x [0..i])⟩ for i
  by (metis assms(2) imageE is-ev-def seqRun-def tickFree-append-iff
    tickFree-map-tick-comp-iff tickFree-seqRun-iff)
  define t' where ⟨t' i ≡ t @ map (ev ∘ of-ev) (map x [0..i])⟩ for i
  from assms(1, 2) have ⟨{a. ev a ∈ set (map x [0..i])} ∩ S = {}⟩ for i
  by (simp add: disjoint-iff image-iff) (metis eventptick.inject(1))
  from tickFree-disjoint-setinterleavesptick-append-tailL[OF tF-x this assms(4)]
  have ⟨seqRun t (ev ∘ of-ev ∘ x) i setinterleaves✓(⊗✓) ((seqRun t-P x i, t-Q), S)⟩
  for i
  by (simp add: seqRun-def)
  moreover have ⟨of-ev (x i) ∈ A⟩ for i
  by (metis assms(2) eventptick.sel(1) image-iff)
  ultimately show ⟨isInfHidden-seqRun (ev ∘ of-ev ∘ x) (P [S]✓ Q) A t⟩
  using assms(2, 3) by (auto simp add: T-Syncptick)
qed

```

lemma (in $\text{Sync}_{ptick}\text{-locale}$) *disjoint-isInfHidden-seqRunR-to-Sync_{ptick}* :
 $\langle [A \cap S = \{\}; \text{isInfHidden-seqRun } x \ Q \ A \ t\text{-}Q; t\text{-}P \in \mathcal{T} \ P; \\ t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), S) \rangle \implies \\ \text{isInfHidden-seqRun } (ev \circ of\text{-}ev \circ x) \ (P \llbracket S \rrbracket_{\checkmark} Q) \ A \ t \rangle$
by (fold $\text{Sync}_{ptick}\text{-sym}$, rule $\text{Sync}_{ptick}\text{-locale.disjoint-isInfHidden-seqRunL-to-Sync}_{ptick}$)
 (use $\text{setinterleaves}_{ptick}\text{-sym}$ in $\langle \text{blast intro: Sync}_{ptick}\text{-locale-sym.Sync}_{ptick}\text{-locale-axioms} \rangle$) +

lemma (in $\text{Sync}_{ptick}\text{-locale}$) *disjoint-Hiding-Sync_{ptick}-FD-Sync_{ptick}-Hiding-aux* :
 — This lemma avoids duplication of the proof work.

assumes $\langle A \cap S = \{\} \rangle \langle tF \ u \rangle \langle ftF \ v \rangle \langle t\text{-}P \in \mathcal{D} \ (P \setminus A) \rangle \langle t\text{-}Q \in \mathcal{T} \ (Q \setminus A) \rangle$
and $\ast : \langle u \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P, t\text{-}Q), S) \rangle$

shows $\langle u \ @ \ v \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$

proof —

let $?th\text{-}A = \langle \lambda t. \text{trace-hide } t \ (ev \text{ ' } A) \rangle$

from $\langle t\text{-}P \in \mathcal{D} \ (P \setminus A) \rangle$ **obtain** $t\text{-}P1 \ t\text{-}P2$

where $D\text{-}P : \langle tF \ t\text{-}P1 \rangle \langle ftF \ t\text{-}P2 \rangle \langle t\text{-}P = ?th\text{-}A \ t\text{-}P1 \ @ \ t\text{-}P2 \rangle$

$\langle t\text{-}P1 \in \mathcal{D} \ P \vee (\exists t\text{-}P\text{-}x. \text{isInfHidden-seqRun-strong } t\text{-}P\text{-}x \ P \ A \ t\text{-}P1) \rangle$

by (blast elim: $D\text{-}Hiding\text{-seqRunE}$)

from $\text{setinterleaves}_{ptick}\text{-appendL}[OF \ \ast[\text{unfolded } D\text{-}P(3)]]$ **obtain** $u1 \ u2 \ t\text{-}Q1 \ t\text{-}Q2$

where $\ast\ast : \langle u = u1 \ @ \ u2 \rangle \langle t\text{-}Q = t\text{-}Q1 \ @ \ t\text{-}Q2 \rangle$

$\langle u1 \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((?th\text{-}A \ t\text{-}P1, t\text{-}Q1), S) \rangle$

$\langle u2 \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P2, t\text{-}Q2), S) \rangle$ **by** blast

from $\langle t\text{-}Q \in \mathcal{T} \ (Q \setminus A) \rangle$ **consider** $t\text{-}Q1'$ **where** $\langle t\text{-}Q = ?th\text{-}A \ t\text{-}Q1' \rangle \langle (t\text{-}Q1', ev \text{ ' } A) \in \mathcal{F} \ Q \rangle$

$| \ (D\text{-}Q) \ t\text{-}Q1' \ t\text{-}Q2' \text{ where } \langle tF \ t\text{-}Q1' \rangle \langle ftF \ t\text{-}Q2' \rangle \langle t\text{-}Q = ?th\text{-}A \ t\text{-}Q1' \ @ \ t\text{-}Q2' \rangle$

$\langle t\text{-}Q1' \in \mathcal{D} \ Q \vee (\exists t\text{-}Q\text{-}x. \text{isInfHidden-seqRun-strong } t\text{-}Q\text{-}x \ Q \ A \ t\text{-}Q1') \rangle$

by (elim $T\text{-}Hiding\text{-seqRunE}$)

thus $\langle u \ @ \ v \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$

proof cases

fix $t\text{-}Q1'$ **assume** $\langle t\text{-}Q = ?th\text{-}A \ t\text{-}Q1' \rangle \langle (t\text{-}Q1', ev \text{ ' } A) \in \mathcal{F} \ Q \rangle$

from $\langle t\text{-}Q = ?th\text{-}A \ t\text{-}Q1' \rangle \ast\ast(2)$ **obtain** $t\text{-}Q1''$

where $\langle t\text{-}Q1 = ?th\text{-}A \ t\text{-}Q1'' \rangle \langle t\text{-}Q1'' \leq t\text{-}Q1' \rangle$

by (metis $\text{Prefix-Order.prefixI le-trace-hide}$)

from $F\text{-}T \ \langle (t\text{-}Q1', ev \text{ ' } A) \in \mathcal{F} \ Q \rangle \langle t\text{-}Q1'' \leq t\text{-}Q1' \rangle$ *is-processT3-TR*

have $\langle t\text{-}Q1'' \in \mathcal{T} \ Q \rangle$ **by** blast

from $\ast\ast(3)[\text{unfolded } \langle t\text{-}Q1 = ?th\text{-}A \ t\text{-}Q1'' \rangle,$

$\text{THEN disjoint-trace-hide-setinterleaves}_{ptick}[OF \ \langle A \cap S = \{\} \rangle]$

obtain $u1'$ **where** $\langle u1 = ?th\text{-}A \ u1' \rangle \langle u1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P1, t\text{-}Q1'), S) \rangle$ **by** blast

from $D\text{-}P(4)$ **show** $\langle u \ @ \ v \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$

proof (elim disjE exE)

assume $\langle t\text{-}P1 \in \mathcal{D} \ P \rangle$

with $\langle u1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t\text{-}P1, t\text{-}Q1''), S) \rangle$ $D\text{-}P(1)$ $\text{setinterleaves}_{ptick}\text{-tickFree-imp}$

have $\langle u1' = u1' \ @ \ [] \rangle \langle tF \ u1' \rangle \langle ftF \ [] \rangle$

$\langle u1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P1, t-Q1''), S) \rangle \langle t-P1 \in \mathcal{D} P \rangle$
 by *simp-all* (*blast intro: is-processT3-TR*) +
 with $\langle t-Q1'' \in \mathcal{T} Q \rangle$ have $\langle u1' \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **unfolding** *D-Sync_{ptick}*
 by *blast*
 moreover have $\langle u @ v = ?th-A \ u1' @ (u2 @ v) \rangle$
 by (*simp add: *(1) **(1) $\langle u1 = ?th-A \ u1' \rangle$*)
 ultimately show $\langle u @ v \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
 unfolding *D-Hiding* using $\langle tF \ u \rangle \langle ftF \ v \rangle ** (1) \langle tF \ u1' \rangle$
 by (*auto intro: front-tickFree-append*)
 next
 fix *t-P-x* assume $\langle isInfHidden-seqRun-strong \ t-P-x \ P \ A \ t-P1 \rangle$
 hence $\langle isInfHidden-seqRun \ (ev \circ of-ev \circ t-P-x) \ (P \llbracket S \rrbracket_{\checkmark} Q) \ A \ u1' \rangle$
 by (*intro disjoint-isInfHidden-seqRunL-to-Sync_{ptick}*
 $[OF \ \langle A \cap S = \{\} \rangle - \langle t-Q1'' \in \mathcal{T} Q \rangle \langle u1' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P1,$
 $t-Q1''), S) \rangle]$ *simp*
 with $** (1) \langle u1 = ?th-A \ u1' \rangle$ *assms*(2, 3) **show** $\langle u @ v \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus$
 $A) \rangle$
 unfolding *D-Hiding-seqRun* by *clarify*
 (*metis append-eq-append-conv2*[*of* $u1 \ \langle u2 @ v \rangle \langle u1 @ u2 \rangle v$]
isInfHidden-seqRun-imp-tickFree[*of* $u1' \ \langle ev \circ of-ev \circ t-P-x \rangle \langle P \llbracket S \rrbracket_{\checkmark} Q \rangle$
 A]
front-tickFree-append[*of* $u2 \ v$] *tickFree-append-iff*[*of* $u1 \ u2$])
 qed
 next
 case *D-Q*
 from *setinterleaves_{ptick}-le-prefixLR*
 $[OF \ *[unfolded \ D-P(3) \ D-Q(3)], of \ \langle ?th-A \ t-P1 \rangle \langle ?th-A \ t-Q1' \rangle]$
 consider (*left*) $u' \ t-Q1''$ where $\langle u' \leq u \rangle \langle t-Q1'' \leq t-Q1' \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((?th-A \ t-P1, ?th-A \ t-Q1''), S) \rangle$
 | (*right*) $u' \ t-P1'$ where $\langle u' \leq u \rangle \langle t-P1' \leq t-P1 \rangle$
 $\langle u' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((?th-A \ t-P1', ?th-A \ t-Q1'), S) \rangle$
 by (*auto dest!: le-trace-hide*)
 thus $\langle u @ v \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
 proof *cases*
 case *left*
 have $\langle t-Q1'' \in \mathcal{T} Q \rangle$ by (*meson D-Q(4) D-T is-processT3-TR left(2)*
t-le-seqRun)
 from *disjoint-trace-hide-setinterleaves_{ptick}*[*OF* $\langle A \cap S = \{\} \rangle$ *left(3)*]
 obtain u'' where $\$: \langle u' = ?th-A \ u'' \rangle$
 $\langle u'' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P1, t-Q1''), S) \rangle$ by *blast*
 from *D-P(4)* show $\langle u @ v \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
 proof (*elim disjE exE*)
 assume $\langle t-P1 \in \mathcal{D} P \rangle$
 hence $\langle u'' \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 by (*simp add: D-Sync_{ptick}*)
 (*metis* $\$(2) \ D-P(1) \ \langle t-Q1'' \in \mathcal{T} Q \rangle$ *append.right-neutral*
front-tickFree-Nil setinterleaves_{ptick}-tickFree-imp)
 with *left(1)* show $\langle u @ v \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$

```

    by (elim Prefix-Order.prefixE, simp add: D-Hiding $(1))
    (metis Hiding-tickFree assms(2, 3) front-tickFree-append tickFree-append-iff)
next
  fix t-P-x assume <isInfHidden-seqRun-strong t-P-x P A t-P1>
  hence <isInfHidden-seqRun (ev ∘ of-ev ∘ t-P-x) (P [S]✓ Q) A u''>
    by (intro disjoint-isInfHidden-seqRunL-to-Syncptick
      [OF <A ∩ S = {}> - <t-Q1'' ∈ T Q> $(2)]) simp
  from left(1) show <u @ v ∈ D (P [S]✓ Q \ A)>
    by (elim Prefix-Order.prefixE, simp add: D-Hiding-seqRun $(1))
    (metis <?this> assms(2, 3) front-tickFree-append
      isInfHidden-seqRun-imp-tickFree tickFree-append-iff)
qed
next
  case right
    have <t-P1' ∈ T P> by (meson D-P(4) D-T is-processT3-TR right(2)
t-le-seqRun)
  from disjoint-trace-hide-setinterleavesptick[OF <A ∩ S = {}> right(3)]
  obtain u'' where $ : <u' = ?th-A u''>
    <u'' setinterleaves✓(⊗✓) ((t-P1', t-Q1'), S)> by blast
  from D-Q(4) show <u @ v ∈ D (P [S]✓ Q \ A)>
  proof (elim disjE exE)
    assume <t-Q1' ∈ D Q>
    hence <u'' ∈ D (P [S]✓ Q)>
      by (simp add: D-Syncptick)
      (metis $(2) D-Q(1) <t-P1' ∈ T P> append.right-neutral
        front-tickFree-Nil setinterleavesptick-tickFree-imp)
    with right(1) show <u @ v ∈ D (P [S]✓ Q \ A)>
      by (elim Prefix-Order.prefixE, simp add: D-Hiding $(1))
      (metis Hiding-tickFree assms(2, 3) front-tickFree-append tickFree-append-iff)
  next
    fix t-Q-x assume <isInfHidden-seqRun-strong t-Q-x Q A t-Q1'>
    hence <isInfHidden-seqRun (ev ∘ of-ev ∘ t-Q-x) (P [S]✓ Q) A u''>
      by (intro disjoint-isInfHidden-seqRunR-to-Syncptick
        [OF <A ∩ S = {}> - <t-P1' ∈ T P> $(2)]) simp
    from right(1) show <u @ v ∈ D (P [S]✓ Q \ A)>
      by (elim Prefix-Order.prefixE, simp add: D-Hiding-seqRun $(1))
      (metis <?this> assms(2, 3) front-tickFree-append
        isInfHidden-seqRun-imp-tickFree tickFree-append-iff)
    qed
  qed
qed
qed

```

theorem (in Sync_{ptick}-locale) disjoint-Hiding-Sync_{ptick}-FD-Sync_{ptick}-Hiding :
 <P [S]✓ Q \ A ⊆_{FD} (P \ A) [S]✓ (Q \ A)> if <A ∩ S = {}>
proof (rule failure-divergence-refine-optimizedI)
 let ?th-A = <λt. trace-hide t (ev 'A)>

fix t **assume** $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
from *this* **obtain** $u \ v \ t\text{-}P \ t\text{-}Q$
where $*$: $\langle t = u \ @ \ v \rangle \langle tF \ u \rangle \langle ftF \ v \rangle$
 $\langle u \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle$
 $\langle t\text{-}P \in \mathcal{D} (P \setminus A) \wedge t\text{-}Q \in \mathcal{T} (Q \setminus A) \vee t\text{-}P \in \mathcal{T} (P \setminus A) \wedge t\text{-}Q \in \mathcal{D} (Q \setminus A) \rangle$
unfolding $D\text{-}Sync_{ptick}$ **by** *blast*
from $*(5)$ **show** $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
proof (*elim disjE conjE*)
show $\langle t\text{-}P \in \mathcal{D} (P \setminus A) \implies t\text{-}Q \in \mathcal{T} (Q \setminus A) \implies t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
by (*simp add: *(1-4) disjoint-Hiding-Sync_{ptick}-FD-Sync_{ptick}-Hiding-aux* $\langle A \cap S = \{\} \rangle$)
next
assume $\langle t\text{-}P \in \mathcal{T} (P \setminus A) \rangle \langle t\text{-}Q \in \mathcal{D} (Q \setminus A) \rangle$
have $\langle u \text{ setinterleaves}_{\checkmark} \lambda s \ r. \ r \otimes \checkmark \ s \ ((t\text{-}Q, t\text{-}P), S) \rangle$
using $*(4)$ *setinterleaves_{ptick}-sym* **by** *blast*
from *Sync_{ptick}-locale-sym.disjoint-Hiding-Sync_{ptick}-FD-Sync_{ptick}-Hiding-aux*
 $[OF \ \langle A \cap S = \{\} \rangle \ *(2, 3) \ \langle t\text{-}Q \in \mathcal{D} (Q \setminus A) \rangle \langle t\text{-}P \in \mathcal{T} (P \setminus A) \rangle \text{ this}]$
have $\langle u \ @ \ v \in \mathcal{D} (Sync_{ptick}\text{-locale-sym}.Sync_{ptick} \ Q \ S \ P \setminus A) \rangle$.
also have $\langle Sync_{ptick}\text{-locale-sym}.Sync_{ptick} \ Q \ S \ P = P \llbracket S \rrbracket_{\checkmark} Q \rangle$ **by** (*fact Sync_{ptick}-sym*)
finally show $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$ **unfolding** $*(1)$.
qed
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
and *subset-div* : $\langle \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \subseteq \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
from *this* (1) **consider** $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
 $| \text{ (fail-Sync) } t\text{-}P \ t\text{-}Q \ X\text{-}P \ X\text{-}Q$ **where** $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F} (P \setminus A) \rangle \langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F} (Q \setminus A) \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t\text{-}P, t\text{-}Q), S) \rangle \langle X \subseteq \text{super-ref-Sync}_{ptick} (\otimes \checkmark) X\text{-}P \ S \ X\text{-}Q \rangle$
unfolding *Sync_{ptick}-projs* **by** *blast*
thus $\langle (t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
proof *cases*
from *subset-div* **show** $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \implies (t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
by (*simp add: in-mono is-processT8*)
next
case *fail-Sync*
from *fail-Sync* $(1, 2)$ **consider** $\langle t\text{-}P \in \mathcal{D} (P \setminus A) \vee t\text{-}Q \in \mathcal{D} (Q \setminus A) \rangle$
 $| \text{ (fail-Hiding) } t\text{-}P' \ t\text{-}Q'$ **where**
 $\langle t\text{-}P = \text{trace-hide } t\text{-}P' \ (ev \ 'A) \rangle \langle (t\text{-}P', X\text{-}P \cup ev \ 'A) \in \mathcal{F} \ P \rangle$
 $\langle t\text{-}Q = \text{trace-hide } t\text{-}Q' \ (ev \ 'A) \rangle \langle (t\text{-}Q', X\text{-}Q \cup ev \ 'A) \in \mathcal{F} \ Q \rangle$
unfolding *F-Hiding D-Hiding* **by** *blast*
thus $\langle (t, X) \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
proof *cases*
assume $\langle t\text{-}P \in \mathcal{D} (P \setminus A) \vee t\text{-}Q \in \mathcal{D} (Q \setminus A) \rangle$
with *fail-Sync* $(1-3)$ **have** $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
by (*simp add: D-Sync_{ptick}'*)

```

      (metis F-T append-self-conv front-tickFree-Nil)
    with subset-div show  $\langle t, X \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A)$ 
      by (simp add: in-mono is-processT8)
  next
    case fail-Hiding
    from disjoint-trace-hide-setinterleavesptick
      [OF  $\langle A \cap S = \{\} \rangle$  fail-Sync(3)[unfolded fail-Hiding(1, 3)]]
    obtain  $t'$  where  $*$  :  $\langle t = \text{trace-hide } t' (ev \text{ ` } A) \rangle$ 
       $\langle t' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P', t-Q'), S) \rangle$  by blast
    from fail-Sync(4) have  $\langle X \cup ev \text{ ` } A \subseteq \text{super-ref-Sync}_{ptick} (\otimes\checkmark) (X-P \cup ev \text{ ` } A) S (X-Q \cup ev \text{ ` } A) \rangle$ 
      by (auto simp add: super-ref-Syncptick-def image-iff)
    with *(2) fail-Hiding(2, 4) have  $\langle t', X \cup ev \text{ ` } A \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q)$ 
      by (auto simp add: F-Syncptick)
    with *(1) show  $\langle t, X \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A)$  unfolding F-Hiding by blast
  qed
qed
qed

```

theorem (in Sync_{ptick}-locale) disjoint-finite-Hiding-Sync_{ptick} :

$\langle P \llbracket S \rrbracket_{\checkmark} Q \setminus A = (P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A) \rangle$ if $\langle A \cap S = \{\} \rangle$ and $\langle \text{finite } A \rangle$
 — Monster theorem!

proof (rule FD-antisym)

from disjoint-Hiding-Sync_{ptick}-FD-Sync_{ptick}-Hiding[OF $\langle A \cap S = \{\} \rangle$]

show $\langle P \llbracket S \rrbracket_{\checkmark} Q \setminus A \sqsubseteq_{FD} (P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A) \rangle$.

next

let ?th-A = $\langle \lambda t. \text{trace-hide } t (ev \text{ ` } A) \rangle$

show $\langle (P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A) \sqsubseteq_{FD} P \llbracket S \rrbracket_{\checkmark} Q \setminus A \rangle$

proof (rule failure-divergence-refine-optimizedI)

fix $t \ X$ assume $\langle t, X \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A)$

and subset-div : $\langle \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \subseteq \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$

from this(1) consider $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$

| t' where $\langle t = ?th-A \ t' \rangle \langle t', X \cup ev \text{ ` } A \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q)$

unfolding F-Hiding D-Hiding by blast

thus $\langle t, X \rangle \in \mathcal{F} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A))$

proof cases

show $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \implies \langle t, X \rangle \in \mathcal{F} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$

using subset-div is-processT8 by blast

next

fix t' assume $\langle t = ?th-A \ t' \rangle \langle t', X \cup ev \text{ ` } A \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q)$

from this(2) consider $\langle t' \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$

| (fail) $t-P \ X-P \ t-Q \ X-Q$ where $\langle t-P, X-P \rangle \in \mathcal{F} P \rangle \langle t-Q, X-Q \rangle \in \mathcal{F} Q \rangle$

$\langle t' \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((t-P, t-Q), S) \rangle$

$\langle X \cup ev \text{ ` } A \subseteq \text{super-ref-Sync}_{ptick} (\otimes\checkmark) X-P \ S \ X-Q \rangle$

unfolding Sync_{ptick}-projs by auto

thus $\langle t, X \rangle \in \mathcal{F} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A))$

proof cases

assume $\langle t' \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 with $\langle t = ?th-A \ t' \rangle$ have $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
 by (metis mem-D-imp-mem-D-Hiding)
 with subset-div is-processT8 show $\langle (t, X) \in \mathcal{F} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
 by blast
 next
 case fail
 from $\langle A \cap S = \{\} \rangle$ fail(4) have $\langle X-P = X-P \cup ev \text{ ` } A \rangle$ $\langle X-Q = X-Q \cup ev$
 $\text{ ` } A \rangle$
 — i.e. $ev \text{ ` } A \subseteq X-P$  and  $ev \text{ ` } A \subseteq X-Q$
 by (auto simp add: super-ref-Sync_{ptick}-def)
 with fail(1, 2) have $\langle (?th-A \ t-P, X-P) \in \mathcal{F} (P \setminus A) \rangle$
 $\langle (?th-A \ t-Q, X-Q) \in \mathcal{F} (Q \setminus A) \rangle$
 by (auto simp add: F-Hiding)
 moreover from fail(3) have $\langle t \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((?th-A \ t-P, ?th-A$
 $t-Q), S) \rangle$
 unfolding $\langle t = ?th-A \ t' \rangle$ by (fact setinterleaves_{ptick}-trace-hide)
 ultimately show $\langle (t, X) \in \mathcal{F} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
 using fail(4) unfolding F-Sync_{ptick} by fast
 qed
 qed
 next
 fix t assume $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$
 then obtain $u \ v$ where $*$: $\langle ftF \ v \rangle \langle tF \ u \rangle \langle t = ?th-A \ u \ @ \ v \rangle$
 $\langle u \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \vee (\exists x. \text{isInfHidden-seqRun-strong } x \ (P \llbracket S \rrbracket_{\checkmark} Q) \ A \ u) \rangle$
 by (blast elim: D-Hiding-seqRunE)
 from $*(4)$ show $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
 proof (elim disjE exE)
 assume $\langle u \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 then obtain $u1 \ u2 \ t-P \ t-Q$ where $**$: $\langle u = u1 \ @ \ u2 \rangle \langle tF \ u1 \rangle \langle ftF \ u2 \rangle$
 $\langle u1 \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((t-P, t-Q), S) \rangle$
 $\langle t-P \in \mathcal{D} \ P \wedge t-Q \in \mathcal{T} \ Q \vee t-P \in \mathcal{T} \ P \wedge t-Q \in \mathcal{D} \ Q \rangle$
 unfolding D-Sync_{ptick} by blast
 have $\langle t = ?th-A \ u1 \ @ \ (?th-A \ u2 \ @ \ v) \rangle$
 by (simp add: $*(3) \ ** (1)$)
 moreover from $**(2)$ have $\langle tF \ (?th-A \ u1) \rangle$ using Hiding-tickFree by blast
 moreover have $\langle ftF \ (?th-A \ u2 \ @ \ v) \rangle$
 by (metis D-imp-front-tickFree $\langle t \in \mathcal{D} (P \llbracket S \rrbracket_{\checkmark} Q \setminus A) \rangle$ calculation(1)
 front-tickFree-append-iff front-tickFree-charn)
 moreover from $**(4)$ have $\langle ?th-A \ u1 \text{ setinterleaves}_{\checkmark} (\otimes \checkmark) ((?th-A \ t-P, ?th-A$
 $t-Q), S) \rangle$
 by (fact setinterleaves_{ptick}-trace-hide)
 moreover from $**(5)$ have $\langle ?th-A \ t-P \in \mathcal{D} (P \setminus A) \wedge ?th-A \ t-Q \in \mathcal{T} (Q \setminus$
 $A) \vee$
 $?th-A \ t-P \in \mathcal{T} (P \setminus A) \wedge ?th-A \ t-Q \in \mathcal{D} (Q \setminus A) \rangle$
 by (metis mem-D-imp-mem-D-Hiding mem-T-imp-mem-T-Hiding)
 ultimately show $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
 unfolding D-Sync_{ptick} by blast
 next

```

fix x assume ** : ⟨isInfHidden-seqRun-strong x (P [S]✓ Q) A u⟩
from ** have *** : ⟨∃ t-P t-Q. t-P ∈ T P ∧ t-Q ∈ T Q ∧
  seqRun u x i setinterleaves✓(⊗✓) ((t-P, t-Q), S)⟩ for i
unfolding Syncptick-projs by blast

define t-P-t-Q where ⟨t-P-t-Q i ≡ SOME (t-P, t-Q). t-P ∈ T P ∧ t-Q ∈ T
Q ∧
  seqRun u x i setinterleaves✓(⊗✓) ((t-P, t-Q), S)⟩ for i
define t-P where ⟨t-P ≡ fst ∘ t-P-t-Q⟩
define t-Q where ⟨t-Q ≡ snd ∘ t-P-t-Q⟩
have **** : ⟨t-P i ∈ T P⟩ ⟨t-Q i ∈ T Q⟩
  ⟨seqRun u x i setinterleaves✓(⊗✓) ((t-P i, t-Q i), S)⟩ for i
by (use ***[of i] in ⟨simp add: t-P-def t-Q-def,
  cases ⟨t-P-t-Q i⟩, simp add: t-P-t-Q-def,
  metis (mono-tags, lifting) case-prod-conv someI-ex⟩)+
from *(2) ** have ⟨set (seqRun u x i) ⊆ {ev a | a. ev a ∈ set u} ∪ ev ‘A⟩
for i
  by (simp add: seqRun-def subset-iff)
  (metis image-iff list.set-map tickFree-iff-is-map-ev)
have ***** : ⟨{ev a | a. ev a ∈ set (t-P i)} ∪ {ev a | a. ev a ∈ set (t-Q i)} ⊆
  {ev a | a. ev a ∈ set u} ∪ ev ‘A⟩ for i
  by (rule subset-trans[OF setinterleavesptick-imp-superset-ev[OF ***** (3)]]
    (use ⟨set (seqRun u x i) ⊆ {ev a | a. ev a ∈ set u} ∪ ev ‘A⟩ in blast))
have ***** : ⟨tF (t-P i)⟩ ⟨tF (t-Q i)⟩ for i
  using tickFree-setinterleavesptick-iff[OF ***** (3)[of i]]
  by (metis *(2) ** eventptick.disc(1) imageE tickFree-seqRun-iff)+

{ fix i
  have ⟨{w. tF w ∧ {ev a | a. ev a ∈ set w} ⊆ set u ∪ ev ‘A ∧ length w ≤ i}
⊆
  map (ev ∘ of-ev) ‘ {w. set w ⊆ set u ∪ ev ‘A ∧ length w ≤ i}⟩
  (is ⟨?S1 ⊆ map (ev ∘ of-ev) ‘ ?S2⟩)
proof (rule subsetI)
  fix w assume ⟨w ∈ ?S1⟩
  hence ⟨map (ev ∘ of-ev) (map (ev ∘ of-ev) w) = w⟩
  by (induct w) (auto simp add: subset-iff)
  moreover from ⟨w ∈ ?S1⟩ have ⟨map (ev ∘ of-ev) w ∈ ?S2⟩
  by (induct w) (auto simp add: subset-iff)
  ultimately show ⟨w ∈ map (ev ∘ of-ev) ‘ ?S2⟩
  by (metis (lifting) image-eqI)
qed
moreover have ⟨finite {w. set w ⊆ set u ∪ ev ‘A ∧ length w ≤ i}⟩
  by (rule finite-lists-length-le) (simp add: ⟨finite A⟩)
ultimately have ⟨finite {w. tF w ∧ {ev a | a. ev a ∈ set w} ⊆ set u ∪ ev ‘
A ∧ length w ≤ i}⟩
  using finite-subset[OF - finite-imageI] by blast
} note ℒ = this

have ⟨inj t-P-t-Q⟩

```


proof (*rule injI*)
fix $i\ j$ **assume** $\langle t\text{-}P\text{-}t\text{-}Q\ i = t\text{-}P\text{-}t\text{-}Q\ j \rangle$
with $****(\mathcal{I})$ **have** $\langle \text{seqRun } u\ x\ i\ \text{setinterleaves}_{\checkmark\text{tick-join}} ((t\text{-}P\ i, t\text{-}Q\ i), S) \rangle$
 $\wedge$
 $\text{seqRun } u\ x\ j\ \text{setinterleaves}_{\checkmark\text{tick-join}} ((t\text{-}P\ i, t\text{-}Q\ i), S) \rangle$
unfolding $t\text{-}P\text{-}t\text{-}Q\text{-def}\ t\text{-}P\text{-def}\ t\text{-}Q\text{-def}$ **by** *fastforce*
with $\text{setinterleaves}_{\text{ptick-eq-length}}$
have $\langle \text{length } (\text{seqRun } u\ x\ i) = \text{length } (\text{seqRun } u\ x\ j) \rangle$ **by** *blast*
thus $\langle i = j \rangle$ **by** *simp*
qed
hence $\langle \text{infinite } (\text{range } t\text{-}P\text{-}t\text{-}Q) \rangle$ **using** *finite-imageD* **by** *blast*
moreover **have** $\langle \text{range } t\text{-}P\text{-}t\text{-}Q \subseteq \text{range } t\text{-}P \times \text{range } t\text{-}Q \rangle$
by (*simp add: t-P-def t-Q-def subset-iff image-iff*) (*metis fst-conv snd-conv*)
ultimately **have** $\langle \text{infinite } (\text{range } t\text{-}P) \vee \text{infinite } (\text{range } t\text{-}Q) \rangle$
by (*meson finite-SigmaI infinite-super*)

thus $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$
proof (*elim disjE*)
assume $\langle \text{infinite } (\text{range } t\text{-}P) \rangle$
have $\langle \text{finite } \{w. \exists t' \in \text{range } t\text{-}P. w = \text{take } i\ t'\} \rangle$ **for** i
using $*****(\mathcal{I})\ ****$
by (*auto intro!: finite-subset[OF - $\mathcal{L}$ [of i]] simp add: image-iff subset-iff*)
(metis append-take-drop-id tickFree-append-iff, metis event_{ptick}.inject(1)
in-set-takeD)
with $\langle \text{infinite } (\text{range } t\text{-}P) \rangle$ **obtain** $t\text{-}P' :: \langle \text{nat} \Rightarrow \text{-} \rangle$
where $\$: \langle \text{strict-mono } t\text{-}P' \rangle\ \langle \text{range } t\text{-}P' \subseteq \{w. \exists t' \in \text{range } t\text{-}P. w \leq t'\} \rangle$
using *KoenigLemma* **by** *blast*
from $\$(2)\ ****(1)$ *is-processT3-TR* **have** $\langle \text{range } t\text{-}P' \subseteq \mathcal{T}\ P \rangle$ **by** *blast*
define $t\text{-}P''$ **where** $\langle t\text{-}P''\ i \equiv t\text{-}P'\ (i + \text{length } u) \rangle$ **for** i
from $\langle \text{range } t\text{-}P' \subseteq \mathcal{T}\ P \rangle$ **have** $\langle \text{range } t\text{-}P'' \subseteq \mathcal{T}\ P \rangle$ **and** $\langle \text{strict-mono } t\text{-}P'' \rangle$
by (*auto simp add: t-P''-def $\$(1)$ strict-monoD strict-monoI*)
have $\$ \$: \langle ?th\text{-}A\ (t\text{-}P''\ i) = ?th\text{-}A\ (t\text{-}P''\ 0) \rangle$ **for** i
proof –
have $\langle \text{length } u \leq \text{length } (t\text{-}P''\ 0) \rangle$
by (*metis $\$(1)$ add-0 add-leD1 t-P''-def length-strict-mono*)
obtain t' **where** $\langle t\text{-}P''\ i = t\text{-}P''\ 0 @ t' \rangle$
by (*meson prefixE $\langle \text{strict-mono } t\text{-}P'' \rangle$ strict-mono-less-eq zero-order(1)*)
moreover **from** $\$(2)$ **obtain** j **where** $\langle t\text{-}P''\ i \leq t\text{-}P\ j \rangle$ **by** (*auto simp*
add: t-P''-def)
ultimately **obtain** t'' **where** $\langle t\text{-}P\ j = t\text{-}P''\ 0 @ t' @ t'' \rangle$ **by** (*metis prefixE*
append.assoc)

have $\langle tF\ (t' @ t'') \rangle$
by (*metis $*****(\mathcal{I})\ \langle t\text{-}P\ j = t\text{-}P''\ 0 @ t' @ t'' \rangle$ tickFree-append-iff*)
with $\text{setinterleaves}_{\text{ptick-set-subsetL}}$
 $[OF\ ****(\mathcal{I})[of\ j], \text{where } n = \langle \text{length } (t\text{-}P''\ 0) \rangle, \text{unfolded } \langle t\text{-}P\ j = t\text{-}P''\ 0 @ t' @ t'' \rangle]$
have $\langle e \in \text{set } (t' @ t'') \implies e \in \{ev\ a \mid a. ev\ a \in \text{set } (\text{drop } (\text{length } (t\text{-}P''\ 0))\ (\text{seqRun } u\ x\ j))\} \rangle$ **for** e

$\subseteq$

 by (cases e) (auto simp add: tickFree-def)

 moreover have $\langle \{a. \text{ev } a \in \text{set } (\text{drop } (\text{length } (t\text{-}P'' 0)) (\text{seqRun } u \ x \ j))\} \rangle$

 $\{a. \text{ev } a \in \text{set } (\text{drop } (\text{length } u) (\text{seqRun } u \ x \ j))\}$

 by (simp add: subset-iff)

 (meson $\langle \text{length } u \leq \text{length } (t\text{-}P'' 0) \rangle$ in-mono set-drop-subset-set-drop)

 moreover from ** have $\langle \text{set } (\text{drop } (\text{length } u) (\text{seqRun } u \ x \ j)) \subseteq \text{ev } 'A \rangle$

 by (auto simp add: seqRun-def)

 ultimately have $\langle \text{set } (t' @ t'') \subseteq \text{ev } 'A \rangle$ by blast

 thus $\langle ?th\text{-}A \ (t\text{-}P'' i) = ?th\text{-}A \ (t\text{-}P'' 0) \rangle$

 by (simp add: $\langle t\text{-}P'' i = t\text{-}P'' 0 @ t' \rangle$ subset-iff)

 qed

 from $\S(2)$ obtain i where $\langle t\text{-}P'' 0 \leq t\text{-}P \ i \rangle$ by (auto simp add: t-P''-def)

 with prefixE obtain w where $\langle t\text{-}P \ i = t\text{-}P'' 0 @ w \rangle$ by blast

 have $\langle \text{ftF } v \rangle$ by (fact $\ast(1)$)

 moreover have $\langle tF \ (?th\text{-}A \ (\text{seqRun } u \ x \ i)) \rangle$

 by (metis $\ast(2)$ ** Hiding-tickFree trace-hide-seqRun-eq-iff)

 moreover have $\langle t = ?th\text{-}A \ (\text{seqRun } u \ x \ i) @ v \rangle$

 by (metis $\ast(3)$ ** trace-hide-seqRun-eq-iff)

 moreover have $\langle ?th\text{-}A \ (\text{seqRun } u \ x \ i) \text{ setinterleaves } \checkmark_{(\otimes \checkmark)} ((?th\text{-}A \ (t\text{-}P \ i),$

 $?th\text{-}A \ (t\text{-}Q \ i)), S) \rangle$

 by (intro setinterleaves_{ptick}-trace-hide **** $\ast(3)$)

 moreover have $\langle ?th\text{-}A \ (t\text{-}P \ i) \in \mathcal{D} \ (P \setminus A) \rangle$

 proof (unfold D-Hiding, clarify, intro exI conjI)

 show $\langle \text{ftF } (?th\text{-}A \ w) \rangle$

 by (metis ***** $\ast(1)$ Hiding-front-tickFree $\langle t\text{-}P \ i = t\text{-}P'' 0 @ w \rangle$

 tickFree-append-iff tickFree-imp-front-tickFree)

 next

 show $\langle tF \ (t\text{-}P'' 0) \rangle$

 by (metis ***** $\ast(1)$ $\langle t\text{-}P \ i = t\text{-}P'' 0 @ w \rangle$ tickFree-append-iff)

 next

 show $\langle ?th\text{-}A \ (t\text{-}P \ i) = ?th\text{-}A \ (t\text{-}P'' 0) @ ?th\text{-}A \ w \rangle$

 by (simp add: $\langle t\text{-}P \ i = t\text{-}P'' 0 @ w \rangle$)

 next

 show $\langle t\text{-}P'' 0 \in \mathcal{D} \ P \vee (\exists f. \text{isInfHiddenRun } f \ P \ A \wedge t\text{-}P'' 0 \in \text{range } f) \rangle$

 using $\S\ \$\ \langle \text{range } t\text{-}P'' \subseteq \mathcal{T} \ P \rangle \langle \text{strict-mono } t\text{-}P'' \rangle$ by blast

 qed

 moreover have $\langle ?th\text{-}A \ (t\text{-}Q \ i) \in \mathcal{T} \ (Q \setminus A) \rangle$

 proof (cases $\langle \exists t'. ?th\text{-}A \ t' = ?th\text{-}A \ (t\text{-}Q \ i) \wedge (t', \text{ev } 'A) \in \mathcal{F} \ Q \rangle$)

 assume $\langle \exists t'. ?th\text{-}A \ t' = ?th\text{-}A \ (t\text{-}Q \ i) \wedge (t', \text{ev } 'A) \in \mathcal{F} \ Q \rangle$

 then obtain t' where $\langle ?th\text{-}A \ (t\text{-}Q \ i) = ?th\text{-}A \ t' \rangle \langle (t', \text{ev } 'A) \in \mathcal{F} \ Q \rangle$ by

 metis

 thus $\langle ?th\text{-}A \ (t\text{-}Q \ i) \in \mathcal{T} \ (Q \setminus A) \rangle$ unfolding T-Hiding by blast

 next

 assume $\nexists t'. ?th\text{-}A \ t' = ?th\text{-}A \ (t\text{-}Q \ i) \wedge (t', \text{ev } 'A) \in \mathcal{F} \ Q$

 with inf-hidden[OF - ***** $\ast(2)$] obtain t-Q' j

 where $\langle \text{isInfHiddenRun } t\text{-}Q' \ Q \ A \rangle \langle t\text{-}Q \ i = t\text{-}Q' \ j \rangle$ by blast

 thus $\langle ?th\text{-}A \ (t\text{-}Q \ i) \in \mathcal{T} \ (Q \setminus A) \rangle$

 unfolding T-Hiding using ***** $\ast(2)$ front-tickFree-Nil by blast

qed
 ultimately show $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$ unfolding $D\text{-Sync}_{ptick}$
 by blast
 next
 assume $\langle \text{infinite } (\text{range } t\text{-}Q) \rangle$
 have $\langle \text{finite } \{w. \exists t' \in \text{range } t\text{-}Q. w = \text{take } i \ t'\} \rangle$ for i
 using ***** (2) *****
 by (auto intro!: finite-subset[OF - $\mathcal{L}$ [of i]] simp add: image-iff subset-iff)
 (metis append-take-drop-id tickFree-append-iff, metis event $_{ptick}$.inject (1))
 in-set-takeD)
 with $\langle \text{infinite } (\text{range } t\text{-}Q) \rangle$ obtain $t\text{-}Q' :: \langle \text{nat} \Rightarrow \cdot \rangle$
 where $\$: \langle \text{strict-mono } t\text{-}Q' \rangle \langle \text{range } t\text{-}Q' \subseteq \{w. \exists t' \in \text{range } t\text{-}Q. w \leq t'\} \rangle$
 using KoenigLemma by blast
 from $\$(2)$ **** (2) is-processT3-TR have $\langle \text{range } t\text{-}Q' \subseteq \mathcal{T} \ Q \rangle$ by blast
 define $t\text{-}Q''$ where $\langle t\text{-}Q'' \ i \equiv t\text{-}Q' \ (i + \text{length } u) \rangle$ for i
 from $\langle \text{range } t\text{-}Q' \subseteq \mathcal{T} \ Q \rangle$ have $\langle \text{range } t\text{-}Q'' \subseteq \mathcal{T} \ Q \rangle$ and $\langle \text{strict-mono } t\text{-}Q'' \rangle$
 by (auto simp add: $t\text{-}Q''$ -def $\$(1)$ strict-monoD strict-monoI)
 have $\$ \$: \langle ?th\text{-}A \ (t\text{-}Q'' \ i) = ?th\text{-}A \ (t\text{-}Q'' \ 0) \rangle$ for i
 proof -
 have $\langle \text{length } u \leq \text{length } (t\text{-}Q'' \ 0) \rangle$
 by (metis $\$(1)$ add-0 add-leD1 $t\text{-}Q''$ -def length-strict-mono)
 obtain t' where $\langle t\text{-}Q'' \ i = t\text{-}Q'' \ 0 \ @ \ t' \rangle$
 by (meson prefixE $\langle \text{strict-mono } t\text{-}Q'' \rangle$ strict-mono-less-eq zero-order (1))
 moreover from $\$(2)$ obtain j where $\langle t\text{-}Q'' \ i \leq t\text{-}Q \ j \rangle$ by (auto simp
 add: $t\text{-}Q''$ -def)
 ultimately obtain t'' where $\langle t\text{-}Q \ j = t\text{-}Q'' \ 0 \ @ \ t' \ @ \ t'' \rangle$ by (metis
 prefixE append.assoc)
 have $\langle tF \ (t' \ @ \ t'') \rangle$
 by (metis ***** (2) $\langle t\text{-}Q \ j = t\text{-}Q'' \ 0 \ @ \ t' \ @ \ t'' \rangle$ tickFree-append-iff)
 with setinterleaves $_{ptick}$ -set-subsetR
 [OF ***** (3) [of j], where $n = \langle \text{length } (t\text{-}Q'' \ 0) \rangle$, unfolded $\langle t\text{-}Q \ j = t\text{-}Q''$
 $0 \ @ \ t' \ @ \ t'' \rangle$]
 have $\langle e \in \text{set } (t' \ @ \ t'') \implies e \in \{ev \ a \mid a. ev \ a \in \text{set } (\text{drop } (\text{length } (t\text{-}Q''$
 $0)) \ (\text{seqRun } u \ x \ j))\} \rangle$ for e
 by (cases e) (auto simp add: tickFree-def)
 moreover have $\langle \{a. ev \ a \in \text{set } (\text{drop } (\text{length } (t\text{-}Q'' \ 0)) \ (\text{seqRun } u \ x \ j))\}$
 $\subseteq$
 $\{a. ev \ a \in \text{set } (\text{drop } (\text{length } u) \ (\text{seqRun } u \ x \ j))\} \rangle$
 by (simp add: subset-iff)
 (meson $\langle \text{length } u \leq \text{length } (t\text{-}Q'' \ 0) \rangle$ in-mono set-drop-subset-set-drop)
 moreover from ** have $\langle \text{set } (\text{drop } (\text{length } u) \ (\text{seqRun } u \ x \ j)) \subseteq ev \ 'A \rangle$
 by (auto simp add: seqRun-def)
 ultimately have $\langle \text{set } (t' \ @ \ t'') \subseteq ev \ 'A \rangle$ by blast
 thus $\langle ?th\text{-}A \ (t\text{-}Q'' \ i) = ?th\text{-}A \ (t\text{-}Q'' \ 0) \rangle$
 by (simp add: $\langle t\text{-}Q'' \ i = t\text{-}Q'' \ 0 \ @ \ t' \rangle$ subset-iff)
 qed
 from $\$(2)$ obtain i where $\langle t\text{-}Q'' \ 0 \leq t\text{-}Q \ i \rangle$ by (auto simp add: $t\text{-}Q''$ -def)
 with prefixE obtain w where $\langle t\text{-}Q \ i = t\text{-}Q'' \ 0 \ @ \ w \rangle$ by blast
 have $\langle ftF \ v \rangle$ by (fact $\ast(1)$)

```

moreover have  $\langle tF \text{ (?th-A (seqRun u x i))} \rangle$ 
  by (metis *(2) ** Hiding-tickFree trace-hide-seqRun-eq-iff)
moreover have  $\langle t = \text{?th-A (seqRun u x i)} @ v \rangle$ 
  by (metis *(3) ** trace-hide-seqRun-eq-iff)
moreover have  $\langle \text{?th-A (seqRun u x i) setinterleaves}_{\checkmark}(\otimes\checkmark) ((\text{?th-A (t-P i)},$ 
 $\text{?th-A (t-Q i)}, S) \rangle$ 
  by (intro setinterleavesptick-trace-hide ****(3))
moreover have  $\langle \text{?th-A (t-Q i)} \in \mathcal{D} (Q \setminus A) \rangle$ 
proof (unfold D-Hiding, clarify, intro exI conjI)
  show  $\langle tF \text{ (?th-A w)} \rangle$ 
    by (metis ***** (2) Hiding-front-tickFree  $\langle t-Q i = t-Q'' 0 @ w \rangle$ 
      tickFree-append-iff tickFree-imp-front-tickFree)
next
  show  $\langle tF (t-Q'' 0) \rangle$ 
    by (metis ***** (2)  $\langle t-Q i = t-Q'' 0 @ w \rangle$  tickFree-append-iff)
next
  show  $\langle \text{?th-A (t-Q i)} = \text{?th-A (t-Q'' 0)} @ \text{?th-A w} \rangle$ 
    by (simp add:  $\langle t-Q i = t-Q'' 0 @ w \rangle$ )
next
  show  $\langle t-Q'' 0 \in \mathcal{D} Q \vee (\exists f. \text{isInfHiddenRun } f \text{ } Q \text{ } A \wedge t-Q'' 0 \in \text{range } f) \rangle$ 
    using $$  $\langle \text{range } t-Q'' \subseteq \mathcal{T} Q \rangle \langle \text{strict-mono } t-Q'' \rangle$  by blast
qed
moreover have  $\langle \text{?th-A (t-P i)} \in \mathcal{T} (P \setminus A) \rangle$ 
proof (cases  $\langle \exists t'. \text{?th-A } t' = \text{?th-A (t-P i)} \wedge (t', \text{ev } A) \in \mathcal{F} P \rangle$ )
  assume  $\langle \exists t'. \text{?th-A } t' = \text{?th-A (t-P i)} \wedge (t', \text{ev } A) \in \mathcal{F} P \rangle$ 
  then obtain  $t'$  where  $\langle \text{?th-A (t-P i)} = \text{?th-A } t' \rangle \langle (t', \text{ev } A) \in \mathcal{F} P \rangle$  by
metis
    thus  $\langle \text{?th-A (t-P i)} \in \mathcal{T} (P \setminus A) \rangle$  unfolding T-Hiding by blast
next
  assume  $\langle \nexists t'. \text{?th-A } t' = \text{?th-A (t-P i)} \wedge (t', \text{ev } A) \in \mathcal{F} P \rangle$ 
  with  $\text{inf-hidden}[OF - ****(1)]$  obtain  $t-P' j$ 
    where  $\langle \text{isInfHiddenRun } t-P' P \text{ } A \rangle \langle t-P i = t-P' j \rangle$  by blast
  thus  $\langle \text{?th-A (t-P i)} \in \mathcal{T} (P \setminus A) \rangle$ 
    unfolding T-Hiding using ***** (1)  $\text{front-tickFree-Nil}$  by blast
qed
ultimately show  $\langle t \in \mathcal{D} ((P \setminus A) \llbracket S \rrbracket_{\checkmark} (Q \setminus A)) \rangle$  unfolding D-Syncptick
by blast
  qed
  qed
  qed
  qed

```

lemma *disjoint-Hiding-MultiSync_{ptick}-FD-MultiSync_{ptick}-Hiding* :

$\langle \llbracket S \rrbracket_{\checkmark} l \in @ L. P \text{ } l \setminus A \sqsubseteq_{FD} \llbracket S \rrbracket_{\checkmark} l \in @ L. (P \text{ } l \setminus A) \rangle$ **if** $\langle A \cap S = \{\} \rangle$

proof (induct L rule: induct-list012)

case 1 show ?case **by** simp

next

```

    case (2 l0)
    show ?case by (simp add: RenamingTick-Hiding)
next
    case (3 l0 l1 L)
    have ⟨[S]✓ l ∈@ (l0 # l1 # L). P l \ A =
      P l0 [S]✓Rlist [S]✓ l ∈@ (l1 # L). P l \ A⟩ by simp
    also have ⟨... ⊆FD (P l0 \ A) [S]✓Rlist ([S]✓ l ∈@ (l1 # L). P l \ A)⟩
      by (simp add: SyncRlist.disjoint-Hiding-Syncptick-FD-Syncptick-Hiding ⟨A ∩
S = {}⟩)
    also have ⟨... ⊆FD (P l0 \ A) [S]✓Rlist [S]✓ l ∈@ (l1 # L). (P l \ A)⟩
      by (simp add: 3.hyps(2) SyncRlist.mono-Syncptick-FD)
    also have ⟨... = [S]✓ l ∈@ (l0 # l1 # L). (P l \ A)⟩ by simp
    finally show ?case .
qed

```

lemma *disjoint-finite-Hiding-MultiSync_{ptick}* :

⟨[S]_✓ l ∈@ L. P l \ A = [S]_✓ l ∈@ L. (P l \ A)⟩ if ⟨A ∩ S = {}⟩ and ⟨finite A⟩

proof (induct L rule: induct-list012)

case 1 show ?case by simp

next

case (2 l0)

show ?case by (simp add: RenamingTick-Hiding)

next

case (3 l0 l1 L)

have ⟨[S]_✓ l ∈@ (l0 # l1 # L). P l \ A =

P l0 [S]_{✓Rlist} [S]_✓ l ∈@ (l1 # L). P l \ A⟩ by simp

also have ⟨... = (P l0 \ A) [S]_{✓Rlist} ([S]_✓ l ∈@ (l1 # L). P l \ A)⟩

by (simp add: Sync_{Rlist}.disjoint-finite-Hiding-Sync_{ptick} ⟨A ∩ S = {}⟩ ⟨finite A⟩)

also have ⟨... = (P l0 \ A) [S]_{✓Rlist} [S]_✓ l ∈@ (l1 # L). (P l \ A)⟩

by (simp add: 3.hyps(2) Sync_{Rlist}.mono-Sync_{ptick}-FD)

also have ⟨... = [S]_✓ l ∈@ (l0 # l1 # L). (P l \ A)⟩ by simp

finally show ?case .

qed

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 ∈ set u ∨ ev a ∈ set v} ⊆ A ⟹

t setinterleaves_{✓tick-join} ((u, v), S) ⟷

t setinterleaves_{✓tick-join} ((u, v), S ∩ A)⟩

by (induct ⟨(tick-join, u, S, v)⟩ arbitrary: t u v)

(auto simp add: subset-iff split: option.split-asm)

lemma (in Sync_{ptick}-locale) *Sync_{ptick}-is-restrictable-on-events-of* :

$\langle P \llbracket S \rrbracket_{\checkmark} Q = P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q \rangle$
proof –
have $*$: $\langle t-P \in \mathcal{D} \ P \wedge t-Q \in \mathcal{T} \ Q \vee t-P \in \mathcal{T} \ P \wedge t-Q \in \mathcal{D} \ Q \implies$
 $\{a. \text{ ev } a \in \text{set } t-P \vee \text{ev } a \in \text{set } t-Q\} \subseteq \alpha(P) \cup \alpha(Q) \rangle$
 $\langle (t-P, X-P) \in \mathcal{F} \ P \implies (t-Q, X-Q) \in \mathcal{F} \ Q \implies$
 $\{a. \text{ ev } a \in \text{set } t-P \vee \text{ev } a \in \text{set } t-Q\} \subseteq \alpha(P) \cup \alpha(Q) \rangle$ **for** $t-P \ t-Q \ X-P \ X-Q$
by (*auto intro: events-of-memI dest: F-T D-T*)
show $\langle P \llbracket S \rrbracket_{\checkmark} Q = P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q \rangle$
proof (*rule Process-eq-optimizedI*)
show $\langle t \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \implies t \in \mathcal{D} \ (P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q) \rangle$ **for** t
using *setinterleaves_{ptick}-is-restrictable-on-superset-events-of*[*OF* $*(1)$]
unfolding *D-Sync_{ptick}* **by** *blast*
next
show $\langle t \in \mathcal{D} \ (P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q) \implies t \in \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **for** t
using *setinterleaves_{ptick}-is-restrictable-on-superset-events-of*[*OF* $*(1)$]
unfolding *D-Sync_{ptick}* **by** *blast*
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ $\langle t \notin \mathcal{D} \ (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
then obtain $t-P \ t-Q \ X-P \ X-Q$
where $\$$: $\langle (t-P, X-P) \in \mathcal{F} \ P \rangle$ $\langle (t-Q, X-Q) \in \mathcal{F} \ Q \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} \ ((t-P, t-Q), S) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{ tick-join } X-P \ S \ X-Q \rangle$
unfolding *Sync_{ptick}-projs* **by** *blast*
from $\$(1)$ **have** $\langle (t-P, X-P \cup \{ \text{ev } a \mid a. a \notin \alpha(P) \}) \in \mathcal{F} \ P \rangle$
by (*rule is-processT5*) (*auto intro: events-of-memI dest!: F-T*)
moreover from $\$(2)$ **have** $\langle (t-Q, X-Q \cup \{ \text{ev } a \mid a. a \notin \alpha(Q) \}) \in \mathcal{F} \ Q \rangle$
by (*rule is-processT5*) (*auto intro: events-of-memI dest!: F-T*)
moreover from *setinterleaves_{ptick}-is-restrictable-on-superset-events-of*
 $[OF \ *(2)[OF \ \$(1, 2)]] \ \(3)
have $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} \ ((t-P, t-Q), S \cap (\alpha(P) \cup \alpha(Q))) \rangle$ **by** *blast*
moreover from $\$(4)$
have $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{ tick-join} \ (X-P \cup \{ \text{ev } a \mid a. a \notin \alpha(P) \})$
 $(S \cap (\alpha(P) \cup \alpha(Q))) \ (X-Q \cup \{ \text{ev } a \mid a. a \notin \alpha(Q) \}) \rangle$
by (*auto simp add: super-ref-Sync_{ptick}-def*)
ultimately show $\langle (t, X) \in \mathcal{F} \ (P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q) \rangle$
by (*auto simp add: F-Sync_{ptick}*)
next
fix $t \ X$ **assume** $\langle (t, X) \in \mathcal{F} \ (P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q) \rangle$
 $\langle t \notin \mathcal{D} \ (P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q) \rangle$
then obtain $t-P \ t-Q \ X-P \ X-Q$
where $\$$: $\langle (t-P, X-P) \in \mathcal{F} \ P \rangle$ $\langle (t-Q, X-Q) \in \mathcal{F} \ Q \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} \ ((t-P, t-Q), S \cap (\alpha(P) \cup \alpha(Q))) \rangle$
 $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{ tick-join } X-P \ (S \cap (\alpha(P) \cup \alpha(Q))) \ X-Q \rangle$
unfolding *Sync_{ptick}-projs* **by** *blast*
from *setinterleaves_{ptick}-is-restrictable-on-superset-events-of*
 $[OF \ *(2)[OF \ \$(1, 2)]] \ \(3)
have $\langle t \text{ setinterleaves}_{\checkmark} \text{tick-join} \ ((t-P, t-Q), S) \rangle$ **by** *blast*
moreover from $\$(4)$ **have** $\langle X \subseteq \text{super-ref-Sync}_{\text{ptick}} \text{ tick-join } X-P \ S \ X-Q \rangle$

```

    by (meson Int-lower1 in-mono subsetI super-ref-Syncptick-mono)
    ultimately show  $\langle t, X \rangle \in \mathcal{F} (P \llbracket S \rrbracket_{\checkmark} Q)$ 
    using  $\$(1, 2)$  by (auto simp add: F-Syncptick)
qed
qed

corollary (in Syncptick-locale) Syncptick-is-restrictable-on-superset-events-of :
 $\langle P \llbracket S \rrbracket_{\checkmark} Q = P \llbracket S \cap A \rrbracket_{\checkmark} Q \rangle$  if  $\langle \alpha(P) \cup \alpha(Q) \subseteq A \rangle$ 
proof (rule trans[OF Syncptick-is-restrictable-on-events-of],
      rule trans[OF - Syncptick-is-restrictable-on-events-of[symmetric]])
  show  $\langle P \llbracket S \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q = P \llbracket S \cap A \cap (\alpha(P) \cup \alpha(Q)) \rrbracket_{\checkmark} Q \rangle$ 
  using  $\langle \alpha(P) \cup \alpha(Q) \subseteq A \rangle$  by (auto intro: arg-cong[where f =  $\langle \lambda S. P \llbracket S \rrbracket_{\checkmark} Q \rangle$ ])
qed

```

```

lemma  $\langle tF t \implies \{a. \text{ev } a \in \text{set } u\} \cap S = \{\} \implies a \in S \implies$ 
 $\neg t \text{ setinterleaves}_{\checkmark \text{ tick-join}} ((u, \text{ev } a \# v), S) \rangle$ 
proof (induct  $\langle \text{tick-join}, u, S, v \rangle$  arbitrary:  $t \ u \ v$ )
  case (ev-setinterleavingptick-Nil a u)
  thus ?case by (simp add: disjoint-iff) (meson tickFree-Cons-iff)
next
  case (ev-setinterleavingptick-ev a u b v)
  then show ?case by (simp add: disjoint-iff) (meson tickFree-Cons-iff)
next
  case (ev-setinterleavingptick-tick a u s v)
  thus ?case
    by (simp add: disjoint-iff)
    (metis empty-iff list.set-intros(1)
      ev-notin-both-sets-imp-empty-setinterleavingptick)
qed simp-all

```

16.5.2 Some Refinements

context Sync_{ptick}-locale begin

```

lemma Mndetprefix-Syncptick-Det-distr-FD :
 $\langle (\sqcap a \in A \rightarrow (P \ a \llbracket C \rrbracket_{\checkmark} (\sqcap b \in B \rightarrow Q \ b))) \sqcap$ 
 $(\sqcap b \in B \rightarrow ((\sqcap a \in A \rightarrow P \ a) \llbracket C \rrbracket_{\checkmark} Q \ b))$ 
 $\sqsubseteq_{FD} (\sqcap a \in A \rightarrow P \ a) \llbracket C \rrbracket_{\checkmark} (\sqcap b \in B \rightarrow Q \ b) \rangle$ 
(is  $\langle ?lhs1 \sqcap ?lhs2 \sqsubseteq_{FD} ?rhs \rangle$ )
if  $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle \langle A \cap C = \{\} \rangle \langle B \cap C = \{\} \rangle$ 
proof -
  have  $\langle ?lhs1 = \sqcap b \in B. \sqcap a \in A. (a \rightarrow (P \ a \llbracket C \rrbracket_{\checkmark} (b \rightarrow Q \ b))) \rangle$  (is  $\langle - = ?lhs1 \rangle$ )
  by (simp add:  $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle$  Mndetprefix-GlobalNdet
    Syncptick-distrib-GlobalNdet-left Syncptick-distrib-GlobalNdet-right
    write0-def GlobalNdet-Mprefix-distr[OF  $\langle B \neq \{\} \rangle$ , symmetric])
    (subst GlobalNdet-sets-commute, simp)
  moreover have  $\langle ?lhs2 = \sqcap b \in B. \sqcap a \in A. (b \rightarrow (a \rightarrow P \ a \llbracket C \rrbracket_{\checkmark} Q \ b)) \rangle$  (is  $\langle - =$ 

```

$?lhs2'$)
by (*simp add*: $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle$ *Mndetprefix-GlobalNdet*
Sync_{ptick}-distrib-GlobalNdet-left Sync_{ptick}-distrib-GlobalNdet-right
write0-def GlobalNdet-Mprefix-distr[OF $\langle A \neq \{\} \rangle$, symmetric])
ultimately have $\langle ?lhs1 \sqcap ?lhs2 = ?lhs1' \sqcap ?lhs2' \rangle$ **by** *simp*
moreover have $\langle ?lhs1' \sqcap ?lhs2' \sqsubseteq_{FD} \sqcap b \in B. \sqcap a \in A. (a \rightarrow (P a \llbracket C \rrbracket_{\checkmark} (b \rightarrow Q b))) \rangle$
 $\sqcap (b \rightarrow ((a \rightarrow P a) \llbracket C \rrbracket_{\checkmark} Q b)) \rangle$
by (*auto simp add*: $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle$ *refine-defs GlobalNdet-projs Det-projs*
write0-def)
moreover have $\langle \dots = \sqcap b \in B. \sqcap a \in A. ((a \rightarrow P a) \llbracket C \rrbracket_{\checkmark} (b \rightarrow Q b)) \rangle$
by (*rule mono-GlobalNdet-eq, rule mono-GlobalNdet-eq,*
simp add: *write0-def, subst Mprefix-Sync_{ptick}-Mprefix-indep*)
(use $\langle A \cap C = \{\} \rangle \langle B \cap C = \{\} \rangle$ in auto)
moreover have $\langle \dots = ?rhs \rangle$
by (*simp add*: $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle$ *Mndetprefix-GlobalNdet*
Sync_{ptick}-distrib-GlobalNdet-left Sync_{ptick}-distrib-GlobalNdet-right)
ultimately show $\langle ?lhs1 \sqcap ?lhs2 \sqsubseteq_{FD} ?rhs \rangle$ **by** *argo*
qed

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 :*
 $\langle \llbracket A \neq \{\}; B \neq \{\}; A \cap C = \{\}; B \cap C = \{\} \rrbracket \implies$
 $(\sqcap a \in A \rightarrow (P a \llbracket C \rrbracket_{\checkmark} (\sqcap b \in B \rightarrow Q b))) \sqcap$
 $(\sqcap b \in B \rightarrow ((\sqcap a \in A \rightarrow P a) \llbracket C \rrbracket_{\checkmark} Q b))$
 $\sqsubseteq_{DT} (\sqcap a \in A \rightarrow P a) \llbracket C \rrbracket_{\checkmark} (\sqcap b \in B \rightarrow Q b) \rangle$
by (*simp add*: *Mndetprefix-Sync_{ptick}-Det-distr-D*
Mndetprefix-Sync_{ptick}-Det-distr-T leD-leT-imp-leDT)

end

Chapter 17

Deadlock Results

17.1 First Results

17.1.1 Non Terminating

Keep in mind $\text{lifelock-free}_{SKIPS} P = (\mathcal{D} P = \{\})$.

Sequential Composition

lemma $\langle \text{non-terminating } P \implies P ;_{\checkmark} Q = \text{RenamingTick } P \ g \rangle$
— Already proven earlier.
by (*fact non-terminating-Seq_{ptick}*)

Synchronization Product

lemma (*in Sync_{ptick}-locale*) *non-terminating-Sync_{ptick}* :
 $\langle \text{non-terminating } P \implies \text{lifelock-free}_{SKIPS} Q \implies \text{non-terminating } (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
 $\langle \text{lifelock-free}_{SKIPS} P \implies \text{non-terminating } Q \implies \text{non-terminating } (P \llbracket A \rrbracket_{\checkmark} Q) \rangle$
by (*simp add: lifelock-free_{SKIPS}-iff-div-free T-Sync_{ptick}*
non-terminating-is-right nonterminating-implies-div-free,
use setinterleaves_{ptick}-tickFree-imp in blast) $+$

17.1.2 Deadlock Free

Sequential Composition

lemma $\langle \text{deadlock-free } P \implies \text{deadlock-free } (P ;_{\checkmark} Q) \rangle$
by (*metis deadlock-free-imp-deadlock-free-Renaming deadlock-free-implies-lifelock-free*
lifelock-free-is-non-terminating non-terminating-Seq_{ptick})

The next lemma is of course more interesting.

lemma *deadlock-free_{SKIPS}-Seq_{ptick}* :
 $\langle \text{deadlock-free}_{SKIPS} (P ;_{\checkmark} Q) \rangle$
if *df-assms* : $\langle \text{deadlock-free}_{SKIPS} P \rangle \langle \bigwedge r. r \in \checkmark s(P) \implies \text{deadlock-free}_{SKIPS} (Q \ r) \rangle$

proof (*unfold deadlock-free_{SKIPS}-is-right, intro ballI impI*)
show $\langle t \in \mathcal{T} (P ; \checkmark Q) \implies tF t \implies (t, UNIV) \notin \mathcal{F} (P ; \checkmark Q) \rangle$ **for** t
proof (*induct t rule: rev-induct*)
from *df-assms* **show** $\langle [], UNIV \rangle \notin \mathcal{F} (P ; \checkmark Q)$
by (*simp add: Seq_{ptick}-projs deadlock-free_{SKIPS}-implies-div-free deadlock-free_{SKIPS}-is-right ref-Seq_{ptick}-UNIV*)
(metis F-T append-Nil deadlock-free_{SKIPS}-implies-div-free
deadlock-free_{SKIPS}-is-right empty-iff strict-ticks-of-memI tickFree-Nil)
next
from *df-assms*(1) **have** $\langle \mathcal{D} P = \{\} \rangle$
by (*simp add: deadlock-free_{SKIPS}-implies-div-free*)
fix $t e$ **assume** *hyp* : $\langle t \in \mathcal{T} (P ; \checkmark Q) \implies tF t \implies (t, UNIV) \notin \mathcal{F} (P ; \checkmark Q) \rangle$
assume $\langle t @ [e] \in \mathcal{T} (P ; \checkmark Q) \rangle \langle tF (t @ [e]) \rangle$
then consider $u v$ **where** $\langle t @ [e] = \text{map } (ev \circ of-ev) u \rangle \langle u \in \mathcal{T} P \rangle \langle tF u \rangle$
 $| u r v$ **where** $\langle t @ [e] = \text{map } (ev \circ of-ev) u @ v \rangle \langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle \langle tF u \rangle$
 $\langle v \in \mathcal{T} (Q r) \rangle$
by (*auto simp add: Seq_{ptick}-projs* $\langle \mathcal{D} P = \{\} \rangle$)
thus $\langle (t @ [e], UNIV) \notin \mathcal{F} (P ; \checkmark Q) \rangle$
by (*cases; simp-all add: Seq_{ptick}-projs ref-Seq_{ptick}-UNIV*)
(metis (no-types) F-T $\langle \mathcal{D} P = \{\} \rangle \langle tF (t @ [e]) \rangle *deadlock-free_{SKIPS}-is-right*
empty-iff strict-ticks-of-memI that tickFree-append-iff) +
qed
qed$

corollary *deadlock-free-Seq_{ptick}* :
 $\langle \llbracket \text{deadlock-free}_{SKIPS} P ; \bigwedge r. r \in \checkmark s(P) \implies \text{deadlock-free} (Q r) \rrbracket \implies \text{deadlock-free} (P ; \checkmark Q) \rangle$
by (*simp add: AfterExt.deadlock-free-iff-empty-ticks-of-and-deadlock-free_{SKIPS} ticks-of-Seq_{ptick}*)
(meson deadlock-free_{SKIPS}-Seq_{ptick} deadlock-free_{SKIPS}-implies-div-free)

Synchronization Product

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

lemma *deadlock-free-Det-bis* :
 $\langle P = STOP \wedge Q \neq STOP \vee \text{deadlock-free } P \implies Q = STOP \wedge P \neq STOP \vee \text{deadlock-free } Q \implies \text{deadlock-free} (P \sqcap Q) \rangle$
using *deadlock-free-Det* **by** *auto*

lemma *deadlock-free-Mprefix-Sync_{ptick}-Mprefix* :
assumes *not-all-empty*: $\langle \neg A \subseteq S \vee \neg B \subseteq S \vee A \cap B \cap S \neq \{\} \rangle$
and $\langle \bigwedge a. a \in A - S \implies \text{deadlock-free} (P a \llbracket S \rrbracket_{\checkmark} \sqcap b \in B \rightarrow Q b) \rangle$
and $\langle \bigwedge b. b \in B - S \implies \text{deadlock-free} (\sqcap a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} Q b) \rangle$
and $\langle \bigwedge x. x \in A \cap B \cap S \implies \text{deadlock-free} (P x \llbracket S \rrbracket_{\checkmark} Q x) \rangle$
shows $\langle \text{deadlock-free} (\sqcap a \in A \rightarrow P a \llbracket S \rrbracket_{\checkmark} \sqcap b \in B \rightarrow Q b) \rangle$
unfolding *Mprefix-Sync_{ptick}-Mprefix* **using** *not-all-empty*
by (*auto intro!: deadlock-free-Det-bis*)

simp add: Mprefix-is-STOP-iff Det-is-STOP-iff deadlock-free-Mprefix-iff assms(2-4))

lemma *deadlock-free-Mprefix-Sync_{ptick}-Mprefix-subset* :
 $\langle [A \subseteq S; B \subseteq S; A \cap B \neq \{\};$
 $\bigwedge x. x \in A \cap B \cap S \implies \text{deadlock-free } (P\ x\ [S]_{\checkmark}\ Q\ x)]$
 $\implies \text{deadlock-free } (\Box a \in A \rightarrow P\ a\ [S]_{\checkmark}\ \Box b \in B \rightarrow Q\ b)\rangle$
and *deadlock-free-Mprefix-Sync_{ptick}-Mprefix-indep* :
 $\langle [A \cap S = \{\}; B \cap S = \{\}; A \neq \{\} \vee B \neq \{\};$
 $\bigwedge a. a \in A - S \implies \text{deadlock-free } (P\ a\ [S]_{\checkmark}\ \Box b \in B \rightarrow Q\ b);$
 $\bigwedge b. b \in B - S \implies \text{deadlock-free } (\Box a \in A \rightarrow P\ a\ [S]_{\checkmark}\ Q\ b)]$
 $\implies \text{deadlock-free } (\Box a \in A \rightarrow P\ a\ [S]_{\checkmark}\ \Box b \in B \rightarrow Q\ b)\rangle$
and *deadlock-free-Mprefix-Sync_{ptick}-Mprefix-right* :
 $\langle [A \subseteq S; B \cap S = \{\}; B \neq \{\};$
 $\bigwedge b. b \in B - S \implies \text{deadlock-free } (\Box a \in A \rightarrow P\ a\ [S]_{\checkmark}\ Q\ b)]$
 $\implies \text{deadlock-free } (\Box a \in A \rightarrow P\ a\ [S]_{\checkmark}\ \Box b \in B \rightarrow Q\ b)\rangle$
and *deadlock-free-Mprefix-Sync_{ptick}-Mprefix-left* :
 $\langle [A \cap S = \{\}; B \subseteq S; A \neq \{\};$
 $\bigwedge a. a \in A - S \implies \text{deadlock-free } (P\ a\ [S]_{\checkmark}\ \Box b \in B \rightarrow Q\ b)]$
 $\implies \text{deadlock-free } (\Box a \in A \rightarrow P\ a\ [S]_{\checkmark}\ \Box b \in B \rightarrow Q\ b)\rangle$
by (*auto intro!: deadlock-free-Mprefix-Sync_{ptick}-Mprefix*)

end

17.2 Renaming and reference Processes

lemma *DF-empty* $[simp] : \langle DF\ \{\} = STOP \rangle$
and *DF_{SKIPS}-empty* $[simp] : \langle DF_{SKIPS}\ \{\}\ \{\} = STOP \rangle$
and *RUN-empty* $[simp] : \langle RUN\ \{\} = STOP \rangle$
and *CHAOS-empty* $[simp] : \langle CHAOS\ \{\} = STOP \rangle$
and *CHAOS_{SKIPS}-empty* $[simp] : \langle CHAOS_{SKIPS}\ \{\}\ \{\} = STOP \rangle$
by (*subst DF-unfold DF_{SKIPS}-unfold RUN-unfold CHAOS-unfold CHAOS_{SKIPS}-unfold,*
simp)+

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 ($[finitary\ f; finitary\ g] \implies Renaming\ (DF\ A)\ f\ g \sqsubseteq_{FD}\ DF\ (f\ 'A)$).

lemma *DF-restriction-fix-def* : $\langle DF\ A = (\nu\ X. \Box a \in A \rightarrow X) \rangle$
unfolding *DF-def* **by** (*rule restriction-fix-is-fix[symmetric]*) *simp-all*

lemma *DF_{SKIPS}-restriction-fix-def* : $\langle DF_{SKIPS}\ A\ R = (\nu\ X. (\Box a \in A \rightarrow X) \Box SKIPS\ R) \rangle$
unfolding *DF_{SKIPS}-def* **by** (*rule restriction-fix-is-fix[symmetric]*) *simp-all*

lemma *RUN-restriction-fix-def* : $\langle \text{RUN } A = (\nu X. \Box a \in A \rightarrow X) \rangle$
unfolding *RUN-def* **by** (rule *restriction-fix-is-fix[symmetric]*) *simp-all*

lemma *CHAOS-restriction-fix-def* : $\langle \text{CHAOS } A = (\nu X. \text{STOP} \sqcap (\Box a \in A \rightarrow X)) \rangle$
unfolding *CHAOS-def* **by** (rule *restriction-fix-is-fix[symmetric]*) *simp-all*

lemma *CHAOS_{SKIPS}-restriction-fix-def* : $\langle \text{CHAOS}_{\text{SKIPS}} A R = (\nu X. \text{SKIPS } R \sqcap \text{STOP} \sqcap (\Box a \in A \rightarrow X)) \rangle$
unfolding *CHAOS_{SKIPS}-def* **by** (rule *restriction-fix-is-fix[symmetric]*) *simp-all*

17.2.2 Stronger Results

With *restriction-fix* induction, removing these assumptions is trivial.

lemma *Renaming-DF* : $\langle \text{Renaming } (DF A) f g = DF (f \text{ ' } A) \rangle$
proof (unfold *DF-restriction-fix-def*, induct rule: *parallel-restriction-fix-ind*)
show $\langle \text{Renaming } \text{STOP} f g = \text{STOP} \rangle$ **by** *simp*
qed (auto *simp* add: *Renaming-Mndetprefix intro!*: *mono-Mndetprefix-eq*)

lemma *Renaming-DF_{SKIPS}* : $\langle \text{Renaming } (DF_{\text{SKIPS}} A R) f g = DF_{\text{SKIPS}} (f \text{ ' } A) (g \text{ ' } R) \rangle$
proof (unfold *DF_{SKIPS}-restriction-fix-def*, induct rule: *parallel-restriction-fix-ind*)
show $\langle \text{Renaming } \text{STOP} f g = \text{STOP} \rangle$ **by** *simp*
qed (auto *simp* add: *Renaming-Mndetprefix Renaming-Ndet*
intro!: *mono-Mndetprefix-eq arg-cong2[where f = $\langle (\Box) \rangle$]*)

lemma *Renaming-RUN* : $\langle \text{Renaming } (RUN A) f g = RUN (f \text{ ' } A) \rangle$
proof (unfold *RUN-restriction-fix-def*, induct rule: *parallel-restriction-fix-ind*)
show $\langle \text{Renaming } \text{STOP} f g = \text{STOP} \rangle$ **by** *simp*
qed (auto *simp* add: *Renaming-Mprefix intro!*: *mono-Mprefix-eq*)

lemma *Renaming-CHAOS* : $\langle \text{Renaming } (\text{CHAOS } A) f g = \text{CHAOS } (f \text{ ' } A) \rangle$
proof (unfold *CHAOS-restriction-fix-def*, induct rule: *parallel-restriction-fix-ind*)
show $\langle \text{Renaming } \text{STOP} f g = \text{STOP} \rangle$ **by** *simp*
qed (auto *simp* add: *Renaming-Mprefix Renaming-Ndet*
intro!: *mono-Mprefix-eq arg-cong2[where f = $\langle (\Box) \rangle$]*)

lemma *Renaming-CHAOS_{SKIPS}* : $\langle \text{Renaming } (\text{CHAOS}_{\text{SKIPS}} A R) f g = \text{CHAOS}_{\text{SKIPS}} (f \text{ ' } A) (g \text{ ' } R) \rangle$
proof (unfold *CHAOS_{SKIPS}-restriction-fix-def*, induct rule: *parallel-restriction-fix-ind*)
show $\langle \text{Renaming } \text{STOP} f g = \text{STOP} \rangle$ **by** *simp*
qed (auto *simp* add: *Renaming-Mprefix Renaming-Ndet*
intro!: *mono-Mprefix-eq arg-cong2[where f = $\langle (\Box) \rangle$]*)

17.3 Data Independence

When working with the new interleaving $P \llbracket \{ \} \rrbracket_{\checkmark} Q$, 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*:
 $\langle (\forall P Q. \text{deadlock-free } P \longrightarrow \text{deadlock-free } Q \longrightarrow \text{deadlock-free } (P \llbracket S \rrbracket_{\checkmark} Q))$
 $\longleftrightarrow DF \text{ UNIV} \sqsubseteq_{FD} (DF \text{ UNIV} \llbracket S \rrbracket_{\checkmark} DF \text{ UNIV}) \rangle$ (is $\langle ?lhs \longleftrightarrow ?rhs \rangle$)

proof (rule *iffI*)

assume *?lhs*

show *?rhs* by (fold *deadlock-free-def*, rule $\langle ?lhs \rangle[\text{rule-format}]$)
(simp-all add: *deadlock-free-def*)

next

assume *?rhs*

show *?lhs* unfolding *deadlock-free-def*

by (intro *allI impI trans-FD[OF ?rhs]*) (rule *mono-Sync_{ptick}-FD*)

qed

17.3.2 STOP and SKIP synchronized with DF A

The two results below form a stronger (and generalized) version of $r = s \implies (DF A \sqsubseteq_{FD} DF A \llbracket S \rrbracket_{\checkmark} SKIP r) = (A \cap S = \{\})$.

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

lemma (in *Sync_{ptick}-locale*) *DF-FD-DF-Sync_{ptick}-SKIPS-imp-disjoint* :
 $\langle A \cap S = \{\} \rangle$ if $\langle DF A \sqsubseteq_{FD} DF A \llbracket S \rrbracket_{\checkmark} SKIPS R \rangle$

proof (rule *ccontr*)

assume $\langle A \cap S \neq \{\} \rangle$

then obtain *a* where $\langle a \in A \rangle$ and $\langle a \in S \rangle$ by *blast*

have $\langle DF A \llbracket S \rrbracket_{\checkmark} SKIPS R \sqsubseteq_{FD} DF \{a\} \llbracket S \rrbracket_{\checkmark} SKIPS R \rangle$

by (intro *mono-Sync_{ptick}-FD[OF - idem-FD]*) (simp add: *DF-subset* $\langle a \in A \rangle$)

also have $\langle \dots = STOP \rangle$

by (subst *DF-unfold*)

(simp add: $\langle a \in S \rangle$ *SKIPS-def Sync_{ptick}-distrib-GlobalNdet-left*

write0-Sync_{ptick}-STOP write0-Sync_{ptick}-SKIP)

finally show *False*

by (metis that $\langle a \in A \rangle$ *DF-Univ-freeness empty-iff non-deadlock-free-STOP trans-FD*)

qed

lemma *disjoint-imp-DF-eq-DF-Sync_{ptick}-SKIPS* :

$\langle DF A = DF A \llbracket S \rrbracket_{\checkmark} SKIPS R \rangle$ if $\langle A \cap S = \{\} \rangle$

proof (subst *DF-restriction-fix-def*, induct rule: *restriction-fix-ind*)

show $\langle X = DF A \llbracket S \rrbracket_{\checkmark} SKIPS R \implies \bigcap a \in A \rightarrow X = DF A \llbracket S \rrbracket_{\checkmark} SKIPS R \rangle$ for

X

by (subst *DF-unfold*)

(auto simp add: *SKIPS-def Sync_{ptick}-distrib-GlobalNdet-left*

Mndetprefix-Sync_{ptick}-SKIP Mndetprefix-Sync_{ptick}-STOP)

$\langle A \cap S = \{\} \rangle$ *Mndetprefix-distrib-GlobalNdet*)
qed *simp-all*

corollary *DF-FD-DF-Sync_{ptick}-STOP-imp-disjoint* :
 $\langle DF\ A \sqsubseteq_{FD}\ DF\ A\ \llbracket S \rrbracket_{\checkmark}\ STOP \implies A \cap S = \{\} \rangle$
and *DF-FD-DF-Sync_{ptick}-SKIP-imp-disjoint* :
 $\langle DF\ A \sqsubseteq_{FD}\ DF\ A\ \llbracket S \rrbracket_{\checkmark}\ SKIP\ r \implies A \cap S = \{\} \rangle$
and *disjoint-imp-DF-eq-DF-Sync_{ptick}-STOP* :
 $\langle A \cap S = \{\} \implies DF\ A = DF\ A\ \llbracket S \rrbracket_{\checkmark}\ STOP \rangle$
and *disjoint-imp-DF-eq-DF-Sync_{ptick}-SKIP* :
 $\langle A \cap S = \{\} \implies DF\ A = DF\ A\ \llbracket S \rrbracket_{\checkmark}\ SKIP\ r \rangle$
by (*fact DF-FD-DF-Sync_{ptick}-SKIPS-imp-disjoint*[**where** $R = \langle \{\} \rangle$, *simplified*]
DF-FD-DF-Sync_{ptick}-SKIPS-imp-disjoint[**where** $R = \langle \{r\} \rangle$, *simplified*]
disjoint-imp-DF-eq-DF-Sync_{ptick}-SKIPS[**where** $R = \langle \{\} \rangle$, *simplified*]
disjoint-imp-DF-eq-DF-Sync_{ptick}-SKIPS[**where** $R = \langle \{r\} \rangle$, *simplified*])+

end

corollary (**in** *Sync_{ptick}-locale*) *DF-FD-SKIPS-Sync_{ptick}-DF-imp-disjoint* :
 $\langle DF\ A \sqsubseteq_{FD}\ SKIPS\ R\ \llbracket S \rrbracket_{\checkmark}\ DF\ A \implies A \cap S = \{\} \rangle$
by (*metis Sync_{ptick}-locale-sym.DF-FD-DF-Sync_{ptick}-SKIPS-imp-disjoint Sync_{ptick}-sym*)

lemma (**in** *Sync_{ptick}-locale*) *disjoint-imp-DF-eq-SKIPS-Sync_{ptick}-DF* :
 $\langle A \cap S = \{\} \implies DF\ A = SKIPS\ R\ \llbracket S \rrbracket_{\checkmark}\ DF\ A \rangle$
by (*metis Sync_{ptick}-locale-sym.disjoint-imp-DF-eq-DF-Sync_{ptick}-SKIPS Sync_{ptick}-sym*)

corollary (**in** *Sync_{ptick}-locale*) *DF-FD-STOP-Sync_{ptick}-DF-imp-disjoint* :
 $\langle DF\ A \sqsubseteq_{FD}\ STOP\ \llbracket S \rrbracket_{\checkmark}\ DF\ A \implies A \cap S = \{\} \rangle$
and *DF-FD-SKIP-Sync_{ptick}-DF-imp-disjoint* :
 $\langle DF\ A \sqsubseteq_{FD}\ SKIP\ r\ \llbracket S \rrbracket_{\checkmark}\ DF\ A \implies A \cap S = \{\} \rangle$
and *disjoint-imp-DF-eq-STOP-Sync_{ptick}-DF* :
 $\langle A \cap S = \{\} \implies DF\ A = STOP\ \llbracket S \rrbracket_{\checkmark}\ DF\ A \rangle$
and *disjoint-imp-DF-eq-SKIP-Sync_{ptick}-DF* :
 $\langle A \cap S = \{\} \implies DF\ A = SKIP\ r\ \llbracket S \rrbracket_{\checkmark}\ DF\ A \rangle$
by (*fact DF-FD-SKIPS-Sync_{ptick}-DF-imp-disjoint*[**where** $R = \langle \{\} \rangle$, *simplified*]
DF-FD-SKIPS-Sync_{ptick}-DF-imp-disjoint[**where** $R = \langle \{r\} \rangle$, *simplified*]
disjoint-imp-DF-eq-SKIPS-Sync_{ptick}-DF[**where** $R = \langle \{\} \rangle$, *simplified*]
disjoint-imp-DF-eq-SKIPS-Sync_{ptick}-DF[**where** $R = \langle \{r\} \rangle$, *simplified*])+

17.3.3 Finally, deadlock-free ($P \parallel Q$)

theorem (**in** *Sync_{ptick}-locale*) *DF-F-DF-Sync_{ptick}-DF-weak* : $\langle DF\ (A \cup B) \sqsubseteq_F\ DF\ A\ \llbracket S \rrbracket_{\checkmark}\ DF\ B \rangle$
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 –
have $\langle \llbracket (u, X) \in \mathcal{F} (DF A); (v, Y) \in \mathcal{F} (DF B); t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u, v), S) \rrbracket$
 $\implies (t, \text{super-ref-Sync}_{ptick} (\otimes\checkmark) X S Y) \in \mathcal{F} (DF (A \cup B)) \rangle$ **for** $v t u X Y$
proof (induct t arbitrary: $u v$)
case *Nil*
from *Nil.prem*s(3) **have** $\langle u = [] \rangle \langle v = [] \rangle$ **by** (*simp-all add: Nil-setinterleaves_{ptick}*)
from *Nil.prem*s(1) **obtain** a **where** $\langle a \in A \rangle \langle ev a \notin X \rangle$
by (*subst (asm) F-DF*) (*auto simp add: nonempty* $\langle u = [] \rangle$)
moreover from *Nil.prem*s(2) **obtain** b **where** $\langle b \in B \rangle \langle ev b \notin Y \rangle$
by (*subst (asm) F-DF*) (*auto simp add: nonempty* $\langle v = [] \rangle$)
ultimately show *?case*
using *intersect-hyp*
by (*subst F-DF, simp add: nonempty super-ref-Sync_{ptick}-def subset-iff*)
(metis Int-iff empty-iff insert-iff)
next
case (*Cons e t*)
from *Cons.prem*s(3) **consider** (*mv-left*) $a u'$ **where** $\langle a \notin S \rangle \langle e = ev a \rangle \langle u =$
 $ev a \# u' \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u', v), S) \rangle$
 $| (mv-right) b v'$ **where** $\langle b \notin S \rangle \langle e = ev b \rangle \langle v = ev b \# v' \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u, v'), S) \rangle$
 $| (mv-both-ev) a u' v'$ **where** $\langle a \in S \rangle \langle e = ev a \rangle \langle u = ev a \# u' \rangle \langle v = ev a \#$
 $v' \rangle$
 $\langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u', v'), S) \rangle$
 $| (mv-both-tick) r s r-s u' v'$ **where** $\langle r \otimes\checkmark s = \text{Some } r-s \rangle \langle e = \checkmark(r-s) \rangle$
 $\langle u = \checkmark(r) \# u' \rangle \langle v = \checkmark(s) \# v' \rangle \langle t \text{ setinterleaves}_{\checkmark(\otimes\checkmark)} ((u', v'), S) \rangle$
by (*cases e*) (*auto elim: Cons-ev-setinterleaves_{ptick} E Cons-tick-setinterleaves_{ptick} E*)
thus *?case*
proof *cases*
case *mv-left*
from *Cons.prem*s(1) **have** $\langle a \in A \rangle$
by (*subst (asm) F-DF*) (*simp add: mv-left(3) split: if-split-asm*)
from *Cons.prem*s(1)[*unfolded mv-left(3), THEN Cons-F-DF*] **have** $\langle (u', X)$
 $\in \mathcal{F} (DF A) \rangle$.
from *Cons.hyps*[*OF this Cons.prem*s(2) *mv-left(4)*] **show** *?thesis*
by (*subst F-DF*) (*simp add: nonempty* $\langle e = ev a \rangle \langle a \in A \rangle$)
next
case *mv-right*
from *Cons.prem*s(2) **have** $\langle b \in B \rangle$
by (*subst (asm) F-DF*) (*simp add: mv-right(3) split: if-split-asm*)
from *Cons.prem*s(2)[*unfolded mv-right(3), THEN Cons-F-DF*] **have** $\langle (v', Y)$
 $\in \mathcal{F} (DF B) \rangle$.
from *Cons.hyps*[*OF Cons.prem*s(1) *this mv-right(4)*] **show** *?thesis*
by (*subst F-DF*) (*simp add: nonempty* $\langle e = ev b \rangle \langle b \in B \rangle$)
next
case *mv-both-ev*
from *Cons.prem*s(1) **have** $\langle a \in A \rangle$

by (subst (asm) F-DF) (simp add: mv-both-ev(3) split: if-split-asm)
 from Cons.prem(1)[unfolded mv-both-ev(3), THEN Cons-F-DF]
 Cons.prem(2)[unfolded mv-both-ev(4), THEN Cons-F-DF]
 have $\langle (u', X) \in \mathcal{F} (DF A) \rangle \langle (v', Y) \in \mathcal{F} (DF B) \rangle$.
 from Cons.hyps[OF this mv-both-ev(5)] show ?thesis
 by (subst F-DF) (simp add: nonempty $\langle e = ev a \rangle \langle a \in A \rangle$)
 next
 case mv-both-tick
 from Cons.prem(1) have False
 by (subst (asm) F-DF) (simp add: mv-both-tick(3) split: if-split-asm)
 thus ?thesis ..
 qed
 qed
 thus $\langle DF (A \cup B) \sqsubseteq_F DF A \llbracket S \rrbracket_{\checkmark} DF B \rangle$
 by (simp add: failure-refine-def F-Sync_{ptick} div-free-DF)
 (use is-processT4 in blast)
 qed

theorem (in Sync_{ptick}-locale) DF-F-DF-Sync_{ptick}-DF :
 $\langle DF (A \cup B) \sqsubseteq_F DF A \llbracket S \rrbracket_{\checkmark} DF B \rangle$ 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 –
 from that(3) consider $\langle A \cap S = \{\} \vee (\exists a. A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \rangle$
 | $\langle B \cap S = \{\} \vee (\exists b. B \cap S = \{b\} \wedge A \cap S \subseteq \{b\}) \rangle$ by metis
 thus $\langle DF (A \cup B) \sqsubseteq_F DF A \llbracket S \rrbracket_{\checkmark} DF B \rangle$
proof cases
 from that(1, 2) show $\langle B \cap S = \{\} \vee (\exists b. B \cap S = \{b\} \wedge A \cap S \subseteq \{b\}) \implies$
 $DF (A \cup B) \sqsubseteq_F DF A \llbracket S \rrbracket_{\checkmark} DF B \rangle$
 by (rule DF-F-DF-Sync_{ptick}-DF-weak)
 next
 from that(1, 2) show $\langle A \cap S = \{\} \vee (\exists a. A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \implies$
 $DF (A \cup B) \sqsubseteq_F DF A \llbracket S \rrbracket_{\checkmark} DF B \rangle$
 by (fold Sync_{ptick}-sym, subst Un-commute)
 (simp add: Sync_{ptick}-locale-sym.DF-F-DF-Sync_{ptick}-DF-weak)
 qed
 qed

lemma (in Sync_{ptick}-locale) DF-FD-DF-Sync_{ptick}-DF :
 $\langle DF (A \cup B) \sqsubseteq_{FD} DF A \llbracket S \rrbracket_{\checkmark} DF B \rangle$ if $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle$
 and $\langle A \cap S = \{\} \vee (\exists a. A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \vee$
 $B \cap S = \{\} \vee (\exists b. B \cap S = \{b\} \wedge A \cap S \subseteq \{b\}) \rangle$
 using DF-F-DF-Sync_{ptick}-DF[OF that]
 by (simp add: refine-defs div-free-DF D-Sync_{ptick})

theorem (in Sync_{ptick}-locale) DF-FD-DF-Sync_{ptick}-DF-iff:

$\langle DF (A \cup B) \sqsubseteq_{FD} DF A \llbracket S \rrbracket_{\checkmark} DF B \longleftrightarrow$
 $($
 $\quad \text{if } A = \{\} \text{ then } B \cap S = \{\}$
 $\quad \text{else if } B = \{\} \text{ then } A \cap S = \{\}$
 $\quad \text{else } A \cap S = \{\} \vee (\exists a. A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \vee$
 $\quad \quad B \cap S = \{\} \vee (\exists b. B \cap S = \{b\} \wedge A \cap S \subseteq \{b\})) \rangle$
 $(\text{is } \langle ?FD\text{-ref} \longleftrightarrow ($
 $\quad \text{if } A = \{\} \text{ then } B \cap S = \{\}$
 $\quad \text{else if } B = \{\} \text{ then } A \cap S = \{\}$
 $\quad \text{else } ?cases) \rangle)$

proof –

$\{$ **assume** $\langle A \neq \{\} \rangle$ **and** $\langle B \neq \{\} \rangle$ **and** $?FD\text{-ref}$ **and** $\langle \neg ?cases \rangle$
from $\langle \neg ?cases \rangle [simplified]$
obtain a **and** b **where** $\langle a \in A \rangle \langle a \in S \rangle \langle b \in B \rangle \langle b \in S \rangle \langle a \neq b \rangle$ **by** *blast*
have $\langle DF A \llbracket S \rrbracket_{\checkmark} DF B \sqsubseteq_{FD} (a \rightarrow DF A) \llbracket S \rrbracket_{\checkmark} (b \rightarrow DF B) \rangle$
by (*intro mono-Sync_{ptick}-FD; subst DF-unfold, meson Mndetprefix-FD-write0*
 $\langle a \in A \rangle \langle b \in B \rangle)$
also have $\langle \dots = STOP \rangle$ **by** (*simp add: $\langle a \in S \rangle \langle a \neq b \rangle \langle b \in S \rangle$ write0-Sync_{ptick}-write0-subset*)
finally have *False*
by (*metis DF-Univ-freeness Un-empty $\langle A \neq \{\} \rangle$*
 $trans\text{-}FD[OF \langle ?FD\text{-ref} \rangle] non\text{-}deadlock\text{-}free\text{-}STOP)$
 $\}$ **note** $*$ = *this*
show $?thesis$
proof (*cases $\langle A = \{\} \rangle$; cases $\langle B = \{\} \rangle$*)
show $\langle A = \{\} \implies B = \{\} \implies ?thesis \rangle$ **by** *simp*
next
show $\langle A = \{\} \implies B \neq \{\} \implies ?thesis \rangle$
by *simp (metis DF-FD-STOP-Sync_{ptick}-DF-imp-disjoint*
 $disjoint\text{-}imp\text{-}DF\text{-eq}\text{-}STOP\text{-}Sync_{ptick}\text{-}DF\text{-order-refl})$
next
show $\langle A \neq \{\} \implies B = \{\} \implies ?thesis \rangle$
by *simp (metis DF-FD-DF-Sync_{ptick}-STOP-imp-disjoint*
 $disjoint\text{-}imp\text{-}DF\text{-eq}\text{-}DF\text{-Sync}_{ptick}\text{-}STOP\text{-order-refl})$
next
show $\langle A \neq \{\} \implies B \neq \{\} \implies ?thesis \rangle$
by *simp (metis * DF-FD-DF-Sync_{ptick}-DF)*
qed
qed

lemma *DF-FD-DF-MultiSync_{ptick}-DF* :

$\langle \llbracket \bigwedge l. l \in set\ L \implies X\ l \neq \{\}; \exists s. (\bigcup l \in set\ L. X\ l) \cap S \subseteq \{s\} \rrbracket$
 $\implies DF (\bigcup l \in set\ L. X\ l) \sqsubseteq_{FD} \llbracket S \rrbracket_{\checkmark} l \in @\ L. (DF (X\ l) :: ('a, 'r)\ process_{ptick}) \rangle$

proof (*induct L rule: induct-list012*)

case 1 **show** $?case$ **by** *simp*

next

case (2 l0) **show** $?case$ **by** (*simp add: Renaming-DF*)

next

case (3 l0 l1 L)

have $\langle (DF (\bigcup l \in set\ (l0 \# l1 \# L). X\ l) :: ('a, 'r)\ list)\ process_{ptick} \rangle =$

$DF (X \ l0 \cup (\bigcup l \in set (l1 \# L). X \ l)) \rangle$ **by** *simp*
also have $\langle \dots \sqsubseteq_{FD} DF (X \ l0) \llbracket S \rrbracket_{\checkmark}^{Rlist} DF (\bigcup l \in set (l1 \# L). X \ l) \rangle$
by (*rule Sync_{Rlist}.DF-FD-DF-Sync_{ptick}-DF-iff [THEN iffD2]*)
(use 3.premis(2) in <simp add: 3.premis(1) subset-singleton-iff
Int-Un-distrib2 Un-singleton-iff, safe, simp-all>)
also have $\langle \dots \sqsubseteq_{FD} DF (X \ l0) \llbracket S \rrbracket_{\checkmark}^{Rlist} \llbracket S \rrbracket_{\checkmark}^{l \in @ (l1 \# L). (DF (X \ l))} \rangle$
by (*intro Sync_{Rlist}.mono-Sync_{ptick}-FD [OF idem-FD] 3.hyps(2)*)
(use 3.premis in auto)
also have $\langle \dots = \llbracket S \rrbracket_{\checkmark}^{l \in @ (l0 \# l1 \# L). DF (X \ l)} \rangle$ **by** *simp*
finally show *?case .*
qed

lemma (*in Sync_{ptick}-locale*) $\langle DF \{a\} = DF \{a\} \llbracket S \rrbracket_{\checkmark} STOP \longleftrightarrow a \notin S \rangle$
by (*metis DF-FD-DF-Sync_{ptick}-STOP-imp-disjoint boolean-algebra.conj-zero-left*
disjoint-imp-DF-eq-DF-Sync_{ptick}-STOP insert-disjoint(1) order-refl)

lemma (*in Sync_{ptick}-locale*) $\langle DF \{a\} \llbracket S \rrbracket_{\checkmark} STOP = STOP \longleftrightarrow a \in S \rangle$
by (*metis DF-unfold Diff-eq-empty-iff Diff-triv Int-empty-left Int-insert-left*
Mndetprefix-Sync_{ptick}-Mprefix-right Mndetprefix-Sync_{ptick}-STOP
Mndetprefix-is-STOP-iff Mprefix-empty empty-not-insert insert-Diff1)

corollary (*in Sync_{ptick}-locale*) $DF-FD-DF-Inter_{ptick}-DF : \langle DF \ A \sqsubseteq_{FD} DF \ A \llbracket \checkmark \rrbracket DF \ A \rangle$
by (*metis DF-FD-DF-Sync_{ptick}-DF-iff inf-bot-right sup.idem*)

corollary (*in Sync_{ptick}-locale*) $DF-UNIV-FD-DF-UNIV-Inter_{ptick}-DF-UNIV :$
 $\langle DF \ UNIV \sqsubseteq_{FD} DF \ UNIV \llbracket \checkmark \rrbracket DF \ UNIV \rangle$
by (*fact DF-FD-DF-Inter_{ptick}-DF*)

corollary (*in Sync_{ptick}-locale*) $Inter_{ptick}-deadlock-free :$
 $\langle deadlock-free \ P \implies deadlock-free \ Q \implies deadlock-free \ (P \llbracket \checkmark \rrbracket Q) \rangle$
using $DF-FD-DF-Inter_{ptick}-DF \ deadlock-free-of-Sync_{ptick}-iff-DF-FD-DF-Sync_{ptick}-DF$
by *blast*

theorem *MultiInter_{ptick}-deadlock-free :*
 $\langle \llbracket L \neq [] ; \bigwedge l \in set \ L \implies deadlock-free \ (P \ l) \rrbracket \implies$
 $deadlock-free \ (\llbracket \checkmark \rrbracket l \in @ \ L. \ P \ l) \rangle$

proof (*induct L rule: induct-list012*)

case 1

from 1.premis(1) **have** *False* **by** *simp*

thus *?case ..*

next

case (2 *l0*)

```

from 2.prems(2) show ?case
  by (simp add: deadlock-free-imp-deadlock-free-Renaming)
next
  case (3 l0 l1 L)
  have  $\langle ||| \checkmark l \in @ (l0 \# l1 \# L). P \ l = P \ l0 \ ||| \checkmark_{Rlist} \ ||| \checkmark l \in @ (l1 \# L). P \ l \rangle$ 
by simp
  moreover have  $\langle \text{deadlock-free } (P \ l0) \rangle$  by (simp add: 3.prems(2))
  moreover have  $\langle \text{deadlock-free } (||| \checkmark l \in @ (l1 \# L). P \ l) \rangle$ 
    by (rule 3.hyps(2)) (simp-all add: 3.prems(2))
  ultimately show ?case
    by (simp add: SyncRlist.Interptick-deadlock-free)
qed

```


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 $(\lambda a. \lambda r) \text{ process}_{ptick} \Rightarrow (\lambda r \Rightarrow (\lambda a. \lambda s) \text{ process}_{ptick}) \Rightarrow (\lambda a. \lambda s) \text{ process}_{ptick}$, 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.

$$P ;_{\checkmark} \text{SKIP} = P \quad \text{SKIP } r ;_{\checkmark} Q = Q \ r$$

Additionally, with the following associativity property :

$$P ;_{\checkmark} (\lambda r. Q \ r ;_{\checkmark} R) = P ;_{\checkmark} Q ;_{\checkmark} R$$

we can conclude that this generalized sequential composition fulfills the monad laws.

Unsurprisingly, the correspondence with classical version is very intuitive.

$$P ;_{\checkmark} (\lambda r. Q) = P ; Q$$

The expected step law has also been established.

$$\Box a \in A \rightarrow P \ a ;_{\checkmark} Q = \Box a \in A \rightarrow (P \ a ;_{\checkmark} Q)$$

Additionally, in the same way as described in [4], operational laws have been derived.

$$\frac{\frac{P \ \alpha \rightsquigarrow_{\tau} P'}{P ;_{\checkmark} Q \ \beta \rightsquigarrow_{\tau} P' ;_{\checkmark} Q} \quad \frac{a \ \alpha \rightsquigarrow_P P'}{a ;_{\checkmark} Q \ \beta \rightsquigarrow_P P' ;_{\checkmark} Q}}{\frac{r \ \alpha \rightsquigarrow_{\checkmark P} P'}{r ;_{\checkmark} Q \ \beta \rightsquigarrow_{\tau} Q'}}$$

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.

$$\text{SEQ}_{\checkmark} \ l \in @ (L1 \ @ \ L2). \ P \ l = (\lambda r. (\text{SEQ}_{\checkmark} \ l \in @ \ L1. \ P \ l) \ r ;_{\checkmark} \text{SEQ}_{\checkmark} \ l \in @ \ L2. \ P \ l)$$

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).

$$SKIP\ r\ \llbracket A \rrbracket\ SKIP\ s = (if\ r = s\ then\ SKIP\ r\ else\ STOP)$$

With the new definition, this is not the case anymore.

$$SKIP\ r\ \llbracket A \rrbracket_{\checkmark} SKIP\ s = (case\ r \otimes \checkmark\ s\ of\ None \Rightarrow STOP \mid Some\ r-s \Rightarrow SKIP\ r-s)$$

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.

$$\begin{aligned} SKIP\ r\ \llbracket A \rrbracket_{\checkmark Pair} SKIP\ s &= SKIP\ (r, s) \\ SKIP\ r\ \llbracket A \rrbracket_{\checkmark Pairlist} SKIP\ s &= SKIP\ [r, s] \\ SKIP\ r\ \llbracket A \rrbracket_{\checkmark Rlist} SKIP\ s &= SKIP\ (r \cdot s) \\ SKIP\ r\ \llbracket A \rrbracket_{\checkmark Llist} SKIP\ s &= SKIP\ (r @ [s]) \\ SKIP\ r\ n\ \llbracket A \rrbracket_{\checkmark ListslenL} SKIP\ s &= (if\ |r| = n\ then\ SKIP\ (r @ s)\ else\ STOP) \\ SKIP\ r\ n\ \llbracket A \rrbracket_{\checkmark ListslenR} SKIP\ s &= (if\ |s| = n\ then\ SKIP\ (r @ s)\ else\ STOP) \\ SKIP\ r\ n\ \llbracket A \rrbracket_{\checkmark m} SKIP\ s &= (if\ |r| = n \wedge |s| = m\ then\ SKIP\ (r @ s)\ else\ STOP) \\ SKIP\ r\ \llbracket A \rrbracket_{\checkmark Classic} SKIP\ s &= (if\ r = s\ then\ SKIP\ r\ else\ STOP) \end{aligned}$$

Moreover, the last declension is proved to be equal to the old version, ensuring that this work is actually a generalization.

$$P\ \llbracket A \rrbracket_{\checkmark Classic}\ Q = P\ \llbracket A \rrbracket\ Q$$

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.

$$\begin{aligned} RenamingTick\ (P\ \llbracket A \rrbracket_{\checkmark Pair}\ Q)\ prod.swap &= Q\ \llbracket A \rrbracket_{\checkmark Pair}\ P \\ P\ \llbracket A \rrbracket_{\checkmark Pair}\ (Q\ \llbracket A \rrbracket_{\checkmark Pair}\ R) &= \\ RenamingTick\ (P\ \llbracket A \rrbracket_{\checkmark Pair}\ Q\ \llbracket A \rrbracket_{\checkmark Pair}\ R)\ (\lambda((r, s), t). (r, s, t)) \end{aligned}$$

Again, the expected step law has been established.

$$\Box a \in A \rightarrow P \ a ;\checkmark Q = \Box a \in A \rightarrow (P \ a ;\checkmark Q)$$

In this abstract setup, the operational laws have also been derived.

$$\begin{array}{c}
\frac{P \text{ lhs} \rightsquigarrow_{\tau} P'}{P \llbracket A \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_{\tau} P' \llbracket A \rrbracket_{\checkmark} Q} \quad \frac{Q \text{ rhs} \rightsquigarrow_{\tau} Q'}{A \llbracket P \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_{\tau} A \llbracket P \rrbracket_{\checkmark} Q'} \\
\frac{a \notin A \quad P \text{ lhs} \rightsquigarrow_a P'}{P \llbracket A \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_a P' \llbracket A \rrbracket_{\checkmark} Q} \quad \frac{a \notin A \quad Q \text{ rhs} \rightsquigarrow_a Q'}{P \llbracket A \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_a P \llbracket A \rrbracket_{\checkmark} Q'} \\
\frac{a \in A \quad P \text{ lhs} \rightsquigarrow_a P' \quad Q \text{ rhs} \rightsquigarrow_a Q'}{P \llbracket A \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_a P' \llbracket A \rrbracket_{\checkmark} Q'} \\
\frac{P \text{ lhs} \rightsquigarrow_{\checkmark r} P'}{P \llbracket A \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_{\tau} \text{SKIP } r \llbracket A \rrbracket_{\checkmark} Q} \\
\frac{Q \text{ rhs} \rightsquigarrow_{\checkmark s} Q'}{P \llbracket A \rrbracket_{\checkmark} Q \text{ ptick} \rightsquigarrow_{\tau} P \llbracket A \rrbracket_{\checkmark} \text{SKIP } s} \\
\frac{r \otimes \checkmark s = \text{Some } r-s}{\text{SKIP } r \llbracket A \rrbracket_{\checkmark} \text{SKIP } s \text{ ptick} \rightsquigarrow_{\checkmark r-s} \Omega_{\text{ptick}} (\text{SKIP } r-s) \ r-s}
\end{array}$$

Continuity and non-destructiveness hold in general, and an architectural version is defined. It satisfies the following property.

$$\frac{L1 \neq [] \quad L2 \neq []}{\llbracket S \rrbracket_{\checkmark} l \in @ (L1 @ L2). P \ l = \llbracket S \rrbracket_{\checkmark} l \in @ L1. P \ l \mid_{|L1|} \llbracket S \rrbracket_{\checkmark} l \in @ L2. P \ l}$$

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.

$$\frac{f \text{ permutes } \{..<|L|\}}{\llbracket S \rrbracket_{\checkmark} l \in @ \text{permute-list } f \ L. P \ l = \text{RenamingTick } (\llbracket S \rrbracket_{\checkmark} l \in @ \ L. P \ l) \ (\text{permute-list } f)}$$

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.