# HOL-CSP_PTick
# Parameterized Termination for Sequential Composition and Synchronization Product

Benoît Ballenghien

February 3, 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

# Chapter 1

# Introduction

## 1.1 Motivations

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

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

(isomorphic to option type) by

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

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

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

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

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

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

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

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

## 1.2 The Global Architecture of HOL-CSP_PTick

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

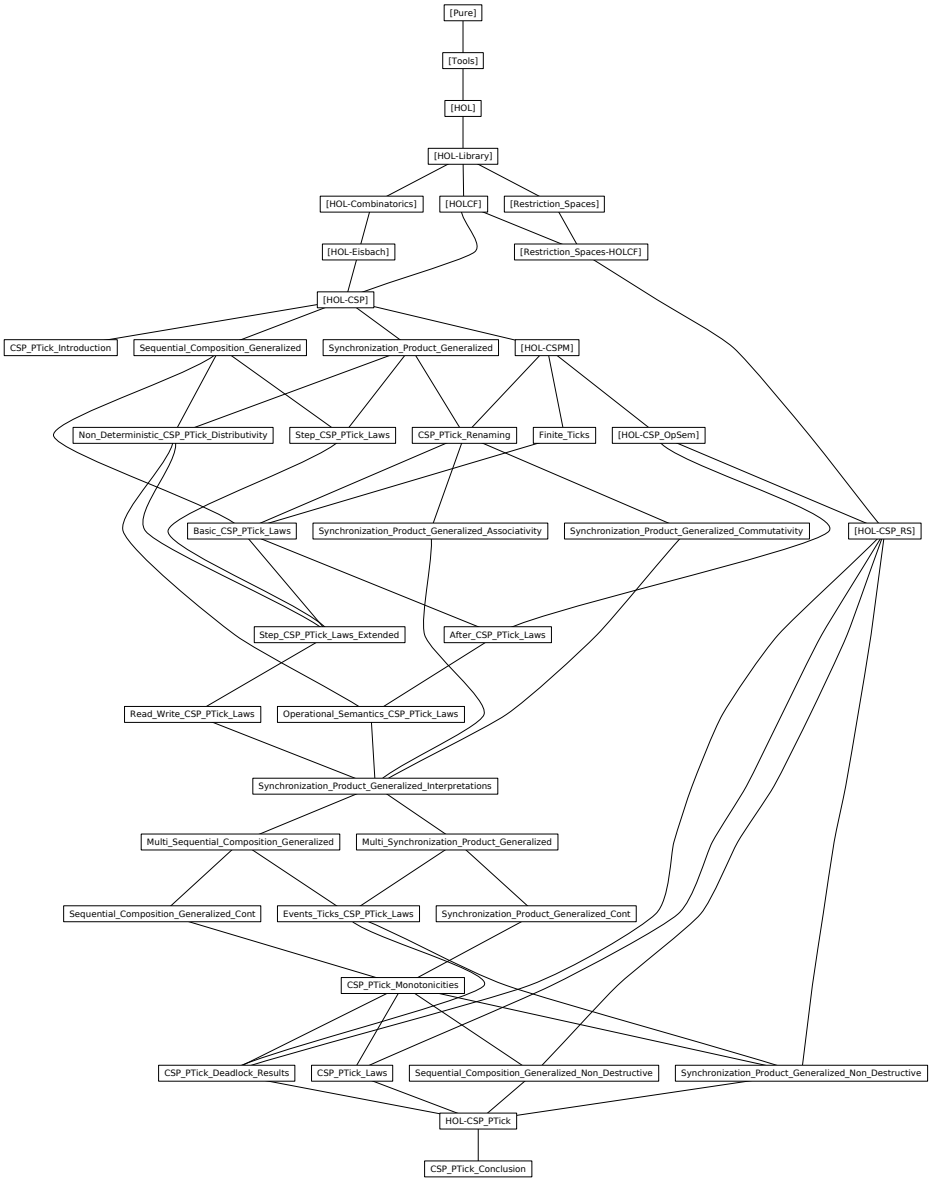


Figure 1.1: The overall architecture

# Chapter 2

# Finite Ticks Predicate

## 2.1 Definitions

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

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

**lemma** *finite-all-ticksI* : ‹(⋀*t*. *t* ∈ $\mathcal{T}$ *P* ⟹ *finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}) ⟹ *finite-all-ticks P*›
  **by** (*simp add*: *finite-all-ticks-def*)

**lemma** *finite-all-ticksD* : ‹*finite-all-ticks P* ⟹ *finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}›
  **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* :: ‹('a, 'r) $process_{ptick}$ ⇒ *bool*› (‹$\mathbb{F}_{✓}$'(-')›)
  **where** ‹$\mathbb{F}_{✓}$(*P*) ≡ ∀ *t* ∈ $\mathcal{T}$ *P*. *finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ *P* − $\mathcal{D}$ *P*}›

**lemma** *finite-ticksI* :
  ‹(⋀*t*. *t* ∈ $\mathcal{T}$ *P* ⟹ *t* ∉ $\mathcal{D}$ *P* ⟹ *finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}) ⟹ $\mathbb{F}_{✓}$(*P*)›
  **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* :
  ‹$\mathbb{F}_{✓}$(*P*) ⟹ *t* ∉ $\mathcal{D}$ *P* ⟹ *finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}›
  **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*] : ‹*finite-all-ticks* $P \Longrightarrow \mathbb{F}_{\checkmark}(P)$›
**by** (*simp add*: *finite-all-ticksD finite-ticksI*)

**lemma** *finite-all-ticks-is-finite-ticks-or-finite-UNIV* :
‹*finite-all-ticks* $P \longleftrightarrow$ (*if* $\mathcal{D}\ P = \{\}$ *then* $\mathbb{F}_{\checkmark}(P)$ *else finite* (*UNIV* :: *'r set*))›
— This is justifying why *finite-all-ticks* is not really interesting.
**for** $P$ :: ‹(*'a*, *'r*) $process_{ptick}$›
**proof** (*rule iffI*)
**show** ‹*if* $\mathcal{D}\ P = \{\}$ *then* $\mathbb{F}_{\checkmark}(P)$ *else finite* (*UNIV* :: *'r set*)›
**if** ‹*finite-all-ticks* $P$›
**proof** (*split if-split*, *intro conjI impI*)
**from** ‹*finite-all-ticks* $P$› **show** ‹$\mathcal{D}\ P = \{\} \Longrightarrow \mathbb{F}_{\checkmark}(P)$›
**by** (*simp add*: *finite-ticksI finite-all-ticksD*)
**next**
**assume** ‹$\mathcal{D}\ P \neq \{\}$›
**with** *nonempty-divE* **obtain** $t$ **where** ‹*tF* $t$› ‹$t \in \mathcal{D}\ P$› **by** *blast*
**hence** ‹$t\ @\ [\checkmark(r)] \in \mathcal{D}\ P$› **for** $r$ **by** (*simp add*: *is-processT7*)
**with** ‹*finite-all-ticks* $P$› **show** ‹*finite* (*UNIV* :: *'r set*)›
**by** (*metis* (*mono-tags*, *lifting*) *Collect-cong D-T UNIV-I* ‹$t \in \mathcal{D}\ P$›
*finite-all-ticks-def mem-Collect-eq top-set-def*)
**qed**
**next**
**show** ‹*if* $\mathcal{D}\ P = \{\}$ *then* $\mathbb{F}_{\checkmark}(P)$ *else finite* (*UNIV* :: *'r set*) $\Longrightarrow$ *finite-all-ticks* $P$›
**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* :: ‹((*'a*, *'r*) $process_{ptick}$ ⇒ (*'b*, *'s*) $process_{ptick}$) ⇒ *bool*›
(‹$\mathbb{F}_{\checkmark\Rightarrow}$'(-')›)
**where** ‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \equiv \forall P.\ \mathbb{F}_{\checkmark}(P) \longrightarrow \mathbb{F}_{\checkmark}(f\ P)$›

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

**lemma** *finite-ticks-funD*: ‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark}(P) \Longrightarrow \mathbb{F}_{\checkmark}(f\ P)$›
**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*] : ‹$\mathbb{F}_{\checkmark}(\bot)$›
**by** (*simp add*: *finite-ticks-def BOT-projs*)

**lemma** *finite-ticks-fun-BOT* [*finite-ticks-fun-simps*] : ‹$\mathbb{F}_{\checkmark\Rightarrow}(\bot)$›
**by** (*simp add*: *finite-ticks-fun-def finite-ticks-BOT*)

**lemma** *finite-ticks-SKIP* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(SKIP\ r)$›
**by** (*simp add*: *finite-ticks-def SKIP-projs*)

**lemma** *finite-ticks-STOP* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(STOP)$›
**by** (*simp add*: *finite-ticks-def T-STOP*)

**lemma** *finite-ticks-SKIPS-iff* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(SKIPS\ R) \longleftrightarrow$ *finite* $R$›
**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*] :
‹*finite* $\checkmark\mathbf{s}(P) \Longrightarrow \mathbb{F}_{\checkmark}(P)$›
**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*] :
‹$(\bigwedge x.$ *finite* $\checkmark\mathbf{s}(f\ x)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f)$›
**by** (*simp add*: *finite-strict-ticks-of-imp-finite-ticks finite-ticks-funI*)

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

**lemma** *anti-mono-finite-ticks-fun* [*finite-ticks-fun-simps*] :
‹$f \sqsubseteq g \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f)$›
**by** (*metis anti-mono-finite-ticks finite-ticks-fun-def fun-below-iff*)

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

‹$\mathbb{F}_{\checkmark}(\bigsqcup i.\ Y\ i) \longleftrightarrow (\forall\, i.\ \mathbb{F}_{\checkmark}(Y\ i))$› **if** ‹*chain* $Y$›
**proof** *safe*
**from** *anti-mono-finite-ticks is-ub-thelub* ‹*chain* $Y$›
**show** ‹$\mathbb{F}_{\checkmark}(\bigsqcup i.\ Y\ i) \Longrightarrow \mathbb{F}_{\checkmark}(Y\ i)$› **for** $i$ **by** *blast*
**next**
**show** ‹$\mathbb{F}_{\checkmark}(\bigsqcup i.\ Y\ i)$› **if** ‹$\forall\, i.\ \mathbb{F}_{\checkmark}(Y\ i)$›
**proof** (*rule finite-ticksI*)
**fix** $t$ **assume** ‹$t \in \mathcal{T}\ (\bigsqcup i.\ Y\ i)$› ‹$t \notin \mathcal{D}\ (\bigsqcup i.\ Y\ i)$›
**from** ‹$t \notin \mathcal{D}\ (\bigsqcup i.\ Y\ i)$› **obtain** $j$ **where** ‹$t \notin \mathcal{D}\ (Y\ j)$›
**by** (*metis D-LUB-2* ‹*chain* $Y$› *limproc-is-thelub*)
**have** ‹$\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (\bigsqcup i.\ Y\ i)\} = \{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (Y\ j)\}$›
**by** (*meson* ‹*chain* $Y$› ‹$t \notin \mathcal{D}\ (Y\ j)$› *is-processT9 is-ub-thelub le-approx2T*)
**also have** ‹*finite* ...›
**by** (*fact* ‹$\forall\, i.\ \mathbb{F}_{\checkmark}(Y\ i)$›[*THEN spec*, *THEN finite-ticksD*, *OF* ‹$t \notin \mathcal{D}\ (Y\ j)$›])
**finally show** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (\bigsqcup i.\ Y\ i)\}$› **.**
**qed**
**qed**

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

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

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

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

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

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

**lemma** *finite-ticks-Det* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P \mathbin{\square} Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P)$› ‹$\mathbb{F}_{\checkmark}(Q)$›
**proof** −
**have** ‹$\mathbb{F}_{\checkmark}(P \mathbin{\square} Q) = \mathbb{F}_{\checkmark}(P \sqcap Q)$› **by** (*simp add*: *finite-ticks-def Det-projs Ndet-projs*)
**with** ‹$\mathbb{F}_{\checkmark}(P)$› ‹$\mathbb{F}_{\checkmark}(Q)$› **show** ‹$\mathbb{F}_{\checkmark}(P \mathbin{\square} Q)$› **by** (*simp add*: *finite-ticks-Ndet*)
**qed**

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

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

**show** ‹$Q = \bot \Longrightarrow \mathbb{F}_{\checkmark}(P \triangle Q)$› **by** (*simp add*: *finite-ticks-BOT*)
**next**
**show** ‹$\mathbb{F}_{\checkmark}(P \triangle Q)$› **if** ‹$Q \neq \bot$›
**proof** (*rule finite-ticksI*)
**fix** $t$ **assume** ‹$t \in \mathcal{T}\ (P \triangle Q)$› ‹$t \notin \mathcal{D}\ (P \triangle Q)$›
**have** ‹$\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P \triangle Q)\} \subseteq$
$\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\} \cup$
$(\bigcup u \in \{u.\ \exists v.\ t = u\ @\ v \wedge u \in \mathcal{T}\ P\}.\ \{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\})$›
**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 butlast-snoc*)
**moreover have** ‹*finite* ...›
**proof** (*rule finite-UnI*)
**from** *D-Interrupt* ‹$t \notin \mathcal{D}\ (P \triangle Q)$› **have** ‹$t \notin \mathcal{D}\ P$› **by** *blast*
**thus** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$› **by** (*simp add*: *finite-ticksD* ‹$\mathbb{F}_{\checkmark}(P)$›)
**next**
**show** ‹*finite* $(\bigcup u \in \{u.\ \exists v.\ t = u\ @\ v \wedge u \in \mathcal{T}\ P\}.\ \{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\})$›
**proof** (*rule finite-UN-I*)
**show** ‹*finite* $\{u.\ \exists v.\ t = u\ @\ v \wedge u \in \mathcal{T}\ P\}$› **by** (*prove-finite-subset-of-prefixes t*)
**next**
**fix** $u$ **assume** ‹$u \in \{u.\ \exists v.\ t = u\ @\ v \wedge u \in \mathcal{T}\ P\}$›
**then obtain** $v$ **where** ‹$u \in \mathcal{T}\ P$› ‹$t = u\ @\ v$› **by** *blast*
**with** ‹$t \in \mathcal{T}\ (P \triangle Q)$› *append-T-imp-tickFree* **consider** ‹$tF\ u$› | ‹$v = []$› **by** *blast*
**thus** ‹*finite* $\{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\}$›
**proof** *cases*
**assume** ‹$tF\ u$›
**with** ‹$u \in \mathcal{T}\ P$› ‹$t \notin \mathcal{D}\ (P \triangle Q)$› **have** ‹$v \notin \mathcal{D}\ Q$›
**by** (*simp add*: *D-Interrupt* ‹$t = u\ @\ v$›)
**thus** ‹$tF\ u \Longrightarrow$ *finite* $\{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\}$›
**by** (*simp add*: ‹$t = u\ @\ v$› *finite-ticksD* ‹$\mathbb{F}_{\checkmark}(Q)$›)
**next**
**from** *BOT-iff-Nil-D* ‹$Q \neq \bot$› **have** ‹$[] \notin \mathcal{D}\ Q$› **by** *blast*
**with** ‹$\mathbb{F}_{\checkmark}(Q)$› *finite-ticksD* **have** ‹*finite* $\{r.\ [\checkmark(r)] \in \mathcal{T}\ Q\}$› **by** *force*
**thus** ‹$v = [] \Longrightarrow$ *finite* $\{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\}$›
**by** (*simp add*: ‹$t = u\ @\ v$›)
**qed**
**qed**
**qed**
**ultimately show** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P \triangle Q)\}$› **by** (*fact finite-subset*)
**qed**
**qed**

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

```
  ‹𝔽✓(P ⊖ a∈A. Q a)› if ‹𝔽✓(P)› ‹⋀a. a ∈ A ⟹ 𝔽✓(Q a)›
proof (rule finite-ticksI)
  fix t assume ‹t ∈ 𝒯 (P ⊖ a∈A. Q a)› ‹t ∉ 𝒟 (P ⊖ a∈A. Q a)›
  then consider ‹t ∈ 𝒯 P› ‹set t ∩ ev ‘ A = {}›
    | t1 a t2 where ‹t = t1 @ ev a # t2› ‹t1 @ [ev a] ∈ 𝒯 P›
      ‹set t1 ∩ ev ‘ A = {}› ‹a ∈ A› ‹t2 ∈ 𝒯 (Q a)›
    unfolding Throw-projs by blast
  thus ‹finite {r. t @ [✓(r)] ∈ 𝒯 (P ⊖ a∈A. Q a)}›
  proof cases
    assume ‹t ∈ 𝒯 P› ‹set t ∩ ev ‘ A = {}›
    hence ‹{r. t @ [✓(r)] ∈ 𝒯 (P ⊖ a∈A. Q a)} ⊆ {r. t @ [✓(r)] ∈ 𝒯 P}›
      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 ‹finite …›
    proof (rule ‹𝔽✓(P)›[THEN finite-ticksD])
      from ‹set t ∩ ev ‘ A = {}›
      have ‹t ∈ 𝒟 P ⟹ (if tF t then t else butlast t) ∈ 𝒟 (P ⊖ a∈A. Q a)›
        by (cases t rule: rev-cases, simp-all add: D-Throw)
           (metis D-imp-front-tickFree ‹set t ∩ ev ‘ A = {}› 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 ‹t ∉ 𝒟 (P ⊖ a∈A. Q a)› D-imp-front-tickFree div-butlast-when-non-tickFree-iff
      show ‹t ∉ 𝒟 P› by blast
    qed
     ultimately show ‹finite {r. t @ [✓(r)] ∈ 𝒯 (P ⊖ a∈A. Q a)}› by (fact
finite-subset)
  next
    fix t1 a t2 assume ∗ : ‹t = t1 @ ev a # t2› ‹t1 @ [ev a] ∈ 𝒯 P›
      ‹set t1 ∩ ev ‘ A = {}› ‹a ∈ A› ‹t2 ∈ 𝒯 (Q a)›
    from ‹t ∉ 𝒟 (P ⊖ a∈A. Q a)›
    have ‹t ∉ {t1 @ t2 |t1 t2. t1 ∈ 𝒟 P ∧ tF t1 ∧ set t1 ∩ ev ‘ A = {} ∧ ftF t2}›
      by (simp add: D-Throw UnI1)

    with ∗ have ‹{r. t @ [✓(r)] ∈ 𝒯 (P ⊖ a∈A. Q a)} = {r. t2 @ [✓(r)] ∈ 𝒯 (Q
a)}›
      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 ‹simp add: Throw-T-third-clause-breaker›, metis)
    also have ‹finite …›
    proof (rule ‹⋀a. a ∈ A ⟹ 𝔽✓(Q a)›[THEN finite-ticksD, OF ‹a ∈ A›])
      from ‹t ∉ 𝒟 (P ⊖ a∈A. Q a)› ‹t1 @ [ev a] ∈ 𝒯 P› ‹set t1 ∩ ev ‘ A = {}›
      show ‹t2 ∉ 𝒟 (Q a)› by (auto simp add: D-Throw ‹t = t1 @ ev a # t2› ‹a
∈ A›)
    qed
    finally show ‹finite {r. t @ [✓(r)] ∈ 𝒯 (P ⊖ a∈A. Q a)}› .
  qed
qed
```

**lemma** *finite-ticks-Renaming* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}$(*Renaming P f g*)› **if** ‹*finitary f*› ‹*finitary g*› ‹$\mathbb{F}_{\checkmark}$(*P*)›
**proof** (*rule finite-ticksI*)
**fix** *t* **assume** ‹$t \notin \mathcal{D}$ (*Renaming P f g*)›
**hence** ‹{*s*. *t* @ [✓(*s*)] ∈ $\mathcal{T}$ (*Renaming P f g*)} ⊆
($\bigcup u \in \{u.\ t = map\ (map\text{-}event_{ptick}\ f\ g)\ u \wedge u \in \mathcal{T}\ P\}.\ \{g\ r \mid r.\ u$ @ [✓(*r*)] ∈ $\mathcal{T}$ *P*})›
**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** ‹*finite* ...›
**proof** (*rule finite-UN-I*)
**have** ‹*finitary* (*map-event*$_{ptick}$ *f g*)› **by** (*simp add*: *Cont-RenH2* ‹*finitary f*› ‹*finitary g*›)
**have** ‹{*u*. *t* = *map* (*map-event*$_{ptick}$ *f g*) *u* ∧ *u* ∈ $\mathcal{T}$ *P*} ⊆ {*u*. *t* = *map* (*map-event*$_{ptick}$ *f g*) *u*}› **by** *blast*
**moreover from** *Cont-RenH4* ‹*finitary* (*map-event*$_{ptick}$ *f g*)› **have** ‹*finite* ...› **by** *blast*
**ultimately show** ‹*finite* {*u*. *t* = *map* (*map-event*$_{ptick}$ *f g*) *u* ∧ *u* ∈ $\mathcal{T}$ *P*}› **by** (*fact finite-subset*)
**next**
**fix** *u* **assume** ‹*u* ∈ {*u*. *t* = *map* (*map-event*$_{ptick}$ *f g*) *u* ∧ *u* ∈ $\mathcal{T}$ *P*}›
**hence** ‹*t* = *map* (*map-event*$_{ptick}$ *f g*) *u*› ‹*u* ∈ $\mathcal{T}$ *P*› **by** *simp-all*
**with** ‹*t* ∉ $\mathcal{D}$ (*Renaming P f g*)› **have** ‹*u* ∉ $\mathcal{D}$ *P*›
**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-charn map-butlast*
*map-event*$_{ptick}$*-tickFree snoc-eq-iff-butlast tickFree-Nil*)
**thus** ‹*finite* {*g r* |*r*. *u* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}›
**by** (*simp add*: *finite-ticksD* ‹$\mathbb{F}_{\checkmark}$(*P*)›)
**qed**
**ultimately show** ‹*finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ (*Renaming P f g*)}› **by** (*fact finite-subset*)
**qed**

**lemma** *finite-ticks-Seq* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}$(*P* ; *Q*)› **if** ‹$\mathbb{F}_{\checkmark}$(*Q*)›
**proof** (*cases* ‹*Q* = ⊥›)
**from** *not-finite-existsD* **show** ‹*Q* = ⊥ ⟹ $\mathbb{F}_{\checkmark}$(*P* ; *Q*)›
**by** (*auto simp add*: *finite-ticks-def Seq-projs BOT-projs*)
**next**
**show** ‹$\mathbb{F}_{\checkmark}$(*P* ; *Q*)› **if** ‹*Q* ≠ ⊥›
**proof** (*rule finite-ticksI*)
**fix** *t* **assume** ‹*t* ∉ $\mathcal{D}$ (*P* ; *Q*)›
**hence** ‹{*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ (*P* ; *Q*)} ⊆

$(\bigcup u \in \{u.\ \exists\, v\ r.\ t = u\ @\ v \wedge u\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}.\ \{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\})$›
**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** ‹*finite* ...›
**proof** (*rule finite-UN-I*)
**show** ‹*finite* $\{u.\ \exists\, v\ r.\ t = u\ @\ v \wedge u\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$›
**by** (*prove-finite-subset-of-prefixes t*)
**next**
**fix** *u* **assume** ‹$u \in \{u.\ \exists\, v\ r.\ t = u\ @\ v \wedge u\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$›
**then obtain** *v r* **where** ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$t = u\ @\ v$› **by** *blast*
**with** *append-T-imp-tickFree* **consider** ‹*tF u*› | ‹$v = []$› **by** *blast*
**thus** ‹*finite* $\{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\}$›
**proof** *cases*
**assume** ‹*tF u*›
**with** ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$t \notin \mathcal{D}\ (P\ ;;\ Q)$› **have** ‹$v \notin \mathcal{D}\ Q$›
**by** (*auto simp add*: *D-Seq* ‹$t = u\ @\ v$›)
**thus** ‹*tF u* $\Longrightarrow$ *finite* $\{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\}$›
**by** (*simp add*: ‹$t = u\ @\ v$› *finite-ticksD* ‹$\mathbb{F}_{\checkmark}(Q)$›)
**next**
**from** *BOT-iff-Nil-D* ‹$Q \neq \bot$› **have** ‹$[] \notin \mathcal{D}\ Q$› **by** *blast*
**with** ‹$\mathbb{F}_{\checkmark}(Q)$› *finite-ticksD* **have** ‹*finite* $\{r.\ [\checkmark(r)] \in \mathcal{T}\ Q\}$› **by** *force*
**thus** ‹$v = [] \Longrightarrow$ *finite* $\{r.\ drop\ (length\ u)\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ Q\}$›
**by** (*simp add*: ‹$t = u\ @\ v$›)
**qed**
**qed**
**ultimately show** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ ;;\ Q)\}$› **by** (*fact finite-subset*)
**qed**
**qed**

**lemma** *finite-ticks-Sync* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(P\ [\![S]\!]\ Q)$› **if** ‹$\mathbb{F}_{\checkmark}(P) \vee \mathbb{F}_{\checkmark}(Q)$›
**proof** (*rule finite-ticksI*)
**fix** *t* **assume** ‹$t \notin \mathcal{D}\ (P\ [\![S]\!]\ Q)$›
**have** ‹$\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ [\![S]\!]\ Q)\} \subseteq$
$(\bigcup (t\text{-}P,\ t\text{-}Q) \in \{(t\text{-}P,\ t\text{-}Q).\ t\ setinterleaves\ ((t\text{-}P,\ t\text{-}Q),\ range\ tick \cup ev\ `\ S)\}.$
$\{r.\ t\text{-}P\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge t\text{-}P \notin \mathcal{D}\ P \wedge t\text{-}Q\ @\ [\checkmark(r)] \in \mathcal{T}\ Q \wedge t\text{-}Q \notin \mathcal{D}\ Q\})$›
(**is** ‹- $\subseteq$ *?rhs*›)
**proof** (*rule subsetI*)
**fix** *r* **assume** ‹$r \in \{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ [\![S]\!]\ Q)\}$›
**hence** ‹$t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ [\![S]\!]\ Q)$› **..**
**moreover from** ‹$t \notin \mathcal{D}\ (P\ [\![S]\!]\ Q)$› **have** ‹$t\ @\ [\checkmark(r)] \notin \mathcal{D}\ (P\ [\![S]\!]\ Q)$›
**by** (*meson is-processT9*)
**ultimately obtain** *t-P t-Q* **where** ‹$t\text{-}P \in \mathcal{T}\ P$› ‹$t\text{-}Q \in \mathcal{T}\ Q$› ‹$t\text{-}P \notin \mathcal{D}\ P$›

‹*t-Q* ∉ $\mathcal{D}$ *Q*›
‹(*t* @ [✓(*r*)]) *setinterleaves* ((*t-P*, *t-Q*), *range tick* ∪ *ev* ‘ *S*)›
**by** (*simp add*: *Sync-projs*)
(*metis* (*no-types*, *lifting*) *append.right-neutral front-tickFree-Nil setinterleaving-sym*)
**from** *this*(*1*−*4*) *SyncWithTick-imp-NTF*[*OF this*(*5*)] **show** ‹*r* ∈ *?rhs*›
**by** *simp* (*metis T-imp-front-tickFree front-tickFree-append-iff is-processT7 not-Cons-self2*)
**qed**
**moreover have** ‹*finite* ...›
**proof** (*rule finite-UN-I*, *safe*)
**show** ‹*finite* {(*t-P*, *t-Q*). *t setinterleaves* ((*t-P*, *t-Q*), *range tick* ∪ *ev* ‘ *S*)}›
**by** (*fact finite-interleaves*)
**next**
**from** ‹$\mathbb{F}_{\checkmark}$(*P*) ∨ $\mathbb{F}_{\checkmark}$(*Q*)› *finite-ticksD*
**show** ‹*finite* {*r*. *t-P* @ [✓(*r*)] ∈ $\mathcal{T}$ *P* ∧ *t-P* ∉ $\mathcal{D}$ *P* ∧
*t-Q* @ [✓(*r*)] ∈ $\mathcal{T}$ *Q* ∧ *t-Q* ∉ $\mathcal{D}$ *Q*}› **for** *t-P t-Q* **by** *fastforce*
**qed**
**ultimately show** ‹*finite* {*r*. *t* @ [✓(*r*)] ∈ $\mathcal{T}$ (*P* ⟦*S*⟧ *Q*)}› **by** (*fact finite-subset*)
**qed**

**corollary** ‹$\mathbb{F}_{\checkmark}$(*P*) ∨ $\mathbb{F}_{\checkmark}$(*Q*) ⟹ $\mathbb{F}_{\checkmark}$(*P* || *Q*)›
**and** ‹$\mathbb{F}_{\checkmark}$(*P*) ∨ $\mathbb{F}_{\checkmark}$(*Q*) ⟹ $\mathbb{F}_{\checkmark}$(*P* ||| *Q*)›
**by** (*fact finite-ticks-Sync*)+

**lemma** *finite-ticks-GlobalNdet* [*finite-ticks-simps*] :
‹*finite A* ⟹ (⋀*a*. *a* ∈ *A* ⟹ $\mathbb{F}_{\checkmark}$(*P a*)) ⟹ $\mathbb{F}_{\checkmark}$(⊓*a*∈*A*. *P a*)›
— We can’t expect *infinite A* here, see $\mathbb{F}_{\checkmark}$(*SKIPS R*) = *finite R*.
**by** (*induct A rule*: *induct-subset-empty-single*)
(*simp-all add*: *GlobalNdet-distrib-unit finite-ticks-Ndet finite-ticks-STOP*)

**lemma** *finite-ticks-GlobalDet* [*finite-ticks-simps*] :
‹*finite A* ⟹ (⋀*a*. *a* ∈ *A* ⟹ $\mathbb{F}_{\checkmark}$(*P a*)) ⟹ $\mathbb{F}_{\checkmark}$(□*a*∈*A*. *P a*)›
**by** (*induct A rule*: *finite-induct*)
(*simp-all add*: *GlobalDet-distrib-unit-bis finite-ticks-Det finite-ticks-STOP*)

**lemma** ‹*L* = [] ⟹ $\mathbb{F}_{\checkmark}$(*SEQ l*∈@*L*. *P l*)› **by** (*simp add*: *finite-ticks-SKIP*)

**lemma** *finite-ticks-MultiSeq-nonempty* [*finite-ticks-simps*] :
‹*L* ≠ [] ⟹ $\mathbb{F}_{\checkmark}$(*P* (*last L*)) ⟹ $\mathbb{F}_{\checkmark}$(*SEQ l*∈@*L*. *P l*)›
**by** (*induct L rule*: *rev-induct*) (*simp-all add*: *finite-ticks-Seq*)

**lemma** *finite-ticks-MultiSync* [*finite-ticks-simps*] :
‹(⋀*m*. *m* ∈# *M* ⟹ $\mathbb{F}_{\checkmark}$(*P m*)) ⟹ $\mathbb{F}_{\checkmark}$(⟦*S*⟧ *m*∈#*M*. *P m*)›
**by** (*induct M rule*: *induct-subset-mset-empty-single*)
(*simp-all add*: *finite-ticks-Sync finite-ticks-STOP*)

**corollary** ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark}(P\ m)) \Longrightarrow \mathbb{F}_{\checkmark}(||\ \ m{\in}\#M.\ P\ m)$›
**and** ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark}(P\ m)) \Longrightarrow \mathbb{F}_{\checkmark}(|||\ m{\in}\#M.\ P\ m)$›
**by** (*fact finite-ticks-MultiSync*)+

**lemma** *finite-ticks-Mprefix-iff* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(\Box a{\in}A \rightarrow P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
**proof** (*safe intro*!: *finite-ticksI*)
**fix** *t a* **assume** ‹$\mathbb{F}_{\checkmark}(\Box a{\in}A \rightarrow P\ a)$› ‹$a \in A$› ‹$t \in \mathcal{T}\ (P\ a)$› ‹$t \notin \mathcal{D}\ (P\ a)$›
**have** ‹$\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ a)\} = \{r.\ (ev\ a\ \#\ t)\ @\ [\checkmark(r)] \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)\}$›
**by** (*auto simp add*: ‹$a \in A$› *T-Mprefix*)
**also have** ‹*finite* ...›
**by** (*rule* ‹$\mathbb{F}_{\checkmark}(\Box a{\in}A \rightarrow P\ a)$›[*THEN finite-ticksD*])
(*simp add*: *D-Mprefix* ‹$t \notin \mathcal{D}\ (P\ a)$›)
**finally show** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ a)\}$› **.**
**next**
**fix** *t* **assume** ‹$\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a)$› ‹$t \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)$› ‹$t \notin \mathcal{D}\ (\Box a{\in}A \rightarrow P\ a)$›
**from** ‹$t \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)$› **consider** ‹$t = []$› | *u a* **where** ‹$a \in A$› ‹$t = ev\ a\ \#\ u$›
**by** (*auto simp add*: *T-Mprefix*)
**thus** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)\}$›
**proof** *cases*
**show** ‹$t = [] \Longrightarrow$ *finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)\}$› **by** (*simp add*: *T-Mprefix*)
**next**
**fix** *u a* **assume** ‹$a \in A$› ‹$t = ev\ a\ \#\ u$›
**hence** ‹$\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)\} = \{r.\ u\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ a)\}$›
**by** (*simp add*: *set-eq-iff T-Mprefix*)
**also have** ‹*finite* ...›
**by** (*rule* ‹$\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a)$›[*THEN bspec*, *OF* ‹$a \in A$›, *THEN finite-ticksD*])
(*use* ‹$t \notin \mathcal{D}\ (\Box a{\in}A \rightarrow P\ a)$› **in** ‹*simp add*: ‹$t = ev\ a\ \#\ u$› *D-Mprefix* ‹$a \in A$››)
**finally show** ‹*finite* $\{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ (\Box a{\in}A \rightarrow P\ a)\}$› **.**
**qed**
**qed**

**lemma** *finite-ticks-Mndetprefix-iff* [*finite-ticks-simps*] :
‹$\mathbb{F}_{\checkmark}(\sqcap a{\in}A \rightarrow P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
**proof** −
**have** ‹$\mathbb{F}_{\checkmark}(\sqcap a{\in}A \rightarrow P\ a) \longleftrightarrow \mathbb{F}_{\checkmark}(\Box a{\in}A \rightarrow P\ a)$›
**by** (*simp add*: *finite-ticks-def Mndetprefix-projs Mprefix-projs*)
**thus** ‹$\mathbb{F}_{\checkmark}(\sqcap a{\in}A \rightarrow P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$› **by** (*simp add*: *finite-ticks-Mprefix-iff*)
**qed**

**lemma** *finite-ticks-write0-iff* [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}(a \rightarrow P) \longleftrightarrow \mathbb{F}_{\checkmark}(P)$›
**by** (*simp add*: *write0-def finite-ticks-Mprefix-iff*)

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

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

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

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

**lemma** *finite-ticks-inj-on-ndet-write-iff* [*finite-ticks-simps*] :
‹$\textit{inj-on}\ c\ A \Longrightarrow \mathbb{F}_{\checkmark}(c!!a{\in}A \rightarrow P\ a) \longleftrightarrow (\forall\, a{\in}A.\ \mathbb{F}_{\checkmark}(P\ a))$›
**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*] :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x \ \Box\ g\ x)$›
**by** (*simp add*: *finite-ticks-Det finite-ticks-fun-def*)

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

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

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

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

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

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

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

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

**corollary** ‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ ||\ \ g\ x)$›
**and** ‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(g) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ |||\ g\ x)$›
**by** (*fact finite-ticks-fun-Sync*)+

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

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

**lemma** *finite-ticks-fun-MultiSeq* [*finite-ticks-fun-simps*] :
‹$L = [] \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ SEQ\ l{\in}@L.\ f\ l\ x)$›
‹$L \neq [] \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ (last\ L)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ SEQ\ l{\in}@L.\ f\ l\ x)$›
**by** (*simp-all add*: *finite-ticks-MultiSeq-nonempty finite-ticks-fun-def finite-ticks-SKIP*)

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

**corollary** ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ m)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ ||\ \ m{\in}\#M.\ f\ m\ x)$›
**and** ‹$(\bigwedge m.\ m \in\# M \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(f\ m)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ |||\ m{\in}\#M.\ f\ m\ x)$›
**by** (*fact finite-ticks-fun-MultiSync*)+

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

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

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

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

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

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

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

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

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

**lemma** *finite-ticks-fun-ndet-write* [*finite-ticks-fun-simps*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ a\ x)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ c!!a{\in}A \rightarrow f\ a\ x)$›
**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 $'r \Rightarrow ('a, 'r)\ process_{ptick}$ (instead of $('a, 'r)\ process_{ptick}$).

**lift-definition** $Seq_{ptick}$ ::
‹$[('a, 'r)\ process_{ptick}, 'r \Rightarrow ('a, 'r)\ process_{ptick}] \Rightarrow ('a, 'r)\ process_{ptick}$› (**infixl** ‹;✓› 74)
**is** ‹$\lambda P\ Q.\ (\{(t, X)\ |t\ X.\ (t, X \cup range\ tick) \in \mathcal{F}\ P \wedge tF\ t\} \cup$
$\{(t\ @\ u, X)\ |t\ u\ r\ X.\ t\ @\ [✓(r)] \in \mathcal{T}\ P \wedge (u, X) \in \mathcal{F}\ (Q\ r)\} \cup$
$\{(t, X).\ t \in \mathcal{D}\ P\},$
$\mathcal{D}\ P \cup \{t\ @\ u\ |t\ u\ r.\ t\ @\ [✓(r)] \in \mathcal{T}\ P \wedge u \in \mathcal{D}\ (Q\ r)\})$›
**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 $'r \Rightarrow ('a, 'r)\ process_{ptick}$ while we would like to have in full generality $'r \Rightarrow ('a, 's)\ process_{ptick}$. The final definition given below is not immediate, and involves a precise understanding of the behaviour of the sequential composition.

### 3.1.1 Preliminaries

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

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

**lemma** *tickFree-map-ev-of-ev-eq-iff* :
‹*tF t* ⟹ *map* (*ev* ∘ *of-ev*) *t* = *t'* ⟹ *t* = *map* (*ev* ∘ *of-ev*) *t'*›
**by** (*induct t arbitrary*: *t'*) *auto*

**lemma** *tickFree-map-ev-of-ev-inj* :
‹*tF t* ⟹ *tF t'* ⟹ *map* (*ev* ∘ *of-ev*) *t* = *map* (*ev* ∘ *of-ev*) *t'* ⟷ *t* = *t'*›
**by** (*induct t arbitrary*: *t'*) (*use* $event_{ptick}$*.expand* **in** *auto*)+

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

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

**lemma** *tickFree-map-ev-of-ev-eq-imp-ev-mem-iff* :
‹*tF t'* ⟹ *t* = *map* (*ev* ∘ *of-ev*) *t'* ⟹ *ev a* ∈ *set t* ⟷ *ev a* ∈ *set t'*›
**by** (*induct t' arbitrary*: *t*) *auto*

The second key is to understand that $X \cup$ *range tick* can be rewritten as (*ev* ∘ *of-ev*) ‘ (*X* ∩ *range ev*) ∪ *range tick*, and that this second expression also allows for changing the type of termination.

**definition** $ref\text{-}Seq_{ptick}$ :: ‹('*a*, '*r*) $event_{ptick}$ *set* ⇒ ('*a*, '*s*) $event_{ptick}$ *set*›
**where** ‹$ref\text{-}Seq_{ptick}$ *X* ≡ (*ev* ∘ *of-ev*) ‘ (*X* ∩ *range ev*) ∪ *range tick*›

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

**lemma** $mono\text{-}ref\text{-}Seq_{ptick}$ : ‹*X* ⊆ *Y* ⟹ $ref\text{-}Seq_{ptick}$ *X* ⊆ $ref\text{-}Seq_{ptick}$ *Y*›
**unfolding** $ref\text{-}Seq_{ptick}$*-def* **by** *fast*

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

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

**lemma** $ref\text{-}Seq_{ptick}$*-eq-iff* :
‹$ref\text{-}Seq_{ptick}$ *X* = $ref\text{-}Seq_{ptick}$ *Y* ⟷ *X* ∩ *range ev* = *Y* ∩ *range ev*›
**proof** (*rule iffI*)

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

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

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

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

### 3.1.2 Formal Definition

**definition** $div\text{-}Seq_{ptick}$ ::
‹$[('a, 'r)\ process_{ptick}, 'r \Rightarrow ('a, 's)\ process_{ptick}] \Rightarrow ('a, 's)\ trace_{ptick}\ set$›
**where** ‹$div\text{-}Seq_{ptick}\ P\ Q \equiv$
$\{map\ (ev \circ of\text{-}ev)\ t\ @\ u\ |t\ u.\ t \in \mathcal{D}\ P \wedge tF\ t \wedge ftF\ u\} \cup$
$\{map\ (ev \circ of\text{-}ev)\ t\ @\ u\ |t\ u\ r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge tF\ t \wedge u \in \mathcal{D}\ (Q\ r)\}$›

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

**lift-definition** $Seq_{ptick}$ ::
‹$[('a, 'r)\ process_{ptick}, 'r \Rightarrow ('a, 's)\ process_{ptick}] \Rightarrow ('a, 's)\ process_{ptick}$› (**infixl** ‹;$_\checkmark$› 74)

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

```
      thus ‹(t, {}) ∈ ?f› by (auto simp add: fail-Seq_ptick-def)
    qed
  next
    case F-P
    have ‹(butlast t', {}) ∈ ℱ P›
      by (metis F-P(1, 2) is-processT3 is-processT4-empty list.map-disc-iff
snoc-eq-iff-butlast)
    with F-P(2) show ‹(t, {}) ∈ ?f›
   by (elim trace-tick-continuation-or-all-tick-failuresE, simp-all add: fail-Seq_ptick-def
ref-Seq_ptick-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 ‹(t, {}) ∈ ?f›
    proof (cases u rule: rev-cases)
      assume ‹u = []›
      have ‹(butlast t', {}) ∈ ℱ P›
        by (metis F-Q(2) T-F-spec append-butlast-last-id butlast.simps(1)
is-processT3-TR-append)
      thus ‹(t, {}) ∈ ?f›
        by (elim trace-tick-continuation-or-all-tick-failuresE, simp-all add:
fail-Seq_ptick-def ref-Seq_ptick-def)
          (metis (no-types, lifting) F-Q(1) ‹u = []› 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 ‹u = []› append-self-conv
butlast-snoc
          front-tickFree-iff-tickFree-butlast is-processT3-TR-append map-butlast)
    next
      from F-Q show ‹u = u' @ [e'] ⟹ (t, {}) ∈ ?f› for u' e'
        by (simp add: fail-Seq_ptick-def)
          (metis append-T-imp-tickFree is-processT3 list.distinct(1))
    qed
  qed
qed
next
  fix t X Y assume ‹(t, Y) ∈ ?f ∧ X ⊆ Y›
  hence ‹(t, Y)∈ ?f› ‹X ⊆ Y› by simp-all
  from ‹(t, Y)∈ ?f› consider (F-P) t' where ‹t = map (ev ∘ of-ev) t'›
    ‹(t', (ev ∘ of-ev) ‘ (Y ∩ range ev) ∪ range tick) ∈ ℱ P› ‹tF t'›
  | (F-Q) t' r u where ‹t = map (ev ∘ of-ev) t' @ u› ‹t' @ [✓(r)] ∈ 𝒯 P› ‹tF
t'› ‹(u, Y) ∈ ℱ (Q r)›
  | (D-P) t' u where ‹t = map (ev ∘ of-ev) t' @ u› ‹t' ∈ 𝒟 P› ‹tF t'› ‹ftF u›
    by (auto simp add: fail-Seq_ptick-def ref-Seq_ptick-def)
  thus ‹(t, X) ∈ ?f›
```

**proof** *cases*
**case** *F-P*
**from** ‹$X \subseteq Y$› **have** ‹$(ev \circ of\text{-}ev)\ `\ (X \cap range\ ev) \cup range\ tick \subseteq (ev \circ of\text{-}ev)\ `\ (Y \cap range\ ev) \cup range\ tick$› **by** *blast*
**with** *F-P(2)* **have** ‹$(t', (ev \circ of\text{-}ev)\ `\ (X \cap range\ ev) \cup range\ tick) \in \mathcal{F}\ P$›
**by** (*metis is-processT4*)
**with** *F-P(1, 3)* **show** ‹$(t, X) \in ?f$›
**by** (*auto simp add*: *fail-Seq$_{ptick}$-def ref-Seq$_{ptick}$-def*)
**next**
**case** *F-Q* **thus** ‹$(t, X) \in ?f$› **by** (*simp add*: *fail-Seq$_{ptick}$-def*) (*metis* ‹$X \subseteq Y$› *is-processT4*)
**next**
**case** *D-P* **thus** ‹$(t, X) \in ?f$› **by** (*auto simp add*: *fail-Seq$_{ptick}$-def*)
**qed**
**next**
**fix** *t X Y* **assume** ∗ : ‹$(t, X) \in ?f \wedge (\forall e.\ e \in Y \longrightarrow (t\ @\ [e], \{\}) \notin ?f)$›
**from** ∗ **consider** ‹$t \in ?d$›
| (*F-P*) *t'* **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'$› ‹$(t', (ev \circ of\text{-}ev)\ `\ (X \cap range\ ev) \cup range\ tick) \in \mathcal{F}\ P$› ‹*tF t'*›
| (*F-Q*) *t' r u* **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t'\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹*tF t'*› ‹$(u, X) \in \mathcal{F}\ (Q\ r)$›
**unfolding** *fail-Seq$_{ptick}$-def div-Seq$_{ptick}$-def ref-Seq$_{ptick}$-def* **by** *auto*
**thus** ‹$(t, X \cup Y) \in ?f$›
**proof** *cases*
**show** ‹$t \in ?d \Longrightarrow (t, X \cup Y) \in ?f$›
**by** (*simp add*: *div-Seq$_{ptick}$-def fail-Seq$_{ptick}$-def*) (*metis is-processT8*)
**next**
**case** *F-P*
**have** ‹$(t', (ev \circ of\text{-}ev)\ `\ (X \cap range\ ev) \cup range\ tick \cup (ev \circ of\text{-}ev)\ `\ (Y \cap range\ ev)) \in \mathcal{F}\ P$›
**proof** (*intro is-processT5*[*OF F-P(2)*] *allI impI*)
**fix** *e* :: ‹$('a, 'r)\ event_{ptick}$› **assume** ‹$e \in (ev \circ of\text{-}ev)\ `\ (Y \cap range\ ev)$›
**then obtain** *a* **where** ‹$e = ev\ a$› ‹$ev\ a \in Y$› **by** *auto*
**from** ∗[*THEN conjunct2*, *rule-format*, *OF this*(*2*), *unfolded fail-Seq$_{ptick}$-def*] *F-P(1, 3)*
**show** ‹$(t'\ @\ [e], \{\}) \notin \mathcal{F}\ P$›
**apply** (*simp add*: *fail-Seq$_{ptick}$-def* ‹$e = ev\ a$› *append-eq-map-conv ref-Seq$_{ptick}$-def*)
**by** (*smt* (*verit*, *del-insts*) *append-Nil2 comp-apply event$_{ptick}$.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** ‹$(ev \circ of\text{-}ev)\ `\ (X \cap range\ ev) \cup range\ tick \cup (ev \circ of\text{-}ev)\ `\ (Y \cap range\ ev) =$
*ref-Seq$_{ptick}$* $(X \cup Y)$› **unfolding** *ref-Seq$_{ptick}$-def* **by** *blast*
**finally show** ‹$(t, X \cup Y) \in ?f$›
**using** *F-P(1, 3)* **by** (*auto simp add*: *fail-Seq$_{ptick}$-def*)
**next**

**case** *F-Q*
**from** ∗ **have** ‹∀ *e*. *e* ∈ *Y* ⟶ (*u* @ [*e*], {}) ∉ $\mathcal{F}$ (*Q r*)›
**by** (*simp add*: *fail-Seq*$_{ptick}$*-def F-Q*(*1*, *2*))
(*metis F-Q*(*2*) *append-T-imp-tickFree not-Cons-self2*)
**with** *F-Q*(*3*, *4*) **have** ‹(*u*, *X* ∪ *Y*) ∈ $\mathcal{F}$ (*Q r*)› **by** (*simp add*: *is-processT5*)
**with** *F-Q*(*1−3*) **show** ‹(*t*, *X* ∪ *Y*) ∈ *?f*› **by** (*auto simp add*: *fail-Seq*$_{ptick}$*-def*)
**qed**
**next**
**show** ‹*t* ∈ *?d* ∧ *tF t* ∧ *ftF u* ⟹ *t* @ *u* ∈ *?d*› **for** *t u*
**by** (*simp add*: *div-Seq*$_{ptick}$*-def*, *elim conjE disjE exE*)
(*solves* ‹*use front-tickFree-append in auto*›,
*meson append.assoc is-processT7 tickFree-append-iff*)
**next**
**show** ‹*t* ∈ *?d* ⟹ (*t*, *X*) ∈ *?f*› **for** *t X*
**by** (*simp add*: *div-Seq*$_{ptick}$*-def fail-Seq*$_{ptick}$*-def*) (*metis is-processT8*)
**next**
**show** ∗ : ‹*t* @ [✓(*r*′)] ∈ *?d* ⟹ *t* ∈ *?d*› **for** *t r*′
**by** (*simp add*: *div-Seq*$_{ptick}$*-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** ‹(*t* @ [✓(*r*′)], {}) ∈ *?f*›
**then consider** ‹*t* @ [✓(*r*′)] ∈ *?d*›
| (*F-Q*) *t*′ *r u* **where** ‹*t* @ [✓(*r*′)] = *map* (*ev* ∘ *of-ev*) *t*′ @ *u*›
‹*t*′ @ [✓(*r*)] ∈ $\mathcal{T}$ *P*› ‹*tF t*′› ‹(*u*, *X*) ∈ $\mathcal{F}$ (*Q r*)›
**by** (*auto simp add*: *fail-Seq*$_{ptick}$*-def div-Seq*$_{ptick}$*-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** ‹(*t*, *X* − {✓(*r*′)}) ∈ *?f*›
**proof** *cases*
**assume** ‹*t* @ [✓(*r*′)] ∈ *?d*›
**with** ∗ **have** ‹*t* ∈ *?d*› **.**
**thus** ‹(*t*, *X* − {✓(*r*′)}) ∈ *?f*›
**by** (*simp add*: *fail-Seq*$_{ptick}$*-def div-Seq*$_{ptick}$*-def*) (*metis is-processT8*)
**next**
**case** *F-Q*
**from** *F-Q*(*1*, *2*) **obtain** *u*′ **where** ‹*u* = *u*′ @ [✓(*r*′)]›
**by** (*cases u rule*: *rev-cases*, *simp-all*)
(*metis non-tickFree-tick tickFree-append-iff tickFree-map-ev-comp*)
**with** *F-Q*(*4*) **have** ‹(*u*′, *X* − {✓(*r*′)}) ∈ $\mathcal{F}$ (*Q r*)› **by** (*simp add*: *F-T*
*is-processT6-TR*)
**with** *F-Q*(*1−3*) ‹*u* = *u*′ @ [✓(*r*′)]› **show** ‹(*t*, *X* − {✓(*r*′)}) ∈ *?f*›
**by** (*auto simp add*: *fail-Seq*$_{ptick}$*-def*)
**qed**
**qed**

**qed**

## 3.2 Projections

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

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

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

**lemma** *mono-Seq$_{ptick}$-eq* : ‹$P$ ;$_{\checkmark}$ $Q$ = $P'$ ;$_{\checkmark}$ $Q'$› **if** ∗ : ‹$P$ = $P'$› ‹⋀$r$. $r$ ∈ ✓**s**($P$) ⟹ $Q$ $r$ = $Q'$ $r$›
**for** $P$ $P'$ :: ‹('$a$, '$r$) *process$_{ptick}$*› **and** $Q$ $Q'$ :: ‹'$r$ ⇒ ('$a$, '$s$) *process$_{ptick}$*›
**proof** (*fold* ∗(1), *subst Process-eq-spec-optimized*, *safe*)
**{ fix** $t$ **and** $Q$ $Q'$ :: ‹'$r$ ⇒ ('$a$, '$s$) *process$_{ptick}$*›
**assume** ‹$t$ ∈ $\mathcal{D}$ ($P$ ;$_{\checkmark}$ $Q$)› **and** ∗ : ‹$r$ ∈ ✓**s**($P$) ⟹ $Q$ $r$ = $Q'$ $r$› **for** $r$
**from** ‹$t$ ∈ $\mathcal{D}$ ($P$ ;$_{\checkmark}$ $Q$)› **consider** (*D-P*) $t'$ $u$ **where** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$ @ $u$› ‹$t'$ ∈ $\mathcal{D}$ $P$› ‹*tF* $t'$› ‹*ftF* $u$›
| (*D-Q*) $t'$ $r$ $u$ **where** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$ @ $u$› ‹$t'$ @ [✓($r$)] ∈ $\mathcal{T}$ $P$› ‹*tF* $t'$› ‹$u$ ∈ $\mathcal{D}$ ($Q$ $r$)›
**unfolding** *Seq$_{ptick}$-projs* **by** *blast*
**hence** ‹$t$ ∈ $\mathcal{D}$ ($P$ ;$_{\checkmark}$ $Q'$)›
**proof** *cases*
**case** *D-P* **thus** ‹$t$ ∈ $\mathcal{D}$ ($P$ ;$_{\checkmark}$ $Q'$)› **by** (*auto simp add*: *Seq$_{ptick}$-projs*)

```
    next
      case D-Q thus ‹t ∈ 𝒟 (P ;✓ Q')›
        by (simp add: Seq_ptick-projs)
          (metis ∗ D-imp-front-tickFree is-processT9 strict-ticks-of-memI)
    qed
  } note $ = this
  show ‹t ∈ 𝒟 (P ;✓ Q) ⟹ t ∈ 𝒟 (P ;✓ Q')›
    and ‹t ∈ 𝒟 (P ;✓ Q') ⟹ t ∈ 𝒟 (P ;✓ Q)› for t
    by (erule $, simp add: ∗(2))+
next
  { fix t X and Q Q' :: ‹'r ⇒ ('a, 's) process_ptick›
    assume ‹(t, X) ∈ ℱ (P ;✓ Q)› and same-div : ‹𝒟 (P ;✓ Q) = 𝒟 (P ;✓ Q')›
      and ∗ : ‹r ∈ ✓s(P) ⟹ Q r = Q' r› for r
    from ‹(t, X) ∈ ℱ (P ;✓ Q)›
    consider (F-P) t' where ‹t = map (ev ∘ of-ev) t'› ‹(t', ref-Seq_ptick X) ∈ ℱ
P› ‹tF t'›
      | (F-Q) t' r u where ‹t = map (ev ∘ of-ev) t' @ u› ‹t' @ [✓(r)] ∈ 𝒯 P›  ‹tF
t'› ‹(u, X) ∈ ℱ (Q r)›
      | (D-P) t' u where ‹t = map (ev ∘ of-ev) t' @ u› ‹t' ∈ 𝒟 P› ‹tF t'› ‹ftF u›
      unfolding Seq_ptick-projs  by blast
    hence ‹(t, X) ∈ ℱ (P ;✓ Q')›
    proof cases
      case F-P thus ‹(t, X) ∈ ℱ (P ;✓ Q')› by (auto simp add: Seq_ptick-projs)
    next
      case F-Q thus ‹(t, X) ∈ ℱ (P ;✓ Q')›
        by (simp add: Seq_ptick-projs)
          (metis ∗ F-imp-front-tickFree is-processT9 strict-ticks-of-memI)
    next
      case D-P thus ‹(t, X) ∈ ℱ (P ;✓ Q')› by (auto simp add: Seq_ptick-projs)
    qed
  } note $ = this
  show ‹𝒟 (P ;✓ Q) = 𝒟 (P ;✓ Q') ⟹ (t, X) ∈ ℱ (P ;✓ Q) ⟹ (t, X) ∈ ℱ (P
;✓ Q')›
    and ‹𝒟 (P ;✓ Q) = 𝒟 (P ;✓ Q') ⟹ (t, X) ∈ ℱ (P ;✓ Q') ⟹ (t, X) ∈ ℱ (P
;✓ Q)› 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 $('a, 'r)$ $process_{ptick}$ and $'r \Rightarrow ('a, 's)$ $process_{ptick}$ respectively, the projections can be rewritten as follows.

```
lemma F-Seq_ptick-same-type :
  ‹ℱ (P ;✓ Q) = {(t, X) |t X. (t, X ∪ range tick) ∈ ℱ P ∧ tF t} ∪
                {(t @ u, X) |t u r X. t @ [✓(r)] ∈ 𝒯 P ∧ (u, X) ∈ ℱ (Q r)} ∪
                {(t, X). t ∈ 𝒟 P}›
  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-Seq$_{ptick}$-same-type* : ‹$\mathcal{D}$ ($P$ ;$_{\checkmark}$ $Q$) = $\mathcal{D}$ $P$ ∪ {$t$ @ $u$ |$t$ $u$ $r$. $t$ @ [✓($r$)] ∈ $\mathcal{T}$ $P$ ∧ $u$ ∈ $\mathcal{D}$ ($Q$ $r$)}›
**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-Seq$_{ptick}$-same-type-bis* :
‹$\mathcal{T}$ ($P$ ;$_{\checkmark}$ $Q$) = {$t$. ($t$, *range tick*) ∈ $\mathcal{F}$ $P$ ∧ $tF$ $t$} ∪
{$t$ @ $u$ |$t$ $u$ $r$. $t$ @ [✓($r$)] ∈ $\mathcal{T}$ $P$ ∧ $u$ ∈ $\mathcal{T}$ ($Q$ $r$)} ∪
$\mathcal{D}$ $P$›
**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-Seq$_{ptick}$-same-type* :
‹$\mathcal{T}$ ($P$ ;$_{\checkmark}$ $Q$) = {$t$ ∈ $\mathcal{T}$ $P$. $tF$ $t$} ∪ {$t$ @ $u$ |$t$ $u$ $r$. $t$ @ [✓($r$)] ∈ $\mathcal{T}$ $P$ ∧ $u$ ∈ $\mathcal{T}$ ($Q$ $r$)} ∪ $\mathcal{D}$ $P$›
— 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** *Seq$_{ptick}$-same-type-projs* = *F-Seq$_{ptick}$-same-type D-Seq$_{ptick}$-same-type T-Seq$_{ptick}$-same-type*

# Chapter 4

# Generalization of the Synchronization Product

## 4.1 Trace Interleaving

### 4.1.1 Motivation

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

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

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

**bundle** *option-type-syntax*
**begin**

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

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

**end**

**unbundle** *option-type-syntax*

### 4.1.2 Definition

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

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

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

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

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

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

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

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

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

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

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

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

**abbreviation** $setinterleaves_{ptick}$ ::
$\langle[('a,\ 't)\ trace_{ptick},\ 'r \Rightarrow 's \Rightarrow\ 't\ option,$
$('a,\ 'r)\ trace_{ptick},\ ('a,\ 's)\ trace_{ptick},\ 'a\ set] \Rightarrow bool\rangle$
$(\langle(\text{-}\ /(setinterleaves_{\checkmark})\text{-}/\ '(()'(\text{-},\ \text{-}')(),\ \text{-}'))\rangle\ [63,0,0,0,0]\ 64)$
**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** $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}$ :
$\langle t\ setinterleaves\ ((u,\ v),\ range\ tick \cup ev\ `\ A) \longleftrightarrow$
$t\ setinterleaves_{\checkmark \lambda r\ s.\ if\ r = s\ then\ \lfloor r \rfloor\ else\ \Diamond}\ ((u,\ v),\ A)\rangle$
**for** $t :: \langle('a,\ 'r)\ trace_{ptick}\rangle$
**by** (*induct* $\langle(\lambda r :: 'r.\ \lambda s :: 'r.\ if\ r = s\ then\ \lfloor r \rfloor\ else\ \Diamond,\ u,\ A,\ v)\rangle$
*arbitrary*: *t u v*) (*simp-all add*: *image-iff*)

**corollary** $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}\text{-}unit$ :
$\langle t\ setinterleaves\ ((u,\ v),\ insert\ \checkmark\ (ev\ `\ A)) \longleftrightarrow$
$t\ setinterleaves_{\checkmark \lambda r\ s.\ \lfloor r \rfloor}\ ((u,\ v),\ A)\rangle$ (**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** $\langle \ldots \longleftrightarrow ?rhs\rangle$
**by** (*simp add*: $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}$)
**finally show** $\langle ?lhs \longleftrightarrow ?rhs\rangle$ **.**
**qed**

**lemma** $setinterleaves_{ptick}$*-sym* :

— Of course not suitable for simplifier.
‹*t setinterleaves*$_{\checkmark \lambda s\ r.\ tick\text{-}join\ r\ s}$ ((*v*, *u*), *A*) ⟷
*t setinterleaves*$_{\checkmark \lambda r\ s.\ tick\text{-}join\ r\ s}$ ((*u*, *v*), *A*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*) (*auto split*: *option.split*)

**lemma** *setinterleaves*$_{Pair}$*-UNIV-iff* :
‹*t setinterleaves*$_{\checkmark \lambda r\ s.\ \lfloor (r,\ s) \rfloor}$ ((*u*, *v*), *UNIV*) ⟷
*u* = *map* (*map-event*$_{ptick}$ *id fst*) *t* ∧
*v* = *map* (*map-event*$_{ptick}$ *id snd*) *t*› **for** *t* :: ‹('*a*, '*r* × '*s*) *trace*$_{ptick}$›
**by** (*induct* ‹(λ*r* :: '*r*. λ*s* :: '*s*. ⌊(*r*, *s*)⌋, *u*, *UNIV* :: '*a set*, *v*)› *arbitrary*: *t u v*)
(*auto simp add*: *ev-eq-map-event*$_{ptick}$*-iff tick-eq-map-event*$_{ptick}$*-iff*)

**lemma** *setinterleaves*$_{ptick}$*-empty* :
‹*t setinterleaves*$_{\checkmark tick\text{-}join}$ ((*u*, *v*), {}) ⟹
*ev a* ∈ *set t* ⟷ *ev a* ∈ *set u* ∨ *ev a* ∈ *set v*›
**for** *u* :: ‹('*a*, '*r*) *trace*$_{ptick}$ ›
**by** (*induct* ‹(*tick-join*, *u*, {} :: '*a set*, *v*)› *arbitrary*: *t u v*)
(*auto split*: *option.split-asm*)

**lemma** *tickFree-setinterleaves*$_{ptick}$*-any-tick-join* :
‹*t setinterleaves*$_{\checkmark tick\text{-}join}$ ((*u*, *v*), *A*) ⟷
*t setinterleaves*$_{\checkmark tick\text{-}join'}$ ((*u*, *v*), *A*)›
**if** ‹*tF t* ∨ *tF u* ∨ *tF v*›
**proof** (*rule iffI*)
**from** ‹*tF t* ∨ *tF u* ∨ *tF v*›
**show** ‹*t setinterleaves*$_{\checkmark tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
*t setinterleaves*$_{\checkmark tick\text{-}join'}$ ((*u*, *v*), *A*)›
**for** *tick-join tick-join'*
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto split*: *if-split-asm option.split-asm*)
**thus** ‹*t setinterleaves*$_{\checkmark tick\text{-}join'}$ ((*u*, *v*), *A*) ⟹
*t setinterleaves*$_{\checkmark tick\text{-}join}$ ((*u*, *v*), *A*)› .
**qed**

**lemma** *tickFree-setinterleaves*$_{ptick}$*-iff* :
‹*t setinterleaves*$_{\checkmark tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *tF t* ⟷ *tF u* ∧ *tF v*›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto split*: *if-split-asm option.split-asm*)

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

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

**lemma** *setinterleaves*$_{ptick}$*-NilL-iff* :
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ (([], *v*), *A*) ⟷
*tF v* ∧ *set v* ∩ *ev* ‘ *A* = {} ∧ *t* = *map ev* (*map of-ev v*)›
**for** *tick-join* :: ‹'*r* ⇒ '*s* ⇒ '*t option*›
**by** (*induct* ‹(*tick-join*, [] :: ('*a*, '*r*) *trace*$_{ptick}$, *A*, *v*)›
*arbitrary*: *t v*) (*auto split*: *if-split-asm*)

**lemma** *setinterleaves*$_{ptick}$*-NilR-iff* :
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, []), *A*) ⟷
*tF u* ∧ *set u* ∩ *ev* ‘ *A* = {} ∧ *t* = *map ev* (*map of-ev u*)›
**for** *tick-join* :: ‹'*r* ⇒ '*s* ⇒ '*t option*›
**by** (*induct* ‹(*tick-join*, *u*, *A*, [] :: ('*a*, '*s*) *trace*$_{ptick}$)›
*arbitrary*: *t u*) (*auto split*: *if-split-asm*)

**lemma** *setinterleaves*$_{ptick}$*-subsetL* :
‹*tF t* ⟹ {*a*. *ev a* ∈ *set u*} ⊆ *A* ⟹
*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
*t* = *map ev* (*map of-ev v*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto simp add*: *subset-iff split*: *if-split-asm option.split-asm*)

**lemma** *setinterleaves*$_{ptick}$*-subsetR* :
‹*tF t* ⟹ {*a*. *ev a* ∈ *set v*} ⊆ *A* ⟹
*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
*t* = *map ev* (*map of-ev u*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto simp add*: *subset-iff split*: *if-split-asm option.split-asm*)

**lemma** *Nil-setinterleaves*$_{ptick}$ :
‹[] *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *u* = [] ∧ *v* = []›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *u v*)
(*simp-all split*: *if-split-asm option.split-asm*)

**lemma** *front-tickFree-setinterleaves*$_{ptick}$*-iff* :
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *ftF t* ⟷ *ftF u* ∧ *ftF v*›
**proof** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *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*: $setinterleaves_{ptick}$*-NilL-iff split*: *if-split-asm*)
**next**
**case** (*Nil-setinterleaving*$_{ptick}$*-tick s v*) **thus** *?case* **by** *simp*
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-ev a u b v*)
**thus** *?case* **by** (*simp split*: *if-split-asm*)
(*metis event*$_{ptick}$*.disc*(*1*) *front-tickFree-Cons-iff front-tickFree-Nil*)+
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-tick a u s v*)
**thus** *?case* **by** (*simp split*: *if-split-asm*)
(*metis event*$_{ptick}$*.disc*(*1*) *front-tickFree-Cons-iff front-tickFree-Nil*)
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-ev r u b v*)
**thus** *?case* **by** (*simp split*: *if-split-asm*)
(*metis event*$_{ptick}$*.disc*(*1*) *front-tickFree-Cons-iff front-tickFree-Nil*)
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-tick r u s v*) **thus** *?case*
**by** (*simp split*: *option.split-asm*)
(*metis Nil-setinterleaves*$_{ptick}$ *Nil-setinterleaving*$_{ptick}$*-Nil*
*event*$_{ptick}$*.disc*(*2*) *front-tickFree-Cons-iff singletonD*)
**qed**

**lemma** $setinterleaves_{ptick}$*-snoc-notinL* :
‹*t* $setinterleaves_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *a* ∉ *A* ⟹
*t* @ [*ev a*] $setinterleaves_{✓\,tick\text{-}join}$ ((*u* @ [*ev a*], *v*), *A*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto split*: *if-split-asm option.split-asm*)

**lemma** $setinterleaves_{ptick}$*-snoc-notinR* :
‹*t* $setinterleaves_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *a* ∉ *A* ⟹
*t* @ [*ev a*] $setinterleaves_{✓\,tick\text{-}join}$ ((*u*, *v* @ [*ev a*]), *A*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto split*: *if-split-asm option.split-asm*)

**lemma** $setinterleaves_{ptick}$*-snoc-inside* :
‹*t* $setinterleaves_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *a* ∈ *A* ⟹
*t* @ [*ev a*] $setinterleaves_{✓\,tick\text{-}join}$ ((*u* @ [*ev a*], *v* @ [*ev a*]), *A*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto split*: *if-split-asm option.split-asm*)

**lemma** $setinterleaves_{ptick}$*-snoc-tick* :
‹*t* $setinterleaves_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *tick-join r s* = ⌊*r-s*⌋ ⟹
*t* @ [✓(*r-s*)] $setinterleaves_{✓\,tick\text{-}join}$ ((*u* @ [✓(*r*)], *v* @ [✓(*s*)]), *A*)›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)

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

**lemma** *Cons-tick-setinterleaves*$_{ptick}$*E* :
‹✓(*r-s*) # *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
(⋀*u′ v′ r s*. ⟦*tick-join r s* = ⌊*r-s*⌋; *u* = ✓(*r*) # *u′*; *v* = ✓(*s*) # *v′*;
*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u′*, *v′*), *A*)⟧ ⟹ *thesis*) ⟹ *thesis*›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*simp-all split*: *if-split-asm option.split-asm*)

**lemma** *Cons-ev-setinterleaves*$_{ptick}$*E* :
‹*ev a* # *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
(⋀*u′*. *a* ∉ *A* ⟹ *u* = *ev a* # *u′* ⟹ *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u′*, *v*), *A*) ⟹ *thesis*) ⟹
(⋀*v′*. *a* ∉ *A* ⟹ *v* = *ev a* # *v′* ⟹ *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v′*), *A*) ⟹ *thesis*) ⟹
(⋀*u′ v′*. *a* ∈ *A* ⟹ *u* = *ev a* # *u′* ⟹ *v* = *ev a* # *v′* ⟹
*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u′*, *v′*), *A*) ⟹ *thesis*) ⟹ *thesis*›
**proof** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *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.prems*(*1*) **show** *?case*
**by** (*simp add*: *ev-setinterleaving*$_{ptick}$*-Nil.prems*(*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.prems*(*1*) **show** *?case*
**by** (*simp add*: *Nil-setinterleaving*$_{ptick}$*-ev.prems*(*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.prems*(*1*) **show** *?case*
**by** (*simp add*: *ev-setinterleaving*$_{ptick}$*-ev.prems*(*2*, *3*, *4*) *split*: *if-split-asm*)
(*use ev-setinterleaving*$_{ptick}$*-ev.prems*(*2*, *3*) **in** *presburger*)
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-tick b u s v*)
**from** *ev-setinterleaving*$_{ptick}$*-tick.prems*(*1*) **show** *?case*
**by** (*simp add*: *ev-setinterleaving*$_{ptick}$*-tick.prems*(*2*) *split*: *if-split-asm*)
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-ev r u c v*)
**from** *tick-setinterleaving*$_{ptick}$*-ev.prems*(*1*) **show** *?case*
**by** (*simp add*: *tick-setinterleaving*$_{ptick}$*-ev.prems*(*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-setinterleaves$_{ptick}$-rev-rev-iff* :
  ‹*rev t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*rev u*, *rev v*), *A*)
   ⟷ *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*)›
  **for** *u* :: ‹('*a*, '*r*) *trace$_{ptick}$*› **and** *v* :: ‹('*a*, '*s*) *trace$_{ptick}$*›
**proof** (*rule iffI*)
  **show** ‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
        *rev t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*rev u*, *rev v*), *A*)›
    **for** *u* :: ‹('*a*, '*r*) *trace$_{ptick}$*› **and** *v* :: ‹('*a*, '*s*) *trace$_{ptick}$*› **and** *t*
  **proof** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u v*)
    **case** *Nil-setinterleaving$_{ptick}$-Nil* **thus** *?case* **by** *simp*
  **next**
    **case** (*ev-setinterleaving$_{ptick}$-Nil a u*)
   **thus** *?case* **by** (*auto simp add*: *setinterleaves$_{ptick}$-snoc-notinL split*: *if-split-asm*
)
  **next**
    **case** (*tick-setinterleaving$_{ptick}$-Nil r v*) **thus** *?case* **by** *simp*
  **next**
    **case** (*Nil-setinterleaving$_{ptick}$-ev b v*)
   **thus** *?case* **by** (*auto simp add*: *setinterleaves$_{ptick}$-snoc-notinR split*: *if-split-asm*
)
  **next**
    **case** (*Nil-setinterleaving$_{ptick}$-tick s v*) **thus** *?case* **by** *simp*
  **next**
    **case** (*ev-setinterleaving$_{ptick}$-ev a u b v*)
    **from** *ev-setinterleaving$_{ptick}$-ev.prems*
    **consider** (*both-in*)  *t'* **where** ‹*a* ∈ *A*› ‹*a* = *b*› ‹*t* = *ev a* # *t'*›
      ‹*t'* *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*)›
    | (*inR-mvL*)  *t'* **where** ‹*a* ∉ *A*› ‹*b* ∈ *A*› ‹*t* = *ev a* # *t'*›
      ‹*t'* *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *ev b* # *v*), *A*)›
    | (*inL-mvR*)  *t'* **where** ‹*a* ∈ *A*› ‹*b* ∉ *A*› ‹*t* = *ev b* # *t'*›
      ‹*t'* *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*ev a* # *u*, *v*), *A*)›
    | (*notin-mvL*) *t'* **where** ‹*a* ∉ *A*› ‹*b* ∉ *A*› ‹*t* = *ev a* # *t'*›
      ‹*t'* *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *ev b* # *v*), *A*)›
    | (*notin-mvR*) *t'* **where** ‹*a* ∉ *A*› ‹*b* ∉ *A*› ‹*t* = *ev b* # *t'*›
      ‹*t'* *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*ev a* # *u*, *v*), *A*)›
      **by** (*auto split*: *if-split-asm*)
    **thus** *?case*
    **proof** *cases*
      **case** *both-in* **thus** *?thesis*
      **by** (*simp add*: *ev-setinterleaving$_{ptick}$-ev.hyps*(*1*) *setinterleaves$_{ptick}$-snoc-inside*)
    **next**
      **case** *inR-mvL* **thus** *?thesis*
      **by** (*metis ev-setinterleaving$_{ptick}$-ev.hyps*(*3*) *rev.simps*(*2*) *setinterleaves$_{ptick}$-snoc-notinL*)
    **next**
      **case** *inL-mvR* **thus** *?thesis*
      **by** (*metis ev-setinterleaving$_{ptick}$-ev.hyps*(*2*) *rev.simps*(*2*) *setinterleaves$_{ptick}$-snoc-notinR*)

**next**
**case** *notin-mvL* **thus** *?thesis*
**by** (*metis ev-setinterleaving*$_{ptick}$*-ev.hyps*(*4*) *rev.simps*(*2*) *setinterleaves*$_{ptick}$*-snoc-notinL*)
**next**
**case** *notin-mvR* **thus** *?thesis*
**by** (*metis ev-setinterleaving*$_{ptick}$*-ev.hyps*(*5*) *rev.simps*(*2*) *setinterleaves*$_{ptick}$*-snoc-notinR*)
**qed**
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-tick a u s v*) **thus** *?case*
**by** (*auto simp add*: *setinterleaves*$_{ptick}$*-snoc-notinL split*: *if-split-asm*)
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-ev r u b v*) **thus** *?case*
**by** (*auto simp add*: *setinterleaves*$_{ptick}$*-snoc-notinR split*: *if-split-asm*)
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-tick r u s v*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems*
**obtain** *t′ r-s* **where** ‹*tick-join r s* = ⌊*r-s*⌋› ‹*t* = ✓(*r-s*) # *t′*›
‹*t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*)›
**by** (*auto split*: *option.split-asm*)
**from** ‹*t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*)›
**have** ‹*rev t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u*, *rev v*), *A*)›
**by** (*simp add*: ‹*tick-join r s* = ⌊*r-s*⌋› *tick-setinterleaving*$_{ptick}$*-tick.hyps*)
**hence** ‹*rev t′* @ [✓(*r-s*)] *setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u* @ [✓(*r*)], *rev v* @ [✓(*s*)]), *A*)›
**by** (*simp add*: ‹*tick-join r s* = ⌊*r-s*⌋› *setinterleaves*$_{ptick}$*-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*$_{✓\,tick\text{-}join}$ ((*rev u*, *rev v*), *A*) ⟹
*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*)› .
**qed**

**lemma** *setinterleaves*$_{ptick}$*-preserves-ev-notin-set* :
‹⟦*ev a* ∉ *set u*; *ev a* ∉ *set v*; *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*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*$_{✓\,tick\text{-}join}$ ((*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*$_{✓\,tick\text{-}join}$ ((*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** *ev-notin-both-sets-imp-empty-setinterleaving*$_{ptick}$ :
‹⟦*ev a* ∈ *set u* ∧ *ev a* ∉ *set v* ∨ *ev a* ∉ *set u* ∧ *ev a* ∈ *set v*; *a* ∈ *A*⟧ ⟹
*setinterleaving*$_{ptick}$ (*tick-join*, *u*, *A*, *v*) = {}›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *u v*)
(*simp-all*, *safe*, *auto split*: *option.split-asm*)

**lemma** *setinterleaves*$_{ptick}$*-snoc-tick-snoc-tickE*:
‹(⋀*t′ r-s*. *tick-join r s* = ⌊*r-s*⌋ ⟹ *t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
*t* = *t′* @ [✓(*r-s*)] ⟹ *thesis*) ⟹ *thesis*›
**if** ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u* @ [✓(*r*)], *v* @ [✓(*s*)]), *A*)›
**proof** −
**from** *that* **have** ‹*rev t setinterleaves*$_{✓\,tick\text{-}join}$ ((✓(*r*) # *rev u*, ✓(*s*) # *rev v*), *A*)›
**by** (*metis* (*no-types*) *rev.simps*(*2*) *rev-rev-ident rev-setinterleaves*$_{ptick}$*-rev-rev-iff*)
**then obtain** *t′ r-s* **where** ‹*tick-join r s* = ⌊*r-s*⌋› ‹*rev t* = ✓(*r-s*) # *t′*›
‹*t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u*, *rev v*), *A*)›
**by** (*cases t rule*: *rev-cases*) (*simp-all split*: *option.split-asm*)
**hence** ‹*rev t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*) ∧ *t* = *rev t′* @ [✓(*r-s*)]›
**using** *rev-setinterleaves*$_{ptick}$*-rev-rev-iff* **by** *force*
**with** ‹*tick-join r s* = ⌊*r-s*⌋›
**show** ‹(⋀*t′ r-s*. *tick-join r s* = ⌊*r-s*⌋ ⟹ *t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*)
⟹
*t* = *t′* @ [✓(*r-s*)] ⟹ *thesis*) ⟹ *thesis*› **by** *blast*
**qed**

**lemma** *snoc-tick-setinterleaves*$_{ptick}$*E* :
‹(⋀*u′ v′ r s*. ⟦*tick-join r s* = ⌊*r-s*⌋; *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u′*, *v′*), *A*);
*u* = *u′* @ [✓(*r*)]; *v* = *v′* @ [✓(*s*)]⟧ ⟹ *thesis*) ⟹ *thesis*›
**if** ‹*t* @ [✓(*r-s*)] *setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *A*)›
**proof** −
**from** *that* **have** ‹*rev* (*t* @ [✓(*r-s*)]) *setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u*, *rev v*), *A*)›
**by** (*metis* (*no-types*) *rev.simps*(*2*) *rev-rev-ident rev-setinterleaves*$_{ptick}$*-rev-rev-iff*)
**hence** ‹✓(*r-s*) # *rev t setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u*, *rev v*), *A*)› **by** *simp*
**then obtain** *u′ v′ r s* **where** ‹*tick-join r s* = ⌊*r-s*⌋›
‹*rev t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u′*, *v′*), *A*)›
‹*rev u* = ✓(*r*) # *u′*› ‹*rev v* = ✓(*s*) # *v′*›
**by** (*elim Cons-tick-setinterleaves*$_{ptick}$*E*)
**hence** ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u′*, *rev v′*), *A*) ∧
*u* = *rev u′* @ [✓(*r*)] ∧ *v* = *rev v′* @ [✓(*s*)]›
**using** *rev-setinterleaves*$_{ptick}$*-rev-rev-iff* **by** *fastforce*
**with** ‹*tick-join r s* = ⌊*r-s*⌋›
**show** ‹(⋀*u′ v′ r s*. ⟦*tick-join r s* = ⌊*r-s*⌋; *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u′*, *v′*), *A*);
*u* = *u′* @ [✓(*r*)]; *v* = *v′* @ [✓(*s*)]⟧ ⟹ *thesis*) ⟹ *thesis*› **by** *blast*

**qed**

### 4.1.4 Lengths

**lemma** *length-setinterleaves$_{ptick}$-eq-sum-minus-filterL* :
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$ ((u, v), A) ⟹*
*length t = length u + length v − length (filter (λe. e ∈ range tick ∪ ev ‘ A) u)*›
**proof** (*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.prems* **consider** (*mv-left*) *a u′* **where** ‹*a ∉ A*› ‹*e = ev a*› ‹*u = ev a # u′*›
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$ ((u′, v), A)*›
| (*mv-right*) *a v′* **where** ‹*a ∉ A*› ‹*e = ev a*› ‹*v = ev a # v′*›
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$ ((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$_{\checkmark\,tick\text{-}join}$ ((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$_{\checkmark\,tick\text{-}join}$ ((u′, v′), A)*›
**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* :
‹*t setinterleaves$_{\checkmark\,tick\text{-}join}$ ((u, v), A) ⟹*
*length t = length u + length v − length (filter (λe. e ∈ range tick ∪ ev ‘ A) v)*›
**by** (*subst* (*asm*) *setinterleaves$_{ptick}$-sym*)
(*auto dest*: *length-setinterleaves$_{ptick}$-eq-sum-minus-filterL*)

**lemma** *setinterleaves$_{ptick}$-eq-length* :
‹*t* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
*t′* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹ *length t* = *length t′*›
**by** (*simp add*: *length-setinterleaves$_{ptick}$-eq-sum-minus-filterL*)

**lemma** *setinterleaves$_{ptick}$-imp-lengthLR-le* :
‹*t* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u*, *v*), *A*) ⟹
*length u* ≤ *length t* ∧ *length v* ≤ *length t*›
**by** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *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* :
‹*t* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*) ⟹
∃ *t1 t2 v1 v2*. *t* = *t1* @ *t2* ∧ *v* = *v1* @ *v2* ∧
*t1* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1*, *v1*), *A*) ∧
*t2* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u2*, *v2*), *A*)›
**proof** (*induct* ‹(*tick-join*, *u1*, *A*, *v*)› *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.prems*
**have** ‹*a* ∉ *A*› ‹*t* = *ev a* # *map ev* (*map of-ev* (*u1* @ *u2*))›
‹*map ev* (*map of-ev* (*u1* @ *u2*)) ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1* @ *u2*, []), *A*)›
**by** (*simp-all add*: *setinterleaves$_{ptick}$-NilR-iff split*: *if-split-asm*)
**from** *ev-setinterleaving$_{ptick}$-Nil.hyps*[*OF* ‹*a* ∉ *A*› *this*(*3*)]
**obtain** *t1 t2 v1 v2* **where** ‹*map ev* (*map of-ev* (*u1* @ *u2*)) = *t1* @ *t2*›
‹[] = *v1* @ *v2*› ‹*t1* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u1*, *v1*), *A*)›
‹*t2* ∈ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*u2*, *v2*), *A*)› **by** *blast*
**thus** *?case*
**by** (*simp add*: ‹*a* ∉ *A*› ‹*t* = *ev a* # *map ev* (*map of-ev* (*u1* @ *u2*))›)
(*metis append-Cons*)
**next**
**case** (*tick-setinterleaving$_{ptick}$-Nil r u1*)
**from** *tick-setinterleaving$_{ptick}$-Nil.prems* **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-setinterleaving*$_{ptick}$*-tick s v*)
**thus** *?case* **by** (*cases u2*, *simp-all add*: *setinterleaving*$_{ptick}$*-simps*
*split*: *event*$_{ptick}$*.split-asm*) *fastforce+*
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-ev a u1 b v*)
**from** *ev-setinterleaving*$_{ptick}$*-ev.prems* [*simplified*]
**consider** (*mv-both*) $t'$ **where** ‹$a \in A$› ‹$b \in A$› ‹$a = b$› ‹$t = ev\ b\ \#\ t'$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*)›
| (*mvR-inL*) $t'$ **where** ‹$a \in A$› ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ (((*ev a* # *u1*) @ *u2*, *v*), *A*)›
| (*mvL-inR*) $t'$ **where** ‹$a \notin A$› ‹$b \in A$› ‹$t = ev\ a\ \#\ t'$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u1* @ *u2*, *ev b* # *v*), *A*)›
| (*mvR-notin*) $t'$ **where** ‹$a \notin A$› ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ (((*ev a* # *u1*) @ *u2*, *v*), *A*)›
| (*mvL-notin*) $t'$ **where** ‹$a \notin A$› ‹$b \notin A$› ‹$t = ev\ a\ \#\ t'$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u1* @ *u2*, *ev b* # *v*), *A*)›
**by** (*auto split*: *if-split-asm*)
**thus** *?case*
**proof** *cases*
**case** *mv-both*
**from** *ev-setinterleaving*$_{ptick}$*-ev.hyps*(*1*)[*OF mv-both*(*1*−*3*, *5*)] **obtain** *t1 t2 v1 v2*
**where** ‹$t' = t1\ @\ t2$› ‹$v = v1\ @\ v2$› ‹*t1 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u1*, *v1*), *A*)›
‹*t2 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u2*, *v2*), *A*)› **by** *blast*
**hence** ‹$t = (ev\ b\ \#\ t1)\ @\ t2 \wedge ev\ b\ \#\ v = (ev\ b\ \#\ v1)\ @\ v2 \wedge$
*ev b* # *t1 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*ev a* # *u1*, *ev b* # *v1*), *A*) ∧
*t2 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u2*, *v2*), *A*)› **by** (*simp add*: *mv-both*(*1*−*4*))
**thus** *?thesis* **by** *blast*
**next**
**case** *mvR-inL*
**from** *ev-setinterleaving*$_{ptick}$*-ev.hyps*(*2*)[*OF mvR-inL*(*1*, *2*, *4*)] **obtain** *t1 t2 v1 v2*
**where** ‹$t' = t1\ @\ t2$› ‹$v = v1\ @\ v2$› ‹*t1 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*ev a* # *u1*, *v1*), *A*)›
‹*t2 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u2*, *v2*), *A*)› **by** *blast*
**hence** ‹$t = (ev\ b\ \#\ t1)\ @\ t2 \wedge ev\ b\ \#\ v = (ev\ b\ \#\ v1)\ @\ v2 \wedge$
*ev b* # *t1 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*ev a* # *u1*, *ev b* # *v1*), *A*) ∧
*t2 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u2*, *v2*), *A*)› **by** (*simp add*: *mvR-inL*(*1*−*3*))
**thus** *?thesis* **by** *blast*
**next**
**case** *mvL-inR*
**from** *ev-setinterleaving*$_{ptick}$*-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*$_{\checkmark\ tick\text{-}join}$ ((*u1*, *v1*), *A*)›
‹*t2 setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u2*, *v2*), *A*)› **by** *blast*

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

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

**corollary** *setinterleaves*$_{ptick}$*-appendR* :
 ‹∃ *t1 t2 u1 u2*. $t = t1$ @ $t2 \wedge u = u1$ @ $u2 \wedge$
      *t1 setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1*, *v1*), *A*) ∧
      *t2 setinterleaves*$_{✓\,tick\text{-}join}$ ((*u2*, *v2*), *A*)›
 **if** ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v1* @ *v2*), *A*)›
**proof** −
 **from** *that* **have** ‹*t setinterleaves*$_{✓\,\lambda s\ r.\ tick\text{-}join\ r\ s}$ ((*v1* @ *v2*, *u*), *A*)›
  **using** *setinterleaves*$_{ptick}$*-sym* **by** *blast*
 **from** *setinterleaves*$_{ptick}$*-appendL*[*OF this*]
 **obtain** *t1 t2 u1 u2* **where** ‹$t = t1$ @ $t2$› ‹$u = u1$ @ $u2$›
  ‹*t1 setinterleaves*$_{✓\,\lambda s\ r.\ tick\text{-}join\ r\ s}$ ((*v1*, *u1*), *A*)›
  ‹*t2 setinterleaves*$_{✓\,\lambda s\ r.\ tick\text{-}join\ r\ s}$ ((*v2*, *u2*), *A*)› **by** *blast*
 **from** *this*(*3*, *4*) **have** ‹*t1 setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1*, *v1*), *A*)›
  ‹*t2 setinterleaves*$_{✓\,tick\text{-}join}$ ((*u2*, *v2*), *A*)›
  **using** *setinterleaves*$_{ptick}$*-sym* **by** *blast*+
 **with** ‹$t = t1$ @ $t2$› ‹$u = u1$ @ $u2$› **show** *?thesis* **by** *blast*
**qed**

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

```
  ∃ u1 u2 v1 v2. u = u1 @ u2 ∧ v = v1 @ v2 ∧
                t1 setinterleaves✓tick-join ((u1, v1), A) ∧
                t2 setinterleaves✓tick-join ((u2, v2), A)›
proof (induct t1 arbitrary: u v)
  case Nil
  hence ‹u = [] @ u› ‹v = [] @ v›
    ‹[] setinterleaves✓tick-join (([], []), A)›
    ‹t2 setinterleaves✓tick-join ((u, v), A)› by simp-all
  thus ?case by blast
next
  case (Cons e t1)
  from Cons.prems consider (mv-left) a u' where ‹a ∉ A› ‹e = ev a› ‹u = ev a
# u'›
    ‹t1 @ t2 setinterleaves✓tick-join ((u', v), A)›
  | (mv-right) a v' where ‹a ∉ A› ‹e = ev a› ‹v = ev a # v'›
    ‹t1 @ t2 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'›
    ‹t1 @ t2 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'› ‹t1 @ t2 setinterleaves✓tick-join ((u', v'), A)›
   by (cases e) (auto elim: Cons-ev-setinterleaves_ptickE Cons-tick-setinterleaves_ptickE)
  thus ?case
  proof cases
    case mv-left
    from Cons.hyps[OF mv-left(4)] obtain u1 u2 v1 v2
      where ‹u' = u1 @ u2› ‹t1 setinterleaves✓tick-join ((u1, v1), A)›
        and ∗ : ‹v = v1 @ v2› ‹t2 setinterleaves✓tick-join ((u2, v2), A)› by blast
    from this(2) have ‹e # t1 setinterleaves✓tick-join ((ev a # u1, v1), A)›
      by (cases v1) (auto simp add: ‹a ∉ A› ‹e = ev a› setinterleaving_ptick-simps
          split: event_ptick.split)
    moreover from ‹u' = u1 @ u2› have ‹u = (ev a # u1) @ u2›
      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 ‹v' = v1 @ v2› ‹t1 setinterleaves✓tick-join ((u1, v1), A)›
        and ∗ : ‹u = u1 @ u2› ‹t2 setinterleaves✓tick-join ((u2, v2), A)› by blast
    from this(2) have ‹e # t1 setinterleaves✓tick-join ((u1, ev a # v1), A)›
      by (cases u1) (auto simp add: ‹a ∉ A› ‹e = ev a› setinterleaving_ptick-simps
          split: event_ptick.split)
    moreover from ‹v' = v1 @ v2› have ‹v = (ev a # v1) @ v2›
      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** ‹$u' = u1 \mathbin{@} u2$› ‹$v' = v1 \mathbin{@} v2$› ‹$t1$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ $((u1,\ v1),\ A)$›
**and** $*$ : ‹$t2$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ $((u2,\ v2),\ A)$› **by** *blast*
**from** *this*(*3*) **have** ‹$e \mathbin{\#} t1$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ $((ev\ a \mathbin{\#} u1,\ ev\ a \mathbin{\#} v1),\ A)$›
**by** (*simp add*: ‹$a \in A$› ‹$e = ev\ a$›)
**moreover from** ‹$u' = u1 \mathbin{@} u2$› **have** ‹$u = (ev\ a \mathbin{\#} u1) \mathbin{@} u2$›
**by** (*simp add*: *mv-both-ev*(*3*))
**moreover from** ‹$v' = v1 \mathbin{@} v2$› **have** ‹$v = (ev\ a \mathbin{\#} v1) \mathbin{@} v2$›
**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** ‹$u' = u1 \mathbin{@} u2$› ‹$v' = v1 \mathbin{@} v2$› ‹$t1$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ $((u1,\ v1),\ A)$›
**and** $*$ : ‹$t2$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ $((u2,\ v2),\ A)$› **by** *blast*
**from** *this*(*3*) **have** ‹$e \mathbin{\#} t1$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ $((\checkmark(r) \mathbin{\#} u1,\ \checkmark(s) \mathbin{\#} v1),\ A)$›
**by** (*simp add*: *mv-both-tick*(*1*, *2*))
**moreover from** ‹$u' = u1 \mathbin{@} u2$› **have** ‹$u = (\checkmark(r) \mathbin{\#} u1) \mathbin{@} u2$›
**by** (*simp add*: *mv-both-tick*(*3*))
**moreover from** ‹$v' = v1 \mathbin{@} v2$› **have** ‹$v = (\checkmark(s) \mathbin{\#} v1) \mathbin{@} v2$›
**by** (*simp add*: *mv-both-tick*(*4*))
**ultimately show** *?thesis* **using** $*$ **by** *blast*
**qed**
**qed**

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

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

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

**lemma** *setinterleaves$_{ptick}$-less-prefixL* :
‹*t setinterleaves$_{\checkmark tick\text{-}join}$ ((u, v), A) ⟹ u′ < u ⟹*
*∃ t′ v′. t′ < t ∧ v′ ≤ v ∧ t′ setinterleaves$_{\checkmark tick\text{-}join}$ ((u′, v′), A)*›
**proof** (*induct* ‹(*tick-join*, *u*, *A*, *v*)› *arbitrary*: *t u u′ v*)
**case** *Nil-setinterleaving$_{ptick}$-Nil* **thus** *?case* **by** *simp*
**next**
**case** (*ev-setinterleaving$_{ptick}$-Nil a u*)
**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-setinterleaving$_{ptick}$-Nil.prems*(*1*)
**show** ‹*u′ = [] ⟹ ?case*› **by** (*auto split*: *if-split-asm*)
**next**
**fix** *u″* **assume** ‹*u′ = ev a # u″*› ‹*u″ < u*›
**from** *ev-setinterleaving$_{ptick}$-Nil.prems*(*1*)
**obtain** *t′* **where** ‹*a ∉ A*› ‹*t = ev a # t′*› ‹*t′ setinterleaves$_{\checkmark tick\text{-}join}$ ((u, []), A)*›
**by** (*auto split*: *if-split-asm*)
**from** *ev-setinterleaving$_{ptick}$-Nil.hyps*[*OF* ‹*a ∉ A*› *this*(*3*) ‹*u″ < u*›]
**obtain** *t″ v′* **where** ‹*t″ < t′*› ‹*v′ ≤ []*› ‹*t″ setinterleaves$_{\checkmark tick\text{-}join}$ ((u″, v′), A)*› **by** *blast*
**hence** ‹*ev a # t″ < t ∧ v′ ≤ [] ∧ ev a # t″ setinterleaves$_{\checkmark tick\text{-}join}$ ((u′, v′), A)*›
**by** (*simp add*: ‹*u′ = ev a # u″*› ‹*t = ev a # t′*› ‹*a ∉ A*›)
**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** ‹*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-setinterleaving$_{ptick}$-ev.prems*(*1*)
**show** ‹*u′ = [] ⟹ ?case*› **by** (*simp split*: *if-split-asm*) *force+*
**next**
**fix** *u″* **assume** ‹*u′ = ev a # u″*› ‹*u″ < u*›
**hence** ‹*ev a # u″ < ev a # u*› **by** *simp*
**from** *ev-setinterleaving$_{ptick}$-ev.prems*(*1*)
**consider** (*both-in*) *t′* **where** ‹*a ∈ A*› ‹*b ∈ A*› ‹*a = b*› ‹*t = ev a # t′*›
‹*t′ setinterleaves$_{\checkmark tick\text{-}join}$ ((u, v), A)*›

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

**case** *notin-mvR*
**from** *ev-setinterleaving$_{ptick}$-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*$_{✓\,tick\text{-}join}$ ((*ev a # u''*, *v'*), *A*)› **by** *blast*
**hence** ‹*ev b # t''* < *t* ∧ *ev b # v'* ≤ *ev b # v* ∧
*ev b # t''* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*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-setinterleaving$_{ptick}$-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-setinterleaving$_{ptick}$-tick.prems(1)*
**show** ‹*u'* = [] ⟹ *?case*› **by** (*simp split*: *if-split-asm*) *force+*
**next**
**fix** *u''* **assume** ‹*u'* = *ev a # u''*› ‹*u''* < *u*›
**from** *ev-setinterleaving$_{ptick}$-tick.prems* **obtain** *t'*
**where** ‹*a* ∉ *A*› ‹*t* = *ev a # t'*› ‹*t'* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, ✓(*s*) # *v*), *A*)›
**by** (*auto split*: *if-split-asm*)
**from** *ev-setinterleaving$_{ptick}$-tick.hyps*[*OF* ‹*a* ∉ *A*› *this(3)* ‹*u''* < *u*›]
**obtain** *t'' v'* **where** ‹*t''* < *t'* ∧ *v'* ≤ ✓(*s*) # *v* ∧ *t''* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*u''*, *v'*), *A*)› **by** *blast*
**hence** ‹*ev a # t''* < *t* ∧ *v'* ≤ ✓(*s*) # *v* ∧ *ev a # t''* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*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-setinterleaving$_{ptick}$-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-setinterleaving$_{ptick}$-ev.prems(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-setinterleaving$_{ptick}$-ev.prems* **obtain** *t'*
**where** ‹*b* ∉ *A*› ‹*t* = *ev b # t'*› ‹*t'* *setinterleaves*$_{✓\,tick\text{-}join}$ ((✓(*r*) # *u*, *v*), *A*)›
**by** (*auto split*: *if-split-asm*)
**from** *tick-setinterleaving$_{ptick}$-ev.hyps*[*OF* ‹*b* ∉ *A*› *this(3)* ‹✓(*r*) # *u''* < ✓(*r*)

$\#\ u\rangle$]
**obtain** $t''\ v'$ **where** $\langle t'' < t' \wedge v' \leq v \wedge t''\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((\checkmark(r)\ \#\ u'',\ v'),\ A)\rangle$ **by** *blast*
**hence** $\langle ev\ b\ \#\ t'' < t \wedge ev\ b\ \#\ v' \leq ev\ b\ \#\ v \wedge$
$ev\ b\ \#\ t''\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((u',\ ev\ b\ \#\ v'),\ A)\rangle$
**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** ($tick\text{-}setinterleaving_{ptick}\text{-}tick\ r\ u\ s\ v$)
**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** $tick\text{-}setinterleaving_{ptick}\text{-}tick.prems(1)$
**show** $\langle u' = [] \Longrightarrow ?case\rangle$ **by** (*force split*: *option.split-asm*)
**next**
**fix** $u''$ **assume** $\langle u' = \checkmark(r)\ \#\ u''\rangle$ $\langle u'' < u\rangle$
**from** $tick\text{-}setinterleaving_{ptick}\text{-}tick.prems(1)$
**obtain** $t'$ *r-s*
**where** $\langle tick\text{-}join\ r\ s = \lfloor r\text{-}s\rfloor\rangle$ $\langle t = \checkmark(r\text{-}s)\ \#\ t'\rangle$ $\langle t'\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((u,\ v),\ A)\rangle$
**by** (*auto split*: *option.split-asm*)
**from** $tick\text{-}setinterleaving_{ptick}\text{-}tick.hyps[OF\ this(1,\ 3)\ \langle u'' < u\rangle]$
**obtain** $t''\ v'$ **where** $\langle t'' < t' \wedge v' \leq v \wedge t''\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((u'',\ v'),\ A)\rangle$ **by** *blast*
**hence** $\langle \checkmark(r\text{-}s)\ \#\ t'' < t \wedge \checkmark(s)\ \#\ v' \leq \checkmark(s)\ \#\ v \wedge \checkmark(r\text{-}s)\ \#\ t''\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((u',\ \checkmark(s)\ \#\ v'),\ A)\rangle$
**by** (*simp add*: $\langle tick\text{-}join\ r\ s = \lfloor r\text{-}s\rfloor\rangle$ $\langle u' = \checkmark(r)\ \#\ u''\rangle$ $\langle t = \checkmark(r\text{-}s)\ \#\ t'\rangle$)
**thus** *?case* **by** *blast*
**qed**
**qed**

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

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

**next**
**case** (*ev-setinterleaving*$_{ptick}$*-Nil a u*) **thus** *?case* **by** *simp fastforce*
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-Nil r u*) **thus** *?case* **by** *simp*
**next**
**case** (*Nil-setinterleaving*$_{ptick}$*-ev b v*) **thus** *?case* **by** *simp fastforce*
**next**
**case** (*Nil-setinterleaving*$_{ptick}$*-tick s v*) **thus** *?case* **by** *simp*
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-ev a u b v*)
**show** *?case*
**proof** (*cases* ‹$u' = [] \vee v' = []$›)
**show** ‹$u' = [] \vee v' = [] \Longrightarrow$ *?case*› **by** *force*
**next**
**assume** ‹$\neg\ (u' = [] \vee v' = [])$›
**with** *ev-setinterleaving*$_{ptick}$*-ev.prems*(*2*, *3*)
**obtain** $u''\ v''$ **where** ‹$u' = ev\ a\ \#\ u''$› ‹$u'' \leq u$› ‹$v' = ev\ b\ \#\ v''$› ‹$v'' \leq v$›
**by** (*meson Prefix-Order.prefix-Cons*)
**from** *ev-setinterleaving*$_{ptick}$*-ev.prems*(*1*)
**consider** (*both-in*) $t'$ **where** ‹$a \in A$› ‹$b \in A$› ‹$a = b$› ‹$t = ev\ a\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u, v), A)$›
| (*inR-mvL*) $t'$ **where** ‹$a \notin A$› ‹$b \in A$› ‹$t = ev\ a\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u, ev\ b\ \#\ v), A)$›
| (*inL-mvR*) $t'$ **where** ‹$a \in A$› ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((ev\ a\ \#\ u, v), A)$›
| (*notin-mvL*) $t'$ **where** ‹$a \notin A$› ‹$b \notin A$› ‹$t = ev\ a\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u, ev\ b\ \#\ v), A)$›
| (*notin-mvR*) $t'$ **where** ‹$a \notin A$› ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((ev\ a\ \#\ u, v), A)$›
**by** (*auto split*: *if-split-asm*)
**thus** *?case*
**proof** *cases*
**case** *both-in*
**from** *ev-setinterleaving*$_{ptick}$*-ev.hyps*(*1*)[*OF both-in*(*1−3*, *5*) ‹$u'' \leq u$› ‹$v'' \leq v$›]
**show** *?thesis*
**proof** (*elim disjE exE conjE*)
**fix** $t''\ v'''$
**assume** ‹$t'' \leq t'$› ‹$v''' \leq v''$› ‹$t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u'', v'''), A)$›
**hence** ‹$ev\ b\ \#\ t'' \leq t \wedge ev\ b\ \#\ v''' \leq v' \wedge$
$ev\ b\ \#\ t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u', ev\ b\ \#\ v'''), A)$›
**by** (*simp add*: ‹$u' = ev\ a\ \#\ u''$› ‹$v' = ev\ b\ \#\ v''$› *both-in*(*2−4*))
**thus** *?thesis* **by** *blast*
**next**
**fix** $t''\ u'''$
**assume** ‹$t'' \leq t'$› ‹$u''' \leq u''$› ‹$t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u''', v''), A)$›
**hence** ‹$ev\ a\ \#\ t'' \leq t \wedge ev\ a\ \#\ u''' \leq u' \wedge$
$ev\ a\ \#\ t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((ev\ a\ \#\ u''', v'), A)$›

**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-setinterleaving*$_{ptick}$-*ev.hyps*(3)[*OF inR-mvL*(1, 2, 4) ‹$u'' \leq u$› ‹$v' \leq ev\ b\ \#\ v$›]
**show** *?thesis*
**proof** (*elim disjE exE conjE*)
**fix** $t''\ v'''$
**assume** ‹$t'' \leq t'$› ‹$v''' \leq v'$› ‹$t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u'', v'''), A)$›
**hence** ‹$ev\ a\ \#\ t'' \leq t \wedge v''' \leq v' \wedge$
$ev\ a\ \#\ t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u', v'''), A)$›
**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** ‹$t'' \leq t'$› ‹$u''' \leq u''$› ‹$t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u''', v'), A)$›
**hence** ‹$ev\ a\ \#\ t'' \leq t \wedge ev\ a\ \#\ u''' \leq u' \wedge$
$ev\ a\ \#\ t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((ev\ a\ \#\ u''', v'), A)$›
**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-setinterleaving*$_{ptick}$-*ev.hyps*(2)[*OF inL-mvR*(1, 2, 4) ‹$u' \leq ev\ a\ \#\ u$› ‹$v'' \leq v$›]
**show** *?thesis*
**proof** (*elim disjE exE conjE*)
**fix** $t''\ v'''$
**assume** ‹$t'' \leq t'$› ‹$v''' \leq v''$› ‹$t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u', v'''), A)$›
**hence** ‹$ev\ b\ \#\ t'' \leq t \wedge ev\ b\ \#\ v''' \leq v' \wedge$
$ev\ b\ \#\ t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u', ev\ b\ \#\ v'''), A)$›
**by** (*simp add*: ‹$u' = ev\ a\ \#\ u''$› ‹$v' = ev\ b\ \#\ v''$› *inL-mvR*(2, 3))
**thus** *?thesis* **by** *blast*
**next**
**fix** $t''\ u'''$
**assume** ‹$t'' \leq t'$› ‹$u''' \leq u'$› ‹$t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u''', v''), A)$›
**hence** ‹$ev\ b\ \#\ t'' \leq t \wedge u''' \leq u' \wedge$
$ev\ b\ \#\ t''$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((u''', v'), A)$›
**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-setinterleaving*$_{ptick}$-*ev.hyps*(4)[*OF notin-mvL*(1, 2, 4) ‹$u'' \leq u$› ‹$v' \leq ev\ b\ \#\ v$›]

```
      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-setinterleaving_ptick-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-setinterleaving_ptick-tick a u s v)
  show ?case
  proof (cases ‹u' = [] ∨ v' = []›)
    show ‹u' = [] ∨ v' = [] ⟹ ?case› by force
  next
    assume ‹¬ (u' = [] ∨ v' = [])›
    with ev-setinterleaving_ptick-tick.prems(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-setinterleaving*$_{ptick}$*-tick.prems*(*1*)
**obtain** $t'$ **where** ‹$a \notin A$› ‹$t = ev\ a\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u, \checkmark(s)\ \#\ v), A)$›
**by** (*auto split*: *if-split-asm*)
**from** *ev-setinterleaving*$_{ptick}$*-tick.hyps*[*OF this*(*1*, *3*) ‹$u'' \leq u$› ‹$v' \leq \checkmark(s)\ \#\ v$›]
**show** *?case*
**proof** (*elim disjE exE conjE*)
**fix** $t''\ v'''$ **assume** ‹$t'' \leq t'$› ‹$v''' \leq v'$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u'', v'''), A)$›
**hence** ‹$ev\ a\ \#\ t'' \leq t \wedge v''' \leq v' \wedge ev\ a\ \#\ t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u', v'''), A)$›
**by** (*cases* $v'''$) (*simp-all add*: ‹$a \notin A$› ‹$t = ev\ a\ \#\ t'$› ‹$u' = ev\ a\ \#\ u''$› ‹$v' = \checkmark(s)\ \#\ v''$›)
**thus** *?case* **by** *blast*
**next**
**fix** $t''\ u'''$ **assume** ‹$t'' \leq t'$› ‹$u''' \leq u''$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u''', v'), A)$›
**hence** ‹$ev\ a\ \#\ t'' \leq t \wedge ev\ a\ \#\ u''' \leq u' \wedge ev\ a\ \#\ t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((ev\ a\ \#\ u''', v'), A)$›
**by** (*simp add*: ‹$a \notin A$› ‹$t = ev\ a\ \#\ t'$› ‹$u' = ev\ a\ \#\ u''$› ‹$v' = \checkmark(s)\ \#\ v''$›)
**thus** *?case* **by** *blast*
**qed**
**qed**
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-ev r u b v*)
**show** *?case*
**proof** (*cases* ‹$u' = [] \vee v' = []$›)
**show** ‹$u' = [] \vee v' = [] \Longrightarrow$ *?case*› **by** *force*
**next**
**assume** ‹$\neg\ (u' = [] \vee v' = [])$›
**with** *tick-setinterleaving*$_{ptick}$*-ev.prems*(*2*, *3*)
**obtain** $u''\ v''$ **where** ‹$u' = \checkmark(r)\ \#\ u''$› ‹$u'' \leq u$› ‹$v' = ev\ b\ \#\ v''$› ‹$v'' \leq v$›
**by** (*meson Prefix-Order.prefix-Cons*)
**from** *tick-setinterleaving*$_{ptick}$*-ev.prems*(*1*)
**obtain** $t'$ **where** ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$›
‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((\checkmark(r)\ \#\ u, v), A)$›
**by** (*auto split*: *if-split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-ev.hyps*[*OF this*(*1*, *3*) ‹$u' \leq \checkmark(r)\ \#\ u$› ‹$v'' \leq v$›]
**show** *?case*
**proof** (*elim disjE exE conjE*)
**fix** $t''\ v'''$ **assume** ‹$t'' \leq t'$› ‹$v''' \leq v''$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u', v'''), A)$›
**hence** ‹$ev\ b\ \#\ t'' \leq t \wedge ev\ b\ \#\ v''' \leq v' \wedge ev\ b\ \#\ t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u', ev\ b\ \#\ v'''), A)$›
**by** (*simp add*: ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$› ‹$u' = \checkmark(r)\ \#\ u''$› ‹$v' = ev\ b\ \#\ v''$›)
**thus** *?case* **by** *blast*
**next**
**fix** $t''\ u'''$ **assume** ‹$t'' \leq t'$› ‹$u''' \leq u'$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u''', v''),$

$A$)›
**hence** ‹$ev\ b\ \#\ t'' \leq t \wedge u''' \leq u' \wedge ev\ b\ \#\ t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u''', v'), A)$›
**by** (*cases* $u'''$) (*simp-all add*: ‹$b \notin A$› ‹$t = ev\ b\ \#\ t'$› ‹$u' = \checkmark(r)\ \#\ u''$› ‹$v' = ev\ b\ \#\ v''$›)
**thus** *?case* **by** *blast*
**qed**
**qed**
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-tick* $r\ u\ s\ v$)
**show** *?case*
**proof** (*cases* ‹$u' = [] \vee v' = []$›)
**show** ‹$u' = [] \vee v' = [] \Longrightarrow$ *?case*› **by** *force*
**next**
**assume** ‹$\neg\ (u' = [] \vee v' = [])$›
**with** *tick-setinterleaving*$_{ptick}$*-tick.prems*(*2*, *3*)
**obtain** $u''\ v''$ **where** ‹$u' = \checkmark(r)\ \#\ u''$› ‹$u'' \leq u$› ‹$v' = \checkmark(s)\ \#\ v''$› ‹$v'' \leq v$›
**by** (*meson Prefix-Order.prefix-Cons*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems*(*1*)
**obtain** $t'$ *r-s* **where** ‹$t = \checkmark(r\text{-}s)\ \#\ t'$› ‹*tick-join* $r\ s = \lfloor r\text{-}s \rfloor$›
‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u, v), A)$›
**by** (*auto split*: *option.split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.hyps*[*OF this*(*2*, *3*) ‹$u'' \leq u$› ‹$v'' \leq v$›]
**show** *?case*
**proof** (*elim disjE exE conjE*)
**fix** $t''\ v'''$
**assume** ‹$t'' \leq t'$› ‹$v''' \leq v''$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u'', v'''), A)$›
**hence** ‹$\checkmark(r\text{-}s)\ \#\ t'' \leq t \wedge \checkmark(s)\ \#\ v''' \leq v' \wedge$
$\checkmark(r\text{-}s)\ \#\ t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u', \checkmark(s)\ \#\ v'''), A)$›
**by** (*simp add*: ‹*tick-join* $r\ s = \lfloor r\text{-}s \rfloor$› ‹$t = \checkmark(r\text{-}s)\ \#\ t'$›
‹$u' = \checkmark(r)\ \#\ u''$› ‹$v' = \checkmark(s)\ \#\ v''$›)
**thus** *?case* **by** *blast*
**next**
**fix** $t''\ u'''$
**assume** ‹$t'' \leq t'$› ‹$u''' \leq u''$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u''', v''), A)$›
**hence** ‹$\checkmark(r\text{-}s)\ \#\ t'' \leq t \wedge \checkmark(r)\ \#\ u''' \leq u' \wedge$
$\checkmark(r\text{-}s)\ \#\ t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((\checkmark(r)\ \#\ u''', v'), A)$›
**by** (*simp add*: ‹*tick-join* $r\ s = \lfloor r\text{-}s \rfloor$› ‹$t = \checkmark(r\text{-}s)\ \#\ t'$›
‹$u' = \checkmark(r)\ \#\ u''$› ‹$v' = \checkmark(s)\ \#\ v''$›)
**thus** *?case* **by** *blast*
**qed**
**qed**
**qed**

### 4.1.6 Hiding Events

**lemma** *setinterleaves*$_{ptick}$*-trace-hide* :
‹$t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u, v), S) \Longrightarrow$
*trace-hide* $t$ ($ev$ ‘ $A$) *setinterleaves*$_{\checkmark\ tick\text{-}join}$

```
   ((trace-hide u (ev ‘ A), trace-hide v (ev ‘ A)), S)›
proof (induct ‹(tick-join, u, S, v)› arbitrary: t u v)
  case Nil-setinterleaving_ptick-Nil
  thus ?case by simp
next
  case (ev-setinterleaving_ptick-Nil a u)
  from ev-setinterleaving_ptick-Nil.prems obtain t' where ‹a ∉ S› ‹t = ev a # t'›
    ‹t' setinterleaves✓tick-join ((u, []), S)› by (auto split: if-split-asm)
  from ev-setinterleaving_ptick-Nil.hyps[OF this(1, 3)]
  show ?case by (simp add: image-iff[of ‹ev -›] ‹a ∉ S› ‹t = ev a # t'›)
next
  case (tick-setinterleaving_ptick-Nil r u)
  from tick-setinterleaving_ptick-Nil have False by simp
  thus ?case ..
next
  case (Nil-setinterleaving_ptick-ev b v)
  from Nil-setinterleaving_ptick-ev.prems obtain t' where ‹b ∉ S› ‹t = ev b # t'›
    ‹t' setinterleaves✓tick-join (([], v), S)› by (auto split: if-split-asm)
  from Nil-setinterleaving_ptick-ev.hyps[OF this(1, 3)]
  show ?case by (simp add: image-iff[of ‹ev -›] ‹b ∉ S› ‹t = ev b # t'›)
next
  case (Nil-setinterleaving_ptick-tick s v)
  from Nil-setinterleaving_ptick-tick.prems have False by simp
  thus ?case ..
next
  case (ev-setinterleaving_ptick-ev a u b v)
  from ev-setinterleaving_ptick-ev.prems
  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-setinterleaving_ptick-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-setinterleaving_ptick-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) setinterleaving_ptick-simps
          split: if-split-asm event_ptick.split)
```

**next**
**case** *inL-mvR*
**from** *ev-setinterleaving*$_{ptick}$*-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) *setinterleaving*$_{ptick}$*-simps*
*split*: *if-split-asm event*$_{ptick}$*.split*)
**next**
**case** *notin-mvL*
**from** *ev-setinterleaving*$_{ptick}$*-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) *setinterleaving*$_{ptick}$*-simps*
*split*: *if-split-asm event*$_{ptick}$*.split*)
**next**
**case** *notin-mvR*
**from** *ev-setinterleaving*$_{ptick}$*-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) *setinterleaving*$_{ptick}$*-simps*
*split*: *if-split-asm event*$_{ptick}$*.split*)
**qed**
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-tick a u s v*)
**from** *ev-setinterleaving*$_{ptick}$*-tick.prems* **obtain** *t′* **where** ‹*a* ∉ *S*› ‹*t* = *ev a* # *t′*›
‹*t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, ✓(*s*) # *v*), *S*)› **by** (*auto split*: *if-split-asm*)
**from** *ev-setinterleaving*$_{ptick}$*-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-setinterleaving*$_{ptick}$*-ev r u b v*)
**from** *tick-setinterleaving*$_{ptick}$*-ev.prems* **obtain** *t′* **where** ‹*b* ∉ *S*› ‹*t* = *ev b* # *t′*›
‹*t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((✓(*r*) # *u*, *v*), *S*)› **by** (*auto split*: *if-split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-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-setinterleaving*$_{ptick}$*-tick r u s v*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems*
**obtain** *r-s t′* **where** ‹*tick-join r s* = ⌊*r-s*⌋› ‹*t* = ✓(*r-s*) # *t′*›
‹*t′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *S*)› **by** (*auto split*: *option.split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-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**

**lemma** *trace-hide-map-map-event*$_{ptick}$ :
‹*trace-hide* (*map* (*map-event*$_{ptick}$ *f g*) *t*) *S* =
*map* (*map-event*$_{ptick}$ *f g*) (*trace-hide t* (*map-event*$_{ptick}$ *f g* −‘ *S*))›
**by** (*induct t*) *simp-all*

**lemma** *tickFree-trace-hide-map-ev-comp-of-ev* :
‹*tF t* ⟹ *trace-hide* (*map* (*ev* ∘ *of-ev*) *t*) (*ev* ‘ *A*) =
*map* (*ev* ∘ *of-ev*) (*trace-hide t* (*ev* ‘ *A*))›
**by** (*induct t*) (*auto simp add*: *image-iff*)

**lemma** *tickFree-disjoint-setinterleaves*$_{ptick}$*-appendL* :
‹*tF u1* ⟹ {*a*. *ev a* ∈ *set u1*} ∩ *A* = {} ⟹ *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u2*, *v*), *A*)
⟹ *map* (*ev* ∘ *of-ev*) *u1* @ *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*)›
**proof** (*induct u1*)
**case** *Nil*
**from** *Nil.prems*(*3*) **show** *?case* **by** *simp*
**next**
**case** (*Cons e u1*)
**from** *Cons.prems*(*1*, *2*) **obtain** *a*
**where** ‹*e* = *ev a*› ‹*a* ∉ *A*› ‹*tF u1*› ‹{*a*. *ev a* ∈ *set u1*} ∩ *A* = {} ›
**by** (*auto simp add*: *disjoint-iff is-ev-def*)
**from** *Cons.hyps*[*OF this*(*3*, *4*) *Cons.prems*(*3*)]
**have** ‹*map* (*ev* ∘ *of-ev*) *u1* @ *t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*)› .
**with** ‹*e* = *ev a*› ‹*a* ∉ *A*›
**show** *?case* **by** (*cases v*)
(*auto simp add*: *setinterleaving*$_{ptick}$*-simps comp-def split*: *event*$_{ptick}$*.split*)
**qed**

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

**lemma** *tickFree-disjoint-setinterleaves*$_{ptick}$*-append-tailL* :
‹*t* @ *map* (*ev* ∘ *of-ev*) *u2 setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*)›
**if** ‹*tF u2*› ‹{*a*. *ev a* ∈ *set u2*} ∩ *A* = {}› ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1*, *v*), *A*)›
**proof** −
**have** ‹*t* @ *map* (*ev* ∘ *of-ev*) *u2 setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1* @ *u2*, *v*), *A*) ⟷
*map* (*ev* ∘ *of-ev*) (*rev u2*) @ *rev t setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u2* @ *rev u1*, *rev v*), *A*)›
**by** (*subst rev-setinterleaves*$_{ptick}$*-rev-rev-iff*[*symmetric*])
(*simp add*: *rev-map*)
**also have** . . .
**proof** (*rule tickFree-disjoint-setinterleaves*$_{ptick}$*-appendL*)
**show** ‹*tF* (*rev u2*)› **by** (*simp add*: *that*(*1*))
**next**
**show** ‹{*a*. *ev a* ∈ *set* (*rev u2*)} ∩ *A* = {}› **by** (*simp add*: *that*(*2*))
**next**
**show** ‹*rev t setinterleaves*$_{✓\,tick\text{-}join}$ ((*rev u1*, *rev v*), *A*)›

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

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

**lemma** *disjoint-trace-hide-setinterleaves*$_{ptick}$ :
‹*t setinterleaves*$_{✓\,tick\text{-}join}$
((*trace-hide u* (*ev* ‘ *A*), *trace-hide v* (*ev* ‘ *A*)), *S*) ⟹
∃ *t'*. *t* = *trace-hide t'* (*ev* ‘ *A*) ∧
*t' setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, *v*), *S*)› **if** ‹*A* ∩ *S* = {}›
**for** *t* :: ‹('*a*, '*t*) *trace*$_{ptick}$› **and** *u* :: ‹('*a*, '*r*) *trace*$_{ptick}$› **and** *v* :: ‹('*a*, '*s*) *trace*$_{ptick}$›
**proof** −
**let** *?th* = *trace-hide* **and** *?A* = ‹*ev* ‘ *A*›
**show** ‹*t setinterleaves*$_{✓\,tick\text{-}join}$
((*?th u ?A*, *?th v ?A*), *S*) ⟹ ∃ *t'*. *t* = *?th t' ?A* ∧ *t' setinterleaves*$_{✓\,tick\text{-}join}$
((*u*, *v*), *S*)›
**proof** (*induct* ‹(*tick-join*, *u*, *S*, *v*)› *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.prems*
**consider** *t'* **where** ‹*a* ∉ *S*› ‹*a* ∉ *A*› ‹*t* = *ev a* # *t'*›
‹*t' setinterleaves*$_{✓\,tick\text{-}join}$ ((*?th u ?A*, *?th* [] *?A*), *S*)›
| ‹*a* ∈ *A*› ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*?th u ?A*, *?th* [] *?A*), *S*)›
**by** (*auto split*: *if-split-asm*)
**thus** *?case*
**proof** *cases*
**fix** *t'* **assume** ‹*a* ∉ *S*› ‹*a* ∉ *A*› ‹*t* = *ev a* # *t'*›
‹*t' setinterleaves*$_{✓\,tick\text{-}join}$ ((*?th u ?A*, *?th* [] *?A*), *S*)›
**from** *ev-setinterleaving*$_{ptick}$*-Nil.hyps*[*OF this*(*1*, *4*)] **obtain** *t''*
**where** ‹*t'* = *?th t''* *?A* ∧ *t''* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, []), *S*)› **..**
**hence** ‹*t* = *?th* (*ev a* # *t''*) *?A* ∧ *ev a* # *t''* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*ev a* # *u*, []), *S*)›
**by** (*simp add*: ‹*a* ∉ *A*› ‹*a* ∉ *S*› ‹*t* = *ev a* # *t'*› *image-iff*[*of* ‹*ev* -›])
**thus** *?case* **..**
**next**
**assume** ‹*a* ∈ *A*›
**with** ‹*A* ∩ *S* = {}› **have** ‹*a* ∉ *S*› **by** *blast*
**moreover assume** ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*?th u ?A*, *?th* [] *?A*), *S*)›
**ultimately obtain** *t'* **where** ‹*t* = *?th t'* *?A*› ‹*t' setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*,

[]), *S*)›
**using** *ev-setinterleaving*$_{ptick}$*-Nil.hyps* **by** *blast*
**hence** ‹*t* = *?th* (*ev a* # *t'*) *?A* ∧ *ev a* # *t' setinterleaves*$_{✓\ tick\text{-}join}$ ((*ev a* # *u*, []), *S*)›
**by** (*simp add*: ‹*a* ∈ *A*› ‹*a* ∉ *S*›)
**thus** *?case* **..**
**qed**
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-Nil r u*)
**from** *tick-setinterleaving*$_{ptick}$*-Nil.prems* **have** *False* **by** (*simp add*: *image-iff*[*of* ‹✓(-)›])
**thus** *?case* **..**
**next**
**case** (*Nil-setinterleaving*$_{ptick}$*-ev b v*)
**from** *Nil-setinterleaving*$_{ptick}$*-ev.prems*
**consider** *t'* **where** ‹*b* ∉ *S*› ‹*b* ∉ *A*› ‹*t* = *ev b* # *t'*›
‹*t' setinterleaves*$_{✓\ tick\text{-}join}$ ((*?th* [] *?A*, *?th v ?A*), *S*)›
| ‹*b* ∈ *A*› ‹*t setinterleaves*$_{✓\ tick\text{-}join}$ ((*?th* [] *?A*, *?th v ?A*), *S*)›
**by** (*auto split*: *if-split-asm*)
**thus** *?case*
**proof** *cases*
**fix** *t'* **assume** ‹*b* ∉ *S*› ‹*b* ∉ *A*› ‹*t* = *ev b* # *t'*›
‹*t' setinterleaves*$_{✓\ tick\text{-}join}$ ((*?th* [] *?A*, *?th v ?A*), *S*)›
**from** *Nil-setinterleaving*$_{ptick}$*-ev.hyps*[*OF this*(*1*, *4*)] **obtain** *t''*
**where** ‹*t'* = *?th t''* *?A* ∧ *t''* *setinterleaves*$_{✓\ tick\text{-}join}$ (([], *v*), *S*)› **..**
**hence** ‹*t* = *?th* (*ev b* # *t''*) *?A* ∧ *ev b* # *t''* *setinterleaves*$_{✓\ tick\text{-}join}$ (([], *ev b* # *v*), *S*)›
**by** (*simp add*: ‹*b* ∉ *A*› ‹*b* ∉ *S*› ‹*t* = *ev b* # *t'*› *image-iff*[*of* ‹*ev* -›])
**thus** *?case* **..**
**next**
**assume** ‹*b* ∈ *A*›
**with** ‹*A* ∩ *S* = {}› **have** ‹*b* ∉ *S*› **by** *blast*
**moreover assume** ‹*t setinterleaves*$_{✓\ tick\text{-}join}$ ((*?th* [] *?A*, *?th v ?A*), *S*)›
**ultimately obtain** *t'* **where** ‹*t* = *?th t'* *?A*› ‹*t' setinterleaves*$_{✓\ tick\text{-}join}$ (([], *v*), *S*)›
**using** *Nil-setinterleaving*$_{ptick}$*-ev.hyps* **by** *blast*
**hence** ‹*t* = *?th* (*ev b* # *t'*) *?A* ∧ *ev b* # *t'* *setinterleaves*$_{✓\ tick\text{-}join}$ (([], *ev b* # *v*), *S*)›
**by** (*simp add*: ‹*b* ∈ *A*› ‹*b* ∉ *S*›)
**thus** *?case* **..**
**qed**
**next**
**case** (*Nil-setinterleaving*$_{ptick}$*-tick s v*)
**from** *Nil-setinterleaving*$_{ptick}$*-tick.prems* **have** *False* **by** (*simp add*: *image-iff*[*of* ‹✓(-)›])
**thus** *?case* **..**
**next**
**case** (*ev-setinterleaving*$_{ptick}$*-ev a u b v*)

**show** *?case*
**proof** (*cases* ‹$a \in A$›; *cases* ‹$b \in A$›)
**assume** ‹$a \in A$› ‹$b \in A$›
**with** *ev-setinterleaving*$_{ptick}$-*ev.prems*
**have** $*$ : ‹$t$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*?th* (*ev a* # $u$) *?A*, *?th v ?A*), $S$)›
‹$t$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*?th u ?A*, *?th* (*ev b* # $v$) *?A*), $S$)› **by** *simp-all*
**from** ‹$A \cap S = \{\}$› ‹$a \in A$› ‹$b \in A$› **have** ‹$a \notin S$› ‹$b \notin S$› **by** *blast+*
**from** *ev-setinterleaving*$_{ptick}$-*ev.hyps*(*4*)[*OF this* $*$(*2*)]
*ev-setinterleaving*$_{ptick}$-*ev.hyps*(*5*)[*OF this* $*$(*1*)]
**obtain** $t'$ **where** ‹$t$ = *?th* $t'$ *?A*›
‹$t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a* # $u$, $v$), $S$) ∨
$t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ (($u$, *ev b* # $v$), $S$)› **by** *blast*
**hence** ‹$t$ = *?th* (*ev b* # $t'$) *?A* ∧ *ev b* # $t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a* # $u$, *ev b* # $v$), $S$) ∨
$t$ = *?th* (*ev a* # $t'$) *?A* ∧ *ev a* # $t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a* # $u$, *ev b* # $v$), $S$)›
**by** (*auto simp add*: ‹$a \in A$› ‹$b \in A$› ‹$a \notin S$› ‹$b \notin S$›)
**thus** *?case* **by** *blast*
**next**
**assume** ‹$a \in A$› ‹$b \notin A$›
**with** *ev-setinterleaving*$_{ptick}$-*ev.prems*
**have** $*$ : ‹$t$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*?th u ?A*, *?th* (*ev b* # $v$) *?A*), $S$)› **by** *simp*
**from** ‹$A \cap S = \{\}$› ‹$a \in A$› **have** ‹$a \notin S$› **by** *blast*
**from** *ev-setinterleaving*$_{ptick}$-*ev.hyps*(*3*)[*OF* ‹$a \notin S$› - $*$(*1*)]
*ev-setinterleaving*$_{ptick}$-*ev.hyps*(*4*)[*OF* ‹$a \notin S$› - $*$] **obtain** $t'$
**where** ‹$t$ = *?th* $t'$ *?A*› ‹$t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ (($u$, *ev b* # $v$), $S$)› **by** *blast*
**hence** ‹$t$ = *?th* (*ev a* # $t'$) *?A* ∧
*ev a* # $t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a* # $u$, *ev b* # $v$), $S$)›
**by** (*simp add*: ‹$a \in A$› ‹$a \notin S$›)
**thus** *?case* **..**
**next**
**assume** ‹$a \notin A$› ‹$b \in A$›
**with** *ev-setinterleaving*$_{ptick}$-*ev.prems*
**have** $*$ : ‹$t$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*?th* (*ev a* # $u$) *?A*, *?th v ?A*), $S$)› **by** *simp*
**from** ‹$A \cap S = \{\}$› ‹$b \in A$› **have** ‹$b \notin S$› **by** *blast*
**from** *ev-setinterleaving*$_{ptick}$-*ev.hyps*(*2*)[*OF* - ‹$b \notin S$› $*$]
*ev-setinterleaving*$_{ptick}$-*ev.hyps*(*5*)[*OF* - ‹$b \notin S$› $*$] **obtain** $t'$
**where** ‹$t$ = *?th* $t'$ *?A*› ‹$t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ (($u$, *ev a* # $u$, $v$), $S$)› **by** *blast*
**hence** ‹$t$ = *?th* (*ev b* # $t'$) *?A* ∧
*ev b* # $t'$ *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a* # $u$, *ev b* # $v$), $S$)›
**by** (*simp add*: ‹$b \in A$› ‹$b \notin S$›)
**thus** *?case* **..**
**next**
**assume** ‹$a \notin A$› ‹$b \notin A$›
**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-setinterleaving_ptick-ev.prems[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-setinterleaving_ptick-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-setinterleaving_ptick-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-setinterleaving_ptick-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-setinterleaving_ptick-tick a u s v)
  from ev-setinterleaving_ptick-tick.prems
  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** ‹$a \notin S$› ‹$a \notin A$› ‹$t = ev\ a \# t'$›
‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*?th u ?A*, *?th* (✓($s$) # $v$) *?A*), $S$)›
**from** *ev-setinterleaving*$_{ptick}$-*tick.hyps*[*OF this*(*1*, *4*)] **obtain** $t''$
**where** ‹$t' = ?th\ t''\ ?A$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ (($u$, ✓($s$) # $v$), $S$)› **by** *blast*
**hence** ‹$t = ?th\ (ev\ a \# t'')\ ?A \wedge$
$ev\ a \# t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ (($ev\ a \# u$, ✓($s$) # $v$), $S$)›
**by** (*simp add*: ‹$a \notin A$› ‹$a \notin S$› ‹$t = ev\ a \# t'$› *image-iff*[*of* ‹*ev* -›])
**thus** *?case* **..**
**next**
**assume** ‹$a \in A$›
**with** ‹$A \cap S = \{\}$› **have** ‹$a \notin S$› **by** *blast*
**moreover assume** ‹$t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*?th u ?A*, *?th* (✓($s$) # $v$) *?A*), $S$)›
**ultimately obtain** $t'$ **where** ‹$t = ?th\ t'\ ?A$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ (($u$, ✓($s$) # $v$), $S$)›
**using** *ev-setinterleaving*$_{ptick}$-*tick.hyps* **by** *blast*
**hence** ‹$t = ?th\ (ev\ a \# t')\ ?A \wedge ev\ a \# t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ (($ev\ a \# u$, ✓($s$) # $v$), $S$)›
**by** (*simp add*: ‹$a \in A$› ‹$a \notin S$›)
**thus** *?case* **..**
**qed**
**next**
**case** (*tick-setinterleaving*$_{ptick}$-*ev r u b v*)
**from** *tick-setinterleaving*$_{ptick}$-*ev.prems*
**consider** $t'$ **where** ‹$b \notin S$› ‹$b \notin A$› ‹$t = ev\ b \# t'$›
‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*?th* (✓($r$) # $u$) *?A*, *?th v ?A*), $S$)›
| ‹$b \in A$› ‹$t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*?th* (✓($r$) # $u$) *?A*, *?th v ?A*), $S$)›
**by** (*auto split*: *if-split-asm*)
**thus** *?case*
**proof** *cases*
**fix** $t'$ **assume** ‹$b \notin S$› ‹$b \notin A$› ‹$t = ev\ b \# t'$›
‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*?th* (✓($r$) # $u$) *?A*, *?th v ?A*), $S$)›
**from** *tick-setinterleaving*$_{ptick}$-*ev.hyps*[*OF this*(*1*, *4*)] **obtain** $t''$
**where** ‹$t' = ?th\ t''\ ?A$› ‹$t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((✓($r$) # $u$, $v$), $S$)› **by** *blast*
**hence** ‹$t = ?th\ (ev\ b \# t'')\ ?A \wedge$
$ev\ b \# t''$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((✓($r$) # $u$, $ev\ b \# v$), $S$)›
**by** (*simp add*: ‹$b \notin A$› ‹$b \notin S$› ‹$t = ev\ b \# t'$› *image-iff*[*of* ‹*ev* -›])
**thus** *?case* **..**
**next**
**assume** ‹$b \in A$›
**with** ‹$A \cap S = \{\}$› **have** ‹$b \notin S$› **by** *blast*
**moreover assume** ‹$t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*?th* (✓($r$) # $u$) *?A*, *?th v ?A*), $S$)›
**ultimately obtain** $t'$ **where** ‹$t = ?th\ t'\ ?A$› ‹$t'$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((✓($r$)

$\#\ u,\ v),\ S)\rangle$
**using** *tick-setinterleaving*$_{ptick}$*-ev.hyps* **by** *blast*
**hence** $\langle t = ?th\ (ev\ b\ \#\ t')\ ?A \wedge ev\ b\ \#\ t'\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((\checkmark(r)\ \#\ u,\ ev\ b\ \#\ v),\ S)\rangle$
**by** (*simp add*: $\langle b \in A\rangle$ $\langle b \notin S\rangle$)
**thus** *?case* **..**
**qed**
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-tick r u s v*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems* **obtain** *r-s t'*
**where** $\langle$*tick-join r s* $= \lfloor$*r-s*$\rfloor\rangle$ $\langle t = \checkmark(r\text{-}s)\ \#\ t'\rangle$
$\langle t'\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((?th\ u\ ?A,\ ?th\ v\ ?A),\ S)\rangle$
**by** (*auto split*: *if-split-asm option.split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.hyps*[*OF this*(*1*, *3*)] **obtain** *t''*
**where** $\langle t' = ?th\ t''\ ?A\rangle$ $\langle t''\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((u,\ v),\ S)\rangle$ **by** *blast*
**hence** $\langle t = ?th\ (\checkmark(r\text{-}s)\ \#\ t'')\ ?A \wedge$
$\checkmark(r\text{-}s)\ \#\ t''\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((\checkmark(r)\ \#\ u,\ \checkmark(s)\ \#\ v),\ S)\rangle$
**by** (*simp add*: $\langle$*tick-join r s* $= \lfloor$*r-s*$\rfloor\rangle$ $\langle t = \checkmark(r\text{-}s)\ \#\ t'\rangle$ *image-iff*[*of* $\langle\checkmark(\text{-})\rangle$])
**thus** *?case* **..**
**qed**
**qed**

**lemma** *setinterleaves*$_{ptick}$*-inj-map-map-event*$_{ptick}$*-iff-weak* :
$\langle$*map* (*map-event*$_{ptick}$ *f id*) *t setinterleaves*$_{\checkmark\ tick\text{-}join}$
((*map* (*map-event*$_{ptick}$ *f id*) *u*, *map* (*map-event*$_{ptick}$ *f id*) *v*), *f* ' *A*) $\longleftrightarrow$
*t setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u*, *v*), *A*)$\rangle$ **if** $\langle$*inj f*$\rangle$
**by** (*induct* $\langle$(*tick-join*, *u*, *A*, *v*)$\rangle$ *arbitrary*: *t u v*)
(*auto simp add*: *image-iff map-event*$_{ptick}$*-eq-ev-iff map-event*$_{ptick}$*-eq-tick-iff*
*dest*!: *injD*[*OF* $\langle$*inj f*$\rangle$] *split*: *option.split-asm*)

**lemma** *setinterleaves*$_{ptick}$*-inj-map-map-event*$_{ptick}$*-iff-strong* :
$\langle$*t setinterleaves*$_{\checkmark\ tick\text{-}join}$
((*map* (*map-event*$_{ptick}$ *f id*) *u*, *map* (*map-event*$_{ptick}$ *f id*) *v*), *f* ' *A*) $\longleftrightarrow$
($\exists\, t'.\ t = map\ (map\text{-}event_{ptick}\ f\ id)\ t' \wedge$
*t'* *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*u*, *v*), *A*))$\rangle$ **if** $\langle$*inj f*$\rangle$
— We could probably prove a stronger version with *inj-on f* (*A* $\cup$ {*a*. *ev a* $\in$ *set u* $\vee$ *ev a* $\in$ *set v*}) instead of *inj f*.
**proof** −
**let** *?map* = $\langle$*map* (*map-event*$_{ptick}$ *f id*)$\rangle$
**have** $\langle t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((?map\ u,\ ?map\ v),\ f\ `\ A) \Longrightarrow \exists\, t'.\ t = ?map\ t'\rangle$
**proof** (*induct* $\langle$(*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*)

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

```
    next
      case mv-both
      from ev-setinterleaving_ptick-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-setinterleaving_ptick-tick a u s v)
    from ev-setinterleaving_ptick-tick.prems 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-setinterleaving_ptick-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-setinterleaving_ptick-ev r u b v)
    from tick-setinterleaving_ptick-ev.prems 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-setinterleaving_ptick-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-setinterleaving_ptick-tick r u s v)
    from tick-setinterleaving_ptick-tick.prems 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-setinterleaving_ptick-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 setinterleaves_ptick-inj-map-map-event_ptick-iff-weak[OF ‹inj f›]
  show ?thesis by blast
qed

lemma setinterleaves_ptick-append-setinterleaves_ptick :
  ‹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 ‹(tick-join, u1, A, v1)› arbitrary: t1 u1 v1)
  case Nil-setinterleaving_ptick-Nil
  from Nil-setinterleaving_ptick-Nil.prems(1) have ‹t1 = []› by simp
  with that(2) show ?case by simp
next
  case (ev-setinterleaving_ptick-Nil a u1)
  from ev-setinterleaving_ptick-Nil.prems(1) obtain t1' where ‹a ∉ A› ‹t1 = ev
a # t1'›
    ‹t1' setinterleaves✓tick-join ((u1, []), A)› by (auto split: if-split-asm)
  from ev-setinterleaving_ptick-Nil.hyps[OF this(1, 3)]
  show ?case
    by (cases v2)
      (auto simp add: ‹a ∉ A› ‹t1 = ev a # t1'› setinterleaving_ptick-simps
        split: event_ptick.split)
next
  case (tick-setinterleaving_ptick-Nil r u1)
  from tick-setinterleaving_ptick-Nil.prems(1) have False by simp
  thus ?case ..
next
  case (Nil-setinterleaving_ptick-ev b v1)
  from Nil-setinterleaving_ptick-ev.prems(1) obtain t1' where ‹b ∉ A› ‹t1 = ev b
# t1'›
    ‹t1' setinterleaves✓tick-join (([], v1), A)› by (auto split: if-split-asm)
  from Nil-setinterleaving_ptick-ev.hyps[OF this(1, 3)]
  show ?case
    by (cases u2)
      (auto simp add: ‹b ∉ A› ‹t1 = ev b # t1'› setinterleaving_ptick-simps
        split: event_ptick.split)
next
  case (Nil-setinterleaving_ptick-tick s v1)
  from Nil-setinterleaving_ptick-tick.prems(1) have False by simp
  thus ?case ..
next
  case (ev-setinterleaving_ptick-ev a u1 b v1)
  from ev-setinterleaving_ptick-ev.prems
  consider (mv-both) t' where ‹a ∈ A› ‹b ∈ A› ‹a = b› ‹t1 = ev a # t'›
    ‹t' setinterleaves✓tick-join ((u1, v1), A)›
  | (mvL) t' where ‹a ∉ A› ‹t1 = ev a # t'›
    ‹t' setinterleaves✓tick-join ((u1, ev b # v1), A)›
  | (mvR) t' where ‹b ∉ A› ‹t1 = ev b # t'›
    ‹t' setinterleaves✓tick-join ((ev a # u1, v1), A)›
    by (auto split: if-split-asm)
  thus ?case
  proof cases
    case mv-both
    from ev-setinterleaving_ptick-ev.hyps(1)[OF mv-both(1−3, 5)]
    show ?thesis by (simp add: mv-both(2−4))
  next
    case mvL
```

**from** *ev-setinterleaving$_{ptick}$-ev.hyps*(*3*, *4*)[*OF mvL*(*1*) - *mvL*(*3*)]
**show** *?thesis* **by** (*simp add*: *mvL*(*1*, *2*))
**next**
**case** *mvR*
**from** *ev-setinterleaving$_{ptick}$-ev.hyps*(*2*, *5*)[*OF* - *mvR*(*1*, *3*)]
**show** *?thesis* **by** (*simp add*: *mvR*(*1*, *2*))
**qed**
**next**
**case** (*ev-setinterleaving$_{ptick}$-tick a u1 s v1*)
**from** *ev-setinterleaving$_{ptick}$-tick.prems*(*1*)
**obtain** *t1'* **where** ‹*a* ∉ *A*› ‹*t1* = *ev a* # *t1'*›
‹*t1'* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1*, ✓(*s*) # *v1*), *A*)› **by** (*auto split*: *if-split-asm*)
**from** *ev-setinterleaving$_{ptick}$-tick.hyps*[*OF this*(*1*, *3*)]
**show** *?case*
**by** (*cases v2*)
(*auto simp add*: ‹*a* ∉ *A*› ‹*t1* = *ev a* # *t1'*› *setinterleaving$_{ptick}$-simps*
*split*: *event$_{ptick}$.split*)
**next**
**case** (*tick-setinterleaving$_{ptick}$-ev r u1 b v1*)
**from** *tick-setinterleaving$_{ptick}$-ev.prems*(*1*) **obtain** *t1'* **where** ‹*b* ∉ *A*› ‹*t1* = *ev b* # *t1'*›
‹*t1'* *setinterleaves*$_{✓\,tick\text{-}join}$ ((✓(*r*) # *u1*, *v1*), *A*)› **by** (*auto split*: *if-split-asm*)
**from** *tick-setinterleaving$_{ptick}$-ev.hyps*[*OF this*(*1*, *3*)]
**show** *?case*
**by** (*cases u2*)
(*auto simp add*: ‹*b* ∉ *A*› ‹*t1* = *ev b* # *t1'*› *setinterleaving$_{ptick}$-simps*
*split*: *event$_{ptick}$.split*)
**next**
**case** (*tick-setinterleaving$_{ptick}$-tick r u1 s v1*)
**from** *tick-setinterleaving$_{ptick}$-tick.prems*(*1*) **obtain** *r-s t1'*
**where** ‹*tick-join r s* = ⌊*r-s*⌋› ‹*t1* = ✓(*r-s*) # *t1'*›
‹*t1'* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*u1*, *v1*), *A*)›
**by** (*auto split*: *option.split-asm*)
**from** *tick-setinterleaving$_{ptick}$-tick.hyps*[*OF this*(*1*, *3*)]
**show** *?case* **by** (*simp add*: ‹*tick-join r s* = ⌊*r-s*⌋› ‹*t1* = ✓(*r-s*) # *t1'*›)
**qed**

**lemma** *setinterleaves$_{ptick}$-set-subsetL* :
‹*t* *setinterleaves*$_{✓\,tick\text{-}join}$ ((*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-setinterleaves$_{ptick}$*)
**next**
**case** (*Cons e t*)
**from** *Cons.prems* **consider** (*mv-left*) *a u'* **where** ‹*a* ∉ *A*› ‹*e* = *ev a*› ‹*u* = *ev a*

$\#\ u'\rangle$
  $\langle t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v),\ A)\rangle$
 | (*mv-right*) $a\ v'$ **where** $\langle a \notin A\rangle$ $\langle e = ev\ a\rangle$ $\langle v = ev\ a\ \#\ v'\rangle$
  $\langle t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v'),\ A)\rangle$
 | (*mv-both-ev*) $a\ u'\ v'$ **where** $\langle a \in A\rangle$ $\langle e = ev\ a\rangle$ $\langle u = ev\ a\ \#\ u'\rangle$ $\langle v = ev\ a\ \#$
$v'\rangle$
  $\langle t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v'),\ A)\rangle$
 | (*mv-both-tick*) $r\ s\ r\text{-}s\ u'\ v'$ **where** $\langle$*tick-join* $r\ s = \lfloor r\text{-}s\rfloor\rangle$ $\langle e = \checkmark(r\text{-}s)\rangle$
  $\langle u = \checkmark(r)\ \#\ u'\rangle$ $\langle v = \checkmark(s)\ \#\ v'\rangle$ $\langle t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u',\ v'),\ A)\rangle$
 **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** (*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** *setinterleaves*$_{ptick}$*-set-subsetR* :
 $\langle t$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$
  $\{a.\ ev\ a \in set\ (drop\ n\ v)\} \subseteq \{a.\ ev\ a \in set\ (drop\ n\ t)\}\rangle$
 **by** (*rule setinterleaves*$_{ptick}$*-set-subsetL*)
  (*fact setinterleaves*$_{ptick}$*-sym*[*THEN iffD2*])

## 4.2 Synchronization Product

### 4.2.1 Definition

**definition** *super-ref-Sync*$_{ptick}$ ::
 $\langle['r \Rightarrow 's \Rightarrow 't\ option,\ ('a,\ 'r)\ refusal_{ptick},\ 'a\ set,\ ('a,\ 's)\ refusal_{ptick}] \Rightarrow ('a,\ 't)$
*refusal*$_{ptick}\rangle$
 **where** $\langle$*super-ref-Sync*$_{ptick}$ *tick-join X-P A X-Q* $\equiv$
  $\{ev\ a\ |a.\ ev\ a \in X\text{-}P \wedge ev\ a \in X\text{-}Q \vee (a \in A \wedge (ev\ a \in X\text{-}P \vee ev\ a \in$
*X-Q*))$\}$ $\cup$
  $\{\checkmark(r\text{-}s)\ |r\ s\ r\text{-}s.\ tick\text{-}join\ r\ s = \lfloor r\text{-}s\rfloor \wedge (\checkmark(r) \in X\text{-}P \vee \checkmark(s) \in X\text{-}Q)\} \cup$
  — This is the last addition: since we generalize with the parameter *tick-join*,

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

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

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

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

**lift-definition** $Sync_{ptick}$ ::
  $\langle['a, 'r)\ process_{ptick}, 'a\ set, ('a, 's)\ process_{ptick}] \Rightarrow ('a, 't)\ process_{ptick}\rangle$
  $(\langle(\text{-}\ [\![\text{-}]\!]_{\checkmark}\ \text{-})\rangle\ [70, 0, 71]\ 70)$
  **is** $\langle\lambda P\ A\ Q.$
    $(\{(t, X).\ \exists\ t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q.$
      $(t\text{-}P, X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q \wedge$
      $t\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A) \wedge$
      $X \subseteq super\text{-}ref\text{-}Sync_{ptick}\ (\otimes\checkmark)\ X\text{-}P\ A\ X\text{-}Q\} \cup$
    $\{(t\ @\ u, X)\ |t\ u\ t\text{-}P\ t\text{-}Q\ X.$
      $ftF\ u \wedge (tF\ t \vee u = []) \wedge t\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A) \wedge$
      $(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\},$
    $\{t\ @\ u\ |t\ u\ t\text{-}P\ t\text{-}Q.$
      $ftF\ u \wedge (tF\ t \vee u = []) \wedge t\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A) \wedge$
      $(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\})\rangle$
**proof** −
  **show** $\langle ?thesis\ P\ A\ Q\rangle$
    (**is** $\langle is\text{-}process(?f, ?d)\rangle$) **for** $P$ **and** $Q$ :: $\langle('a, 's)\ process_{ptick}\rangle$ **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** $Nil\text{-}setinterleaving_{ptick}\text{-}Nil$ **show** $\langle([], \{\}) \in ?f\rangle$ **by** *fast*
  **next**
    **show** $\langle(t, X) \in ?f \Longrightarrow ftF\ t\rangle$ **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** $\langle(t\ @\ u, \{\}) \in ?f\rangle$
    **then consider** (*fail*) *t-P t-Q X-P X-Q* **where**
      $\langle(t\text{-}P, X\text{-}P) \in \mathcal{F}\ P\rangle\ \langle(t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q\rangle\ \langle t\ @\ u\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q), A)\rangle$
    | (*div*) $t'\ u'$ *t-P t-Q* **where**
      $\langle t\ @\ u = t'\ @\ u'\rangle\ \langle ftF\ u'\rangle\ \langle tF\ t' \vee u' = []\rangle\ \langle t'\ setinterleaves_{\checkmark(\otimes\checkmark)}\ ((t\text{-}P, t\text{-}Q),$

```
A)›
      ‹t-P ∈ 𝒟 P ∧ t-Q ∈ 𝒯 Q ∨ t-P ∈ 𝒯 P ∧ t-Q ∈ 𝒟 Q› by simp blast
    thus ‹(t, {}) ∈ ?f›
    proof cases
      case fail
      from fail(3) obtain t' u'
        where ∗ : ‹t' ≤ t-P› ‹u' ≤ t-Q› ‹t setinterleaves✓(⊗✓) ((t', u'), A)›
        by (auto dest!: append-setinterleaves_ptick intro: prefixI)
       from fail(1, 2) ∗(1, 2) F-T is-processT3-TR have ‹t' ∈ 𝒯 P› ‹u' ∈ 𝒯 Q›
by blast+
      thus ‹(t, {}) ∈ ?f› by simp (metis T-F-spec ∗(3))
    next
      case div
      show ‹(t, {}) ∈ ?f›
      proof (cases ‹length t' ≤ length t›)
        assume ‹length t' ≤ length t›
        with div(1−3) have ‹ftF (take (length t − length t') u') ∧
                            (tF t' ∨ take (length t − length t') u' = []) ∧
                            t = t' @ take (length t − length t') u'›
          by (simp add: append-eq-conv-conj)
            (metis append-take-drop-id front-tickFree-dw-closed)
        with div(4, 5) show ‹(t, {}) ∈ ?f› by blast
      next
        assume ‹¬ length t' ≤ length t›
        with div obtain r' where ‹t' = t @ r'›
          by (metis append-eq-append-conv-if append-take-drop-id)
        with div(4) obtain t'' u''
          where ∗ : ‹t'' ≤ t-P› ‹u'' ≤ t-Q› ‹t setinterleaves✓(⊗✓) ((t'', u''), A)›
          by (auto dest!: append-setinterleaves_ptick intro: prefixI)
           from ∗(1, 2) have ‹t'' ∈ 𝒯 P ∧ u'' ∈ 𝒯 Q› by (meson D-T div(5)
is-processT3-TR)
        hence $ : ‹(t'', {}) ∈ ℱ P› ‹(u'', {}) ∈ ℱ Q› by (simp-all add: T-F)
        have $$ : ‹{ev a| a. ev a ∈ {} ∧ ev a ∈ {} ∨ (a ∈ A ∧ (ev a ∈ {} ∨ ev a ∈
{}))} ∪
                    {✓(r ⊗✓ s)| r s. ✓(r) ∈ {} ∨ ✓(s) ∈ {}} = {}› by simp
        show ‹(t, {}) ∈ ?f› by (auto intro!: $ ∗(3))
      qed
    qed
  next
    { fix t X Y
      assume ‹(t, Y) ∈ ?f ∧ X ⊆ Y›
      then consider ‹t ∈ ?d›
        | (fail) t-P t-Q X-P X-Q where
          ‹(t-P, X-P) ∈ ℱ P› ‹(t-Q, X-Q) ∈ ℱ Q›
          ‹t setinterleaves✓(⊗✓) ((t-P, t-Q), A)›
          ‹Y ⊆ super-ref-Sync_ptick (⊗✓) X-P A X-Q› by blast
      thus ‹(t, X) ∈ ?f›
      proof cases
        show ‹t ∈ ?d ⟹ (t, X) ∈ ?f› by blast
```

**next**
**case** *fail*
**define** *X-P′* **where** ‹*X-P′* ≡ *X-P* ∩ ({*ev a*| *a*. *ev a* ∈ *X*} ∪
{✓(*r*)| *r s r-s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧ ✓(*r-s*) ∈ *X*})›
**define** *X-Q′* **where** ‹*X-Q′* ≡ *X-Q* ∩ ({*ev a*| *a*. *ev a* ∈ *X*} ∪
{✓(*s*)| *r s r-s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧ ✓(*r-s*) ∈ *X*})›
**have** ‹(*t-P*, *X-P′*) ∈ *F P*› **unfolding** *X-P′-def* **by** (*meson fail*(*1*) *inf-le1 process-charn*)
**moreover have** ‹(*t-Q*, *X-Q′*) ∈ *F Q*› **unfolding** *X-Q′-def* **by** (*meson fail*(*2*) *inf-le1 process-charn*)
**moreover have** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P′ A X-Q′*›
**by** (*subst* ‹(*t*, *Y*) ∈ *?f* ∧ *X* ⊆ *Y*›[*THEN conjunct2*, *THEN Int-absorb1*, *symmetric*])
(*use fail*(*4*) **in** ‹*fastforce simp add*: *X-P′-def X-Q′-def subset-iff super-ref-Sync*$_{ptick}$*-def*›)
**ultimately show** ‹(*t*, *X*) ∈ *?f*› **using** *fail*(*3*) **by** *simp blast*
**qed** } **note** *processT4* = *this*

**fix** *t X Y*
**assume** ‹(*t*, *X*) ∈ *?f* ∧ (∀ *e*. *e* ∈ *Y* ⟶ (*t* @ [*e*], {}) ∉ *?f*)›
**then consider** ‹*t* ∈ *?d*› | ‹(*t*, *X*) ∈ *?f* ∧ *t* ∉ *?d*› **by** *linarith*
**thus** ‹(*t*, *X* ∪ *Y*) ∈ *?f*›
**proof** *cases*
**show** ‹*t* ∈ *?d* ⟹ (*t*, *X* ∪ *Y*) ∈ *?f*› **by** *blast*
**next**
**assume** ‹(*t*, *X*) ∈ *?f* ∧ *t* ∉ *?d*›
**then obtain** *t-P X-P t-Q X-Q*
**where** *assms* : ‹(*t-P*, *X-P*) ∈ *F P*› ‹(*t-Q*, *X-Q*) ∈ *F Q*›
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), *A*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P A X-Q*› **by** *blast*
**have** *assms5* : ‹*e* ∈ *Y* ⟹ *t* @ [*e*] *setinterleaves*$_{✓(⊗✓)}$ ((*t′*, *u′*), *A*) ⟹
((*t′*, {}) ∈ *F P* ⟶ (*u′*, {}) ∉ *F Q*) ∧
((*u′*, {}) ∈ *F Q* ⟶ (*t′*, {}) ∉ *F P*)› **for** *e t′ u′*
**using** ‹(*t*, *X*) ∈ *?f* ∧ (∀ *e*. *e* ∈ *Y* ⟶ (*t* @ [*e*], {}) ∉ *?f*)› **by** *auto*

**define** *Y-ev-inside* **and** *Y-ev-notin* **and** *Y-tick*
**where** ∗ : ‹*Y-ev-inside* ≡ {*a*. *ev a* ∈ *Y* ∧ *a* ∈ *A*}›
‹*Y-ev-notin* ≡ {*a*. *ev a* ∈ *Y* ∧ *a* ∉ *A*}›
‹*Y-tick* ≡ {*r-s* |*r s r-s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧ ✓(*r-s*) ∈ *Y*}›

**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** ∗∗ : ‹*Y-ev-inside-P* ≡ {*a*∈*Y-ev-inside*. (*t-P* @ [*ev a*], {}) ∉ *F P*}›
‹*Y-ev-inside-Q* ≡ {*a*∈*Y-ev-inside*. (*t-Q* @ [*ev a*], {}) ∉ *F Q*}›
‹*Y-ev-notin-P* ≡ {*a*∈*Y-ev-notin*. (*t-P* @ [*ev a*], {}) ∉ *F P*}›
‹*Y-ev-notin-Q* ≡ {*a*∈*Y-ev-notin*. (*t-Q* @ [*ev a*], {}) ∉ *F Q*}›
‹*Y-tick-P* ≡ {*r-s*∈*Y-tick*. ∃ *r s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧ (*t-P* @ [✓(*r*)], {}) ∉ *F P*}›
‹*Y-tick-Q* ≡ {*r-s*∈*Y-tick*. ∃ *r s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧ (*t-Q* @ [✓(*s*)], {})

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

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

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

**define** *X-P′* **and** *X-Q′*

```
      where *** : ‹X-P' ≡ X-P ∪ ev ‘ Y-ev-inside-P ∪ ev ‘ Y-ev-notin-P ∪
                        {✓(r) |r s r-s. r ⊗✓ s = ⌊r-s⌋ ∧ r-s ∈ Y-tick-P}›
        ‹X-Q' ≡ X-Q ∪ ev ‘ Y-ev-inside-Q ∪ ev ‘ Y-ev-notin-Q ∪
              {✓(s) |r s r-s. r ⊗✓ s = ⌊r-s⌋ ∧ r-s ∈ Y-tick-Q}›

    have $ : ‹(t-P, X-P') ∈ ℱ P› ‹(t-Q, X-Q') ∈ ℱ Q›
      by (auto simp add: ** *** intro!: is-processT5 assms dest: inj-tick-join)

    have ‹Y ⊆ super-ref-Sync_ptick (⊗✓) X-P' A X-Q'›
    proof (rule subsetI)
      show ‹e ∈ super-ref-Sync_ptick (⊗✓) X-P' A X-Q'› if ‹e ∈ Y› for e
      proof (cases e)
        from ‹e ∈ Y› show ‹e = ev a ⟹ e ∈ super-ref-Sync_ptick (⊗✓) X-P' A
X-Q'› for a
        by (cases ‹a ∈ A›, simp-all add: * ** *** image-iff super-ref-Sync_ptick-def)
            (use *(1) ∈ in blast,
              meson $(2) assms(1, 3) assms5 is-processT4-empty
              setinterleaves_ptick-snoc-notinL setinterleaves_ptick-snoc-notinR)
      next
       show ‹e ∈ super-ref-Sync_ptick (⊗✓) X-P' A X-Q'› if ‹e = ✓(r-s)› for r-s
       proof (cases ‹∃ r s. r ⊗✓ s = ⌊r-s⌋›)
         assume ‹∃ r s. r ⊗✓ s = ⌊r-s⌋›
         then obtain r s where ‹r ⊗✓ s = ⌊r-s⌋› by blast
         with ‹e ∈ Y› ‹e = ✓(r-s)› have ‹r-s ∈ Y-tick›
           by (auto simp add: *)
         thus ‹e ∈ super-ref-Sync_ptick (⊗✓) X-P' A X-Q'›
           by (simp add: * super-ref-Sync_ptick-def)
             (metis (mono-tags, lifting) *(3) ***(1,2) ℒℒℒℒ
               Un-iff mem-Collect-eq ‹e = ✓(r-s)›)
       next
         show ‹∄ r s. r ⊗✓ s = ⌊r-s⌋ ⟹
               e ∈ super-ref-Sync_ptick (⊗✓) X-P' A X-Q'›
           by (simp add: ‹e = ✓(r-s)› super-ref-Sync_ptick-def)
       qed
      qed
     moreover from assms(4) have ‹X ⊆ super-ref-Sync_ptick (⊗✓) X-P' A X-Q'›
       by (fastforce simp add: *** subset-iff super-ref-Sync_ptick-def)
     ultimately show ‹(t, X ∪ Y) ∈ ?f› using $ assms(3) by auto
    qed
  next
    show processT9: ‹t ∈ ?d› if ‹t @ [✓(r-s)] ∈ ?d› for t r-s
    proof −
      from ‹t @ [✓(r-s)] ∈ ?d› obtain u v t-P t-Q
        where assms : ‹ftF v› ‹tF u ∨ v = []›
          ‹t @ [✓(r-s)] = u @ v›
          ‹u setinterleaves_✓(⊗✓) ((t-P, t-Q), A)›
          ‹t-P ∈ 𝒟 P ∧ t-Q ∈ 𝒯 Q ∨ t-P ∈ 𝒯 P ∧ t-Q ∈ 𝒟 Q› by blast
      from assms(2) show ‹t ∈ ?d›
```

**proof** (*elim disjE*)
**assume** ‹*tF u*›
**with** *assms*(*3*) **obtain** *v'* **where** ‹*v* = *v'* @ [✓(*r-s*)]› ‹*t* = *u* @ *v'*›
**by** (*cases v rule*: *rev-cases*) *auto*
**from** ‹*v* = *v'* @ [✓(*r-s*)]› *assms*(*1*) *front-tickFree-dw-closed*
**have** ‹*ftF v'*› **by** *blast*
**with** ‹*t* = *u* @ *v'*› ‹*tF u*› *assms*(*1*, *4*, *5*) **show** ‹*t* ∈ *?d*› **by** *blast*
**next**
**assume** ‹*v* = []›
**with** *assms*(*3*) **obtain** *u'* **where** ‹*u* = *u'* @ [✓(*r-s*)]› ‹*t* = *u'*› **by** *auto*
**from** *snoc-tick-setinterleaves*$_{ptick}$*E*[*OF assms*(*4*)[*unfolded this*(*1*)]]
**obtain** *r s t-P' t-Q'* **where** ‹*r* ⊗✓ *s* = ⌊*r-s*⌋›
‹*u' setinterleaves*$_{✓(⊗✓)}$ ((*t-P'*, *t-Q'*), *A*)›
‹*t-P* = *t-P'* @ [✓(*r*)]› ‹*t-Q* = *t-Q'* @ [✓(*s*)]› **by** *metis*
**with** *assms*(*5*) ‹*t* = *u'*› **show** ‹*t* ∈ *?d*›
**by** *simp* (*metis append.right-neutral front-tickFree-Nil*
*is-processT3-TR-append is-processT9*)
**qed**
**qed**

**fix** *t X r-s*
**assume** ‹(*t* @ [✓(*r-s*)], {}) ∈ *?f*›
**then consider** (*div*) ‹*t* @ [✓(*r-s*)] ∈ *?d*›
| (*fail*) *t-P t-Q X-P X-Q*
**where** ‹(*t-P*, *X-P*) ∈ ℱ *P*› ‹(*t-Q*, *X-Q*) ∈ ℱ *Q*›
‹(*t* @ [✓(*r-s*)]) *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), *A*)› **by** *auto*
**thus** ‹(*t*, *X* − {✓(*r-s*)}) ∈ *?f*›
**proof** *cases*
**show** ‹*t* @ [✓(*r-s*)] ∈ *?d* ⟹ (*t*, *X* − {✓(*r-s*)}) ∈ *?f*› **by** (*drule processT9*)
*simp*
**next**
**case** *fail*
**from** *fail*(*3*)[*THEN snoc-tick-setinterleaves*$_{ptick}$*E*]
**obtain** *r s t-P' t-Q'* **where** ∗ : ‹*r* ⊗✓ *s* = ⌊*r-s*⌋›
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*t-P'*, *t-Q'*), *A*)›
‹*t-P* = *t-P'* @ [✓(*r*)]› ‹*t-Q* = *t-Q'* @ [✓(*s*)]› **by** *metis*
**from** *fail*(*1*, *2*) **have** ‹*t-P'* @ [✓(*r*)] ∈ 𝒯 *P*› ‹*t-Q'* @ [✓(*s*)] ∈ 𝒯 *Q*›
**by** (*simp-all add*: ∗(*3*, *4*) *F-T*)
**hence** ‹(*t-P'*, *UNIV* − {✓(*r*)}) ∈ ℱ *P*›
‹(*t-Q'*, *UNIV* − {✓(*s*)}) ∈ ℱ *Q*› **by** (*meson is-processT6-TR*)+
**moreover have** ‹*X* − {✓(*r-s*)} ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) (*UNIV* − {✓(*r*)})
*A* (*UNIV* − {✓(*s*)})›
**by** (*simp add*: *subset-iff super-ref-Sync*$_{ptick}$*-def*)
(*metis* ∗(*1*) *event*$_{ptick}$*.exhaust option.inject*)
**ultimately show** ‹(*t*, *X* − {✓(*r-s*)}) ∈ *?f*› **using** ∗(*2*) **by** *fast*
**qed**
**next**
**show** ‹*s* ∈ *?d* ∧ *tF s* ∧ *ftF t* ⟹ *s* @ *t* ∈ *?d*› **for** *s t*
**using** *front-tickFree-append* **by** *fastforce*

**next**
**show** ‹$s \in ?d \Longrightarrow (s, X) \in ?f$› **for** $s$ $X$ **by** *blast*
**qed**
**qed**

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

**end**

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

**abbreviation** (**in** *Sync*$_{ptick}$-*locale*) *Par*$_{ptick}$ ::
‹[('$a$, '$r$) *process*$_{ptick}$, ('$a$, '$s$) *process*$_{ptick}$] ⇒
('$a$, '$t$) *process*$_{ptick}$› (‹(- $||_{\checkmark}$ -)› [74, 75] 74)
**where** ‹$P$ $||_{\checkmark}$ $Q \equiv P$ ⟦ *UNIV* ⟧$_{\checkmark}$ $Q$›

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

**notation** (**in** *Sync*$_{ptick}$-*locale*) *Sync*$_{ptick}$-*locale-sym.Inter*$_{ptick}$
(‹(- $|||_{\checkmark sym}$ -)› [72, 73] 72)

**notation** (**in** *Sync*$_{ptick}$-*locale*) *Sync*$_{ptick}$-*locale-sym.Par*$_{ptick}$
(‹(- $||_{\checkmark sym}$ -)› [74, 75] 74)

### 4.2.2 Projections

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

**lemma** *D-Sync*$_{ptick}$$'$ :
‹$\mathcal{D}$ ($P$ ⟦$A$⟧$_{\checkmark}$ $Q$) =
{$t$ @ $u$ |$t$ $u$ *t-P t-Q*.
*ftF* $u$ ∧ (*tF* $t$ ∨ $u = []$) ∧ $t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), $A$) ∧
(*t-P* ∈ $\mathcal{D}$ $P$ ∧ *t-Q* ∈ $\mathcal{T}$ $Q$ ∨ *t-P* ∈ $\mathcal{T}$ $P$ ∧ *t-Q* ∈ $\mathcal{D}$ $Q$)}›
**by** (*simp add*: *Divergences.rep-eq Sync*$_{ptick}$*.rep-eq DIVERGENCES-def*)

**corollary** *D-Sync*$_{ptick}$ :
— This version is easier to use.
‹$\mathcal{D}$ ($P$ ⟦$A$⟧$_{\checkmark}$ $Q$) =
{$t$ @ $u$ |$t$ $u$ *t-P t-Q*.
*tF* $t$ ∧ *ftF* $u$ ∧ $t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), $A$) ∧
(*t-P* ∈ $\mathcal{D}$ $P$ ∧ *t-Q* ∈ $\mathcal{T}$ $Q$ ∨ *t-P* ∈ $\mathcal{T}$ $P$ ∧ *t-Q* ∈ $\mathcal{D}$ $Q$)}›
(**is** ‹- = *?rhs*›)
**proof** (*intro subset-antisym subsetI*)

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

**lemma** *F-Sync*$_{ptick}{}'$ :
‹$\mathcal{F}\ (P\ [\![A]\!]_{\checkmark}\ Q) =$
$\{(t,\ X).\ \exists$ *t-P t-Q X-P X-Q*.
$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ Q \wedge$
$t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$
$X \subseteq$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)$ *X-P A X-Q*$\} \cup$
$\{(t\ @\ u,\ X)\ |t\ u$ *t-P t-Q X*.
*ftF* $u \wedge (tF\ t \vee u = []) \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}$›
**by** (*simp add*: *Failures.rep-eq Sync*$_{ptick}$*.rep-eq FAILURES-def*)

**lemma** *F-Sync*$_{ptick}$ :
‹$\mathcal{F}\ (P\ [\![A]\!]_{\checkmark}\ Q) =$
$\{(t,\ X).\ \exists$ *t-P t-Q X-P X-Q*.
$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}\ P \wedge (t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ Q \wedge$
$t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A) \wedge$

$X \subseteq$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)$ *X-P A X-Q*$\}$ $\cup$
$\{(t$ @ $u,\ X)\ |t\ u$ *t-P t-Q X*.
*tF t* $\wedge$ *ftF u* $\wedge$ *t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)$ $\wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}\rangle$
**unfolding** *F-Sync*$_{ptick}{}'$ **using** *D-Sync*$_{ptick}$[*of P A Q*, *unfolded D-Sync*$_{ptick}{}'$]
**by** (*intro arg-cong2*[**where** $f = \langle(\cup)\rangle$], *simp*)
(*simp add*: *set-eq-iff*, *blast*)

**lemma** *T-Sync*$_{ptick}{}'$ :
$\langle\mathcal{T}\ (P\ [\![A]\!]_{\checkmark}\ Q) =$
$\{t.\ \exists\, t\text{-}P\ t\text{-}Q.\ t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)\}$
$\cup$
$\{t$ @ $u\ |t\ u\ t\text{-}P\ t\text{-}Q.$
*ftF u* $\wedge$ $(tF\ t \vee u = [])\ \wedge$
*t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)\ \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}\rangle$
**by** (*simp add*: *Traces.rep-eq TRACES-def Failures.rep-eq*[*symmetric*] *F-Sync*$_{ptick}{}'$)
*blast*

**lemma** *T-Sync*$_{ptick}$ :
$\langle\mathcal{T}\ (P\ [\![A]\!]_{\checkmark}\ Q) =$
$\{t.\ \exists\, t\text{-}P\ t\text{-}Q.\ t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \wedge t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)\}$
$\cup$
$\{t$ @ $u\ |t\ u\ t\text{-}P\ t\text{-}Q.$
*tF t* $\wedge$ *ftF u* $\wedge$ *t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ A)\ \wedge$
$(t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q)\}\rangle$
**unfolding** *T-Sync*$_{ptick}{}'$ **using** *D-Sync*$_{ptick}$[*of P A Q*, *unfolded D-Sync*$_{ptick}{}'$]
**by** (*intro arg-cong2*[**where** $f = \langle(\cup)\rangle$]) (*simp-all add*: *set-eq-iff*)

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

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

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *Sync*$_{ptick}$*-same-tick-join-on-strict-ticks-of* :
$\langle$*Sync*$_{ptick}$*-locale.Sync*$_{ptick}$ *tick-join*$'$ *P S Q* $=$ *P* $[\![S]\!]_{\checkmark}$ *Q*$\rangle$
**if** $\langle$*Sync*$_{ptick}$*-locale tick-join*$'\rangle$ **and** $\langle\bigwedge r\ s.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow s \in \checkmark\mathbf{s}(Q) \Longrightarrow$ *tick-join*$'$ *r s* $=$ *r* $\otimes\checkmark$ *s*$\rangle$
**proof** −
**interpret** *tjoin-interpreted* : *Sync*$_{ptick}$*-locale tick-join*$'$
**by** (*fact* $\langle$*Sync*$_{ptick}$*-locale tick-join*$'\rangle$)
**show** $\langle$*Sync*$_{ptick}$*-locale.Sync*$_{ptick}$ *tick-join*$'$ *P S Q* $=$ *P* $[\![S]\!]_{\checkmark}$ *Q*$\rangle$
**proof** (*rule Process-eq-optimizedI*)
**show** $\langle t \in \mathcal{D}$ (*tjoin-interpreted.Sync*$_{ptick}$ *P S Q*$) \Longrightarrow t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)\rangle$ **for** *t*
**by** (*simp add*: *D-Sync*$_{ptick}$ *tjoin-interpreted.D-Sync*$_{ptick}$)
(*metis tickFree-setinterleaves*$_{ptick}$*-any-tick-join*)

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

**moreover from** $*(2)$ *is-processT5-S7′ is-processT8 is-processT9*
**have** ‹(*t-Q*, *X-Q-plus*) ∈ $\mathcal{F}$ *Q*› **by** (*fastforce simp add*: *X-Q-plus-def*)
**moreover have** ‹*e* ∈ *X* ⟹ *e* ∈ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P-plus S X-Q-plus*›
**for** *e*
**using** $*(4)$[*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) *tjoin-interpreted.inj-tick-join*)
**ultimately show** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)› **by** (*simp add*: *F-Sync*$_{ptick}$) *blast*
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)› ‹*t* ∉ $\mathcal{D}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)›
**then obtain** *t-P X-P t-Q X-Q* **where** ∗ : ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), *S*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**define** *X-P-plus* **where** ‹*X-P-plus* ≡ *X-P* ∪ {✓(*r*) |*r*. *t-P* @ [✓(*r*)] ∉ $\mathcal{T}$ *P* − $\mathcal{D}$ *P*}›
**define** *X-Q-plus* **where** ‹*X-Q-plus* ≡ *X-Q* ∪ {✓(*s*) |*s*. *t-Q* @ [✓(*s*)] ∉ $\mathcal{T}$ *Q* − $\mathcal{D}$ *Q*}›
**have** ‹*t setinterleaves*$_{\checkmark\,tick\text{-}join'}$ ((*t-P*, *t-Q*), *S*)›
**proof** (*cases* ‹*tF t*›)
**show** ‹*tF t* ⟹ *t setinterleaves*$_{\checkmark\,tick\text{-}join'}$ ((*t-P*, *t-Q*), *S*)›
**using** $*(3)$ *tickFree-setinterleaves*$_{ptick}$*-any-tick-join* **by** *blast*
**next**
**assume** ‹¬ *tF t*›
**then obtain** *t′ r-s* **where** ‹*tF t′*› ‹*t* = *t′* @ [✓(*r-s*)]›
**by** (*metis F-imp-front-tickFree* ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)›
*front-tickFree-append-iff nonTickFree-n-frontTickFree not-Cons-self2*)
**with** $*(3)$ **obtain** *t-P′ r t-Q′ s* **where** ∗∗ : ‹*r* ⊗✓ *s* = ⌊*r-s*⌋›
‹*t′ setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P′*, *t-Q′*), *S*)›
‹*t-P* = *t-P′* @ [✓(*r*)]› ‹*t-Q* = *t-Q′* @ [✓(*s*)]›
**by** (*auto elim*: *snoc-tick-setinterleaves*$_{ptick}$*E*)
**have** ‹*r* ∈ ✓**s**(*P*) ∧ *s* ∈ ✓**s**(*Q*)›
**proof** (*rule ccontr*)
**assume** ‹¬ (*r* ∈ ✓**s**(*P*) ∧ *s* ∈ ✓**s**(*Q*))›
**hence** ‹*t-P′* @ [✓(*r*)] ∈ $\mathcal{D}$ *P* ∨ *t-Q′* @ [✓(*s*)] ∈ $\mathcal{D}$ *Q*›
**by** (*metis* $*(1, 2)$ $**(3, 4)$ *F-T strict-ticks-of-memI*)
**with** ‹*t* ∉ $\mathcal{D}$ (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)› **show** *False*
**by** (*simp add*: *D-Sync*$_{ptick}$*′*)
(*metis* $*(1-3)$ $**(3, 4)$ *F-T append.right-neutral front-tickFree-Nil*)
**qed**
**moreover from** $**(2)$ **have** ‹*t′ setinterleaves*$_{\checkmark\,tick\text{-}join'}$ ((*t-P′*, *t-Q′*), *S*)›
**using** ‹*tF t′*› *tickFree-setinterleaves*$_{ptick}$*-any-tick-join* **by** *blast*
**ultimately show** ‹*t setinterleaves*$_{\checkmark\,tick\text{-}join'}$ ((*t-P*, *t-Q*), *S*)›
**by** (*subst rev-setinterleaves*$_{ptick}$*-rev-rev-iff*[*symmetric*],
*subst* (*asm*) *rev-setinterleaves*$_{ptick}$*-rev-rev-iff*[*symmetric*])
(*use* $**(1)$ *that*(2) **in** ‹*auto simp add*: ‹*t* = *t′* @ [✓(*r-s*)]› $**(3, 4)$›)

**qed**
**moreover from** *∗(1) is-processT5-S7′ is-processT8 is-processT9*
**have** ‹*(t-P, X-P-plus) ∈ ℱ P*› **by** (*fastforce simp add*: *X-P-plus-def*)
**moreover from** *∗(2) is-processT5-S7′ is-processT8 is-processT9*
**have** ‹*(t-Q, X-Q-plus) ∈ ℱ Q*› **by** (*fastforce simp add*: *X-Q-plus-def*)
**moreover have** ‹*e ∈ X ⟹ e ∈ super-ref-Sync*$_{ptick}$ *tick-join′ X-P-plus S X-Q-plus*› **for** *e*
**using** *∗(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** ‹*(t, X) ∈ ℱ* (*tjoin-interpreted.Sync*$_{ptick}$ *P S Q*)›
**by** (*simp add*: *tjoin-interpreted.F-Sync*$_{ptick}$) *blast*
**qed**
**qed**

### 4.2.3 First Properties

**abbreviation** *range-tick-join* :: ‹*'t set*›
**where** ‹*range-tick-join* ≡ {*r-s* |*r-s r s. r* ⊗✓ *s* = ⌊*r-s*⌋}›

**lemma** *setinterleaves*$_{ptick}$*-imp-set-range-tick-join* :
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*u*, *v*), *A*) ⟹
{*r-s*. ✓(*r-s*) ∈ *set t*} ⊆ *range-tick-join*›
**by** (*induct* ‹((⊗✓), *u*, *A*, *v*)› *arbitrary*: *t u v*)
(*auto simp add*: *subset-iff split*: *if-split-asm option.split-asm*)+

**end**

**lemma**
— Of course not suitable for simplifier.
‹*t setinterleaves*$_{✓λs\ r.\ tick\text{-}join\ r\ s}$ ((*v*, *u*), *A*) ⟷
*t setinterleaves*$_{✓λr\ s.\ tick\text{-}join\ r\ s}$ ((*u*, *v*), *A*)›
**by** (*fact setinterleaves*$_{ptick}$*-sym*)

**lemma** *super-ref-Sync*$_{ptick}$*-sym* :
— Of course not suitable for simplifier.
‹*super-ref-Sync*$_{ptick}$ (*λs r. tick-join r s*) *X-Q S X-P* =
*super-ref-Sync*$_{ptick}$ (*λr s. tick-join r s*) *X-P S X-Q*›
**by** (*auto simp add*: *super-ref-Sync*$_{ptick}$*-def*)

**lemma** *super-ref-Sync*$_{ptick}$*-mono* :
‹*A ⊆ A′ ⟹ X-P ⊆ X-P′ ⟹ X-Q ⊆ X-Q′ ⟹*
*super-ref-Sync*$_{ptick}$ *tick-join X-P A X-Q ⊆*
*super-ref-Sync*$_{ptick}$ *tick-join X-P′ A′ X-Q′*›
**by** (*auto simp add*: *super-ref-Sync*$_{ptick}$*-def*)

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

**lemma** $Sync_{ptick}$-*sym* : ‹$Q$ $[\![A]\!]_{✓sym}$ $P$ = $P$ $[\![A]\!]_{✓}$ $Q$›
**proof** (*rule Process-eq-optimizedI*)
**show** ‹$t \in \mathcal{D}$ ($Q$ $[\![A]\!]_{✓sym}$ $P$) $\Longrightarrow$ $t \in \mathcal{D}$ ($P$ $[\![A]\!]_{✓}$ $Q$)› **for** $t$
**by** (*simp add*: $Sync_{ptick}$-*locale-sym.D-Sync*$_{ptick}$ *D-Sync*$_{ptick}$)
(*subst setinterleaves*$_{ptick}$-*sym*, *blast*)
**next**
**show** ‹$t \in \mathcal{D}$ ($P$ $[\![A]\!]_{✓}$ $Q$) $\Longrightarrow$ $t \in \mathcal{D}$ ($Q$ $[\![A]\!]_{✓sym}$ $P$)› **for** $t$
**by** (*simp add*: $Sync_{ptick}$-*locale-sym.D-Sync*$_{ptick}$ *D-Sync*$_{ptick}$)
(*subst setinterleaves*$_{ptick}$-*sym*, *blast*)
**next**
**show** ‹($t$, $X$) $\in \mathcal{F}$ ($Q$ $[\![A]\!]_{✓sym}$ $P$) $\Longrightarrow$ ($t$, $X$) $\in \mathcal{F}$ ($P$ $[\![A]\!]_{✓}$ $Q$)› **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** ‹($t$, $X$) $\in \mathcal{F}$ ($P$ $[\![A]\!]_{✓}$ $Q$) $\Longrightarrow$ ($t$, $X$) $\in \mathcal{F}$ ($Q$ $[\![A]\!]_{✓sym}$ $P$)› **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* :
‹*inj-on g range-tick-join* $\Longrightarrow$
$Sync_{ptick}$-*locale* ($\lambda r\ s$. *case r* $\otimes$✓ *s of* $\lfloor$*r-s*$\rfloor$ $\Rightarrow$ $\lfloor$*g r-s*$\rfloor$ | $\Diamond$ $\Rightarrow$ $\Diamond$)›
**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*$_{ptick}$ :
‹$t$ *setinterleaves*$_{✓(\otimes✓)}$ (($u$, $v$), $A$) $\Longrightarrow$
*map* (*map-event*$_{ptick}$ *id g*) $t$
*setinterleaves*$_{✓}$ $\lambda r$ *s. case r* $\otimes$✓ *s of* $\lfloor$*r-s*$\rfloor$ $\Rightarrow$ $\lfloor$*g r-s*$\rfloor$ | $\Diamond$ $\Rightarrow$ $\Diamond$ (($u$, $v$), $A$)›
(**is** ‹- $\Longrightarrow$ - *setinterleaves*$_{✓}$ *?tick-join*$'$ (($u$, $v$), $A$)›)
**if** *inj-on-g* : ‹*inj-on g range-tick-join*›
**proof** (*induct* ‹((⊗✓), $u$, $A$, $v$)› *arbitrary*: $t$ $u$ $v$)
**case** (*tick-setinterleaving*$_{ptick}$-*tick r u s v*)
**from** *tick-setinterleaving*$_{ptick}$-*tick.prems* [*simplified*]
**obtain** *r-s t*$'$ **where** $*$ : ‹$r$ $\otimes$✓ $s$ = $\lfloor$*r-s*$\rfloor$› ‹$t$ = ✓(*r-s*) # $t'$›
‹$t'$ *setinterleaves*$_{✓(\otimes✓)}$ (($u$, $v$), $A$)›
**by** (*auto split*: *option.split-asm*)
**from** *tick-setinterleaving*$_{ptick}$-*tick.hyps*[*OF* $*$(*1*, *3*)]
**have** ‹*map* (*map-event*$_{ptick}$ *id g*) $t'$
*setinterleaves*$_{✓}$ *?tick-join*$'$ (($u$, $v$), $A$)› .

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



lemma vimage-inj-on-subset-super-ref-Sync_ptick-iff :
  ‹map-event_ptick id g −‘ X ⊆
   super-ref-Sync_ptick (⊗✓) X-P A X-Q ⟷
   X ⊆ super-ref-Sync_ptick (λr s. case r ⊗✓ s of ⌊r-s⌋ ⇒ ⌊g r-s⌋ | ◊ ⇒ ◊) X-P A
X-Q›
  (is ‹?lhs1 ⊆ ?lhs2 ⟷ X ⊆ ?rhs›)
  if inj-on-g : ‹inj-on g range-tick-join›
proof −
  let ?tick-join′ = ‹λr s. case r ⊗✓ s of ⌊r-s⌋ ⇒ ⌊g r-s⌋ | ◊ ⇒ ◊›
  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 :
    ‹tick-join =
    (λr s. case ?tick-join′ r s of ◊ ⇒ ◊ | ⌊r-s⌋ ⇒ ⌊inv-into range-tick-join g r-s⌋)›
    by (fastforce split: split: option.split)
  let ?f1 = ‹map-event_ptick id g›
  let ?f2 = ‹map-event_ptick id (inv-into range-tick-join g)›
  show ‹?lhs1 ⊆ ?lhs2 ⟷ X ⊆ ?rhs›
  proof (intro iffI subsetI)
    show ‹e ∈ ?rhs› if ‹?lhs1 ⊆ ?lhs2› ‹e ∈ X› for e
    proof (cases e)
      fix a assume ‹e = ev a›
      with ‹e ∈ X› have ‹?f2 e ∈ ?f1 −‘ X› by simp
      with ‹?lhs1 ⊆ ?lhs2› have ‹?f2 e ∈ ?lhs2› by blast
      with ‹e = ev a› show ‹e ∈ ?rhs›
        by (auto simp add: super-ref-Sync_ptick-def)
    next
      show ‹e ∈ ?rhs› if ‹e = ✓(r-s)› for r-s
      proof (cases ‹∃ r s. ?tick-join′ r s = ⌊r-s⌋›)
        from ‹e = ✓(r-s)› show ‹∄ r s. ?tick-join′ r s = ⌊r-s⌋ ⟹ e ∈ ?rhs›
          by (simp add: super-ref-Sync_ptick-def)
      next
        assume ‹∃ r s. ?tick-join′ r s = ⌊r-s⌋›
        with ‹e = ✓(r-s)› ‹e ∈ X›
        have ‹?f2 e ∈ ?f1 −‘ X›
          by (auto split: option.split-asm)
            (metis (no-types, lifting) expanded-tick-join option.simps(5))
        with ‹?lhs1 ⊆ ?lhs2› have ‹?f2 e ∈ ?lhs2› by blast
        with ‹e = ✓(r-s)› show ‹e ∈ ?rhs›
          by (simp add: super-ref-Sync_ptick-def)
            (metis (no-types, lifting) expanded-tick-join option.simps(5))
      qed
    qed
  next
```

```
    show ‹e ∈ ?lhs2› if ‹X ⊆ ?rhs› and ‹e ∈ ?lhs1› for e
    proof (cases e)
      fix a assume ‹e = ev a›
      with ‹e ∈ ?lhs1› have ‹ev a ∈ X› by simp
      with ‹X ⊆ ?rhs› have ‹ev a ∈ ?rhs› by blast
      thus ‹e ∈ ?lhs2› by (auto simp add: ‹e = ev a› super-ref-Sync_ptick-def)
    next
      show ‹e ∈ ?lhs2› if ‹e = ✓(s-r)› for s-r
      proof (cases ‹∃ s r. tick-join s r = ⌊s-r⌋›)
        from ‹e = ✓(s-r)› show ‹∄ s r. tick-join s r = ⌊s-r⌋ ⟹ e ∈ ?lhs2›
          by (simp add: super-ref-Sync_ptick-def)
      next
        assume ‹∃ s r. tick-join s r = ⌊s-r⌋›
        with ‹e = ✓(s-r)› ‹e ∈ ?lhs1›
        have ‹✓(g s-r) ∈ X› by simp
        with ‹X ⊆ ?rhs› have ‹✓(g s-r) ∈ ?rhs› by blast
        with ‹e = ✓(s-r)› show ‹e ∈ ?lhs2›
          by (simp add: super-ref-Sync_ptick-def)
            (metis Sync_ptick'.inj-tick-join option.simps(5))
      qed
    qed
  qed
qed
```

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

```
lemma finite-setinterleaves_ptick-tick-join :
  ‹finite {(u, v). t setinterleaves✓(⊗✓) ((u, v), A)}›
  (is ‹finite {(u, v). ?f t u v}›)
proof (induct t)
  have ‹{(u, v). ?f [] u v} = {([], [])}› by (auto simp add: Nil-setinterleaves_ptick)
  thus ‹finite {(u, v). ?f [] u v}› by simp
next
  fix e t assume ‹finite {(u, v). ?f t u v}›
  have ∗ : ‹{(x # u, v) | u v. ?f t u v} = (λ(u, v). (x # u, v)) ‘ {(u, v). ?f t u v}›
    ‹{(u, y # v) | u v. ?f t u v} = (λ(u, v). (u, y # v)) ‘ {(u, v). ?f t u v}›
    ‹{(x # u, y # v) | u v. ?f t u v} = (λ(u, v). (x # u, y # v)) ‘ {(u, v). ?f t u
v}›
    for x y by auto
  show ‹finite {(u, v). ?f (e # t) u v}›
  proof (cases e)
    fix a assume ‹e = ev a›
    hence ‹?f (e # t) u v ⟹
          u ≠ [] ∧ hd u = ev a ∧ ?f t (tl u) v ∨
          v ≠ [] ∧ hd v = ev a ∧ ?f t u (tl v) ∨
          u ≠ [] ∧ hd u = ev a ∧ v ≠ [] ∧ hd v = ev a ∧ ?f t (tl u) (tl v)› for u v
    by (cases e) (auto elim: Cons-ev-setinterleaves_ptickE Cons-tick-setinterleaves_ptickE)
    hence ‹{(u, v). ?f (e # t) u v} ⊆
          {(ev a # u, v) | u v. ?f t u v} ∪
          {(u, ev a # v) | u v. ?f t u v} ∪
```

```
      {(ev a # u, ev a # v) | u v. ?f t u v}›
    by (simp add: subset-iff) (metis list.collapse)
  moreover have ‹finite {(ev a # u, v) | u v. ?f t u v}›
    by (simp add: ∗(1) ‹finite {(u, v). ?f t u v}›)
  moreover have ‹finite {(u, ev a # v) | u v. ?f t u v}›
    by (simp add: ∗(2) ‹finite {(u, v). ?f t u v}›)
  moreover have ‹finite {(ev a # u, ev a # v) | u v. ?f t u v}›
    by (simp add: ∗(3) ‹finite {(u, v). ?f t u v}›)
  ultimately show ‹finite {(u, v). ?f (e # t) u v}›
    by (simp add: finite-subset)
 next
  show ‹finite {(u, v). ?f (e # t) u v}› if ‹e = ✓(r-s)› for r-s
  proof (cases ‹r-s ∈ range-tick-join›)
    assume ‹r-s ∈ range-tick-join›
    then obtain r s where ‹r ⊗✓ s = ⌊r-s⌋› by blast
    hence ‹?f (e # t) u v ⟹
        u ≠ [] ∧ hd u = ✓(r) ∧ v ≠ [] ∧ hd v = ✓(s) ∧ ?f t (tl u) (tl v)› for u v
      by (cases u; cases v)
        (auto simp add: ‹e = ✓(r-s)› setinterleaving_ptick-simps inj-tick-join
          split: event_ptick.splits option.split-asm if-split-asm)
    hence ‹{(u, v). ?f (e # t) u v} ⊆ {(✓(r) # u, ✓(s) # v) |u v. ?f t u v}›
      by (simp add: subset-iff) (metis list.collapse)
    moreover have ‹finite {(✓(r) # u, ✓(s) # v) |u v. ?f t u v}›
      by (simp add: ∗(3) ‹finite {(u, v). ?f t u v}›)
    ultimately show ‹finite {(u, v). ?f (e # t) u v}›
      by (simp add: finite-subset)
  next
    assume ‹r-s ∉ range-tick-join›
    hence ‹¬ ?f (e # t) u v› for u v
      by (cases u; cases v)
        (auto simp add: ‹e = ✓(r-s)› setinterleaving_ptick-simps
          split: event_ptick.splits option.split-asm)
    thus ‹finite {(u, v). ?f (e # t) u v}› by simp
  qed
 qed
qed


lemma finite-setinterleaves_ptick-tick-join-Sync_ptick:
 ‹finite {(t-P, t-Q, u). u setinterleaves✓(⊗✓) ((t-P, t-Q), A) ∧
                    (∃ v. t = u @ v ∧ ftF v ∧ (tF u ∨ v = []))}›
 (is ‹finite {(t-P, t-Q, u). ?f u t-P t-Q ∧ ?g t u}›)
proof −
 have ‹{(t-P, t-Q, u) |t-P t-Q. ?f u t-P t-Q} ⊆
      (λ(t-P, t-Q). (t-P, t-Q, u)) ‘ {(t-P, t-Q). ?f u t-P t-Q}› for u by auto
 hence ‹finite {(t-P, t-Q, u) |t-P t-Q. ?f u t-P t-Q}› for u
   by (rule finite-subset) (simp add: finite-setinterleaves_ptick-tick-join)
 moreover have ‹{(t-P, t-Q, u). ?f u t-P t-Q ∧ ?g t u} ⊆
              (⋃ u ∈ {u. u ≤ t}. {(t-P, t-Q, u) |t-P t-Q. ?f u t-P t-Q})›
```

**unfolding** *less-eq-list-def prefix-def* **by** *blast*
**moreover have** ‹*finite* {*u*. *u* ≤ *t*}› **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* :: ‹$('a, 'r \times 's)$ $event_{ptick}$ ⇒ $('a, 's \times 'r)$ $event_{ptick}$›
  **where** ‹*tick-swap* $(ev\ a)$ = $ev\ a$›
  | ‹*tick-swap* ✓$((r, s))$ = ✓$((s, r))$›

**lemma** *tick-swap-tick* : ‹*tick-swap* ✓$(r\text{-}s)$ = (*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** ‹$e = ✓(r\text{-}s) \Longrightarrow$ *tick-swap* (*tick-swap e*) = *e*› **for** *r-s*
**by** (*cases r-s*) *simp-all*
**qed**

**lemma** *tick-swap-comp-tick-swap* [*simp*] : ‹*tick-swap* ∘ *tick-swap* = *id*›
**by** (*rule ext*) *simp*

**lemma** *inj-tick-swap* [*simp*] : ‹*inj tick-swap*›
**by** (*metis injI tick-swap-tick-swap*)

**lemma** *surj-tick-swap* [*simp*] : ‹*surj tick-swap*›
**by** (*metis surjI tick-swap-tick-swap*)

**lemma** *bij-tick-swap* [*simp*] : ‹*bij tick-swap*›
**by** (*simp add*: *bij-betw-def*)

**lemma** *bij-betw-tick-swap* :
‹*bij-betw tick-swap* (*range ev* ) (*range ev* )›
‹*bij-betw tick-swap* (*range tick*) (*range tick*)›
**by** (*auto simp add*: *bij-betw-def inj-on-def set-eq-iff image-iff*)

**lemma** *ev-eq-tick-swap-iff* [*simp*] : ‹*ev a* = *tick-swap e* ⟷ *e* = *ev a*›
**and** *tick-swap-eq-ev-iff* [*simp*] : ‹*tick-swap e* = *ev a* ⟷ *e* = *ev a*›
**and** *tick-eq-tick-swap-iff* [*simp*] : ‹✓((*r*, *s*)) = *tick-swap e* ⟷ *e* = ✓((*s*, *r*))›
**and** *tick-swap-eq-tick-iff* [*simp*] : ‹*tick-swap e* = ✓((*r*, *s*)) ⟷ *e* = ✓((*s*, *r*))›
**by** (*cases e*, *auto*)+

## Swapping a Trace

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

**lemma** *trace-tick-swap-tick-Cons* :
‹*trace-tick-swap* (✓(*r-s*) # *t*) = (*case r-s of* (*r*, *s*) ⇒ ✓((*s*, *r*)) # *trace-tick-swap t*)›
**by** (*cases r-s*) *simp*

**lemma** *trace-tick-swap-def* : ‹*trace-tick-swap* = *map tick-swap*›
**proof** (*rule ext*)
**show** ‹*trace-tick-swap t* = *map tick-swap t*› **for** *t* :: ‹('*a*, ('*r* × '*s*)) *trace*$_{ptick}$›
**by** (*induct t rule*: *trace-tick-swap.induct*) *simp-all*
**qed**

**lemma** *trace-tick-swap-append* : ‹*trace-tick-swap* (*t* @ *u*) = *trace-tick-swap t* @ *trace-tick-swap u*›
**by** (*simp add*: *trace-tick-swap-def*)

**lemma** *trace-tick-swap-singl* [*simp*] : ‹*trace-tick-swap* [*e*] = [*tick-swap e*]›
**by** (*cases e*) *auto*

**lemma** *trace-tick-swap-comp-trace-tick-swap* [*simp*] : ‹*trace-tick-swap* ∘ *trace-tick-swap* = *id*›
**by** (*simp add*: *trace-tick-swap-def*)

**lemma** *trace-tick-swap-trace-tick-swap* [*simp*] : ‹*trace-tick-swap* (*trace-tick-swap t*) = *t*›
**by** (*metis comp-def id-apply trace-tick-swap-comp-trace-tick-swap*)

**lemma** *inj-trace-tick-swap* [*simp*] : ‹*inj trace-tick-swap*›
**by** (*metis injI trace-tick-swap-trace-tick-swap*)

**lemma** *surj-trace-tick-swap* [*simp*] : ‹*surj trace-tick-swap*›
**by** (*metis surjI trace-tick-swap-trace-tick-swap*)

**lemma** *bij-trace-tick-swap* [*simp*] : ‹*bij trace-tick-swap*›
**by** (*simp add*: *bij-betw-def*)

**lemma** *strict-mono-trace-tick-swap* : ‹*strict-mono trace-tick-swap*›
**by** (*unfold trace-tick-swap-def*)
(*rule strict-mono-map*, *simp add*: *strict-monoI*)

**lemma** *image-trace-tick-swap-min-elems* :
‹*trace-tick-swap* ‘ (*min-elems T*) = *min-elems* (*trace-tick-swap* ‘ *T*)›
**proof** (*intro subset-antisym subsetI*)
**show** ‹*t* ∈ *trace-tick-swap* ‘ *min-elems T* ⟹ *t* ∈ *min-elems* (*trace-tick-swap* ‘ *T*)› **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** ‹*t* ∈ *min-elems* (*trace-tick-swap* ‘ *T*) ⟹ *t* ∈ *trace-tick-swap* ‘ *min-elems T*› **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*] : ‹[] = *trace-tick-swap t* ⟷ *t* = []›
**and** *trace-tick-swap-eq-Nil-iff* [*iff*] : ‹*trace-tick-swap t* = [] ⟷ *t* = []›
**by** (*metis trace-tick-swap.simps*(*1*) *trace-tick-swap-trace-tick-swap*)+

**lemma** *ev-Cons-eq-trace-tick-swap-iff* [*iff*] :
‹*ev a # t = trace-tick-swap u ⟷ u = ev a # trace-tick-swap t*›
**and** *trace-tick-swap-eq-ev-Cons-iff* [*iff*] :
‹*trace-tick-swap u = ev a # t ⟷ u = ev a # trace-tick-swap t*›
**by** (*metis trace-tick-swap.simps*(*2*) *trace-tick-swap-trace-tick-swap*)+

**lemma** *tick-Cons-eq-trace-tick-swap-iff* [*iff*] :
‹✓((*r*, *s*)) *# t = trace-tick-swap u ⟷ u = ✓*((*s*, *r*)) *# trace-tick-swap t*›
**and** *trace-tick-swap-eq-tick-Cons-iff* [*iff*] :
‹*trace-tick-swap u = ✓*((*r*, *s*)) *# t ⟷ u = ✓*((*s*, *r*)) *# trace-tick-swap t*›
**by** (*metis trace-tick-swap.simps*(*3*) *trace-tick-swap-trace-tick-swap*)+

**lemma** *snoc-ev-eq-trace-tick-swap-iff* [*iff*] :
‹*t @* [*ev a*] *= trace-tick-swap u ⟷ u = trace-tick-swap t @* [*ev a*]›
**and** *trace-tick-swap-eq-snoc-ev-iff* [*iff*] :
‹*trace-tick-swap u = t @* [*ev a*] *⟷ u = trace-tick-swap t @* [*ev a*]›
**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*] :
‹*t @* [*✓*((*r*, *s*))] *= trace-tick-swap u ⟷ u = trace-tick-swap t @* [*✓*((*s*, *r*))]›
**and** *trace-tick-swap-eq-snoc-tick-iff* [*iff*] :
‹*trace-tick-swap u = t @* [*✓*((*r*, *s*))] *⟷ u = trace-tick-swap t @* [*✓*((*s*, *r*))]›
**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* :
‹*trace-tick-swap u = ev a # t ⟹* (⋀*u'*. *u = ev a # u' ⟹ t = trace-tick-swap u' ⟹ thesis*) *⟹ thesis*›
**and** *trace-tick-swap-eq-tick-ConsE* :
‹*trace-tick-swap u = ✓*((*r*, *s*)) *# t ⟹* (⋀*u'*. *u = ✓*((*s*, *r*)) *# u' ⟹ t = trace-tick-swap u' ⟹ thesis*) *⟹ thesis*›
**and** *trace-tick-swap-eq-snoc-evE* :
‹*trace-tick-swap u = t @* [*ev a*] *⟹* (⋀*u'*. *u = u' @* [*ev a*] *⟹ t = trace-tick-swap u' ⟹ thesis*) *⟹ thesis*›
**and** *trace-tick-swap-eq-snoc-tickE* :
‹*trace-tick-swap u = t @* [*✓*((*r*, *s*))] *⟹* (⋀*u'*. *u = u' @* [*✓*((*s*, *r*))] *⟹ t = trace-tick-swap u' ⟹ thesis*) *⟹ thesis*›
**by** (*simp*, *metis trace-tick-swap-trace-tick-swap*)+

**lemma** *trace-tick-swap-tickFree* :
‹*tF t ⟹ trace-tick-swap t = map* (*ev ∘ of-ev*) *t*› **for** *t* :: ‹('*a*, ('*r* × '*s*)) *trace*$_{ptick}$›
**proof** (*induct t*)
**show** ‹*trace-tick-swap* [] *= map* (*ev ∘ of-ev*) []› **by** *simp*
**next**
**fix** *e* **and** *t* :: ‹('*a*, ('*r* × '*s*)) *trace*$_{ptick}$›
**assume** ‹*tF* (*e # t*)› **and** ‹*tF t ⟹ trace-tick-swap t = map* (*ev ∘ of-ev*) *t*›

**moreover from** ‹*tF* (*e* # *t*)› **obtain** *a* **where** ‹*e* = *ev a*› ‹*tF t*›
**by** (*meson is-ev-def tickFree-Cons-iff*)
**ultimately show** ‹*trace-tick-swap* (*e* # *t*) = *map* (*ev* ∘ *of-ev*) (*e* # *t*)› **by** *simp*
**qed**

**lemma** *trace-tick-swap-front-tickFree* :
‹*trace-tick-swap t* = ( *if tF t then map* (*ev* ∘ *of-ev*) *t*
*else map* (*ev* ∘ *of-ev*) (*butlast t*) @ [*case last t of* ✓((*r*, *s*)) ⇒ ✓((*s*, *r*))])›
**if** ‹*ftF t*› **for** *t* :: ‹('*a*, ('*r* × '*s*)) *trace*$_{ptick}$›
**proof** −
**show** *?thesis*
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*tF t* ⟹ *trace-tick-swap t* = *map* (*ev* ∘ *of-ev*) *t*›
**by** (*simp add*: *trace-tick-swap-tickFree*)
**next**
**assume** ‹¬ *tF t*›
**with** ‹*ftF t*› **obtain** *t'* *r* *s* **where** ‹*t* = *t'* @ [✓((*r*, *s*))]› ‹*tF t'*›
**by** (*metis front-tickFree-append-iff nonTickFree-n-frontTickFree not-Cons-self2 surj-pair*)
**hence** ‹*trace-tick-swap t* = *trace-tick-swap t'* @ [✓((*s*, *r*))]›
**by** (*metis trace-tick-swap-append trace-tick-swap.simps*(*1*, *3*))
**also from** ‹*tF t'*› ‹*t* = *t'* @ [✓((*r*, *s*))]›
**have** ‹*trace-tick-swap t'* = *map* (*ev* ∘ *of-ev*) (*butlast t*)› **by** (*simp add*: *trace-tick-swap-tickFree*)
**also from** ‹*t* = *t'* @ [✓((*r*, *s*))]›
**have** ‹[✓((*s*, *r*))] = [*case last t of* ✓((*r*, *s*)) ⇒ ✓((*s*, *r*))]› **by** *simp*
**finally show** ‹*trace-tick-swap t* = *map* (*ev* ∘ *of-ev*) (*butlast t*) @
[*case last t of* ✓((*r*, *s*)) ⇒ ✓((*s*, *r*))]› .
**qed**
**qed**

**lemma** *tickFree-trace-tick-swap-iff* [*simp*] : ‹*tF* (*trace-tick-swap t*) ⟷ *tF t*›
**by** (*metis tickFree-map-ev-comp trace-tick-swap-tickFree trace-tick-swap-trace-tick-swap*)

**lemma** *front-tickFree-trace-tick-swap-iff* [*simp*] : ‹*ftF* (*trace-tick-swap t*) ⟷ *ftF t*›
**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* :: ‹('*a*, ('*r* × '*s*)) *refusal*$_{ptick}$ ⇒ ('*a*, ('*s* × '*r*)) *refusal*$_{ptick}$›
**where** ‹*refusal-tick-swap X* = *tick-swap* ‘ *X*›

**lemma** *refusal-tick-swap-empty* [*simp*] : ‹*refusal-tick-swap* {} = {}›
**by** (*simp add*: *refusal-tick-swap-def*)

**lemma** *refusal-tick-swap-insert* [*simp*] :
‹*refusal-tick-swap* (*insert x X*) = *insert* (*tick-swap x*) (*refusal-tick-swap X*)›
**by** (*simp add*: *refusal-tick-swap-def*)

**lemma** *refusal-tick-swap-union* :
‹*refusal-tick-swap* (*X* ∪ *Y*) = *refusal-tick-swap X* ∪ *refusal-tick-swap Y*›
**by** (*simp add*: *refusal-tick-swap-def image-Un*)

**lemma** *refusal-tick-swap-diff* :
‹*refusal-tick-swap* (*X* − *Y*) = *refusal-tick-swap X* − *refusal-tick-swap Y*›
**by** (*simp add*: *refusal-tick-swap-def image-set-diff*)

**lemma** *refusal-tick-swap-inter* :
‹*refusal-tick-swap* (*X* ∩ *Y*) = *refusal-tick-swap X* ∩ *refusal-tick-swap Y*›
**by** (*simp add*: *refusal-tick-swap-def image-Int*)

**lemma** *refusal-tick-swap-singl* : ‹*refusal-tick-swap* {*e*} = {*tick-swap e*}› **by** *simp*

**lemma** *refusal-tick-swap-comp-refusal-tick-swap* [*simp*] :
‹*refusal-tick-swap* ∘ *refusal-tick-swap* = *id*›
**by** (*auto simp add*: *refusal-tick-swap-def image-iff*)

**lemma** *refusal-tick-swap-refusal-tick-swap* [*simp*] :
‹*refusal-tick-swap* (*refusal-tick-swap X*) = *X*›
**by** (*simp add*: *comp-eq-dest-lhs*)

**lemma** *inj-refusal-tick-swap* [*simp*] : ‹*inj refusal-tick-swap*›
**by** (*metis injI refusal-tick-swap-refusal-tick-swap*)

**lemma** *surj-refusal-tick-swap* [*simp*] : ‹*surj refusal-tick-swap*›
**by** (*metis surjI refusal-tick-swap-refusal-tick-swap*)

**lemma** *bij-refusal-tick-swap* [*simp*] : ‹*bij refusal-tick-swap*›
**by** (*simp add*: *bij-betw-def*)

**lemma** *strict-mono-refusal-tick-swap* : ‹*strict-mono refusal-tick-swap*›
**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*] : ‹{} = *refusal-tick-swap X* ⟷ *X* = {}›
**and** *refusal-tick-swap-eq-empty-iff* [*iff*] : ‹*refusal-tick-swap X* = {} ⟷ *X* = {}›
**by** (*simp-all add*: *refusal-tick-swap-def*)

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

**and** *refusal-tick-swap-eq-insert-ev-iff* [*iff*] :
‹*refusal-tick-swap Y* =*insert* (*ev a*) *X* ⟷ *Y* = *insert* (*ev a*) (*refusal-tick-swap X*)›
**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*] :
‹*insert* ✓((*r*, *s*)) *X* = *refusal-tick-swap Y* ⟷ *Y* = *insert* ✓((*s*, *r*)) (*refusal-tick-swap X*)›
**and** *refusal-tick-swap-eq-insert-tick-iff* [*iff*] :
‹*refusal-tick-swap Y* = *insert* ✓((*r*, *s*)) *X* ⟷ *Y* = *insert* ✓((*s*, *r*)) (*refusal-tick-swap X*)›
**by** (*metis refusal-tick-swap-insert refusal-tick-swap-refusal-tick-swap tick-swap.simps*(*2*))+

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

**lemma** *refusal-tick-swap-tickFree* :
‹*X* ⊆ *range ev* ⟹ *refusal-tick-swap X* = (*ev* ∘ *of-ev*) ‘ *X*›
**by** (*force simp add*: *refusal-tick-swap-def*)

**lemma** *tickFree-refusal-tick-swap-iff* :
‹*refusal-tick-swap X* ⊆ *range ev* ⟷ *X* ⊆ *range ev*›
**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* :
‹*t setinterleaves* ((*u*, *v*), *S*) ⟹
*trace-tick-swap t setinterleaves* ((*trace-tick-swap u*, *trace-tick-swap v*), *refusal-tick-swap S*)›
**proof** (*induct* ‹(*u*, *S*, *v*)› *arbitrary*: *t u v rule*: *setinterleaving.induct*)
**case** *1* **thus** *?case* **by** *simp*
**next**
**case** (*2 y v*)
**from** *2.prems* **obtain** *t'* **where** ‹*y* ∉ *S*› ‹*t* = *y* # *t'*› ‹*t' setinterleaves* (([], *v*), *S*)›
**by** (*auto split*: *if-split-asm*)
**from** *2.hyps*[*OF* ‹*y* ∉ *S*› ‹*t' setinterleaves* (([], *v*), *S*)›]
**have** ‹*trace-tick-swap t' setinterleaves* (([], *trace-tick-swap v*), *refusal-tick-swap S*)› **by** *simp*
**with** ‹*y* ∉ *S*› **show** *?case* **by** (*cases y*) (*auto simp add*: ‹*t* = *y* # *t'*› *refusal-tick-swap-def split*: *prod.split*)
**next**

**case** (*3 x u*)
**from** *3.prems* **obtain** $t'$ **where** ‹$x \notin S$› ‹$t = x \# t'$› ‹$t'$ *setinterleaves* (($u$, []), $S$)›
**by** (*auto split*: *if-split-asm*)
**from** *3.hyps*[*OF* ‹$x \notin S$› ‹$t'$ *setinterleaves* (($u$, []), $S$)›]
**have** ‹*trace-tick-swap* $t'$ *setinterleaves* ((*trace-tick-swap* $u$, []), *refusal-tick-swap* $S$)› **by** *simp*
**with** ‹$x \notin S$› **show** *?case* **by** (*cases x*) (*auto simp add*: ‹$t = x \# t'$› *refusal-tick-swap-def split*: *prod.split*)
**next**
**case** (*4 x u y v*)
**from** *4.prems*
**consider** (*both-in*) $t'$ **where** ‹$x \in S$› ‹$y \in S$› ‹$x = y$› ‹$t = x \# t'$› ‹$t'$ *setinterleaves* (($u$, $v$), $S$)›
| (*inR-mvL*) $t'$ **where** ‹$x \notin S$› ‹$y \in S$› ‹$t = x \# t'$› ‹$t'$ *setinterleaves* (($u$, $y \# v$), $S$)›
| (*inL-mvR*) $t'$ **where** ‹$x \in S$› ‹$y \notin S$› ‹$t = y \# t'$› ‹$t'$ *setinterleaves* (($x \# u$, $v$), $S$)›
| (*notin-mvL*) $t'$ **where** ‹$x \notin S$› ‹$y \notin S$› ‹$t = x \# t'$› ‹$t'$ *setinterleaves* (($u$, $y \# v$), $S$)›
| (*notin-mvR*) $t'$ **where** ‹$x \notin S$› ‹$y \notin S$› ‹$t = y \# t'$› ‹$t'$ *setinterleaves* (($x \# u$, $v$), $S$)›
**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** ‹¬ $y \notin S$› **by** (*simp add*: *inR-mvL*(*2*))
**from** *4.hyps*(*5*)[*OF inR-mvL*(*1*) ‹¬ $y \notin S$› *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** $*$ : ‹*a setinterleaves* (($t$, $u$), *tick-swap* ‘ $S$) ⟹ $h \notin$ *tick-swap* ‘ $S$ ⟹
(*h* # *a*) *setinterleaves* (($t$, $h \# u$), *tick-swap* ‘ $S$)› **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* :
‹*trace-tick-swap t setinterleaves* ((*u, v*), *S*) ⟷
*t setinterleaves* ((*trace-tick-swap u, trace-tick-swap v*), *refusal-tick-swap S*)›
**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* :: ‹('*a*, ('*r* × '*s*)) *failure*$_{ptick}$ ⇒ ('*a*, ('*s* × '*r*)) *failure*$_{ptick}$›
**where** ‹*failure-tick-swap F* ≡ *case F of* (*t, X*) ⇒ (*trace-tick-swap t, refusal-tick-swap X*)›

**lemma** *failure-tick-swap-empty* [*simp*] : ‹*failure-tick-swap* ([], {}) = ([], {})›
**by** (*simp add*: *failure-tick-swap-def*)

**lemma** *failure-tick-swap-comp-failure-tick-swap* [*simp*] :
‹*failure-tick-swap* ∘ *failure-tick-swap* = *id*›
**by** (*auto simp add*: *failure-tick-swap-def*)

**lemma** *failure-tick-swap-failure-tick-swap* [*simp*] :
‹*failure-tick-swap* (*failure-tick-swap F*) = *F*›
**by** (*simp add*: *comp-eq-dest-lhs*)

**lemma** *inj-failure-tick-swap* [*simp*] : ‹*inj failure-tick-swap*›
**by** (*metis injI failure-tick-swap-failure-tick-swap*)

**lemma** *surj-failure-tick-swap* [*simp*] : ‹*surj failure-tick-swap*›
**by** (*metis surjI failure-tick-swap-failure-tick-swap*)

**lemma** *bij-failure-tick-swap* [*simp*] : ‹*bij failure-tick-swap*›
**by** (*simp add*: *bij-betw-def*)

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

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

### 5.1.2 The Operator

#### Definition

**lift-definition** *TickSwap* :: ‹('*a*, '*r* × '*s*) *process*$_{ptick}$ ⇒ ('*a*, '*s* × '*r*) *process*$_{ptick}$›
**is** ‹λ*P*. ({(*t*, *X*). *failure-tick-swap* (*t*, *X*) ∈ ℱ *P*}, {*t*. *trace-tick-swap t* ∈ 𝒟 *P*})›
— One might expect λ*P*. (*failure-tick-swap* ‘ ℱ *P*, *trace-tick-swap* ‘ 𝒟 *P*) instead. This is equivalent, see the projections below, but easier for the following proof obligation.
**proof** −
**show** ‹*?thesis P*› (**is** ‹*is-process* (*?f*, *?d*)›) **for** *P*
**proof** (*unfold is-process-def FAILURES-def DIVERGENCES-def fst-conv snd-conv*, *intro conjI impI allI*)
**show** ‹([], {}) ∈ *?f*› **by** (*simp add*: *is-processT1*)
**next**
**show** ‹(*t*, *X*) ∈ *?f* ⟹ *ftF t*› **for** *t X*
**by** (*simp add*: *failure-tick-swap-def*)
(*use is-processT2 front-tickFree-trace-tick-swap-iff* **in** *blast*)
**next**
**show** ‹(*t* @ *u*, {}) ∈ *?f* ⟹ (*t*, {}) ∈ *?f*› **for** *t u*
**by** (*simp add*: *failure-tick-swap-def*) (*metis trace-tick-swap-append is-processT3*)
**next**
**show** ‹(*t*, *Y*) ∈ *?f* ∧ *X* ⊆ *Y* ⟹ (*t*, *X*) ∈ *?f*› **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** ‹(*t*, *X*) ∈ *?f* ∧ (∀ *e*. *e* ∈ *Y* ⟶ (*t* @ [*e*], {}) ∉ *?f*)›
**hence** ‹(*trace-tick-swap t*, *refusal-tick-swap X*) ∈ ℱ *P* ∧
(∀ *e*. *e* ∈ *refusal-tick-swap Y* ⟶ (*trace-tick-swap t* @ [*e*], {}) ∉ ℱ *P*)›
**by** (*auto simp add*: *failure-tick-swap-def refusal-tick-swap-def trace-tick-swap-append*)
**thus** ‹(*t*, *X* ∪ *Y*) ∈ *?f*›
**by** (*simp add*: *failure-tick-swap-def is-processT5 refusal-tick-swap-union*)
**next**
**show** ‹(*t* @ [✓(*s-r*)], {}) ∈ *?f* ⟹ (*t*, *X* − {✓(*s-r*)}) ∈ *?f*› **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** ‹*t* ∈ *?d* ∧ *tF t* ∧ *ftF u* ⟹ *t* @ *u* ∈ *?d*› **for** *t u*
**by** (*simp add*: *trace-tick-swap-append is-processT7*)
**next**
**show** ‹*t* ∈ *?d* ⟹ (*t*, *X*) ∈ *?f*› **for** *t X*
**by** (*simp add*: *failure-tick-swap-def is-processT8*)
**next**
**show** ‹*t* @ [✓(*s-r*)] ∈ *?d* ⟹ *t* ∈ *?d*› **for** *t s-r*
**by** (*cases s-r*) (*auto simp add*: *trace-tick-swap-append intro*: *is-processT9*)
**qed**
**qed**

### Projections

**lemma** *F-TickSwap′* : ‹$\mathcal{F}$ (*TickSwap P*) = {(*t*, *X*). *failure-tick-swap* (*t*, *X*) ∈ $\mathcal{F}$ *P*}›
**by** (*simp add*: *Failures.rep-eq TickSwap.rep-eq FAILURES-def*)

**lemma** *D-TickSwap′* : ‹$\mathcal{D}$ (*TickSwap P*) = {*t*. *trace-tick-swap t* ∈ $\mathcal{D}$ *P*}›
**by** (*simp add*: *Divergences.rep-eq TickSwap.rep-eq DIVERGENCES-def*)

**lemma** *T-TickSwap′* : ‹$\mathcal{T}$ (*TickSwap P*) = {*t*. *trace-tick-swap t* ∈ $\mathcal{T}$ *P*}›
**by** (*simp add*: *set-eq-iff F-TickSwap′ failure-tick-swap-def flip*: *T-F-spec*)

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

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

**lemma** *F-TickSwap* : ‹$\mathcal{F}$ (*TickSwap P*) = *failure-tick-swap* ‘ $\mathcal{F}$ *P*›
**by** (*simp add*: *set-eq-iff F-TickSwap′*)
(*metis* (*no-types*, *lifting*) *failure-tick-swap-failure-tick-swap imageE image-eqI*)

**lemma** *D-TickSwap* : ‹$\mathcal{D}$ (*TickSwap P*) = *trace-tick-swap* ‘ $\mathcal{D}$ *P*›
**by** (*simp add*: *set-eq-iff D-TickSwap′*)
(*metis* (*no-types*, *lifting*) *trace-tick-swap-trace-tick-swap imageE image-eqI*)

**lemma** *T-TickSwap* : ‹$\mathcal{T}$ (*TickSwap P*) = *trace-tick-swap* ‘ $\mathcal{T}$ *P*›
**by** (*simp add*: *set-eq-iff T-TickSwap′*)
(*metis* (*no-types*, *lifting*) *trace-tick-swap-trace-tick-swap imageE image-eqI*)

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

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

**lemma** *F-TickSwap″* : ‹$\mathcal{F}$ (*TickSwap P*) = {(*trace-tick-swap t*, *refusal-tick-swap X*)| *t X*. (*t*, *X*) ∈ $\mathcal{F}$ *P*}›
**by** (*auto simp add*: *F-TickSwap failure-tick-swap-def*)

**lemma** *D-TickSwap″* : ‹$\mathcal{D}$ (*TickSwap P*) = {*trace-tick-swap t*| *t*. *t* ∈ $\mathcal{D}$ *P*}›
**by** (*auto simp add*: *D-TickSwap*)

**lemma** *T-TickSwap″* : ‹$\mathcal{T}$ (*TickSwap P*) = {*trace-tick-swap t*| *t*. *t* ∈ $\mathcal{T}$ *P*}›
**by** (*auto simp add*: *T-TickSwap*)

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

### Properties

**lemma** *events-TickSwap* [*simp*] : ‹*events-of* (*TickSwap P*) = *events-of P*›
**by** (*auto simp add*: *events-of-def T-TickSwap trace-tick-swap-def*)

**lemma** *ticks-TickSwap* [*simp*] : ‹*ticks-of* (*TickSwap P*) = {(*s*, *r*). (*r*, *s*) ∈ *ticks-of P*}›
**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*] :
‹*strict-ticks-of* (*TickSwap P*) = {(*s*, *r*). (*r*, *s*) ∈ *strict-ticks-of P*}›
**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*$_{Pair}$ :
‹*trace-tick-swap* ‘ *setinterleaving*$_{ptick}$ (λ*r s*. ⌊(*r*, *s*)⌋, *u*, *A*, *v*) =
*setinterleaving*$_{ptick}$ (λ*r s*. ⌊(*r*, *s*)⌋, *v*, *A*, *u*)›
**for** *u* :: ‹('*a*, '*r*) *trace*$_{ptick}$› **and** *v* :: ‹('*a*, '*s*) *trace*$_{ptick}$›
**by** (*rule sym*, *induct* ‹(λ*r* :: '*r*. λ*s* :: '*s*. ⌊(*r*, *s*)⌋, *u*, *A*, *v*)›
*arbitrary*: *u v*) (*simp-all*, *safe*, *auto*)

**lemma** *trace-tick-swap-setinterleaves*$_{Pair}$*-iff* [*iff*] :
‹*trace-tick-swap t setinterleaves*$\checkmark_{\lambda r\ s.\ \lfloor(r,\ s)\rfloor}$ ((*u*, *v*), *A*) ⟷
*t setinterleaves*$\checkmark_{\lambda r\ s.\ \lfloor(r,\ s)\rfloor}$ ((*v*, *u*), *A*)›
**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*$_{ptick}$ : ‹*tick-swap* = *map-event*$_{ptick}$ *id prod.swap*›
**proof** (*rule ext*)
**show** ‹*tick-swap e* = *map-event*$_{ptick}$ *id prod.swap e*› **for** *e* :: ‹('*a*, '*r* × '*s*) *event*$_{ptick}$›
**by** (*cases e*) (*auto split*: *event*$_{ptick}$*.splits prod.splits*)
**qed**

**lemma** *trace-tick-swap-is-map-map-event*$_{ptick}$ :
‹*trace-tick-swap* = *map* (*map-event*$_{ptick}$ *id prod.swap*)›
**by** (*simp add*: *tick-swap-is-map-event*$_{ptick}$ *trace-tick-swap-def*)

**lemma** *refusal-tick-swap-is-image-map-event*$_{ptick}$ :
‹*refusal-tick-swap* = (‘) (*map-event*$_{ptick}$ *id prod.swap*)›
**by** (*rule ext*) (*simp add*: *refusal-tick-swap-def tick-swap-is-map-event*$_{ptick}$)

**theorem** *TickSwap-is-Renaming* :
‹*TickSwap P* = *Renaming P id prod.swap*› (**is** ‹*?lhs* = *?rhs*›)
**proof** (*subst Process-eq-spec-optimized*, *safe*)
**fix** *t* **assume** ‹*t* ∈ *𝒟 ?lhs*›
**with** *D-imp-front-tickFree* **have** ‹*ftF t*› **by** *blast*
**define** *t1* **where** ‹*t1* ≡ *trace-tick-swap* (*if tF t then t else butlast t*)›
**define** *t2* **where** ‹*t2* ≡ *if tF t then* [] *else* [*last t*]›
**have** ‹*t* = *map* (*map-event*$_{ptick}$ *id prod.swap*) *t1* @ *t2*›
**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** ‹*ftF t*› *front-tickFree-iff-tickFree-butlast t1-def* **have** ‹*tF t1*› **by**

*auto*
**moreover have** ‹*ftF t2*› **by** (*simp add*: *t2-def*)
**moreover from** *t1-def D-TickSwap′* ‹*ftF t*› ‹*t* ∈ $\mathcal{D}$ *?lhs*›
*div-butlast-when-non-tickFree-iff* **have** ‹*t1* ∈ $\mathcal{D}$ *P*› **by** *blast*
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*› **unfolding** *D-Renaming* **by** *blast*
**next**
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**then obtain** *t1 t2*
**where** ‹*t* = *map* (*map-event*$_{ptick}$ *id prod.swap*) *t1* @ *t2*› ‹*tF t1*› ‹*ftF t2*› ‹*t1* ∈ $\mathcal{D}$ *P*›
**unfolding** *D-Renaming* **by** *blast*
**thus** ‹*t* ∈ $\mathcal{D}$ *?lhs*› **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** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**then obtain** *t′ X′* **where** ‹*t* = *trace-tick-swap t′*› ‹*X* = *refusal-tick-swap X′*› ‹(*t′*, *X′*) ∈ $\mathcal{F}$ *P*›
**unfolding** *F-TickSwap failure-tick-swap-def* **by** *auto*
**moreover have** ‹*map-event*$_{ptick}$ *id prod.swap* −‘ *refusal-tick-swap X′* = *X′*›
**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** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**by** (*auto simp add*: *F-Renaming simp flip*: *trace-tick-swap-is-map-map-event*$_{ptick}$)
**next**
**fix** *t X* **assume** *same-div* : ‹$\mathcal{D}$ *?lhs* = $\mathcal{D}$ *?rhs*›
**assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**then consider** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
| *t′* **where** ‹*t* = *map* (*map-event*$_{ptick}$ *id prod.swap*) *t′*› ‹(*t′*, *map-event*$_{ptick}$ *id prod.swap* −‘ *X*) ∈ $\mathcal{F}$ *P*›
**unfolding** *Renaming-projs* **by** *blast*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**from** *same-div D-F* **show** ‹*t* ∈ $\mathcal{D}$ *?rhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** *blast*
**next**
**fix** *t′* **assume** ∗ : ‹*t* = *map* (*map-event*$_{ptick}$ *id prod.swap*) *t′*›
‹(*t′*, *map-event*$_{ptick}$ *id prod.swap* −‘ *X*) ∈ $\mathcal{F}$ *P*›
**from** ∗(*1*) **have** ‹(*t*, *X*) = *failure-tick-swap* (*t′*, *map-event*$_{ptick}$ *id prod.swap* −‘ *X*)›
**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** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** (*simp add*: *F-TickSwap*)
**qed**
**qed**

**lemma** *TickSwap-TickSwap* [*simp*] : ‹*TickSwap* (*TickSwap P*) = *P*›
**by** (*simp add*: *Process-eq-spec TickSwap-projs′*)

**lemma** *TickSwap-comp-TickSwap* [*simp*] : ‹*TickSwap* ∘ *TickSwap* = *id*›
**by** (*rule ext*) *simp*

**lemma** *TickSwap-eq-iff-eq-TickSwap* : ‹*TickSwap P* = *Q* ⟷ *P* = *TickSwap Q*›
**by** *auto*

**lemma** *inj-TickSwap* [*simp*] : ‹*inj TickSwap*›
**by** (*metis injI TickSwap-TickSwap*)

**lemma** *surj-TickSwap* [*simp*] : ‹*surj TickSwap*›
**by** (*metis surjI TickSwap-TickSwap*)

**lemma** *bij-TickSwap* [*simp*] : ‹*bij TickSwap*›
**by** (*simp add*: *bij-betw-def*)

**lemma** *strict-mono-TickSwap* : ‹*strict-mono TickSwap*›
**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* : ‹*P* ⊑ *Q* ⟹ *TickSwap P* ⊑ *TickSwap Q*›
**by** (*simp add*: *TickSwap-is-Renaming mono-Renaming*)

**lemma** *mono-TickSwap-FD* : ‹*P* ⊑$_{FD}$ *Q* ⟹ *TickSwap P* ⊑$_{FD}$ *TickSwap Q*›
**and** *mono-TickSwap-DT* : ‹*P* ⊑$_{DT}$ *Q* ⟹ *TickSwap P* ⊑$_{DT}$ *TickSwap Q*›
**and** *mono-TickSwap-F* : ‹*P* ⊑$_{F}$ *Q* ⟹ *TickSwap P* ⊑$_{F}$ *TickSwap Q*›
**and** *mono-TickSwap-D* : ‹*P* ⊑$_{D}$ *Q* ⟹ *TickSwap P* ⊑$_{D}$ *TickSwap Q*›
**and** *mono-TickSwap-T* : ‹*P* ⊑$_{T}$ *Q* ⟹ *TickSwap P* ⊑$_{T}$ *TickSwap Q*›
**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* : ‹*TickSwap P* ⊑ *TickSwap Q* ⟷ *P* ⊑ *Q*›
**and** *FD-TickSwap-iff* : ‹*TickSwap P* ⊑$_{FD}$ *TickSwap Q* ⟷ *P* ⊑$_{FD}$ *Q*›
**and** *DT-TickSwap-iff* : ‹*TickSwap P* ⊑$_{DT}$ *TickSwap Q* ⟷ *P* ⊑$_{DT}$ *Q*›
**and** *F-TickSwap-iff* : ‹*TickSwap P* ⊑$_{F}$ *TickSwap Q* ⟷ *P* ⊑$_{F}$ *Q*›
**and** *D-TickSwap-iff* : ‹*TickSwap P* ⊑$_{D}$ *TickSwap Q* ⟷ *P* ⊑$_{D}$ *Q*›
**and** *T-TickSwap-iff* : ‹*TickSwap P* ⊑$_{T}$ *TickSwap Q* ⟷ *P* ⊑$_{T}$ *Q*›
**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*] : ‹*cont P* ⟹ *cont* (λ*x*. *TickSwap* (*P x*))›
**by** (*simp add*: *TickSwap-is-Renaming*)

### Algebraic Laws

**Constant Processes** **lemma** *TickSwap-STOP* [*simp*] : ‹*TickSwap STOP* = *STOP*›
**by** (*simp add*: *STOP-iff-T T-TickSwap T-STOP*)

**lemma** *TickSwap-is-STOP-iff* [*simp*] : ‹*TickSwap P* = *STOP* ⟷ *P* = *STOP*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-BOT* [*simp*] : ‹*TickSwap* ⊥ = ⊥›
**by** (*simp add*: *BOT-iff-Nil-D D-TickSwap D-BOT*)

**lemma** *TickSwap-is-BOT-iff* [*simp*] : ‹*TickSwap P* = ⊥ ⟷ *P* = ⊥›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-SKIP* [*simp*] : ‹*TickSwap* (*SKIP* (*r*, *s*)) = *SKIP* (*s*, *r*)›
**by** (*simp add*: *TickSwap-is-Renaming*)

**lemma** *TickSwap-is-SKIP-iff* [*simp*] : ‹*TickSwap P* = *SKIP* (*r*, *s*) ⟷ *P* = *SKIP* (*s*, *r*)›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-SKIPS* [*simp*] : ‹*TickSwap* (*SKIPS R-S*) = *SKIPS* {(*s*, *r*). (*r*, *s*) ∈ *R-S*}›
**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*] :
‹*TickSwap P* = *SKIPS R-S* ⟷ *P* = *SKIPS* {(*s*, *r*). (*r*, *s*) ∈ *R-S*}›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**Binary (or less) Operators** **lemma** *TickSwap-Ndet* [*simp*] : ‹*TickSwap* (*P* ⊓ *Q*) = *TickSwap P* ⊓ *TickSwap Q*›
**by** (*simp add*: *Process-eq-spec TickSwap-projs Ndet-projs image-Un*)

**lemma** *TickSwap-is-Ndet-iff* [*simp*] : ‹*TickSwap P* = *Q* ⊓ *R* ⟷ *P* = *TickSwap Q* ⊓ *TickSwap R*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Det* [*simp*] :
‹*TickSwap* (*P* □ *Q*) = *TickSwap P* □ *TickSwap Q*›
**by** (*simp add*: *TickSwap-is-Renaming Renaming-Det*)

**lemma** *TickSwap-is-Det-iff* [*simp*] : ‹*TickSwap P* = *Q* □ *R* ⟷ *P* = *TickSwap Q* □ *TickSwap R*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Sliding* [*simp*] : ‹*TickSwap* (*P* ▷ *Q*) = *TickSwap P* ▷ *TickSwap Q*›
**by** (*simp add*: *Sliding-def*)

**lemma** *TickSwap-is-Sliding-iff* [*simp*] : ‹*TickSwap P* = *Q* ▷ *R* ⟷ *P* = *TickSwap Q* ▷ *TickSwap R*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Sync* [*simp*] :
‹*TickSwap* (*P* ⟦*S*⟧ *Q*) = *TickSwap P* ⟦*S*⟧ *TickSwap Q*›
**by** (*simp add*: *TickSwap-is-Renaming bij-Renaming-Sync*)

**lemma** *TickSwap-is-Sync-iff* [*simp*] :
‹*TickSwap P* = *Q* ⟦*S*⟧ *R* ⟷ *P* = *TickSwap Q* ⟦*S*⟧ *TickSwap R*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Seq* [*simp*] :
‹*TickSwap* (*P* ; *Q*) = *TickSwap P* ; *TickSwap Q*›
**by** (*simp add*: *Renaming-Seq TickSwap-is-Renaming*)

**lemma** *TickSwap-is-Seq-iff* [*simp*] :
‹*TickSwap P* = *Q* ; *R* ⟷ *P* = *TickSwap Q* ; *TickSwap R*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Renaming* [*simp*] :
‹*TickSwap* (*Renaming P f g*) =
*Renaming* (*TickSwap P*) *f* (*prod.swap* ∘ *g* ∘ *prod.swap*)›
**by** (*simp add*: *TickSwap-is-Renaming flip*: *Renaming-comp*)
(*metis comp-apply swap-swap*)

**lemma** *TickSwap-Renaming′* :
‹*TickSwap* (*Renaming P f g*) = *Renaming P f* (*prod.swap* ∘ *g*)›
**by** (*simp add*: *TickSwap-is-Renaming flip*: *Renaming-comp*)

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

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

**lemma** *TickSwap-Hiding* [*simp*] : ‹*TickSwap* (*P* \ *S*) = *TickSwap P* \ *S*›
**by** (*simp add*: *TickSwap-is-Renaming bij-Renaming-Hiding*)

**lemma** *TickSwap-is-Hiding-iff* [*simp*] : ‹*TickSwap P* = *Q* \ *S* ⟷ *P* = *TickSwap Q* \ *S*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Interrupt* [*simp*] :
‹*TickSwap* (*P* △ *Q*) = *TickSwap P* △ *TickSwap Q*›
**by** (*simp add*: *TickSwap-is-Renaming Renaming-Interrupt*)

**lemma** *TickSwap-is-Interrupt-iff* [*simp*] :
‹*TickSwap P* = *Q* △ *R* ⟷ *P* = *TickSwap Q* △ *TickSwap R*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

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

**Architectural Operators** **lemma** *TickSwap-GlobalNdet* [*simp*] :
‹*TickSwap* (⊓*a* ∈ *A*. *P a*) = ⊓*a* ∈ *A*. *TickSwap* (*P a*)›
**by** (*simp add*: *TickSwap-is-Renaming Renaming-distrib-GlobalNdet*)

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

**lemma** *TickSwap-GlobalDet* [*simp*] :
‹*TickSwap* (□*a* ∈ *A*. *P a*) = □*a* ∈ *A*. *TickSwap* (*P a*)›
**by** (*simp add*: *TickSwap-is-Renaming Renaming-distrib-GlobalDet*)

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

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

‹*TickSwap* (⟦*S*⟧ *m* ∈# *M*. *P m*) = ⟦*S*⟧ *m* ∈# *M*. *TickSwap* (*P m*)›
**by** (*induct M rule*: *induct-subset-mset-empty-single*) *simp-all*

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

**lemma** *TickSwap-MultiSeq* [*simp*] :
‹*L* ≠ [] ⟹ *TickSwap* (*SEQ l* ∈@ *L*. *P l*) = *SEQ l* ∈@ *L*. *TickSwap* (*P l*)›
**by** (*induct L rule*: *rev-induct*, *simp-all*)
(*metis MultiSeq-Nil SKIP-Seq TickSwap-Seq*)

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

**Communications** **lemma** *TickSwap-write0* [*simp*] : ‹*TickSwap* (*e* → *P*) = *e* → *TickSwap P*›
**by** (*simp add*: *TickSwap-is-Renaming Renaming-write0*)

**lemma** *TickSwap-is-write0-iff* [*simp*] : ‹*TickSwap P* = *e* → *Q* ⟷ *P* = *e* → *TickSwap Q*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-write* [*simp*] : ‹*TickSwap* (*c*!*e* → *P*) = *c*!*e* → *TickSwap P*›
**by** (*simp add*: *TickSwap-is-Renaming Renaming-write*)

**lemma** *TickSwap-is-write-iff* [*simp*] : ‹*TickSwap P* = *c*!*e* → *Q* ⟷ *P* = *c*!*e* → *TickSwap Q*›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Mprefix* [*simp*] :
‹*TickSwap* (□*a* ∈ *A* → *P a*) = □*a* ∈ *A* → *TickSwap* (*P a*)›
**by** (*simp add*: *Mprefix-GlobalDet*)

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

**lemma** *TickSwap-read* [*simp*] : ‹*TickSwap* (*c*?*a*∈*A* → *P a*) = *c*?*a*∈*A* → *TickSwap* (*P a*)›
**by** (*simp add*: *read-def comp-def*)

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

‹*TickSwap P* = *c*?*a*∈*A* → *Q a* ⟷ *P* = *c*?*a*∈*A* → *TickSwap* (*Q a*)›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**lemma** *TickSwap-Mndetprefix* [*simp*] :
‹*TickSwap* (⊓*a* ∈ *A* → *P a*) = ⊓*a* ∈ *A* → *TickSwap* (*P a*)›
**by** (*simp add*: *Mndetprefix-GlobalNdet*)

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

**lemma** *TickSwap-ndet-write* [*simp*] : ‹*TickSwap* (*c*!!*a*∈*A* → *P a*) = *c*!!*a*∈*A* → *TickSwap* (*P a*)›
**by** (*simp add*: *ndet-write-def comp-def*)

**lemma** *TickSwap-is-ndet-write-iff* [*simp*] :
‹*TickSwap P* = *c*!!*a*∈*A* → *Q a* ⟷ *P* = *c*!!*a*∈*A* → *TickSwap* (*Q a*)›
**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* :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*a* ⇒ '*b*] ⇒ ('*b*, '*r*) *process*$_{ptick}$›
**where** ‹*RenamingEv P f* ≡ *Renaming P f id*›

**lemma** *RenamingEv-id-unfolded* [*iff*] :
‹*Renaming P f* (λ*r*. *r*) = *RenamingEv P f*› **by** (*simp add*: *id-def*)

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

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

**lemma** *RenamingEv-SKIP* : ‹*RenamingEv* (*SKIP r*) *f* = *SKIP r*› **by** *simp*

**lemma** *RenamingEv-cont* :
‹*cont P* ⟹ *finitary f* ⟹ *cont* (λ*x*. *RenamingEv* (*P x*) *f*)› **by** *simp*

**lemma** *RenamingEv-Seq* :
‹*RenamingEv* (*P* ; *Q*) *f* = *RenamingEv P f* ; *RenamingEv Q f*›
**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* :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*r* ⇒ '*s*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›

**where** ‹*RenamingTick P g* ≡ *Renaming P id g*›

**lemma** *RenamingTick-id-unfolded* [*iff*] :
‹*Renaming P* (λ*a*. *a*) *g* = *RenamingTick P g*› **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* :
‹*cont P* ⟹ *finitary g* ⟹ *cont* (λ*x*. *RenamingTick* (*P x*) *g*)› **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 (\), see below.

**lemma** *RenamingTick-Throw* :
‹*RenamingTick* (*P* Θ *a*∈*A*. *Q a*) *g* = *RenamingTick P g* Θ *a*∈*A*. *RenamingTick* (*Q a*) *g*›
**proof** (*subst inj-on-Renaming-Throw*)
**show** ‹*inj-on id* (*events-of P* ∪ *A*)› **by** *simp*

**next**
**show** ‹*RenamingTick P g* Θ *b*∈*id* ‘ *A*. *RenamingTick* (*Q* (*inv-into A id b*)) *g* = *RenamingTick P g* Θ *a*∈*A*. *RenamingTick* (*Q a*) *g*›
**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* :
‹*RenamingEv* (*RenamingTick P g*) *f* = *Renaming P f g*›
**and** *RenamingTick-RenamingEv-is-Renaming* :
‹*RenamingTick* (*RenamingEv P f*) *g* = *Renaming P f g*›
**by** (*metis Renaming-comp comp-id fun.map-id*)+

### 5.2.3 Properties

**lemma** *isInfHidden-seqRun-imp-tickFree-seqRun* :
‹*isInfHidden-seqRun x P A t* ⟹ *tF* (*seqRun t x i*)›
**by** (*metis* $event_{ptick}$*.disc*(*1*) *image-iff isInfHidden-seqRun-imp-tickFree tickFree-seqRun-iff*)

**lemma** *tickFree-map-map-*$event_{ptick}$*-is* :
‹*tF t* ⟹ *map* (*map-*$event_{ptick}$ *f g*) *t* = *map* (*ev* ∘ *f* ∘ *of-ev*) *t*›
**by** (*induct t*) (*auto simp add*: *is-ev-def*)

**lemma** *RenamingTick-Hiding* :
‹*RenamingTick* (*P* \ *A*) *g* = *RenamingTick P g* \ *A*›
(**is** ‹*?lhs* = *?rhs*›) **for** *P* :: ‹('*a*, '*r*) $process_{ptick}$›
**proof** −
**let** *?RT* = ‹λ*P*. *RenamingTick P g*›
**let** *?th-A* = ‹λ*t*. *trace-hide t* (*ev* ‘ *A*)›
**let** *?map* = ‹*map* (*map-*$event_{ptick}$ *id g*)›
**have** $ : ‹*?th-A* (*?map t*) = *?map* (*?th-A t*)› **for** *t*
**by** (*induct t*) (*simp-all add*: *image-iff map-*$event_{ptick}$*-eq-ev-iff*)

```
have $$ : ‹map-event_ptick id g −‘ X ∪ ev ‘ A =
        map-event_ptick id g −‘ X ∪ map-event_ptick id g −‘ ev ‘ A› for X
  by (auto simp add: map-event_ptick-eq-ev-iff)
show ‹?lhs = ?rhs›
proof (rule Process-eq-optimizedI)
  fix t assume ‹t ∈ 𝒟 ?lhs›
  then obtain t1 t2 where ∗ : ‹t = ?map t1 @ t2›
    ‹tF t1› ‹ftF t2› ‹t1 ∈ 𝒟 (P \ A)› unfolding D-Renaming by blast
  from ∗(4) obtain u v where ∗∗ : ‹ftF v› ‹tF u› ‹t1 = ?th-A u @ v›
    ‹u ∈ 𝒟 P ∨ (∃ x. isInfHidden-seqRun-strong x P A u)›
    by (blast elim: D-Hiding-seqRunE)
  from ∗∗(4) show ‹t ∈ 𝒟 ?rhs›
  proof (elim disjE exE)
    assume ‹u ∈ 𝒟 P›
    with ∗∗(2) have ‹?map u ∈ 𝒟 (?RT P)›
      by (auto simp add: D-Renaming intro: front-tickFree-Nil)
    thus ‹t ∈ 𝒟 ?rhs›
      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 ∗∗∗ : ‹isInfHidden-seqRun-strong x P A u›
    have ‹isInfHidden-seqRun (ev ∘ of-ev ∘ x) (?RT P) A (?map u)›
    proof (intro allI conjI)
      fix i
      have ‹seqRun (?map u) (ev ∘ of-ev ∘ x) i = ?map (seqRun u x i)›
        by (simp add: seqRun-def image-iff ev-eq-map-event_ptick-iff)
          (metis ∗∗∗ event_ptick.sel(1) imageE)
      also have ‹?map (seqRun u x i) ∈ 𝒯 (?RT P)›
        unfolding T-Renaming using ∗∗∗ Un-iff by blast
      finally show ‹seqRun (?map u) (ev ∘ of-ev ∘ x) i ∈ 𝒯 (?RT P)› .
    next
      show ‹(ev ∘ of-ev ∘ x) i ∈ ev ‘ A› for i
        by (metis ∗∗∗ comp-apply event_ptick.sel(1) image-iff)
    qed
    thus ‹t ∈ 𝒟 ?rhs›
      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 ‹t ∈ 𝒟 ?rhs›
  then obtain u v where ∗ : ‹ftF v› ‹tF u› ‹t = ?th-A u @ v›
    ‹u ∈ 𝒟 (?RT P) ∨ (∃ x. isInfHidden-seqRun-strong x (?RT P) A u)›
    by (blast elim: D-Hiding-seqRunE)
  from ∗(4) show ‹t ∈ 𝒟 ?lhs›
  proof (elim disjE exE)
    assume ‹u ∈ 𝒟 (?RT P)›
    then obtain u1 u2 where ∗∗ : ‹u = ?map u1 @ u2›
```

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


corollary bij-Renaming-Hiding :
```

‹*Renaming* (*P* \ *S*) *f g* = *Renaming P f g* \ *f* ‘ *S*› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*bij f*›
— We already have ⟦*bij fa*; *bij ga*⟧ ⟹ *Renaming* (*Pa* \ *Sa*) *fa ga* = *Renaming Pa fa ga* \ *fa* ‘ *Sa*, but he assumption *bij g* is actually not necessary.
**proof** −
**have** ‹*?lhs* = *RenamingTick* (*RenamingEv* (*P* \ *S*) *f*) *g*›
**by** (*simp only*: *RenamingTick-RenamingEv-is-Renaming*)
**also have** ‹... = *RenamingTick* (*RenamingEv P f* \ *f* ‘ *S*) *g*›
**by** (*simp only*: *bij-RenamingEv-Hiding*[*OF* ‹*bij f*›])
**also have** ‹... = *RenamingTick* (*RenamingEv P f*) *g* \ *f* ‘ *S*›
**by** (*simp only*: *RenamingTick-Hiding*)
**also have** ‹... = *?rhs*›
**by** (*simp only*: *RenamingTick-RenamingEv-is-Renaming*)
**finally show** ‹*?lhs* = *?rhs*› **.**
**qed**

**lemma** *Renaming-is-restrictable-on-events-of-strict-ticks-of* :
‹*Renaming P f g* = *Renaming P f′ g′*›
**if** *fun-hyps* : ‹⋀*a*. *a* ∈ *α*(*P*) ⟹ *f a* = *f′ a*›
‹⋀*r*. *r* ∈ ✓**s**(*P*) ⟹ *g r* = *g′ r*›
**for** *f f′* :: ‹′*a* ⇒ ′*b*› **and** *g g′* :: ‹′*r* ⇒ ′*t*›
— probably also possible to strengthen with *strict-events-of*
**proof** −
**have** ∗ : ‹*Renaming P f g* ⊑$_{FD}$ *Renaming P f′ g′*›
**if** *fun-hyps-bis* : ‹⋀*a*. *a* ∈ *α*(*P*) ⟹ *f a* = *f′ a*› ‹⋀*r*. *r* ∈ ✓**s**(*P*) ⟹ *g r* = *g′ r*›
**for** *f f′* :: ‹′*a* ⇒ ′*b*› **and** *g g′* :: ‹′*r* ⇒ ′*t*›
**proof** −
**have** $ : ‹*map* (*map-event*$_{ptick}$ *f g*) *u* = *map* (*map-event*$_{ptick}$ *f′ g′*) *u*›
**if** ‹*u* ∈ *T P*› **and** ‹*tF u*› **for** *u*
**proof** −
**from** ‹*u* ∈ *T P*› **have** ‹*ev a* ∈ *set u* ⟹ *a* ∈ *α*(*P*)› **for** *a*
**by** (*meson events-of-memI*)
**with** ‹*tF u*› **show** ‹*map* (*map-event*$_{ptick}$ *f g*) *u* = *map* (*map-event*$_{ptick}$ *f′ g′*) *u*›
**by** (*induct u*, *simp-all*)
(*metis event*$_{ptick}$*.collapse*(*1*) *event*$_{ptick}$*.simps*(*9*) *fun-hyps-bis*(*1*))
**qed**
**have** ‹(∀ *t*. *t* ∈ *D* (*Renaming P f′ g′*) ⟶ *t* ∈ *D* (*Renaming P f g*)) ∧
(∀ *t X*. (*t*, *X*) ∈ *F* (*Renaming P f′ g′*) ⟶ *t* ∉ *D* (*Renaming P f′ g′*) ⟶ (*t*, *X*) ∈ *F* (*Renaming P f g*))›
**proof** (*intro conjI allI impI*)
**fix** *t* **assume** ‹*t* ∈ *D* (*Renaming P f′ g′*)›
**then obtain** *u1 u2* **where** ∗ : ‹*t* = *map* (*map-event*$_{ptick}$ *f′ g′*) *u1* @ *u2*›
‹*tF u1*› ‹*ftF u2*› ‹*u1* ∈ *D P*› **unfolding** *D-Renaming* **by** *blast*
**have** ‹*map* (*map-event*$_{ptick}$ *f′ g′*) *u1* = *map* (*map-event*$_{ptick}$ *f g*) *u1*›
**by** (*simp add*: ‹*tF u1*› $ ∗(*4*) *D-T*)
**with** ∗ **show** ‹*t* ∈ *D* (*Renaming P f g*)›
**by** (*auto simp add*: *D-Renaming*)

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

**by** (*simp add*: ‹*tF u'*› *is-processT7 map-event*$_{ptick}$*-tickFree*)
**thus** ‹*map* (*map-event*$_{ptick}$ *f' g'*) *u'* @ [✓(*g' r*)] ∈ $\mathcal{T}$ (*Renaming P f g*)›
**by** (*simp add*: *D-T*)
**qed**
**hence** ‹(*map* (*map-event*$_{ptick}$ *f' g'*) *u'* @ [✓(*g' r*)], *X*) ∈ $\mathcal{F}$ (*Renaming P f g*)›
**by** (*simp add*: *tick-T-F*)
**also have** ‹*map* (*map-event*$_{ptick}$ *f' g'*) *u'* @ [✓(*g' r*)] = *t*›
**by** (*simp add*: ∗(*1*) ‹*u* = *u'* @ [✓(*r*)]›)
**finally show** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*Renaming P f g*)› **.**
**qed**
**qed**
**thus** ‹*Renaming P f g* ⊑$_{FD}$ *Renaming P f' g'*›
**by** (*auto simp add*: *refine-defs intro*: *is-processT8*)
**qed**
**show** ‹*Renaming P f g* = *Renaming P f' g'*›
**proof** (*rule FD-antisym*)
**show** ‹*Renaming P f g* ⊑$_{FD}$ *Renaming P f' g'*› ‹*Renaming P f' g'* ⊑$_{FD}$ *Renaming P f g*›
**by** (*simp-all add*: ∗ *fun-hyps*)
**qed**
**qed**

**corollary** *Renaming-is-restrictable-on-events-of-ticks-of* :
‹⟦⋀*a*. *a* ∈ α(*P*) ⟹ *f a* = *f' a*; ⋀*r*. *r* ∈ ✓*s*(*P*) ⟹ *g r* = *g' r*⟧
⟹ *Renaming P f g* = *Renaming P f' g'*›
**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* :
‹(⋀*a*. *a* ∈ α(*P*) ⟹ *f a* = *f' a*) ⟹ *RenamingEv P f* = *RenamingEv P f'*›
**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* :
‹(⋀*r*. *r* ∈ ✓**s**(*P*) ⟹ *g r* = *g' r*) ⟹ *RenamingTick P g* = *RenamingTick P g'*›
**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* :
‹(⋀*r*. *r* ∈ ✓*s*(*P*) ⟹ *g r* = *g' r*) ⟹ *RenamingTick P g* = *RenamingTick P g'*›
**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** $Sync_{ptick}$-*locale*) *inj-on-RenamingTick-Sync*$_{ptick}$ :
‹*RenamingTick* (*P* $[\![S]\!]_{\checkmark}$ *Q*) *g* =
*Sync*$_{ptick}$-*locale.Sync*$_{ptick}$ (λ*r s*. *case r* ⊗✓ *s of* ⌊*r-s*⌋ ⇒ ⌊*g r-s*⌋ | ◊ ⇒ ◊) *P S Q*›
(**is** ‹*?lhs* = *?rhs*›)
**if** *inj-on-g* : ‹*inj-on g range-tick-join*›
**proof** −
**let** *?map-evt* = ‹λ*g*. *map* (*map-event*$_{ptick}$ *id g*)›
**let** *?tick-join*′ = ‹λ*r s*. *case r* ⊗✓ *s of* ⌊*r-s*⌋ ⇒ ⌊*g r-s*⌋ | ◊ ⇒ ◊›
**interpret** *Sync*$_{ptick}$′ : *Sync*$_{ptick}$-*locale ?tick-join*′
**by** (*intro interpretable-inj-on-range-tick-join inj-on-g*)
— Thus *Sync*$_{ptick}$′.*Sync*$_{ptick}$ *P S Q* is well defined.
**have** *inj-on-inv-into-g* :
‹*inj-on* (*inv-into range-tick-join g*) *Sync*$_{ptick}$′.*range-tick-join*›
**by** (*rule inj-onI*, *simp split*: *option.split-asm*)
(*metis* (*mono-tags*, *lifting*) *f-inv-into-f image-eqI mem-Collect-eq*)
**from** *inv-into-f-f inj-on-g* **have** *expanded-tick-join* :
‹(⊗✓) = (λ*r s*. *case ?tick-join*′ *r s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊*inv-into range-tick-join g r-s*⌋)›
**by** (*fastforce split*: *split*: *option.split*)
**show** ‹*?lhs* = *?rhs*›
**proof** (*rule Process-eq-optimizedI*)
**fix** *t* **assume** ‹*t* ∈ 𝒟 *?lhs*›
**then obtain** *u1 u2* **where** ∗ : ‹*t* = *map* (*map-event*$_{ptick}$ *id g*) *u1* @ *u2*›
‹*tF u1*› ‹*ftF u2*› ‹*u1* ∈ 𝒟 (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**unfolding** *D-Renaming* **by** *blast*
**from** ∗(*4*) **obtain** *v1 w1 t-P t-Q*
**where** ∗∗ : ‹*u1* = *v1* @ *w1*› ‹*tF v1*› ‹*ftF w1*›
‹*v1 setinterleaves*$_{\checkmark}$ *tick-join* ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ 𝒟 *P* ∧ *t-Q* ∈ 𝒯 *Q* ∨ *t-P* ∈ 𝒯 *P* ∧ *t-Q* ∈ 𝒟 *Q*›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** *inj-on-map-map-event*$_{ptick}$-*setinterleaves*$_{ptick}$[*OF inj-on-g* ∗∗(*4*)]
**have** ‹*?map-evt g v1 setinterleaves*$_{\checkmark}$ *?tick-join*′ ((*t-P*, *t-Q*), *S*)› .
**moreover from** ∗(*1*−*3*) ∗∗(*1*, *2*)
**have** ‹*t* = *?map-evt g v1* @ (*?map-evt g w1* @ *u2*) ∧
*tF* (*?map-evt g v1*) ∧ *ftF* (*?map-evt g w1* @ *u2*)›
**by** (*simp add*: *front-tickFree-append-iff map-event*$_{ptick}$-*tickFree*)
**ultimately show** ‹*t* ∈ 𝒟 *?rhs*›
**using** ∗∗(*5*) **by** (*simp* (*no-asm*) *add*: *Sync*$_{ptick}$′.*D-Sync*$_{ptick}$) *blast*
**next**
**fix** *t* **assume** ‹*t* ∈ 𝒟 *?rhs*›
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹*t* = *u* @ *v*› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{\checkmark}$ *?tick-join*′ ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ 𝒟 *P* ∧ *t-Q* ∈ 𝒯 *Q* ∨ *t-P* ∈ 𝒯 *P* ∧ *t-Q* ∈ 𝒟 *Q*›
**unfolding** *Sync*$_{ptick}$′.*D-Sync*$_{ptick}$ **by** *blast*

**from** ‹*tF u*› **have** ‹*e* ∈ *set u* ⟹ *map-event*$_{ptick}$ *id* (*g* ∘ *inv-into range-tick-join g*) *e* = *e*› **for** *e*
**by** (*cases e*) (*simp-all add*: *tickFree-def disjoint-iff*)
**hence** ‹*t* = *?map-evt g* (*?map-evt* (*inv-into range-tick-join g*) *u*) @ *v*›
**by** (*simp add*: ∗(*1*) *flip*: *map-event*$_{ptick}$*-comp*)
(*induct u*, *simp-all*)
**moreover have** ‹*tF* (*?map-evt* (*inv-into range-tick-join g*) *u*)›
**by** (*simp add*: ∗(*2*) *map-event*$_{ptick}$*-tickFree*)
**moreover**
**{**
**have** ‹*?map-evt* (*inv-into range-tick-join g*) *u* =
*?map-evt* (*inv-into range-tick-join g*) *u* @ []› **by** *simp*
**moreover have** ‹*tF* ((*map* (*map-event*$_{ptick}$ *id* (*inv-into range-tick-join g*))) *u*)›
**by** (*simp add*: ∗(*2*) *map-event*$_{ptick}$*-tickFree*)
**moreover have** ‹*ftF* []› **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** ‹*?map-evt* (*inv-into range-tick-join g*) *u*
*setinterleaves*$\checkmark_{tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)› **.**
**ultimately have** ‹*?map-evt* (*inv-into range-tick-join g*) *u* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_\checkmark$ *Q*)›
**unfolding** *D-Sync*$_{ptick}$ **using** ∗(*5*) **by** *blast*
**}**
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming* **using** ∗(*3*) **by** *blast*
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› ‹*t* ∉ $\mathcal{D}$ *?lhs*›
**then obtain** *u*
**where** ∗ : ‹*t* = *?map-evt g u*› ‹(*u*, *map-event*$_{ptick}$ *id g* −‘ *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧$_\checkmark$ *Q*)›
**unfolding** *Renaming-projs* **by** *blast*
**with** ‹*t* ∉ $\mathcal{D}$ *?lhs*› **obtain** *t-P t-Q X-P X-Q*
**where** ∗∗ : ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*u setinterleaves*$\checkmark_{tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*map-event*$_{ptick}$ *id g* −‘ *X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P S X-Q*›
**by** (*auto simp add*: *D-Renaming Sync*$_{ptick}$*-projs*)
(*metis append.right-neutral front-tickFree-Nil map-event*$_{ptick}$*-front-tickFree*)+
**have** ‹*t setinterleaves*$\checkmark_{?tick\text{-}join'}$ ((*t-P*, *t-Q*), *S*)›
**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** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *?tick-join′ X-P S X-Q*› **.**
**ultimately show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**unfolding** *Sync*$_{ptick}$*′.F-Sync*$_{ptick}$ **using** ∗∗(*1*, *2*) **by** *fast*
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› ‹*t* ∉ $\mathcal{D}$ *?rhs*›
**then obtain** *t-P t-Q X-P X-Q*
**where** ∗ : ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›

```
      ‹t setinterleaves✓ ?tick-join' ((t-P, t-Q), S)›
      ‹X ⊆ super-ref-Sync_ptick ?tick-join' X-P S X-Q›
    unfolding Sync_ptick'.Sync_ptick-projs by blast
  from Sync_ptick'.setinterleaves_ptick-imp-set-range-tick-join[OF ∗(3)]
  have ‹{r-s. ✓(r-s) ∈ set t} ⊆ Sync_ptick'.range-tick-join› .
  hence ‹e ∈ set t ⟹ map-event_ptick id (g ∘ inv-into range-tick-join g) e = e›
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 ‹t = ?map-evt g (?map-evt (inv-into range-tick-join g) t)›
    by (simp add: ∗(1) flip: map-event_ptick-comp)
      (induct t, simp-all)
  moreover
  { from Sync_ptick'.inj-on-map-map-event_ptick-setinterleaves_ptick
      [OF inj-on-inv-into-g ∗(3), folded expanded-tick-join]
    have ‹?map-evt (inv-into range-tick-join g) t
        setinterleaves✓ tick-join ((t-P, t-Q), S)› .
    moreover from vimage-inj-on-subset-super-ref-Sync_ptick-iff
      [OF inj-on-g, THEN iffD2, OF ∗(4)]
    have ‹map-event_ptick id g −‘ X ⊆
        super-ref-Sync_ptick (⊗✓) X-P S X-Q› .
    ultimately have ‹(?map-evt (inv-into range-tick-join g) t,
                  map-event_ptick id g −‘ X) ∈ ℱ (P ⟦S⟧✓ Q)›
      by (auto simp add: F-Sync_ptick ∗(1, 2))
  }
  ultimately show ‹(t, X) ∈ ℱ ?lhs› unfolding F-Renaming by blast
 qed
qed
```

```
lemma (in Sync_ptick-locale) inj-RenamingTick-Sync_ptick-inj-RenamingTick :
 ‹RenamingTick P g ⟦S⟧✓ RenamingTick Q h =
  Sync_ptick-locale.Sync_ptick (λr s. g r ⊗✓ h s) P S Q› (is ‹?lhs = ?rhs›)
 if ‹inj g› and ‹inj h›

 for P :: ‹('a, 'r') process_ptick› and Q :: ‹('a, 's') process_ptick›
proof −
 interpret tjoin-interpreted : Sync_ptick-locale ‹λr s. g r ⊗✓ h s›
   by unfold-locales (meson injD inj-tick-join ‹inj g› ‹inj h›)
 let ?map-evt = ‹λg. map (map-event_ptick id g)›
 let ?map-ev  = ‹λt. map ev (map of-ev t)›
 let ?RT      = RenamingTick
 have ∗ : ‹tF t ⟹ ?map-evt g t = ?map-ev t› for t :: ‹('a, 'r') trace_ptick›
   by (induct t) (auto simp add: is-ev-def)
 have ∗∗ : ‹tF t ⟹ ?map-evt h t = ?map-ev t› for t :: ‹('a, 's') trace_ptick›
   by (induct t) (auto simp add: is-ev-def)
 have ∗∗∗ : ‹t setinterleaves✓(⊗✓) ((?map-evt g u, ?map-evt h v), S)
     ⟷ t setinterleaves✓λr s. g r ⊗✓ h s ((u, v), S)› for t u v
```

**by** (*induct* ‹(λr s. *g r* ⊗✓ *h s*, *u*, *S*, *v*)› *arbitrary*: *t u v*) (*auto split*: *option.split*)
**have** ∗∗∗∗ : ‹*?map-evt g t* = *?map-evt g t'* ⟷ *t* = *t'*› **for** *t t'* :: ‹('*a*, '*r*') *trace*$_{ptick}$›
**by** (*rule iffI*, *induct t arbitrary*: *t'*, *auto*)
(*metis event*$_{ptick}$*.inj-map id-apply inj-def* ‹*inj g*›)
**have** ∗∗∗∗∗ : ‹*?map-evt h t* = *?map-evt h t'* ⟷ *t* = *t'*› **for** *t t'* :: ‹('*a*, '*s*') *trace*$_{ptick}$›
**by** (*rule iffI*, *induct t arbitrary*: *t'*, *auto*)
(*metis event*$_{ptick}$*.inj-map id-apply inj-def* ‹*inj h*›)
**show** ‹*?lhs* = *?rhs*›
**proof** (*rule Process-eq-optimizedI*)
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**then obtain** *u v t-P' t-Q'* **where** $ : ‹*t* = *u* @ *v*› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*✓(⊗✓) ((*t-P'*, *t-Q'*), *S*)›
‹*t-P'* ∈ $\mathcal{D}$ (*?RT P g*) ∧ *t-Q'* ∈ $\mathcal{T}$ (*?RT Q h*) ∧ *t-Q'* ∉ $\mathcal{D}$ (*?RT Q h*) ∨
*t-P'* ∈ $\mathcal{T}$ (*?RT P g*) ∧ *t-P'* ∉ $\mathcal{D}$ (*?RT P g*) ∧ *t-Q'* ∈ $\mathcal{D}$ (*?RT Q h*) ∨
*t-P'* ∈ $\mathcal{D}$ (*?RT P g*) ∧ *t-Q'* ∈ $\mathcal{D}$ (*?RT Q h*)›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** $(5) **show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**proof** (*elim disjE conjE*)
**assume** ‹*t-P'* ∈ $\mathcal{D}$ (*?RT P g*)› ‹*t-Q'* ∈ $\mathcal{T}$ (*?RT Q h*)› ‹*t-Q'* ∉ $\mathcal{D}$ (*?RT Q h*)›
**then obtain** *t-P*$_1$ *t-P*$_2$ *t-Q*
**where** $$ : ‹*t-P'* = *?map-evt g t-P*$_1$ @ *t-P*$_2$› ‹*tF t-P*$_1$› ‹*ftF t-P*$_2$› ‹*t-P*$_1$ ∈ $\mathcal{D}$ *P*›
‹*t-Q'* = *?map-evt h t-Q*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› **unfolding** *Renaming-projs* **by** *blast*
**from** $(4)[*unfolded* $$(1), *THEN setinterleaves*$_{ptick}$*-appendL*]
**obtain** *u*$_1$ *u*$_2$ *t-Q'*$_1$ *t-Q'*$_2$ **where** $$$ : ‹*u* = *u*$_1$ @ *u*$_2$› ‹*t-Q'* = *t-Q'*$_1$ @ *t-Q'*$_2$›
‹*u*$_1$ *setinterleaves*✓(⊗✓) ((*?map-evt g t-P*$_1$, *t-Q'*$_1$), *S*)› **by** *blast*
**obtain** *t-Q*$_1$ *t-Q*$_2$ **where** ‹*t-Q* = *t-Q*$_1$ @ *t-Q*$_2$› ‹*t-Q'*$_1$ = *?map-evt h t-Q*$_1$›
**by** (*metis* $$(5) $$$(2) *map-eq-append-conv*)
**from** $$$(3)[*unfolded this*(2), *THEN* ∗∗∗[*THEN iffD1*]]
**have** ‹*u*$_1$ *setinterleaves*✓λr s. *g r* ⊗✓ *h s* ((*t-P*$_1$, *t-Q*$_1$), *S*)› **.**
**moreover from** ‹*t-Q* = *t-Q*$_1$ @ *t-Q*$_2$› *is-processT3-TR-append* $$(6)
**have** ‹*t-Q*$_1$ ∈ $\mathcal{T}$ *Q*› **by** *blast*
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**using** $(1−3) $$(4) $$$(1) *front-tickFree-append*
**by** (*auto simp add*: *tjoin-interpreted.D-Sync*$_{ptick}$)
**next**
**assume** ‹*t-Q'* ∈ $\mathcal{D}$ (*?RT Q h*)› ‹*t-P'* ∈ $\mathcal{T}$ (*?RT P g*)› ‹*t-P'* ∉ $\mathcal{D}$ (*?RT P g*)›
**then obtain** *t-Q*$_1$ *t-Q*$_2$ *t-P*
**where** $$ : ‹*t-Q'* = *?map-evt h t-Q*$_1$ @ *t-Q*$_2$› ‹*tF t-Q*$_1$› ‹*ftF t-Q*$_2$› ‹*t-Q*$_1$ ∈ $\mathcal{D}$ *Q*›
‹*t-P'* = *?map-evt g t-P*› ‹*t-P* ∈ $\mathcal{T}$ *P*› **unfolding** *Renaming-projs* **by** *blast*
**from** $(4)[*unfolded* $$(1), *THEN setinterleaves*$_{ptick}$*-appendR*]
**obtain** *u*$_1$ *u*$_2$ *t-P'*$_1$ *t-P'*$_2$ **where** $$$ : ‹*u* = *u*$_1$ @ *u*$_2$› ‹*t-P'* = *t-P'*$_1$ @ *t-P'*$_2$›
‹*u*$_1$ *setinterleaves*✓(⊗✓) ((*t-P'*$_1$, *?map-evt h t-Q*$_1$), *S*)› **by** *blast*
**obtain** *t-P*$_1$ *t-P*$_2$ **where** ‹*t-P* = *t-P*$_1$ @ *t-P*$_2$› ‹*t-P'*$_1$ = *?map-evt g t-P*$_1$›
**by** (*metis* $$(5) $$$(2) *map-eq-append-conv*)
**from** $$$(3)[*unfolded this*(2), *THEN* ∗∗∗[*THEN iffD1*]]

**have** ‹$u_1$ *setinterleaves*$_{✓\lambda r\ s.\ g\ r\ \otimes✓\ h\ s}$ ((*t-P*$_1$, *t-Q*$_1$), *S*)› **.**
**moreover from** ‹*t-P* = *t-P*$_1$ @ *t-P*$_2$› *is-processT3-TR-append* $$(*6*)
**have** ‹*t-P*$_1$ ∈ $\mathcal{T}$ *P*› **by** *blast*
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**using** $(*1−3*) $$(*4*) $$$(*1*) *front-tickFree-append*
**by** (*auto simp add*: *tjoin-interpreted.D-Sync*$_{ptick}$)
**next**
**assume** ‹*t-P′* ∈ $\mathcal{D}$ (*?RT P g*)› ‹*t-Q′* ∈ $\mathcal{D}$ (*?RT Q h*)›
**then obtain** *t-P*$_1$ *t-P*$_2$ *t-Q*$_1$ *t-Q*$_2$
**where** $$ : ‹*t-P′* = *?map-evt g t-P*$_1$ @ *t-P*$_2$› ‹*tF t-P*$_1$› ‹*ftF t-P*$_2$› ‹*t-P*$_1$ ∈ $\mathcal{D}$ *P*›
‹*t-Q′* = *?map-evt h t-Q*$_1$ @ *t-Q*$_2$› ‹*tF t-Q*$_1$› ‹*ftF t-Q*$_2$› ‹*t-Q*$_1$ ∈ $\mathcal{D}$ *Q*›
**unfolding** *D-Renaming* **by** *blast*
**from** $(*4*)[*unfolded* $$(*1*, *5*), *THEN setinterleaves*$_{ptick}$*-appendL*]
**obtain** *u*$_1$ *u*$_2$ *t-Q*$_1$*-bis t-Q*$_2$*-bis*
**where** $$$ : ‹*u* = *u*$_1$ @ *u*$_2$› ‹*?map-evt h t-Q*$_1$ @ *t-Q*$_2$ = *t-Q*$_1$*-bis* @ *t-Q*$_2$*-bis*›
‹*u*$_1$ *setinterleaves*$_{✓(\otimes✓)}$ ((*?map-evt g t-P*$_1$, *t-Q*$_1$*-bis*), *S*)› **by** *blast*
**from** $$$(*2*) **have** ‹*t-Q*$_1$*-bis* = *?map-evt h* (*take* (*length t-Q*$_1$*-bis*) *t-Q*$_1$) ∨
*t-Q*$_1$*-bis* = *?map-evt h t-Q*$_1$ @ *take* (*length t-Q*$_1$*-bis* − *length t-Q*$_1$) *t-Q*$_2$›
**by** (*cases* ‹*length t-Q*$_1$*-bis* ≤ *length t-Q*$_1$›)
(*simp-all add*: *append-eq-append-conv-if take-map split*: *if-split-asm*)
**thus** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**proof** (*elim disjE*)
**assume** ‹*t-Q*$_1$*-bis* = *?map-evt h* (*take* (*length t-Q*$_1$*-bis*) *t-Q*$_1$)›
**hence** ‹*u*$_1$ *setinterleaves*$_{✓\lambda r\ s.\ g\ r\ \otimes✓\ h\ s}$ ((*t-P*$_1$, *take* (*length t-Q*$_1$*-bis*) *t-Q*$_1$), *S*)›
**by** (*metis* $$$(*3*) ***)
**moreover have** ‹*take* (*length t-Q*$_1$*-bis*) *t-Q*$_1$ ∈ $\mathcal{T}$ *Q*›
**by** (*metis* $$(*8*) *D-T append-take-drop-id is-processT3-TR-append*)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**using** $(*1−3*) $$(*4*) $$$(*1*) *front-tickFree-append*
**by** (*auto simp add*: *tjoin-interpreted.D-Sync*$_{ptick}$)
**next**
**assume** ‹*t-Q*$_1$*-bis* = *?map-evt h t-Q*$_1$ @ *take* (*length t-Q*$_1$*-bis* − *length t-Q*$_1$) *t-Q*$_2$›
**with** $$$(*3*)
**have** ‹*u*$_1$ *setinterleaves*$_{✓(\otimes✓)}$ ((*?map-evt g t-P*$_1$,
*?map-evt h t-Q*$_1$ @ *take* (*length t-Q*$_1$*-bis* − *length t-Q*$_1$) *t-Q*$_2$), *S*)›
**by** *simp*
**from** *setinterleaves*$_{ptick}$*-appendR*[*OF this*] **obtain** *u*$_{11}$ *u*$_{12}$ *t-P*$_{11}$ *t-P*$_{12}$
**where** $$$$ : ‹*u*$_1$ = *u*$_{11}$ @ *u*$_{12}$› ‹*?map-evt g t-P*$_1$ = *t-P*$_{11}$ @ *t-P*$_{12}$›
‹*u*$_{11}$ *setinterleaves*$_{✓(\otimes✓)}$ ((*t-P*$_{11}$, *?map-evt h t-Q*$_1$), *S*)› **by** *blast*
**have** ‹*t-P*$_{11}$ = *?map-evt g* (*take* (*length t-P*$_{11}$) *t-P*$_1$)›
**by** (*metis* $$$$(*2*) *append-eq-conv-conj take-map*)
**hence** ‹*u*$_{11}$ *setinterleaves*$_{✓\lambda r\ s.\ g\ r\ \otimes✓\ h\ s}$((*take* (*length t-P*$_{11}$) *t-P*$_1$, *t-Q*$_1$), *S*)›
**by** (*metis* $$$$(*3*) ***)
**moreover have** ‹*take* (*length t-P*$_{11}$) *t-P*$_1$ ∈ $\mathcal{T}$ *P*›

```
      by (metis $$(4) D-T append-take-drop-id is-processT3-TR-append)
    ultimately show ‹t ∈ 𝒟 ?rhs›
      by (simp add: tjoin-interpreted.D-Sync_ptick)
        (metis (no-types, lifting) $(1,2,3) $$(8) $$$(1) $$$$(1)
          append.assoc front-tickFree-append tickFree-append-iff)
  qed
 qed
next
 fix t assume ‹t ∈ 𝒟 ?rhs›
 then obtain u v t-P t-Q where $ : ‹t = u @ v› ‹tF u› ‹ftF v›
  ‹u setinterleaves✓ λr s. g r ⊗✓ h s ((t-P, t-Q), S)›
  ‹t-P ∈ 𝒟 P ∧ t-Q ∈ ℱ Q ∨ t-P ∈ ℱ P ∧ t-Q ∈ 𝒟 Q›
  unfolding tjoin-interpreted.D-Sync_ptick by blast
 from tickFree-setinterleaves_ptick-iff[THEN iffD1, OF $(4, 2)]
 have ‹tF t-P› ‹tF t-Q› by simp-all

 with $(5) have ‹?map-evt g t-P ∈ 𝒟 (?RT P g) ∧ ?map-evt h t-Q ∈ ℱ (?RT
Q h) ∨
                ?map-evt g t-P ∈ ℱ (?RT P g) ∧ ?map-evt h t-Q ∈ 𝒟 (?RT Q h)›
  by (simp add: Renaming-projs) (metis append.right-neutral front-tickFree-Nil)
 moreover from ***[THEN iffD2, OF $(4)]
 have ‹u setinterleaves✓(⊗✓) ((?map-evt g t-P, ?map-evt h t-Q), S)› .
 ultimately show ‹t ∈ 𝒟 ?lhs›
  using $(1−3) by (auto simp add: D-Sync_ptick)
next
 fix t X assume ‹(t, X) ∈ ℱ ?lhs› ‹t ∉ 𝒟 ?lhs›
 then obtain t-P′ X-P′ t-Q′ X-Q′
  where $ : ‹(t-P′, X-P′) ∈ ℱ (?RT P g)› ‹(t-Q′, X-Q′) ∈ ℱ (?RT Q h)›
   ‹t setinterleaves✓(⊗✓) ((t-P′, t-Q′), S)› ‹X ⊆ super-ref-Sync_ptick (⊗✓) X-P′
S X-Q′›
  unfolding Sync_ptick-projs by blast
 from ‹t ∉ 𝒟 ?lhs› $(1, 2)[THEN F-T] $(3)
 have ‹t-P′ ∉ 𝒟 (?RT P g) ∧ t-Q′ ∉ 𝒟 (?RT Q h)›
  by (simp add: D-Sync_ptick′) (metis append-self-conv front-tickFree-Nil)
 with $(1, 2) obtain t-P t-Q
  where $$ : ‹t-P′ = ?map-evt g t-P› ‹(t-P, map-event_ptick id g −‘ X-P′) ∈ ℱ
P›
    ‹t-Q′ = ?map-evt h t-Q› ‹(t-Q, map-event_ptick id h −‘ X-Q′) ∈ ℱ Q›
  unfolding Renaming-projs by blast
 from $(3)[unfolded $$(1, 3), THEN ***[THEN iffD1]]
 have ‹t setinterleaves✓ λr s. g r ⊗✓ h s ((t-P, t-Q), S)› .
 moreover from $(4) inj-tick-join
 have ‹X ⊆ super-ref-Sync_ptick (λr s. g r ⊗✓ h s)
      (map-event_ptick id g −‘ X-P′) S (map-event_ptick id h −‘ X-Q′)›
  by (simp add: super-ref-Sync_ptick-def, safe) blast
 ultimately show ‹(t, X) ∈ ℱ ?rhs›
  using $$(2, 4) by (auto simp add: tjoin-interpreted.F-Sync_ptick)
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*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*t setinterleaves*$_{✓\lambda r\ s.\ g\ r \otimes ✓\ h\ s}$ ((*t-P*, *t-Q*), *S*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ (λ*r s*. *g r* ⊗✓ *h s*) *X-P S X-Q*›
**unfolding** *tjoin-interpreted.Sync*$_{ptick}$*-projs* **by** *blast*
**from** ‹*t* ∉ $\mathcal{D}$ *?rhs*› **have** ‹*t-P* ∉ $\mathcal{D}$ *P* ∧ *t-Q* ∉ $\mathcal{D}$ *Q*›
**by** (*simp add*: *tjoin-interpreted.D-Sync*$_{ptick}$′)
(*metis* $(1−3) *F-T append.right-neutral front-tickFree-Nil*)
**hence** $$ : ‹*t-P* @ [✓(*r*)] ∈ $\mathcal{T}$ *P* ⟹ *r* ∈ ✓**s**(*P*)›
‹*t-Q* @ [✓(*s*)] ∈ $\mathcal{T}$ *Q* ⟹ *s* ∈ ✓**s**(*Q*)› **for** *r s*
**by** (*meson is-processT9 strict-ticks-of-memI*)+
**have** $$$ : ‹*?map-evt g t-P* @ [✓(*g-r*)] ∈ $\mathcal{T}$ (*?RT P g*) ⟷ (∃ *r*. *g-r* = *g r* ∧ *t-P* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*)› **for** *g-r*
**proof** (*rule iffI*)
**from** ‹*t-P* ∉ $\mathcal{D}$ *P* ∧ *t-Q* ∉ $\mathcal{D}$ *Q*› **have** ‹*?map-evt g t-P* ∉ $\mathcal{D}$ (*?RT P g*)›
**by** (*simp add*: *D-Renaming map-eq-append-conv* ∗∗∗∗)
(*use is-processT7 map-event*$_{ptick}$*-front-tickFree* **in** *blast*)
**hence** ‹*?map-evt g t-P* @ [✓(*g-r*)] ∉ $\mathcal{D}$ (*?RT P g*)› **by** (*meson is-processT9*)
**moreover assume** ‹*?map-evt g t-P* @ [✓(*g-r*)] ∈ $\mathcal{T}$ (*?RT P g*)›
**ultimately show** ‹*?map-evt g t-P* @ [✓(*g-r*)] ∈ $\mathcal{T}$ (*?RT P g*) ⟹ ∃ *r*. *g-r* = *g r* ∧ *t-P* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*›
**by** (*auto simp add*: *Renaming-projs append-eq-map-conv tick-eq-map-event*$_{ptick}$*-iff* ∗∗∗∗)
**next**
**show** ‹∃ *r*. *g-r* = *g r* ∧ *t-P* @ [✓(*r*)] ∈ $\mathcal{T}$ *P* ⟹ *?map-evt g t-P* @ [✓(*g-r*)] ∈ $\mathcal{T}$ (*?RT P g*)›
**by** (*auto simp add*: *T-Renaming*)
**qed**
**have** $$$$ : ‹*?map-evt h t-Q* @ [✓(*h-s*)] ∈ $\mathcal{T}$ (*?RT Q h*) ⟷ (∃ *s*. *h-s* = *h s* ∧ *t-Q* @ [✓(*s*)] ∈ $\mathcal{T}$ *Q*)› **for** *h-s*
**proof** (*rule iffI*)
**from** ‹*t-P* ∉ $\mathcal{D}$ *P* ∧ *t-Q* ∉ $\mathcal{D}$ *Q*› **have** ‹*?map-evt h t-Q* ∉ $\mathcal{D}$ (*?RT Q h*)›
**by** (*simp add*: *D-Renaming map-eq-append-conv* ∗∗∗∗∗)
(*use is-processT7 map-event*$_{ptick}$*-front-tickFree* **in** *blast*)
**hence** ‹*?map-evt h t-Q* @ [✓(*h-s*)] ∉ $\mathcal{D}$ (*?RT Q h*)› **by** (*meson is-processT9*)
**moreover assume** ‹*?map-evt h t-Q* @ [✓(*h-s*)] ∈ $\mathcal{T}$ (*?RT Q h*)›
**ultimately show** ‹*?map-evt h t-Q* @ [✓(*h-s*)] ∈ $\mathcal{T}$ (*?RT Q h*) ⟹ ∃ *s*. *h-s* = *h s* ∧ *t-Q* @ [✓(*s*)] ∈ $\mathcal{T}$ *Q*›
**by** (*auto simp add*: *Renaming-projs append-eq-map-conv tick-eq-map-event*$_{ptick}$*-iff* ∗∗∗∗∗)
**next**
**show** ‹∃ *s*. *h-s* = *h s* ∧ *t-Q* @ [✓(*s*)] ∈ $\mathcal{T}$ *Q* ⟹ *?map-evt h t-Q* @ [✓(*h-s*)] ∈ $\mathcal{T}$ (*?RT Q h*)›
**by** (*auto simp add*: *T-Renaming*)
**qed**

**define** *X-P*′ **where** ‹*X-P*′ ≡ *map-event*$_{ptick}$ *id g* ‘ *X-P* ∪ {✓(*g-r*) |*g-r*. *?map-evt g t-P* @ [✓(*g-r*)] ∉ $\mathcal{T}$ (*?RT P g*)}›
**define** *X-Q*′ **where** ‹*X-Q*′ ≡ *map-event*$_{ptick}$ *id h* ‘ *X-Q* ∪ {✓(*h-s*) |*h-s*. *?map-evt*

*h t-Q* @ [✓(*h-s*)] ∉ $\mathcal{T}$ (*?RT Q h*)}›

**have** ‹*map-event*$_{ptick}$ *id g* −‘ (*map-event*$_{ptick}$ *id g* ‘ *X-P*) = *X-P*›
‹*map-event*$_{ptick}$ *id h* −‘ (*map-event*$_{ptick}$ *id h* ‘ *X-Q*) = *X-Q*›
**by** (*simp-all add*: *set-eq-iff image-iff*)
(*metis event*$_{ptick}$*.inj-map injD inj-on-id* ‹*inj g*›,
*metis event*$_{ptick}$*.inj-map injD inj-on-id* ‹*inj h*›)
**with** $(*1*, *2*) **have** ‹(*?map-evt g t-P*, *map-event*$_{ptick}$ *id g* ‘ *X-P*) ∈ $\mathcal{F}$ (*?RT P g*)›
‹(*?map-evt h t-Q*, *map-event*$_{ptick}$ *id h* ‘ *X-Q*) ∈ $\mathcal{F}$ (*?RT Q h*)›
**by** (*auto simp add*: *F-Renaming*)
**hence** ‹(*?map-evt g t-P*, *X-P′*) ∈ $\mathcal{F}$ (*?RT P g*)›
‹(*?map-evt h t-Q*, *X-Q′*) ∈ $\mathcal{F}$ (*?RT Q h*)›
**by** (*auto simp add*: *X-P′-def X-Q′-def intro*: *is-processT5 F-T*)
**moreover have** ‹*t setinterleaves*$_{✓(⊗✓)}$ ((*?map-evt g t-P*, *?map-evt h t-Q*), *S*)›
**by** (*simp add*: $(*3*) ∗∗∗)
**moreover have** ‹*e* ∈ *X* ⟹ *e* ∈ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P′ S X-Q′*› **for** *e*
**using** $(*4*)[*THEN set-mp*, *of e*]
**by** (*cases e*,
*simp-all add*: *X-P′-def X-Q′-def super-ref-Sync*$_{ptick}$*-def image-iff*
*ev-eq-map-event*$_{ptick}$*-iff tick-eq-map-event*$_{ptick}$*-iff* $$$ $$$$)
*metis*
**ultimately show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **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: $P \ [\![A]\!] \ Q = Q \ [\![A]\!] \ P$ but in our generalization such a law cannot be obtained in all generality. Imagine for example that the $(\otimes\checkmark)$ parameter is actually $\lambda r\ s.\ \lfloor(r,\ s)\rfloor$: we easily figure out that in this case the corresponding law should be something like $P \ [\![A]\!]_{\checkmark Pair} \ Q = \mathit{TickSwap} \ (Q \ [\![A]\!]_{\checkmark Pair} \ P)$. More generally, in the **locale**, when writing $P \ [\![A]\!]_{\checkmark} \ Q$, $P$ is of type $('a,\ 'r)\ \mathit{process}_{ptick}$ while $Q$ is of type $('a,\ 's)\ \mathit{process}_{ptick}$ so we want to find an abstract setup in which we can establish a quasi-commutativity. This is done in the next subsection.

### 6.1.2 Formalization

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

There is an obvious symmetry over the variables.

**sublocale** $Sync_{ptick}$-*comm-locale-sym* :
  $Sync_{ptick}$-*comm-locale* ‹$(\otimes\checkmark_{rev})$› ‹$(\otimes\checkmark)$› ‹$\otimes\checkmark_{rev} \Rightarrow \otimes\checkmark$› ‹$\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$›
**proof** *unfold-locales*
  **show** ‹$s \otimes\checkmark_{rev} r = \lfloor s\text{-}r \rfloor \Longrightarrow s' \otimes\checkmark_{rev} r' = \lfloor s\text{-}r \rfloor$
      $\Longrightarrow s' = s \wedge r' = r$› **for** *s r s-r s' r'*
    **using** *inj-tick-join tick-join-rev-Some-imp* **by** *blast*
**next**
  **show** ‹$s \otimes\checkmark_{rev} r = \Diamond \longleftrightarrow r \otimes\checkmark s = \Diamond$› **for** *s r*
    **by** (*simp add*: *tick-join-None-iff*)
**next**
  **show** ‹$s \otimes\checkmark_{rev} r = \lfloor s\text{-}r \rfloor \Longrightarrow r \otimes\checkmark s = \lfloor \otimes\checkmark_{rev} \Rightarrow \otimes\checkmark\ s\text{-}r \rfloor$› **for** *s r s-r*
    **by** (*simp add*: *tick-join-rev-Some-imp*)
**next**
  **show** ‹$r \otimes\checkmark s = \lfloor r\text{-}s \rfloor \Longrightarrow s \otimes\checkmark_{rev} r = \lfloor \otimes\checkmark \Rightarrow \otimes\checkmark_{rev}\ r\text{-}s \rfloor$› **for** *r s r-s*
    **by** (*simp add*: *tick-join-Some-imp*)
**qed**

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

### 6.1.3 First Properties

**lemma** *tick-join-conv-image-range-tick-join* :
  ‹$\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$ ‘ *range-tick-join* = $Sync_{ptick}$-*comm-locale-sym.range-tick-join*›
  **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*] :
  ‹*r-s* ∈ *range-tick-join* $\Longrightarrow \otimes\checkmark_{rev} \Rightarrow \otimes\checkmark\ (\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}\ r\text{-}s) = r\text{-}s$›
  **using** *tick-join-Some-imp tick-join-rev-Some-imp* **by** *fastforce*

**lemma** *inj-on-tick-join-conv* : ‹*inj-on* $\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$ *range-tick-join*›
  **by** (*rule inj-onI*, *simp*)
    (*metis option.inject tick-join-Some-imp tick-join-rev-Some-imp*)

**lemma** *bij-betw-tick-join-conv* :
  ‹*bij-betw* $\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$ *range-tick-join* $Sync_{ptick}$-*comm-locale-sym.range-tick-join*›
**proof** (*rule bij-betw-imageI*)
  **show** ‹*inj-on* $\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$ *range-tick-join*›
    **by** (*fact inj-on-tick-join-conv*)
**next**
  **show** ‹$\otimes\checkmark \Rightarrow \otimes\checkmark_{rev}$ ‘ *range-tick-join* = $Sync_{ptick}$-*comm-locale-sym.range-tick-join*›
    **using** *tick-join-conv-image-range-tick-join* **by** *blast*
**qed**

**lemma** *map-tick-join-rev-conv-map-tick-join-conv* :
‹{*r-s*. ✓(*r-s*) ∈ *set t*} ⊆ *range-tick-join* ⟹
*map* (*map-event*$_{ptick}$ *id* ⊗✓$_{rev}$⇒⊗✓) (*map* (*map-event*$_{ptick}$ *id* ⊗✓⇒⊗✓$_{rev}$) *t*)
= *t*›
**proof** (*induct t*)
**case** *Nil* **show** *?case* **by** *simp*
**next**
**let** *?f1* = ‹*map-event*$_{ptick}$ *id* ⊗✓⇒⊗✓$_{rev}$›
**let** *?f2* = ‹*map-event*$_{ptick}$ *id* ⊗✓$_{rev}$⇒⊗✓›
**case** (*Cons e t*)
**have** ‹*map ?f2* (*map ?f1* (*e* # *t*)) = *?f2* (*?f1 e*) # *map ?f2* (*map ?f1 t*)› **by** *simp*
**also have** ‹*?f2* (*?f1 e*) = *e*›
**proof** (*cases e*)
**show** ‹*e* = *ev a* ⟹ *?f2* (*?f1 e*) = *e*› **for** *a* **by** *simp*
**next**
**fix** *r-s* **assume** ‹*e* = ✓(*r-s*)›
**with** *Cons.prems* **have** ‹*r-s* ∈ *range-tick-join*› **by** *auto*
**with** ‹*e* = ✓(*r-s*)› *inj-on-tick-join-conv*
**show** ‹*?f2* (*?f1 e*) = *e*› **by** *simp*
**qed**
**also have** ‹*map ?f2* (*map ?f1 t*) = *t*›
**by** (*rule Cons.hyps*) (*use Cons.prems* **in** *auto*)
**finally show** ‹*map ?f2* (*map ?f1* (*e* # *t*)) = *e* # *t*› **.**
**qed**

**end**

### 6.1.4 Commutativity

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

**lemma** *setinterleaves*$_{ptick}$*-imp-setinterleaves*$_{ptick}$*-rev* :
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*u*, *v*), *A*) ⟹
*map* (*map-event*$_{ptick}$ *id* ⊗✓⇒⊗✓$_{rev}$) *t*
*setinterleaves*$_{✓(⊗✓_{rev})}$ ((*v*, *u*), *A*)›
— Finally not used, and probably obtainable as a corollary of *t setinterleaves*$_{✓(⊗✓)}$
((*u*, *v*), *A*) ⟹ *map* (*map-event*$_{ptick}$ *id* ⊗✓⇒⊗✓$_{rev}$) *t setinterleaves*$_{✓}$*λr s. case r* ⊗✓ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊⊗✓⇒⊗✓
((*u*, *v*), *A*)
**proof** (*induct* ‹((⊗✓), *u*, *A*, *v*)› *arbitrary*: *t u v*)
**case** (*tick-setinterleaving*$_{ptick}$*-tick r u s v*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems*
**obtain** *r-s t*′
**where** ∗ : ‹*r* ⊗✓ *s* = ⌊*r-s*⌋› ‹*t* = ✓(*r-s*) # *t*′›
‹*t*′ *setinterleaves*$_{✓(⊗✓)}$ ((*u*, *v*), *A*)›
**by** (*auto split*: *option.split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.hyps*[*OF* ∗(*1*), *OF* ∗(*3*)]
**have** ‹*map* (*map-event*$_{ptick}$ *id* ⊗✓⇒⊗✓$_{rev}$) *t*′
*setinterleaves*$_{✓(⊗✓_{rev})}$ ((*v*, *u*), *A*)› **.**

**moreover from** *tick-join-Some-imp*[*OF* ∗(*1*)]
**have** ‹$s \otimes\checkmark_{rev}\ r = \lfloor\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}\ r\text{-}s\rfloor$› **.**
**ultimately show** ‹*map* (*map-event*$_{ptick}$ *id* $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$) *t*
*setinterleaves*$_{\checkmark(\otimes\checkmark_{rev})}$ ((✓(*s*) # *v*, ✓(*r*) # *u*), *A*)›
**by** (*simp add*: ∗(*1*, *2*))
**qed** *auto*

**lemma** *vimage-tick-join-rev-conv-subset-super-ref-Sync*$_{ptick}$*-iff* :
‹*map-event*$_{ptick}$ *id* $\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ −‘ *X* ⊆ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark_{rev}$) *X-Q A X-P*
⟷ *X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P A X-Q*›
(**is** ‹*?lhs1* ⊆ *?lhs2* ⟷ *X* ⊆ *?rhs*›)
— Same: finally not used, and probably obtainable as a corollary of (*map-event*$_{ptick}$ *id* $\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ −‘ *?X* ⊆ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark_{rev}$) *?X-P ?A ?X-Q*) = (*?X* ⊆ *super-ref-Sync*$_{ptick}$ (λ*r s*. *case r* $\otimes\checkmark_{rev}$ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊$\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ *r-s*⌋) *?X-P ?A ?X-Q*).
**proof** −
**have** ∗ : ‹(λ*r s*. *case r* $\otimes\checkmark_{rev}$ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊$\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ *r-s*⌋) =
(λ*s r*. *r* ⊗✓ *s*)›
**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** ‹*inj-on* $\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ *Sync*$_{ptick}$*-comm-locale-sym.range-tick-join*›
**by** (*fact Sync*$_{ptick}$*-comm-locale-sym.inj-on-tick-join-conv*)
**next**
**show** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$
(λ*r s*. *case r* $\otimes\checkmark_{rev}$ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊$\otimes\checkmark_{rev}\Rightarrow\otimes\checkmark$ *r-s*⌋) *X-Q A X-P*
⟷ *X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P A X-Q*›
**using** *super-ref-Sync*$_{ptick}$*-sym* **by** (*simp add*: ∗) *blast*
**qed**
**qed**

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

**theorem** *Sync*$_{ptick}$*-commute* :
‹*RenamingTick* (*P* ⟦*A*⟧$_{\checkmark}$ *Q*) $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ = *Q* ⟦*A*⟧$_{\checkmark rev}$ *P*›
**proof** −
**from** *inj-on-RenamingTick-Sync*$_{ptick}$[*OF inj-on-tick-join-conv*]
**have** ‹*RenamingTick* (*P* ⟦*A*⟧$_{\checkmark}$ *Q*) $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ =
*Sync*$_{ptick}$*-locale.Sync*$_{ptick}$
(λ*r s*. *case r* ⊗✓ *s of* ◊ ⇒ ◊ | ⌊*r-s*⌋ ⇒ ⌊$\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ *r-s*⌋) *P A Q*›
(**is** ‹- = *Sync*$_{ptick}$*-locale.Sync*$_{ptick}$ *?tick-join*′ *P A Q*›) **.**
**also have** ‹*?tick-join*′ = (λ*r s*. *s* $\otimes\checkmark_{rev}$ *r*)›
**by** (*intro ext*)
(*simp add*: *Sync*$_{ptick}$*-comm-locale-sym.tick-join-rev-Some-imp*
*tick-join-None-iff split*: *option.split*)
**finally show** ‹*RenamingTick* (*P* ⟦*A*⟧$_{\checkmark}$ *Q*) $\otimes\checkmark\Rightarrow\otimes\checkmark_{rev}$ = *Q* ⟦*A*⟧$_{\checkmark rev}$ *P*›

**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: $P \ [\![A]\!] \ (Q \ [\![A]\!] \ R) = P \ [\![A]\!] \ Q \ [\![A]\!] \ R$ but in our generalization such a law cannot be obtained in all generality. We already encountered a similar issue for the commutativity: we have to find a setup in which the different combinations of the ticks that we need make sense, and prove the quasi-associativity.

### 6.2.2 Formalization

**locale** $Sync_{ptick}$*-assoc-locale* =
  $Sync_{ptick1}$ : $Sync_{ptick}$*-locale* ‹$(\otimes\checkmark 1)$› +
  $Sync_{ptick2}$ : $Sync_{ptick}$*-locale* ‹$(\otimes\checkmark 2)$› +
  $Sync_{ptick3}$ : $Sync_{ptick}$*-locale* ‹$(\otimes\checkmark 3)$› +
  $Sync_{ptick4}$ : $Sync_{ptick}$*-locale* ‹$(\otimes\checkmark 4)$›
  **for** *tick-join1* :: ‹$'r \Rightarrow 's \Rightarrow 't$ *option*› (**infixl** ‹$\otimes\checkmark 1$› *100*)
    **and** *tick-join2* :: ‹$'t \Rightarrow 'u \Rightarrow 'v$ *option*› (**infixl** ‹$\otimes\checkmark 2$› *100*)
    **and** *tick-join3* :: ‹$'r \Rightarrow 'w \Rightarrow 'x$ *option*› (**infixl** ‹$\otimes\checkmark 3$› *100*)
    **and** *tick-join4* :: ‹$'s \Rightarrow 'u \Rightarrow 'w$ *option*› (**infixl** ‹$\otimes\checkmark 4$› *100*) +
  **fixes** *tick-assoc-ren* :: ‹$'v \Rightarrow 'x$› (‹$\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$›)
    **and** *tick-assoc-ren-conv* :: ‹$'x \Rightarrow 'v$› (‹$\otimes\checkmark 3 \Rightarrow \otimes\checkmark 2$›)
  **assumes** *None-assms-tick-join* :
    ‹$r \otimes\checkmark 1\ s = \Diamond \Longrightarrow s \otimes\checkmark 4\ u = \Diamond \vee r \otimes\checkmark 3\ \lceil s \otimes\checkmark 4\ u\rceil = \Diamond$›
    ‹$r \otimes\checkmark 1\ s \neq \Diamond \Longrightarrow \lceil r \otimes\checkmark 1\ s\rceil \otimes\checkmark 2\ u = \Diamond \Longrightarrow s \otimes\checkmark 4\ u = \Diamond \vee r \otimes\checkmark 3\ \lceil s \otimes\checkmark 4\ u\rceil = \Diamond$›
    ‹$s \otimes\checkmark 4\ u = \Diamond \Longrightarrow r \otimes\checkmark 1\ s = \Diamond \vee \lceil r \otimes\checkmark 1\ s\rceil \otimes\checkmark 2\ u = \Diamond$›
    ‹$s \otimes\checkmark 4\ u \neq \Diamond \Longrightarrow r \otimes\checkmark 3\ \lceil s \otimes\checkmark 4\ u\rceil = \Diamond \Longrightarrow r \otimes\checkmark 1\ s = \Diamond \vee \lceil r \otimes\checkmark 1\ s\rceil \otimes\checkmark 2\ u = \Diamond$›
    **and** *tick-assoc-ren-hyp* :
    ‹$r \otimes\checkmark 1\ s = \lfloor t\rfloor \Longrightarrow t \otimes\checkmark 2\ u = \lfloor v\rfloor \Longrightarrow \lceil r \otimes\checkmark 3\ \lceil s \otimes\checkmark 4\ u\rceil\rceil = \otimes\checkmark 2 \Rightarrow \otimes\checkmark 3\ v$›
    **and** *tick-assoc-ren-conv-hyp* :
    ‹$s \otimes\checkmark 4\ u = \lfloor w\rfloor \Longrightarrow r \otimes\checkmark 3\ w = \lfloor x\rfloor \Longrightarrow \lceil\lceil r \otimes\checkmark 1\ s\rceil \otimes\checkmark 2\ u\rceil = \otimes\checkmark 3 \Rightarrow \otimes\checkmark 2\ x$›
**begin**

There is a symmetry over the variables.

**sublocale** $Sync_{ptick}$*-assoc-locale-sym* :
  $Sync_{ptick}$*-assoc-locale* ‹$\lambda u\ s.\ s \otimes\checkmark 4\ u$› ‹$\lambda w\ r.\ r \otimes\checkmark 3\ w$› ‹$\lambda u\ t.\ t \otimes\checkmark 2\ u$›

‹λs r. r ⊗✓1 s› ‹⊗✓3⇒⊗✓2› ‹⊗✓2⇒⊗✓3›
**by** *unfold-locales*
(*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* :
‹∃ r s u w. s ⊗✓4 u = ⌊w⌋ ∧ r ⊗✓3 w = ⌊x⌋ ⟹
⊗✓2⇒⊗✓3 (⊗✓3⇒⊗✓2 x) = x›
**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* :
‹∃ r s t u. r ⊗✓1 s = ⌊t⌋ ∧ t ⊗✓2 u = ⌊v⌋ ⟹ ⊗✓3⇒⊗✓2 (⊗✓2⇒⊗✓3 v) = v›
**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* :
‹*inj-on* ⊗✓2⇒⊗✓3 {v. ∃ r s t u. r ⊗✓1 s = ⌊t⌋ ∧ t ⊗✓2 u = ⌊v⌋}›
**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* :
‹*inj-on* ⊗✓3⇒⊗✓2 {x. ∃ r s u w. s ⊗✓4 u = ⌊w⌋ ∧ r ⊗✓3 w = ⌊x⌋}›
**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* :
‹⟦$t_t$ $setinterleaves_{✓(⊗✓1)}$ (($t_r$, $t_s$), A);
$t_v$ $setinterleaves_{✓(⊗✓2)}$ (($t_t$, $t_u$), A)⟧ ⟹
∃ $t_w$. *map* (*map-event*$_{ptick}$ *id* ⊗✓2⇒⊗✓3) $t_v$ $setinterleaves_{✓(⊗✓3)}$ (($t_r$, $t_w$), A) ∧
$t_w$ $setinterleaves_{✓(⊗✓4)}$ (($t_s$, $t_u$), A)›
**proof** −
**let** *?map* = ‹λt. *map ev* (*map of-ev t*)›
**let** *?map-event* = ‹λt. *map* (*map-event*$_{ptick}$ *id* ⊗✓2⇒⊗✓3) t›
**show** ‹⟦$t_t$ $setinterleaves_{✓(⊗✓1)}$ (($t_r$, $t_s$), A);
$t_v$ $setinterleaves_{✓(⊗✓2)}$ (($t_t$, $t_u$), A)⟧ ⟹ *?thesis*›
**proof** (*induct* ‹((⊗✓2), $t_t$, A, $t_u$)› *arbitrary*: $t_r$ $t_s$ $t_t$ $t_u$ $t_v$)
**case** $Nil\text{-}setinterleaving_{ptick}$-*Nil*
**thus** *?case*
**by** (*cases* $t_r$; *cases* $t_s$)
(*auto intro*: $Nil\text{-}setinterleaves_{ptick}$ *simp add*: $setinterleaving_{ptick}$-*simps*
*split*: *if-split-asm* $event_{ptick}$.*split-asm option.split-asm*)
**next**
**case** ($ev\text{-}setinterleaving_{ptick}$-*Nil a* $t_t$)
**from** $ev\text{-}setinterleaving_{ptick}$-*Nil.prems*(2)

**have** ‹$a \notin A$› ‹$tF\ t_t$› ‹$set\ t_t \cap ev\ `\ A = \{\}$› ‹$t_v = ?map\ (ev\ a\ \#\ t_t)$›
**and** $ : ‹$?map\ t_t\ setinterleaves_{✓(\otimes✓2)}\ ((t_t, []), A)$›
**by** (*auto simp add*: $setinterleaves_{ptick}$*-NilR-iff split*: *if-split-asm*)
**from** $ev\text{-}setinterleaving_{ptick}$*-Nil.prems*(*1*)
**consider** (*mv-left*) $t_r'$ **where** ‹$t_r = ev\ a\ \#\ t_r'$› ‹$t_t\ setinterleaves_{✓(\otimes✓1)}\ ((t_r', t_s), A)$›
| (*mv-right*) $t_s'$ **where** ‹$t_s = ev\ a\ \#\ t_s'$› ‹$t_t\ setinterleaves_{✓(\otimes✓1)}\ ((t_r, t_s'), A)$›
**by** (*auto simp add*: ‹$a \notin A$› *elim*: $Cons\text{-}ev\text{-}setinterleaves_{ptick}E$)
**thus** *?case*
**proof** *cases*
**case** *mv-left*
**from** $ev\text{-}setinterleaving_{ptick}$*-Nil.hyps*[*OF* ‹$a \notin A$› *mv-left*(*2*) $]
**obtain** $t_w$ **where** $*$ : ‹$?map\text{-}event\ (?map\ t_t)\ setinterleaves_{✓(\otimes✓3)}((t_r', t_w), A)$›
‹$t_w\ setinterleaves_{✓(\otimes✓4)}\ ((t_s, []), A)$› **by** *blast*
**from** $*(1)$ **have** ‹$?map\text{-}event\ t_v\ setinterleaves_{✓(\otimes✓3)}\ ((t_r, t_w), A)$›
**by** (*cases* $t_w$, *auto simp add*: *mv-left*(*1*) ‹$a \notin A$› ‹$t_v = ?map\ (ev\ a\ \#\ t_t)$›
$setinterleaving_{ptick}$*-simps split*: $event_{ptick}$*.split*)
**with** $*(2)$ **show** *?thesis* **by** *blast*
**next**
**case** *mv-right*
**from** $ev\text{-}setinterleaving_{ptick}$*-Nil.hyps*[*OF* ‹$a \notin A$› *mv-right*(*2*) $]
**obtain** $t_w$ **where** $*$ : ‹$?map\text{-}event\ (?map\ t_t)\ setinterleaves_{✓(\otimes✓3)}\ ((t_r, t_w), A)$›
‹$t_w\ setinterleaves_{✓(\otimes✓4)}\ ((t_s', []), A)$› **by** *blast*
**from** $*(2)$ **have** ‹$ev\ a\ \#\ t_w\ setinterleaves_{✓(\otimes✓4)}\ ((t_s, []), A)$›
**by** (*simp add*: ‹$a \notin A$› ‹$t_v = ?map\ (ev\ a\ \#\ t_t)$› *mv-right*(*1*))
**moreover from** $*(1)$
**have** ‹$?map\text{-}event\ t_v\ setinterleaves_{✓(\otimes✓3)}\ ((t_r, ev\ a\ \#\ t_w), A)$›
**by** (*cases* $t_r$, *auto simp add*: ‹$a \notin A$› ‹$t_v = ?map\ (ev\ a\ \#\ t_t)$› $setinterleaving_{ptick}$*-simps*
*split*: $event_{ptick}$*.split*)
**ultimately show** *?thesis* **by** *blast*
**qed**
**next**
**case** ($tick\text{-}setinterleaving_{ptick}$*-Nil r* $t_t$)
**from** $tick\text{-}setinterleaving_{ptick}$*-Nil.prems*(*2*) **have** *False* **by** *simp*
**thus** *?case* **..**
**next**
**case** ($Nil\text{-}setinterleaving_{ptick}$*-ev b* $t_u$)
**from** $Nil\text{-}setinterleaving_{ptick}$*-ev.prems*(*1*)[*THEN* $setinterleaves_{ptick}$*-imp-lengthLR-le*]
**have** ‹$t_r = []$› ‹$t_s = []$› **by** *simp-all*
**from** $Nil\text{-}setinterleaving_{ptick}$*-ev.prems*(*2*)
**have** ‹$b \notin A$› ‹$tF\ t_u$› ‹$set\ t_u \cap ev\ `\ A = \{\}$› ‹$t_v = ?map\ (ev\ b\ \#\ t_u)$›
**and** $ : ‹$?map\ t_u\ setinterleaves_{✓(\otimes✓2)}\ (([], t_u), A)$›
**by** (*auto simp add*: $setinterleaves_{ptick}$*-NilL-iff split*: *if-split-asm*)

**from** *Nil-setinterleaving$_{ptick}$-ev.hyps*[*OF* ‹$b \notin A$› *Nil-setinterleaving$_{ptick}$-ev.prems*(*1*)
$]
**obtain** $t_w$ **where** ‹*?map-event* (*?map* $t_u$) *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, $t_w$), $A$)›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, $t_u$), $A$)› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, *ev* $b$ # $t_w$), $A$) ∧
*ev* $b$ # $t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, *ev* $b$ # $t_u$), $A$)›
**by** (*simp add*: ‹$t_v$ = *?map* (*ev* $b$ # $t_u$)› ‹$t_r$ = []› ‹$t_s$ = []› ‹$b \notin A$›)
**thus** *?case* **..**
**next**
**case** (*Nil-setinterleaving$_{ptick}$-tick* $r_u$ $t_u$)
**from** *Nil-setinterleaving$_{ptick}$-tick.prems*(*2*) **have** *False* **by** *simp*
**thus** *?case* **..**
**next**
**case** (*ev-setinterleaving$_{ptick}$-ev* $a$ $t_t$ $b$ $t_u$)
**from** *ev-setinterleaving$_{ptick}$-ev.prems*(*2*)
**consider** (*mv-both*) $t_v{'}$ **where** ‹$a \in A$› ‹$b \in A$› ‹$a = b$› ‹$t_v$ = *ev* $b$ # $t_v{'}$› ‹$t_v{'}$ *setinterleaves*$_{✓(\otimes✓2)}$ (($t_t$, $t_u$), $A$)›
| (*mvR-inL*) $t_v{'}$ **where** ‹$a \in A$› ‹$b \notin A$› ‹$t_v$ = *ev* $b$ # $t_v{'}$› ‹$t_v{'}$ *setinterleaves*$_{✓(\otimes✓2)}$ ((*ev* $a$ # $t_t$, $t_u$), $A$)›
| (*mvL-inR*) $t_v{'}$ **where** ‹$a \notin A$› ‹$b \in A$› ‹$t_v$ = *ev* $a$ # $t_v{'}$› ‹$t_v{'}$ *setinterleaves*$_{✓(\otimes✓2)}$ (($t_t$, *ev* $b$ # $t_u$), $A$)›
| (*mvR-notin*) $t_v{'}$ **where** ‹$a \notin A$› ‹$b \notin A$› ‹$t_v$ = *ev* $b$ # $t_v{'}$› ‹$t_v{'}$ *setinterleaves*$_{✓(\otimes✓2)}$ ((*ev* $a$ # $t_t$, $t_u$), $A$)›
| (*mvL-notin*) $t_v{'}$ **where** ‹$a \notin A$› ‹$b \notin A$› ‹$t_v$ = *ev* $a$ # $t_v{'}$› ‹$t_v{'}$ *setinterleaves*$_{✓(\otimes✓2)}$ (($t_t$, *ev* $b$ # $t_u$), $A$)›
**by** (*auto split*: *if-split-asm*)
**thus** *?case*
**proof** *cases*
**case** *mv-both*
**from** *ev-setinterleaving$_{ptick}$-ev.prems*(*1*)
**obtain** $t_r{'}$ $t_s{'}$ **where** ‹$t_r$ = *ev* $a$ # $t_r{'}$› ‹$t_s$ = *ev* $a$ # $t_s{'}$›
‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r{'}$, $t_s{'}$), $A$)›
**by** (*auto simp add*: ‹$a \in A$› *elim*: *Cons-ev-setinterleaves$_{ptick}$E*)
**from** *ev-setinterleaving$_{ptick}$-ev.hyps*(*1*)[*OF mv-both*(*1−3*) ‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r{'}$, $t_s{'}$), $A$)› *mv-both*(*5*)]
**obtain** $t_w$ **where** ‹*?map-event* $t_v{'}$ *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r{'}$, $t_w$), $A$)›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s{'}$, $t_u$), $A$)› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, *ev* $a$ # $t_w$), $A$) ∧
*ev* $a$ # $t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, *ev* $b$ # $t_u$), $A$)›
**by** (*simp add*: *mv-both*(*2−4*) ‹$t_s$ = *ev* $a$ # $t_s{'}$› ‹$t_r$ = *ev* $a$ # $t_r{'}$›)
**thus** *?thesis* **..**
**next**
**case** *mvR-inL*
**from** *ev-setinterleaving$_{ptick}$-ev.prems*(*1*)
**obtain** $t_r{'}$ $t_s{'}$ **where** ‹$t_r$ = *ev* $a$ # $t_r{'}$› ‹$t_s$ = *ev* $a$ # $t_s{'}$›
‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r{'}$, $t_s{'}$), $A$)›

**by** (*auto simp add*: ‹$a \in A$› *elim*: *Cons-ev-setinterleaves*$_{ptick}E$)
**from** *ev-setinterleaving*$_{ptick}$*-ev.hyps*(*2*)[*OF mvR-inL*(*1*, *2*) *ev-setinterleaving*$_{ptick}$*-ev.prems*(*1*) *mvR-inL*(*4*)]
**obtain** $t_w$ **where** ‹*?map-event* $t_v{}'$ *setinterleaves*$_{✓(\otimes✓3)}$ $((t_r, t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ $((t_s, t_u), A)$› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(\otimes✓3)}$ $((t_r, ev\ b\ \#\ t_w), A) \wedge$
*ev b* # $t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ $((t_s, ev\ b\ \#\ t_u), A)$›
**by** (*simp add*: ‹$t_r = ev\ a\ \#\ t_r{}'$› ‹$t_s = ev\ a\ \#\ t_s{}'$› *mvR-inL*(*1−3*))
**thus** *?thesis* **..**
**next**
**case** *mvL-inR*
**from** *ev-setinterleaving*$_{ptick}$*-ev.prems*(*1*)
**consider** (*mv-left*) $t_r{}'$ **where** ‹$t_r = ev\ a\ \#\ t_r{}'$› ‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ $((t_r{}', t_s), A)$›
| (*mv-right*) $t_s{}'$ **where** ‹$t_s = ev\ a\ \#\ t_s{}'$› ‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ $((t_r, t_s{}'), A)$›
**by** (*auto simp add*: ‹$a \notin A$› *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** ‹*?map-event* $t_v{}'$ *setinterleaves*$_{✓(\otimes✓3)}$ $((t_r{}', t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ $((t_s, ev\ b\ \#\ t_u), A)$› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(\otimes✓3)}$ $((t_r, t_w), A) \wedge$
$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ $((t_s, ev\ b\ \#\ t_u), A)$›
**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** ‹*?map-event* $t_v{}'$ *setinterleaves*$_{✓(\otimes✓3)}$ $((t_r, t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ $((t_s{}', ev\ b\ \#\ t_u), A)$› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(\otimes✓3)}$ $((t_r, ev\ a\ \#\ t_w), A) \wedge$
*ev a* # $t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ $((t_s, ev\ b\ \#\ t_u), A)$›
**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.prems*(*1*)
**consider** (*mv-left*) $t_r{}'$ **where** ‹$t_r = ev\ a\ \#\ t_r{}'$› ‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ $((t_r{}', t_s), A)$›
| (*mv-right*) $t_s{}'$ **where** ‹$t_s = ev\ a\ \#\ t_s{}'$› ‹$t_t$ *setinterleaves*$_{✓(\otimes✓1)}$ $((t_r, t_s{}'), A)$›

$A$)›
**by** (*auto simp add*: ‹$a \notin A$› *elim*: *Cons-ev-setinterleaves*$_{ptick}$ *E*)
**thus** *?thesis*
**proof** *cases*
**case** *mv-left*
**from** *mv-left*(*2*) **have** ‹*ev a # t*$_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r$, $t_s$), $A$)›
**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** ‹*?map-event t*$_v$*'* *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, $t_w$), $A$)›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, $t_u$), $A$)› **by** *blast*
**hence** ‹*?map-event t*$_v$ *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, *ev b # t*$_w$), $A$) ∧
*ev b # t*$_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, *ev b # t*$_u$), $A$)›
**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** ‹*ev a # t*$_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r$, $t_s$), $A$)›
**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** ‹*?map-event t*$_v$*'* *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, $t_w$), $A$)›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, $t_u$), $A$)› **by** *blast*
**hence** ‹*?map-event t*$_v$ *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, *ev b # t*$_w$), $A$) ∧
*ev b # t*$_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, *ev b # t*$_u$), $A$)›
**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.prems*(*1*)
**consider** (*mv-left*) $t_r'$ **where** ‹$t_r$ = *ev a # t*$_r$*'*› ‹*t*$_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r'$, $t_s$), $A$)›
| (*mv-right*) $t_s'$ **where** ‹$t_s$ = *ev a # t*$_s$*'*› ‹*t*$_t$ *setinterleaves*$_{✓(\otimes✓1)}$ (($t_r$, $t_s'$), $A$)›
**by** (*auto simp add*: ‹$a \notin A$› *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** ‹*?map-event t*$_v$*'* *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r'$, $t_w$), $A$)›
‹$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, *ev b # t*$_u$), $A$)› **by** *blast*
**hence** ‹*?map-event t*$_v$ *setinterleaves*$_{✓(\otimes✓3)}$ (($t_r$, $t_w$), $A$) ∧
$t_w$ *setinterleaves*$_{✓(\otimes✓4)}$ (($t_s$, *ev b # t*$_u$), $A$)›

**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** ‹*?map-event* $t_v{}'$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s{}', ev\ b\ \#\ t_u), A)$› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, ev\ a\ \#\ t_w), A) \wedge$
$ev\ a\ \#\ t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s, ev\ b\ \#\ t_u), A)$›
**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.prems*(*2*) **obtain** $t_v{}'$
**where** ‹$a \notin A$› ‹$t_v = ev\ a\ \#\ t_v{}'$›
**and** $ : ‹$t_v{}'$ *setinterleaves*$_{✓(⊗✓2)}$ $((t_t, ✓(s)\ \#\ t_u), A)$›
**by** (*auto split*: *if-split-asm*)
**from** *ev-setinterleaving*$_{ptick}$-*tick.prems*(*1*)
**have** ‹$t_r \neq [] \wedge hd\ t_r = ev\ a \wedge t_t$ *setinterleaves*$_{✓(⊗✓1)}$ $((tl\ t_r, t_s), A) \vee$
$t_s \neq [] \wedge hd\ t_s = ev\ a \wedge t_t$ *setinterleaves*$_{✓(⊗✓1)}$ $((t_r, tl\ t_s), A)$›
**by** (*auto simp add*: ‹$a \notin A$› *elim*: *Cons-ev-setinterleaves*$_{ptick}$*E*)
**thus** *?case*
**proof** (*elim disjE conjE*)
**assume** ‹$t_r \neq []$› ‹$hd\ t_r = ev\ a$› ‹$t_t$ *setinterleaves*$_{✓(⊗✓1)}$ $((tl\ t_r, t_s), A)$›
**from** *ev-setinterleaving*$_{ptick}$-*tick.hyps*[*OF* ‹$a \notin A$› *this*(*3*) $]
**obtain** $t_w$ **where** $ : ‹*?map-event* $t_v{}'$ *setinterleaves*$_{✓(⊗✓3)}$$((tl\ t_r, t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s, ✓(s)\ \#\ t_u), A)$› **by** *blast*
**from** $(*1*)$ **have** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, t_w), A)$›
**by** (*subst list.collapse*[*OF* ‹$t_r \neq []$›, *symmetric*])
(*cases* $t_w$, *auto simp add*: ‹$a \notin A$› ‹$hd\ t_r = ev\ a$› ‹$t_v = ev\ a\ \#\ t_v{}'$›
*setinterleaving*$_{ptick}$-*simps split*: *event*$_{ptick}$.*split*)
**with** $(*2*)$ **show** *?case* **by** *blast*
**next**
**assume** ‹$t_s \neq []$› ‹$hd\ t_s = ev\ a$› ‹$t_t$ *setinterleaves*$_{✓(⊗✓1)}$ $((t_r, tl\ t_s), A)$›
**from** *ev-setinterleaving*$_{ptick}$-*tick.hyps*[*OF* ‹$a \notin A$› *this*(*3*) $]
**obtain** $t_w$ **where** $ : ‹*?map-event* $t_v{}'$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((tl\ t_s, ✓(s)\ \#\ t_u), A)$› **by** *blast*
**from** $(*2*)$ **have** ‹$ev\ a\ \#\ t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s, ✓(s)\ \#\ t_u), A)$›
**by** (*subst list.collapse*[*OF* ‹$t_s \neq []$›, *symmetric*])
(*simp add*: ‹$a \notin A$› ‹$t_v = ev\ a\ \#\ t_v{}'$› ‹$hd\ t_s = ev\ a$›)
**moreover from** $(*1*)$
**have** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, ev\ a\ \#\ t_w), A)$›

**by** (*cases* $t_r$, *auto simp add*: ‹$a \notin A$› ‹$t_v = ev\ a\ \#\ t_v'$› *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.prems*(*1*)
**obtain** $r_r\ r_s\ t_r'\ t_s'$ **where** ‹$(r_r \otimes\checkmark 1\ r_s) = \lfloor r_t \rfloor$› ‹$t_r = \checkmark(r_r)\ \#\ t_r'$› ‹$t_s = \checkmark(r_s)\ \#\ t_s'$›
**by** (*auto elim*: *Cons-tick-setinterleaves*$_{ptick}E$)
**from** *tick-setinterleaving*$_{ptick}$*-ev.prems*(*2*) **obtain** $t_v'$
**where** ‹$b \notin A$› ‹$t_v = ev\ b\ \#\ t_v'$›
**and** \$ : ‹$t_v'$ *setinterleaves*$_{\checkmark(\otimes\checkmark 2)}$ $((\checkmark(t_t)\ \#\ t_t,\ t_u),\ A)$›
**by** (*auto split*: *if-split-asm*)
**from** *tick-setinterleaving*$_{ptick}$*-ev.hyps*[*OF* ‹$b \notin A$› *tick-setinterleaving*$_{ptick}$*-ev.prems*(*1*) \$]
**obtain** $t_w$ **where** ‹*?map-event* $t_v'$ *setinterleaves*$_{\checkmark(\otimes\checkmark 3)}$ $((t_r,\ t_w),\ A)$›
‹$t_w$ *setinterleaves*$_{\checkmark(\otimes\checkmark 4)}$ $((t_s,\ t_u),\ A)$› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{\checkmark(\otimes\checkmark 3)}$ $((t_r,\ ev\ b\ \#\ t_w),\ A)\ \wedge$
$ev\ b\ \#\ t_w$ *setinterleaves*$_{\checkmark(\otimes\checkmark 4)}$ $((t_s,\ ev\ b\ \#\ t_u),\ A)$›
**by** (*simp add*: ‹$b \notin A$› ‹$t_v = ev\ b\ \#\ t_v'$› ‹$t_r = \checkmark(r_r)\ \#\ t_r'$› ‹$t_s = \checkmark(r_s)\ \#\ t_s'$›)
**thus** *?case* **..**
**next**
**case** (*tick-setinterleaving*$_{ptick}$*-tick* $r_t\ t_t\ r_u\ t_u$)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems*(*1*)
**obtain** $r_r\ r_s\ t_r'\ t_s'$ **where** ‹$r_r \otimes\checkmark 1\ r_s = \lfloor r_t \rfloor$› ‹$t_r = \checkmark(r_r)\ \#\ t_r'$› ‹$t_s = \checkmark(r_s)\ \#\ t_s'$›
‹$t_t$ *setinterleaves*$_{\checkmark(\otimes\checkmark 1)}$ $((t_r',\ t_s'),\ A)$›
**by** (*auto elim*: *Cons-tick-setinterleaves*$_{ptick}E$)
**from** *tick-setinterleaving*$_{ptick}$*-tick.prems*(*2*)
**obtain** $r_v\ t_v'$ **where** ‹$r_t \otimes\checkmark 2\ r_u = \lfloor r_v \rfloor$› ‹$t_v = \checkmark(r_v)\ \#\ t_v'$›
‹$t_v'$ *setinterleaves*$_{\checkmark(\otimes\checkmark 2)}$ $((t_t,\ t_u),\ A)$›
**by** (*auto split*: *option.split-asm*)
**from** ‹$r_r \otimes\checkmark 1\ r_s = \lfloor r_t \rfloor$› ‹$r_t \otimes\checkmark 2\ r_u = \lfloor r_v \rfloor$› **obtain** $r_w$ **where** ‹$r_s \otimes\checkmark 4\ r_u = \lfloor r_w \rfloor$›
**by** (*metis None-assms-tick-join*(*3*) *not-None-eq option.sel*)
**from** ‹$r_s \otimes\checkmark 4\ r_u = \lfloor r_w \rfloor$› ‹$r_r \otimes\checkmark 1\ r_s = \lfloor r_t \rfloor$› ‹$r_t \otimes\checkmark 2\ r_u = \lfloor r_v \rfloor$›
**obtain** $r_x$ **where** ‹$r_r \otimes\checkmark 3\ r_w = \lfloor r_x \rfloor$›
**by** (*metis None-assms-tick-join*(*4*) *option.distinct*(*1*) *option.exhaust-sel option.sel*)
**have** ‹$\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3\ r_v = r_x$›
**by** (*metis* ‹$r_r \otimes\checkmark 1\ r_s = \lfloor r_t \rfloor$› ‹$r_r \otimes\checkmark 3\ r_w = \lfloor r_x \rfloor$› ‹$r_s \otimes\checkmark 4\ r_u = \lfloor r_w \rfloor$›
‹$r_t \otimes\checkmark 2\ r_u = \lfloor r_v \rfloor$› *tick-assoc-ren-hyp option.sel*)
**from** *tick-setinterleaving*$_{ptick}$*-tick.hyps*
[*OF* ‹$r_t \otimes\checkmark 2\ r_u = \lfloor r_v \rfloor$› ‹$t_t$ *setinterleaves*$_{\checkmark(\otimes\checkmark 1)}$ $((t_r',\ t_s'),\ A)$›
‹$t_v'$ *setinterleaves*$_{\checkmark(\otimes\checkmark 2)}$ $((t_t,\ t_u),\ A)$›]

**obtain** $t_w$ **where** ‹*?map-event* $t_v'$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r', t_w), A)$›
‹$t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s', t_u), A)$› **by** *blast*
**hence** ‹*?map-event* $t_v$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, ✓(r_w) \# t_w), A) \wedge$
$✓(r_w) \# t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s, ✓(r_u) \# t_u), A)$›
**by** (*simp add*: ‹$t_r = ✓(r_r) \# t_r'$› ‹$t_s = ✓(r_s) \# t_s'$› ‹$t_v = ✓(r_v) \# t_v'$›
‹$r_s \otimes ✓4\ r_u = \lfloor r_w \rfloor$› ‹$r_r \otimes ✓3\ r_w = \lfloor r_x \rfloor$› ‹$\otimes ✓2 \Rightarrow \otimes ✓3\ r_v = r_x$›)
**thus** *?case* **..**
**qed**
**qed**

**lemma** (**in** *Sync*$_{ptick}$*-assoc-locale*) *setinterleaves*$_{ptick}$*-assoc-right* :
‹$t_w$ *setinterleaves*$_{✓(⊗✓4)}$ $((t_s, t_u), A) \Longrightarrow$
$t_x$ *setinterleaves*$_{✓(⊗✓3)}$ $((t_r, t_w), A) \Longrightarrow$
$\exists\, t_t.$ *map* (*map-event*$_{ptick}$ *id* $⊗✓3 \Rightarrow ⊗✓2$) $t_x$ *setinterleaves*$_{✓(⊗✓2)}$ $((t_t, t_u), A)$
$\wedge$
$t_t$ *setinterleaves*$_{✓(⊗✓1)}$ $((t_r, t_s), A)$›
**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*$_{ptick1}$.*Sync*$_{ptick}$ (‹(- $[\![$-$]\!]_{✓1}$ -)› [*70*, *0*, *71*] *70*)
**notation** *Sync*$_{ptick2}$.*Sync*$_{ptick}$ (‹(- $[\![$-$]\!]_{✓2}$ -)› [*70*, *0*, *71*] *70*)
**notation** *Sync*$_{ptick3}$.*Sync*$_{ptick}$ (‹(- $[\![$-$]\!]_{✓3}$ -)› [*70*, *0*, *71*] *70*)
**notation** *Sync*$_{ptick4}$.*Sync*$_{ptick}$ (‹(- $[\![$-$]\!]_{✓4}$ -)› [*70*, *0*, *71*] *70*)

**lemma** *Sync*$_{ptick}$*-assoc-oneside* :
‹$P\ [\![S]\!]_{✓3}\ (Q\ [\![S]\!]_{✓4}\ R) \sqsubseteq_{FD}$ *RenamingTick* $(P\ [\![S]\!]_{✓1}\ Q\ [\![S]\!]_{✓2}\ R) \otimes ✓2 \Rightarrow \otimes ✓3$›
(**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
**proof** −
**let** *?map-event* = ‹$\lambda t.$ *map* (*map-event*$_{ptick}$ *id* $\otimes ✓2 \Rightarrow \otimes ✓3$) $t$›
**let** *?map-event-conv* = ‹$\lambda t.$ *map* (*map-event*$_{ptick}$ *id* $\otimes ✓3 \Rightarrow \otimes ✓2$) $t$›
**show** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›
**proof** (*rule failure-divergence-refine-optimizedI*)
**fix** $t$ **assume** ‹$t \in \mathcal{D}$ *?rhs*›
**then obtain** $t_1\ t_2$ **where** ‹$t =$ *?map-event* $t_1$ @ $t_2$›
‹*tF* $t_1$› ‹*ftF* $t_2$› ‹$t_1 \in \mathcal{D}\ (P\ [\![S]\!]_{✓1}\ Q\ [\![S]\!]_{✓2}\ R)$›
**unfolding** *D-Renaming* **by** *blast*
**from** ‹$t_1 \in \mathcal{D}\ (P\ [\![S]\!]_{✓1}\ Q\ [\![S]\!]_{✓2}\ R)$› **obtain** $t_{11}\ t_{12}$ *t-P-Q t-R*
**where** $*$ : ‹$t_1 = t_{11}$ @ $t_{12}$› ‹*tF* $t_{11}$› ‹*ftF* $t_{12}$›
‹$t_{11}$ *setinterleaves*$_{✓(⊗✓2)}$ $((t\text{-}P\text{-}Q, t\text{-}R), S)$›
‹*t-P-Q* $\in \mathcal{D}\ (P\ [\![S]\!]_{✓1}\ Q) \wedge$ *t-R* $\in \mathcal{T}\ R \vee$

$t\text{-}P\text{-}Q \in \mathcal{T}\ (P\ [\![S]\!]_{\checkmark 1}\ Q) \wedge t\text{-}P\text{-}Q \notin \mathcal{D}\ (P\ [\![S]\!]_{\checkmark 1}\ Q) \wedge t\text{-}R \in \mathcal{D}\ R\rangle$
**unfolding** $Sync_{ptick2}.D\text{-}Sync_{ptick}$ **using** $D\text{-}T$ **by** *blast*
**from** $*(5)$ **show** $\langle t \in \mathcal{D}$ *?lhs*$\rangle$
**proof** (*elim disjE conjE*)
**assume** $\langle t\text{-}P\text{-}Q \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark 1}\ Q)\rangle$ $\langle t\text{-}R \in \mathcal{T}\ R\rangle$
**from** $\langle t\text{-}P\text{-}Q \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark 1}\ Q)\rangle$ **obtain** $t\text{-}P\text{-}Q_1\ t\text{-}P\text{-}Q_2\ t\text{-}P\ t\text{-}Q$
**where** $**$ : $\langle t\text{-}P\text{-}Q = t\text{-}P\text{-}Q_1 \mathbin{@} t\text{-}P\text{-}Q_2\rangle$ $\langle tF\ t\text{-}P\text{-}Q_1\rangle$ $\langle ftF\ t\text{-}P\text{-}Q_2\rangle$
$\langle t\text{-}P\text{-}Q_1\ setinterleaves_{\checkmark(\otimes\checkmark 1)}\ ((t\text{-}P,\ t\text{-}Q),\ S)\rangle$
$\langle t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q\rangle$
**unfolding** $Sync_{ptick1}.D\text{-}Sync_{ptick}$ **by** *blast*
**from** $*(4)[unfolded\ **(1),\ THEN\ setinterleaves_{ptick}\text{-}appendL]$
**obtain** $t_{111}\ t_{112}\ t\text{-}R_1\ t\text{-}R_2$
**where** $***$ : $\langle t_{11} = t_{111} \mathbin{@} t_{112}\rangle$ $\langle t\text{-}R = t\text{-}R_1 \mathbin{@} t\text{-}R_2\rangle$
$\langle t_{111}\ setinterleaves_{\checkmark(\otimes\checkmark 2)}\ ((t\text{-}P\text{-}Q_1,\ t\text{-}R_1),\ S)\rangle$
$\langle t_{112}\ setinterleaves_{\checkmark(\otimes\checkmark 2)}\ ((t\text{-}P\text{-}Q_2,\ t\text{-}R_2),\ S)\rangle$ **by** *blast*
**from** $setinterleaves_{ptick}\text{-}assoc\text{-}left[OF\ **(4)\ ***(3)]$
**obtain** $t\text{-}Q\text{-}R$ **where** $****$ : $\langle$*?map-event* $t_{111}\ setinterleaves_{\checkmark(\otimes\checkmark 3)}\ ((t\text{-}P,\ t\text{-}Q\text{-}R),\ S)\rangle$
$\langle t\text{-}Q\text{-}R\ setinterleaves_{\checkmark(\otimes\checkmark 4)}\ ((t\text{-}Q,\ t\text{-}R_1),\ S)\rangle$ **by** *blast*
**have** $\langle tF$ (*?map-event* $t_1)\rangle$
**by** (*simp add*: $\langle tF\ t_1\rangle$ $map\text{-}event_{ptick}\text{-}tickFree$)
**moreover have** $\langle ftF$ (*?map-event* $(t_{112} \mathbin{@} t_{12}) \mathbin{@} t_2)\rangle$
**by** (*metis* $*(1)$ $***(1)$ $\langle ftF\ t_2\rangle$ $\langle tF\ t_1\rangle$ *front-tickFree-append*
$map\text{-}event_{ptick}\text{-}tickFree$ *tickFree-append-iff*)
**moreover from** $**(5)$
**have** $\langle t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q\text{-}R \in \mathcal{T}\ (Q\ [\![S]\!]_{\checkmark 4}\ R) \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q\text{-}R \in \mathcal{D}\ (Q\ [\![S]\!]_{\checkmark 4}\ R)\rangle$
**proof** (*elim disjE conjE*)
**assume** $\langle t\text{-}P \in \mathcal{D}\ P\rangle$ $\langle t\text{-}Q \in \mathcal{T}\ Q\rangle$
**hence** $\langle t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q\text{-}R \in \mathcal{T}\ (Q\ [\![S]\!]_{\checkmark 4}\ R)\rangle$
**by** (*simp add*: $Sync_{ptick4}.T\text{-}Sync_{ptick}$)
(*metis* $****(2)$ $***(2)$ $\langle t\text{-}R \in \mathcal{T}\ R\rangle$ *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** $**(2,\ 4)$ **have** $\langle tF\ t\text{-}Q\rangle$ **by** (*simp add*: $tickFree\text{-}setinterleaves_{ptick}\text{-}iff$)
**with** $****(2)$ $setinterleaves_{ptick}\text{-}tickFree\text{-}imp$ **have** $\langle tF\ t\text{-}Q\text{-}R\rangle$ **by** *blast*
**moreover from** $***(2)$ $\langle t\text{-}R \in \mathcal{T}\ R\rangle$ *is-processT3-TR-append* **have** $\langle t\text{-}R_1 \in \mathcal{T}\ R\rangle$ **by** *blast*
**ultimately have** $\langle t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q\text{-}R \in \mathcal{D}\ (Q\ [\![S]\!]_{\checkmark 4}\ R)\rangle$
**unfolding** $Sync_{ptick4}.D\text{-}Sync_{ptick}$
**using** $****(2)$ $\langle t\text{-}P \in \mathcal{T}\ P\rangle$ $\langle t\text{-}Q \in \mathcal{D}\ Q\rangle$ *front-tickFree-Nil* **by** *blast*
**thus** *?thesis* **..**
**qed**
**ultimately show** $\langle t \in \mathcal{D}$ *?lhs*$\rangle$
**using** $****(1)$ **by** (*auto simp add*: $\langle t =$ *?map-event* $t_1 \mathbin{@} t_2\rangle$
$*(1)$ $***(1)$ $Sync_{ptick3}.D\text{-}Sync_{ptick}$)
**next**

**assume** ‹*t-P-Q* ∈ $\mathcal{T}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q*)› ‹*t-P-Q* ∉ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q*)› ‹*t-R* ∈ $\mathcal{D}$ *R*›
**from** *this*(*1*, *2*) **obtain** *t-P t-Q*
**where** ∗∗ : ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› ‹*t-P-Q setinterleaves*$_{\checkmark(\otimes\checkmark 1)}$ ((*t-P*, *t-Q*), *S*)›
**unfolding** *Sync*$_{ptick1}$.*Sync*$_{ptick}$*-projs* **by** *blast*
**from** *setinterleaves*$_{ptick}$*-assoc-left*[*OF* ∗∗(*3*) ∗(*4*)] **obtain** *t-Q′*
**where** ∗∗∗ : ‹*?map-event t*$_{11}$ *setinterleaves*$_{\checkmark(\otimes\checkmark 3)}$ ((*t-P*, *t-Q′*), *S*)›
‹*t-Q′ setinterleaves*$_{\checkmark(\otimes\checkmark 4)}$ ((*t-Q*, *t-R*), *S*)› **by** *blast*
**from** ∗(*2*) ∗∗(*2*) ∗∗∗ ‹*t-R* ∈ $\mathcal{D}$ *R*› **have** ‹*t-Q′* ∈ $\mathcal{D}$ (*Q* $[\![S]\!]_{\checkmark 4}$ *R*)›
**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** ‹*t* = *?map-event t*$_{11}$ @ (*?map-event t*$_{12}$ @ *t*$_2$)›
**by** (*simp add*: ∗(*1*) ‹*t* = *?map-event t*$_1$ @ *t*$_2$›)
**moreover have** ‹*tF* (*?map-event t*$_{11}$)›
**by** (*simp add*: ∗(*2*) *map-event*$_{ptick}$*-tickFree*)
**moreover from** ∗(*1*) ‹*ftF t*$_2$› ‹*tF t*$_1$› **have** ‹*ftF* (*?map-event t*$_{12}$ @ *t*$_2$)›
**using** *front-tickFree-append map-event*$_{ptick}$*-tickFree tickFree-append-iff* **by** *blast*
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *Sync*$_{ptick3}$.*D-Sync*$_{ptick}$ **using** ∗∗(*1*) ∗∗∗(*1*) **by** *blast*
**qed**
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› ‹$\mathcal{D}$ *?rhs* ⊆ $\mathcal{D}$ *?lhs*›
**then consider** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
| $t_1$ **where** ‹*t* = *?map-event t*$_1$› ‹*t* ∉ $\mathcal{D}$ *?rhs*›
‹(*t*$_1$, *map-event*$_{ptick}$ *id* $\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$ −‘ *X*) ∈ $\mathcal{F}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q* $[\![S]\!]_{\checkmark 2}$ *R*)›
**unfolding** *Renaming-projs* **by** *blast*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**assume** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**with** ‹$\mathcal{D}$ *?rhs* ⊆ $\mathcal{D}$ *?lhs*› **have** ‹*t* ∈ $\mathcal{D}$ *?lhs*› **by** *blast*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** (*fact is-processT8*)
**next**
**fix** $t_1$ **assume** ∗ : ‹*t* = *?map-event t*$_1$› ‹*t* ∉ $\mathcal{D}$ *?rhs*›
‹(*t*$_1$, *map-event*$_{ptick}$ *id* $\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$ −‘ *X*) ∈ $\mathcal{F}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q* $[\![S]\!]_{\checkmark 2}$ *R*)›
**from** ∗(*1*) ‹*t* ∉ $\mathcal{D}$ *?rhs*› **have** ‹*t*$_1$ ∉ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q* $[\![S]\!]_{\checkmark 2}$ *R*)›
**by** (*cases* ‹*tF t*$_1$›, *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** ∗∗ : ‹(*t-P-Q*, *X-P-Q*) ∈ $\mathcal{F}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q*)› ‹(*t-R*, *X-R*) ∈ $\mathcal{F}$ *R*›
‹*t*$_1$ *setinterleaves*$_{\checkmark(\otimes\checkmark 2)}$ ((*t-P-Q*, *t-R*), *S*)›
‹*map-event*$_{ptick}$ *id* $\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$ −‘ *X* ⊆ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 2$) *X-P-Q S X-R*›
**unfolding** *Sync*$_{ptick2}$.*Sync*$_{ptick}$*-projs* **by** *blast*
**from** ∗∗(*1*) **consider** ‹*t-P-Q* ∈ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark 1}$ *Q*)›

| (*fail*) *t-P X-P t-Q X-Q* **where** ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*t-P-Q setinterleaves*$_{✓(⊗✓1)}$ ((*t-P*, *t-Q*), *S*)›
‹*X-P-Q* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓*1*) *X-P S X-Q*›
**unfolding** *Sync*$_{ptick1}$.*Sync*$_{ptick}$*-projs* **by** *blast*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**assume** ‹*t-P-Q* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q*)›
**have** ‹$t_1$ ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q* ⟦*S*⟧$_{✓2}$ *R*)›
**proof** (*cases* ‹*tF t-P-Q*›)
**assume** ‹*tF t-P-Q*›
**with** ∗∗(*3*)[*THEN setinterleaves*$_{ptick}$*-tickFree-imp*[*rotated*]] **have** ‹*tF* $t_1$›
**by** *simp*
**with** ∗∗(*3*) ∗∗(*2*)[*THEN F-T*] ‹*t-P-Q* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q*)›
**show** ‹$t_1$ ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q* ⟦*S*⟧$_{✓2}$ *R*)›
**by** (*simp add*: *Sync*$_{ptick2}$.*D-Sync*$_{ptick}$)
(*meson front-tickFree-Nil self-append-conv*)
**next**
**assume** ‹¬ *tF t-P-Q*›
**then obtain** *t-P-Q′ r* **where** ‹*tF t-P-Q′*› ‹*t-P-Q* = *t-P-Q′* @ [✓(*r*)]›
**by** (*metis D-imp-front-tickFree* ‹*t-P-Q* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q*)› *butlast-snoc*
*front-tickFree-iff-tickFree-butlast nonTickFree-n-frontTickFree*)
**moreover from** ∗∗(*2*,*3*) ‹¬ *tF t-P-Q*› **obtain** *t-R′ s*
**where** ‹*tF t-R′*› ‹*t-R* = *t-R′* @ [✓(*s*)]›
**by** (*metis F-imp-front-tickFree butlast-snoc front-tickFree-iff-tickFree-butlast*
*nonTickFree-n-frontTickFree setinterleaves*$_{ptick}$*-tickFree-imp*)
**ultimately obtain** *r-s* $t_1$′ **where** ‹$t_1$ = $t_1$′ @ [✓(*r-s*)]›
‹$t_1$′ *setinterleaves*$_{✓(⊗✓2)}$ ((*t-P-Q′*, *t-R′*), *S*)›
**using** ∗∗(*3*) **by** (*auto elim*!: *setinterleaves*$_{ptick}$*-snoc-tick-snoc-tickE*)
**moreover have** ‹*t-P-Q′* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q*)›
**by** (*metis D-imp-front-tickFree* ‹¬ *tF t-P-Q*› ‹*t-P-Q* = *t-P-Q′* @ [✓(*r*)]›
‹*t-P-Q* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q*)› *butlast-snoc div-butlast-when-non-tickFree-iff*)
**moreover have** ‹*t-R′* ∈ $\mathcal{T}$ *R*›
**using** ∗∗(*2*) *F-T* ‹*t-R* = *t-R′* @ [✓(*s*)]› *is-processT3-TR-append* **by** *blast*
**ultimately have** ‹$t_1$′ ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q* ⟦*S*⟧$_{✓2}$ *R*)›
**by** (*simp add*: *Sync*$_{ptick2}$.*D-Sync*$_{ptick}$)
(*metis* ∗∗(*3*) *D-imp-front-tickFree* ‹*tF t-R′*› ‹*t-P-Q* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q*)›
‹*t-R* = *t-R′* @ [✓(*s*)]› *append.right-neutral butlast-snoc front-tickFree-charn*
*front-tickFree-setinterleaves*$_{ptick}$*-iff tickFree-Nil tickFree-append-iff*)
**thus** ‹$t_1$ ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q* ⟦*S*⟧$_{✓2}$ *R*)›
**by** (*simp add*: ‹$t_1$ = $t_1$′ @ [✓(*r-s*)]›)
(*metis* ∗(*3*) *F-imp-front-tickFree* ‹$t_1$ = $t_1$′ @ [✓(*r-s*)]› *butlast-snoc*
*div-butlast-when-non-tickFree-iff non-tickFree-tick tickFree-append-iff*)
**qed**
**with** ‹$t_1$ ∉ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓1}$ *Q* ⟦*S*⟧$_{✓2}$ *R*)› **show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **..**
**next**
**case** *fail*
**from** *setinterleaves*$_{ptick}$*-assoc-left*[*OF fail*(*3*) ∗∗(*3*)] **obtain** *t-Q′*
**where** ∗∗∗ : ‹*?map-event* $t_1$ *setinterleaves*$_{✓(⊗✓3)}$ ((*t-P*, *t-Q′*), *S*)›
‹*t-Q′ setinterleaves*$_{✓(⊗✓4)}$ ((*t-Q*, *t-R*), *S*)› **by** *blast*

**from** $**(2)$ $***(2)$ *fail(2)*
**have** ‹(*t-Q′*, *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*) ∈ $\mathcal{F}$ (*Q* $[\![S]\!]_{\checkmark 4}$ *R*)›
**by** (*auto simp add*: *Sync*$_{ptick4}$.*F-Sync*$_{ptick}$)
**moreover have** ‹*t setinterleaves*$_{\checkmark(\otimes\checkmark 3)}$ ((*t-P*, *t-Q′*), *S*)›
**by** (*simp add*: $*(1)$ $***(1)$)
**have** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 3$) *X-P S* (*super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*)›
**proof** (*rule subsetI*)
**fix** *e* **assume** ‹*e* ∈ *X*›
**show** ‹*e* ∈ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 3$) *X-P S* (*super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*)›
**proof** (*cases e*)
**fix** *a* **assume** ‹*e* = *ev a*›
**obtain** *a′* **where** ‹*map-event*$_{ptick}$ *id* $\otimes\checkmark 2\Rightarrow\otimes\checkmark 3$ (*ev a′*) = *ev a*› **by** *simp*
**with** ‹*e* ∈ *X*› **have** ‹*ev a′* ∈ *map-event*$_{ptick}$ *id* $\otimes\checkmark 2\Rightarrow\otimes\checkmark 3$ −‘ *X*›
**by** (*simp add*: ‹*e* = *ev a*›)
**with** $**(4)$[*THEN set-mp*, *OF this*] *fail(4)*
‹*map-event*$_{ptick}$ *id* $\otimes\checkmark 2\Rightarrow\otimes\checkmark 3$ (*ev a′*) = *ev a*›
**show** ‹*e* ∈ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 3$) *X-P S* (*super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*)›
**by** (*auto simp add*: ‹*e* = *ev a*› *subset-iff super-ref-Sync*$_{ptick}$*-def*)
**next**
**fix** *r-s-t* **assume** ‹*e* = ✓(*r-s-t*)›
**show** ‹*e* ∈ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 3$) *X-P S* (*super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*)›
**proof** (*cases* ‹∃ *r s t s-t*. *s* $\otimes\checkmark 4$ *t* = ⌊*s-t*⌋ ∧ *r* $\otimes\checkmark 3$ *s-t* = ⌊*r-s-t*⌋›)
**assume** ‹∃ *r s t s-t*. *s* $\otimes\checkmark 4$ *t* = ⌊*s-t*⌋ ∧ *r* $\otimes\checkmark 3$ *s-t* = ⌊*r-s-t*⌋›
**then obtain** *r s t s-t*
**where** $ : ‹*s* $\otimes\checkmark 4$ *t* = ⌊*s-t*⌋› ‹*r* $\otimes\checkmark 3$ *s-t* = ⌊*r-s-t*⌋› **by** *blast*
**then obtain** *r′ s′ t′ r-s′*
**where** $$ : ‹*r′* $\otimes\checkmark 1$ *s′* = ⌊*r-s′*⌋› ‹*r-s′* $\otimes\checkmark 2$ *t′* = ⌊$\otimes\checkmark 3\Rightarrow\otimes\checkmark 2$ *r-s-t*⌋›
**by** (*metis None-assms-tick-join*(*1*,*2*) *option.collapse option.discI option.sel tick-assoc-ren-conv-hyp*)
**have** ‹✓($\otimes\checkmark 3\Rightarrow\otimes\checkmark 2$ *r-s-t*) ∈ *map-event*$_{ptick}$ *id* $\otimes\checkmark 2\Rightarrow\otimes\checkmark 3$ −‘ *X*›
**by** (*metis* ‹*e* = ✓(*r-s-t*)› ‹*e* ∈ *X*› $ *event*$_{ptick}$.*simps*(*10*) *tick-assoc-ren-tick-assoc-ren-conv vimage-eq*)
**from** $**(4)$[*THEN set-mp*, *OF this*] *fail*$(4)$[*THEN set-mp*, *of* ‹✓(*r-s′*)›]
**show** ‹*e* ∈ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 3$) *X-P S* (*super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*)›
**by** (*simp add*: ‹*e* = ✓(*r-s-t*)› *subset-iff super-ref-Sync*$_{ptick}$*-def*)
(*metis* (*no-types*, *lifting*) $$ *None-assms-tick-join*(*3*,*4*) *Sync*$_{ptick2}$.*inj-tick-join option.collapse option.discI option.sel tick-assoc-ren-hyp tick-assoc-ren-tick-assoc-ren-conv*)
**next**
**assume** ‹∄ *r s t s-t*. *s* $\otimes\checkmark 4$ *t* = ⌊*s-t*⌋ ∧ *r* $\otimes\checkmark 3$ *s-t* = ⌊*r-s-t*⌋›
**thus** ‹*e* ∈ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 3$) *X-P S* (*super-ref-Sync*$_{ptick}$ ($\otimes\checkmark 4$) *X-Q S X-R*)›
**by** (*simp add*: ‹*e* = ✓(*r-s-t*)› *super-ref-Sync*$_{ptick}$*-def*) *blast*

**qed**
**qed**
**qed**
**ultimately show** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›
**using** ∗(*1*) ∗∗∗(*1*) *fail*(*1*) **by** (*auto simp add*: $Sync_{ptick3}$.*F-*$Sync_{ptick}$)
**qed**
**qed**
**qed**
**qed**

**end**

**lemma** (**in** $Sync_{ptick}$-*locale*) *strict-ticks-of-*$Sync_{ptick}$-*subset* :
‹✓**s**($P$ $[\![S]\!]_{✓}$ $Q$) ⊆ {*r-s* |*r-s r s*. $r \otimes ✓$ $s = \lfloor$*r-s*$\rfloor \wedge$
$r \in$ ✓**s**($P$) $\wedge$ $s \in$ ✓**s**($Q$)}› (**is** ‹- ⊆ *?S*›)
**proof** (*rule subsetI*, *elim strict-ticks-of-memE*)
**fix** *t r-s* **assume** ‹$t$ @ [✓(*r-s*)] ∈ $\mathcal{T}$ ($P$ $[\![S]\!]_{✓}$ $Q$)› ‹$t \notin \mathcal{D}$ ($P$ $[\![S]\!]_{✓}$ $Q$)›
**from** ‹$t \notin \mathcal{D}$ ($P$ $[\![S]\!]_{✓}$ $Q$)› **have** ‹$t$ @ [✓(*r-s*)] $\notin \mathcal{D}$ ($P$ $[\![S]\!]_{✓}$ $Q$)› **by** (*meson is-processT9*)
**with** ‹$t$ @ [✓(*r-s*)] ∈ $\mathcal{T}$ ($P$ $[\![S]\!]_{✓}$ $Q$)› **obtain** *t-P t-Q*
**where** ‹*t-P* ∈ $\mathcal{T}$ $P$› ‹*t-Q* ∈ $\mathcal{T}$ $Q$› ‹$t$ @ [✓(*r-s*)] $setinterleaves_{✓(\otimes✓)}$ ((*t-P*, *t-Q*), *S*)›
**unfolding** $Sync_{ptick}$-*projs* **by** *blast*
**from** *this*(*3*) **show** ‹*r-s* ∈ *?S*›
**proof** (*elim snoc-tick-*$setinterleaves_{ptick}$*E*)
**fix** *t-P′ t-Q′ r s*
**assume** ∗ : ‹$r \otimes ✓$ $s$ = *Some r-s*› ‹*t-P* = *t-P′* @ [✓(*r*)]› ‹*t-Q* = *t-Q′* @ [✓(*s*)]›
‹$t$ $setinterleaves_{✓(\otimes✓)}$ ((*t-P′*, *t-Q′*), *S*)›
**have** ‹*t-P′* $\notin \mathcal{D}$ $P \wedge$ *t-Q′* $\notin \mathcal{D}$ $Q$›
**proof** (*rule ccontr*)
**assume** ‹¬ (*t-P′* $\notin \mathcal{D}$ $P \wedge$ *t-Q′* $\notin \mathcal{D}$ $Q$)›
**with** ‹*t-P* ∈ $\mathcal{T}$ $P$› ‹*t-Q* ∈ $\mathcal{T}$ $Q$› **have** ‹$t \in \mathcal{D}$ ($P$ $[\![S]\!]_{✓}$ $Q$)›
**by** (*simp add*: *D-*$Sync_{ptick}$ ∗(*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** ‹$t \notin \mathcal{D}$ ($P$ $[\![S]\!]_{✓}$ $Q$)› **show** *False* **..**
**qed**
**with** ∗(*2*, *3*) ‹*t-P* ∈ $\mathcal{T}$ $P$› ‹*t-Q* ∈ $\mathcal{T}$ $Q$› **have** ‹$r \in$ ✓**s**($P$)› ‹$s \in$ ✓**s**($Q$)›
**by** (*metis is-processT9 strict-ticks-of-memI*)+
**with** ∗(*1*) **show** ‹*r-s* ∈ *?S*› **by** *blast*
**qed**
**qed**

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

**proof** (*rule FD-antisym*)
**show** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*› **by** (*fact* $Sync_{ptick}$*-assoc-oneside*)
**next**
**from** $Sync_{ptick}$*-assoc-locale-sym.*$Sync_{ptick}$*-assoc-oneside*[*of R S Q P*]
**have** ‹$Sync_{ptick2}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *R S*
($Sync_{ptick1}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *Q S P*) $\sqsubseteq_{FD}$
*RenamingTick* ($Sync_{ptick3}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$
($Sync_{ptick4}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *R S Q*) *S P*) $\otimes\checkmark 3 \Rightarrow \otimes\checkmark 2$› **.**
**also have** ‹$Sync_{ptick1}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *Q S P* = *P* $[\![S]\!]_{\checkmark 1}$ *Q*›
**by** (*simp add*: $Sync_{ptick1}$*.*$Sync_{ptick}$*-sym*)
**also have** ‹$Sync_{ptick2}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *R S P-Q* = *P-Q* $[\![S]\!]_{\checkmark 2}$ *R*›
**for** *P-Q*
**by** (*simp add*: $Sync_{ptick2}$*.*$Sync_{ptick}$*-sym*)
**also have** ‹$Sync_{ptick4}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *R S Q* = *Q* $[\![S]\!]_{\checkmark 4}$ *R*›
**by** (*simp add*: $Sync_{ptick4}$*.*$Sync_{ptick}$*-sym*)
**also have** ‹$Sync_{ptick3}$*.*$Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$ *Q-R S P* = *P* $[\![S]\!]_{\checkmark 3}$ *Q-R*›
**for** *Q-R*
**by** (*simp add*: $Sync_{ptick3}$*.*$Sync_{ptick}$*-sym*)
**finally have** ‹*P* $[\![S]\!]_{\checkmark 1}$ *Q* $[\![S]\!]_{\checkmark 2}$ *R* $\sqsubseteq_{FD}$ *RenamingTick ?lhs* $\otimes\checkmark 3 \Rightarrow \otimes\checkmark 2$› **.**
**hence** ‹*?rhs* $\sqsubseteq_{FD}$ *RenamingTick* (*RenamingTick ?lhs* $\otimes\checkmark 3 \Rightarrow \otimes\checkmark 2$) $\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$›
**by** (*fact mono-Renaming-FD*)
**also have** ‹*. . .* = *RenamingTick ?lhs* ($\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$ ∘ $\otimes\checkmark 3 \Rightarrow \otimes\checkmark 2$)›
**by** (*simp add*: *RenamingTick-comp*)
**also have** ‹*. . .* = *RenamingTick ?lhs id*›
**proof** (*rule RenamingTick-is-restrictable-on-strict-ticks-of*)
**fix** *r-s-t* **assume** ‹*r-s-t* ∈ ✓**s**(*P* $[\![S]\!]_{\checkmark 3}$ (*Q* $[\![S]\!]_{\checkmark 4}$ *R*))›
**with** $Sync_{ptick3}$*.strict-ticks-of-*$Sync_{ptick}$*-subset* **obtain** *r s-t*
**where** ‹*r* $\otimes\checkmark 3$ *s-t* = $\lfloor$*r-s-t*$\rfloor$› ‹*r* ∈ ✓**s**(*P*)› ‹*s-t* ∈ ✓**s**(*Q* $[\![S]\!]_{\checkmark 4}$ *R*)› **by** *blast*
**from** *this*(*3*) $Sync_{ptick4}$*.strict-ticks-of-*$Sync_{ptick}$*-subset* **obtain** *s t*
**where** ‹*s* $\otimes\checkmark 4$ *t* = $\lfloor$*s-t*$\rfloor$› ‹*s* ∈ ✓**s**(*Q*)› ‹*t* ∈ ✓**s**(*R*)› **by** *blast*
**from** ‹*r* $\otimes\checkmark 3$ *s-t* = $\lfloor$*r-s-t*$\rfloor$› ‹*s* $\otimes\checkmark 4$ *t* = $\lfloor$*s-t*$\rfloor$›
**show** ‹($\otimes\checkmark 2 \Rightarrow \otimes\checkmark 3$ ∘ $\otimes\checkmark 3 \Rightarrow \otimes\checkmark 2$) *r-s-t* = *id r-s-t*›
**by** (*auto intro*!: *tick-assoc-ren-tick-assoc-ren-conv*)
**qed**
**also have** ‹*. . .* = *?lhs*› **by** *simp*
**finally show** ‹*?rhs* $\sqsubseteq_{FD}$ *?lhs*› **.**
**qed**

# Chapter 7

# First Laws

**unbundle** *option-type-syntax*

## 7.1 Behaviour with Constant Processes

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

**lemma** $Seq_{ptick}$*-const* [*simp*] : ‹$P \mathbin{;_{\checkmark}} (\lambda r.\ Q) = P \mathbin{;} Q$›
— Very basic law.
**by** (*simp add*: *Process-eq-spec* $Seq_{ptick}$*-same-type-projs Seq-projs*)

### 7.1.1 The Laws of $\bot$

**lemma** $Seq_{ptick}$*-is-BOT-iff* : ‹$P \mathbin{;_{\checkmark}} Q = \bot \longleftrightarrow P = \bot \vee (\exists r.\ [\checkmark(r)] \in \mathcal{T}\ P \wedge Q\ r = \bot)$›
**by** (*simp add*: *BOT-iff-Nil-D* $Seq_{ptick}$*-projs*)

**lemma** *BOT-*$Seq_{ptick}$ [*simp*] : ‹$\bot \mathbin{;_{\checkmark}} P = \bot$› **by** (*simp add*: $Seq_{ptick}$*-is-BOT-iff*)

**lemma** (**in** $Sync_{ptick}$*-locale*) $Sync_{ptick}$*-is-BOT-iff* : ‹$P\ [\![S]\!]_{\checkmark}\ Q = \bot \longleftrightarrow P = \bot \vee Q = \bot$›
**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*] : ‹$P\ [\![S]\!]_{\checkmark}\ \bot = \bot$› **and** *BOT-*$Sync_{ptick}$ [*simp*] : ‹$\bot\ [\![S]\!]_{\checkmark}\ Q = \bot$›
**by** (*simp-all add*: $Sync_{ptick}$*-is-BOT-iff*)

### 7.1.2 The Laws of *STOP*

**lemma** $Seq_{ptick}$*-is-STOP-iff* :
‹$P \mathbin{;_{\checkmark}} Q = STOP \longleftrightarrow \mathcal{T}\ P \subseteq insert\ []\ \{[\checkmark(r)] \mid\mid\ r.\ True\} \wedge$

$(\forall\, r.\ [\checkmark(r)] \in \mathcal{T}\ P \longrightarrow Q\ r = STOP)\rangle$ (**is** ‹*?lhs* $\longleftrightarrow$ *?rhs*›)
**proof** (*intro iffI conjI subsetI allI impI*)
**show** ‹*?lhs* $\Longrightarrow$ $t \in \mathcal{T}\ P \Longrightarrow t \in insert\ []\ \{[\checkmark(r)]\ |r.\ True\}\rangle$ **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** ‹*?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** ‹*?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** ‹*?lhs* $\longleftrightarrow$ *?rhs*›)
**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** ‹*?lhs*› **show** *?rhs*
**by** (*subst* (*asm*) $Seq_{ptick}$*-is-STOP-iff*)
(*auto simp add*: *refine-defs SKIPS-projs*)
**next**
**show** ‹*?rhs* $\Longrightarrow$ *?lhs*›
**unfolding** $Seq_{ptick}$*-is-STOP-iff* **by** (*auto simp add*: *refine-defs SKIPS-projs*)
**qed**

**corollary** $STOP\text{-}Seq_{ptick}$ [*simp*] : $\langle STOP\ ;_{\checkmark}\ P = STOP\rangle$
**by** (*simp add*: $Seq_{ptick}$*-is-STOP-iff T-STOP*)

**lemma** (**in** $Sync_{ptick}$*-locale*) $STOP\text{-}Sync_{ptick}\text{-}STOP$ [*simp*] : $\langle STOP\ [\![S]\!]_{\checkmark}\ STOP = STOP\rangle$
**by** (*simp add*: *STOP-iff-T T-*$Sync_{ptick}$ *STOP-projs*)

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

**let** *?f* = ‹$\lambda r$. *the* (*tick-join r* (*g r*))›
**have** ∗ : ‹*tF t* ⟹ *map* (*map-event*$_{ptick}$ *id ?f*) *t*
= *map ev* (*map of-ev t*)› **for** *t* :: ‹('*a*, '*r*) *trace*$_{ptick}$›
**by** (*induct t*, *simp-all*)
(*metis* (*no-types*, *lifting*) *event*$_{ptick}$.*collapse*(*1*) *event*$_{ptick}$.*simps*(*9*) *id-apply*)
**show** ‹*?lhs* = *?rhs*›
**proof** (*rule Process-eq-optimizedI*)
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**then obtain** *u v t-P*
**where** ‹*t* = *u* @ *v*› ‹*ftF v*› ‹*tF u* ∨ *v* = []›
‹*u setinterleaves*$_{✓ tick\text{-}join}$ ((*t-P*, []), {})› ‹*t-P* ∈ $\mathcal{D}$ *P*›
**unfolding** *D-Sync*$_{ptick}$ *STOP-projs* **by** *blast*
**from** *this*(*4*) *setinterleaves*$_{ptick}$*-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-event*$_{ptick}$ *id ?f*) *t-P* @ *v*›
**by** (*simp add*: ‹*u* = *map ev* (*map of-ev t-P*)›)
**moreover have** ‹*t-P* ∈ $\mathcal{D}$ (*P* **;** *STOP*)› **by** (*simp add*: *D-Seq* ‹*t-P* ∈ $\mathcal{D}$ *P*›)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**using** ‹*ftF v*› ‹*tF t-P*› **by** (*auto simp add*: *D-Renaming* ‹*t* = *u* @ *v*›)
**next**
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**then obtain** *u v*
**where** $ : ‹*t* = *map* (*map-event*$_{ptick}$ *id ?f*) *u* @ *v*›
**and** ‹*tF u*› ‹*ftF v*› ‹*u* ∈ $\mathcal{D}$ *P*› **unfolding** *D-Renaming D-Seq D-STOP* **by** *blast*
**have** ‹*tF* (*map* (*map-event*$_{ptick}$ *id ?f*) *u*)›
**by** (*simp add*: ‹*tF u*› *map-event*$_{ptick}$*-tickFree*)
**moreover from** ‹*tF u*›
**have** ‹*map* (*map-event*$_{ptick}$ *id ?f*) *u*
*setinterleaves*$_{✓ tick\text{-}join}$ ((*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.prems 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* ∈ $\mathcal{D}$ (*P* |||$_{✓}$ *STOP*)›
**using** ‹*ftF v*› ‹*u* ∈ $\mathcal{D}$ *P*› **by** (*auto simp add*: *D-Sync*$_{ptick}$ *STOP-projs* $)
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› ‹*t* ∉ $\mathcal{D}$ *?lhs*›
**then obtain** *t-P X-P X-Q*
**where** ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹*t setinterleaves*$_{✓ tick\text{-}join}$ ((*t-P*, []), {})›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *tick-join X-P* {} *X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs F-STOP* **by** *blast*

**from** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *tick-join X-P* {} *X-Q*›

**have** \$ : ‹$e \in$ *map-event*$_{ptick}$ *id ?f* $-$‘ $X \cup$ *range tick*
$\Longrightarrow e \in$ *X-P* $\cup$ *range tick*› **for** *e*
**by** (*cases e*) (*auto simp add*: *super-ref-Sync*$_{ptick}$*-def*)
**from** *setinterleaves*$_{ptick}$*-NilR-iff*
[*THEN iffD1*, *OF* ‹*t setinterleaves*$_{\checkmark}$ *tick-join* ((*t-P*, []), {})›]
**have** ‹*tF t-P*› ‹*t* = *map ev* (*map of-ev t-P*)› **by** *simp-all*
**have** \$\$ : ‹*t* = *map* (*map-event*$_{ptick}$ *id ?f*) *t-P*›
**by** (*simp add*: ∗ ‹*t* = *map ev* (*map of-ev t-P*)› ‹*tF t-P*›)
**show** ‹(*t*, *X*) $\in \mathcal{F}$ *?rhs*›
**proof** (*cases* ‹$\exists$ *r*. *t-P* @ [✓(*r*)] $\in \mathcal{T}$ *P*›)
**show** ‹$\exists$ *r*. *t-P* @ [✓(*r*)] $\in \mathcal{T}$ *P* $\Longrightarrow$ (*t*, *X*) $\in \mathcal{F}$ *?rhs*›
**by** (*auto simp add*: *F-Renaming F-Seq F-STOP* \$\$)
**next**
**assume** ‹$\nexists$ *r*. *t-P* @ [✓(*r*)] $\in \mathcal{T}$ *P*›
**hence** ‹(*t-P*, *X-P* $\cup$ *range tick*) $\in \mathcal{F}$ *P*›
**by** (*auto intro*!: *is-processT5* ‹(*t-P*, *X-P*) $\in \mathcal{F}$ *P*› *F-T*)
**with** \$ **have** ‹(*t-P*, *map-event*$_{ptick}$ *id ?f* $-$‘ *X* $\cup$ *range tick*) $\in \mathcal{F}$ *P*›
**by** (*meson is-processT4 subsetI*)
**with** ‹*tF t-P*› **show** ‹(*t*, *X*) $\in \mathcal{F}$ *?rhs*› **by** (*auto simp add*: *F-Renaming F-Seq*
\$\$)
**qed**
**next**
**fix** *t X* **assume** ‹(*t*, *X*) $\in \mathcal{F}$ *?rhs*› ‹*t* $\notin \mathcal{D}$ *?rhs*›
**then obtain** *u* **where** \$ : ‹*t* = *map* (*map-event*$_{ptick}$ *id ?f*) *u*›
‹(*u*, *map-event*$_{ptick}$ *id ?f* $-$‘ *X*) $\in \mathcal{F}$ (*P* ; *STOP*)›
**unfolding** *Renaming-projs* **by** *blast*
**from** \$(*2*) **consider** ‹*u* $\in \mathcal{D}$ *P*› | *r* **where** ‹*u* @ [✓(*r*)] $\in \mathcal{T}$ *P*›
| ‹(*u*, *map-event*$_{ptick}$ *id ?f* $-$‘ *X* $\cup$ *range tick*) $\in \mathcal{F}$ *P*› ‹*tF u*›
**by** (*auto simp add*: *Seq-projs F-STOP*)
**thus** ‹(*t*, *X*) $\in \mathcal{F}$ *?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* $\in \mathcal{D}$ *P*›
**hence** ‹*?u*′ $\in \mathcal{D}$ *P*› **by** (*simp add*: *D-imp-front-tickFree div-butlast-when-non-tickFree-iff*)
**moreover from** *D-imp-front-tickFree* ‹*u* $\in \mathcal{D}$ *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-event*$_{ptick}$ *id ?f*) *?u*′ @ *?v*′›
**by** (*cases u rule*: *rev-cases*, *auto simp add*: \$(*1*) *split*: *if-split-asm*)
(*metis event*$_{ptick}$*.collapse*(*2*) *event*$_{ptick}$*.simps*(*10*))
**ultimately have** ‹*t* $\in \mathcal{D}$ *?rhs*› **unfolding** *D-Renaming D-Seq* **by** *blast*
**with** ‹*t* $\notin \mathcal{D}$ *?rhs*› **show** ‹(*t*, *X*) $\in \mathcal{F}$ *?lhs*› **..**
**next**
**fix** *r* **assume** ‹*u* @ [✓(*r*)] $\in \mathcal{T}$ *P*›
**hence** ‹(*u*, *UNIV* $-$ {✓(*r*)}) $\in \mathcal{F}$ *P*› **by** (*simp add*: *is-processT6-TR*)
**moreover from** \$(*1*) ∗ ‹*u* @ [✓(*r*)] $\in \mathcal{T}$ *P*› *append-T-imp-tickFree*
**have** ‹*t setinterleaves*$_{\checkmark}$ *tick-join* ((*u*, []), {})›
**by** (*auto simp add*: *setinterleaves*$_{ptick}$*-NilR-iff*)

**moreover have** ‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ *tick-join* (*UNIV* − {✓(*r*)}) {} *UNIV*›
**by** (*simp add*: *subset-iff super-ref-Sync*$_{ptick}$*-def*) (*metis event*$_{ptick}$*.exhaust*)
**ultimately show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **unfolding** *F-Sync*$_{ptick}$ *F-STOP* **by** *clarify blast*
**next**
**assume** ‹(*u*, *map-event*$_{ptick}$ *id ?f* −‘ *X* ∪ *range tick*) ∈ $\mathcal{F}$ *P*› ‹*tF u*›
**moreover from** $(1) ∗ ‹*tF u*› **have** ‹*t setinterleaves*$_{✓\,tick\text{-}join}$ ((*u*, []), {})›
**by** (*auto simp add*: *setinterleaves*$_{ptick}$*-NilR-iff*)
**moreover have** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *tick-join* (*map-event*$_{ptick}$ *id ?f* −‘ *X* ∪ *range tick*) {} *UNIV*›
**by** (*simp add*: *subset-iff super-ref-Sync*$_{ptick}$*-def*) (*metis event*$_{ptick}$*.exhaust*)
**ultimately show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **unfolding** *F-Sync*$_{ptick}$ *F-STOP* **by** *clarify blast*
**qed**
**qed**
**qed**

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *STOP-Inter*$_{ptick}$ :
‹*STOP* $|||_{✓}$ *Q* = *RenamingTick* (*Q* ; *STOP*)
(λ*s*. *the* (*tick-join* (*g s*) *s*))›
**by** (*metis Sync*$_{ptick}$*-locale-sym.Inter*$_{ptick}$*-STOP Sync*$_{ptick}$*-sym*)

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *Par*$_{ptick}$*-STOP* [*simp*] : ‹*P* $||_{✓}$ *STOP* = (*if P* = ⊥ *then* ⊥ *else STOP*)›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*P* = ⊥ ⟹ *P* $||_{✓}$ *STOP* = ⊥› **by** *simp*
**next**
**show** ‹*P* ≠ ⊥ ⟹ *P* $||_{✓}$ *STOP* = *STOP*›
**by** (*auto simp add*: *STOP-iff-T T-Sync*$_{ptick}$ *STOP-projs set-eq-iff*
*BOT-iff-Nil-D setinterleaves*$_{ptick}$*-NilR-iff image-iff*)
(*metis event*$_{ptick}$*.collapse*(*1*) *last-in-set tickFree-butlast*)+
**qed**

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *STOP-Par*$_{ptick}$ [*simp*] : ‹*STOP* $||_{✓}$ *P* = (*if P* = ⊥ *then* ⊥ *else STOP*)›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*P* = ⊥ ⟹ *STOP* $||_{✓}$ *P* = ⊥› **by** *simp*
**next**
**show** ‹*P* ≠ ⊥ ⟹ *STOP* $||_{✓}$ *P* = *STOP*›
**by** (*auto simp add*: *STOP-iff-T T-Sync*$_{ptick}$ *STOP-projs set-eq-iff*
*BOT-iff-Nil-D setinterleaves*$_{ptick}$*-NilL-iff image-iff*)
(*metis event*$_{ptick}$*.collapse*(*1*) *last-in-set tickFree-butlast*)+
**qed**

### 7.1.3 The Laws of *SKIP*

#### Sequential Composition

*SKIP* is neutral for $Seq_{ptick}$!

**lemma** *SKIP-Seq*$_{ptick}$ [*simp*] : ‹*SKIP r* ;$_{\checkmark}$ *P* = *P r*›
**by** (*simp add*: *Process-eq-spec Seq*$_{ptick}$*-projs ref-Seq*$_{ptick}$*-def SKIP-projs*)

**lemma** *Seq*$_{ptick}$*-SKIP* [*simp*] : ‹*P* ;$_{\checkmark}$ *SKIP* = *P*›
**proof** (*subst Process-eq-spec-optimized*, *safe*)
**show** ‹*s* ∈ $\mathcal{D}$ (*P* ;$_{\checkmark}$ *SKIP*) ⟹ *s* ∈ $\mathcal{D}$ *P*›
**and** ‹*s* ∈ $\mathcal{D}$ *P* ⟹ *s* ∈ $\mathcal{D}$ (*P* ;$_{\checkmark}$ *SKIP*)› **for** *s*
**by** (*simp-all add*: *D-Seq*$_{ptick}$*-same-type D-SKIP*)
**next**
**show** ‹(*s*, *X*) ∈ $\mathcal{F}$ (*P* ;$_{\checkmark}$ *SKIP*) ⟹ (*s*, *X*) ∈ $\mathcal{F}$ *P*› **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** ‹(*s*, *X*) ∈ $\mathcal{F}$ *P* ⟹ (*s*, *X*) ∈ $\mathcal{F}$ (*P* ;$_{\checkmark}$ *SKIP*)› **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-Seq*$_{ptick}$ [*simp*] : ‹*SKIPS R* ;$_{\checkmark}$ *P* = ⊓*r* ∈ *R*. *P r*›
**by** (*auto simp add*: *Process-eq-spec GlobalNdet-projs Seq*$_{ptick}$*-projs STOP-projs SKIPS-projs ref-Seq*$_{ptick}$*-def*)

**lemma** *finite-ticks-Seq*$_{ptick}$ [*finite-ticks-simps*] : ‹$\mathbb{F}_{\checkmark}$(*P* ;$_{\checkmark}$ *Q*)›
**if** ‹$\mathbb{F}_{\checkmark}$(*P*)› **and** ‹(⋀*r*. *r* ∈ ✓**s**(*P*) ⟹ $\mathbb{F}_{\checkmark}$(*Q r*))›
**proof** (*rule finite-ticksI*)
**fix** *t* **assume** ‹*t* ∈ $\mathcal{T}$ (*P* ;$_{\checkmark}$ *Q*)› ‹*t* ∉ $\mathcal{D}$ (*P* ;$_{\checkmark}$ *Q*)›
**have** ‹{*r'*. *t* @ [✓(*r'*)] ∈ $\mathcal{T}$ (*P* ;$_{\checkmark}$ *Q*)} ⊆
(⋃ *u* ∈ {*u*. ∃ *v r*. *t* = *map* (*ev* ∘ *of-ev*) *u* @ *v* ∧ *u* @ [✓(*r*)] ∈ $\mathcal{T}$ *P* ∧ *u* ∉ $\mathcal{D}$ *P* ∧ *tF u* ∧ *ftF v*}.
⋃ *r* ∈ {*r*. *u* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}. {*s*. *drop* (*length u*) *t* @ [✓(*s*)] ∈ $\mathcal{T}$ (*Q r*)})›
(**is** ‹*?lhs* ⊆ *?rhs*›)
**proof** (*rule subsetI*)
**fix** *r'* **assume** ‹*r'* ∈ *?lhs*›
**hence** ‹*t* @ [✓(*r'*)] ∈ $\mathcal{T}$ (*P* ;$_{\checkmark}$ *Q*)› **by** *simp*
**with** ‹*t* ∉ $\mathcal{D}$ (*P* ;$_{\checkmark}$ *Q*)› **obtain** *t' r u'*
**where** ‹*t* @ [✓(*r'*)] = *map* (*ev* ∘ *of-ev*) *t'* @ *u'*› ‹*t'* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*› ‹*t'* ∉ $\mathcal{D}$ *P*› ‹*tF t'*› ‹*u'* ∈ $\mathcal{T}$ (*Q r*)›
**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 butlast-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** ‹$r' \in$ *?rhs*›
**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 event*$_{ptick}$*.disc*(*2*) *front-tickFree-dw-closed length-map tickFree-Cons-iff tickFree-append-iff*
*tickFree-map-ev-comp*)
**qed**
**moreover have** ‹*finite* ...›
**proof** (*rule finite-UN-I*)
**show** ‹*finite* {$u$. $\exists\, v\ r.\ t = map\ (ev \circ of\text{-}ev)\ u\ @\ v \wedge u\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge u \notin \mathcal{D}\ P \wedge tF\ u \wedge ftF\ v$}›
**by** (*rule finite-subset*[*of* - ‹{$u$. $u \leq map\ (ev \circ of\text{-}ev)$ (*if tF t then t else butlast t*)}›])
(*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** ‹$u \in$ {$u$. $\exists\, v\ r.\ t = map\ (ev \circ of\text{-}ev)\ u\ @\ v \wedge u\ @\ [\checkmark(r)] \in \mathcal{T}\ P \wedge u \notin \mathcal{D}\ P \wedge tF\ u \wedge ftF\ v$}›
**then obtain** $r\ v$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ u\ @\ v$› ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$u \notin \mathcal{D}\ P$› ‹*tF u*› ‹*ftF v*› **by** *blast*
**show** ‹*finite* ($\bigcup r \in \{r.\ u\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}.$ {$s$. *drop* (*length u*) $t\ @\ [\checkmark(s)] \in \mathcal{T}\ (Q\ r)$})›
**proof** (*rule finite-UN-I*)
**show** ‹*finite* {$r$. $u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$}›
**by** (*simp add*: ‹$\mathbb{F}_{\checkmark}(P)$› ‹$u \notin \mathcal{D}\ P$› *finite-ticksD*)
**next**
**fix** $r$ **assume** ‹$r \in$ {$r$. $u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$}›
**hence** ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› **..**
**with** ‹$t \notin \mathcal{D}\ (P\ ;_{\checkmark}\ Q)$› ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹*tF u*› **have** ‹$v \notin \mathcal{D}\ (Q\ r)$›
**by** (*auto simp add*: ‹$t = map\ (ev \circ of\text{-}ev)\ u\ @\ v$› *Seq*$_{ptick}$*-projs*)
**show** ‹*finite* {*s*. *drop* (*length u*) $t\ @\ [\checkmark(s)] \in \mathcal{T}\ (Q\ r)$}›
**by** (*simp add*: ‹$t = map\ (ev \circ of\text{-}ev)\ u\ @\ v$›)
(*metis* ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$u \notin \mathcal{D}\ P$› ‹$v \notin \mathcal{D}\ (Q\ r)$› *finite-ticksD*
*is-processT9 strict-ticks-of-memI* ‹($\bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow \mathbb{F}_{\checkmark}(Q\ r)$)›)
**qed**
**qed**
**ultimately show** ‹*finite* {$r$. $t\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)$}› **by** (*fact finite-subset*)
**qed**

**lemma** *finite-ticks-fun-Seq$_{ptick}$-bis* :
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow (\bigwedge x\ r.\ r \in \checkmark\mathbf{s}(f\ x) \Longrightarrow \mathbb{F}_{\checkmark}(x) \Longrightarrow \mathbb{F}_{\checkmark}(g\ x\ r)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
**by** (*simp add*: *finite-ticks-fun-def finite-ticks-Seq$_{ptick}$*)

**lemma** *finite-ticks-fun-Seq$_{ptick}$* [*finite-ticks-fun-simps*] :
— Big approximation.
‹$\mathbb{F}_{\checkmark\Rightarrow}(f) \Longrightarrow (\bigwedge r.\ r \in (\bigcup x.\ \checkmark\mathbf{s}(f\ x)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ g\ x\ r)) \Longrightarrow \mathbb{F}_{\checkmark\Rightarrow}(\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
**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* $[\![A]\!]$ *SKIP s* = (*if r* = *s then SKIP r else STOP*)).

**theorem** (**in** *Sync$_{ptick}$-locale*) *SKIP-Sync$_{ptick}$-SKIP* [*simp*] :
‹*SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s* = (*case tick-join r s of* $\lfloor$*r-s*$\rfloor$ ⇒ *SKIP r-s* | ◊ ⇒ *STOP*)›
**proof** (*split option.split*, *intro conjI impI allI*)
**show** ‹*tick-join r s* = ◊ ⟹ *SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s* = *STOP*›
**unfolding** *STOP-iff-T T-Sync$_{ptick}$ SKIP-projs set-eq-iff*
**by** (*safe*, *simp-all*, *metis Nil-setinterleaving$_{ptick}$-Nil insertCI*)
**next**
**show** ‹*SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s* = *SKIP r-s*› **if** ‹*tick-join r s* = $\lfloor$*r-s*$\rfloor$› **for** *r-s*
**proof** (*rule Process-eq-optimizedI*)
**show** ‹*t* ∈ $\mathcal{D}$ (*SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s*) ⟹ *t* ∈ $\mathcal{D}$ (*SKIP r-s*)› **for** *t*
**by** (*simp add*: *D-Sync$_{ptick}$ SKIP-projs*)
**next**
**show** ‹*t* ∈ $\mathcal{D}$ (*SKIP r-s*) ⟹ *t* ∈ $\mathcal{D}$ (*SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s*)› **for** *t*
**by** (*simp add*: *D-Sync$_{ptick}$ SKIP-projs*)
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s*)› ‹*t* ∉ $\mathcal{D}$ (*SKIP r* $[\![A]\!]_{\checkmark}$ *SKIP s*)›
**then obtain** *t-P t-Q X-P X-Q*
**where** *fail*: ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ (*SKIP r*)› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ (*SKIP s*)›
‹*t setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*t-P*, *t-Q*), *A*)›
‹*X* ⊆ *super-ref-Sync$_{ptick}$ tick-join X-P A X-Q*›
**unfolding** *Sync$_{ptick}$-projs* **by** *blast*
**from** *fail*(*1*−*3*) **consider** ‹*t* = []› ‹✓(*r*) ∉ *X-P*› ‹✓(*s*) ∉ *X-Q*› | ‹*t* = [✓(*r-s*)]›
**by** (*cases t-P*; *cases t-Q*) (*simp-all add*: *F-SKIP* ‹*tick-join r s* = $\lfloor$*r-s*$\rfloor$›)
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*SKIP r-s*)›
**proof** *cases*
**assume** ‹*t* = []› ‹✓(*r*) ∉ *X-P*› ‹✓(*s*) ∉ *X-Q*›
**from** ‹✓(*r*) ∉ *X-P*› ‹✓(*s*) ∉ *X-Q*› *fail*(*4*) ‹*tick-join r s* = $\lfloor$*r-s*$\rfloor$›
**have** ‹✓(*r-s*) ∉ *X*›
**by** (*simp add*: *super-ref-Sync$_{ptick}$-def subset-iff*)
(*metis event$_{ptick}$.distinct*(*1*) *event$_{ptick}$.inject*(*2*) *inj-tick-join*)

```
    with ‹t = []› show ‹(t, X) ∈ ℱ (SKIP r-s)› by (simp add: F-SKIP)
  next
    show ‹t = [✓(r-s)] ⟹ (t, X) ∈ ℱ (SKIP r-s)› by (simp add: F-SKIP)
  qed
next
  fix t :: ‹('a, -) trace_ptick› and X assume ‹(t, X) ∈ ℱ (SKIP r-s)›
  then consider ‹t = []› ‹✓(r-s) ∉ X› | ‹t = [✓(r-s)]›
    unfolding F-SKIP by blast
  thus ‹(t, X) ∈ ℱ (SKIP r ⟦A⟧✓ SKIP s)›
  proof cases
    assume ‹t = []› ‹✓(r-s) ∉ X›
    have ‹([], - {✓(r)}) ∈ ℱ (SKIP r)›
      by (simp add: F-SKIP)
    moreover have ‹([], - {✓(s)}) ∈ ℱ (SKIP s)›
      by (simp add: F-SKIP)
    moreover have  ‹t setinterleaves✓ tick-join (([], []), A)›
      by (simp add: ‹t = []›)
   moreover have ‹X ⊆ super-ref-Sync_ptick tick-join (- {✓(r)}) A (- {✓(s)})›
      using ‹✓(r-s) ∉ X› ‹tick-join r s = ⌊r-s⌋›
      by (simp add: super-ref-Sync_ptick-def subset-iff)
        (metis event_ptick.exhaust option.inject)
    ultimately show ‹(t, X) ∈ ℱ (SKIP r ⟦A⟧✓ SKIP s)›
      by (simp (no-asm) add: F-Sync_ptick) blast
  next
    assume ‹t = [✓(r-s)]›
    have ‹([✓(r)], UNIV) ∈ ℱ (SKIP r)›
      by (simp add: F-SKIP)
    moreover have ‹([✓(s)], UNIV) ∈ ℱ (SKIP s)›
      by (simp add: F-SKIP)
    moreover have ‹t setinterleaves✓ tick-join (([✓(r)], [✓(s)]), A)›
      by (simp add: ‹t = [✓(r-s)]› ‹tick-join r s = ⌊r-s⌋›)
    moreover have ‹X ⊆ super-ref-Sync_ptick tick-join UNIV A UNIV›
      by (simp add: super-ref-Sync_ptick-def subset-iff)
        (metis event_ptick.exhaust)
    ultimately show ‹(t, X) ∈ ℱ (SKIP r ⟦A⟧✓ SKIP s)›
      by (simp (no-asm) add: F-Sync_ptick) blast
  qed
qed
qed
```

**lemma** (**in** *Sync_ptick-locale*) *STOP-Sync_ptick-SKIP* [*simp*] : ‹*STOP* ⟦*A*⟧✓ *SKIP s* = *STOP*›
**and** *SKIP-Sync_ptick-STOP* [*simp*] : ‹*SKIP r* ⟦*A*⟧✓ *STOP* = *STOP*›
**by** (*force simp add*: *STOP-iff-T T-Sync_ptick STOP-projs SKIP-projs*)+

**lemma** (**in** *Sync_ptick-locale*) *Mprefix-Sync_ptick-SKIP* :

‹$\Box a \in A \to P\ a\ [\![S]\!]_{\checkmark}$ *SKIP* $r$ =
$\Box a \in (A - S) \to (P\ a\ [\![S]\!]_{\checkmark}$ *SKIP* $r)$› (**is** ‹*?lhs* = *?rhs*›)
**proof** (*rule Process-eq-optimizedI*)
**fix** $t$ **assume** ‹$t \in \mathcal{D}$ *?lhs*›
**then obtain** $u\ v$ *t-P a t-Q*
**where** $* :$ ‹$t = u$ @ $v$› ‹*tF u*› ‹*ftF v*› ‹$a \in A$› ‹*t-P* $\in \mathcal{D}$ $(P\ a)$›
‹*u setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((ev\ a \#$ *t-P*, *t-Q*$)$, *S*)› ‹*t-Q* = [] $\vee$ *t-Q* = [✓$(r)$]›
**unfolding** *D-Sync*$_{ptick}$ *SKIP-projs Mprefix-projs* **by** *blast*
**from** $*(6, 7)$ **obtain** $u'$
**where** ‹$a \notin S$› ‹$u = ev\ a \#\ u'$› ‹$u'$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P, t\text{-}Q), S)$›
**by** (*auto split*: *if-split-asm*)
**from** *this*$(2, 3)$ $*(2, 5, 7)$ *front-tickFree-charn* **have** ‹$u' \in \mathcal{D}$ $(P\ a\ [\![S]\!]_{\checkmark}$ *SKIP* $r)$›
**by** (*auto simp add*: *D-Sync*$_{ptick}$ *SKIP-projs*)
**with** $*(1{-}4)$ ‹$a \notin S$› ‹$u = ev\ a \#\ u'$› *is-processT7* **show** ‹$t \in \mathcal{D}$ *?rhs*›
**by** (*auto simp add*: *D-Mprefix*)
**next**
**fix** $t$ **assume** ‹$t \in \mathcal{D}$ *?rhs*›
**then obtain** $a\ t'$ **where** ‹$t = ev\ a \#\ t'$› ‹$a \in A$› ‹$a \notin S$› ‹$t' \in \mathcal{D}$ $(P\ a\ [\![S]\!]_{\checkmark}$ *SKIP* $r)$›
**unfolding** *D-Mprefix* **by** *blast*
**then obtain** $u\ v$ *t-P t-Q*
**where** $* :$ ‹$t' = u$ @ $v$› ‹*tF u*› ‹*ftF v*› ‹*t-P* $\in \mathcal{D}$ $(P\ a)$›
‹*u setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P, t\text{-}Q), S)$› ‹*t-Q* = [] $\vee$ *t-Q* = [✓$(r)$]›
**unfolding** *D-Sync*$_{ptick}$ *SKIP-projs* **by** *blast*
**have** ‹$t = (ev\ a \#\ u)$ @ $v$› **by** (*simp add*: $*(1)$ ‹$t = ev\ a \#\ t'$›)
**moreover from** $*(2)$ **have** ‹*tF* $(ev\ a \#\ u)$› **by** *simp*
**moreover from** $*(5, 6)$ **have** ‹$ev\ a \#\ u$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((ev\ a \#$ *t-P*, *t-Q*$)$, *S*)›
**by** (*cases t-Q*) (*simp-all add*: ‹$a \notin S$› *Cons-setinterleaving*$_{ptick}$*-Cons*)
**moreover have** ‹$ev\ a \#$ *t-P* $\in \mathcal{D}$ $(\Box a \in A \to P\ a)$›
**by** (*simp add*: $*(4)$ *D-Mprefix* ‹$a \in A$›)
**ultimately show** ‹$t \in \mathcal{D}$ *?lhs*›
**unfolding** *D-Sync*$_{ptick}$ *SKIP-projs* **using** $*(3, 6)$ **by** *blast*
**next**
**fix** $t\ X$ **assume** ‹$(t, X) \in \mathcal{F}$ *?lhs*› ‹$t \notin \mathcal{D}$ *?lhs*›
**then obtain** *t-P t-Q X-P X-Q*
**where** $* :$ ‹(*t-P*, *X-P*) $\in \mathcal{F}$ $(\Box a \in A \to P\ a)$› ‹(*t-Q*, *X-Q*) $\in \mathcal{F}$ $(SKIP\ r)$›
‹*t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P, t\text{-}Q), S)$›
‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)$ *X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**from** $*(1)$ **consider** ‹*t-P* = []› ‹*X-P* $\cap$ *ev* ‘ $A = \{\}$›
| $a$ *t-P*$'$ **where** ‹*t-P* = $ev\ a \#$ *t-P*$'$› ‹$a \in A$› ‹(*t-P*$'$, *X-P*) $\in \mathcal{F}$ $(P\ a)$›
**unfolding** *F-Mprefix* **by** *blast*
**thus** ‹$(t, X) \in \mathcal{F}$ *?rhs*›
**proof** *cases*
**assume** ‹*t-P* = []› ‹*X-P* $\cap$ *ev* ‘ $A = \{\}$›
**from** ‹*t-P* = []› $*(2, 3)$ **have** ‹$t$ = []› ‹*t-Q* = []› ‹✓$(r) \notin$ *X-Q*›

**by** (*auto simp add*: *F-SKIP*)
**with** ∗(*4*) **show** ‹*t-P* = [] ⟹ *X-P* ∩ *ev* ‘ *A* = {} ⟹ (*t*, *X*) ∈ ℱ *?rhs*›
**by** (*auto simp add*: *F-Mprefix super-ref-Sync*$_{ptick}$*-def*)
**next**
**fix** *a t-P*′ **assume** ‹*t-P* = *ev a* # *t-P*′› ‹*a* ∈ *A*› ‹(*t-P*′, *X-P*) ∈ ℱ (*P a*)›
**from** ∗(*2*, *3*) **obtain** *t*′
**where** ‹*t* = *ev a* # *t*′› ‹*a* ∉ *S*› ‹*t*′ *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*′, *t-Q*), *S*)›
**by** (*auto simp add*: ‹*t-P* = *ev a* # *t-P*′› *F-SKIP split*: *if-split-asm*)
**from** ∗(*2*, *4*) ‹(*t-P*′, *X-P*) ∈ ℱ (*P a*)› *this*(*3*)
**have** ‹(*t*′, *X*) ∈ ℱ (*P a* ⟦*S*⟧$_{✓}$ *SKIP r*)› **by** (*auto simp add*: *F-Sync*$_{ptick}$)
**thus** ‹(*t*, *X*) ∈ ℱ *?rhs*› **by** (*simp add*: ‹*t* = *ev a* # *t*′› *F-Mprefix* ‹*a* ∈ *A*› ‹*a* ∉ *S*›)
**qed**
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ ℱ *?rhs*› ‹*t* ∉ 𝒟 *?rhs*›
**from** ‹(*t*, *X*) ∈ ℱ *?rhs*› **consider** ‹*t* = []› ‹*X* ∩ *ev* ‘ (*A* − *S*) = {}›
| *a t*′ **where** ‹*t* = *ev a* # *t*′› ‹*a* ∈ *A*› ‹*a* ∉ *S*› ‹(*t*′, *X*) ∈ ℱ (*P a* ⟦*S*⟧$_{✓}$ *SKIP r*)›
**unfolding** *F-Mprefix* **by** *blast*
**thus** ‹(*t*, *X*) ∈ ℱ *?lhs*›
**proof** *cases*
**assume** ‹*t* = []› ‹*X* ∩ *ev* ‘ (*A* − *S*) = {}›
**from** ‹*X* ∩ *ev* ‘ (*A* − *S*) = {}›
**have** ‹([], *range tick* ∪ {*ev a* |*a*. *ev a* ∈ *X* ∧ *a* ∉ *S*}) ∈ ℱ (□*a* ∈ *A* → *P a*)›
**by** (*auto simp add*: *F-Mprefix*)
**moreover have** ‹([], *UNIV* − {✓(*r*)}) ∈ ℱ (*SKIP r*)› **by** (*simp add*: *F-SKIP*)
**moreover have** ‹*X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓)
(*range tick* ∪ {*ev a* |*a*. *ev a* ∈ *X* ∧ *a* ∉ *S*}) *S* (*UNIV* − {✓(*r*)})›
**by** (*simp add*: *subset-iff super-ref-Sync*$_{ptick}$*-def*) (*metis event*$_{ptick}$*.exhaust*)
**moreover have** ‹[] *setinterleaves*$_{✓(⊗✓)}$ (([], []), *S*)› **by** *simp*
**ultimately show** ‹(*t*, *X*) ∈ ℱ *?lhs*› **by** (*simp* (*no-asm*) *add*: ‹*t* = []› *F-Sync*$_{ptick}$) *blast*
**next**
**fix** *a t*′ **assume** ‹*t* = *ev a* # *t*′› ‹*a* ∈ *A*› ‹*a* ∉ *S*› ‹(*t*′, *X*) ∈ ℱ (*P a* ⟦*S*⟧$_{✓}$ *SKIP r*)›
**from** *this*(*1*, *4*) ‹*t* ∉ 𝒟 *?rhs*› ‹*a* ∈ *A*› ‹*a* ∉ *S*›
**obtain** *t-P t-Q X-P X-Q*
**where** ∗ : ‹(*t-P*, *X-P*) ∈ ℱ (*P a*)› ‹(*t-Q*, *X-Q*) ∈ ℱ (*SKIP r*)›
‹*t*′ *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*, *t-Q*), *S*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) *X-P S X-Q*›
**unfolding** *D-Mprefix Sync*$_{ptick}$*-projs* **by** *force*
**have** ‹(*ev a* # *t-P*, *X-P*) ∈ ℱ (□*a* ∈ *A* → *P a*)›
**by** (*simp add*: ∗(*1*) *F-Mprefix* ‹*a* ∈ *A*›)
**moreover from** ∗(*2*, *3*) **have** ‹*t setinterleaves*$_{✓(⊗✓)}$ ((*ev a* # *t-P*, *t-Q*), *S*)›
**by** (*auto simp add*: ‹*t* = *ev a* # *t*′› *F-SKIP* ‹*a* ∉ *S*›)
**ultimately show** ‹(*t*, *X*) ∈ ℱ *?lhs*› **unfolding** *F-Sync*$_{ptick}$ **using** ∗(*2*, *4*) **by** *auto*
**qed**
**qed**

**corollary** (**in** *Sync$_{ptick}$-locale*) *SKIP-Sync$_{ptick}$-Mprefix* :
‹*SKIP r* ⟦*S*⟧$_{\checkmark}$ □*b* ∈ *B* → *Q b* = □*b* ∈ (*B* − *S*) → (*SKIP r* ⟦*S*⟧$_{\checkmark}$ *Q b*)› (**is** ‹*?lhs* = *?rhs*›)
**by** (*subst* (*1 2*) *Sync$_{ptick}$-locale-sym.Sync$_{ptick}$-sym mono-Mprefix-eq*)
(*fact Sync$_{ptick}$-locale-sym.Mprefix-Sync$_{ptick}$-SKIP*)

**lemma** (**in** *Sync$_{ptick}$-locale*) *finite-ticks-Sync$_{ptick}$* [*finite-ticks-simps*] :
‹𝔽$_{\checkmark}$(*P* ⟦*S*⟧$_{\checkmark}$ *Q*)› **if** ‹𝔽$_{\checkmark}$(*P*)› **and** ‹𝔽$_{\checkmark}$(*Q*)›
**proof** (*rule finite-ticksI*)
**fix** *t* **assume** ‹*t* ∉ 𝒟 (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)›
**have** ‹{*r-s*. *t* @ [✓(*r-s*)] ∈ 𝒯 (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)} ⊆
(⋃(*t-P*, *t-Q*)∈{(*t-P*, *t-Q*). *t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), *S*)}.
{*r-s*. ∃ *r s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧
*t-P* @ [✓(*r*)] ∈ 𝒯 *P* ∧ *t-P* ∉ 𝒟 *P* ∧
*t-Q* @ [✓(*s*)] ∈ 𝒯 *Q* ∧ *t-Q* ∉ 𝒟 *Q*})›
(**is** ‹- ⊆ *?rhs*›)
**proof** (*rule subsetI*)
**fix** *r-s* **assume** ‹*r-s* ∈ {*r-s*. *t* @ [✓(*r-s*)] ∈ 𝒯 (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)}›
**hence** ‹*t* @ [✓(*r-s*)] ∈ 𝒯 (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)› **..**
**moreover from** ‹*t* ∉ 𝒟 (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)› **have** ‹*t* @ [✓(*r-s*)] ∉ 𝒟 (*P* ⟦*S*⟧$_{\checkmark}$ *Q*)›
**by** (*meson is-processT9*)
**ultimately obtain** *t-P t-Q* **where** ‹*t-P* ∈ 𝒯 *P*› ‹*t-Q* ∈ 𝒯 *Q*› ‹*t-P* ∉ 𝒟 *P*› ‹*t-Q* ∉ 𝒟 *Q*›
‹*t* @ [✓(*r-s*)] *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), *S*)›
**by** (*simp add*: *T-Sync$_{ptick}$ D-Sync$_{ptick}$′*) (*use front-tickFree-Nil* **in** *blast*)
**from** *this*(*5*) **show** ‹*r-s* ∈ *?rhs*›
**proof** (*elim snoc-tick-setinterleaves$_{ptick}$E*)
**fix** *r s t-P′ t-Q′*
**assume** *assms* : ‹*t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P′*, *t-Q′*), *S*)›
‹*r* ⊗✓ *s* = ⌊*r-s*⌋› ‹*t-P* = *t-P′* @ [✓(*r*)]› ‹*t-Q* = *t-Q′* @ [✓(*s*)]›
**from** ‹*t-P* ∈ 𝒯 *P*› ‹*t-Q* ∈ 𝒯 *Q*› ‹*t-P* ∉ 𝒟 *P*› ‹*t-Q* ∉ 𝒟 *Q*› *assms*(*3*, *4*)
**have** ‹*t-P′* ∉ 𝒟 *P*› ‹*t-Q′* ∉ 𝒟 *Q*›
**by** (*meson T-imp-front-tickFree front-tickFree-append-iff is-processT7 not-Cons-self2*)+
**with** ‹*t-P* ∈ 𝒯 *P*› ‹*t-Q* ∈ 𝒯 *Q*› *assms* **show** ‹*r-s* ∈ *?rhs*› **by** *auto*
**qed**
**qed**
**moreover have** ‹*finite* ...›
**proof** (*rule finite-UN-I*, *safe*)
**show** ‹*finite* {(*t-P*, *t-Q*). *t setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), *S*)}›
**by** (*fact finite-setinterleaves$_{ptick}$-tick-join*)
**next**
**fix** *t-P t-Q*
**let** *?S* = ‹{*r-s*. ∃ *r s*. *r* ⊗✓ *s* = ⌊*r-s*⌋ ∧
*t-P* @ [✓(*r*)] ∈ 𝒯 *P* ∧ *t-P* ∉ 𝒟 *P* ∧
*t-Q* @ [✓(*s*)] ∈ 𝒯 *Q* ∧ *t-Q* ∉ 𝒟 *Q*}›
**have** ‹*Some* ‘ *?S* ⊆ (λ(*r*, *s*). *r* ⊗✓ *s*) ‘
({*r*. *t-P* @ [✓(*r*)] ∈ 𝒯 *P* ∧ *t-P* ∉ 𝒟 *P*} ×

$\{s.\ t\text{-}Q\ @\ [\checkmark(s)] \in \mathcal{T}\ Q \wedge t\text{-}Q \notin \mathcal{D}\ Q\}\rangle$ **by** *force*
**moreover have** ‹*finite* ...› **by** (*simp add*: *finite-ticksD* ‹$\mathbb{F}_\checkmark(P)$› ‹$\mathbb{F}_\checkmark(Q)$›)
**ultimately have** ‹*finite* (*Some* ‘ *?S*)› **by** (*fact finite-subset*)
**thus** ‹*finite ?S*› **by** (*simp add*: *finite-image-iff*)
**qed**
**ultimately show** ‹*finite* $\{r\text{-}s.\ t\ @\ [\checkmark(r\text{-}s)] \in \mathcal{T}\ (P\ [\![S]\!]_\checkmark\ Q)\}$› **by** (*fact finite-subset*)
**qed**

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

## 7.2 Associativity of Sequential Composition

**lemma** $Seq_{ptick}$-*assoc* : ‹$P\ ;_\checkmark\ (\lambda r.\ Q\ r\ ;_\checkmark\ R) = P\ ;_\checkmark\ Q\ ;_\checkmark\ R$›
**for** $P :: \langle('a, 'r)\ process_{ptick}\rangle$
**and** $Q :: \langle'r \Rightarrow ('a, 's)\ process_{ptick}\rangle$
**and** $R :: \langle's \Rightarrow ('a, 't)\ process_{ptick}\rangle$
**proof** (*rule Process-eq-optimizedI*)
**fix** $t$ **assume** ‹$t \in \mathcal{D}\ (P\ ;_\checkmark\ (\lambda r.\ Q\ r\ ;_\checkmark\ R))$›
**then consider** (*D-P*) $t'\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t' \in \mathcal{D}\ P$› ‹*tF* $t'$› ‹*ftF* $u$›
| (*D-Q-R*) $t'\ r\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t'\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$u \in \mathcal{D}\ (Q\ r\ ;_\checkmark\ R)$›
**unfolding** $Seq_{ptick}$-*projs*[*of P*] **by** *fast*
**thus** ‹$t \in \mathcal{D}\ (P\ ;_\checkmark\ Q\ ;_\checkmark\ R)$›
**proof** *cases*
**case** *D-P*
**define** $t'' :: \langle('a, 's)\ trace_{ptick}\rangle$ **where** ‹$t'' = map\ (ev \circ of\text{-}ev)\ t'$›
**from** *D-P* **have** ‹$t'' \in \mathcal{D}\ (P\ ;_\checkmark\ Q)$›
**by** (*auto simp add*: *t''-def Seq*$_{ptick}$-*projs intro*: *front-tickFree-Nil*)
**moreover have** ‹*tF t''*› **by** (*simp add*: *t''-def*)
**ultimately have** ‹$map\ (ev \circ of\text{-}ev)\ t'' \in \mathcal{D}\ (P\ ;_\checkmark\ Q\ ;_\checkmark\ R)$›
**by** (*simp add*: $Seq_{ptick}$-*projs*[*of* ‹$P\ ;_\checkmark\ Q$›])
(*metis append.right-neutral front-tickFree-Nil*)
**also have** ‹$map\ (ev \circ of\text{-}ev)\ t'' = map\ (ev \circ of\text{-}ev)\ t'$›
**by** (*simp add*: *t''-def*)
**finally have** ‹$map\ (ev \circ of\text{-}ev)\ t' \in \mathcal{D}\ (P\ ;_\checkmark\ Q\ ;_\checkmark\ R)$› .
**with** *D-P*(*1*, *3*, *4*) **show** ‹$t \in \mathcal{D}\ (P\ ;_\checkmark\ Q\ ;_\checkmark\ R)$› **by** (*simp add*: *is-processT7*)
**next**
**case** *D-Q-R*
**from** *D-Q-R*(*3*) **consider** (*D-Q*) $u'\ v$ **where** ‹$u = map\ (ev \circ of\text{-}ev)\ u'\ @\ v$› ‹$u' \in \mathcal{D}\ (Q\ r)$› ‹*tF u'*› ‹*ftF v*›
| (*D-R*) $u'\ s\ v$ **where** ‹$u = map\ (ev \circ of\text{-}ev)\ u'\ @\ v$› ‹$u'\ @\ [\checkmark(s)] \in \mathcal{T}\ (Q\ r)$› ‹*tF u'*› ‹$v \in \mathcal{D}\ (R\ s)$›
**unfolding** $Seq_{ptick}$-*projs* **by** *blast*
**thus** ‹$t \in \mathcal{D}\ (P\ ;_\checkmark\ Q\ ;_\checkmark\ R)$›
**proof** *cases*

**case** *D-Q*
**define** $t'' :: \langle('a, 's)\ trace_{ptick}\rangle$ **where** $\langle t'' = map\ (ev \circ of\text{-}ev)\ t' @ map\ (ev \circ of\text{-}ev)\ u'\rangle$
**have** $\langle t'' \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q)\rangle$
**by** (*simp add*: *t''-def Seq$_{ptick}$-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** $\langle tF\ t''\rangle$ **by** (*simp add*: *t''-def*)
**ultimately have** $\langle map\ (ev \circ of\text{-}ev)\ t'' \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)\rangle$
**by** (*simp add*: *Seq$_{ptick}$-projs*[*of* $\langle P\ ;_{\checkmark}\ Q\rangle$])
(*metis append.right-neutral front-tickFree-Nil*)
**also have** $\langle map\ (ev \circ of\text{-}ev)\ t'' = map\ (ev \circ of\text{-}ev)\ t' @ map\ (ev \circ of\text{-}ev)\ u'\rangle$
**by** (*simp add*: *t''-def*)
**finally have** $\langle map\ (ev \circ of\text{-}ev)\ t' @ map\ (ev \circ of\text{-}ev)\ u' \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)\rangle$
.
**with** *D-Q(1,4) D-Q-R(1) is-processT7* **show** $\langle t \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)\rangle$ **by** *fastforce*
**next**
**case** *D-R*
**define** $t'' :: \langle('a, 's)\ trace_{ptick}\rangle$ **where** $\langle t'' = map\ (ev \circ of\text{-}ev)\ (map\ (ev \circ of\text{-}ev)\ t' @ u')\rangle$
**have** $\langle t'' @ [\checkmark(s)] \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)\rangle$
**by** (*simp add*: *t''-def Seq$_{ptick}$-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** $\langle tF\ t''\rangle$ **unfolding** *t''-def* **by** (*blast intro*: *tickFree-map-ev-comp*)

**ultimately have** $\langle map\ (ev \circ of\text{-}ev)\ t'' @ v \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)\rangle$
**unfolding** *Seq$_{ptick}$-projs*[*of* $\langle P\ ;_{\checkmark}\ Q\rangle$] **using** *D-R(4)* **by** *blast*
**also have** $\langle map\ (ev \circ of\text{-}ev)\ t'' @ v = t\rangle$
**by** (*simp add*: *D-Q-R(1) D-R(1) t''-def*)
**finally show** $\langle t \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)\rangle$ .
**qed**
**qed**
**next**

**fix** *t* **assume** $\langle t \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)\rangle$
**then consider** (*D-P-Q*) *t' u* **where** $\langle t = map\ (ev \circ of\text{-}ev)\ t' @ u\rangle$ $\langle t' \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q)\rangle$ $\langle tF\ t'\rangle$ $\langle ftF\ u\rangle$
| (*D-R*) *t' s u* **where** $\langle t = map\ (ev \circ of\text{-}ev)\ t' @ u\rangle$ $\langle t' @ [\checkmark(s)] \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)\rangle$ $\langle u \in \mathcal{D}\ (R\ s)\rangle$
**unfolding** *Seq$_{ptick}$-projs*[*of* $\langle P\ ;_{\checkmark}\ Q\rangle$] **by** *blast*
**thus** $\langle t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))\rangle$
**proof** *cases*
**case** *D-P-Q*
**from** *D-P-Q(2)* **consider** (*D-P*) *t'' u'* **where** $\langle t' = map\ (ev \circ of\text{-}ev)\ t'' @ u'\rangle$ $\langle t'' \in \mathcal{D}\ P\rangle$ $\langle tF\ t''\rangle$ $\langle ftF\ u'\rangle$
| (*D-Q*) *t'' r u'* **where** $\langle t' = map\ (ev \circ of\text{-}ev)\ t'' @ u'\rangle$ $\langle t'' @ [\checkmark(r)] \in \mathcal{T}\ P\rangle$ $\langle tF\ t''\rangle$ $\langle u' \in \mathcal{D}\ (Q\ r)\rangle$

**unfolding** $Seq_{ptick}$*-projs* **by** *blast*
**thus** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**proof** *cases*
**case** *D-P*
**from** *D-P*(*2*, *3*) **have** ‹*map* (*ev* ∘ *of-ev*) $t'' \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*[*of P*] *intro*: *front-tickFree-Nil*)
**thus** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**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*) $u' \in \mathcal{D}\ (Q\ r\ ;_{\checkmark}\ R)$›
**by** (*simp add*: $Seq_{ptick}$*-projs*) (*metis append.right-neutral front-tickFree-Nil*)
**with** *D-Q*(*2*, *3*) **have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*ev* ∘ *of-ev*) $t'' @ u'$) $\in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*[*of P*])
**with** *D-P-Q*(*4*) *is-processT7* **show** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*fastforce simp add*: *D-P-Q*(*1*) *D-Q*(*1*))
**qed**
**next**
**case** *D-R*
**then consider** (*T-P*) *t'' r u'* **where** ‹$t' = map\ (ev \circ of\text{-}ev)\ t'' @ u'$›
‹$t'' @ [\checkmark(r)] \in \mathcal{T}\ P$› ‹*tF t''*› ‹$u' @ [\checkmark(s)] \in \mathcal{T}\ (Q\ r)$›
| (*D-P*) *t'' u'* **where** ‹$t' = map\ (ev \circ of\text{-}ev)\ t'' @ u'$› ‹$t'' \in \mathcal{D}\ P$› ‹*tF t''*› ‹*tF u'*›
**by** (*auto simp add*: $Seq_{ptick}$*-projs append-eq-append-conv2 append-eq-map-conv*
*Cons-eq-append-conv front-tickFree-append-iff intro*: *D-P T-P*)
**thus** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**proof** *cases*
**case** *T-P*
**from** *D-R*(*3*) *T-P*(*4*) **have** ‹*map* (*ev* ∘ *of-ev*) $u' @ u \in \mathcal{D}\ (Q\ r\ ;_{\checkmark}\ R)$›
**by** (*simp add*: $Seq_{ptick}$*-projs*) (*metis append-T-imp-tickFree not-Cons-self2*)
**with** *T-P*(*2*, *3*) **have** ‹*map* (*ev* ∘ *of-ev*) $t'' @$ *map* (*ev* ∘ *of-ev*) $u' @ u \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*auto simp add*: $Seq_{ptick}$*-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** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$› **.**
**next**
**case** *D-P*
**from** *D-P*(*2*, *3*) **have** ‹*map* (*ev* ∘ *of-ev*) $t'' \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*[*of P*] *intro*: *front-tickFree-Nil*)
**with** *D-R*(*3*) **show** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**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** ‹$(t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$› ‹$t \notin \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**then consider** (*F-P*) $t'$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'$› ‹$(t',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ P$› ‹$tF\ t'$›
| (*F-Q-R*) $t'\ r\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t' \mathbin{@} u$› ‹$t' \mathbin{@} [\checkmark(r)] \in \mathcal{T}\ P$›
‹$tF\ t'$› ‹$(u,\ X) \in \mathcal{F}\ (Q\ r\ ;_{\checkmark}\ R)$›
**unfolding** $Seq_{ptick}$*-projs*[*of P*] **by** *fast*
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)$›
**proof** *cases*
**case** *F-P*
**from** *F-P*(*2*) **have** ‹$(t',\ ref\text{-}Seq_{ptick}\ (ref\text{-}Seq_{ptick}\ X)) \in \mathcal{F}\ P$›
**by** (*simp add*: $ref\text{-}Seq_{ptick}$*-idem*)
**with** *F-P*(*3*) **have** ‹$(map\ (ev \circ of\text{-}ev)\ t',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*)
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)$›
**by** (*simp add*: $Seq_{ptick}$*-projs*[*of* ‹$P\ ;_{\checkmark}\ Q$›] *F-P*(*1*))
(*metis map-ev-of-ev-map-ev-of-ev tickFree-map-ev-comp*)
**next**
**case** *F-Q-R*
**from** *F-Q-R*(*4*)
**consider** (*F-Q*) $u'$ **where** ‹$u = map\ (ev \circ of\text{-}ev)\ u'$›
‹$(u',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ (Q\ r)$› ‹$tF\ u'$›
| (*F-R*) $u'\ s\ u''$ **where** ‹$u = map\ (ev \circ of\text{-}ev)\ u' \mathbin{@} u''$› ‹$u' \mathbin{@} [\checkmark(s)] \in \mathcal{T}\ (Q\ r)$› ‹$tF\ u'$› ‹$(u'',\ X) \in \mathcal{F}\ (R\ s)$›
| (*D-Q*) $u'\ u''$ **where** ‹$u = map\ (ev \circ of\text{-}ev)\ u' \mathbin{@} u''$› ‹$u' \in \mathcal{D}\ (Q\ r)$› ‹$tF\ u'$› ‹$ftF\ u''$›
**unfolding** $Seq_{ptick}$*-projs* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)$›
**proof** *cases*
**case** *F-Q*
**from** *F-Q*(*2*) *F-Q-R*(*2*, *3*)
**have** ‹$(map\ (ev \circ of\text{-}ev)\ t' \mathbin{@} u', ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*)
**with** *F-Q*(*1*,*3*) *F-Q-R*(*1*) **show** ‹$(t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)$›
**by** (*simp add*: $Seq_{ptick}$*-projs*[*of* ‹$P\ ;_{\checkmark}\ Q$›])
(*metis map-append map-ev-of-ev-map-ev-of-ev tickFree-append-iff tickFree-map-ev-comp*)
**next**
**case** *F-R*
**from** *F-Q-R*(*2*, *3*) *F-R*(*2*) **have** $ : ‹$map\ (ev \circ of\text{-}ev)\ t' \mathbin{@} u' \mathbin{@} [\checkmark(s)] \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*)
**have** $$ : ‹$tF\ (map\ (ev \circ of\text{-}ev)\ t' \mathbin{@} u')$› **by** (*simp add*: *F-R*(*3*))
**show** ‹$(t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q\ ;_{\checkmark}\ R)$›
**by** (*simp add*: $Seq_{ptick}$*-projs*[*of* ‹$P\ ;_{\checkmark}\ Q$›])
(*metis map-append*[*of* ‹$ev \circ of\text{-}ev$› ‹$map\ (ev \circ of\text{-}ev)\ t'$› $u'$]
*F-Q-R*(*1*) *F-R*(*1*, *4*) $ $$ *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 $ : ‹map (ev ∘ of-ev) t' @ u' ∈ 𝒟 (P ;✓ Q)›
       by (auto simp add: Seq_ptick-projs)
      have $$ : ‹tF (map (ev ∘ of-ev) t' @ u')› by (simp add: D-Q(3))
      have ‹t ∈ 𝒟 (P ;✓ Q ;✓ R)›
       by (simp add: Seq_ptick-projs[of ‹P ;✓ Q›])
      (metis D-Q(1, 4) F-Q-R(1) $ $$ append-assoc map-append map-ev-of-ev-map-ev-of-ev)
      thus ‹(t, X) ∈ ℱ (P ;✓ Q ;✓ R)› by (fact is-processT8)
    qed
  qed
next
  fix t X assume ‹(t, X) ∈ ℱ (P ;✓ Q ;✓ R)› ‹t ∉ 𝒟 (P ;✓ Q ;✓ R)›
  then consider (F-P-Q) t' where ‹t = map (ev ∘ of-ev) t'›
    ‹(t', ref-Seq_ptick X) ∈ ℱ (P ;✓ Q)› ‹tF t'›
  | (F-R) t' s u where ‹t = map (ev ∘ of-ev) t' @ u›
    ‹t' @ [✓(s)] ∈ 𝒯 (P ;✓ Q)› ‹tF t'› ‹(u, X) ∈ ℱ (R s)›
    unfolding Seq_ptick-projs[of ‹P ;✓ Q›] by fast
  thus ‹(t, X) ∈ ℱ (P ;✓ (λr. Q r ;✓ R))›
  proof cases
    case F-P-Q
    from F-P-Q(1, 3) ‹t ∉ 𝒟 (P ;✓ Q ;✓ R)› have ‹t' ∉ 𝒟 (P ;✓ Q)›
      by (simp add: Seq_ptick-projs) (metis front-tickFree-Nil self-append-conv)
    with F-P-Q(2) consider (F-P) t'' where ‹t' = map (ev ∘ of-ev) t''›
      ‹(t'', ref-Seq_ptick (ref-Seq_ptick X :: ('a, 's) refusal_ptick)) ∈ ℱ P› ‹tF t''›
    | (F-Q) t'' r u where ‹t' = map (ev ∘ of-ev) t'' @ u› ‹t'' @ [✓(r)] ∈ 𝒯 P› ‹tF
t''›
      ‹(u, ref-Seq_ptick X) ∈ ℱ (Q r)›
      unfolding Seq_ptick-projs by fast
    thus ‹(t, X) ∈ ℱ (P ;✓ (λr. Q r ;✓ R))›
    proof cases
      case F-P
      from F-P(2, 3) have ‹(t'',ref-Seq_ptick X) ∈ ℱ P›
        by (simp add: ref-Seq_ptick-idem)
      moreover have ‹t = map (ev ∘ of-ev) t''›
        by (simp add: F-P-Q(1) F-P(1))
      ultimately show ‹(t, X) ∈ ℱ (P ;✓ (λr. Q r ;✓ R))›
        by (auto simp add: Seq_ptick-projs[of P] F-P(3))
    next
      case F-Q
      from F-P-Q(3) F-Q(1, 4) have ‹(map (ev ∘ of-ev) u, X) ∈ ℱ (Q r ;✓ R)›
        by (auto simp add: Seq_ptick-projs)
      with F-Q(2, 3) show ‹(t, X) ∈ ℱ (P ;✓ (λr. Q r ;✓ R))›
        by (auto simp add: Seq_ptick-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 ‹t' = map (ev ∘ of-ev) t'' @ u'›
      ‹t'' @ [✓(r)] ∈ 𝒯 P› ‹tF t''› ‹u' @ [✓(s)] ∈ 𝒯 (Q r)›
```

| (*D-P*) $t''$ $u'$ **where** ‹$t' = map\ (ev \circ of\text{-}ev)\ t'' @ u'$› ‹$t'' \in \mathcal{D}\ P$› ‹$tF\ t''$› ‹$tF\ u'$›
**by** (*auto simp add*: $Seq_{ptick}$*-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** ‹$(t, X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**proof** *cases*
**case** *T-P*
**with** *F-R*(*3*, *4*) *T-P*(*1*, *4*) **have** ‹$(map\ (ev \circ of\text{-}ev)\ u' @ u, X) \in \mathcal{F}\ (Q\ r\ ;_{\checkmark}\ R)$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*)
**with** *T-P*(*2*, *3*) **show** ‹$(t, X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs*[*of P*] *F-R*(*1*) *T-P*(*1*))
**next**
**case** *D-P*
**with** *D-P*(*2*,*3*) *F-R*(*4*) **have** ‹$t \in \mathcal{D}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$›
**by** (*simp add*: $Seq_{ptick}$*-projs F-R*(*1*) *D-P*(*1*))
(*metis F-imp-front-tickFree front-tickFree-append tickFree-map-ev-comp*)
**thus** ‹$(t, X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ (\lambda r.\ Q\ r\ ;_{\checkmark}\ R))$› **by** (*fact is-processT8*)
**qed**
**qed**
**qed**

## 7.3 Distributivity of Non-Determinism

### 7.3.1 Sequential Composition

**lemma** $Seq_{ptick}$*-distrib-GlobalNdet-left* :
‹$P\ ;_{\checkmark}\ (\lambda r.\ \sqcap\ a{\in}A.\ Q\ a\ r) = (if\ A = \{\}\ then\ P\ ;_{\checkmark}\ (\lambda r.\ STOP)\ else\ \sqcap\ a{\in}A.\ (P\ ;_{\checkmark}\ Q\ a))$›
**by** *simp* (*auto simp add*: *Process-eq-spec GlobalNdet-projs STOP-projs* $Seq_{ptick}$*-projs*)

**lemma** $Seq_{ptick}$*-distrib-GlobalNdet-right* : ‹$(\sqcap\ a{\in}A.\ P\ a)\ ;_{\checkmark}\ Q = \sqcap\ a{\in}A.\ (P\ a\ ;_{\checkmark}\ Q)$›
**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* : ‹$P\ ;_{\checkmark}\ (\lambda r.\ Q\ r \sqcap R\ r) = (P\ ;_{\checkmark}\ Q) \sqcap (P\ ;_{\checkmark}\ R)$›
**by** (*fact* $Seq_{ptick}$*-distrib-GlobalNdet-left*[*of P* ‹$\{0 :: nat, 1\}$›
‹$\lambda n.\ if\ n = 0\ then\ Q\ else\ if\ n = 1\ then\ R\ else\ undefined$›,
*simplified GlobalNdet-distrib-unit-bis*, *simplified*])

**lemma** *Seq$_{ptick}$-distrib-Ndet-right* : ‹$P \sqcap Q \;;_{\checkmark} R = (P \;;_{\checkmark} R) \sqcap (Q \;;_{\checkmark} R)$›
**by** (*fact Seq$_{ptick}$-distrib-GlobalNdet-right*[*of* ‹{*0* :: *nat*, *1*}›
‹λ*n*. *if n = 0 then P else if n = 1 then Q else undefined*› *R*,
*simplified GlobalNdet-distrib-unit-bis*, *simplified*])

### 7.3.2 Synchronization Product

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

**lemma** *Sync$_{ptick}$-distrib-GlobalNdet-left* :
‹$P \;[\![S]\!]_{\checkmark} \sqcap a{\in}A.\ Q\ a = (\mathit{if}\ A = \{\}\ \mathit{then}\ P\ [\![S]\!]_{\checkmark}\ \mathit{STOP}\ \mathit{else} \sqcap a{\in}A.\ (P\ [\![S]\!]_{\checkmark}\ Q\ a))$›
(**is** ‹*?lhs* = (*if A* = {} *then P* $[\![S]\!]_{\checkmark}$ *STOP else ?rhs*)›)
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*A* = {} ⟹ *?lhs* = *P* $[\![S]\!]_{\checkmark}$ *STOP*›
**by** (*simp add*: *GlobalNdet.abs-eq STOP.abs-eq*)
**next**
**show** ‹*?lhs* = *?rhs*› **if** ‹*A* ≠ {}›
**proof** (*subst Process-eq-spec-optimized*, *safe*)
**show** ‹*t* ∈ $\mathcal{D}$ *?lhs* ⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **for** *t*
**by** (*simp add*: *D-Sync$_{ptick}$ GlobalNdet-projs*)
(*metis* ‹*A* ≠ {}› *ex-in-conv is-processT1-TR*)
**next**
**show** ‹*t* ∈ $\mathcal{D}$ *?rhs* ⟹ *t* ∈ $\mathcal{D}$ *?lhs*› **for** *t*
**by** (*simp add*: *GlobalNdet-projs D-Sync$_{ptick}$*) *blast*
**next**
**assume** *same-div* : ‹$\mathcal{D}$ *?lhs* = $\mathcal{D}$ *?rhs*›
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**with** ‹*A* ≠ {}› **consider** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
| (*fail*) *t-P t-Q X-P X-Q a* **where**
‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹*a* ∈ *A*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ (*Q a*)›
‹*t setinterleaves*$_{\checkmark}$ *tick-join* ((*t-P*, *t-Q*), *S*)›
‹*X* ⊆ *super-ref-Sync$_{ptick}$ tick-join X-P S X-Q*›
**unfolding** *Sync$_{ptick}$-projs F-GlobalNdet* **by** *force*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**proof** *cases*
**from** *same-div D-F* **show** ‹*t* ∈ $\mathcal{D}$ *?lhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› **by** *blast*
**next**
**case** *fail*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› **by** (*simp add*: *F-GlobalNdet F-Sync$_{ptick}$*) *blast*
**qed**
**next**
**show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **for** *t X*
**by** (*simp add*: *GlobalNdet-projs F-Sync$_{ptick}$* ‹*A* ≠ {}›) *blast*
**qed**
**qed**

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

$Q))\rangle$
(**is** $\langle ?lhs = (if\ A = \{\}\ then\ STOP\ [\![S]\!]_{\checkmark}\ Q\ else\ ?rhs)\rangle$)
**proof** (*split if-split, intro conjI impI*)
**show** $\langle A = \{\} \Longrightarrow ?lhs = STOP\ [\![S]\!]_{\checkmark}\ Q\rangle$
**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** $\langle t \in \mathcal{D}\ ?lhs \Longrightarrow t \in \mathcal{D}\ ?rhs\rangle$ **for** *t*
**by** (*simp add*: $D\text{-}Sync_{ptick}$ *GlobalNdet-projs*)
(*metis* $\langle A \neq \{\}\rangle$ *ex-in-conv is-processT1-TR*)
**next**
**show** $\langle t \in \mathcal{D}\ ?rhs \Longrightarrow t \in \mathcal{D}\ ?lhs\rangle$ **for** *t*
**by** (*simp add*: *GlobalNdet-projs* $D\text{-}Sync_{ptick}$) *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-P t-Q X-P X-Q a* **where**
$\langle (t\text{-}P, X\text{-}P) \in \mathcal{F}\ (P\ a)\rangle$ $\langle a \in A\rangle$ $\langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q\rangle$
$\langle t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P, t\text{-}Q), S)\rangle$
$\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick}\ tick\text{-}join\ X\text{-}P\ S\ X\text{-}Q\rangle$
**unfolding** $Sync_{ptick}$*-projs F-GlobalNdet* **by** *force*
**thus** $\langle (t, X) \in \mathcal{F}\ ?rhs\rangle$
**proof** *cases*
**from** *same-div D-F* **show** $\langle t \in \mathcal{D}\ ?lhs \Longrightarrow (t, X) \in \mathcal{F}\ ?rhs\rangle$ **by** *blast*
**next**
**case** *fail*
**thus** $\langle (t, X) \in \mathcal{F}\ ?rhs\rangle$ **by** (*simp add*: *F-GlobalNdet* $F\text{-}Sync_{ptick}$) *blast*
**qed**
**next**
**show** $\langle (t, X) \in \mathcal{F}\ ?rhs \Longrightarrow (t, X) \in \mathcal{F}\ ?lhs\rangle$ **for** *t X*
**by** (*simp add*: *GlobalNdet-projs* $F\text{-}Sync_{ptick}$ $\langle A \neq \{\}\rangle$) *blast*
**qed**
**qed**

**lemma** (**in** $Sync_{ptick}$*-locale*) $Sync_{ptick}$*-GlobalNdet-cartprod*:
$\langle (\sqcap\ (a, b) \in A \times B.\ (P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) =$
$(if\ A = \{\} \vee B = \{\}\ then\ STOP\ else\ (\sqcap a \in A.\ P\ a)\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B.\ Q\ b))\rangle$
**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* :
$\langle P\ [\![S]\!]_{\checkmark}\ Q \sqcap R = (P\ [\![S]\!]_{\checkmark}\ Q) \sqcap (P\ [\![S]\!]_{\checkmark}\ R)\rangle$
**by** (*rule trans*[*OF trans*[*OF - Sync*$_{ptick}$*-distrib-GlobalNdet-left*
[*of P S* $\langle \{True, False\}\rangle$ $\langle \lambda b.\ if\ b\ then\ Q\ else\ R\rangle$]]])
(*simp-all add*: *GlobalNdet-distrib-unit*)

**corollary** *Sync$_{ptick}$-distrib-Ndet-right* :
‹$P \sqcap Q \ [\![S]\!]_{\checkmark} \ R = (P \ [\![S]\!]_{\checkmark} \ R) \sqcap (Q \ [\![S]\!]_{\checkmark} \ R)$›
**by** (*rule trans*[*OF trans*[*OF* - *Sync$_{ptick}$-distrib-GlobalNdet-right*
[*of* ‹{*True*, *False*}› ‹$\lambda b$. *if b then P else Q*› *S R*]]])
(*simp-all add*: *GlobalNdet-distrib-unit*)

**end**

# Chapter 8

# Communications

## 8.1 Step Laws

### 8.1.1 Sequential Composition

**lemma** *Mprefix-Seq$_{ptick}$*: ‹$\Box a \in A \rightarrow P\ a$ ;$_{\checkmark}$ $Q = \Box a \in A \rightarrow (P\ a$ ;$_{\checkmark}$ $Q)$› (**is** ‹*?lhs* = *?rhs*›)
**proof** (*rule Process-eq-optimizedI*)
  **show** ‹$t \in \mathcal{D}$ *?lhs* $\Longrightarrow$ $t \in \mathcal{D}$ *?rhs*› **for** $t$
   **by** (*cases t*, *auto simp add*: *Seq$_{ptick}$-projs Mprefix-projs image-iff Cons-eq-append-conv*) *blast*
**next**
  **show** ‹$t \in \mathcal{D}$ *?rhs* $\Longrightarrow$ $t \in \mathcal{D}$ *?lhs*› **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** ‹$(t, X) \in \mathcal{F}$ *?lhs*› ‹$t \notin \mathcal{D}$ *?lhs*›
  **then consider** (*F-P*) $t'$ **where** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$›
    ‹$(t'$, *ref-Seq$_{ptick}$* $X) \in \mathcal{F}$ ($\Box a \in A \rightarrow P\ a$)› ‹*tF t'*›
  | (*F-Q*) $t'$ $r$ $u$ **where** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$ @ $u$› ‹$t'$ @ [✓($r$)] $\in \mathcal{T}$ ($\Box a \in A \rightarrow P\ a$)› ‹*tF t'*› ‹$(u, X) \in \mathcal{F}$ ($Q\ r$)›
    **unfolding** *Seq$_{ptick}$-projs* **by** *fast*
  **thus** ‹$(t, X) \in \mathcal{F}$ *?rhs*›
  **proof** *cases*
    **case** *F-P* **thus** ‹$(t, X) \in \mathcal{F}$ *?rhs*›
     **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** ‹$(t, X) \in \mathcal{F}$ *?rhs*›
    **by** (*cases t*) (*auto simp add*: *Seq$_{ptick}$-projs Mprefix-projs Cons-eq-append-conv*)+
  **qed**
**next**

**fix** $t$ $X$ **assume** ‹$(t,\ X) \in \mathcal{F}$ *?rhs*› ‹$t \notin \mathcal{D}$ *?rhs*›
**from** ‹$(t,\ X) \in \mathcal{F}$ *?rhs*› **consider** ‹$t = []$› ‹$X \cap ev\ `\ A = \{\}$›
| $a\ t'$ **where** ‹$t = ev\ a\ \#\ t'$› ‹$a \in A$› ‹$(t',\ X) \in \mathcal{F}\ (P\ a\ ;_{\checkmark}\ Q)$›
**unfolding** *F-Mprefix* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›
**proof** *cases*
**show** ‹$t = [] \Longrightarrow X \cap ev\ `\ A = \{\} \Longrightarrow (t,\ X) \in \mathcal{F}$ *?lhs*›
**by** (*auto simp add*: $Seq_{ptick}$*-projs ref-*$Seq_{ptick}$*-def F-Mprefix*)
**next**
**fix** $a\ t'$ **assume** ‹$t = ev\ a\ \#\ t'$› ‹$a \in A$› ‹$(t',\ X) \in \mathcal{F}\ (P\ a\ ;_{\checkmark}\ Q)$›
**from** ‹$(t',\ X) \in \mathcal{F}\ (P\ a\ ;_{\checkmark}\ Q)$› ‹$t \notin \mathcal{D}$ *?rhs*› ‹$a \in A$› ‹$t = ev\ a\ \#\ t'$›
**consider** (*F-P*) $t''$ **where** ‹$t' = map\ (ev \circ of\text{-}ev)\ t''$› ‹$(t'',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ (P\ a)$› ‹$tF\ t''$›
| (*F-Q*) $t''\ r\ u$ **where** ‹$t' = map\ (ev \circ of\text{-}ev)\ t''\ @\ u$› ‹$t''\ @\ [\checkmark(r)] \in \mathcal{T}\ (P\ a)$› ‹$tF\ t''$› ‹$(u,\ X) \in \mathcal{F}\ (Q\ r)$›
**by** (*auto simp add*: $Seq_{ptick}$*-projs D-Mprefix*)
**thus** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›
**proof** *cases*
**case** *F-P* **thus** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›
**by** (*simp add*: *Mprefix-projs* $Seq_{ptick}$*-projs* ‹$t = ev\ a\ \#\ t'$› *Cons-eq-map-conv*)
(*metis* ‹$a \in A$› $event_{ptick}$*.disc*(1) $event_{ptick}$*.sel*(1) *tickFree-Cons-iff*)
**next**
**case** *F-Q* **thus** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›
**by** (*simp add*: *Mprefix-projs* $Seq_{ptick}$*-projs* ‹$t = ev\ a\ \#\ t'$› *Cons-eq-map-conv append-eq-Cons-conv*)
(*metis* (*no-types*, *lifting*) ‹$a \in A$› *append-Cons comp-apply* $event_{ptick}$*.disc*(1) $event_{ptick}$*.sel*(1) *list.simps*(9) *tickFree-Cons-iff*)
**qed**
**qed**
**qed**

### 8.1.2 Synchronization Product

**lemma** (**in** $Sync_{ptick}$*-locale*) *Mprefix-*$Sync_{ptick}$*-Mprefix-bis* :
‹$\Box a{\in}(A \cup A') \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}(B \cup B') \to Q\ b =$
$(\Box a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}(B \cup B') \to Q\ b)) \Box$
$(\Box b{\in}B \to (\Box a{\in}(A \cup A') \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
$(\Box x{\in}(A' \cap B') \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))$›
(**is** ‹*?lhs1* $[\![S]\!]_{\checkmark}$ *?lhs2* = *?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*›)
**if** *sets-assms*: ‹$A \cap S = \{\}$› ‹$A' \subseteq S$› ‹$B \cap S = \{\}$› ‹$B' \subseteq S$›
**proof** (*rule Process-eq-optimizedI*)
**fix** $t$ **assume** ‹$t \in \mathcal{D}$ (*?lhs1* $[\![S]\!]_{\checkmark}$ *?lhs2*)›
**then obtain** $u\ v$ *t-P t-Q*
**where** $* :$ ‹$t = u\ @\ v$› ‹$tF\ u$› ‹$ftF\ v$›
‹$u$ *setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*t-P*, *t-Q*), $S$)›
‹*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*›
**unfolding** *D-*$Sync_{ptick}$ **by** *blast*
**from** $*(5)$ **show** ‹$t \in \mathcal{D}$ (*?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*)›

**proof** (*elim disjE conjE*)
**assume** ‹*t-P* ∈ $\mathcal{D}$ *?lhs1*› ‹*t-Q* ∈ $\mathcal{T}$ *?lhs2*›
**from** ‹*t-P* ∈ $\mathcal{D}$ *?lhs1*› **obtain** *a t-P′*
**where** ** : ‹*a* ∈ *A* ∨ *a* ∈ *A′*› ‹*t-P* = *ev a* # *t-P′*› ‹*t-P′* ∈ $\mathcal{D}$ (*P a*)›
**unfolding** *D-Mprefix* **by** *blast*
**from** ‹*t-Q* ∈ $\mathcal{T}$ *?lhs2*› **consider** ‹*t-Q* = []›
| *b t-Q′* **where** ‹*b* ∈ *B* ∨ *b* ∈ *B′*› ‹*t-Q* = *ev b* # *t-Q′*› ‹*t-Q′* ∈ $\mathcal{T}$ (*Q b*)›
**unfolding** *T-Mprefix* **by** *blast*
**thus** ‹*t* ∈ $\mathcal{D}$ (*?rhs1* □ *?rhs2* □ *?rhs3*)›
**proof** *cases*
**assume** ‹*t-Q* = []›
**with** *(4)* **obtain** *u′* **where** ‹*a* ∉ *S*› ‹*u* = *ev a* # *u′*›
‹*u′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*t-P′*, *t-Q*), *S*)›
**by** (*auto simp add*: **(2) *split*: *if-split-asm*)
**moreover from** ‹*u* = *ev a* # *u′*› *(2) **have** ‹*tF u′*› **by** *simp*
**ultimately have** ‹*t* ∈ $\mathcal{D}$ *?rhs1*›
**using** *(1, 3) **(1, 3) ‹*t-Q* ∈ $\mathcal{T}$ *?lhs2*› ‹*A′* ⊆ *S*›
**by** (*auto simp add*: *simp add*: *D-Mprefix D-Sync*$_{ptick}$)
**thus** ‹*t* ∈ $\mathcal{D}$ (*?rhs1* □ *?rhs2* □ *?rhs3*)› **by** (*simp add*: *D-Det*)
**next**
**fix** *b t-Q′* **assume** *** : ‹*b* ∈ *B* ∨ *b* ∈ *B′*› ‹*t-Q* = *ev b* # *t-Q′*› ‹*t-Q′* ∈ $\mathcal{T}$ (*Q b*)›
**from** *(2) **have** $ : ‹*u* = *ev x* # *u′* ⟹ *tF u′*› **for** *x u′* **by** *simp*
**from** *(4) *sets-assms* **(1) ***(1)*
**consider** (*mv-both*) *u′* **where** ‹*a* ∈ *S*› ‹*b* = *a*› ‹*a* ∈ *A′*› ‹*a* ∈ *B′*› ‹*u* = *ev a* # *u′*›
‹*u′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*t-P′*, *t-Q′*), *S*)›
| (*mv-left*) *u′* **where** ‹*a* ∉ *S*› ‹*a* ∈ *A*› ‹*u* = *ev a* # *u′*›
‹*u′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*t-P′*, *t-Q*), *S*)›
| (*mv-right*) *u′* **where** ‹*b* ∉ *S*› ‹*b* ∈ *B*› ‹*u* = *ev b* # *u′*›
‹*u′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*t-P*, *t-Q′*), *S*)›
**by** (*auto simp add*: **(2) ***(2) *disjoint-iff*
*split*: *if-split-asm*)
**thus** ‹*t* ∈ $\mathcal{D}$ (*?rhs1* □ *?rhs2* □ *?rhs3*)›
**proof** *cases*
**case** *mv-both*
**have** ‹*tF u′*› **by** (*simp add*: $ *mv-both(5)*)
**with** *(3) **(3) ***(3) *mv-both(2, 6)*
**have** ‹*u′* @ *v* ∈ $\mathcal{D}$ (*P a* $[\![S]\!]_{✓}$ *Q a*)› **by** (*auto simp add*: *D-Sync*$_{ptick}$)
**hence** ‹*t* ∈ $\mathcal{D}$ *?rhs3*› **by** (*simp add*: *D-Mprefix *(1) mv-both(3−5)*)
**thus** ‹*t* ∈ $\mathcal{D}$ (*?rhs1* □ *?rhs2* □ *?rhs3*)› **by** (*simp add*: *D-Det*)
**next**
**case** *mv-left*
**have** ‹*tF u′*› **by** (*simp add*: $ *mv-left(3)*)
**with** *(3) **(3) ‹*t-Q* ∈ $\mathcal{T}$ *?lhs2*› *mv-left(4)*
**have** ‹*u′* @ *v* ∈ $\mathcal{D}$ (*P a* $[\![S]\!]_{✓}$ □*b*∈(*B* ∪ *B′*) → *Q b*)› **by** (*auto simp add*: *D-Sync*$_{ptick}$)
**hence** ‹*t* ∈ $\mathcal{D}$ *?rhs1*› **by** (*simp add*: *D-Mprefix *(1) mv-left(2, 3)*)
**thus** ‹*t* ∈ $\mathcal{D}$ (*?rhs1* □ *?rhs2* □ *?rhs3*)› **by** (*simp add*: *D-Det*)

```
    next
      case mv-right
      have ‹tF u'› by (simp add: $ mv-right(3))
      with ∗(3) ∗∗∗(3) mv-right(4) ‹t-P ∈ 𝒟 ?lhs1›
      have ‹u' @ v ∈ 𝒟 (□a∈(A ∪ A') → P a ⟦S⟧✓ Q b)›
        by (auto simp add: D-Sync_ptick)
      hence ‹t ∈ 𝒟 ?rhs2› by (simp add: D-Mprefix ∗(1) mv-right(2, 3))
      thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› by (simp add: D-Det)
    qed
  qed
next
  assume ‹t-P ∈ 𝒯 ?lhs1› ‹t-Q ∈ 𝒟 ?lhs2›
  from ‹t-Q ∈ 𝒟 ?lhs2› obtain b t-Q'
    where ∗∗ : ‹b ∈ B ∨ b ∈ B'› ‹t-Q = ev b # t-Q'› ‹t-Q' ∈ 𝒟 (Q b)›
    unfolding D-Mprefix by blast
  from ‹t-P ∈ 𝒯 ?lhs1› consider ‹t-P = []›
    | a t-P' where ‹a ∈ A ∨ a ∈ A'› ‹t-P = ev a # t-P'› ‹t-P' ∈ 𝒯 (P a)›
    unfolding T-Mprefix by blast
  thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)›
  proof cases
    assume ‹t-P = []›
    with ∗(4) obtain u' where ‹b ∉ S› ‹u = ev b # u'›
      ‹u' setinterleaves✓tick-join ((t-P, t-Q'), S)›
      by (auto simp add: ∗∗(2) split: if-split-asm)
    moreover from ‹u = ev b # u'› ‹tF u› have ‹tF u'› by simp
    ultimately have ‹t ∈ 𝒟 ?rhs2›
      using ∗(1, 3) ∗∗(1, 3) ‹t-P ∈ 𝒯 ?lhs1› ‹B' ⊆ S›
      by (auto simp add: simp add: D-Mprefix D-Sync_ptick)
    thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› by (simp add: D-Det)
  next
    fix a t-P' assume ∗∗∗ : ‹a ∈ A ∨ a ∈ A'› ‹t-P = ev a # t-P'› ‹t-P' ∈ 𝒯 (P
a)›
    from ‹tF u› have $ : ‹u = ev x # u' ⟹ tF u'› for x u' by simp
    from ∗(4) sets-assms ∗∗(1) ∗∗∗(1)
    consider (mv-both)  u' where ‹a ∈ S› ‹b = a› ‹a ∈ A'› ‹a ∈ B'›
      ‹u = ev a # u'› ‹u' setinterleaves✓tick-join ((t-P', t-Q'), S)›
    |        (mv-left)  u' where ‹a ∉ S› ‹a ∈ A› ‹u = ev a # u'›
      ‹u' setinterleaves✓tick-join ((t-P', t-Q), S)›
    |        (mv-right) u' where ‹b ∉ S› ‹b ∈ B› ‹u = ev b # u'›
      ‹u' setinterleaves✓tick-join ((t-P, t-Q'), S)›
      by (auto simp add: ∗∗(2) ∗∗∗(2) disjoint-iff split: if-split-asm)
    thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)›
    proof cases
      case mv-both
      have ‹tF u'› by (simp add: $ mv-both(5))
      with ∗(3) ∗∗(3) ∗∗∗(3) mv-both(2, 6)
      have ‹u' @ v ∈ 𝒟 (P a ⟦S⟧✓ Q a)› by (auto simp add: D-Sync_ptick)
      hence ‹t ∈ 𝒟 ?rhs3› by (simp add: D-Mprefix ∗(1) mv-both(3−5))
      thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› by (simp add: D-Det)
```

```
      next
        case mv-left
        have ‹tF u'› by (simp add: $ mv-left(3))
        with ∗(3) ∗∗∗(3) mv-left(4) ‹t-Q ∈ 𝒟 ?lhs2›
         have ‹u' @ v ∈ 𝒟 (P a ⟦S⟧✓ □b∈(B ∪ B') → Q b)› by (auto simp add:
D-Sync_ptick)
        hence ‹t ∈ 𝒟 ?rhs1› by (simp add: D-Mprefix ∗(1) mv-left(2, 3))
        thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› by (simp add: D-Det)
      next
        case mv-right
        have ‹tF u'› by (simp add: $ mv-right(3))
        with ∗(3) ∗∗(3) ‹t-P ∈ 𝒯 ?lhs1› mv-right(4)
        have ‹u' @ v ∈ 𝒟 (□a∈(A ∪ A') → P a ⟦S⟧✓ Q b)›
          by (auto simp add: D-Sync_ptick)
        hence ‹t ∈ 𝒟 ?rhs2› by (simp add: D-Mprefix ∗(1) mv-right(2, 3))
        thus ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› by (simp add: D-Det)
      qed
    qed
  qed
next

  fix t assume ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)›
  consider ‹t = []› | r-s t' where ‹t = ✓(r-s) # t'› | x t' where ‹t = ev x # t'›
    by (metis event_ptick.exhaust neq-Nil-conv)
  thus ‹t ∈ 𝒟 (?lhs1 ⟦S⟧✓ ?lhs2)›
  proof cases
    assume ‹t = []›
    with ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› have False
      by (simp add: D-Det D-Mprefix)
    thus ‹t ∈ 𝒟 (?lhs1 ⟦S⟧✓ ?lhs2)› ..
  next
    fix r-s t' assume ‹t = ✓(r-s) # t'›
    with ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› have False
      by (simp add: D-Det D-Mprefix)
    thus ‹t ∈ 𝒟 (?lhs1 ⟦S⟧✓ ?lhs2)› ..
  next
    fix x t' assume ‹t = ev x # t'›
    with ‹t ∈ 𝒟 (?rhs1 □ ?rhs2 □ ?rhs3)› consider
      (mv-left)  ‹x ∈ A› ‹t' ∈ 𝒟 (P x ⟦S⟧✓ ?lhs2)›
      | (mv-right) ‹x ∈ B› ‹t' ∈ 𝒟 (?lhs1 ⟦S⟧✓ Q x)›
      | (mv-both)  ‹x ∈ A'› ‹x ∈ B'› ‹t' ∈ 𝒟 (P x ⟦S⟧✓ Q x)›
      by (auto simp add: D-Det D-Mprefix)
    thus ‹t ∈ 𝒟 (?lhs1 ⟦S⟧✓ ?lhs2)›
    proof cases
      case mv-left
      from ‹x ∈ A› ‹A ∩ S = {}› have ‹x ∉ S› by blast
      from mv-left(2) obtain u v t-P t-Q
        where ∗ : ‹t' = u @ v› ‹tF u› ‹ftF v›
          ‹u setinterleaves✓ tick-join ((t-P, t-Q), S)›
```

‹t-P ∈ 𝒟 (P x) ∧ t-Q ∈ 𝒯 ?lhs2 ∨
t-P ∈ 𝒯 (P x) ∧ t-Q ∈ 𝒟 ?lhs2›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**have** ‹t = (ev x # u) @ v› **by** (*simp add*: ∗(*1*) ‹t = ev x # t'›)
**moreover have** ‹tF (ev x # u)› **by** (*simp add*: ∗(*2*))
**moreover from** ∗(*4*) **have** ‹ev x # u *setinterleaves*$_{✓\,tick\text{-}join}$ ((ev x # t-P, t-Q), S)›
**by** (*cases t-Q*) (*auto simp add*: ‹x ∉ S› *setinterleaving*$_{ptick}$*-simps split*: *event*$_{ptick}$*.split*)
**moreover from** ∗(*5*) *mv-left*(*1*)
**have** ‹ev x # t-P ∈ 𝒟 ?lhs1 ∧ t-Q ∈ 𝒯 ?lhs2 ∨
ev x # t-P ∈ 𝒯 ?lhs1 ∧ t-Q ∈ 𝒟 ?lhs2› **by** (*simp add*: *Mprefix-projs*)
**ultimately show** ‹t ∈ 𝒟 (?lhs1 ⟦S⟧$_{✓}$ ?lhs2)›
**using** ∗(*3*) **by** (*simp* (*no-asm*) *add*: *D-Sync*$_{ptick}$) *blast*
**next**
**case** *mv-right*
**from** ‹x ∈ B› ‹B ∩ S = {}› **have** ‹x ∉ S› **by** *blast*
**from** *mv-right*(*2*) **obtain** u v t-P t-Q
**where** ∗ : ‹t' = u @ v› ‹tF u› ‹ftF v›
‹u *setinterleaves*$_{✓\,tick\text{-}join}$ ((t-P, t-Q), S)›
‹t-P ∈ 𝒟 ?lhs1 ∧ t-Q ∈ 𝒯 (Q x) ∨
t-P ∈ 𝒯 ?lhs1 ∧ t-Q ∈ 𝒟 (Q x)›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**have** ‹t = (ev x # u) @ v› **by** (*simp add*: ∗(*1*) ‹t = ev x # t'›)
**moreover have** ‹tF (ev x # u)› **by** (*simp add*: ∗(*2*))
**moreover from** ∗(*4*) **have** ‹ev x # u *setinterleaves*$_{✓\,tick\text{-}join}$ ((t-P, ev x # t-Q), S)›
**by** (*cases t-P*) (*auto simp add*: ‹x ∉ S› *setinterleaving*$_{ptick}$*-simps split*: *event*$_{ptick}$*.split*)
**moreover from** ∗(*5*) *mv-right*(*1*)
**have** ‹t-P ∈ 𝒟 ?lhs1 ∧ ev x # t-Q ∈ 𝒯 ?lhs2 ∨
t-P ∈ 𝒯 ?lhs1 ∧ ev x # t-Q ∈ 𝒟 ?lhs2› **by** (*simp add*: *Mprefix-projs*)
**ultimately show** ‹t ∈ 𝒟 (?lhs1 ⟦S⟧$_{✓}$ ?lhs2)›
**using** ∗(*3*) **by** (*simp* (*no-asm*) *add*: *D-Sync*$_{ptick}$) *blast*
**next**
**case** *mv-both*
**from** ‹x ∈ A'› ‹A' ⊆ S› **have** ‹x ∈ S› **by** *blast*
**from** *mv-both*(*3*) **obtain** u v t-P t-Q
**where** ∗ : ‹t' = u @ v› ‹tF u› ‹ftF v›
‹u *setinterleaves*$_{✓\,tick\text{-}join}$ ((t-P, t-Q), S)›
‹t-P ∈ 𝒟 (P x) ∧ t-Q ∈ 𝒯 (Q x) ∨
t-P ∈ 𝒯 (P x) ∧ t-Q ∈ 𝒟 (Q x)›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**have** ‹t = (ev x # u) @ v› **by** (*simp add*: ∗(*1*) ‹t = ev x # t'›)
**moreover have** ‹tF (ev x # u)› **by** (*simp add*: ∗(*2*))
**moreover from** ∗(*4*) **have** ‹ev x # u *setinterleaves*$_{✓\,tick\text{-}join}$ ((ev x # t-P, ev x # t-Q), S)›
**by** (*auto simp add*: ‹x ∈ S›)
**moreover from** ∗(*5*) *mv-both*(*1*, *2*)

```
    have ‹ev x # t-P ∈ 𝒟 ?lhs1 ∧ ev x # t-Q ∈ 𝒯 ?lhs2 ∨
      ev x # t-P ∈ 𝒯 ?lhs1 ∧ ev x # t-Q ∈ 𝒟 ?lhs2› by (simp add: Mprefix-projs)
    ultimately show ‹t ∈ 𝒟 (?lhs1 ⟦S⟧✓ ?lhs2)›
     using ∗(3) by (simp (no-asm) add: D-Sync_ptick) blast
  qed
 qed
next

 fix t X assume ‹(t, X) ∈ ℱ (?lhs1 ⟦S⟧✓ ?lhs2)› ‹t ∉ 𝒟 (?lhs1 ⟦S⟧✓ ?lhs2)›
 then obtain t-P t-Q X-P X-Q
  where fail : ‹(t-P, X-P) ∈ ℱ ?lhs1› ‹(t-Q, X-Q) ∈ ℱ ?lhs2›
   ‹t setinterleaves✓tick-join ((t-P, t-Q), S)›
   ‹X ⊆ super-ref-Sync_ptick tick-join X-P S X-Q›
  unfolding Sync_ptick-projs by blast
 consider ‹t = []› | r-s t' where ‹t = ✓(r-s) # t'› | a t' where ‹t = ev a # t'›
  by (metis event_ptick.exhaust neq-Nil-conv)
 thus ‹(t, X) ∈ ℱ (?rhs1 □ ?rhs2 □ ?rhs3)›
 proof cases
  assume ‹t = []›
  with Nil-setinterleaves_ptick fail(3) have ‹t-P = []› ‹t-Q = []› by blast+
  with fail(1, 2) have ‹X-P ∩ ev ‘ (A ∪ A') = {}› ‹X-Q ∩ ev ‘ (B ∪ B') = {}›
   by (simp-all add: F-Mprefix)
  with fail(4) ‹A ∩ S = {}› ‹B ∩ S = {}› show ‹(t, X) ∈ ℱ (?rhs1 □ ?rhs2 □
?rhs3)›
   by (simp add: ‹t = []› Det-projs Mprefix-projs super-ref-Sync_ptick-def)
    (use event_ptick.distinct(1) in blast)
 next
  fix r-s t' assume ‹t = ✓(r-s) # t'›
  hence ‹t = [✓(r-s)]›
   by (metis F-imp-front-tickFree ‹(t, X) ∈ ℱ (?lhs1 ⟦S⟧✓ ?lhs2)›
     event_ptick.disc(2) front-tickFree-Cons-iff)
  with fail(3) obtain r s where ‹tick-join r s = Some r-s›
   by (auto elim: Cons-tick-setinterleaves_ptick E)
  from ‹t = [✓(r-s)]› fail(3) ‹tick-join r s = Some r-s›
  have ‹t-P = [✓(r)]›
   by (auto dest: inj-tick-join Nil-setinterleaves_ptick
     elim: Cons-tick-setinterleaves_ptick E)
  with fail(1) ‹t = [✓(r-s)]› have False by (simp add: F-Mprefix)
  thus ‹(t, X) ∈ ℱ (?rhs1 □ ?rhs2 □ ?rhs3)› ..
 next
  fix a t' assume ‹t = ev a # t'›
  from fail(1−3) sets-assms consider
   (mv-left) t-P' where
   ‹a ∉ S› ‹a ∈ A› ‹t-P = ev a # t-P'› ‹(t-P', X-P) ∈ ℱ (P a)›
   ‹t' setinterleaves✓tick-join ((t-P', t-Q), S)›
  | (mv-right) t-Q' where
   ‹a ∉ S› ‹a ∈ B› ‹t-Q = ev a # t-Q'› ‹(t-Q', X-Q) ∈ ℱ (Q a)›
   ‹t' setinterleaves✓tick-join ((t-P, t-Q'), S)›
  | (mv-both) t-P' t-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' setinterleaves✓tick-join
((t-P', t-Q'), S)›
    by (unfold ‹t = ev a # t'›, elim Cons-ev-setinterleaves_ptick E)
      (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-Sync_ptick ‹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-Sync_ptick ‹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-Sync_ptick ‹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 event_ptick.exhaust neq-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-Sync_ptick tick-join X-P S X-Q›
      by (simp add: super-ref-Sync_ptick-def X-P-def X-Q-def
          subset-iff disjoint-iff image-iff)
        (metis IntI event_ptick.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*: ‹$t = []$› *F-Sync*$_{ptick}$) (*use Nil-setinterleaving*$_{ptick}$*-Nil* **in** *blast*)
**next**
**fix** *r-s t′* **assume** ‹$t = ✓(r\text{-}s) \# t'$›
**with** ‹$(t, X) \in \mathcal{F}$ (*?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*)›
**have** *False* **by** (*simp add*: *F-Det F-Mprefix*)
**thus** ‹$(t, X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ *?lhs2*)› **..**
**next**
**fix** *x t′* **assume** ‹$t = ev\ x \# t'$›
**with** ‹$(t, X) \in \mathcal{F}$ (*?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*)›
**consider** (*mv-left*) ‹$x \in A$› ‹$(t', X) \in \mathcal{F}$ ($P\ x$ $[\![S]\!]_✓$ *?lhs2*)›
| (*mv-right*) ‹$x \in B$› ‹$(t', X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ $Q\ x$)›
| (*mv-both*) ‹$x \in A'$› ‹$x \in B'$› ‹$(t', X) \in \mathcal{F}$ ($P\ x$ $[\![S]\!]_✓$ $Q\ x$)›
**unfolding** *F-Det F-Mprefix* **by** *auto*
**thus** ‹$(t, X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ *?lhs2*)›
**proof** *cases*
**case** *mv-left*
**from** *mv-left*(*2*) **consider** ‹$t' \in \mathcal{D}$ ($P\ x$ $[\![S]\!]_✓$ *?lhs2*)›
| (*fail*) *t-P t-Q X-P X-Q* **where**
‹$(t\text{-}P, X\text{-}P) \in \mathcal{F}$ ($P\ x$)› ‹$(t\text{-}Q, X\text{-}Q) \in \mathcal{F}$ *?lhs2*›
‹$t'$ *setinterleaves*$_{✓\,tick\text{-}join}$ $((t\text{-}P, t\text{-}Q), S)$›
‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ *tick-join X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**thus** ‹$(t, X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ *?lhs2*)›
**proof** *cases*
**assume** ‹$t' \in \mathcal{D}$ ($P\ x$ $[\![S]\!]_✓$ *?lhs2*)›
**hence** ‹$t \in \mathcal{D}$ (*?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*)›
**by** (*simp add*: *D-Det D-Mprefix* ‹$t = ev\ x \# t'$› *mv-left*(*1*))
**with** ‹$t \notin \mathcal{D}$ (*?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*)› **have** *False* **..**
**thus** ‹$(t, X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ *?lhs2*)› **..**
**next**
**case** *fail*
**have** ‹$(ev\ x \# t\text{-}P, X\text{-}P) \in \mathcal{F}$ *?lhs1*›
**by** (*simp add*: *F-Mprefix fail*(*1*) *mv-left*(*1*))
**moreover from** ‹$t = ev\ x \# t'$› *fail*(*3*) *mv-left*(*1*) ‹$A \cap S = \{\}$›
**have** ‹$t$ *setinterleaves*$_{✓\,tick\text{-}join}$ $((ev\ x \# t\text{-}P, t\text{-}Q), S)$›
**by** (*cases t-Q*) (*auto simp add*: *setinterleaving*$_{ptick}$*-simps split*: *event*$_{ptick}$*.split*)
**ultimately show** ‹$(t, X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ *?lhs2*)›
**using** *fail*(*2*, *4*) **by** (*auto simp add*: *F-Sync*$_{ptick}$)
**qed**
**next**
**case** *mv-right*
**from** *mv-right*(*2*) **consider** ‹$t' \in \mathcal{D}$ (*?lhs1* $[\![S]\!]_✓$ $Q\ x$)›
| (*fail*) *t-P t-Q X-P X-Q* **where**
‹$(t\text{-}P, X\text{-}P) \in \mathcal{F}$ *?lhs1*› ‹$(t\text{-}Q, X\text{-}Q) \in \mathcal{F}$ ($Q\ x$)›
‹$t'$ *setinterleaves*$_{✓\,tick\text{-}join}$ $((t\text{-}P, t\text{-}Q), S)$›
‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ *tick-join X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**thus** ‹$(t, X) \in \mathcal{F}$ (*?lhs1* $[\![S]\!]_✓$ *?lhs2*)›
**proof** *cases*

**assume** ‹$t' \in \mathcal{D}\ (?lhs1\ [\![S]\!]_{\checkmark}\ Q\ x)$›
**hence** ‹$t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3)$›
**by** (*simp add*: *D-Det D-Mprefix* ‹$t = ev\ x\ \#\ t'$› *mv-right*(*1*))
**with** ‹$t \notin \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3)$› **have** *False* **..**
**thus** ‹$(t, X) \in \mathcal{F}\ (?lhs1\ [\![S]\!]_{\checkmark}\ ?lhs2)$› **..**
**next**
**case** *fail*
**have** ‹$(ev\ x\ \#\ t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ ?lhs2$›
**by** (*simp add*: *F-Mprefix fail*(*2*) *mv-right*(*1*))
**moreover from** ‹$t = ev\ x\ \#\ t'$› *fail*(*3*) *mv-right*(*1*) ‹$B \cap S = \{\}$›
**have** ‹$t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P, ev\ x\ \#\ t\text{-}Q), S)$›
**by** (*cases t-P*) (*auto simp add*: $setinterleaving_{ptick}$*-simps split*: $event_{ptick}$*.split*)
**ultimately show** ‹$(t, X) \in \mathcal{F}\ (?lhs1\ [\![S]\!]_{\checkmark}\ ?lhs2)$›
**using** *fail*(*1*, *4*) **by** (*auto simp add*: $F\text{-}Sync_{ptick}$)
**qed**
**next**
**case** *mv-both*
**from** *mv-both*(*3*) **consider** ‹$t' \in \mathcal{D}\ (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
| (*fail*) *t-P t-Q X-P X-Q* **where**
‹$(t\text{-}P, X\text{-}P) \in \mathcal{F}\ (P\ x)$› ‹$(t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ (Q\ x)$›
‹$t'\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P, t\text{-}Q), S)$› ‹$X \subseteq super\text{-}ref\text{-}Sync_{ptick}\ tick\text{-}join\ X\text{-}P\ S\ X\text{-}Q$›
**unfolding** $Sync_{ptick}$*-projs* **by** *blast*
**thus** ‹$(t, X) \in \mathcal{F}\ (?lhs1\ [\![S]\!]_{\checkmark}\ ?lhs2)$›
**proof** *cases*
**assume** ‹$t' \in \mathcal{D}\ (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
**hence** ‹$t \in \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3)$›
**by** (*simp add*: *D-Det D-Mprefix* ‹$t = ev\ x\ \#\ t'$› *mv-both*(*1*, *2*))
**with** ‹$t \notin \mathcal{D}\ (?rhs1 \Box ?rhs2 \Box ?rhs3)$› **have** *False* **..**
**thus** ‹$(t, X) \in \mathcal{F}\ (?lhs1\ [\![S]\!]_{\checkmark}\ ?lhs2)$› **..**
**next**
**case** *fail*
**have** ‹$(ev\ x\ \#\ t\text{-}P, X\text{-}P) \in \mathcal{F}\ ?lhs1$›
**by** (*simp add*: *F-Mprefix fail*(*1*) *mv-both*(*1*))
**moreover have** ‹$(ev\ x\ \#\ t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ ?lhs2$›
**by** (*simp add*: *F-Mprefix fail*(*2*) *mv-both*(*2*))
**moreover from** ‹$t = ev\ x\ \#\ t'$› *fail*(*3*) *mv-both*(*1*) ‹$A' \subseteq S$›
**have** ‹$t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((ev\ x\ \#\ t\text{-}P, ev\ x\ \#\ t\text{-}Q), S)$› **by** *auto*
**ultimately show** ‹$(t, X) \in \mathcal{F}\ (?lhs1\ [\![S]\!]_{\checkmark}\ ?lhs2)$›
**using** *fail*(*4*) **by** (*simp* (*no-asm*) *add*: $F\text{-}Sync_{ptick}$) *blast*
**qed**
**qed**
**qed**
**qed**

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

$(\Box a{\in}(A - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box$
$(\Box b{\in}(B - S) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box x{\in}(A \cap B \cap S) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))\rangle$
**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.
$\langle \Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b =$
$(\Box a{\in}(A - S - B) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box$
$(\Box b{\in}(B - S - A) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box x{\in}(A \cap B - S) \to (P\ x\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b) \sqcap (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ x))\ \Box$
$(\Box x{\in}(A \cap B \cap S) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))\rangle$
**proof** −
**have** $*$ : $\langle \Box a{\in}(A - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b) =$
$(\Box a{\in}(A - S - B) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box$
$(\Box a{\in}(A \cap B - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\rangle$
**by** (*metis Diff-Int2 Diff-Int-distrib2 Mprefix-Un-distrib Un-Diff-Int*)
**have** $**$ : $\langle \Box b{\in}(B - S) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b) =$
$(\Box b{\in}(B - S - A) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box b{\in}(A \cap B - S) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\rangle$
**by** (*metis* (*no-types*) *Int-Diff Int-commute Mprefix-Un-distrib Un-Diff-Int*)
**have** $\langle \Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b =$
$(\Box a{\in}(A - S - B) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box$
$(\Box b{\in}(B - S - A) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$((\Box a{\in}(A \cap B - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box$
$(\Box b{\in}(A \cap B - S) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)))\ \Box$
$(\Box x{\in}(A \cap B \cap S) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))\rangle$
**unfolding** *Mprefix-Sync*$_{ptick}$*-Mprefix*
**by** (*auto simp add*: $**$ *Det-assoc intro*!: *arg-cong*[**where** $f = \langle \lambda P.\ P\ \Box$ -$\rangle$])
(*subst* (*3*) *Det-commute*, *subst Det-assoc*,
*auto simp add*: $*$ *Det-commute intro*: *arg-cong*[**where** $f = \langle \lambda P.\ P\ \Box$ -$\rangle$])
**also have** $\langle(\Box a{\in}(A \cap B - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b))\ \Box$
$(\Box b{\in}(A \cap B - S) \to (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) =$
$\Box x{\in}(A \cap B - S) \to ((P\ x\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \to Q\ b)) \sqcap (\Box a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}$
$Q\ x)\rangle$
**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$_{ptick}$*: ‹$\sqcap a \in A \rightarrow P\ a\ ;_{\checkmark}\ Q = \sqcap a \in A \rightarrow (P\ a\ ;_{\checkmark}\ Q)$›
**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* :
‹*SKIPS R* $[\![A]\!]_{\checkmark}$ *SKIPS S* $= \sqcap(r,\ s) \in R \times S.$ (*case r* $\otimes\checkmark$ *s of* $\lfloor$*r-s*$\rfloor \Rightarrow$ *SKIP r-s* | $\Diamond \Rightarrow$ *STOP*)›
**by** (*simp add*: *SKIPS-def Sync$_{ptick}$-distrib-GlobalNdet-left Sync$_{ptick}$-distrib-GlobalNdet-right*
(*simp add*: *GlobalNdet-cartprod*[*of R S* ‹$\lambda r\ s.$ *case r* $\otimes\checkmark$ *s of* $\Diamond \Rightarrow$ *STOP* | $\lfloor$*r-s*$\rfloor$ $\Rightarrow$ *SKIP r-s*›]
*GlobalNdet-sets-commute*[*of R S* ‹$\lambda r\ s.$ *case r* $\otimes\checkmark$ *s of* $\Diamond \Rightarrow$ *STOP* | $\lfloor$*r-s*$\rfloor \Rightarrow$ *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* :
‹$\sqcap(r,\ s) \in R \times S.$ *SKIP* $\lceil$*tick-join r s*$\rceil$ =
*SKIPS* ((*the* $\circ$ ($\lambda(r,\ s).$ *tick-join r s*)) ‘ ($R \times S$))›
**by** (*simp add*: *SKIPS-def mono-GlobalNdet-eq2 split-def*)

**lemma** *GlobalNdet-prod-case-SKIP-STOP-is-GlobalNdet-prod-SKIP-iff* :
‹$\sqcap(r,\ s) \in R \times S.$ (*case tick-join r s of* $\Diamond \Rightarrow$ *STOP* | $\lfloor$*r-s*$\rfloor \Rightarrow$ *SKIP r-s*) =
$\sqcap(r,\ s) \in R \times S.$ *SKIP* $\lceil$*tick-join r s*$\rceil$
$\longleftrightarrow$ ($\forall\ r\ s.\ r \in R \longrightarrow s \in S \longrightarrow$ *tick-join r s* $\neq \Diamond$)›
(**is** ‹*?lhs1* = *?lhs2* $\longleftrightarrow$ *?rhs*›)
**proof** (*rule iffI*)
**show** ‹*?rhs* $\Longrightarrow$ *?lhs1* = *?lhs2*›
**by** (*force intro*: *mono-GlobalNdet-eq*)
**next**
**have** ‹*UNIV* $\in \mathcal{R}$ *?lhs2* $\longleftrightarrow$ $R = \{\} \vee S = \{\}$›
**by** (*simp add*: *Refusals-iff F-GlobalNdet F-SKIP*)
**moreover have** ‹*UNIV* $\in \mathcal{R}$ *?lhs1* $\longleftrightarrow$ $R = \{\} \vee S = \{\} \vee (\exists\ r\ s.\ r \in R \wedge s \in S \wedge$ *tick-join r s* $= \Diamond)$›
**by** (*auto simp add*: *Refusals-iff F-GlobalNdet F-SKIP F-STOP split*: *option.split*)
**ultimately show** ‹*?lhs1* = *?lhs2* $\Longrightarrow$ *?rhs*› **by** (*metis empty-iff*)
**qed**

**lemma** (**in** *Sync$_{ptick}$-locale*) *SKIPS-Sync$_{ptick}$-SKIPS-bis* :
‹*SKIPS R* $[\![A]\!]_{\checkmark}$ *SKIPS S* = *SKIPS* ((*the* $\circ$ ($\lambda(r,\ s).\ r \otimes\checkmark s$)) ‘ ($R \times S$))›
**if** ‹$\bigwedge r\ s.\ r \in R \Longrightarrow s \in S \Longrightarrow r \otimes\checkmark s \neq \Diamond$›
**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*] : ‹*SKIPS R* $[\![A]\!]_{\checkmark}$ *STOP* = *STOP*›
**and** *STOP-Sync$_{ptick}$-SKIPS* [*simp*] : ‹*STOP* $[\![A]\!]_{\checkmark}$ *SKIPS S* = *STOP*›
**by** (*fact SKIPS-Sync$_{ptick}$-SKIPS*[**where** *S* = ‹{}›, *simplified*]
*SKIPS-Sync$_{ptick}$-SKIPS*[**where** *R* = ‹{}›, *simplified*])+

## Derived step Laws with Non-Determinism

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

**lemma** *Mprefix-Inter$_{ptick}$-Mprefix* :
‹$\Box a{\in}A \rightarrow P\ a\ |||_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b$ =
($\Box a{\in}A \rightarrow (P\ a\ |||_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b)) \ \Box\ (\Box b{\in}B \rightarrow (\Box a \in A \rightarrow P\ a\ |||_{\checkmark}\ Q\ b))$›
**by** (*fact Mprefix-Sync$_{ptick}$-Mprefix*[**where** *S* = ‹{}›, *simplified*])

**lemma** *Mprefix-Par$_{ptick}$-Mprefix* : ‹$\Box a{\in}A \rightarrow P\ a\ ||_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b = \Box x{\in}(A \cap B) \rightarrow (P\ x\ ||_{\checkmark}\ Q\ x)$›
**by** (*fact Mprefix-Sync$_{ptick}$-Mprefix*[**where** *S* = ‹*UNIV*›, *simplified*])

**lemma** *Mprefix-Sync$_{ptick}$-Mprefix-subset* :
‹$[\![A \subseteq S;\ B \subseteq S]\!] \Longrightarrow \Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b\ = \Box x{\in}(A \cap B) \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
**by** (*fact Mprefix-Sync$_{ptick}$-Mprefix-bis*[*of* ‹{}› *S A* ‹{}› *B*, *simplified*])

**lemma** *Mprefix-Sync$_{ptick}$-Mprefix-indep* :
‹$[\![A \cap S = \{\};\ B \cap S = \{\}]\!] \Longrightarrow$
$\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b =$
($\Box a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b)) \ \Box\ (\Box b{\in}B \rightarrow (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))$›
**by** (*fact Mprefix-Sync$_{ptick}$-Mprefix-bis*[*of A S* ‹{}› *B* ‹{}›, *simplified*])

**lemma** *Mprefix-Sync$_{ptick}$-Mprefix-left* :
‹$[\![A \cap S = \{\};\ B \subseteq S]\!] \Longrightarrow \Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b = \Box a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b)$›
**by** (*fact Mprefix-Sync$_{ptick}$-Mprefix-bis*[*of A S* ‹{}› ‹{}› *B*, *simplified*])

**lemma** *Mprefix-Sync$_{ptick}$-Mprefix-right* :
‹$[\![A \subseteq S;\ B \cap S = \{\}]\!] \Longrightarrow \Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b = \Box b{\in}B \rightarrow (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**by** (*fact Mprefix-Sync$_{ptick}$-Mprefix-bis*[*of* ‹{}› *S A B* ‹{}›, *simplified*])

**lemma** *Mprefix-Sync$_{ptick}$-STOP* : ‹$\Box a \in A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ STOP = \Box a \in (A - S) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$›
**by** (*subst Mprefix-empty*[*symmetric*], *subst Mprefix-Sync$_{ptick}$-Mprefix*, *simp*)

**lemma** *STOP-Sync$_{ptick}$-Mprefix* : ‹*STOP* $[\![S]\!]_{\checkmark}$ $\Box b \in B \to Q\ b = \Box b \in (B - S) \to (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**by** (*subst Mprefix-empty*[*symmetric*], *subst Mprefix-Sync$_{ptick}$-Mprefix*, *simp*)

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

**lemma** *Mprefix-Sync$_{ptick}$-Mndetprefix* :
‹$(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B \to Q\ b) =$
( *if* $B = \{\}$ *then* $(\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}$ *STOP*
*else* $\sqcap b{\in}B.$ (*if* $b \in S$ *then STOP else* $(b \to ((\Box a \in A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b)))\ \Box$
$(\Box a{\in}(A - S) \to (P\ a\ [\![S]\!]_{\checkmark}\ (b \to Q\ b)))\ \Box$
(*if* $b \in A \cap S$ *then* $(b \to (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b))$ *else STOP*))›
**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** $f = \langle(\Box)\rangle$] *split*: *if-split-asm*)

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

**lemma** *Mndetprefix-Sync$_{ptick}$-STOP* :
‹$(\sqcap a \in A \to P\ a)\ [\![S]\!]_{\checkmark}$ *STOP* =
( *if* $A \cap S = \{\}$ *then* $\sqcap a \in A \to (P\ a\ [\![S]\!]_{\checkmark}$ *STOP*)
*else* $(\sqcap a \in (A - S) \to (P\ a\ [\![S]\!]_{\checkmark}$ *STOP*)) $\sqcap$ *STOP*)›
(**is** ‹*?lhs* = (*if* $A \cap S = \{\}$ *then ?rhs1 else ?rhs2* $\sqcap$ *STOP*)›)
**proof** −
**have** ‹$(\sqcap a \in A \to P\ a)\ [\![S]\!]_{\checkmark}$ *STOP* =
$\sqcap a{\in}A.$ (*if* $a \in S$ *then STOP else* $(a \to (P\ a\ [\![S]\!]_{\checkmark}$ *STOP*)))› (**is** ‹*?lhs* = *?rhs′*›)
**by** (*subst Mndetprefix-Sync$_{ptick}$-Mprefix*[**where** $B = \langle\{\}\rangle$, *simplified*])
(*auto intro*: *mono-GlobalNdet-eq*)
**also have** ‹*?rhs′* = (*if* $A \cap S = \{\}$ *then ?rhs1 else ?rhs2* $\sqcap$ *STOP*)›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$A \cap S = \{\} \Longrightarrow$ *?rhs′* = *?rhs1*›
**by** (*auto simp add*: *Mndetprefix-GlobalNdet intro*!: *mono-GlobalNdet-eq*)
**next**
**show** ‹*?rhs′* = *?rhs2* $\sqcap$ *STOP*› **if** ‹$A \cap S \neq \{\}$›
**proof** (*cases* ‹$A - S = \{\}$›)
**show** ‹*?rhs′* = *?rhs2* $\sqcap$ *STOP*› **if** ‹$A - S = \{\}$›

```
      by (simp add: ‹A − S = {}› GlobalNdet-is-STOP-iff)
        (use ‹A − S = {}› in blast)
    next
      show ‹?rhs′ = ?rhs2 ⊓ STOP› if ‹A − S ≠ {}›
      proof (subst Int-Diff-Un[symmetric],
          subst GlobalNdet-factorization-union
          [OF ‹A ∩ S ≠ {}› ‹A − S ≠ {}›, symmetric])
        have ‹(⊓a∈(A ∩ S). (if a ∈ S then STOP else (a → (P a ⟦S⟧✓ STOP))))
            = STOP› (is ‹?fact1 = STOP›)
          by (simp add: GlobalNdet-is-STOP-iff)
        moreover have ‹(⊓a∈(A − S). (if a ∈ S then STOP else (a → (P a ⟦S⟧✓
STOP))))
                    = ?rhs2› (is ‹?fact2 = ?rhs2›)
          by (auto simp add: Mndetprefix-GlobalNdet intro: mono-GlobalNdet-eq)
      ultimately show ‹?fact1 ⊓ ?fact2 = ?rhs2 ⊓ STOP› by (metis Ndet-commute)
      qed
    qed
  qed
  finally show ‹?lhs = (if A ∩ S = {} then ?rhs1 else ?rhs2 ⊓ STOP)› .
qed

lemma STOP-Sync_ptick-Mndetprefix :
  ‹STOP ⟦S⟧✓ (⊓b ∈ B → Q b) =
   ( if B ∩ S = {} then ⊓b ∈ B → (STOP ⟦S⟧✓ Q b)
    else (⊓b ∈ (B − S) → (STOP ⟦S⟧✓ Q b)) ⊓ STOP)›
  (is ‹?lhs = (if B ∩ S = {} then ?rhs1 else ?rhs2 ⊓ STOP)›)
proof −
  have ‹STOP ⟦S⟧✓ (⊓b ∈ B → Q b) =
        ⊓b∈B. (if b ∈ S then STOP else (b → (STOP ⟦S⟧✓ Q b)))› (is ‹?lhs =
?rhs′›)
    by (subst Mprefix-Sync_ptick-Mndetprefix[where A = ‹{}›, simplified])
      (auto intro: mono-GlobalNdet-eq)
  also have ‹?rhs′ = (if B ∩ S = {} then ?rhs1 else ?rhs2 ⊓ STOP)›
  proof (split if-split, intro conjI impI)
    show ‹B ∩ S = {} ⟹ ?rhs′ = ?rhs1›
      by (auto simp add: Mndetprefix-GlobalNdet intro!: mono-GlobalNdet-eq)
  next
    show ‹?rhs′ = ?rhs2 ⊓ STOP› if ‹B ∩ S ≠ {}›
    proof (cases ‹B − S = {}›)
      show ‹?rhs′ = ?rhs2 ⊓ STOP› if ‹B − S = {}›
        by (simp add: ‹B − S = {}› GlobalNdet-is-STOP-iff)
          (use ‹B − S = {}› in blast)
    next
      show ‹?rhs′ = ?rhs2 ⊓ STOP› if ‹B − S ≠ {}›
      proof (subst Int-Diff-Un[symmetric],
          subst GlobalNdet-factorization-union
          [OF ‹B ∩ S ≠ {}› ‹B − S ≠ {}›, symmetric])
        have ‹(⊓b∈(B ∩ S). (if b ∈ S then STOP else (b → (STOP ⟦S⟧✓ Q b))))
            = STOP› (is ‹?fact1 = STOP›)
```

**by** (*simp add*: *GlobalNdet-is-STOP-iff*)
**moreover have** ‹(⊓*b*∈(*B* − *S*). (*if b* ∈ *S then STOP else* (*b* → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*))))
= *?rhs2*› (**is** ‹*?fact2* = *?rhs2*›)
**by** (*auto simp add*: *Mndetprefix-GlobalNdet intro*: *mono-GlobalNdet-eq*)
**ultimately show** ‹*?fact1* ⊓ *?fact2* = *?rhs2* ⊓ *STOP*› **by** (*metis Ndet-commute*)
**qed**
**qed**
**qed**
**finally show** ‹*?lhs* = (*if B* ∩ *S* = {} *then ?rhs1 else ?rhs2* ⊓ *STOP*)› **.**
**qed**

**corollary** *Mndetprefix-Sync*$_{ptick}$*-Mprefix-subset* :
‹(⊓*a* ∈ *A* → *P a*) $[\![S]\!]_{\checkmark}$ (□*b* ∈ *B* → *Q b*) =
( *if A* ⊆ *B then* ⊓ *a* ∈ *A* → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)
*else* (⊓*a* ∈ (*A* ∩ *B*) → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if A* ⊆ *B then ?rhs1 else ?rhs2*)›) **if** ‹*A* ⊆ *S*› ‹*B* ⊆ *S*›
**proof** (*cases* ‹*A* = {}›)
**show** ‹*A* = {} ⟹ *?lhs* = (*if A* ⊆ *B then ?rhs1 else ?rhs2*)›
**by** (*simp add*: *Mprefix-is-STOP-iff STOP-Sync*$_{ptick}$*-Mprefix* ‹*B* ⊆ *S*›)
**next**
**from** ‹*A* ⊆ *S*› **have** ∗ : ‹*a* ∈ *A* ⟹ *a* ∈ *S*› **for** *a* **by** *blast*
**from** ‹*B* ⊆ *S*› **have** ∗∗ : ‹*B* − *S* = {}› ‹*b* ∈ *B* ∧ *b* ∈ *S* ⟷ *b* ∈ *B*› **for** *b* **by** *auto*
**assume** ‹*A* ≠ {}›
**have** ‹*?lhs* = ⊓*a*∈*A*. (*if a* ∈ *B then* (*a* → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)) *else STOP*)› (**is** ‹*?lhs* = *?rhs*′›)
**by** (*auto simp add*: *Mndetprefix-Sync*$_{ptick}$*-Mprefix* ∗ ∗∗ ‹*A* ≠ {}› *intro*: *mono-GlobalNdet-eq*)
**also have** ‹*?rhs*′ = (*if A* ⊆ *B then ?rhs1 else ?rhs2*)›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*A* ⊆ *B* ⟹ *?rhs*′ = ⊓*a*∈*A* → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)›
**by** (*auto simp add*: *Mndetprefix-GlobalNdet intro*!: *mono-GlobalNdet-eq*)
**next**
**show** ‹*?rhs*′ = (⊓*a*∈(*A* ∩ *B*) → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)) ⊓ *STOP*› **if** ‹¬ *A* ⊆ *B*›
**proof** (*cases* ‹*A* ∩ *B* = {}›)
**show** ‹*A* ∩ *B* = {} ⟹ *?rhs*′ = (⊓*a*∈(*A* ∩ *B*) → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)) ⊓ *STOP*›
**by** (*auto simp add*: *GlobalNdet-is-STOP-iff*)
**next**
**assume** ‹*A* ∩ *B* ≠ {}›
**from** ‹¬ *A* ⊆ *B*› **have** ‹*A* − *B* ≠ {}› **by** *blast*
**show** ‹*?rhs*′ = (⊓*a*∈(*A* ∩ *B*) → (*P a* $[\![S]\!]_{\checkmark}$ *Q a*)) ⊓ *STOP*›
**by** (*auto simp add*: *Mndetprefix-GlobalNdet GlobalNdet-is-STOP-iff*
*simp flip*: *GlobalNdet-factorization-union*
[*OF* ‹*A* ∩ *B* ≠ {}› ‹*A* − *B* ≠ {}›, *unfolded Int-Diff-Un*]
*intro*!: *arg-cong2*[**where** *f* = ‹(⊓)›] *mono-GlobalNdet-eq*)
**qed**
**qed**
**finally show** ‹*?lhs* = (*if A* ⊆ *B then ?rhs1 else ?rhs2*)› **by** *simp*

**qed**

**corollary** *Mprefix-Sync$_{ptick}$-Mndetprefix-subset* :
‹$\Box a \in A \to P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b \in B \to Q\ b =$
( *if* $B \subseteq A$ *then* $\sqcap b \in B \to (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b)$
*else* $(\sqcap b \in (A \cap B) \to (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b)) \sqcap STOP$)›
(**is** ‹*?lhs* = (*if* $B \subseteq A$ *then ?rhs1 else ?rhs2*)›) **if** ‹$A \subseteq S$› ‹$B \subseteq S$›
**proof** (*cases* ‹$B = \{\}$›)
**show** ‹$B = \{\} \Longrightarrow$ *?lhs* = (*if* $B \subseteq A$ *then ?rhs1 else ?rhs2*)›
**by** (*simp add*: *Mprefix-is-STOP-iff Mprefix-Sync$_{ptick}$-STOP* ‹$A \subseteq S$›)
**next**
**from** ‹$B \subseteq S$› **have** $*$ : ‹$b \in B \Longrightarrow b \in S$› **for** $b$ **by** *blast*
**from** ‹$A \subseteq S$› **have** $**$ : ‹$A - S = \{\}$› ‹$a \in A \wedge a \in S \longleftrightarrow a \in A$› **for** $a$ **by** *auto*
**assume** ‹$B \neq \{\}$›
**have** ‹*?lhs* = $\sqcap b{\in}B.$ (*if* $b \in A$ *then* $(b \to (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b))$ *else STOP*)› (**is** ‹*?lhs* = *?rhs′*›)
**by** (*auto simp add*: *Mprefix-Sync$_{ptick}$-Mndetprefix* $*$ $**$ ‹$B \neq \{\}$› *intro*: *mono-GlobalNdet-eq*)
**also have** ‹*?rhs′* = (*if* $B \subseteq A$ *then ?rhs1 else ?rhs2*)›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$B \subseteq A \Longrightarrow$ *?rhs′* = $\sqcap b{\in}B \to (P\ b\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**by** (*auto simp add*: *Mndetprefix-GlobalNdet intro*!: *mono-GlobalNdet-eq*)
**next**
**show** ‹*?rhs′* = $(\sqcap a{\in}(A \cap B) \to (P\ a\ [\![S]\!]_{\checkmark}\ Q\ a)) \sqcap STOP$› **if** ‹$\neg\ B \subseteq A$›
**proof** (*cases* ‹$A \cap B = \{\}$›)
**show** ‹$A \cap B = \{\} \Longrightarrow$ *?rhs′* = $(\sqcap a{\in}(A \cap B) \to (P\ a\ [\![S]\!]_{\checkmark}\ Q\ a)) \sqcap STOP$›
**by** (*auto simp add*: *GlobalNdet-is-STOP-iff*)
**next**
**assume** ‹$A \cap B \neq \{\}$›
**hence** ‹$B \cap A \neq \{\}$› **by** *blast*
**from** ‹$\neg\ B \subseteq A$› **have** ‹$B - A \neq \{\}$› **by** *blast*
**show** ‹*?rhs′* = $(\sqcap a{\in}(A \cap B) \to (P\ a\ [\![S]\!]_{\checkmark}\ Q\ a)) \sqcap STOP$›
**by** (*auto simp add*: *Mndetprefix-GlobalNdet GlobalNdet-is-STOP-iff*
*simp flip*: *Int-commute GlobalNdet-factorization-union*
[*OF* ‹$B \cap A \neq \{\}$› ‹$B - A \neq \{\}$›, *unfolded Int-Diff-Un*]
*intro*!: *arg-cong2*[**where** $f$ = ‹$(\sqcap)$›] *mono-GlobalNdet-eq*)
**qed**
**qed**
**finally show** ‹*?lhs* = (*if* $B \subseteq A$ *then ?rhs1 else ?rhs2*)› **by** *simp*
**qed**

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

**if** ‹$A \cap S = \{\}$› **and** ‹$B \cap S = \{\}$›
**proof** (*cases* ‹$A = \{\}$›)
**show** ‹$A = \{\} \Longrightarrow$ *?thesis*›
**by** (*simp add*: *Diff-triv STOP-Sync*$_{ptick}$*-Mprefix* ‹$B \cap S = \{\}$›)
**next**
**from** *that*(*1*) **have** $*$ : ‹$a \in A \Longrightarrow a \notin S$› **for** $a$ **by** *blast*
**from** *that*(*2*) **have** $**$ : ‹$B - S = B$› **by** *blast*
**show** *?thesis* **if** ‹$A \neq \{\}$›
**by** (*simp add*: *Mndetprefix-Sync*$_{ptick}$*-Mprefix* ‹$A \neq \{\}$›)
(*rule mono-GlobalNdet-eq*, *simp add*: $*$ $**$)
**qed**

**corollary** *Mprefix-Sync*$_{ptick}$*-Mndetprefix-indep* :
‹$(\Box a \in A \rightarrow P\ a)\ [\![S]\!]_{\checkmark}\ (\sqcap b \in B \rightarrow Q\ b) =$
( *if* $B = \{\}$ *then* $\Box a \in A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
*else* $\sqcap b \in B.\ (b \rightarrow ((\Box a \in A \rightarrow P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box a \in A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ (b \rightarrow Q\ b))))$›
**if** ‹$A \cap S = \{\}$› ‹$B \cap S = \{\}$›
**proof** (*cases* ‹$B = \{\}$›)
**show** ‹$B = \{\} \Longrightarrow$ *?thesis*›
**by** (*simp add*: *Diff-triv Mprefix-Sync*$_{ptick}$*-STOP* ‹$A \cap S = \{\}$›)
**next**
**from** *that*(*2*) **have** $*$ : ‹$b \in B \Longrightarrow b \notin S$› **for** $b$ **by** *blast*
**from** *that*(*1*) **have** $**$ : ‹$A - S = A$› **by** *blast*
**show** *?thesis* **if** ‹$B \neq \{\}$›
**by** (*simp add*: *Mprefix-Sync*$_{ptick}$*-Mndetprefix* ‹$B \neq \{\}$›)
(*rule mono-GlobalNdet-eq*, *simp add*: $*$ $**$)
**qed**

**corollary** *Mndetprefix-Sync*$_{ptick}$*-Mprefix-left* :
‹$(\sqcap a \in A \rightarrow P\ a)\ [\![S]\!]_{\checkmark}\ (\Box b \in B \rightarrow Q\ b) =$
( *if* $A = \{\}$ *then* $STOP\ [\![S]\!]_{\checkmark}\ (\Box b \in B \rightarrow Q\ b)$
*else* $\sqcap a \in A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ (\Box b \in B \rightarrow Q\ b)))$›
**if** ‹$A \cap S = \{\}$› **and** ‹$B \subseteq S$›
**proof** (*cases* ‹$A = \{\}$›)
**show** ‹$A = \{\} \Longrightarrow$ *?thesis*› **by** *simp*
**next**
**from** *that*(*1*) **have** $*$ : ‹$a \in A \Longrightarrow a \notin S$› **for** $a$ **by** *blast*
**from** *that*(*2*) **have** $**$ : ‹$B - S = \{\}$› **by** *blast*
**show** *?thesis* **if** ‹$A \neq \{\}$›
**by** (*simp add*: *Mndetprefix-Sync*$_{ptick}$*-Mprefix* ‹$A \neq \{\}$›, *unfold Mndetprefix-GlobalNdet*)
(*rule mono-GlobalNdet-eq*, *simp add*: $*$ $**$)
**qed**

**corollary** *Mndetprefix-Sync*$_{ptick}$*-Mprefix-right* :
‹$(\sqcap a \in A \rightarrow P\ a)\ [\![S]\!]_{\checkmark}\ (\Box b \in B \rightarrow Q\ b) =$
( *if* $A = \{\}$ *then* $STOP\ [\![S]\!]_{\checkmark}\ (\Box b \in B \rightarrow Q\ b)$

```
    else □b∈B → ((⊓a∈A → P a) ⟦S⟧✓ Q b))›
  if ‹A ⊆ S› and ‹B ∩ S = {}›
proof (cases ‹A = {}›)
  show ‹A = {} ⟹ ?thesis› by simp
next
  from that(1) have  ∗ : ‹a ∈ A ⟹ a ∈ S› for a by blast
  from that(2) have  ∗∗ : ‹B − S = B› by blast
  show ?thesis if ‹A ≠ {}›
    by (simp add: Mndetprefix-Sync_ptick-Mprefix ‹A ≠ {}›,
        simp add: Mndetprefix-GlobalNdet Sync_ptick-distrib-GlobalNdet-right ‹A ≠
{}›
        flip: GlobalNdet-Mprefix-distr)
       (rule mono-GlobalNdet-eq, use ∗ ∗∗ in auto)
qed

corollary Mprefix-Sync_ptick-Mndetprefix-left :
  ‹(□a ∈ A → P a) ⟦S⟧✓ (⊓b ∈ B → Q b) =
   (  if B = {} then (□a ∈ A → P a) ⟦S⟧✓ STOP
    else □a∈A → (P a ⟦S⟧✓ (⊓b ∈ B → Q b)))›
  if ‹A ∩ S = {}› ‹B ⊆ S›
proof (cases ‹B = {}›)
  show ‹B = {} ⟹ ?thesis› by simp
next
  from that(2) have  ∗ : ‹b ∈ B ⟹ b ∈ S› for b by blast
  from that(1) have  ∗∗ : ‹A − S = A› by blast
  show ?thesis if ‹B ≠ {}›
    by (simp add: Mprefix-Sync_ptick-Mndetprefix ‹B ≠ {}›,
      simp add: Mndetprefix-GlobalNdet Sync_ptick-distrib-GlobalNdet-left ‹B ≠ {}›
       flip: GlobalNdet-Mprefix-distr)
      (rule mono-GlobalNdet-eq, use ∗ ∗∗ in auto)
qed

corollary Mprefix-Sync_ptick-Mndetprefix-right :
  ‹(□a ∈ A → P a) ⟦S⟧✓ (⊓b ∈ B → Q b) =
   (  if B = {} then (□a ∈ A → P a) ⟦S⟧✓ STOP
    else ⊓b∈B → ((□a ∈ A → P a) ⟦S⟧✓ Q b))›
  if ‹A ⊆ S› ‹B ∩ S = {}›
proof (cases ‹B = {}›)
  show ‹B = {} ⟹ ?thesis› by simp
next
  from that(2) have  ∗ : ‹b ∈ B ⟹ b ∉ S› for b by blast
  from that(1) have  ∗∗ : ‹A − S = {}› by blast
  show ?thesis if ‹B ≠ {}›
    by (simp add: Mprefix-Sync_ptick-Mndetprefix ‹B ≠ {}›,
        unfold Mndetprefix-GlobalNdet)
      (rule mono-GlobalNdet-eq, simp add: ∗ ∗∗)
qed
```

**corollary** *Mndetprefix-Par*$_{ptick}$*-Mprefix* :
‹$\sqcap a \in A \rightarrow P\ a\ ||_{\checkmark}\ \Box b \in B \rightarrow Q\ b =$
($if\ A \subseteq B\ then\ \sqcap a \in A \rightarrow (P\ a\ ||_{\checkmark}\ Q\ a)\ else\ (\sqcap a \in (A \cap B) \rightarrow (P\ a\ ||_{\checkmark}\ Q\ a))$
$\sqcap\ STOP$)›
**by** (*simp add*: *Mndetprefix-Sync*$_{ptick}$*-Mprefix-subset*)

**corollary** *Mprefix-Par*$_{ptick}$*-Mndetprefix* :
‹$\Box a \in A \rightarrow P\ a\ ||_{\checkmark}\ \sqcap b \in B \rightarrow Q\ b =$
($if\ B \subseteq A\ then\ \sqcap b \in B \rightarrow (P\ b\ ||_{\checkmark}\ Q\ b)\ else\ (\sqcap b \in (A \cap B) \rightarrow (P\ b\ ||_{\checkmark}\ Q\ b))$
$\sqcap\ STOP$)›
**by** (*simp add*: *Mprefix-Sync*$_{ptick}$*-Mndetprefix-subset*)

**corollary** *Mndetprefix-Inter*$_{ptick}$*-Mprefix* :
‹$\sqcap a \in A \rightarrow P\ a\ |||_{\checkmark}\ \Box b \in B \rightarrow Q\ b =$
($if\ A = \{\}\ then\ \Box b \in B \rightarrow RenamingTick\ (Q\ b\ ;\ STOP)\ (\lambda s.\ the\ (tick\text{-}join\ (g$
$s)\ s)$)
$else\ \sqcap a{\in}A.\ (a \rightarrow (P\ a\ |||_{\checkmark}\ \Box b \in B \rightarrow Q\ b)) \Box$
$(\Box b{\in}B \rightarrow (a \rightarrow P\ a\ |||_{\checkmark}\ Q\ b)))$›
**by** (*simp add*: *Mndetprefix-Sync*$_{ptick}$*-Mprefix-indep*
*Mprefix-Seq STOP-Inter*$_{ptick}$[*of* - *g*])

**corollary** *Mprefix-Inter*$_{ptick}$*-Mndetprefix* :
‹$\Box a \in A \rightarrow P\ a\ |||_{\checkmark}\ \sqcap b \in B \rightarrow Q\ b =$
($if\ B = \{\}\ then\ \Box a \in A \rightarrow RenamingTick\ (P\ a\ ;\ STOP)\ (\lambda r.\ the\ (tick\text{-}join\ r$
$(g\ r)$))
$else\ \sqcap b{\in}B.\ (b \rightarrow (\Box a \in A \rightarrow P\ a\ |||_{\checkmark}\ Q\ b)) \Box$
$(\Box a{\in}A \rightarrow (P\ a\ |||_{\checkmark}\ b \rightarrow Q\ b)))$›
**by** (*simp add*: *Mprefix-Sync*$_{ptick}$*-Mndetprefix-indep*
*Mprefix-Seq Inter*$_{ptick}$*-STOP*[*of* - *g*])

**Mixing two non deterministic prefix choices** **lemma** *Mndetprefix-Sync*$_{ptick}$*-Mndetprefix*
:
‹$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b =$
($if\ A = \{\}\ then\quad if\ B \cap S = \{\}\ then\ \sqcap b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
$else\ (\sqcap x \in (B - S) \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ x)) \sqcap STOP$
$else\quad if\ B = \{\}\ then\quad if\ A \cap S = \{\}\ then\ \sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
$else\ (\sqcap x \in (A - S) \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ STOP)) \sqcap STOP$
$else\ \sqcap b{\in}B.\ \sqcap a{\in}A.$
$(if\ a \in S\ then\ STOP\ else\ a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ b)) \Box$
$(if\ b \in S\ then\ STOP\ else\ b \rightarrow (a \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)) \Box$
$(if\ a = b \wedge b \in S\ then\ b \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)\ else\ STOP))$›
(**is** ‹*?lhs* = ($if\ A = \{\}\ then\ if\ B \cap S = \{\}\ then\ ?mv\text{-}right\ else\ ?mv\text{-}right'\ \sqcap$
$STOP$
$else\quad if\ B = \{\}\ then\ if\ A \cap S = \{\}\ then\ ?mv\text{-}left\ else\ ?mv\text{-}left'\ \sqcap$
$STOP$
$else\ ?huge\text{-}mess$)›)
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$A = \{\} \Longrightarrow$ *?lhs* = ($if\ B \cap S = \{\}\ then\ ?mv\text{-}right\ else\ ?mv\text{-}right'\ \sqcap$

*STOP*)›
**by** (*auto simp add*: *STOP-Sync$_{ptick}$-Mndetprefix intro*: *mono-Mndetprefix-eq*)
**next**
**show** ‹*?lhs* = (*if B* = {} *then if A* ∩ *S* = {} *then ?mv-left else ?mv-left′* ⊓ *STOP else ?huge-mess*)› **if** ‹*A* ≠ {}›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*B* = {} ⟹ *?lhs* = (*if A* ∩ *S* = {} *then ?mv-left else ?mv-left′* ⊓ *STOP*)›
**by** (*auto simp add*: *Mndetprefix-Sync$_{ptick}$-STOP intro*: *mono-Mndetprefix-eq*)
**next**
**assume** ‹*B* ≠ {}›
**have** ‹*?lhs* = ⊓*b*∈*B*. ⊓*a*∈*A*. (*a* → *P a* ⟦*S*⟧$_{\checkmark}$ (*b* → *Q b*))›
**by** (*simp add*: *Mndetprefix-GlobalNdet* ‹*A* ≠ {}› ‹*B* ≠ {}›
*Sync$_{ptick}$-distrib-GlobalNdet-left Sync$_{ptick}$-distrib-GlobalNdet-right*)
**also have** ‹. . . = *?huge-mess*›
**by** (*auto simp add*: *write0-def Mprefix-Sync$_{ptick}$-Mprefix Diff-triv Mprefix-is-STOP-iff*
*intro*!: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])
**finally show** ‹*?lhs* = *?huge-mess*› **.**
**qed**
**qed**

**lemma** *Mndetprefix-Sync$_{ptick}$-Mndetprefix-subset* :
‹⊓*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ ⊓*b*∈*B* → *Q b* =
( *if* ∃ *b*. *A* = {*b*} ∧ *B* = {*b*}
*then* (*THE b*. *B* = {*b*}) → (*P* (*THE a*. *A* = {*a*}) ⟦*S*⟧$_{\checkmark}$ *Q* (*THE b*. *B* = {*b*}))
*else* (⊓*x* ∈ (*A* ∩ *B*) → (*P x* ⟦*S*⟧$_{\checkmark}$ *Q x*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if ?cond then ?rhs1 else ?rhs2*)›) **if** ‹*A* ⊆ *S*› ‹*B* ⊆ *S*›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*?cond* ⟹ *?lhs* = *?rhs1*›
**by** (*elim exE*, *simp add*: *write0-def*)
(*subst Mprefix-Sync$_{ptick}$-Mprefix-subset*, *use* ‹*A* ⊆ *S*› **in** *simp-all*)
**next**
**assume** ‹¬ *?cond*›
**let** *?term* = ‹λ*a b*. (*b* → (*P a* ⟦*S*⟧$_{\checkmark}$ *Q b*))›
**have** ‹*?lhs* = ⊓*b*∈*B*. ⊓*a*∈*A*. (*if a* = *b then ?term a b else STOP*)›
(**is** ‹*?lhs* = ⊓*b*∈*B*. ⊓*a*∈*A*. *?rhs′ b a*›)
**proof** (*cases* ‹*A* = {} ∨ *B* = {}›)
**from** ‹*A* ⊆ *S*› ‹*B* ⊆ *S*› **show** ‹*A* = {} ∨ *B* = {} ⟹ *?lhs* = (⊓*b*∈*B*. ⊓*a*∈*A*. *?rhs′ b a*)›
**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** ‹¬ (*A* = {} ∨ *B* = {}) ⟹ *?lhs* = (⊓*b*∈*B*. ⊓*a*∈*A*. *?rhs′ b a*)›
**by** (*simp add*: *Mndetprefix-Sync$_{ptick}$-Mndetprefix*)
(*intro mono-GlobalNdet-eq*, *use* ‹*A* ⊆ *S*› ‹*B* ⊆ *S*› **in** *auto*)
**qed**

**also have** ‹(⊓*b*∈*B*. ⊓*a*∈*A*. *?rhs′ b a*) = *?rhs2*›

**proof** (*cases* ‹$B \cap A = \{\}$›)
**assume** ‹$B \cap A = \{\}$›
**hence** ‹$A \cap B = \{\}$› **by** *blast*
**hence** ‹$(\sqcap b{\in}B.\ \sqcap a{\in}A.\ ?rhs'\ b\ a) = STOP$› **by** (*auto simp add*: *Global-Ndet-is-STOP-iff*)
**thus** ‹$(\sqcap b{\in}B.\ \sqcap a{\in}A.\ ?rhs'\ b\ a) = ?rhs2$› **by** (*auto simp add*: ‹$A \cap B = \{\}$›)
**next**
**show** ‹$(\sqcap b{\in}B.\ \sqcap a{\in}A.\ ?rhs'\ b\ a) = ?rhs2$› **if** ‹$B \cap A \neq \{\}$›
**proof** (*cases* ‹$B - A = \{\}$›)
**assume** ‹$B - A = \{\}$›
**hence** ‹$A \cap B = B$› **by** *blast*
**have** ‹$(\sqcap a{\in}A.\ ?rhs'\ b\ a) = (if\ A = \{b\}\ then\ ?term\ b\ b\ else\ ?term\ b\ b \sqcap STOP)$›
(**is** ‹$(\sqcap a{\in}A.\ ?rhs'\ b\ a) = ?rhs''\ b$›) **if** ‹$b \in B$› **for** $b$
**proof** (*cases* ‹$A \cap \{b\} = \{\}$›)
**from** ‹$B - A = \{\}$› ‹$b \in B$›
**show** ‹$A \cap \{b\} = \{\} \Longrightarrow (\sqcap a{\in}A.\ ?rhs'\ b\ a) = ?rhs''\ b$› **by** *auto*
**next**
**show** ‹$(\sqcap a{\in}A.\ ?rhs'\ b\ a) = ?rhs''\ b$› **if** ‹$A \cap \{b\} \neq \{\}$›
**proof** (*cases* ‹$A - \{b\} = \{\}$›)
**show** ‹$A - \{b\} = \{\} \Longrightarrow (\sqcap a{\in}A.\ ?rhs'\ b\ a) = ?rhs''\ b$›
**using** ‹$A \cap \{b\} \neq \{\}$› **by** *auto*
**next**
**show** ‹$\sqcap a{\in}A.\ ?rhs'\ b\ a = ?rhs''\ b$› **if** ‹$A - \{b\} \neq \{\}$›
**using** ‹$A - \{b\} \neq \{\}$› ‹$A \cap \{b\} \neq \{\}$›
**by** (*auto simp add*: *GlobalNdet-is-STOP-iff*
*simp flip*: *GlobalNdet-factorization-union*
[*OF* ‹$A \cap \{b\} \neq \{\}$› ‹$A - \{b\} \neq \{\}$›, *unfolded Int-Diff-Un*]
*intro*: *arg-cong2*[**where** $f = \langle(\sqcap)\rangle$])
**qed**
**qed**
**hence** ‹$(\sqcap b \in B.\ \sqcap a \in A.\ ?rhs'\ b\ a) = \sqcap b \in B.\ ?rhs''\ b$›
**by** (*fact mono-GlobalNdet-eq*)
**also have** ‹$(\sqcap b \in B.\ ?rhs''\ b) = ?rhs2$›
**proof** −
**from** ‹$\neg$ *?cond*› **have** ‹$(\sqcap b \in B.\ ?rhs''\ b) = \sqcap b \in B.\ ?term\ b\ b \sqcap STOP$›
**by** (*metis Diff-eq-empty-iff Int-commute* ‹$A \cap B = B$›
‹$B - A = \{\}$› *subset-singleton-iff* ‹$B \cap A \neq \{\}$›)
**also have** ‹$\ldots = (\sqcap b \in B.\ ?term\ b\ b) \sqcap STOP$›
**by** (*simp add*: *Process-eq-spec Ndet-projs GlobalNdet-projs STOP-projs*)
**finally show** ‹$(\sqcap b \in B.\ ?rhs''\ b) = ?rhs2$›
**by** (*simp add*: *Mndetprefix-GlobalNdet* ‹$A \cap B = B$›)
**qed**
**finally show** ‹$(\sqcap b \in B.\ \sqcap a \in A.\ ?rhs'\ b\ a) = ?rhs2$› **.**
**next**
**assume** ‹$B - A \neq \{\}$›
**have** ‹$\sqcap a{\in}A.\ (if\ a = b\ then\ ?term\ a\ b\ else\ STOP) =$
$(if\ b \in A\ then\ if\ A = \{b\}\ then\ ?term\ b\ b\ else\ (?term\ b\ b) \sqcap STOP\ else\ STOP)$›

```
      if ‹b ∈ B› for b
    proof (split if-split, intro conjI impI)
      show ‹⊓a∈A. (if a = b then ?term a b else STOP) =
            (if A = {b} then ?term b b else (?term b b) ⊓ STOP)› if ‹b ∈ A›
      proof (split if-split, intro conjI impI)
       show ‹A = {b} ⟹ ⊓a ∈ A. (if a = b then ?term a b else STOP) = ?term
b b› by simp
      next
        assume ‹A ≠ {b}›
        with ‹b ∈ A› have ‹insert b A = A› ‹A − {b} ≠ {}› by auto
       show ‹A ≠ {b} ⟹ ⊓a∈A. (if a = b then ?term a b else STOP) = ?term
b b ⊓ STOP›
         by (auto simp add: GlobalNdet-is-STOP-iff intro!: arg-cong2[where f =
‹(⊓)›]
             simp flip: GlobalNdet-factorization-union
              [of ‹{b}›, OF - ‹A − {b} ≠ {}›, simplified, unfolded ‹insert b A =
A›])
      qed
    next
      show ‹b ∉ A ⟹ ⊓a∈A. (if a = b then ?term a b else STOP) = STOP›
        by (auto simp add: GlobalNdet-is-STOP-iff)
    qed

    hence ‹⊓b ∈ B. ⊓a∈A. ?rhs' b a =
          ⊓b ∈ B. (if b ∈ A then if A = {b} then ?term b b else (?term b b) ⊓
STOP else STOP)›
      by (fact mono-GlobalNdet-eq)
    also from ‹B − A ≠ {}› have ‹... = (⊓b ∈ B. (if b ∈ A then ?term b b else
STOP)) ⊓ STOP›
      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 ‹... = ?rhs2›
    proof (fold GlobalNdet-factorization-union
       [OF ‹B ∩ A ≠ {}› ‹B − A ≠ {}›, unfolded Int-Diff-Un])
      have ‹⊓b∈(B ∩ A). (if b ∈ A then ?term b b else STOP) =
          ⊓b∈(B ∩ A). ?term b b› by (auto intro: mono-GlobalNdet-eq)
       moreover have ‹⊓b∈(B − A). (if b ∈ A then ?term b b else STOP) =
STOP›
        by (simp add: GlobalNdet-is-STOP-iff)
      ultimately show ‹(⊓b∈(B ∩ A). (if b ∈ A then ?term b b else STOP)) ⊓
                  (⊓b∈(B − A). (if b ∈ A then ?term b b else STOP)) ⊓ STOP
= ?rhs2›
        by (metis Mndetprefix-GlobalNdet Int-commute Ndet-assoc Ndet-id)
    qed
    finally show ‹(⊓b ∈ B. ⊓a ∈ A. ?rhs' b a) = ?rhs2› .
  qed
 qed
 finally show ‹?lhs = ?rhs2› .
```

**qed**

**lemma** *Mndetprefix-Sync$_{ptick}$-Mndetprefix-indep* :
‹$A \cap S = \{\} \Longrightarrow B \cap S = \{\} \Longrightarrow$
$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b =$
( *if* $A = \{\}$ *then* $\sqcap b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
*else if* $B = \{\}$ *then* $\sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
*else* $\sqcap b{\in}B.\ \sqcap a{\in}A.$
$((a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ b)))\ \Box$
$((b \rightarrow (a \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))))$›
**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* :
‹$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b = \sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$A \cap S = \{\}$› ‹$B \subseteq S$›
**proof** −
**let** *?rhs′* = ‹$\sqcap b{\in}B.\ \sqcap a{\in}A.\ a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ b)$›
**have** ‹*?lhs* = ( *if* $A = \{\}$ *then* *if* $B \cap S = \{\}$ *then* $\sqcap b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
*else* $(\sqcap x{\in}(B - S) \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ x)) \sqcap STOP$
*else* *if* $B = \{\}$ *then* *if* $A \cap S = \{\}$ *then* $\sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
*else* $(\sqcap x{\in}(A - S) \rightarrow (P\ x\ [\![S]\!]_{\checkmark}\ STOP)) \sqcap STOP$
*else* $\sqcap b{\in}B.\ \sqcap a{\in}A.$
(*if* $a \in S$ *then* *STOP* *else* $(a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ b)))\ \Box$
(*if* $b \in S$ *then* *STOP* *else* $(b \rightarrow (a \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)))\ \Box$
(*if* $a = b \wedge b \in S$ *then* $b \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$ *else* *STOP*))›
(**is** ‹*?lhs* = (*if* $A = \{\}$ *then* *?rhs1* *else if* $B = \{\}$ *then* *?rhs2* *else* *?rhs3*)›)
**by** (*fact Mndetprefix-Sync$_{ptick}$-Mndetprefix*)
**also from** ‹$B \subseteq S$› **have** ‹*?rhs1* = *STOP*›
**by** (*auto simp add*: *Ndet-is-STOP-iff Mndetprefix-GlobalNdet GlobalNdet-is-STOP-iff*)
**also from** ‹$A \cap S = \{\}$› **have** ‹*?rhs2* = $\sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$› **by** *presburger*
**also from** ‹$A \cap S = \{\}$› ‹$B \subseteq S$›
**have** ‹*?rhs3* = $\sqcap b{\in}B.\ \sqcap a{\in}A.\ a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ b)$›
**by** (*intro mono-GlobalNdet-eq*) *auto*
**finally have** ‹*?lhs* = ( *if* $A = \{\}$ *then* *STOP*
*else* *if* $B = \{\}$ *then* $\sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
*else* *?rhs′*)› **.**
**moreover have** ‹$B \neq \{\} \Longrightarrow$ *?rhs′* = $\sqcap a{\in}A.\ a \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B.\ b \rightarrow Q\ b)$›
**by** (*subst GlobalNdet-sets-commute*)
(*simp add*: *Sync$_{ptick}$-distrib-GlobalNdet-left write0-GlobalNdet*)
**moreover have** ‹$\ldots = \sqcap a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b)$›
**by** (*simp add*: *Mndetprefix-GlobalNdet*)
**ultimately show** ‹*?lhs* = *?rhs*› **by** *simp*

**qed**

**end**

**corollary** (**in** $Sync_{ptick}$-*locale*) *Mndetprefix-Sync*$_{ptick}$*-Mndetprefix-right* :
‹$\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \sqcap b{\in}B \rightarrow Q\ b = \sqcap b{\in}B \rightarrow (\sqcap a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$A \subseteq S$› ‹$B \cap S = \{\}$›
**by** (*subst* (*1 2*) *Sync*$_{ptick}$*-locale-sym.Sync*$_{ptick}$*-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*$_{ptick}$ : ‹$c?a{\in}A \rightarrow P\ a\ ;_{\checkmark}\ Q = c?a{\in}A \rightarrow (P\ a\ ;_{\checkmark}\ Q)$›
**by** (*simp add*: *read-def Mprefix-Seq*$_{ptick}$ *comp-def*)

**lemma** *write0-Seq*$_{ptick}$ : ‹$a \rightarrow P\ ;_{\checkmark}\ Q = a \rightarrow (P\ ;_{\checkmark}\ Q)$›
**by** (*simp add*: *write0-def Mprefix-Seq*$_{ptick}$)

**lemma** *ndet-write-Seq*$_{ptick}$ : ‹$c!!a{\in}A \rightarrow P\ a\ ;_{\checkmark}\ Q = c!!a{\in}A \rightarrow (P\ a\ ;_{\checkmark}\ Q)$›
**by** (*simp add*: *ndet-write-is-GlobalNdet-write0 Seq*$_{ptick}$*-distrib-GlobalNdet-right write0-Seq*$_{ptick}$)

**lemma** *write-Seq*$_{ptick}$ : ‹$c!a \rightarrow P\ ;_{\checkmark}\ Q = c!a \rightarrow (P\ ;_{\checkmark}\ Q)$›
**by** (*simp add*: *write0-Seq*$_{ptick}$ *write-is-write0*)

### 8.3.2 Synchronization Product

#### General Laws

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

*read* **lemma** *read-Sync*$_{ptick}$*-read* :
— This is the general case.
‹$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
$(c?a{\in}(A - c\ -`\ S) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b))\ \Box$
$(d?b{\in}(B - d\ -`\ S) \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(\Box x{\in}(c\ `\ A \cap d\ `\ B \cap S) \rightarrow (P\ (\textit{inv-into}\ A\ c\ x)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ x)))$›
(**is** ‹*?lhs* = *?rhs1* $\Box$ *?rhs2* $\Box$ *?rhs3*›)
**if** ‹$c\ `\ A \cap S = \{\} \vee c\ `\ A \subseteq S \vee$ *inj-on c A*›
‹$d\ `\ B \cap S = \{\} \vee d\ `\ B \subseteq S \vee$ *inj-on d B*›

— Assumptions may seem strange, but the motivation is that when $A - c -` S \neq \{\}$ (which is equivalent to $\neg\ c\ `\ A \subseteq S$), we need to ensure that *inv-into* $(A - c -` S)\ c$ is equal to *inv-into* $A\ c$. This requires $A - c -` S = A$ (which is equivalent to $c\ `\ A \cap S = \{\}$) or *inj-on* $c\ A$. We need obviously a similar assumption for $B$.

**proof** −
**have** $* : \langle \bigwedge e\ X.\ e\ `\ (X - e -` S) = e\ `\ X - S \rangle$ **by** *auto*
**have** $\langle ?lhs = (\square a{\in}(c\ `\ A - S) \to (P\ (\textit{inv-into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ (\square x{\in}d\ `\ B \to Q\ (\textit{inv-into}\ B\ d\ x)))) \square$
$(\square b{\in}(d\ `\ B - S) \to ((\square x{\in}c\ `\ A \to P\ (\textit{inv-into}\ A\ c\ x))\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ b))) \square$
$(\square x{\in}(c\ `\ A \cap d\ `\ B \cap S) \to (P\ (\textit{inv-into}\ A\ c\ x)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ x)))\rangle$
(**is** $\langle ?lhs = ?rhs1\,' \square ?rhs2\,' \square ?rhs3 \rangle$)
**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*: $\langle ?this \rangle$)
**next**
**assume** $\langle$*inj-on c A*$\rangle$
**hence** $\langle$*inj-on c* $(A - c -` S)\rangle$ **by** (*simp add*: *inj-on-diff*)
**with** $\langle$*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*: $\langle ?this \rangle$)
**next**
**assume** $\langle$*inj-on d B*$\rangle$
**hence** $\langle$*inj-on d* $(B - d -` S)\rangle$ **by** (*simp add*: *inj-on-diff*)
**with** $\langle$*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** $\langle ?lhs = ?rhs1 \square ?rhs2 \square ?rhs3 \rangle$ **.**
**qed**

**Enforce read** **lemma** *read-Sync*$_{ptick}$*-read-forced-read-left* :
$\langle c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$

$(c?a{\in}(A - c -` S) \to (P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b))\ \Box$
$(d?b{\in}(B - d -` S) \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q\ b))\ \Box$
$(c?x{\in}(A \cap c -` (d `` B \cap S)) \to (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x))\rangle$
(**is** ‹*?lhs* = *?rhs1* □ *?rhs2* □ *?rhs3*›)
**if** ‹$c$ `` $A \cap S = \{\}$ ∨ *inj-on c A*›
‹$d$ `` $B \cap S = \{\}$ ∨ *inj-on d B*›
‹$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow c\ a = d\ b \Longrightarrow d\ b \in S \Longrightarrow a = b$›
**proof** −
**let** *?rhs3′* = ‹$(\Box x{\in}(c `` A \cap d `` B \cap S) \to (P\ (\textit{inv-into}\ A\ c\ x)\ [\![S]\!]_{\checkmark}\ Q\ (\textit{inv-into}\ B\ d\ x)))$›
**have** $*$ : ‹$c `` (A \cap c -` (d `` B \cap S)) = c `` A \cap d `` B \cap S$› **by** *blast*
**have** $**$ : ‹$c `` (A \cap c -` d `` B) = c `` A \cap d `` B$› **by** *blast*
**from** *that*(*1*, *2*) **consider** ‹$c `` A \cap S = \{\} \vee d `` B \cap S = \{\}$›
| ‹*inj-on c A*› ‹*inj-on d B*› **by** *blast*
**hence** ‹*?rhs3′* = *?rhs3*›
**proof** *cases*
**assume** ‹$c `` A \cap S = \{\} \vee d `` B \cap S = \{\}$›
**hence** ‹$c `` A \cap d `` B \cap S = \{\} \wedge A \cap c -` (d `` B \cap S) = \{\}$› **by** *blast*
**thus** ‹*?rhs3′* = *?rhs3*› **by** *simp*
**next**
**assume** ‹*inj-on c A*› ‹*inj-on d B*›
**show** ‹*?rhs3′* = *?rhs3*›
**proof** (*unfold read-def* $*$ *comp-def*,
*intro mono-Mprefix-eq arg-cong2*[**where** $f$ = ‹$\lambda P\ Q.\ P\ [\![S]\!]_{\checkmark}\ Q$›])
**fix** $x$ **assume** ‹$x \in c `` A \cap d `` B \cap S$›
**moreover from** ‹*inj-on c A*› *inj-on-Int*
**have** ‹*inj-on c A* ∧ *inj-on c* $(A \cap c -` (d `` B \cap S))$› **by** *blast*
**ultimately show** ‹$P\ (\textit{inv-into}\ A\ c\ x) = P\ (\textit{inv-into}\ (A \cap c -` (d `` B \cap S))\ c\ x)$›
**by** (*simp add*: *image-iff*, *elim conjE bexE*, *simp*)
**next**
**fix** $x$ **assume** $ : ‹$x \in c `` A \cap d `` B \cap S$›
**then obtain** $a$ $b$ **where** $$ : ‹$x = c\ a$› ‹$a \in A$› ‹$x = d\ b$› ‹$b \in B$› **by** *blast*
**from** ‹*inj-on c A*› *inj-on-Int* **have** $$$ : ‹*inj-on c* $(A \cap c -` (d `` B \cap S))$›
**by** *blast*
**have** ‹*inv-into B d x* = *b*› **by** (*simp add*: $$(*3*, *4*) ‹*inj-on d B*›)
**also have** ‹*b* = *a*› **by** (*metis* $ $$ *Int-iff that*(*3*))
**also have** ‹*a* = *inv-into* $(A \cap c -` (d `` B \cap S))\ c\ x$›
**by** (*metis* $ $$(*1*, *2*) $$$ $*$ *Int-lower1*
‹*inj-on c A*› *inj-on-image-mem-iff inv-into-f-eq*)
**finally have** ‹*inv-into B d x* = *inv-into* $(A \cap c -` (d `` B \cap S))\ c\ x$› **.**
**thus** ‹$Q\ (\textit{inv-into}\ B\ d\ x) = Q\ (\textit{inv-into}\ (A \cap c -` (d `` B \cap S))\ c\ x)$› **by** *simp*
**qed**
**qed**
**moreover have** ‹*?lhs* = *?rhs1* □ *?rhs2* □ *?rhs3′*›
**using** *that*(*1*, *2*) **by** (*subst read-Sync*$_{ptick}$*-read*) *auto*
**ultimately show** ‹*?lhs* = *?rhs1* □ *?rhs2* □ *?rhs3*› **by** *argo*
**qed**

**corollary** (**in** *Sync$_{ptick}$-locale*) *read-Sync$_{ptick}$-read-forced-read-right*:
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *d*?*b*∈*B* → *Q b* =
(*c*?*a*∈(*A* − *c* −‘ *S*) → (*P a* ⟦*S*⟧$_{\checkmark}$ *d*?*b*∈*B* → *Q b*)) □
(*d*?*b*∈(*B* − *d* −‘ *S*) → (*c*?*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *Q b*)) □
(*d*?*x*∈(*B* ∩ *d* −‘ (*c* ‘ *A* ∩ *S*)) → (*P x* ⟦*S*⟧$_{\checkmark}$ *Q x*))›
(**is** ‹*?lhs* = *?rhs1* □ *?rhs2* □ *?rhs3*›)
**if** ‹*c* ‘ *A* ∩ *S* = {} ∨ *inj-on c A*›
‹*d* ‘ *B* ∩ *S* = {} ∨ *inj-on d B*›
‹⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *B* ⟹ *c a* = *d b* ⟹ *d b* ∈ *S* ⟹ *a* = *b*›
**unfolding** *Sync$_{ptick}$-locale-sym.Sync$_{ptick}$-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* = ‹(□)›])

**Special Cases** **lemma** *read-Sync$_{ptick}$-read-subset* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *d*?*b*∈*B* → *Q b* =
□*x*∈(*c* ‘ *A* ∩ *d* ‘ *B*) → (*P* (*inv-into A c x*) ⟦*S*⟧$_{\checkmark}$ *Q* (*inv-into B d x*))›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*d* ‘ *B* ⊆ *S*›
**proof** −
**from** *that* **have** ∗ : ‹*A* − *c* −‘ *S* = {}› ‹*B* − *d* −‘ *S* = {}› **by** *auto*
**from** *that*(*1*) **have** ∗∗ : ‹*c* ‘ *A* ∩ *d* ‘ *B* ∩ *S* = *c* ‘ *A* ∩ *d* ‘ *B*› **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* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *d*?*b*∈*B* → *Q b* = *c*?*x*∈(*A* ∩ *c* −‘ *d* ‘ *B*) → (*P x* ⟦*S*⟧$_{\checkmark}$ *Q x*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*d* ‘ *B* ⊆ *S*› ‹*inj-on c A*› ‹*inj-on d B*›
‹⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *B* ⟹ *c a* = *d b* ⟹ *d b* ∈ *S* ⟹ *a* = *b*›
**proof** −
**from** *that* **have** ∗ : ‹*A* − *c* −‘ *S* = {}› ‹*B* − *d* −‘ *S* = {}› **by** *auto*
**from** *that*(*1*) **have** ∗∗ : ‹*A* ∩ (*c* −‘ *d* ‘ *B* ∩ *c* −‘ *S*) = *A* ∩ *c* −‘ *d* ‘ *B*› **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* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *d*?*b*∈*B* → *Q b* = *d*?*x*∈(*B* ∩ *d* −‘ *c* ‘ *A*) → (*P x* ⟦*S*⟧$_{\checkmark}$ *Q x*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*d* ‘ *B* ⊆ *S*› ‹*inj-on c A*› ‹*inj-on d B*›
‹⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *B* ⟹ *c a* = *d b* ⟹ *d b* ∈ *S* ⟹ *a* = *b*›
**proof** −
**from** *that* **have** ∗ : ‹*B* − *d* −‘ *S* = {}› ‹*A* − *c* −‘ *S* = {}› **by** *auto*
**from** *that*(*1*) **have** ∗∗ : ‹*B* ∩ (*d* −‘ *c* ‘ *A* ∩ *d* −‘ *S*) = *B* ∩ *d* −‘ *c* ‘ *A*› **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* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b =$
$(c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (d?b{\in}B \to Q\ b))) \Box (d?b{\in}B \to ((c?a{\in}A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b))$›
**if** ‹$c$ ‘ $A \cap S = \{\}$› ‹$d$ ‘ $B \cap S = \{\}$›
**proof** −
**from** *that* **have** $*$ : ‹$A - c -‘ S = A$› ‹$B - d -‘ S = B$› **by** *auto*
**from** *that(1)* **have** $**$ : ‹$c$ ‘ $A \cap d$ ‘ $B \cap S = \{\}$› **by** *blast*
**show** *?thesis* **by** (*subst read-Sync$_{ptick}$-read*) (*use that* **in** ‹*simp-all add*: $* **$›)
**qed**

**lemma** *read-Sync$_{ptick}$-read-left* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ (d?b{\in}B \to Q\ b))$›
**if** ‹$c$ ‘ $A \cap S = \{\}$› ‹$d$ ‘ $B \subseteq S$›
**proof** −
**from** *that(1)* **have** $*$ : ‹$A - c -‘ S = A$› ‹$c$ ‘ $A \cap d$ ‘ $B \cap S = \{\}$› **by** *auto*
**from** *that(2)* **have** $**$ : ‹$B - d -‘ S = \{\}$› **by** *blast*
**show** *?thesis* **by** (*subst read-Sync$_{ptick}$-read*)
(*use that* **in** ‹*simp-all add*: $* **$›)
**qed**

**lemma** *read-Sync$_{ptick}$-read-right* :
‹$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = d?b{\in}B \to ((c?a{\in}A \to P\ a)\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**if** ‹$c$ ‘ $A \subseteq S$› ‹$d$ ‘ $B \cap S = \{\}$›
**proof** −
**from** *that(2)* **have** $*$ : ‹$B - d -‘ S = B$› ‹$c$ ‘ $A \cap d$ ‘ $B \cap S = \{\}$› **by** *auto*
**from** *that(1)* **have** $**$ : ‹$A - c -‘ S = \{\}$› **by** *blast*
**show** *?thesis* **by** (*subst read-Sync$_{ptick}$-read*)
(*use that* **in** ‹*simp-all add*: $* **$›)
**qed**

**corollary** *read-Par$_{ptick}$-read* :
‹$c?a{\in}A \to P\ a\ ||_{\checkmark}\ d?b{\in}B \to Q\ b =$
$\Box x{\in}(c$ ‘ $A \cap d$ ‘ $B) \to (P\ (inv\text{-}into\ A\ c\ x)\ ||_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ x))$›
**by** (*simp add*: *read-Sync$_{ptick}$-read-subset*)

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

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

**corollary** *read-Inter$_{ptick}$-read* :

‹⟦*inj-on c A*; *inj-on d B*; ⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *B* ⟹ *c a* = *d b* ⟹ *a* = *b*⟧ ⟹
*c*?*a*∈*A* → *P a* |||✓ *d*?*b*∈*B* → *Q b* =
(*c*?*a*∈*A* → (*P a* |||✓ *d*?*b*∈*B* → *Q b*)) □ (*d*?*b*∈*B* → (*c*?*a*∈*A* → *P a* |||✓ *Q b*))›
**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* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *c*?*b*∈*B* → *Q b* =
(*c*?*a*∈(*A* − *c* −‘ *S*) → (*P a* ⟦*S*⟧✓ *c*?*b*∈*B* → *Q b*)) □
(*c*?*b*∈(*B* − *c* −‘ *S*) → (*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)) □
(*c*?*x*∈(*A* ∩ *B* ∩ *c* −‘ *S*) → (*P x* ⟦*S*⟧✓ *Q x*))›
(**is** ‹*?lhs* = *?rhs1* □ *?rhs2* □ *?rhs3*›)
**if** ‹*c* ‘ *A* ∩ *S* = {} ∨ *inj-on c A*› ‹*c* ‘ *B* ∩ *S* = {} ∨ *inj-on c B*›
‹⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *B* ⟹ *c a* = *c b* ⟹ *c b* ∈ *S* ⟹ *a* = *b*›
**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** ‹*inj-on c* ((*A* ∪ *B*) ∩ *c* −‘ *S*) ⟷
(∀ *a b*. *a* ∈ *A* ⟶ *b* ∈ *B* ⟶ *c a* = *c b* ⟶ *c b* ∈ *S* ⟶ *a* = *b*)›
**by** (*elim disjE*, *simp-all add*: *inj-on-def*)
((*auto*)[*3*], *metis Int-iff Un-iff vimageE vimageI*)

**from** *that*(*3*) **have** ∗ : ‹*A* ∩ (*c* −‘ *c* ‘ *B* ∩ *c* −‘ *S*) = *A* ∩ *B* ∩ *c* −‘ *S*› **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.
‹*inj-on c* (*A* ∪ *B*) ⟹
*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *c*?*b*∈*B* → *Q b* =
(*c*?*a*∈(*A* − *c* −‘ *S*) → (*P a* ⟦*S*⟧✓ *c*?*b*∈*B* → *Q b*)) □
(*c*?*b*∈(*B* − *c* −‘ *S*) → (*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)) □
(*c*?*x*∈(*A* ∩ *B* ∩ *c* −‘ *S*) → (*P x* ⟦*S*⟧✓ *Q x*))›
**by** (*rule read-Sync*$_{ptick}$-*read-forced-read-same-chan*)
(*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* ∪ *B*) is equivalent. The result is not weaker anymore.
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *c*?*b*∈*B* → *Q b* = *c*?*x*∈(*A* ∩ *B*) → (*P x* ⟦*S*⟧✓ *Q x*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*c* ‘ *B* ⊆ *S*› ‹*inj-on c* (*A* ∪ *B*)›
**proof** −
**from** *that*(*3*) **have** ‹*A* ∩ *c* −‘ *c* ‘ *B* = *A* ∩ *B*› **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* :
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b* =
( *if A* = {} *then STOP* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b*
*else* ⊓*a*∈*c* ‘ *A*. (*if a* ∈ *S then STOP else a* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b*)) □
(□*b*∈(*d* ‘ *B* − *S*) → (*a* → *P* (*inv-into A c a*) ⟦*S*⟧✓ *Q* (*inv-into B d b*))) □
(*if a* ∈ *d* ‘ *B* ∩ *S then a* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *Q* (*inv-into B d a*)) *else STOP*))›
**by** (*auto simp add*: *ndet-write-def read-def Mndetprefix-Sync$_{ptick}$-Mprefix intro*: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])

**lemma** *read-Sync$_{ptick}$-ndet-write* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* =
( *if B* = {} *then c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *STOP*
*else* ⊓*b*∈*d* ‘ *B*. (*if b* ∈ *S then STOP else b* → (*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *Q* (*inv-into B d b*))) □
(□*a*∈(*c* ‘ *A* − *S*) → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *b* → *Q* (*inv-into B d b*))) □
(*if b* ∈ *c* ‘ *A* ∩ *S then b* → (*P* (*inv-into A c b*) ⟦*S*⟧✓ *Q* (*inv-into B d b*)) *else STOP*))›
**by** (*auto simp add*: *ndet-write-def read-def Mprefix-Sync$_{ptick}$-Mndetprefix intro*: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])

**lemma** *ndet-write-Sync$_{ptick}$-read-subset* :
‹*c* ‘ *A* ⊆ *S* ⟹ *d* ‘ *B* ⊆ *S* ⟹
*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b* =
( *if c* ‘ *A* ⊆ *d* ‘ *B then* ⊓*a*∈*c* ‘ *A* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *Q* (*inv-into B d a*))
*else* (⊓*a*∈(*c* ‘ *A* ∩ *d* ‘ *B*) → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *Q* (*inv-into B d a*))) ⊓ *STOP*)›
**by** (*simp add*: *read-def ndet-write-def Mndetprefix-Sync$_{ptick}$-Mprefix-subset*)

**lemma** *read-Sync$_{ptick}$-ndet-write-subset* :
‹*c* ‘ *A* ⊆ *S* ⟹ *d* ‘ *B* ⊆ *S* ⟹
*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* =
( *if d* ‘ *B* ⊆ *c* ‘ *A then* ⊓*b*∈*d* ‘ *B* → (*P* (*inv-into A c b*) ⟦*S*⟧✓ *Q* (*inv-into B d b*))
*else* (⊓*b*∈(*c* ‘ *A* ∩ *d* ‘ *B*) → (*P* (*inv-into A c b*) ⟦*S*⟧✓ *Q* (*inv-into B d b*))) ⊓ *STOP*)›
**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*:
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *c*?*b*∈*B* → *Q b* =
(*if A* ⊆ *B then c*!!*a*∈*A* → (*P a* ⟦*S*⟧✓ *Q a*) *else* (*c*!!*a*∈(*A* ∩ *B*) → (*P a* ⟦*S*⟧✓ *Q a*)) □ *STOP*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*c* ‘ *B* ⊆ *S*› ‹*inj-on c* (*A* ∪ *B*)›
**proof** −
**from** ‹*inj-on c* (*A* ∪ *B*)› **have** ∗ : ‹*c* ‘ *A* ⊆ *c* ‘ *B* ⟷ *A* ⊆ *B*›
**by** (*auto simp add*: *inj-on-eq-iff*)
**from** ‹*inj-on c* (*A* ∪ *B*)› **have** ∗∗ : ‹*c* ‘ *A* ∩ *c* ‘ *B* = *c* ‘ (*A* ∩ *B*)›
**by** (*auto simp add*: *inj-on-Un*)
**from** ‹*inj-on c* (*A* ∪ *B*)› **show** *?thesis*
**by** (*unfold ndet-write-Sync$_{ptick}$-read-subset*[*OF* ‹*c* ‘ *A* ⊆ *S*› ‹*c* ‘ *B* ⊆ *S*›] ∗ ∗∗)
(*auto simp add*: *ndet-write-def inj-on-Un inj-on-Int*
*intro*!: *mono-Mndetprefix-eq arg-cong2*[**where** *f* = ‹(⊓)›])
**qed**

**corollary** (**in** *Sync$_{ptick}$-locale*) *read-Sync$_{ptick}$-ndet-write-subset-same-chan*:
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *c*!!*b*∈*B* → *Q b* =
(*if B* ⊆ *A then c*!!*b*∈*B* → (*P b* ⟦*S*⟧✓ *Q b*) *else* (*c*!!*b*∈(*A* ∩ *B*) → (*P b* ⟦*S*⟧✓ *Q b*)) □ *STOP*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*c* ‘ *B* ⊆ *S*› ‹*inj-on c* (*A* ∪ *B*)›
**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* :
‹*c* ‘ *A* ∩ *S* = {} ⟹ *d* ‘ *B* ∩ *S* = {} ⟹
*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b* =
( *if A* = {} *then d*?*b*∈*B* → (*STOP* ⟦*S*⟧✓ *Q b*)
*else* □*a*∈*c* ‘ *A*. (*a* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b*)) □
(*d*?*b*∈*B* → (*a* → *P* (*inv-into A c a*) ⟦*S*⟧✓ *Q b*)))›
**by** (*auto simp add*: *ndet-write-def read-def Mndetprefix-Sync$_{ptick}$-Mprefix-indep comp-def*
*intro*: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])

**lemma** *read-Sync$_{ptick}$-ndet-write-indep* :
‹*c* ‘ *A* ∩ *S* = {} ⟹ *d* ‘ *B* ∩ *S* = {} ⟹
*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* =
( *if B* = {} *then c*?*a*∈*A* → (*P a* ⟦*S*⟧✓ *STOP*)
*else* □*b*∈*d* ‘ *B*. (*b* → (*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *Q* (*inv-into B d b*))) □
(*c*?*a*∈*A* → (*P a* ⟦*S*⟧✓ *b* → *Q* (*inv-into B d b*))))›
**by** (*auto simp add*: *ndet-write-def read-def Mprefix-Sync$_{ptick}$-Mndetprefix-indep comp-def*
*intro*: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])

**lemma** *ndet-write-Sync$_{ptick}$-read-left* :
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b* = *c*!!*a*∈*A* → (*P a* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b*)›

(**is** ‹*?lhs* = *?rhs*›) **if** ‹*c* ‘ *A* ∩ *S* = {}› ‹*d* ‘ *B* ⊆ *S*›
**proof** −
**from** *that* **have** ‹*?lhs* = (*if A* = {} *then STOP* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b else ?rhs*)›
**by** (*auto simp add*: *ndet-write-def read-def*
*Mndetprefix-Sync*$_{ptick}$*-Mprefix-left comp-def*
*intro*: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])
**also have** ‹... = *?rhs*›
**by** (*simp add*: *read-def ndet-write-def Mprefix-is-STOP-iff*
*STOP-Sync*$_{ptick}$*-Mprefix that*(*2*))
**finally show** ‹*?lhs* = *?rhs*› **.**
**qed**

**lemma** *read-Sync*$_{ptick}$*-ndet-write-left* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* = *c*?*a*∈*A* → (*P a* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b*)›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*c* ‘ *A* ∩ *S* = {}› ‹*d* ‘ *B* ⊆ *S*›
**proof** −
**from** *that* **have** ‹*?lhs* = (*if B* = {} *then* (*c*?*a*∈*A* → *P a*) ⟦*S*⟧✓ *STOP else ?rhs*)›
**by** (*auto simp add*: *ndet-write-def read-def*
*Mprefix-Sync*$_{ptick}$*-Mndetprefix-left comp-def*
*intro*: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›])
**also have** ‹... = *?rhs*›
**by** (*simp add*: *read-def comp-def*)
(*use Mprefix-Sync*$_{ptick}$*-Mprefix-left that*(*1*) **in** *force*)
**finally show** ‹*?lhs* = *?rhs*› **.**
**qed**

**corollary** (**in** *Sync*$_{ptick}$*-locale*) *ndet-write-Sync*$_{ptick}$*-read-right* :
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b* = *d*?*b*∈*B* → (*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*d* ‘ *B* ∩ *S* = {}›
**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* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* = *d*!!*b*∈*B* → (*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *Q b*)›
**if** ‹*c* ‘ *A* ⊆ *S*› ‹*d* ‘ *B* ∩ *S* = {}›
**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* :
‹*c*!*a* → *P* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b* =
(*if c a* ∈ *S then STOP else c*!*a* → (*P* ⟦*S*⟧✓ *d*?*b*∈*B* → *Q b*)) □
(□*b*∈(*d* ‘ *B* − *S*) → (*c*!*a* → *P* ⟦*S*⟧✓ *Q* (*inv-into B d b*))) □
(*if c a* ∈ *d* ‘ *B* ∩ *S then c*!*a* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d* (*c a*))) *else STOP*)›
**by** (*subst ndet-write-Sync*$_{ptick}$*-read*[**where** *A* = ‹{*a*}›, *simplified*])
(*simp add*: *write-is-write0 image-iff*)

**lemma** *read-Sync*$_{ptick}$*-write* :
‹*c*?*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!*b* → *Q* =

$(\mathit{if}\ d\ b \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ d!b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q))\ \Box$
$(\Box a{\in}(c\ `\ A - S) \rightarrow (P\ (\mathit{inv\text{-}into}\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q))\ \Box$
$(\mathit{if}\ d\ b \in c\ `\ A \cap S\ \mathit{then}\ d!b \rightarrow (P\ (\mathit{inv\text{-}into}\ A\ c\ (d\ b))\ [\![S]\!]_{\checkmark}\ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
**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* :
$\langle c\ a \in S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
$(\mathit{if}\ c\ a \in d\ `\ B\ \mathit{then}\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ (\mathit{inv\text{-}into}\ B\ d\ (c\ a)))\ \mathit{else}\ \mathit{STOP})\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-read*)
(*metis Det-STOP Det-commute Diff-eq-empty-iff Mprefix-empty*)

**lemma** *read-Sync*$_{ptick}$*-write-subset* :
$\langle c\ `\ A \subseteq S \Longrightarrow d\ b \in S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q =$
$(\mathit{if}\ d\ b \in c\ `\ A\ \mathit{then}\ d!b \rightarrow (P\ (\mathit{inv\text{-}into}\ A\ c\ (d\ b))\ [\![S]\!]_{\checkmark}\ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
**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*:
$\langle c\ a \in S \Longrightarrow c\ `\ B \subseteq S \Longrightarrow \mathit{inj\text{-}on}\ c\ (\mathit{insert}\ a\ B) \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ c?b{\in}B \rightarrow Q\ b = (\mathit{if}\ a \in B\ \mathit{then}\ c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q\ a)\ \mathit{else}\ \mathit{STOP})\rangle$
**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*:
$\langle c\ `\ A \subseteq S \Longrightarrow c\ b \in S \Longrightarrow \mathit{inj\text{-}on}\ c\ (\mathit{insert}\ b\ A) \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ c!b \rightarrow Q = (\mathit{if}\ b \in A\ \mathit{then}\ c!b \rightarrow (P\ b\ [\![S]\!]_{\checkmark}\ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
**by** (*subst read-Sync*$_{ptick}$*-ndet-write-subset-same-chan*[**where** $B = \langle\{b\}\rangle$, *simplified*]) *simp-all*

**lemma** *write-Sync*$_{ptick}$*-read-indep* :
$\langle c\ a \notin S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b =$
$(c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b))\ \Box\ (d?b{\in}B \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ b))\rangle$
**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* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow d\ b \notin S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q =$
$(d!b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q))\ \Box\ (c?a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q))\rangle$
**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* :
‹$c\ a \notin S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b = c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b)$›
**by** (*subst ndet-write-Sync$_{ptick}$-read-left*[**where** $A = \langle\{a\}\rangle$, *simplified*]) *simp-all*

**lemma** *read-Sync$_{ptick}$-write-left* :
‹$c\ `\ A \cap S = \{\} \Longrightarrow d\ b \in S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = c?a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)$›
**by** (*subst read-Sync$_{ptick}$-ndet-write-left*[**where** $B = \langle\{b\}\rangle$, *simplified*]) *simp-all*

**lemma** *write-Sync$_{ptick}$-read-right* :
‹$c\ a \in S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \rightarrow Q\ b = d?b{\in}B \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**by** (*subst ndet-write-Sync$_{ptick}$-read-right*[**where** $A = \langle\{a\}\rangle$, *simplified*]) *simp-all*

**lemma** *read-Sync$_{ptick}$-write-right* :
‹$c\ `\ A \subseteq S \Longrightarrow d\ b \notin S \Longrightarrow$
$c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (c?a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)$›
**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* :
‹$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
( *if* $A = \{\}$ *then* *if* $d\ `\ B \cap S = \{\}$ *then* $d!!b{\in}B \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ b)$
*else* $(\sqcap x{\in}d\ `\ (B - d -`\ S) \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d\ x)))$
$\sqcap\ STOP$
*else* *if* $B = \{\}$ *then* *if* $c\ `\ A \cap S = \{\}$ *then* $c!!a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ STOP)$
*else* $(\sqcap x{\in}c\ `\ (A - c -`\ S) \rightarrow (P\ (inv\text{-}into\ A\ c\ x)\ [\![S]\!]_{\checkmark}$
$STOP)) \sqcap STOP$
*else* $\sqcap b{\in}d\ `\ B.\ \sqcap a{\in}c\ `\ A.$
$(if\ a \in S\ then\ STOP\ else\ a \rightarrow (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ b \rightarrow Q\ (inv\text{-}into$
$B\ d\ b))) \Box$
$(if\ b \in S\ then\ STOP\ else\ b \rightarrow (a \rightarrow P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into$
$B\ d\ b))) \Box$
$(if\ a = b \wedge b \in S\ then\ b \rightarrow (P\ (inv\text{-}into\ A\ c\ a)\ [\![S]\!]_{\checkmark}\ Q\ (inv\text{-}into\ B\ d$
$b))\ else\ STOP))$›
**proof** −
**have** ‹$d\ `\ (B - d -`\ S) = d\ `\ B - S$› ‹$c\ `\ (A - c -`\ S) = c\ `\ A - S$› **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* :
‹$c\ `\ A \subseteq S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$

( *if* $\exists\, b.\ c\ `\ A = \{b\} \wedge d\ `\ B = \{b\}$
*then* (*THE b. d ` B* = {*b*}) → (*P* (*inv-into A c* (*THE a. c ` A* = {*a*})) $[\![S]\!]_{\checkmark}$ *Q* (*inv-into B d* (*THE b. d ` B* = {*b*})))
*else* (⊓*x*∈(*c ` A* ∩ *d ` B*) → (*P* (*inv-into A c x*) $[\![S]\!]_{\checkmark}$ *Q* (*inv-into B d x*))) ⊓ *STOP*)›
**by** (*auto simp add*: *ndet-write-def Mndetprefix-Sync*$_{ptick}$*-Mndetprefix-subset*)

**corollary** *inj-on-ndet-write-Sync*$_{ptick}$*-ndet-write-subset* :
‹*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* =
( *if* $\exists\, b.\ c\ `\ A = \{b\} \wedge d\ `\ B = \{b\}$
*then d* (*THE b. B* = {*b*}) → (*P* (*THE a. A* = {*a*}) $[\![S]\!]_{\checkmark}$ *Q* (*THE b. B* = {*b*}))
*else* (⊓*x*∈(*c ` A* ∩ *d ` B*) → (*P* (*inv-into A c x*) $[\![S]\!]_{\checkmark}$ *Q* (*inv-into B d x*))) ⊓ *STOP*)›
**if** ‹*inj-on c A*› ‹*inj-on d B*› ‹*c ` A* ⊆ *S*› ‹*d ` B* ⊆ *S*›
**proof** −
**from** *that*(*1*) **have** ‹*c ` A* = {*a'*} ⟹ ∃!*a*. *A* = {*a*} ∧ *a'* = *c a*› **for** *a'*
**by** (*fastforce elim*!: *inj-img-insertE*)
**moreover from** *that*(*2*) **have** ‹*d ` B* = {*b'*} ⟹ ∃!*b*. *B* = {*b*} ∧ *b'* = *d b*› **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* = ‹(→)›] *arg-cong2*[**where** *f* = ‹λ*P Q*. *P* $[\![S]\!]_{\checkmark}$ *Q*›])
**qed**

**lemma** *ndet-write-Sync*$_{ptick}$*-ndet-write-indep* :
‹*c ` A* ∩ *S* = {} ⟹ *d ` B* ∩ *S* = {} ⟹
*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* =
( *if A* = {} *then d*!!*b*∈*B* → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)
*else if B* = {} *then c*!!*a*∈*A* → (*P a* $[\![S]\!]_{\checkmark}$ *STOP*)
*else* ⊓*b*∈*d ` B*. ⊓*a*∈*c ` A*.
((*a* → (*P* (*inv-into A c a*) $[\![S]\!]_{\checkmark}$ *b* → *Q* (*inv-into B d b*)))) □
((*b* → (*a* → *P* (*inv-into A c a*) $[\![S]\!]_{\checkmark}$ *Q* (*inv-into B d b*)))))›
**by** (*auto simp add*: *ndet-write-Sync*$_{ptick}$*-ndet-write disjoint-iff intro*!: *mono-GlobalNdet-eq*)

**lemma** *ndet-write-Sync*$_{ptick}$*-ndet-write-left* :
‹*c ` A* ∩ *S* = {} ⟹ *d ` B* ⊆ *S* ⟹
*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* = *c*!!*a*∈*A* → (*P a* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b*)›
**by** (*simp add*: *ndet-write-def Mndetprefix-Sync*$_{ptick}$*-Mndetprefix-left comp-def*)

**lemma** *ndet-write-Sync*$_{ptick}$*-ndet-write-right* :
‹*c ` A* ⊆ *S* ⟹ *d ` B* ∩ *S* = {} ⟹
*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* = *d*!!*b*∈*B* → (*c*!!*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *Q b*)›
**by** (*simp add*: *ndet-write-def Mndetprefix-Sync*$_{ptick}$*-Mndetprefix-right comp-def*)

*ndet-write* **and** *write* **lemma** *write-Sync$_{ptick}$-ndet-write* :
‹*c*!*a* → *P* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* =
( *if B* = {} *then c*!*a* → *P* ⟦*S*⟧✓ *STOP*
*else* ⊓*b*∈*d* ‘ *B*. (*if b* ∈ *S then STOP else b* → (*c*!*a* → *P* ⟦*S*⟧✓ *Q* (*inv-into B d b*))) □
(*if c a* ∈ *S then STOP else c*!*a* → (*P* ⟦*S*⟧✓ *b* → *Q* (*inv-into B d b*))) □
(*if b* = *c a* ∧ *c a* ∈ *S then c*!*a* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d* (*c a*))) *else STOP*))›
**by** (*subst read-Sync$_{ptick}$-ndet-write*[**where** *A* = ‹{*a*}›, *simplified*],
*auto simp add*: *write-def Mprefix-singl split*: *if-split-asm*
*intro*!: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›] *mono-Mprefix-eq*)
(*simp add*: *insert-Diff-if write0-def*)

**lemma** *ndet-write-Sync$_{ptick}$-write* :
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!*b* → *Q* =
( *if A* = {} *then STOP* ⟦*S*⟧✓ *d*!*b* → *Q*
*else* ⊓*a*∈*c* ‘ *A*. (*if a* ∈ *S then STOP else a* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *d*!*b* → *Q*)) □
(*if d b* ∈ *S then STOP else d*!*b* → (*a* → *P* (*inv-into A c a*) ⟦*S*⟧✓ *Q*)) □
(*if a* = *d b* ∧ *d b* ∈ *S then d*!*b* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *Q*) *else STOP*))›
**by** (*subst ndet-write-Sync$_{ptick}$-read*[**where** *B* = ‹{*b*}›, *simplified*],
*auto simp add*: *write-def Mprefix-singl split*: *if-split-asm*
*intro*!: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(□)›] *mono-Mprefix-eq*)
(*simp add*: *insert-Diff-if write0-def*)

**lemma** *write-Sync$_{ptick}$-ndet-write-subset* :
‹*c*!*a* → *P* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* =
( *if c a* ∉ *d* ‘ *B then STOP else if d* ‘ *B* = {*c a*} *then c*!*a* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d* (*c a*)))
*else* (*c*!*a* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d* (*c a*)))) ⊓ *STOP*)› **if** ‹*c a* ∈ *S*› ‹*d* ‘ *B* ⊆ *S*›
**proof** (*subst read-Sync$_{ptick}$-ndet-write-subset*[**where** *A* = ‹{*a*}›, *simplified*])
**from** ‹*c a* ∈ *S*› **show** ‹*c a* ∈ *S*› **.**
**next**
**from** ‹*d* ‘ *B* ⊆ *S*› **show** ‹*d* ‘ *B* ⊆ *S*› **.**
**next**
**show** ‹( *if d* ‘ *B* ⊆ {*c a*} *then* ⊓*b*∈*d* ‘ *B* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d b*))
*else* (⊓*b*∈(*c* ‘ {*a*} ∩ *d* ‘ *B*) → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d b*))) ⊓ *STOP*) =
( *if c a* ∉ *d* ‘ *B then STOP else if d* ‘ *B* = {*c a*} *then c*!*a* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d* (*c a*)))
*else* (*c*!*a* → (*P* ⟦*S*⟧✓ *Q* (*inv-into B d* (*c a*)))) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if c a* ∉ *d* ‘ *B then STOP else if d* ‘ *B* = {*c a*} *then ?rhs else ?rhs* ⊓ *STOP*)›)
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*c a* ∉ *d* ‘ *B* ⟹ *?lhs* = *STOP*›

**by** (*auto simp add*: *GlobalNdet-is-STOP-iff image-subset-iff image-iff*)
**next**
**show** ‹¬ *c a* ∉ *d* ‘ *B* ⟹ *?lhs* = (*if d* ‘ *B* = {*c a*} *then ?rhs else ?rhs* ⊓ *STOP*)›
**by** (*auto simp add*: *image-subset-iff Ndet-is-STOP-iff write-is-write0*)
**qed**
**qed**

**corollary** (**in** $Sync_{ptick}$-*locale*) *ndet-write-*$Sync_{ptick}$*-write-subset* :
‹(*c*!!*a*∈*A* → *P a*) ⟦*S*⟧✓ (*d*!*b* → *Q*) =
( *if d b* ∉ *c* ‘ *A then STOP else if c* ‘ *A* = {*d b*} *then d*!*b* → (*P* (*inv-into A c* (*d b*)) ⟦*S*⟧✓ *Q*)
*else* (*d*!*b* → (*P* (*inv-into A c* (*d b*)) ⟦*S*⟧✓ *Q*)) ⊓ *STOP*)› **if** ‹*c* ‘ *A* ⊆ *S*› ‹*d b* ∈ *S*›
**by** (*subst* (*1 2 3*) $Sync_{ptick}$*-locale-sym.*$Sync_{ptick}$*-sym*)
(*simp add*: $Sync_{ptick}$*-locale-sym.write-*$Sync_{ptick}$*-ndet-write-subset that*)

**lemma** *write-*$Sync_{ptick}$*-ndet-write-indep* :
‹*c a* ∉ *S* ⟹ *d* ‘ *B* ∩ *S* = {} ⟹
*c*!*a* → *P* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* =
( *if B* = {} *then c*!*a* → (*P* ⟦*S*⟧✓ *STOP*)
*else* ⊓*b*∈*d* ‘ *B*. (*c*!*a* → (*P* ⟦*S*⟧✓ *b* → *Q* (*inv-into B d b*))) □
(*b* → (*c*!*a* → *P* ⟦*S*⟧✓ *Q* (*inv-into B d b*))))›
**by** (*subst ndet-write-*$Sync_{ptick}$*-ndet-write-indep*[**where** *A* = ‹{*a*}›, *simplified*])
(*auto simp add*: *write-is-write0 intro*: *mono-GlobalNdet-eq*)

**lemma** *ndet-write-*$Sync_{ptick}$*-write-indep* :
‹*c* ‘ *A* ∩ *S* = {} ⟹ *d b* ∉ *S* ⟹
*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!*b* → *Q* =
( *if A* = {} *then d*!*b* → (*STOP* ⟦*S*⟧✓ *Q*)
*else* ⊓*a*∈*c* ‘ *A*. (*a* → (*P* (*inv-into A c a*) ⟦*S*⟧✓ *d*!*b* → *Q*)) □
(*d*!*b* → (*a* → *P* (*inv-into A c a*) ⟦*S*⟧✓ *Q*)))›
**by** (*subst ndet-write-*$Sync_{ptick}$*-ndet-write-indep*[**where** *B* = ‹{*b*}›, *simplified*])
(*auto simp add*: *write-is-write0 intro*: *mono-GlobalNdet-eq*)

**lemma** *write-*$Sync_{ptick}$*-ndet-write-left* :
‹*c a* ∉ *S* ⟹ *d* ‘ *B* ⊆ *S* ⟹ *c*!*a* → *P* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* = *c*!*a* → (*P* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b*)›
**by** (*subst ndet-write-*$Sync_{ptick}$*-ndet-write-left*[**where** *A* = ‹{*a*}›, *simplified*]) *simp-all*

**lemma** *ndet-write-*$Sync_{ptick}$*-write-left* :
‹*c* ‘ *A* ∩ *S* = {} ⟹ *d b* ∈ *S* ⟹ *c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *d*!*b* → *Q* = *c*!!*a*∈*A* → (*P a* ⟦*S*⟧✓ *d*!*b* → *Q*)›
**by** (*subst ndet-write-*$Sync_{ptick}$*-ndet-write-left*[**where** *B* = ‹{*b*}›, *simplified*]) *simp-all*

**lemma** *write-*$Sync_{ptick}$*-ndet-write-right* :
‹*c a* ∈ *S* ⟹ *d* ‘ *B* ∩ *S* = {} ⟹ *c*!*a* → *P* ⟦*S*⟧✓ *d*!!*b*∈*B* → *Q b* = *d*!!*b*∈*B* →

$(c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q\ b)\rangle$
**by** (*subst ndet-write-Sync*$_{ptick}$*-ndet-write-right*[**where** $A = \langle\{a\}\rangle$, *simplified*])
*simp-all*

**lemma** *ndet-write-Sync*$_{ptick}$*-write-right* :
$\langle c\ `\ A \subseteq S \Longrightarrow d\ b \notin S \Longrightarrow c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)\rangle$
**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* :
$\langle c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q =$
(*if* $d\ b \in S$ *then STOP else* $d!b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)) \Box$
(*if* $c\ a \in S$ *then STOP else* $c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)) \Box$
(*if* $c\ a = d\ b \wedge d\ b \in S$ *then* $c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q)$ *else STOP*)$\rangle$
**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* :
$\langle c!a \rightarrow P\ |||_{\checkmark}\ d!b \rightarrow Q = (c!a \rightarrow (P\ |||_{\checkmark}\ d!b \rightarrow Q)) \Box (d!b \rightarrow (c!a \rightarrow P\ |||_{\checkmark}\ Q))\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-write Det-commute*)

**lemma** *write-Par*$_{ptick}$*-write* :
$\langle c!a \rightarrow P\ ||_{\checkmark}\ d!b \rightarrow Q = (\textit{if } c\ a = d\ b \textit{ then } c!a \rightarrow (P\ ||_{\checkmark}\ Q) \textit{ else STOP})\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-write*)

**lemma** *write-Sync*$_{ptick}$*-write-subset* :
$\langle c\ a \in S \Longrightarrow d\ b \in S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = (\textit{if } c\ a = d\ b \textit{ then } c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q) \textit{ else STOP})\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-write*)

**lemma** *write-Sync*$_{ptick}$*-write-indep* :
$\langle c\ a \notin S \Longrightarrow d\ b \notin S \Longrightarrow$
$c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = (c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)) \Box (d!b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q))\rangle$
**by** (*simp add*: *Det-commute write-Sync*$_{ptick}$*-write*)

**lemma** *write-Sync*$_{ptick}$*-write-left* :
$\langle c\ a \notin S \Longrightarrow d\ b \in S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = c!a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q)\rangle$
**by** (*auto simp add*: *write-Sync*$_{ptick}$*-write*)

**lemma** *write-Sync*$_{ptick}$*-write-right* :
$\langle c\ a \in S \Longrightarrow d\ b \notin S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!b \rightarrow Q = d!b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)\rangle$
**by** (*auto simp add*: *write-Sync*$_{ptick}$*-write*)

*read* **and** $(\rightarrow)$**.** **lemma** *write0-Sync$_{ptick}$-read* :
$\langle a \rightarrow P \ [\![S]\!]_{\checkmark} \ d?b{\in}B \rightarrow Q \ b =$
$(\mathit{if}\ a \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ a \rightarrow (P \ [\![S]\!]_{\checkmark} \ d?b{\in}B \rightarrow Q \ b)) \ \Box$
$(\Box b{\in}(d \ `\ B - S) \rightarrow (a \rightarrow P \ [\![S]\!]_{\checkmark} \ Q \ (\mathit{inv\text{-}into}\ B\ d\ b))) \ \Box$
$(\mathit{if}\ a \in d \ `\ B \cap S\ \mathit{then}\ a \rightarrow (P \ [\![S]\!]_{\checkmark} \ Q \ (\mathit{inv\text{-}into}\ B\ d\ a))\ \mathit{else}\ \mathit{STOP})\rangle$
**by** (*simp add*: *write-Sync$_{ptick}$-read*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *read-Sync$_{ptick}$-write0* :
$\langle c?a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ b \rightarrow Q =$
$(\mathit{if}\ b \in S\ \mathit{then}\ \mathit{STOP}\ \mathit{else}\ b \rightarrow (c?a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ Q)) \ \Box$
$(\Box a{\in}(c \ `\ A - S) \rightarrow (P \ (\mathit{inv\text{-}into}\ A\ c\ a) \ [\![S]\!]_{\checkmark} \ b \rightarrow Q)) \ \Box$
$(\mathit{if}\ b \in c \ `\ A \cap S\ \mathit{then}\ b \rightarrow (P \ (\mathit{inv\text{-}into}\ A\ c\ b) \ [\![S]\!]_{\checkmark} \ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
**by** (*simp add*: *read-Sync$_{ptick}$-write*[**where** $d = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *write0-Sync$_{ptick}$-read-subset* :
$\langle a \in S \Longrightarrow d \ `\ B \subseteq S \Longrightarrow$
$a \rightarrow P \ [\![S]\!]_{\checkmark} \ d?b{\in}B \rightarrow Q \ b =$
$(\mathit{if}\ a \in d \ `\ B\ \mathit{then}\ a \rightarrow (P \ [\![S]\!]_{\checkmark} \ Q \ (\mathit{inv\text{-}into}\ B\ d\ a))\ \mathit{else}\ \mathit{STOP})\rangle$
**by** (*simp add*: *write-Sync$_{ptick}$-read-subset*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *read-Sync$_{ptick}$-write0-subset* :
$\langle c \ `\ A \subseteq S \Longrightarrow b \in S \Longrightarrow$
$c?a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ b \rightarrow Q =$
$(\mathit{if}\ b \in c \ `\ A\ \mathit{then}\ b \rightarrow (P \ (\mathit{inv\text{-}into}\ A\ c\ b) \ [\![S]\!]_{\checkmark} \ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
**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*:
$\langle a \in S \Longrightarrow B \subseteq S \Longrightarrow$
$a \rightarrow P \ [\![S]\!]_{\checkmark} \ id?b{\in}B \rightarrow Q \ b = (\mathit{if}\ a \in B\ \mathit{then}\ a \rightarrow (P \ [\![S]\!]_{\checkmark} \ Q \ a)\ \mathit{else}\ \mathit{STOP})\rangle$
**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*:
$\langle A \subseteq S \Longrightarrow b \in S \Longrightarrow$
$id?a{\in}A \rightarrow P \ a \ [\![S]\!]_{\checkmark} \ b \rightarrow Q = (\mathit{if}\ b \in A\ \mathit{then}\ b \rightarrow (P \ b \ [\![S]\!]_{\checkmark} \ Q)\ \mathit{else}\ \mathit{STOP})\rangle$
**by** (*simp add*: *read-Sync$_{ptick}$-write-subset-same-chan*
[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *write0-Sync$_{ptick}$-read-indep* :
$\langle a \notin S \Longrightarrow d \ `\ B \cap S = \{\} \Longrightarrow$
$a \rightarrow P \ [\![S]\!]_{\checkmark} \ d?b{\in}B \rightarrow Q \ b =$
$(a \rightarrow (P \ [\![S]\!]_{\checkmark} \ d?b{\in}B \rightarrow Q \ b)) \ \Box \ (d?b{\in}B \rightarrow (a \rightarrow P \ [\![S]\!]_{\checkmark} \ Q \ b))\rangle$
**by** (*simp add*: *write-Sync$_{ptick}$-read-indep*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *read-Sync$_{ptick}$-write0-indep* :

$\langle c \mathbin{`} A \cap S = \{\} \Longrightarrow b \notin S \Longrightarrow$
$c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q =$
$(b \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q)) \Box (c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ b \to Q))\rangle$
**by** (*simp add*: *read-Sync*$_{ptick}$*-write-indep*[**where** $d = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *write0-Sync*$_{ptick}$*-read-left* :
$\langle a \notin S \Longrightarrow d \mathbin{`} B \subseteq S \Longrightarrow a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = a \to (P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b)\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-read-left*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *read-Sync*$_{ptick}$*-write0-left* :
$\langle c \mathbin{`} A \cap S = \{\} \Longrightarrow b \in S \Longrightarrow c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q = c?a{\in}A \to (P\ a\ [\![S]\!]_{\checkmark}\ b \to Q)\rangle$
**by** (*simp add*: *read-Sync*$_{ptick}$*-write-left*[**where** $d = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *write0-Sync*$_{ptick}$*-read-right* :
$\langle a \in S \Longrightarrow d \mathbin{`} B \cap S = \{\} \Longrightarrow a \to P\ [\![S]\!]_{\checkmark}\ d?b{\in}B \to Q\ b = d?b{\in}B \to (a \to P\ [\![S]\!]_{\checkmark}\ Q\ b)\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-read-right*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *read-Sync*$_{ptick}$*-write0-right* :
$\langle c \mathbin{`} A \subseteq S \Longrightarrow b \notin S \Longrightarrow c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q = b \to (c?a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ Q)\rangle$
**by** (*simp add*: *read-Sync*$_{ptick}$*-write-right*[**where** $d = id$, *unfolded write-is-write0*, *simplified*])

*ndet-write* **and** $(\to)$ **lemma** *write0-Sync*$_{ptick}$*-ndet-write* :
$\langle a \to P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \to Q\ b =$
$($ *if* $B = \{\}$ *then* $a \to P\ [\![S]\!]_{\checkmark}$ *STOP*
*else* $\sqcap b{\in}d \mathbin{`} B.$ (*if* $b \in S$ *then STOP else* $b \to (a \to P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d\ b$))) $\Box$
(*if* $a \in S$ *then STOP else* $a \to (P\ [\![S]\!]_{\checkmark}\ b \to Q$ (*inv-into* $B\ d\ b$))) $\Box$
(*if* $b = a \wedge a \in S$ *then* $a \to (P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into* $B\ d\ a$)) *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* :
$\langle c!!a{\in}A \to P\ a\ [\![S]\!]_{\checkmark}\ b \to Q =$
$($ *if* $A = \{\}$ *then STOP* $[\![S]\!]_{\checkmark}\ b \to Q$
*else* $\sqcap a{\in}c \mathbin{`} A.$ (*if* $a \in S$ *then STOP else* $a \to (P$ (*inv-into* $A\ c\ a$) $[\![S]\!]_{\checkmark}\ b \to Q)) \Box$
(*if* $b \in S$ *then STOP else* $b \to (a \to P$ (*inv-into* $A\ c\ a$) $[\![S]\!]_{\checkmark}\ Q)) \Box$
(*if* $a = b \wedge b \in S$ *then* $b \to (P$ (*inv-into* $A\ c\ a$) $[\![S]\!]_{\checkmark}\ Q$) *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* :
$\langle a \in S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow$
$a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
$($ *if* $a \notin d\ `\ B$ *then STOP else if* $d\ `\ B = \{a\}$ *then* $a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q$ (*inv-into B d a*))
*else* $(a \rightarrow (P\ [\![S]\!]_{\checkmark}\ Q$ (*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* :
$\langle c\ `\ A \subseteq S \Longrightarrow b \in S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
$($ *if* $b \notin c\ `\ A$ *then STOP else if* $c\ `\ A = \{b\}$ *then* $b \rightarrow (P$ (*inv-into A c b*) $[\![S]\!]_{\checkmark}$ $Q)$
*else* $(b \rightarrow (P$ (*inv-into A c b*) $[\![S]\!]_{\checkmark}\ Q)) \sqcap$ *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* :
$\langle a \notin S \Longrightarrow d\ `\ B \cap S = \{\} \Longrightarrow$
$a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b =$
$($ *if* $B = \{\}$ *then* $a \rightarrow (P\ [\![S]\!]_{\checkmark}\ STOP)$
*else* $\sqcap b{\in}d\ `\ B.\ (a \rightarrow (P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q$ (*inv-into B d b*))) $\Box$
$(b \rightarrow (a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q$ (*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* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow b \notin S \Longrightarrow$
$c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
$($ *if* $A = \{\}$ *then* $b \rightarrow (STOP\ [\![S]\!]_{\checkmark}\ Q)$
*else* $\sqcap a{\in}c\ `\ A.\ (a \rightarrow (P$ (*inv-into A c a*) $[\![S]\!]_{\checkmark}\ b \rightarrow Q)) \Box$
$(b \rightarrow (a \rightarrow P$ (*inv-into A c a*) $[\![S]\!]_{\checkmark}\ Q)))\rangle$
**by** (*simp add*: *ndet-write-Sync*$_{ptick}$*-write-indep*[**where** $d = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *write0-Sync*$_{ptick}$*-ndet-write-left* :
$\langle a \notin S \Longrightarrow d\ `\ B \subseteq S \Longrightarrow a \rightarrow P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b = a \rightarrow (P\ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b)\rangle$
**by** (*simp add*: *write-Sync*$_{ptick}$*-ndet-write-left*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *ndet-write-Sync*$_{ptick}$*-write0-left* :
$\langle c\ `\ A \cap S = \{\} \Longrightarrow b \in S \Longrightarrow c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q = c!!a{\in}A \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q)\rangle$
**by** (*simp add*: *ndet-write-Sync*$_{ptick}$*-write-left*[**where** $d = id$, *unfolded write-is-write0*,

*simplified*])

**lemma** *write-Sync$_{ptick}$-ndet-write0-right* :
‹$a \in S \Longrightarrow d$ ‘ $B \cap S = \{\} \Longrightarrow a \rightarrow P \ [\![S]\!]_{\checkmark}\ d!!b{\in}B \rightarrow Q\ b = d!!b{\in}B \rightarrow (a \rightarrow P \ [\![S]\!]_{\checkmark}\ Q\ b)$›
**by** (*simp add*: *write-Sync$_{ptick}$-ndet-write-right*[**where** $c = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *ndet-write-Sync$_{ptick}$-write0-right* :
‹$c$ ‘ $A \subseteq S \Longrightarrow b \notin S \Longrightarrow c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ b \rightarrow Q = b \rightarrow (c!!a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q)$›
**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* :
‹$a \rightarrow P \ [\![S]\!]_{\checkmark}\ b \rightarrow Q =$
(*if* $b \in S$ *then STOP else* $b \rightarrow (a \rightarrow P \ [\![S]\!]_{\checkmark}\ Q)) \ \Box$
(*if* $a \in S$ *then STOP else* $a \rightarrow (P \ [\![S]\!]_{\checkmark}\ b \rightarrow Q)) \ \Box$
(*if* $a = b \wedge b \in S$ *then* $a \rightarrow (P \ [\![S]\!]_{\checkmark}\ Q)$ *else STOP*)›
**by** (*simp add*: *write-Sync$_{ptick}$-write*[**where** $c = id$ **and** $d = id$, *unfolded write-is-write0*, *simplified*])

**lemma** *write0-Sync$_{ptick}$-write0-bis* :
‹$(a \rightarrow P) \ [\![S]\!]_{\checkmark}\ (b \rightarrow Q) =$
( *if* $a \in S$
*then* *if* $b \in S$
*then* *if* $a = b$
*then* $a \rightarrow (P \ [\![S]\!]_{\checkmark}\ Q)$
*else STOP*
*else* $(b \rightarrow ((a \rightarrow P) \ [\![S]\!]_{\checkmark}\ Q))$
*else* *if* $b \in S$
*then* $a \rightarrow (P \ [\![S]\!]_{\checkmark}\ (b \rightarrow Q))$
*else* $(a \rightarrow (P \ [\![S]\!]_{\checkmark}\ (b \rightarrow Q))) \ \Box \ (b \rightarrow ((a \rightarrow P) \ [\![S]\!]_{\checkmark}\ Q))$)›
**by** (*cases* ‹$a \in S$›; *cases* ‹$b \in S$›) (*auto simp add*: *write0-Sync$_{ptick}$-write0 Det-commute*)

**lemma** *write0-Inter$_{ptick}$-write0* :
‹$a \rightarrow P \ |||_{\checkmark}\ b \rightarrow Q = (a \rightarrow (P \ |||_{\checkmark}\ b \rightarrow Q)) \ \Box \ (b \rightarrow (a \rightarrow P \ |||_{\checkmark}\ Q))$›
**by** (*simp add*: *write0-Sync$_{ptick}$-write0 Det-commute*)

**lemma** *write0-Par$_{ptick}$-write0* :
‹$a \rightarrow P \ ||_{\checkmark}\ b \rightarrow Q = (\textit{if } a = b \textit{ then } a \rightarrow (P \ ||_{\checkmark}\ Q) \textit{ else STOP})$›
**by** (*simp add*: *write0-Sync$_{ptick}$-write0*)

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

**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* :
$\langle a \notin S \Longrightarrow b \notin S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = (a \rightarrow (P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q)) \; \Box \; (b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q))\rangle$
**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* :
$\langle a \notin S \Longrightarrow b \in S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = a \rightarrow (P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q)\rangle$
**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* :
$\langle a \in S \Longrightarrow b \notin S \Longrightarrow a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q)\rangle$
**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* :
$\langle a \rightarrow P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q =$
$(if \; d \; b \in S \; then \; STOP \; else \; d!b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q)) \; \Box$
$(if \; a \in S \; then \; STOP \; else \; a \rightarrow (P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q)) \; \Box$
$(if \; a = d \; b \wedge d \; b \in S \; then \; a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q) \; else \; STOP)\rangle$
**by** (*simp add*: *write0-Sync$_{ptick}$-write0 write-is-write0*)

**lemma** *write-Sync$_{ptick}$-write0* :
$\langle c!a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q =$
$(if \; b \in S \; then \; STOP \; else \; b \rightarrow (c!a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q)) \; \Box$
$(if \; c \; a \in S \; then \; STOP \; else \; c!a \rightarrow (P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q)) \; \Box$
$(if \; c \; a = b \wedge b \in S \; then \; c!a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q) \; else \; STOP)\rangle$
**by** (*simp add*: *write0-Sync$_{ptick}$-write0 write-is-write0*)

**lemma** *write0-Sync$_{ptick}$-write-subset* :
$\langle a \in S \Longrightarrow d \; b \in S \Longrightarrow$
$a \rightarrow P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q = (if \; a = d \; b \; then \; a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q) \; else \; STOP)\rangle$
**by** (*simp add*: *write0-Sync$_{ptick}$-write*)

**lemma** *write-Sync$_{ptick}$-write0-subset* :
$\langle c \; a \in S \Longrightarrow b \in S \Longrightarrow$
$c!a \rightarrow P \; [\![S]\!]_{\checkmark} \; b \rightarrow Q = (if \; c \; a = b \; then \; c!a \rightarrow (P \; [\![S]\!]_{\checkmark} \; Q) \; else \; STOP)\rangle$
**by** (*simp add*: *write-Sync$_{ptick}$-write0*)

**lemma** *write0-Sync$_{ptick}$-write-indep* :
$\langle a \notin S \Longrightarrow d \; b \notin S \Longrightarrow$
$a \rightarrow P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q = (a \rightarrow (P \; [\![S]\!]_{\checkmark} \; d!b \rightarrow Q)) \; \Box \; (d!b \rightarrow (a \rightarrow P \; [\![S]\!]_{\checkmark} \; Q))\rangle$
**by** (*simp add*: *Det-commute write0-Sync$_{ptick}$-write*)

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

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

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

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

**lemma** *write-Sync$_{ptick}$-write0-right* :
‹$c\ a \in S \Longrightarrow b \notin S \Longrightarrow c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ b \rightarrow Q = b \rightarrow (c!a \rightarrow P\ [\![S]\!]_{\checkmark}\ Q)$›
**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* ≡ *Mprefix* (*c* ‘ *A*) (*P* ∘ *inv-into A c*) and *Mprefix A P* $[\![S]\!]_{\checkmark}$ *SKIP r* = □*a*∈(*A* − *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*).

**lemma** *read-Sync$_{ptick}$-SKIP* :
‹$c?a \in A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ SKIP\ r = c?a \in (A - c -` S) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ SKIP\ r)$› **if** ‹*inj-on c A*›
**proof** −
**have** ‹*c* ‘ (*A* − *c* −‘ *S*) = *c* ‘ *A* − *S*› **by** *blast*
**show** ‹$c?a \in A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ SKIP\ r = c?a \in (A - c -` S) \rightarrow (P\ a\ [\![S]\!]_{\checkmark}\ SKIP\ r)$›
**by** (*auto simp add*: *read-def Mprefix-Sync$_{ptick}$-SKIP* ‹*?this*› *inj-on-diff* ‹*inj-on c A*›
*intro*: *mono-Mprefix-eq*)
**qed**

**lemma** *SKIP-Sync$_{ptick}$-read* :
‹$SKIP\ r\ [\![S]\!]_{\checkmark}\ d?b \in B \rightarrow Q\ b = d?b \in (B - d -` S) \rightarrow (SKIP\ r\ [\![S]\!]_{\checkmark}\ Q\ b)$› **if** ‹*inj-on d B*›
**proof** −
**have** ‹*d* ‘ (*B* − *d* −‘ *S*) = *d* ‘ *B* − *S*› **by** *blast*
**show** ‹$SKIP\ r\ [\![S]\!]_{\checkmark}\ d?b \in B \rightarrow Q\ b = d?b \in (B - d -` S) \rightarrow (SKIP\ r\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**by** (*auto simp add*: *read-def SKIP-Sync$_{ptick}$-Mprefix* ‹*?this*› *inj-on-diff* ‹*inj-on d B*›
*intro*: *mono-Mprefix-eq*)

**qed**

**corollary** *write-Sync$_{ptick}$-SKIP* :
‹*c*!*a* → *P* ⟦*S*⟧✓ *SKIP s* = (*if c a* ∈ *S then STOP else c*!*a* → (*P* ⟦*S*⟧✓ *SKIP s*))›
**and** *SKIP-Sync$_{ptick}$-write* :
‹*SKIP r* ⟦*S*⟧✓ *d*!*b* → *Q* = (*if d b* ∈ *S then STOP else d*!*b* → (*SKIP r* ⟦*S*⟧✓ *Q*))›
**by** (*simp-all add*: *write-def Mprefix-Sync$_{ptick}$-SKIP SKIP-Sync$_{ptick}$-Mprefix Diff-triv*)

**corollary** *write0-Sync$_{ptick}$-SKIP* :
‹*a* → *P* ⟦*S*⟧✓ *SKIP s* = (*if a* ∈ *S then STOP else a* → (*P* ⟦*S*⟧✓ *SKIP s*))›
**and** *SKIP-Sync$_{ptick}$-write0* :
‹*SKIP r* ⟦*S*⟧✓ *b* → *Q* = (*if b* ∈ *S then STOP else b* → (*SKIP r* ⟦*S*⟧✓ *Q*))›
**by** (*simp-all add*: *write0-def Mprefix-Sync$_{ptick}$-SKIP SKIP-Sync$_{ptick}$-Mprefix Diff-triv*)

**lemma** *ndet-write-Sync$_{ptick}$-SKIP* :
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *SKIP r* =
( *if c* ‘ *A* ∩ *S* = {} *then c*!!*a*∈*A* → (*P a* ⟦*S*⟧✓ *SKIP r*)
*else* (*c*!!*a*∈(*A* − *c* −‘ *S*) → (*P a* ⟦*S*⟧✓ *SKIP r*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if* - *then ?rhs1 else ?rhs2* ⊓ *STOP*)›) **if** ‹*inj-on c A*›
**proof** (*split if-split*, *intro conjI impI*)
**assume** ‹*c* ‘ *A* ∩ *S* = {}›
**hence** ‹*A* − *c* −‘ *S* = *A*› **by** *blast*
**from** ‹*c* ‘ *A* ∩ *S* = {}› **show** ‹*?lhs* = *?rhs1*›
**by** (*auto simp add*: ‹*?this*› *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** ‹*?lhs* = *?rhs2* ⊓ *STOP*› **if** ‹*c* ‘ *A* ∩ *S* ≠ {}›
**proof** (*cases* ‹*c* ‘ *A* − *S* = {}›)
**assume** ‹*c* ‘ *A* − *S* = {}›
**hence** ‹*A* − *c* −‘ *S* = {}› **by** *blast*
**from** ‹*c* ‘ *A* − *S* = {}› **show** ‹*?lhs* = *?rhs2* ⊓ *STOP*›
**by** (*auto simp add*: *ndet-write-is-GlobalNdet-write0 GlobalNdet-is-STOP-iff* ‹*?this*› *Sync$_{ptick}$-distrib-GlobalNdet-right write0-Sync$_{ptick}$-SKIP*)
**next**
**have** ‹*c* ‘ (*A* − *c* −‘ *S*) = *c* ‘ *A* − *S*› **by** *blast*
**show** ‹*?lhs* = *?rhs2* ⊓ *STOP*› **if** ‹*c* ‘ *A* − *S* ≠ {}›
**by** (*subst Ndet-commute*, *unfold ndet-write-is-GlobalNdet-write0 Sync$_{ptick}$-distrib-GlobalNdet-right*)
(*auto simp add*: *GlobalNdet-is-STOP-iff write0-Sync$_{ptick}$-SKIP* ‹*?this*› ‹*inj-on c A*› *inj-on-diff*
*simp flip*: *GlobalNdet-factorization-union*
[*OF* ‹*c* ‘ *A* ∩ *S* ≠ {}› ‹*c* ‘ *A* − *S* ≠ {}›, *unfolded Int-Diff-Un*]
*intro*!: *arg-cong2*[**where** *f* = ‹(⊓)›] *mono-GlobalNdet-eq*)
**qed**
**qed**

**corollary** (**in** $Sync_{ptick}$-*locale*) *SKIP-Sync*$_{ptick}$*-ndet-write* :
‹*inj-on d B* $\Longrightarrow$ *SKIP r* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* → *Q b* =
( *if d* ‘ *B* ∩ *S* = {} *then d*!!*b*∈*B* → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)
*else* (*d*!!*b*∈(*B* − *d* −‘ *S*) → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)) ⊓ *STOP*)›
**by** (*subst* (*1 2 3*) *Sync*$_{ptick}$*-locale-sym.Sync*$_{ptick}$*-sym*)
(*simp add*: *Sync*$_{ptick}$*-locale-sym.ndet-write-Sync*$_{ptick}$*-SKIP*)

**corollary** (**in** $Sync_{ptick}$-*locale*) *Mndetprefix-Sync*$_{ptick}$*-SKIP* :
‹⊓*a* ∈ *A* → *P a* $[\![S]\!]_{\checkmark}$ *SKIP r* =
(*if A* ∩ *S* = {} *then* ⊓*a* ∈ *A* → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*)
*else* (⊓*a* ∈ (*A* − *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *SKIP r*)) ⊓ *STOP*)›
**using** *ndet-write-Sync*$_{ptick}$*-SKIP*[*of id A P S r*]
**by** (*simp add*: *ndet-write-id-is-Mndetprefix*)

**corollary** (**in** $Sync_{ptick}$-*locale*) *Sync*$_{ptick}$*-SKIP-Mndetprefix* :
‹*SKIP r* $[\![S]\!]_{\checkmark}$ ⊓*b* ∈ *B* → *Q b* =
( *if B* ∩ *S* = {} *then* ⊓*b* ∈ *B* → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)
*else* (⊓*b* ∈ (*B* − *S*) → (*SKIP r* $[\![S]\!]_{\checkmark}$ *Q b*)) ⊓ *STOP*)›
**by** (*subst* (*1 2 3*) *Sync*$_{ptick}$*-locale-sym.Sync*$_{ptick}$*-sym*)
(*simp add*: *Sync*$_{ptick}$*-locale-sym.Mndetprefix-Sync*$_{ptick}$*-SKIP*)

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

**lemma** *read-Sync*$_{ptick}$*-STOP* :
‹*c*?*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *STOP* = *c*?*a*∈(*A* − *c* −‘ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *STOP*)› **if** ‹*inj-on c A*›
**proof** −
**have** ‹*c* ‘ (*A* − *c* −‘ *S*) = *c* ‘ *A* − *S*› **by** *blast*
**show** ‹*c*?*a*∈*A* → *P a* $[\![S]\!]_{\checkmark}$ *STOP* = *c*?*a*∈(*A* − *c* −‘ *S*) → (*P a* $[\![S]\!]_{\checkmark}$ *STOP*)›
**by** (*auto simp add*: ‹*?this*› *read-def Mprefix-Sync*$_{ptick}$*-STOP inj-on-diff* ‹*inj-on c A*›
*intro*: *mono-Mprefix-eq*)
**qed**

**lemma** *STOP-Sync*$_{ptick}$*-read* :
‹*STOP* $[\![S]\!]_{\checkmark}$ *d*?*b*∈*B* → *Q b* = *d*?*b*∈(*B* − *d* −‘ *S*) → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)› **if** ‹*inj-on d B*›
**proof** −
**have** ‹*d* ‘ (*B* − *d* −‘ *S*) = *d* ‘ *B* − *S*› **by** *blast*
**show** ‹*STOP* $[\![S]\!]_{\checkmark}$ *d*?*b*∈*B* → *Q b* = *d*?*b*∈(*B* − *d* −‘ *S*) → (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)›
**by** (*auto simp add*: ‹*?this*› *read-def STOP-Sync*$_{ptick}$*-Mprefix inj-on-diff* ‹*inj-on d B*›
*intro*: *mono-Mprefix-eq*)
**qed**

**corollary** *write-Sync$_{ptick}$-STOP* :
‹*c*!*a* → *P* ⟦*S*⟧✓ *STOP* = (*if c a* ∈ *S then STOP else c*!*a* → (*P* ⟦*S*⟧✓ *STOP*))›
**and** *STOP-Sync$_{ptick}$-write* :
‹*STOP* ⟦*S*⟧✓ *d*!*b* → *Q* = (*if d b* ∈ *S then STOP else d*!*b* → (*STOP* ⟦*S*⟧✓ *Q*))›
**by** (*simp-all add*: *write-def Mprefix-Sync$_{ptick}$-STOP STOP-Sync$_{ptick}$-Mprefix Diff-triv*)

**corollary** *write0-Sync$_{ptick}$-STOP* :
‹*a* → *P* ⟦*S*⟧✓ *STOP* = (*if a* ∈ *S then STOP else a* → (*P* ⟦*S*⟧✓ *STOP*))›
**and** *STOP-Sync$_{ptick}$-write0* :
‹*STOP* ⟦*S*⟧✓ *b* → *Q* = (*if b* ∈ *S then STOP else b* → (*STOP* ⟦*S*⟧✓ *Q*))›
**by** (*simp-all add*: *write0-def Mprefix-Sync$_{ptick}$-STOP STOP-Sync$_{ptick}$-Mprefix Diff-triv*)

**lemma** *ndet-write-Sync$_{ptick}$-STOP* :
‹*c*!!*a*∈*A* → *P a* ⟦*S*⟧✓ *STOP* =
( *if c ‘ A* ∩ *S* = {} *then c*!!*a*∈*A* → (*P a* ⟦*S*⟧✓ *STOP*)
*else* (*c*!!*a*∈(*A* − *c* −‘ *S*) → (*P a* ⟦*S*⟧✓ *STOP*)) ⊓ *STOP*)›
(**is** ‹*?lhs* = (*if* - *then ?rhs1 else ?rhs2* ⊓ *STOP*)›) **if** ‹*inj-on c A*›
**proof** (*split if-split*, *intro conjI impI*)
**assume** ‹*c ‘ A* ∩ *S* = {}›
**hence** ‹*A* − *c* −‘ *S* = *A*› **by** *blast*
**from** ‹*c ‘ A* ∩ *S* = {}› **show** ‹*?lhs* = *?rhs1*›
**by** (*auto simp add*: ‹*?this*› *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** ‹*?lhs* = *?rhs2* ⊓ *STOP*› **if** ‹*c ‘ A* ∩ *S* ≠ {}›
**proof** (*cases* ‹*c ‘ A* − *S* = {}›)
**assume** ‹*c ‘ A* − *S* = {}›
**hence** ‹*A* − *c* −‘ *S* = {}› **by** *blast*
**from** ‹*c ‘ A* − *S* = {}› **show** ‹*?lhs* = *?rhs2* ⊓ *STOP*›
**by** (*auto simp add*: *ndet-write-is-GlobalNdet-write0 GlobalNdet-is-STOP-iff* ‹*?this*› *Sync$_{ptick}$-distrib-GlobalNdet-right write0-Sync$_{ptick}$-STOP*)
**next**
**have** ‹*c ‘* (*A* − *c* −‘ *S*) = *c ‘ A* − *S*› **by** *blast*
**show** ‹*?lhs* = *?rhs2* ⊓ *STOP*› **if** ‹*c ‘ A* − *S* ≠ {}›
**by** (*subst Ndet-commute*, *unfold ndet-write-is-GlobalNdet-write0 Sync$_{ptick}$-distrib-GlobalNdet-right*)
(*auto simp add*: *GlobalNdet-is-STOP-iff write0-Sync$_{ptick}$-STOP* ‹*?this*› ‹*inj-on c A*› *inj-on-diff simp flip*: *GlobalNdet-factorization-union* [*OF* ‹*c ‘ A* ∩ *S* ≠ {}› ‹*c ‘ A* − *S* ≠ {}›, *unfolded Int-Diff-Un*] *intro*!: *arg-cong2*[**where** *f* = ‹(⊓)›] *mono-GlobalNdet-eq*)
**qed**
**qed**

**corollary** (**in** $Sync_{ptick}$-*locale*) *STOP-*$Sync_{ptick}$*-ndet-write* :
‹*inj-on d B* $\Longrightarrow$ *STOP* $[\![S]\!]_{\checkmark}$ *d*!!*b*∈*B* $\rightarrow$ *Q b* =
( *if d* ‘ *B* ∩ *S* = {} *then d*!!*b*∈*B* $\rightarrow$ (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)
*else* (*d*!!*b*∈(*B* − *d* −‘ *S*) $\rightarrow$ (*STOP* $[\![S]\!]_{\checkmark}$ *Q b*)) ⊓ *STOP*)›
**by** (*subst* (*1 2 3*) $Sync_{ptick}$-*locale-sym.*$Sync_{ptick}$-*sym*)
(*simp add*: $Sync_{ptick}$-*locale-sym.ndet-write-*$Sync_{ptick}$-*STOP*)

**end**

# Chapter 9

# Operational Semantics Laws

## 9.1 Behaviour of *initials*

### 9.1.1 *TickSwap*

**lemma** *initials-TickSwap* :
‹(*TickSwap P*)$^0$ = ( *if P* = ⊥ *then UNIV*
*else* {*ev a* |*a*. *ev a* ∈ $P^0$} ∪ {✓((*s*, *r*)) |*r s*. ✓((*r*, *s*)) ∈ $P^0$})›
**by** (*auto simp add*: *TickSwap-is-Renaming initials-Renaming image-iff*
$map\text{-}event_{ptick}$*-eq-tick-iff* $map\text{-}event_{ptick}$*-eq-ev-iff*
$tick\text{-}eq\text{-}map\text{-}event_{ptick}$*-iff* $ev\text{-}eq\text{-}map\text{-}event_{ptick}$*-iff*)
(*metis tick-swap.elims*)

### 9.1.2 Sequential Composition

**lemma** $initials\text{-}Seq_{ptick}$ :
‹(*P* ;✓ *Q*)$^0$ = ( *if P* = ⊥ *then UNIV*
*else* {*ev a* |*a*. *ev a* ∈ $P^0$} ∪ (⋃*r*∈{*r*. ✓(*r*) ∈ $P^0$}. (*Q r*)$^0$))›
(**is** ‹- = (*if* - *then* - *else ?rhs*)›)
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹*P* = ⊥ ⟹ (*P* ;✓ *Q*)$^0$ = *UNIV*› **by** *simp*
**next**
**show** ‹(*P* ;✓ *Q*)$^0$ = {*ev a* |*a*. *ev a* ∈ $P^0$} ∪ (⋃*r*∈{*r*. ✓(*r*) ∈ $P^0$}. (*Q r*)$^0$)› **if** ‹*P* ≠ ⊥›
**proof** (*intro subset-antisym subsetI*)
**fix** *e* **assume** ‹*e* ∈ (*P* ;✓ *Q*)$^0$›
**from** $event_{ptick}$*.exhaust* **consider** *a* **where** ‹*e* = *ev a*› | *r* **where** ‹*e* = ✓(*r*)› **by** *blast*
**thus** ‹*e* ∈ *?rhs*›
**proof** *cases*
**from** ‹*e* ∈ (*P* ;✓ *Q*)$^0$› ‹*P* ≠ ⊥› **show** ‹*e* = *ev a* ⟹ *e* ∈ *?rhs*› **for** *a*
**by** (*auto simp add*: *image-iff initials-def* $Seq_{ptick}$*-projs Cons-eq-append-conv BOT-iff-Nil-D intro*: *D-T dest*: *initials-memD*)
(*use initials-memI* **in** ‹*blast dest*: *initials-memD*›)
**next**
**from** ‹*e* ∈ (*P* ;✓ *Q*)$^0$› ‹*P* ≠ ⊥› **show** ‹*e* = ✓(*r*) ⟹ *e* ∈ *?rhs*› **for** *r*

**by** (*auto simp add*: *image-iff initials-def* $Seq_{ptick}$*-projs BOT-iff-Nil-D Cons-eq-append-conv*)
**qed**
**next**
**show** ‹$e \in$ *?rhs* $\Longrightarrow e \in (P \;;_{\checkmark}\; Q)^0$› **for** $e$
**by** (*simp add*: *initials-def* $Seq_{ptick}$*-projs*, *elim disjE exE conjE*)
(*fastforce*, *metis list.simps*(8) *self-append-conv2 tickFree-Nil*)
**qed**
**qed**

### 9.1.3 Synchronization Product

**lemma** (**in** $Sync_{ptick}$*-locale*) *initials-*$Sync_{ptick}$ :
‹$(P \;[\![S]\!]_{\checkmark}\; Q)^0 =$
(*if* $P = \bot \vee Q = \bot$ *then UNIV else*
$\{ev\ a \mid a.\ a \in S \wedge ev\ a \in P^0 \wedge ev\ a \in Q^0 \vee a \notin S \wedge (ev\ a \in P^0 \vee ev\ a \in Q^0)\}$
$\cup$
$\{\checkmark(r\text{-}s) \mid r\text{-}s\ r\ s.\ tick\text{-}join\ r\ s = Some\ r\text{-}s \wedge \checkmark(r) \in P^0 \wedge \checkmark(s) \in Q^0\}$)›
(**is** ‹$(P \;[\![S]\!]_{\checkmark}\; Q)^0 =$ (*if* $P = \bot \vee Q = \bot$ *then UNIV else ?rhs-ev* $\cup$ *?rhs-tick*)›)
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$P = \bot \vee Q = \bot \Longrightarrow (P \;[\![S]\!]_{\checkmark}\; Q)^0 = UNIV$›
**by** (*metis* $Sync_{ptick}$*-is-BOT-iff initials-BOT*)
**next**
**show** ‹$(P \;[\![S]\!]_{\checkmark}\; Q)^0 =$ *?rhs-ev* $\cup$ *?rhs-tick*› **if** *non-BOT* : ‹$\neg\ (P = \bot \vee Q = \bot)$›
**proof** (*intro subset-antisym subsetI*)
**show** ‹$e \in$ *?rhs-ev* $\cup$ *?rhs-tick* $\Longrightarrow e \in (P \;[\![S]\!]_{\checkmark}\; Q)^0$› **for** $e$
**proof** (*elim UnE*)
**assume** ‹$e \in$ *?rhs-ev*›
**then consider** $a$ **where** ‹$e = ev\ a$› ‹$a \in S$› ‹$ev\ a \in P^0$› ‹$ev\ a \in Q^0$›
| $a$ **where** ‹$e = ev\ a$› ‹$a \notin S$› ‹$ev\ a \in P^0 \vee ev\ a \in Q^0$› **by** *blast*
**thus** ‹$e \in (P \;[\![S]\!]_{\checkmark}\; Q)^0$›
**proof** *cases*
**fix** $a$ **assume** ‹$e = ev\ a$› ‹$a \in S$› ‹$ev\ a \in P^0$› ‹$ev\ a \in Q^0$›
**have** $*$ : ‹$[ev\ a]$ *setinterleaves*$_{\checkmark}$ *tick-join* $(([ev\ a], [ev\ a]), S)$›
**by** (*simp add*: ‹$a \in S$›)
**from** ‹$ev\ a \in P^0$› ‹$ev\ a \in Q^0$› **show** ‹$e \in (P \;[\![S]\!]_{\checkmark}\; Q)^0$›
**by** (*simp add*: ‹$e = ev\ a$› *initials-def T-*$Sync_{ptick}$) (*use* $*$ **in** *blast*)
**next**
**fix** $a$ **assume** ‹$e = ev\ a$› ‹$a \notin S$› ‹$ev\ a \in P^0 \vee ev\ a \in Q^0$›
**from** ‹$ev\ a \in P^0 \vee ev\ a \in Q^0$› **show** ‹$e \in (P \;[\![S]\!]_{\checkmark}\; Q)^0$›
**proof** (*elim disjE*)
**assume** ‹$ev\ a \in P^0$›
**have** $*$ : ‹$[ev\ a]$ *setinterleaves*$_{\checkmark}$ *tick-join* $(([ev\ a], []), S)$›
**by** (*simp add*: ‹$a \notin S$›)
**from** ‹$ev\ a \in P^0$› **show** ‹$e \in (P \;[\![S]\!]_{\checkmark}\; Q)^0$›
**by** (*simp add*: ‹$e = ev\ a$› *initials-def T-*$Sync_{ptick}$) (*meson* $*$ *is-processT1-TR*)
**next**
**assume** ‹$ev\ a \in Q^0$›
**have** $*$ : ‹$[ev\ a]$ *setinterleaves*$_{\checkmark}$ *tick-join* $(([], [ev\ a]), S)$›

```
        by (simp add: ‹a ∉ S›)
      from ‹ev a ∈ Q⁰› show ‹e ∈ (P ⟦S⟧✓ Q)⁰›
      by (simp add: ‹e = ev a› initials-def T-Sync_ptick) (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-Sync_ptick) (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-Sync_ptick 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 setinterleaving_ptick-simps
        split: if-split-asm option.split-asm event_ptick.splits
        dest!: Nil-setinterleaves_ptick)
   next
    case div
   have ‹t ≠ []› by (metis BOT-iff-Nil-D Nil-setinterleaves_ptick 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 setinterleaving_ptick-simps BOT-iff-Nil-D
        split: if-split-asm event_ptick.splits option.split-asm
        dest!: Nil-setinterleaves_ptick intro: D-T)
   qed
  qed
qed
```

## 9.2 Laws of After

### 9.2.1 Sequential Composition

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

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

**lemma** *not-skippable-or-not-initialR-After-Seq*$_{ptick}$:
  ‹(*P* ;✓ *Q*) $after_\beta$ *a* = (*if ev a* ∈ $P^0$ *then P*  $after_\alpha$ *a* ;✓ *Q else* $\Psi_\beta$ (*P* ;✓ *Q*) *a*)›
  **if** ‹*range tick* ∩ $P^0$ = {} ∨ (∀ *r*. ✓(*r*) ∈ $P^0$ ⟶ *ev a* ∉ (*Q r*)$^0$)›
**proof** (*cases* ‹*P* = ⊥›)
  **show** ‹*P* = ⊥ ⟹ (*P* ;✓ *Q*) $after_\beta$ *a* =
      (*if ev a* ∈ $P^0$ *then P* $after_\alpha$ *a* ;✓ *Q else* $\Psi_\beta$ (*P* ;✓ *Q*) *a*)›
    **by** (*simp add*: *After.After-BOT*)
**next**
  **note** *denot-projs* = *After.After-projs Seq*$_{ptick}$*-projs*
  **assume** *non-BOT*: ‹*P* ≠ ⊥›
  **with** *that* **have** $ : ‹*ev a* ∈ (*P* ;✓ *Q*)$^0$ ⟷ *ev a* ∈ $P^0$›
    **by** (*auto simp add*: *initials-Seq*$_{ptick}$)
  **show** ‹(*P* ;✓ *Q*) $after_\beta$ *a* = (*if ev a* ∈ $P^0$ *then P* $after_\alpha$ *a* ;✓ *Q else* $\Psi_\beta$ (*P* ;✓ *Q*) *a*)›
  **proof** (*split if-split*, *intro conjI impI*)
    **from** $ **show** ‹*ev a* ∉ $P^0$ ⟹ (*P* ;✓ *Q*) $after_\beta$ *a* = $\Psi_\beta$ (*P* ;✓ *Q*) *a*›
      **by** (*simp add*: $After_\beta$*.not-initial-After*)
  **next**
    **assume** *initial-P* : ‹*ev a* ∈ $P^0$›
    **show** ‹(*P* ;✓ *Q*) $after_\beta$ *a* = *P* $after_\alpha$ *a* ;✓ *Q*›
    **proof** (*rule Process-eq-optimizedI*)
      **fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ ((*P* ;✓ *Q*) $after_\beta$ *a*)›
      **then consider** (*D-P*) *t'* *u* **where** ‹*ev a* # *t* = *map* (*ev* ∘ *of-ev*) *t'* @ *u*› ‹*t'* ∈ $\mathcal{D}$ *P*› ‹*tF t'*› ‹*ftF u*›
        | (*D-Q*) *t'* *r* *u* **where** ‹*ev a* # *t* = *map* (*ev* ∘ *of-ev*) *t'* @ *u*› ‹*t'* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*› ‹*tF t'*› ‹*u* ∈ $\mathcal{D}$ (*Q r*)›
        **by** (*auto simp add*: *denot-projs* $ *initial-P*)
      **thus** ‹*t* ∈ $\mathcal{D}$ (*P* $after_\alpha$ *a* ;✓ *Q*)›
      **proof** *cases*
        **case** *D-P* **with** *non-BOT initial-P* **show** ‹*t* ∈ $\mathcal{D}$ (*P* $after_\alpha$ *a* ;✓ *Q*)›
          **by** (*cases t'*) (*auto simp add*: *BOT-iff-Nil-D denot-projs*)
      **next**
        **case** *D-Q* **with** *initial-P* **show** ‹*t* ∈ $\mathcal{D}$ (*P* $after_\alpha$ *a* ;✓ *Q*)›
          **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** ‹$t \in \mathcal{D}\ (P\ after_\alpha\ a\ ;\!\!\surd\ Q)$›
**then consider** ($D$-$P$) $t'\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$ev\ a\ \#\ t' \in \mathcal{D}\ P$› ‹$tF\ t'$› ‹$ftF\ u$›
| ($D$-$Q$) $t'\ r\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$ev\ a\ \#\ t'\ @\ [\surd(r)] \in \mathcal{T}\ P$› ‹$tF\ t'$› ‹$u \in \mathcal{D}\ (Q\ r)$›
**by** (*auto simp add*: *denot-projs initial-P*)
**thus** ‹$t \in \mathcal{D}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$›
**proof** *cases*
**case** $D$-$P$ **thus** ‹$t \in \mathcal{D}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$›
**by** (*simp add*: *denot-projs* $ *initial-P Cons-eq-append-conv Cons-eq-map-conv*)
(*metis* $event_{ptick}.disc(1)$ $event_{ptick}.sel(1)$ *tickFree-Cons-iff*)
**next**
**case** $D$-$Q$ **thus** ‹$t \in \mathcal{D}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$›
**by** (*simp add*: *denot-projs* $ *initial-P Cons-eq-append-conv Cons-eq-map-conv*)
(*metis append-Cons* $event_{ptick}.sel(1)$ *is-ev-def tickFree-Cons-iff*)
**qed**
**next**
**fix** $t\ X$ **assume** ‹$(t,\ X) \in \mathcal{F}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$› ‹$t \notin \mathcal{D}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$›
**then consider** ($F$-$P$) $t'$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t'$›
‹$(t',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ P$› ‹$tF\ t'$›
| ($F$-$Q$) $t'\ r\ u$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t'\ @\ [\surd(r)] \in \mathcal{T}\ P$›
‹$tF\ t'$› ‹$(u,\ X) \in \mathcal{F}\ (Q\ r)$›
**by** (*auto simp add*: *denot-projs* $ *initial-P*)
(*metis* (*mono-tags*, *lifting*) *comp-apply list.simps*(9) *tickFree-Cons-iff*)
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ after_\alpha\ a\ ;\!\!\surd\ Q)$›
**proof** *cases*
**case** $F$-$P$ **thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ after_\alpha\ a\ ;\!\!\surd\ Q)$›
**by** (*auto simp add*: *denot-projs initial-P*)
**next**
**case** $F$-$Q$ **thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ after_\alpha\ a\ ;\!\!\surd\ Q)$›
**by** (*cases* $t'$, *auto simp add*:*denot-projs initial-P intro*: *initials-memI*)
(*metis F-T Int-iff empty-iff initials-memI rangeI that*)
**qed**
**next**
**fix** $t\ X$ **assume** ‹$(t,\ X) \in \mathcal{F}\ (P\ after_\alpha\ a\ ;\!\!\surd\ Q)$› ‹$t \notin \mathcal{D}\ (P\ after_\alpha\ a\ ;\!\!\surd\ Q)$›
**then consider** ($F$-$P$) $t'$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'$›
‹$(ev\ a\ \#\ t',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ P$› ‹$tF\ t'$›
| ($F$-$Q$) $t'\ r\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$ev\ a\ \#\ t'\ @\ [\surd(r)] \in \mathcal{T}\ P$›
‹$tF\ t'$› ‹$(u,\ X) \in \mathcal{F}\ (Q\ r)$›
**by** (*auto simp add*: *denot-projs initial-P*)
**thus** ‹$(t,\ X) \in \mathcal{F}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$›
**proof** *cases*
**case** $F$-$P$ **thus** ‹$(t,\ X) \in \mathcal{F}\ ((P\ ;\!\!\surd\ Q)\ after_\beta\ a)$›
**by** (*simp add*: *denot-projs* $ *initial-P Cons-eq-map-conv*)
(*metis* $event_{ptick}.disc(1)$ $event_{ptick}.sel(1)$ *tickFree-Cons-iff*)
**next**

**case** *F-Q* **thus** ‹$(t,\ X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
**by** (*simp add*: *denot-projs* $ *initial-P Cons-eq-append-conv Cons-eq-map-conv*)
(*metis append-Cons* $event_{ptick}.disc(1)$ $event_{ptick}.sel(1)$ *tickFree-Cons-iff*)
**qed**
**qed**
**qed**
**qed**

**lemma** $skippable\text{-}not\text{-}initialL\text{-}After\text{-}Seq_{ptick}$:
‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (\ \ if\ (\exists r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0)$
$then\ \sqcap r{\in}\{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.\ Q\ r\ after_\beta\ a$
$else\ \Psi_\beta\ (P\ ;_{\checkmark}\ Q)\ a)$›
(**is** ‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (if\ ?prem\ then\ ?rhs\ else\ -)$›) **if** ‹$ev\ a \notin P^0$›
**proof** −
**note** *denot-projs* = *After.After-projs* $Seq_{ptick}$*-projs GlobalNdet-projs*
**from** *initials-BOT* ‹$ev\ a \notin P^0$› **have** *non-BOT* : ‹$P \neq \bot$› **by** *blast*
**with** ‹$ev\ a \notin P^0$› **have** $ : ‹$ev\ a \in (P\ ;_{\checkmark}\ Q)^0 \longleftrightarrow ?prem$›
**by** (*auto simp add*: $initials\text{-}Seq_{ptick}$)
**show** ‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (if\ ?prem\ then\ ?rhs\ else\ \Psi_\beta\ (P\ ;_{\checkmark}\ Q)\ a)$›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$\neg\ ?prem \Longrightarrow (P\ ;_{\checkmark}\ Q)\ after_\beta\ a = \Psi_\beta\ (P\ ;_{\checkmark}\ Q)\ a$›
**by** (*rule* $After_\beta.not\text{-}initial\text{-}After$, *use* $ **in** *blast*)
**next**
**show** ‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = ?rhs$› **if** *?prem*
**proof** (*rule Process-eq-optimizedI*)
**fix** $t$ **assume** ‹$t \in \mathcal{D}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
**then consider** (*D-P*) $t'\ u$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t' \in \mathcal{D}\ P$› ‹$tF\ t'$› ‹$ftF\ u$›
| (*D-Q*) $t'\ r\ u$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t' @ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$tF\ t'$› ‹$u \in \mathcal{D}\ (Q\ r)$›
**by** (*auto simp add*: *denot-projs* ‹*?prem*› $ ‹$ev\ a \notin P^0$›)
**thus** ‹$t \in \mathcal{D}\ ?rhs$›
**proof** *cases*
**case** *D-P* **with** *non-BOT* ‹$ev\ a \notin P^0$› **show** ‹$t \in \mathcal{D}\ ?rhs$›
**by** (*simp add*: *denot-projs Cons-eq-append-conv Cons-eq-map-conv BOT-iff-Nil-D*)
(*metis D-T* $event_{ptick}.collapse(1)$ *initials-memI tickFree-Cons-iff*)
**next**
**case** *D-Q* **with** ‹$ev\ a \notin P^0$› **show** ‹$t \in \mathcal{D}\ ?rhs$›
**by** (*simp add*: *denot-projs Cons-eq-append-conv Cons-eq-map-conv*)
(*metis D-T append-Nil* $event_{ptick}.collapse(1)$ *initials-memI*
*is-processT3-TR-append tickFree-Cons-iff*)
**qed**
**next**
**from** ‹*?prem*› **show** ‹$t \in \mathcal{D}\ ?rhs \Longrightarrow t \in \mathcal{D}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$› **for** $t$
**by** (*simp add*: *denot-projs* $ *Cons-eq-append-conv*)
(*metis append-Nil initials-memD tickFree-Nil*)
**next**
**fix** $t\ X$ **assume** ‹$(t,\ X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$› ‹$t \notin \mathcal{D}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta$

$a)$›
    **then consider** $(F\text{-}P)$ $t'$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t'$›
      ‹$(t', ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ P$› ‹$tF\ t'$›
    | $(F\text{-}Q)$ $t'\ r\ u$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t' @ u$›
      ‹$t' @ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$tF\ t'$› ‹$(u, X) \in \mathcal{F}\ (Q\ r)$›
      **by** $(simp\ add{:}\ denot\text{-}projs\ \$\ ‹?prem›)\ metis$
    **thus** ‹$(t, X) \in \mathcal{F}\ ?rhs$›
    **proof** *cases*
      **case** *F-P* **with** ‹$ev\ a \notin P^0$› **show** ‹$(t, X) \in \mathcal{F}\ ?rhs$›
        **by** $(cases\ t', simp\text{-}all\ add{:}\ denot\text{-}projs\ ‹?prem›)$
          $(use\ F\text{-}T\ initials\text{-}memI\ \textbf{in}\ blast)$
    **next**
      **case** *F-Q* **with** ‹$ev\ a \notin P^0$› ‹*?prem*› **show** ‹$(t, X) \in \mathcal{F}\ ?rhs$›
        **by** $(cases\ t', auto\ simp\ add{:}\ denot\text{-}projs\ intro{:}\ initials\text{-}memI)$
          $(metis\ F\text{-}T\ initials\text{-}memI)$
    **qed**
  **next**
    **from** ‹*?prem*› **show** ‹$(t, X) \in \mathcal{F}\ ?rhs \Longrightarrow t \notin \mathcal{D}\ ?rhs \Longrightarrow$
        $(t, X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$› **for** $t\ X$
      **by** $(simp\ add{:}\ denot\text{-}projs\ \$\ Cons\text{-}eq\text{-}append\text{-}conv\ Cons\text{-}eq\text{-}map\text{-}conv\ split{:}$
*if-split-asm*$)$
        $(metis\ initials\text{-}memD\ self\text{-}append\text{-}conv2\ tickFree\text{-}Nil)$
  **qed**
 **qed**
**qed**

**lemma** *skippable-initialL-initialR-After-Seq*$_{ptick}$:
 ‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap (\sqcap r \in \{r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0\}.$
$Q\ r\ after_\beta\ a)$›
 (**is** ‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs$›)
 **if** *assms* : ‹$\exists r.\ \checkmark(r) \in P^0 \wedge ev\ a \in (Q\ r)^0$› ‹$ev\ a \in P^0$›
**proof** $(cases\ ‹P = \bot›)$
 **show** ‹$P = \bot \Longrightarrow (P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs$›
  **by** $(simp\ add{:}\ After.After\text{-}BOT)$
**next**
 **note** $denot\text{-}projs = After.After\text{-}projs\ Seq_{ptick}\text{-}projs\ GlobalNdet\text{-}projs\ Ndet\text{-}projs$
 **show** ‹$(P\ ;_{\checkmark}\ Q)\ after_\beta\ a = (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs$› **if** ‹$P \neq \bot$›
 **proof** $(rule\ Process\text{-}eq\text{-}optimizedI)$
  **fix** $t$ **assume** ‹$t \in \mathcal{D}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
  **then consider** $(D\text{-}P)$ $t'\ u$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t' @ u$› ‹$t' \in$
$\mathcal{D}\ P$› ‹$tF\ t'$› ‹$ftF\ u$›
    | $(D\text{-}Q)$ $t'\ r\ u$ **where** ‹$ev\ a\ \#\ t = map\ (ev \circ of\text{-}ev)\ t' @ u$› ‹$t' @ [\checkmark(r)] \in \mathcal{T}$
$P$› ‹$tF\ t'$› ‹$u \in \mathcal{D}\ (Q\ r)$›
    **by** $(auto\ simp\ add{:}\ denot\text{-}projs\ initials\text{-}Seq_{ptick}\ assms\ split{:}\ if\text{-}split\text{-}asm)$
  **thus** ‹$t \in \mathcal{D}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›
  **proof** *cases*
    **case** *D-P* **with** ‹$P \neq \bot$› **show** ‹$t \in \mathcal{D}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›

**by** (*auto simp add*: *denot-projs assms Cons-eq-append-conv BOT-iff-Nil-D*)
**next**
**case** *D-Q* **thus** ‹$t \in \mathcal{D}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›
**by** (*auto simp add*: *denot-projs assms Cons-eq-append-conv*)
(*meson D-T initials-memI*, *blast*)
**qed**
**next**
**from** ‹$P \neq \bot$› **show** ‹$t \in \mathcal{D}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs) \Longrightarrow t \in \mathcal{D}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$› **for** *t*
**by** (*auto simp add*: *denot-projs assms initials-Seq*$_{ptick}$ *Cons-eq-append-conv Cons-eq-map-conv*)
(*metis event*$_{ptick}$*.disc*(*1*) *event*$_{ptick}$*.sel*(*1*) *tickFree-Cons-iff*,
*metis Cons-eq-appendI append-T-imp-tickFree event*$_{ptick}$*.sel*(*1*) *list.distinct*(*1*),
*metis append-Nil initials-memD tickFree-Nil*)
**next**
**fix** *t X* **assume** ‹$(t, X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$› ‹$t \notin \mathcal{D}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
**then consider** (*F-P*) *t'* **where** ‹*ev a* # *t* = *map* (*ev* ∘ *of-ev*) *t'*›
‹(*t'*, *ref-Seq*$_{ptick}$ *X*) ∈ $\mathcal{F}$ *P*› ‹*tF t'*›
| (*F-Q*) *t' r u* **where** ‹*ev a* # *t* = *map* (*ev* ∘ *of-ev*) *t'* @ *u*›
‹*t'* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*› ‹*tF t'*› ‹(*u*, *X*) ∈ $\mathcal{F}$ (*Q r*)›
**by** (*simp add*: *denot-projs assms initials-Seq*$_{ptick}$ *split*: *if-split-asm*) *meson*+
**thus** ‹$(t, X) \in \mathcal{F}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›
**proof** *cases*
**case** *F-P* **thus** ‹$(t, X) \in \mathcal{F}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›
**by** (*auto simp add*: *denot-projs assms*)
**next**
**case** *F-Q* **with** *assms* **show** ‹$(t, X) \in \mathcal{F}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›
**by** (*cases t'*, *simp-all add*: *denot-projs*)
(*meson F-T initials-memI*, *blast*)
**qed**
**next**
**fix** *t X* **assume** ‹$(t, X) \in \mathcal{F}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$› ‹$t \notin \mathcal{D}\ ((P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \sqcap ?rhs)$›
**hence** ‹$(t, X) \in \mathcal{F}\ (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \wedge t \notin \mathcal{D}\ (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \vee$
$(t, X) \in \mathcal{F}\ ?rhs \wedge t \notin \mathcal{D}\ ?rhs$› **by** (*simp add*: *Ndet-projs*)
**thus** ‹$(t, X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
**proof** (*elim disjE conjE*)
**show** ‹$(t, X) \in \mathcal{F}\ (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \Longrightarrow t \notin \mathcal{D}\ (P\ after_\alpha\ a\ ;_{\checkmark}\ Q) \Longrightarrow$
$(t, X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
**by** (*simp add*: *denot-projs assms initials-Seq*$_{ptick}$ ‹$P \neq \bot$› *Cons-eq-append-conv Cons-eq-map-conv*)
(*metis* (*no-types*, *lifting*) *Cons-eq-appendI event*$_{ptick}$*.sel*(*1*) *is-ev-def tickFree-Cons-iff*)
**next**
**from** *assms* **show** ‹$(t, X) \in \mathcal{F}\ ?rhs \Longrightarrow t \notin \mathcal{D}\ ?rhs \Longrightarrow (t, X) \in \mathcal{F}\ ((P\ ;_{\checkmark}\ Q)\ after_\beta\ a)$›
**by** (*simp add*: *denot-projs initials-Seq*$_{ptick}$ ‹$P \neq \bot$›
*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*$_{ptick}$:
‹*ev a* ∉ $P^0$ ⟹ (⋀*r*. ✓(*r*) ∈ $P^0$ ⟹ *ev a* ∉ (*Q r*)$^0$) ⟹
(*P* ;✓ *Q*) *after*$_\beta$ *a* = $\Psi_\beta$ (*P* ;✓ *Q*) *a*›
**by** (*meson skippable-not-initialL-After-Seq*$_{ptick}$)

**lemma** *After-Seq*$_{ptick}$:
‹(*P* ;✓ *Q*) *after*$_\beta$ *a* =
( *if* ∀ *r*. ✓(*r*) ∈ $P^0$ ⟶ *ev a* ∉ (*Q r*)$^0$
*then* *if ev a* ∈ $P^0$ *then P after*$_\alpha$ *a* ;✓ *Q else* $\Psi_\beta$ (*P* ;✓ *Q*) *a*
*else* *if ev a* ∈ $P^0$
*then* (*P after*$_\alpha$ *a* ;✓ *Q*) ⊓ (⊓*r*∈{*r*. ✓(*r*) ∈ $P^0$ ∧ *ev a* ∈ (*Q r*)$^0$}. *Q r after*$_\beta$ *a*)
*else* ⊓*r*∈{*r*. ✓(*r*) ∈ $P^0$ ∧ *ev a* ∈ (*Q r*)$^0$}. *Q r after*$_\beta$ *a*)›
**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* :: ‹'*r* ⇒ '*s* ⇒ '*t option*›
**and** $\Psi_{lhs}$ :: ‹['*a*, '*r*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
**and** $\Psi_{rhs}$ :: ‹['*a*, '*s*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
**and** $\Psi_{ptick}$ :: ‹['*a*, '*t*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*t*) *process*$_{ptick}$›
**begin**

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

**sublocale** *After-Sync*$_{ptick}$*-locale-sym* :
*After-Sync*$_{ptick}$*-locale* ‹λ*s r*. *tick-join r s*› $\Psi_{rhs}$ $\Psi_{lhs}$ $\Psi_{ptick}$
**by** *unfold-locales*

**lemma** *initialL-not-initialR-not-in-After-Sync*$_{ptick}$:
‹(*P* ⟦*S*⟧✓ *Q*) *after*$_{ptick}$ *a* = *P after*$_{lhs}$ *a* ⟦*S*⟧✓ *Q*› (**is** ‹*?lhs* = *?rhs*›)
**if** *initial-hyps*: ‹*ev a* ∈ $P^0$› ‹*ev a* ∉ $Q^0$› **and** *notin*: ‹*a* ∉ *S*›
**proof** (*cases* ‹*P* = ⊥ ∨ *Q* = ⊥›)

**show** ‹$P = \bot \vee Q = \bot \Longrightarrow$ *?lhs* = *?rhs*›
**by** (*elim disjE*) (*simp-all add*: $After_{ptick}$.*After-BOT* $After_{lhs}$.*After-BOT*)
**next**
**from** *initial-hyps* **and** *notin* **have** *init* : ‹*ev a* ∈ $(P \ [\![S]\!]_{\checkmark}\ Q)^0$›
**by** (*simp add*: $initials\text{-}Sync_{ptick}$)
**show** ‹*?lhs* = *?rhs*› **if** *non-BOT* : ‹¬ $(P = \bot \vee Q = \bot)$›
**proof** (*rule Process-eq-optimizedI*)
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**with** *init* **have** ‹*ev a* # *t* ∈ $\mathcal{D}$ $(P \ [\![S]\!]_{\checkmark}\ Q)$› **by** (*simp add*: $After_{ptick}$.*D-After*)
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹*ev a* # *t* = *u* @ *v*› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ $\mathcal{D}$ *P* ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨ *t-P* ∈ $\mathcal{T}$ *P* ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**unfolding** $D\text{-}Sync_{ptick}$ **by** *blast*
**from** ∗(*4*, *5*) *non-BOT* **have** ‹*u* ≠ []›
**by** (*auto simp add*: *BOT-iff-Nil-D dest*: $Nil\text{-}setinterleaves_{ptick}$)
**with** ∗(*1*) **obtain** *u′* **where** ‹*u* = *ev a* # *u′*› ‹*t* = *u′* @ *v*›
**by** (*cases u*) *simp-all*
**from** ∗(*4*, *5*) *initial-hyps*(*2*) **obtain** *t-P′*
**where** ‹*t-P* = *ev a* # *t-P′*› ‹*u′ setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P′*, *t-Q*), *S*)›
**by** (*cases t-P*; *cases t-Q*)
(*auto simp add*: $setinterleaving_{ptick}$-*simps* ‹*u* = *ev a* # *u′*›
*split*: $event_{ptick}$.*splits if-split-asm option.split-asm*
*intro*: *D-T initials-memI*)
**moreover from** ‹*t-P* = *ev a* # *t-P′*› ∗(*2*, *5*)
**have** ‹*t-P′* ∈ $\mathcal{D}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨
*t-P′* ∈ $\mathcal{T}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**by** (*simp add*: $After_{lhs}$.*After-projs initial-hyps*(*1*))
**moreover from** ∗(*2*) **have** ‹*tF u′*› **by** (*simp add*: ‹*u* = *ev a* # *u′*›)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**using** ∗(*3*) **by** (*auto simp add*: ‹*t* = *u′* @ *v*› $D\text{-}Sync_{ptick}$)
**next**
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹*t* = *u* @ *v*› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ $\mathcal{D}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨
*t-P* ∈ $\mathcal{T}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**unfolding** $D\text{-}Sync_{ptick}$ **by** *blast*
**have** ‹*ev a* # *t* = (*ev a* # *u*) @ *v*› **by** (*simp add*: ∗(*1*))
**moreover from** ∗(*2*) **have** ‹*tF* (*ev a* # *u*)› **by** *simp*
**moreover from** ∗(*4*)
**have** ‹*ev a* # *u setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a* # *t-P*, *t-Q*), *S*)›
**by** (*metis notin rev.simps*(*2*) $rev\text{-}setinterleaves_{ptick}$-*rev-rev-iff*
$setinterleaves_{ptick}$-*snoc-notinL*)
**moreover from** ∗(*5*) **have** ‹*ev a* # *t-P* ∈ $\mathcal{D}$ *P* ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨
*ev a* # *t-P* ∈ $\mathcal{T}$ *P* ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**by** (*simp add*: $After_{lhs}$.*After-projs initial-hyps*(*1*))
**ultimately have** ‹*ev a* # *t* ∈ $\mathcal{D}$ $(P \ [\![S]\!]_{\checkmark}\ Q)$›

**using** ∗(*3*) **unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**thus** ‹$t \in \mathcal{D}$ *?lhs*›
**by** (*simp add*: *After*$_{ptick}$.*D-After init*)
**next**
**fix** *t X* **assume** ‹$(t, X) \in \mathcal{F}$ *?lhs*›
‹$t \notin \mathcal{D}$ *?lhs*›
**hence** ‹(*ev a # t*, *X*) $\in \mathcal{F}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)› ‹*ev a # t* $\notin \mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**by** (*simp-all add*: *After*$_{ptick}$.*After-projs init*)
**then obtain** *t-P t-Q X-P X-Q*
**where** ∗ : ‹(*t-P*, *X-P*) $\in \mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) $\in \mathcal{F}$ *Q*›
‹*ev a # t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*X* $\subseteq$ *super-ref-Sync*$_{ptick}$ *tick-join X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**from** ∗(*2*, *3*) *initial-hyps*(*2*) **obtain** *t-P′*
**where** ‹*t-P* = *ev a # t-P′*› ‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P′*, *t-Q*), *S*)›
**by** (*metis Cons-ev-setinterleaves*$_{ptick}$*E F-T initials-memI*)
**moreover from** ∗(*1*) **have** ‹(*t-P′*, *X-P*) $\in \mathcal{F}$ (*P after*$_{lhs}$ *a*)›
**by** (*simp add*: ‹*t-P* = *ev a # t-P′*› *After*$_{lhs}$.*F-After initial-hyps*(*1*))
**ultimately show** ‹(*t*, *X*) $\in \mathcal{F}$ *?rhs*›
**using** ∗(*2*, *4*) **by** (*auto simp add*: *Sync*$_{ptick}$*-projs*)
**next**
**fix** *t X* **assume** ‹(*t*, *X*) $\in \mathcal{F}$ *?rhs*›
‹$t \notin \mathcal{D}$ *?rhs*›
**then obtain** *t-P t-Q X-P X-Q*
**where** ∗ : ‹(*t-P*, *X-P*) $\in \mathcal{F}$ (*P after*$_{lhs}$ *a*)› ‹(*t-Q*, *X-Q*) $\in \mathcal{F}$ *Q*›
‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*X* $\subseteq$ *super-ref-Sync*$_{ptick}$ *tick-join X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**from** ∗(*1*) **have** ‹(*ev a # t-P*, *X-P*) $\in \mathcal{F}$ *P*›
**by** (*simp add*: *After*$_{lhs}$.*F-After initial-hyps*(*1*))
**moreover from** ∗(*3*)
**have** ‹*ev a # t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*ev a # t-P*, *t-Q*), *S*)›
**by** (*metis notin rev.simps*(*2*) *rev-setinterleaves*$_{ptick}$*-rev-rev-iff*
*setinterleaves*$_{ptick}$*-snoc-notinL*)
**ultimately have** ‹(*ev a # t*, *X*) $\in \mathcal{F}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**using** ∗(*2*, *4*) **by** (*auto simp add*: *F-Sync*$_{ptick}$)
**thus** ‹(*t*, *X*) $\in \mathcal{F}$ *?lhs*› **by** (*simp add*: *After*$_{ptick}$.*F-After init*)
**qed**
**qed**

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

**lemma** *not-initialL-in-After-Sync*$_{ptick}$:
‹*ev a* ∉ $P^0$ ⟹ *a* ∈ *S* ⟹
($P$ ⟦*S*⟧$_{✓}$ *Q*) *after*$_{ptick}$ *a* = (*if Q* = ⊥ *then* ⊥ *else* Ψ$_{ptick}$ ($P$ ⟦*S*⟧$_{✓}$ *Q*) *a*)›
**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*$_{ptick}$:
‹*ev a* ∉ $Q^0$ ⟹ *a* ∈ *S* ⟹
($P$ ⟦*S*⟧$_{✓}$ *Q*) *after*$_{ptick}$ *a* = (*if P* = ⊥ *then* ⊥ *else* Ψ$_{ptick}$ ($P$ ⟦*S*⟧$_{✓}$ *Q*) *a*)›
**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*$_{ptick}$:
‹($P$ ⟦*S*⟧$_{✓}$ *Q*) *after*$_{ptick}$ *a* = *P after*$_{lhs}$ *a* ⟦*S*⟧$_{✓}$ *Q after*$_{rhs}$ *a*› (**is** ‹*?lhs* = *?rhs*›)
**if** *initial-hyps*: ‹*ev a* ∈ $P^0$› ‹*ev a* ∈ $Q^0$› **and** *inside*: ‹*a* ∈ *S*›
**proof** (*cases* ‹*P* = ⊥ ∨ *Q* = ⊥›)
**show** ‹*P* = ⊥ ∨ *Q* = ⊥ ⟹ *?lhs* = *?rhs*›
**by** (*elim disjE*) (*simp-all add*: *After.After-BOT*)
**next**
**from** *initial-hyps inside* **have** *init* : ‹*ev a* ∈ ($P$ ⟦*S*⟧$_{✓}$ *Q*)$^0$›
**by** (*simp add*: *initials-Sync*$_{ptick}$)
**show** ‹*?lhs* = *?rhs*› **if** *non-BOT* : ‹¬ (*P* = ⊥ ∨ *Q* = ⊥)›
**proof** (*rule Process-eq-optimizedI*)
**fix** *t* **assume** ‹*t* ∈ 𝒟 (($P$ ⟦*S*⟧$_{✓}$ *Q*) *after*$_{ptick}$ *a*)›
**hence** ‹*ev a* # *t* ∈ 𝒟 ($P$ ⟦*S*⟧$_{✓}$ *Q*)› **by** (*simp add*: *After*$_{ptick}$.*D-After init*)
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹*ev a* # *t* = *u* @ *v*› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{✓\,tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ 𝒟 *P* ∧ *t-Q* ∈ 𝒯 *Q* ∨ *t-P* ∈ 𝒯 *P* ∧ *t-Q* ∈ 𝒟 *Q*›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** ∗(*4*, *5*) *non-BOT* **have** ‹*u* ≠ []›
**by** (*auto simp add*: *BOT-iff-Nil-D dest*: *Nil-setinterleaves*$_{ptick}$)
**with** ∗(*1*) **obtain** *u′* **where** ‹*u* = *ev a* # *u′*› ‹*t* = *u′* @ *v*›
**by** (*cases u*) *simp-all*
**from** ∗(*4*) *inside initial-hyps*(*1*, *2*) ‹*u* = *ev a* # *u′*›
**obtain** *t-P′ t-Q′* **where** ‹*t-P* = *ev a* # *t-P′*› ‹*t-Q* = *ev a* # *t-Q′*›
‹*u′ setinterleaves*$_{✓\,tick\text{-}join}$ ((*t-P′*, *t-Q′*), *S*)›
**by** (*metis* (*no-types*, *opaque-lifting*) *Cons-ev-setinterleaves*$_{ptick}$*E*)
**moreover from** ∗(*2*) **have** ‹*tF u′*› **by** (*simp add*: ‹*u* = *ev a* # *u′*›)
**moreover from** ∗(*5*)
**have** ‹*t-P′* ∈ 𝒟 (*P after*$_{lhs}$ *a*) ∧ *t-Q′* ∈ 𝒯 (*Q after*$_{rhs}$ *a*) ∨
*t-P′* ∈ 𝒯 (*P after*$_{lhs}$ *a*) ∧ *t-Q′* ∈ 𝒟 (*Q after*$_{rhs}$ *a*)›
**by** (*simp add*: ‹*t-P* = *ev a* # *t-P′*› ‹*t-Q* = *ev a* # *t-Q′*›
*After.After-projs initial-hyps*)
**ultimately show** ‹*t* ∈ 𝒟 *?rhs*›
**using** ∗(*3*) **by** (*auto simp add*: ‹*t* = *u′* @ *v*› *D-Sync*$_{ptick}$)
**next**
**fix** *t* **assume** ‹*t* ∈ 𝒟 *?rhs*›

**then obtain** $u\ v$ *t-P t-Q*
**where** $*$ : ‹$t = u$ @ $v$› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ $\mathcal{D}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{T}$ (*Q after*$_{rhs}$ *a*) ∨
*t-P* ∈ $\mathcal{T}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{D}$ (*Q after*$_{rhs}$ *a*)›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** $*$(*1*) **have** ‹*ev a* # *t* = (*ev a* # *u*) @ *v*› **by** *simp*
**moreover from** $*$(*2*) **have** ‹*tF* (*ev a* # *u*)› **by** *simp*
**moreover from** $*$(*4*) **have** ‹*ev a* # *u setinterleaves*$_{\checkmark\, tick\text{-}join}$
((*ev a* # *t-P*, *ev a* # *t-Q*), *S*)›
**by** (*simp add*: *inside*)
**moreover from** $*$(*5*) **have** ‹*ev a* # *t-P* ∈ $\mathcal{D}$ *P* ∧ *ev a* # *t-Q* ∈ $\mathcal{T}$ *Q* ∨
*ev a* # *t-P* ∈ $\mathcal{T}$ *P* ∧ *ev a* # *t-Q* ∈ $\mathcal{D}$ *Q*›
**by** (*simp add*: *After.After-projs initial-hyps*)
**ultimately have** ‹*ev a* # *t* ∈ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**unfolding** *D-Sync*$_{ptick}$ **using** $*$(*3*) **by** *blast*
**thus** ‹*t* ∈ $\mathcal{D}$ *?lhs*› **by** (*simp add*: *After*$_{ptick}$.*D-After init*)
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› ‹*t* ∉ $\mathcal{D}$ *?lhs*›
**hence** ‹(*ev a* # *t*, *X*) ∈ $\mathcal{F}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)› ‹*ev a* # *t* ∉ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**by** (*simp-all add*: *After.After-projs init*)
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*ev a* # *t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *tick-join X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**from** $*$(*3*) **obtain** *t-P′ t-Q′*
**where** ‹*t-P* = *ev a* # *t-P′*› ‹*t-Q* = *ev a* # *t-Q′*›
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P′*, *t-Q′*), *S*)›
**by** (*metis* (*no-types*) *Cons-ev-setinterleaves*$_{ptick}$*E inside*)
**moreover from** $*$(*1*, *2*) **have** ‹(*t-P′*, *X-P*) ∈ $\mathcal{F}$ (*P after*$_{lhs}$ *a*)›
‹(*t-Q′*, *X-Q*) ∈ $\mathcal{F}$ (*Q after*$_{rhs}$ *a*)›
**by** (*simp-all add*: ‹*t-P* = *ev a* # *t-P′*› ‹*t-Q* = *ev a* # *t-Q′*›
*After.F-After initial-hyps*)
**ultimately show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**by** (*auto simp add*: *F-Sync*$_{ptick}$ *intro*!: $*$(*4*))
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› ‹*t* ∉ $\mathcal{D}$ *?rhs*›
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ (*P after*$_{lhs}$ *a*)›
‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ (*Q after*$_{rhs}$ *a*)›
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *tick-join X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**from** $*$(*1*, *2*) **have** ‹(*ev a* # *t-P*, *X-P*) ∈ $\mathcal{F}$ *P*›
‹(*ev a* # *t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
**by** (*simp-all add*: *After.F-After initial-hyps*)
**moreover from** $*$(*3*) **have** ‹*ev a* # *t setinterleaves*$_{\checkmark\, tick\text{-}join}$
((*ev a* # *t-P*, *ev a* # *t-Q*), *S*)›

**by** (*simp add*: *inside*)
**ultimately have** ‹(*ev a # t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧✓ *Q*)›
**using** ∗(*4*) **by** (*simp* (*no-asm*) *add*: *F-Sync*$_{ptick}$) *blast*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** (*simp add*: *After*$_{ptick}$.*F-After init*)
**qed**
**qed**

**lemma** *initialL-initialR-not-in-After-Sync*$_{ptick}$:
‹(*P* ⟦*S*⟧✓ *Q*) *after*$_{ptick}$ *a* = (*P after*$_{lhs}$ *a* ⟦*S*⟧✓ *Q*) ⊓ (*P* ⟦*S*⟧✓ *Q after*$_{rhs}$ *a*)›
(**is** ‹*?lhs* = *?rhs1* ⊓ *?rhs2*›)
**if** *initial-hyps*: ‹*ev a* ∈ *P*$^0$› ‹*ev a* ∈ *Q*$^0$› **and** *notin*: ‹*a* ∉ *S*›
**proof** (*cases* ‹*P* = ⊥ ∨ *Q* = ⊥›)
**show** ‹*P* = ⊥ ∨ *Q* = ⊥ ⟹ *?lhs* = *?rhs1* ⊓ *?rhs2*›
**by** (*elim disjE*) (*simp-all add*: *After.After-BOT*)
**next**
**from** *initial-hyps*(*1*) *notin* **have** *init* : ‹*ev a* ∈ (*P* ⟦*S*⟧✓ *Q*)$^0$›
**by** (*simp add*: *initials-Sync*$_{ptick}$)
**show** ‹*?lhs* = *?rhs1* ⊓ *?rhs2*› **if** *non-BOT* : ‹¬ (*P* = ⊥ ∨ *Q* = ⊥)›
**proof** (*rule Process-eq-optimizedI*)
**fix** *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**hence** ‹*ev a # t* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧✓ *Q*)›
**by** (*simp add*: *After*$_{ptick}$.*D-After init*)
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹*ev a # t* = *u* @ *v*› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*✓*tick-join* ((*t-P*, *t-Q*), *S*)›
‹*t-P* ∈ $\mathcal{D}$ *P* ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨ *t-P* ∈ $\mathcal{T}$ *P* ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** ∗(*4*, *5*) *non-BOT* **have** ‹*u* ≠ []›
**by** (*auto simp add*: *BOT-iff-Nil-D dest*: *Nil-setinterleaves*$_{ptick}$)
**with** ∗(*1*) **obtain** *u′* **where** ‹*u* = *ev a # u′*› ‹*t* = *u′* @ *v*›
**by** (*cases u*) *simp-all*
**from** ∗(*2*) **have** ‹*tF u′*› **by** (*simp add*: ‹*u* = *ev a # u′*›)
**from** ∗(*4*) *notin* ‹*u* = *ev a # u′*›
**consider** *t-P′* **where** ‹*t-P* = *ev a # t-P′*›
‹*u′ setinterleaves*✓*tick-join* ((*t-P′*, *t-Q*), *S*)›
| *t-Q′* **where** ‹*t-Q* = *ev a # t-Q′*›
‹*u′ setinterleaves*✓*tick-join* ((*t-P*, *t-Q′*), *S*)›
**by** (*metis* (*no-types*) *Cons-ev-setinterleaves*$_{ptick}$*E*)
**thus** ‹*t* ∈ $\mathcal{D}$ (*?rhs1* ⊓ *?rhs2*)›
**proof** *cases*
**fix** *t-P′* **assume** $ : ‹*t-P* = *ev a # t-P′*›
‹*u′ setinterleaves*✓*tick-join* ((*t-P′*, *t-Q*), *S*)›
**from** ∗(*5*) **have** ‹*t-P′* ∈ $\mathcal{D}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨
*t-P′* ∈ $\mathcal{T}$ (*P after*$_{lhs}$ *a*) ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**by** (*simp add*: $(*1*) *After*$_{lhs}$.*After-projs initial-hyps*(*1*))
**with** $(*2*) ∗(*3*) ‹*tF u′*› **have** ‹*t* ∈ $\mathcal{D}$ *?rhs1*›
**by** (*auto simp add*: ‹*t* = *u′* @ *v*› *D-Sync*$_{ptick}$)

**thus** ‹$t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2)$› **by** (*simp add*: *D-Ndet*)
**next**
**fix** *t-Q'* **assume** $ : ‹*t-Q* = *ev a* # *t-Q'*›
‹*u' setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*t-P*, *t-Q'*), *S*)›
**from** ∗(*5*) **have** ‹$t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q' \in \mathcal{T}\ (Q\ after_{rhs}\ a) \vee$
$t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q' \in \mathcal{D}\ (Q\ after_{rhs}\ a)$›
**by** (*simp add*: $(*1*) *After*_{rhs}.*After-projs initial-hyps*(*2*))
**with** $(*2*) ∗(*3*) ‹*tF u'*› **have** ‹$t \in \mathcal{D}$ *?rhs2*›
**by** (*auto simp add*: ‹$t = u' @ v$› *D-Sync*$_{ptick}$)
**thus** ‹$t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2)$› **by** (*simp add*: *D-Ndet*)
**qed**
**next**
**fix** *t* **assume** ‹$t \in \mathcal{D}\ (?rhs1 \sqcap ?rhs2)$›
**then consider** ‹$t \in \mathcal{D}$ *?rhs1*› | ‹$t \in \mathcal{D}$ *?rhs2*›
**by** (*auto simp add*: *D-Ndet*)
**hence** ‹*ev a* # $t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)$›
**proof** *cases*
**assume** ‹$t \in \mathcal{D}$ *?rhs1*›
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹$t = u @ v$› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹$t\text{-}P \in \mathcal{D}\ (P\ after_{lhs}\ a) \wedge t\text{-}Q \in \mathcal{T}\ Q \vee$
$t\text{-}P \in \mathcal{T}\ (P\ after_{lhs}\ a) \wedge t\text{-}Q \in \mathcal{D}\ Q$›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** ∗(*1*) **have** ‹*ev a* # $t = (ev\ a \#\ u) @ v$› **by** *simp*
**moreover from** ∗(*2*) **have** ‹*tF* (*ev a* # *u*)› **by** *simp*
**moreover from** ∗(*4*)
**have** ‹*ev a* # *u setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*ev a* # *t-P*, *t-Q*), *S*)›
**by** (*metis notin rev.simps*(*2*) *rev-setinterleaves*$_{ptick}$-*rev-rev-iff*
*setinterleaves*$_{ptick}$-*snoc-notinL*)
**moreover from** ∗(*5*) **have** ‹*ev a* # $t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee$
*ev a* # $t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q$›
**by** (*simp add*: $After_{lhs}.$*After-projs initial-hyps*(*1*))
**ultimately show** ‹*ev a* # $t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)$›
**using** ∗(*3*) **unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**next**
**assume** ‹$t \in \mathcal{D}$ *?rhs2*›
**then obtain** *u v t-P t-Q*
**where** ∗ : ‹$t = u @ v$› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
‹$t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ (Q\ after_{rhs}\ a) \vee$
$t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ (Q\ after_{rhs}\ a)$›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** ∗(*1*) **have** ‹*ev a* # $t = (ev\ a \#\ u) @ v$› **by** *simp*
**moreover from** ∗(*2*) **have** ‹*tF* (*ev a* # *u*)› **by** *simp*
**moreover from** ∗(*4*) **have** ‹*ev a* # *u setinterleaves*$_{\checkmark\ tick\text{-}join}$ ((*t-P*, *ev a* # *t-Q*), *S*)›
**by** (*metis notin rev.simps*(*2*) *rev-setinterleaves*$_{ptick}$-*rev-rev-iff*
*setinterleaves*$_{ptick}$-*snoc-notinR*)

**moreover from** $*(5)$ **have** $\langle t\text{-}P \in \mathcal{D}\ P \wedge ev\ a \#\ t\text{-}Q \in \mathcal{T}\ Q \vee$
$t\text{-}P \in \mathcal{T}\ P \wedge ev\ a \#\ t\text{-}Q \in \mathcal{D}\ Q\rangle$
**by** (*simp add*: $After_{rhs}$.*After-projs initial-hyps*(*2*))
**ultimately show** $\langle ev\ a \#\ t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)\rangle$
**using** $*(3)$ **unfolding** $D\text{-}Sync_{ptick}$ **by** *blast*
**qed**
**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** $\langle (ev\ a \#\ t, X) \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark}\ Q)\rangle$ $\langle ev\ a \#\ t \notin \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)\rangle$
**by** (*simp-all add*: $After_{ptick}$.*After-projs init*)
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : $\langle (t\text{-}P, X\text{-}P) \in \mathcal{F}\ P\rangle$ $\langle (t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q\rangle$
$\langle ev\ a \#\ t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P, t\text{-}Q), S)\rangle$
$\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick}\ tick\text{-}join\ X\text{-}P\ S\ X\text{-}Q\rangle$
**unfolding** $Sync_{ptick}$*-projs* **by** *blast*
**from** $*(3)$ *notin*
**consider** *t-P′* **where** $\langle t\text{-}P = ev\ a \#\ t\text{-}P'\rangle$
$\langle t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P', t\text{-}Q), S)\rangle$
| *t-Q′* **where** $\langle t\text{-}Q = ev\ a \#\ t\text{-}Q'\rangle$
$\langle t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P, t\text{-}Q'), S)\rangle$
**by** (*metis* (*no-types*) $Cons\text{-}ev\text{-}setinterleaves_{ptick}E$)
**thus** $\langle (t, X) \in \mathcal{F}$ (*?rhs1* $\sqcap$ *?rhs2*)$\rangle$
**proof** *cases*
**fix** *t-P′* **assume** \$ : $\langle t\text{-}P = ev\ a \#\ t\text{-}P'\rangle$
$\langle t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P', t\text{-}Q), S)\rangle$
**from** $*(1)$ **have** $\langle (t\text{-}P', X\text{-}P) \in \mathcal{F}\ (P\ after_{lhs}\ a)\rangle$
**by** (*simp add*: \$(*1*) $After_{lhs}$.*F-After initial-hyps*(*1*))
**with** \$(*2*) $*(2, 4)$ **have** $\langle (t, X) \in \mathcal{F}$ *?rhs1*$\rangle$
**by** (*auto simp add*: $F\text{-}Sync_{ptick}$)
**thus** $\langle (t, X) \in \mathcal{F}$ (*?rhs1* $\sqcap$ *?rhs2*)$\rangle$ **by** (*simp add*: *F-Ndet*)
**next**
**fix** *t-Q′* **assume** \$ : $\langle t\text{-}Q = ev\ a \#\ t\text{-}Q'\rangle$
$\langle t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((t\text{-}P, t\text{-}Q'), S)\rangle$
**from** $*(2)$ **have** $\langle (t\text{-}Q', X\text{-}Q) \in \mathcal{F}\ (Q\ after_{rhs}\ a)\rangle$
**by** (*simp add*: \$(*1*) $After_{rhs}$.*F-After initial-hyps*(*2*))
**with** \$(*2*) $*(1, 4)$ **have** $\langle (t, X) \in \mathcal{F}$ *?rhs2*$\rangle$
**by** (*auto simp add*: $F\text{-}Sync_{ptick}$)
**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** $\langle (ev\ a \#\ t, X) \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark}\ Q)\rangle$
**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** $*$ : ‹(t-P, X-P) ∈ $\mathcal{F}$ (P $after_{lhs}$ a)› ‹(t-Q, X-Q) ∈ $\mathcal{F}$ Q›
‹t $setinterleaves_{\checkmark tick\text{-}join}$ ((t-P, t-Q), S)›
‹X ⊆ $super\text{-}ref\text{-}Sync_{ptick}$ tick-join X-P S X-Q›
**unfolding** $Sync_{ptick}$-projs **by** blast
**from** $*$(1) **have** ‹(ev a # t-P, X-P) ∈ $\mathcal{F}$ P›
**by** (simp add: $After_{lhs}$.F-After initial-hyps(1))
**moreover from** $*$(3)
**have** ‹ev a # t $setinterleaves_{\checkmark tick\text{-}join}$ ((ev a # t-P, t-Q), S)›
**by** (metis notin rev.simps(2) $rev\text{-}setinterleaves_{ptick}$-rev-rev-iff $setinterleaves_{ptick}$-snoc-notinL)
**ultimately show** ‹(ev a # t, X) ∈ $\mathcal{F}$ (P $[\![S]\!]_{\checkmark}$ Q)›
**by** (auto simp add: F-$Sync_{ptick}$ intro!: $*$(2, 4))
**next**
**assume** ‹(t, X) ∈ $\mathcal{F}$ ?rhs2› ‹t ∉ $\mathcal{D}$ ?rhs2›
**then obtain** t-P t-Q X-P X-Q
**where** $*$ : ‹(t-P, X-P) ∈ $\mathcal{F}$ P› ‹(t-Q, X-Q) ∈ $\mathcal{F}$ (Q $after_{rhs}$ a)›
‹t $setinterleaves_{\checkmark tick\text{-}join}$ ((t-P, t-Q), S)›
‹X ⊆ $super\text{-}ref\text{-}Sync_{ptick}$ tick-join X-P S X-Q›
**unfolding** $Sync_{ptick}$-projs **by** blast
**from** $*$(2) **have** ‹(ev a # t-Q, X-Q) ∈ $\mathcal{F}$ Q›
**by** (simp add: $After_{rhs}$.F-After initial-hyps(2))
**moreover from** $*$(3)
**have** ‹ev a # t $setinterleaves_{\checkmark tick\text{-}join}$ ((t-P, ev a # t-Q), S)›
**by** (metis notin rev.simps(2) $rev\text{-}setinterleaves_{ptick}$-rev-rev-iff $setinterleaves_{ptick}$-snoc-notinR)
**ultimately show** ‹(ev a # t, X) ∈ $\mathcal{F}$ (P $[\![S]\!]_{\checkmark}$ Q)›
**by** (auto simp add: F-$Sync_{ptick}$ intro!: $*$(1, 4))
**qed**
**thus** ‹(t, X) ∈ $\mathcal{F}$ ?lhs› **by** (simp add: $After_{ptick}$.F-After init)
**qed**
**qed**

**lemma** $not\text{-}initialL\text{-}not\text{-}initialR\text{-}After\text{-}Sync_{ptick}$ :
‹ev a ∉ $P^0$ ⟹ ev a ∉ $Q^0$ ⟹ (P $[\![S]\!]_{\checkmark}$ Q) $after_{ptick}$ a = $\Psi_{ptick}$ (P $[\![S]\!]_{\checkmark}$ Q) a›
**by** (subst $After_{ptick}$.not-initial-After) (auto simp add: $initials\text{-}Sync_{ptick}$)

Finally, the monster theorem !

**theorem** $After\text{-}Sync_{ptick}$:
‹(P $[\![S]\!]_{\checkmark}$ Q) $after_{ptick}$ a =
( if P = ⊥ ∨ Q = ⊥ then ⊥
else if ev a ∈ $P^0$ ∧ ev a ∈ $Q^0$
then if a ∈ S then P $after_{lhs}$ a $[\![S]\!]_{\checkmark}$ Q $after_{rhs}$ a
else (P $after_{lhs}$ a $[\![S]\!]_{\checkmark}$ Q) ⊓ (P $[\![S]\!]_{\checkmark}$ Q $after_{rhs}$ a)
else if ev a ∈ $P^0$ ∧ a ∉ S then P $after_{lhs}$ a $[\![S]\!]_{\checkmark}$ Q
else if ev a ∈ $Q^0$ ∧ a ∉ S then P $[\![S]\!]_{\checkmark}$ Q $after_{rhs}$ a
else $\Psi_{ptick}$ (P $[\![S]\!]_{\checkmark}$ Q) a)›
**by** (auto simp add: $After_{ptick}$.After-BOT initialL-not-initialR-not-in-$After\text{-}Sync_{ptick}$

*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* $\Psi_\alpha$ $\Omega_\alpha$ $\Psi_\beta$ $\Omega_\beta$
**for** $\Psi_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
**and** $\Omega_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*r*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
**and** $\Psi_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
**and** $\Omega_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*s*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›

**sublocale** *AfterExtDuplicated-same-events* ⊆ *AfterDuplicated-same-events* .
— Recovering *AfterDuplicated-same-events.After-Seq*$_{ptick}$

**context** *AfterExtDuplicated-same-events*
**begin**

**notation** *After*$_{tick\alpha}$.*After* (**infixl** ‹*after*$_\alpha$› 86)
**notation** *After*$_{tick\beta}$.*After* (**infixl** ‹*after*$_\beta$› 86)
**notation** *After*$_{tick\alpha}$.*After*$_{tick}$ (**infixl** ‹*after*$_{\checkmark\alpha}$› 86)
**notation** *After*$_{tick\beta}$.*After*$_{tick}$ (**infixl** ‹*after*$_{\checkmark\beta}$› 86)

**lemma** *After*$_{tick}$-*Seq*$_{ptick}$ :
‹(*P* ;$_\checkmark$ *Q*) *after*$_{\checkmark\beta}$ *e* =
(*case e of* ✓(*r*) ⇒ $\Omega_\beta$ (*P* ;$_\checkmark$ *Q*) *r*
| *ev a* ⇒
*if* ∀ *r*. ✓(*r*) ∈ $P^0$ ⟶ *ev a* ∉ (*Q r*)$^0$
*then* *if ev a* ∈ $P^0$
*then P after*$_{\checkmark\alpha}$ *ev a* ;$_\checkmark$ *Q else* $\Psi_\beta$ (*P* ;$_\checkmark$ *Q*) *a*
*else* *if ev a* ∈ $P^0$
*then* (*P after*$_{\checkmark\alpha}$ *ev a* ;$_\checkmark$ *Q*) ⊓
(⊓*r*∈{*r*. ✓(*r*) ∈ $P^0$ ∧ *ev a* ∈ (*Q r*)$^0$}. *Q r after*$_{\checkmark\beta}$ *ev a*)
*else* ⊓*r*∈{*r*. ✓(*r*) ∈ $P^0$ ∧ *ev a* ∈ (*Q r*)$^0$}. *Q r after*$_{\checkmark\beta}$ *ev a*)›

**by** (*auto simp add*: $After_{tick\beta}.After_{tick}$-*def* $After_{tick\alpha}.After_{tick}$-*def After-Seq*$_{ptick}$ *split*: $event_{ptick}.split$)

**end**

## Synchronization Product

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

**sublocale** $After\text{-}Sync_{ptick}\text{-}locale$ *tick-join* $\Psi_{lhs}$ $\Psi_{rhs}$ $\Psi_{ptick}$ **by** *unfold-locales*

**sublocale** $AfterExt\text{-}Sync_{ptick}\text{-}locale\text{-}sym$ :
  $AfterExt\text{-}Sync_{ptick}\text{-}locale$ ‹$\lambda s\ r.$ *tick-join* $r\ s$› $\Psi_{rhs}$ $\Omega_{rhs}$ $\Psi_{lhs}$ $\Omega_{lhs}$ $\Psi_{ptick}$ $\Omega_{ptick}$
  **by** *unfold-locales*

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

**theorem** $After_{tick}\text{-}Sync_{ptick}$:
  ‹($P$ $[\![S]\!]_{\checkmark}$ $Q$) $after_{\checkmark ptick}$ $e$ =
   (*case* $e$ *of* ✓(*r-s*) ⇒ $\Omega_{ptick}$ ($P$ $[\![S]\!]_{\checkmark}$ $Q$) *r-s*
          | *ev a* ⇒
     *if* $P = \bot \vee Q = \bot$ *then* $\bot$
     *else if ev a* $\in P^0 \wedge$ *ev a* $\in Q^0$
       *then if* $a \in S$ *then* $P$ $after_{\checkmark lhs}$ *ev a* $[\![S]\!]_{\checkmark}$ $Q$ $after_{\checkmark rhs}$ *ev a*
         *else* ($P$ $after_{\checkmark lhs}$ *ev a* $[\![S]\!]_{\checkmark}$ $Q$) ⊓ ($P$ $[\![S]\!]_{\checkmark}$ $Q$ $after_{\checkmark rhs}$ *ev a*)
       *else if ev a* $\in P^0 \wedge a \notin S$ *then* $P$ $after_{\checkmark lhs}$ *ev a* $[\![S]\!]_{\checkmark}$ $Q$
         *else if ev a* $\in Q^0 \wedge a \notin S$ *then* $P$ $[\![S]\!]_{\checkmark}$ $Q$ $after_{\checkmark rhs}$ *ev a*
           *else* $\Psi_{ptick}$ ($P$ $[\![S]\!]_{\checkmark}$ $Q$) $a$)›
  **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* $\Psi_\alpha$ $\Omega_\alpha$ $\tau$-*trans*$_\alpha$ $\Psi_\beta$ $\Omega_\beta$ $\tau$-*trans*$_\beta$
  **for** $\Psi_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
    **and** $\Omega_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, '*r*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
     **and** $\tau$-*trans*$_\alpha$ :: ‹[('*a*, '*r*) *process*$_{ptick}$, ('*a*, '*r*) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\alpha\rightsquigarrow_\tau$› *50*)
    **and** $\Psi_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*a*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
    **and** $\Omega_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, '*s*] ⇒ ('*a*, '*s*) *process*$_{ptick}$›
   **and** $\tau$-*trans*$_\beta$ :: ‹[('*a*, '*s*) *process*$_{ptick}$, ('*a*, '*s*) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\beta\rightsquigarrow_\tau$› *50*)

**sublocale** *OpSemTransitionsDuplicated-same-events* ⊆ *AfterExtDuplicated-same-events*
  **by** *unfold-locales*

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

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

**lemma** $\tau$-*trans-Seq*$_{ptick}$*R*: ‹*P* ;$_\checkmark$ *Q* $_\beta\rightsquigarrow_\tau$ *Q*'› **if** ‹*P* $_\alpha\rightsquigarrow_\checkmark$ *r* *P*'› **and** ‹*Q* *r* $_\beta\rightsquigarrow_\tau$ *Q*'›
**proof** −
  **from** *that*(*1*) **have** ‹*P* ⊑$_{FD}$ *SKIP r*›
   **by** (*meson OpSemTransitions*$_\alpha$.*exists-tick-trans-is-initial-tick initial-tick-iff-FD-SKIP*)
  **with** *FD-iff-eq-Ndet* **have** ‹*P* = *P* ⊓ *SKIP r*› **..**
  **hence** ‹*P* ;$_\checkmark$ *Q* = (*P* ;$_\checkmark$ *Q*) ⊓ (*SKIP r* ;$_\checkmark$ *Q*)›
    **by** (*metis Seq*$_{ptick}$-*distrib-Ndet-right*)
  **also have** ‹... = (*P* ;$_\checkmark$ *Q*) ⊓ *Q r*› **by** *simp*
  **also have** ‹... $_\beta\rightsquigarrow_\tau$ *Q r*› **by** (*simp add*: *OpSemTransitions*$_\beta$.$\tau$-*trans-NdetR*)
  **finally show** ‹*P* ;$_\checkmark$ *Q* $_\beta\rightsquigarrow_\tau$ *Q*'› **by** (*rule OpSemTransitions*$_\beta$.$\tau$-*trans-transitivity*) (*fact that*(*2*))
**qed**

**lemma** ‹✓(*r*) ∈ *P*$^0$ ⟹ *Q r* $_\beta\rightsquigarrow_e$ *Q*' ⟹ *P* ;$_\checkmark$ *Q* $_\beta\rightsquigarrow_e$ *Q*'› **for** *P* :: ‹('*a*, '*r*) *process*$_{ptick}$›
  **by** (*meson OpSemTransitions*$_\beta$.$\tau$-*trans-eq OpSemTransitions*$_\beta$.$\tau$-*trans-ev-trans*
        $\tau$-*trans-Seq*$_{ptick}$*R OpSemTransitions*$_\alpha$.*exists-tick-trans-is-initial-tick*)

**end**

**locale** *OpSemTransitionsSeq*$_{ptick}$ =
  *OpSemTransitionsDuplicated-same-events* $\Psi_\alpha$ $\Omega_\alpha$ $\tau$-*trans*$_\alpha$ $\Psi_\beta$ $\Omega_\beta$ $\tau$-*trans*$_\beta$

**for** $\Psi_\alpha$ :: ‹[('a, 'r) *process*$_{ptick}$, 'a] ⇒ ('a, 'r) *process*$_{ptick}$›
**and** $\Omega_\alpha$ :: ‹[('a, 'r) *process*$_{ptick}$, 'r] ⇒ ('a, 'r) *process*$_{ptick}$›
**and** $\tau$-*trans*$_\alpha$ :: ‹[('a, 'r) *process*$_{ptick}$, ('a, 'r) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\alpha\!\rightsquigarrow_\tau$› *50*)
**and** $\Psi_\beta$ :: ‹[('a, 's) *process*$_{ptick}$, 'a] ⇒ ('a, 's) *process*$_{ptick}$›
**and** $\Omega_\beta$ :: ‹[('a, 's) *process*$_{ptick}$, 's] ⇒ ('a, 's) *process*$_{ptick}$›
**and** $\tau$-*trans*$_\beta$ :: ‹[('a, 's) *process*$_{ptick}$, ('a, 's) *process*$_{ptick}$] ⇒ *bool*› (**infixl** ‹$_\beta\!\rightsquigarrow_\tau$› *50*) +
**assumes** $\tau$-*trans-Seq*$_{ptick}L$ : ‹$P\ {}_\alpha\!\rightsquigarrow_\tau\ P' \Longrightarrow P\ ;_{\checkmark}\ Q\ {}_\beta\!\rightsquigarrow_\tau\ P'\ ;_{\checkmark}\ Q$›
**begin**

**lemma** *ev-trans-Seq*$_{ptick}L$: ‹$P\ {}_\alpha\!\rightsquigarrow_e\ P' \Longrightarrow P\ ;_{\checkmark}\ Q\ {}_\beta\!\rightsquigarrow_e\ P'\ ;_{\checkmark}\ Q$›
**by** (*cases* ‹$P = \bot$›, *solves* ‹*simp add*: *OpSemTransitions*$_\beta$.*BOT-ev-trans-anything*›)
(*auto simp add*: *OpSemTransitions*$_\beta$.*ev-trans-def After*$_{tick}$*-Seq*$_{ptick}$ *initials-Seq*$_{ptick}$
*OpSemTransitions*$_\alpha$.*ev-trans-def*
*intro*: *OpSemTransitions*$_\beta$.$\tau$*-trans-NdetL OpSemTransitions*$_\beta$.$\tau$*-trans-transitivity* $\tau$*-trans-Seq*$_{ptick}L$)

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

**end**

## Synchronization Product

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

**sublocale** *AfterExt-Sync*$_{ptick}$*-locale* **by** *unfold-locales*

**sublocale** *OpSemTransitions-Sync$_{ptick}$-locale-sym* :
*OpSemTransitions-Sync$_{ptick}$-locale*
‹$\lambda s\ r.\ r \otimes\checkmark\ s$› $\Psi_{rhs}$ $\Omega_{rhs}$ ‹$(_{rhs}\!\rightsquigarrow_{\tau})$› $\Psi_{lhs}$ $\Omega_{lhs}$ ‹$(_{lhs}\!\rightsquigarrow_{\tau})$› $\Psi_{ptick}$ $\Omega_{ptick}$ ‹$(_{ptick}\!\rightsquigarrow_{\tau})$›
**proof** *unfold-locales*
**show** ‹$P\ {}_{rhs}\!\rightsquigarrow_{\tau}\ P' \Longrightarrow P\ [\![A]\!]_{\checkmark sym}\ Q\ {}_{ptick}\!\rightsquigarrow_{\tau}\ P'\ [\![A]\!]_{\checkmark sym}\ Q$› **for** $P\ P'\ A\ Q$
**by** (*simp add*: *Sync$_{ptick}$-sym* $\tau$*-trans-Sync$_{ptick}R$*)
**next**
**show** ‹$Q\ {}_{lhs}\!\rightsquigarrow_{\tau}\ Q' \Longrightarrow P\ [\![A]\!]_{\checkmark sym}\ Q\ {}_{ptick}\!\rightsquigarrow_{\tau}\ P\ [\![A]\!]_{\checkmark sym}\ Q'$› **for** $Q\ Q'\ A\ P$
**by** (*simp add*: *Sync$_{ptick}$-sym* $\tau$*-trans-Sync$_{ptick}L$*)
**qed**

**notation** *OpSemTransitions$_{lhs}$.ev-trans* (‹- $_{lhs}\!\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
**notation** *OpSemTransitions$_{lhs}$.tick-trans* (‹- $_{lhs}\!\rightsquigarrow_{\checkmark}$- -› [*50*, *3*, *51*] *50*)
**notation** *OpSemTransitions$_{rhs}$.ev-trans* (‹- $_{rhs}\!\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
**notation** *OpSemTransitions$_{rhs}$.tick-trans* (‹- $_{rhs}\!\rightsquigarrow_{\checkmark}$- -› [*50*, *3*, *51*] *50*)
**notation** *OpSemTransitions$_{ptick}$.ev-trans* (‹- $_{ptick}\!\rightsquigarrow$- -› [*50*, *3*, *51*] *50*)
**notation** *OpSemTransitions$_{ptick}$.tick-trans* (‹- $_{ptick}\!\rightsquigarrow_{\checkmark}$- -› [*50*, *3*, *51*] *50*)

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

**lemma** $\tau$*-trans-SKIP-Sync$_{ptick}L$* :
‹$P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow_{\tau}\ SKIP\ r\ [\![A]\!]_{\checkmark}\ Q$› **if** ‹$P\ {}_{lhs}\!\rightsquigarrow_{\checkmark} r\ P'$›
**proof** −
**from** *that* **have** ‹$P \sqsubseteq_{FD} SKIP\ r$›
**by** (*simp add*: *OpSemTransitions$_{lhs}$.tick-trans-def initial-tick-iff-FD-SKIP*)
**with** *FD-iff-eq-Ndet* **have** ‹$P = P \sqcap SKIP\ r$› **..**
**hence** ‹$P\ [\![A]\!]_{\checkmark}\ Q = (P\ [\![A]\!]_{\checkmark}\ Q) \sqcap (SKIP\ r\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*metis Sync$_{ptick}$-distrib-Ndet-right*)
**also have** ‹$\ldots\ {}_{ptick}\!\rightsquigarrow_{\tau}\ SKIP\ r\ [\![A]\!]_{\checkmark}\ Q$›
**by** (*fact OpSemTransitions$_{ptick}$.*$\tau$*-trans-NdetR*)
**finally show** ‹$P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow_{\tau}\ SKIP\ r\ [\![A]\!]_{\checkmark}\ Q$› **.**
**qed**

**lemma** $\tau$*-trans-SKIP-Sync$_{ptick}R$* :
‹$P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow_{\tau}\ P\ [\![A]\!]_{\checkmark}\ SKIP\ s$› **if** ‹$Q\ {}_{rhs}\!\rightsquigarrow_{\checkmark} s\ Q'$›
**proof** −
**from** *that* **have** ‹$Q \sqsubseteq_{FD} SKIP\ s$›
**by** (*simp add*: *OpSemTransitions$_{rhs}$.tick-trans-def initial-tick-iff-FD-SKIP*)
**with** *FD-iff-eq-Ndet* **have** ‹$Q = Q \sqcap SKIP\ s$› **..**
**hence** ‹$P\ [\![A]\!]_{\checkmark}\ Q = (P\ [\![A]\!]_{\checkmark}\ Q) \sqcap (P\ [\![A]\!]_{\checkmark}\ SKIP\ s)$›
**by** (*metis Sync$_{ptick}$-distrib-Ndet-left*)
**also have** ‹$\ldots\ {}_{ptick}\!\rightsquigarrow_{\tau}\ P\ [\![A]\!]_{\checkmark}\ SKIP\ s$›
**by** (*fact OpSemTransitions$_{ptick}$.*$\tau$*-trans-NdetR*)
**finally show** ‹$P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow_{\tau}\ P\ [\![A]\!]_{\checkmark}\ SKIP\ s$› **.**
**qed**

**lemma** *tick-trans-SKIP-Sync$_{ptick}$-SKIP*:

$\langle r \otimes\checkmark\ s = \mathit{Some}\ r\text{-}s \Longrightarrow \mathit{SKIP}\ r\ [\![A]\!]_{\checkmark}\ \mathit{SKIP}\ s\ {}_{ptick}\!\rightsquigarrow_{\checkmark} r\text{-}s\ \Omega_{ptick}\ (\mathit{SKIP}\ r\text{-}s)\ r\text{-}s\rangle$
**by** (*simp add*: $\mathit{OpSemTransitions}_{ptick}.\mathit{SKIP\text{-}trans\text{-}tick\text{-}}\Omega\text{-}\mathit{SKIP}$)

**lemma** $\mathit{ev\text{-}trans\text{-}Sync}_{ptick}L$ :
$\langle a \notin A \Longrightarrow P\ {}_{lhs}\!\rightsquigarrow a\ P' \Longrightarrow P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow a\ P'\ [\![A]\!]_{\checkmark}\ Q\rangle$
**by** (*auto simp add*: $\mathit{OpSemTransitions}_{ptick}.\mathit{ev\text{-}trans\text{-}def}\ \mathit{After}_{tick}\text{-}\mathit{Sync}_{ptick}$
$\mathit{initials\text{-}Sync}_{ptick}\ \mathit{OpSemTransitions}_{lhs}.\mathit{ev\text{-}trans\text{-}def}$
*intro*: $\mathit{OpSemTransitions}_{ptick}.\mathit{BOT}\text{-}\tau\text{-}\mathit{trans\text{-}anything}\ \mathit{OpSemTransitions}_{ptick}.\tau\text{-}\mathit{trans\text{-}NdetL}$
$\mathit{OpSemTransitions}_{ptick}.\tau\text{-}\mathit{trans\text{-}transitivity}\ \tau\text{-}\mathit{trans\text{-}Sync}_{ptick}L$)

**lemma** $\mathit{ev\text{-}trans\text{-}Sync}_{ptick}R$ :
$\langle a \notin A \Longrightarrow Q\ {}_{rhs}\!\rightsquigarrow a\ Q' \Longrightarrow P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow a\ P\ [\![A]\!]_{\checkmark}\ Q'\rangle$
**by** (*auto simp add*: $\mathit{OpSemTransitions}_{ptick}.\mathit{ev\text{-}trans\text{-}def}\ \mathit{After}_{tick}\text{-}\mathit{Sync}_{ptick}$
$\mathit{initials\text{-}Sync}_{ptick}\ \mathit{OpSemTransitions}_{rhs}.\mathit{ev\text{-}trans\text{-}def}$
*intro*: $\mathit{OpSemTransitions}_{ptick}.\mathit{BOT}\text{-}\tau\text{-}\mathit{trans\text{-}anything}\ \mathit{OpSemTransitions}_{ptick}.\tau\text{-}\mathit{trans\text{-}NdetR}$
$\mathit{OpSemTransitions}_{ptick}.\tau\text{-}\mathit{trans\text{-}transitivity}\ \tau\text{-}\mathit{trans\text{-}Sync}_{ptick}R$)

**lemma** $\mathit{ev\text{-}trans\text{-}Sync}_{ptick}LR$ :
$\langle a \in A \Longrightarrow P\ {}_{lhs}\!\rightsquigarrow a\ P' \Longrightarrow Q\ {}_{rhs}\!\rightsquigarrow a\ Q' \Longrightarrow P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\rightsquigarrow a\ P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$
**by** (*auto simp add*: $\mathit{OpSemTransitions}_{ptick}.\mathit{ev\text{-}trans\text{-}def}\ \mathit{OpSemTransitions}_{lhs}.\mathit{ev\text{-}trans\text{-}def}$
$\mathit{OpSemTransitions}_{rhs}.\mathit{ev\text{-}trans\text{-}def}\ \mathit{After}_{tick}\text{-}\mathit{Sync}_{ptick}\ \mathit{initials\text{-}Sync}_{ptick}$
*intro*: $\mathit{OpSemTransitions}_{ptick}.\mathit{BOT}\text{-}\tau\text{-}\mathit{trans\text{-}anything}$
$\mathit{OpSemTransitions}_{ptick}.\tau\text{-}\mathit{trans\text{-}transitivity}$
$\tau\text{-}\mathit{trans\text{-}Sync}_{ptick}L\ \tau\text{-}\mathit{trans\text{-}Sync}_{ptick}R$)

**lemmas** $\mathit{Sync}_{ptick}\text{-}\mathit{OpSem\text{-}rules} = \tau\text{-}\mathit{trans\text{-}Sync}_{ptick}L\ \tau\text{-}\mathit{trans\text{-}Sync}_{ptick}R$
$\mathit{ev\text{-}trans\text{-}Sync}_{ptick}L\ \mathit{ev\text{-}trans\text{-}Sync}_{ptick}R$
$\mathit{ev\text{-}trans\text{-}Sync}_{ptick}LR$
$\tau\text{-}\mathit{trans\text{-}SKIP\text{-}Sync}_{ptick}L\ \tau\text{-}\mathit{trans\text{-}SKIP\text{-}Sync}_{ptick}R$
$\mathit{tick\text{-}trans\text{-}SKIP\text{-}Sync}_{ptick}\text{-}\mathit{SKIP}$

**end**

# Chapter 10

# Declensions of the Generalized Synchronization Product

**unbundle** *option-type-syntax*

## 10.1 Interpretations

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

### 10.1.1 Classical Version

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

**interpretation** $Sync_{Classic}$ : $Sync_{ptick}$-*comm-locale*
  ‹$\lambda r\ s.$ *if* $r = s$ *then* $\lfloor r \rfloor$ *else* $\Diamond$›
  ‹$\lambda s\ r.$ *if* $s = r$ *then* $\lfloor s \rfloor$ *else* $\Diamond$› *id id*
  **by** *unfold-locales* (*auto split*: *if-split-asm*)

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

### 10.1.2 Product Type

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

‹λr s. ⌊(r, s)⌋› ‹λs r. ⌊(s, r)⌋› *prod.swap prod.swap*
**by** *unfold-locales* (*auto split*: *if-split-asm*)

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

### 10.1.3 List Type

#### Pair

**interpretation** $Sync_{Pairlist}$ : $Sync_{ptick}$-*comm-locale*
‹λr s. ⌊[r, s]⌋› ‹λs r. ⌊[s, r]⌋›
‹λrs. [rs ! *Suc 0*, rs ! *0*]› ‹λrs. [rs ! *Suc 0*, rs ! *0*]›
**by** *unfold-locales* (*auto intro*: *inj-onI*)

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

#### Right List

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

**interpretation** $Sync_{Rlist}$ : $Sync_{ptick}$-*comm-locale*
‹λr s. ⌊r # s⌋› ‹λs r. ⌊s @ [r]⌋›
‹*rotate1*› ‹λrs. *if rs* = [] *then* [] *else last rs # butlast rs*›
— *λrs. last rs # butlast rs* is not injective.
**by** *unfold-locales* (*auto intro*: *inj-onI*)

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

#### Left List

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

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

### Arbitrary Lists

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

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

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

**end**

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

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

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

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

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

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

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

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

**global-interpretation** $Sync_{Lists}$ : $Sync_{ptick}$-*comm-locale*
‹λr s. *if length r* = *lenL* ∧ *length s* = *lenR then* ⌊*r* @ *s*⌋ *else* ◊›
‹λs r. *if length s* = *lenR* ∧ *length r* = *lenL then* ⌊*s* @ *r*⌋ *else* ◊›
‹λrs. *drop lenL rs* @ *take lenL rs*›
‹λrs. *drop lenR rs* @ *take lenR rs*›
**by** *unfold-locales* (*auto split*: *if-split-asm*)

**end**

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

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

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

## 10.2 Associativities

### 10.2.1 Classical Version

**lemma** $Sync_{Classic}$-*assoc* :
‹*P* $[\![S]\!]_{\checkmark Classic}$ (*Q* $[\![S]\!]_{\checkmark Classic}$ *R*) = *P* $[\![S]\!]_{\checkmark Classic}$ *Q* $[\![S]\!]_{\checkmark Classic}$ *R*›
**proof** −
**let** *?f* = ‹λr s. *if r* = *s then* ⌊*r*⌋ *else* ◊›
**interpret** ∗ : $Sync_{ptick}$-*assoc-locale ?f ?f ?f ?f id id*
**by** (*unfold-locales*) (*auto split*: *if-split-asm*)
**show** *?thesis* **by** (*fact* ∗.$Sync_{ptick}$-*assoc*[*simplified Renaming-id*])
**qed**

### 10.2.2 Product Type

**lemma** $Sync_{Pair}$-*assoc* :

$\langle P\ [\![S]\!]_{\checkmark Pair}\ (Q\ [\![S]\!]_{\checkmark Pair}\ R) = RenamingTick\ (P\ [\![S]\!]_{\checkmark Pair}\ Q\ [\![S]\!]_{\checkmark Pair}\ R)\ (\lambda((r,\ s),\ t).\ (r,\ s,\ t))\rangle$
**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-locales auto*
**show** *?thesis* **by** (*fact* ∗.$Sync_{ptick}$-*assoc*)
**qed**

### 10.2.3 List Type

**lemma** $Sync_{Rlist}$-$Sync_{Pairlist}$-*assoc* :
$\langle P\ [\![S]\!]_{\checkmark Rlist}\ (Q\ [\![S]\!]_{\checkmark Pairlist}\ R) = (P\ [\![S]\!]_{\checkmark Pairlist}\ Q)\ [\![S]\!]_{\checkmark Llist}\ R\rangle$
**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-locales auto*
**show** *?thesis* **by** (*fact* ∗.$Sync_{ptick}$-*assoc*[*unfolded Renaming-id*])
**qed**

**lemma** $Sync_{Rlist}$-$Sync_{Llist}$-*assoc* :
$\langle P\ [\![S]\!]_{\checkmark Rlist}\ (Q\ [\![S]\!]_{\checkmark Llist}\ R) = (P\ [\![S]\!]_{\checkmark Rlist}\ Q)\ [\![S]\!]_{\checkmark Llist}\ R\rangle$
**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-locales auto*
**show** *?thesis* **by** (*fact* ∗.$Sync_{ptick}$-*assoc*[*unfolded Renaming-id*])
**qed**

**lemma** $Sync_{Rlist}$-$Sync_{ListslenL}$-*assoc* :
$\langle P\ [\![S]\!]_{\checkmark Rlist}\ (Q\ _{lenQ}[\![S]\!]_{\checkmark ListslenL}\ R) = (P\ [\![S]\!]_{\checkmark Rlist}\ Q)\ _{Suc\ lenQ}[\![S]\!]_{\checkmark ListslenL}\ R\rangle$
**proof** −
**interpret** ∗ : $Sync_{ptick}$-*assoc-locale*
$\langle\lambda r\ s.\ \lfloor r\ \#\ s\rfloor\rangle$
$\langle\lambda r\ s.$ *if length* $r = Suc\ lenQ$ *then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond\rangle$
$\langle\lambda r\ s.\ \lfloor r\ \#\ s\rfloor\rangle$
$\langle\lambda r\ s.$ *if length* $r = lenQ$ *then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond\rangle$ *id id*
**by** *unfold-locales* (*auto split*: *if-split-asm*)
**show** *?thesis* **by** (*fact* ∗.$Sync_{ptick}$-*assoc*[*unfolded Renaming-id*])
**qed**

**lemma** $Sync_{ListslenR}$-$Sync_{Llist}$-*assoc* :
$\langle P\ _{Suc\ lenQ}[\![S]\!]_{\checkmark ListslenR}\ (Q\ [\![S]\!]_{\checkmark Llist}\ R) = (P\ _{lenQ}[\![S]\!]_{\checkmark ListslenR}\ Q)\ [\![S]\!]_{\checkmark Llist}\ R\rangle$
**proof** −

**interpret** $*$ : $Sync_{ptick}$-*assoc-locale*
‹$\lambda r\ s.$ *if length* $s =$ *lenQ then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.\ \lfloor r$ @ $[s]\rfloor$›
‹$\lambda r\ s.$ *if length* $s =$ *Suc lenQ then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.\ \lfloor r$ @ $[s]\rfloor$› *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* :
‹$P$ $_{lenP}[\![S]\!]\!\checkmark_{lenQ}$ $+$ $_{lenR}$ ($Q$ $_{lenQ}[\![S]\!]\!\checkmark_{lenR}$ $R$) =
$P$ $_{lenP}[\![S]\!]\!\checkmark_{lenQ}$ $Q$ $_{lenP}$ $+$ $_{lenQ}[\![S]\!]\!\checkmark_{lenR}$ $R$›
**proof** −
**interpret** $*$ : $Sync_{ptick}$-*assoc-locale*
‹$\lambda r\ s.$ *if length* $r = lenP \wedge$ *length* $s = lenQ$ *then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.$ *if length* $r = lenP +$ *lenQ* $\wedge$ *length* $s = lenR$ *then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.$ *if length* $r = lenP \wedge$ *length* $s = lenQ +$ *lenR then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.$ *if length* $r = lenQ \wedge$ *length* $s = lenR$ *then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$› *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* :
‹$P$ $[\![S]\!]\!\checkmark_{Rlist}$ ($Q$ $[\![S]\!]\!\checkmark_{Rlist}$ $R$) = ($P$ $[\![S]\!]\!\checkmark_{Pairlist}$ $Q$) $_{Suc\ (Suc\ 0)}[\![S]\!]\!\checkmark_{ListslenL}$
$R$›
**proof** −
**interpret** $*$ : $Sync_{ptick}$-*assoc-locale*
‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$›
‹$\lambda r\ s.$ *if length* $r =$ *Suc* (*Suc* 0) *then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.\ \lfloor r$ # $s\rfloor$›
‹$\lambda r\ s.\ \lfloor r$ # $s\rfloor$› *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* :
‹$P$ $_{Suc\ (Suc\ 0)}[\![S]\!]\!\checkmark_{ListslenR}$ ($Q$ $[\![S]\!]\!\checkmark_{Pairlist}$ $R$) = ($P$ $[\![S]\!]\!\checkmark_{Llist}$ $Q$) $[\![S]\!]\!\checkmark_{Llist}$
$R$›
**proof** −
**interpret** $*$ : $Sync_{ptick}$-*assoc-locale*
‹$\lambda r\ s.\ \lfloor r$ @ $[s]\rfloor$›
‹$\lambda r\ s.\ \lfloor r$ @ $[s]\rfloor$›
‹$\lambda r\ s.$ *if length* $s =$ *Suc* (*Suc* 0) *then* $\lfloor r$ @ $s\rfloor$ *else* $\Diamond$›
‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$› *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** $Sync_{Classic}$*-is-Sync* : ‹$P$ $[\![A]\!]_{\checkmark Classic}$ $Q$ = $P$ $[\![A]\!]$ $Q$›
**proof** (*rule Process-eq-optimizedI*)
**show** ‹$t \in \mathcal{D}$ ($P$ $[\![A]\!]$ $Q$) $\Longrightarrow$ $t \in \mathcal{D}$ ($P$ $[\![A]\!]_{\checkmark Classic}$ $Q$)› **for** $t$
**by** (*simp add*: *D-Sync* $Sync_{Classic}$.*D-*$Sync_{ptick}{}'$
*flip*: $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}$)
(*metis setinterleaving-sym*)
**next**
**show** ‹$t \in \mathcal{D}$ ($P$ $[\![A]\!]_{\checkmark Classic}$ $Q$) $\Longrightarrow$ $t \in \mathcal{D}$ ($P$ $[\![A]\!]$ $Q$)› **for** $t$
**by** (*simp add*: *D-Sync* $Sync_{Classic}$.*D-*$Sync_{ptick}$
*flip*: $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}$)
(*metis setinterleaving-sym*)
**next**
**fix** $t$ $X$ **assume** ‹($t$, $X$) $\in \mathcal{F}$ ($P$ $[\![A]\!]$ $Q$)› ‹$t \notin \mathcal{D}$ ($P$ $[\![A]\!]$ $Q$)›
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : ‹(*t-P*, *X-P*) $\in \mathcal{F}$ $P$› ‹(*t-Q*, *X-Q*) $\in \mathcal{F}$ $Q$›
‹$t$ *setinterleaves* ((*t-P*, *t-Q*), *range tick* $\cup$ *ev* ‘ $A$)›
‹$X$ = (*X-P* $\cup$ *X-Q*) $\cap$ (*range tick* $\cup$ *ev* ‘ $A$) $\cup$ *X-P* $\cap$ *X-Q*›
**unfolding** *Sync-projs* **by** *blast*
**from** $*(4)$ **have** ‹$X \subseteq$ $super\text{-}ref\text{-}Sync_{ptick}$ ($\lambda r$ *s. if* $r = s$ *then* $\lfloor r \rfloor$ *else* $\Diamond$) *X-P A X-Q*›
**by** (*auto simp add*: $super\text{-}ref\text{-}Sync_{ptick}$*-def subset-iff*)
(*metis* $event_{ptick}$.*exhaust*)
**with** $*(1-3)$ **show** ‹($t$, $X$) $\in \mathcal{F}$ ($P$ $[\![A]\!]_{\checkmark Classic}$ $Q$)›
**by** (*auto simp add*: $Sync_{Classic}$.$F\text{-}Sync_{ptick}$ $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}$)
**next**
**fix** $t$ $X$ **assume** ‹($t$, $X$) $\in \mathcal{F}$ ($P$ $[\![A]\!]_{\checkmark Classic}$ $Q$)› ‹$t \notin \mathcal{D}$ ($P$ $[\![A]\!]_{\checkmark Classic}$ $Q$)›
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : ‹(*t-P*, *X-P*) $\in \mathcal{F}$ $P$› ‹(*t-Q*, *X-Q*) $\in \mathcal{F}$ $Q$›
‹$t$ $setinterleaves_{\checkmark \lambda r\ s.\ if\ r = s\ then\ \lfloor r \rfloor\ else\ \Diamond}$ ((*t-P*, *t-Q*), $A$)›
‹$X \subseteq$ $super\text{-}ref\text{-}Sync_{ptick}$ ($\lambda r$ *s. if* $r = s$ *then* $\lfloor r \rfloor$ *else* $\Diamond$) *X-P A X-Q*›
**unfolding** $Sync_{Classic}$.$Sync_{ptick}$*-projs* **by** *blast*
**from** $*(1-3)$ **have** ‹($t$, (*X-P* $\cup$ *X-Q*) $\cap$ (*range tick* $\cup$ *ev* ‘ $A$) $\cup$ *X-P* $\cap$ *X-Q*) $\in$
$\mathcal{F}$ ($P$ $[\![A]\!]$ $Q$)›
**by** (*simp add*: *F-Sync* $setinterleaves\text{-}is\text{-}setinterleaves_{ptick}$) *blast*
**moreover from** $*(4)$ **have** ‹$X \subseteq$ (*X-P* $\cup$ *X-Q*) $\cap$ (*range tick* $\cup$ *ev* ‘ $A$) $\cup$ *X-P*
$\cap$ *X-Q*›
**by** (*auto simp add*: $super\text{-}ref\text{-}Sync_{ptick}$*-def subset-iff split*: *if-split-asm*)
**ultimately show** ‹($t$, $X$) $\in \mathcal{F}$ ($P$ $[\![A]\!]$ $Q$)› **by** (*meson is-processT4*)
**qed**

### 10.3.2 Other Properties

**lemma** ‹$Sync_{Lists}$.$Sync_{ptick}$-locale-sym.$Sync_{ptick}$ lenL lenR Q A P = P $_{lenL}[\![A]\!]_{✓lenR}$ Q›
**by** (*simp add*: $Sync_{Lists}$.$Sync_{ptick}$-locale-sym.$Sync_{ptick}$-sym)

**corollary** *TickSwap-Sync*$_{Pair}$ [*simp*] : ‹*TickSwap* (P $[\![S]\!]_{✓Pair}$ Q) = Q $[\![S]\!]_{✓Pair}$ P›
**by** (*simp add*: $Sync_{Pair}$.$Sync_{ptick}$-*commute TickSwap-is-Renaming*)

**lemma** *TickSwap-is-Sync*$_{Pair}$-*iff* [*simp*] :
‹*TickSwap* P = Q $[\![S]\!]_{✓Pair}$ R ⟷ P = R $[\![S]\!]_{✓Pair}$ Q›
**by** (*simp add*: *TickSwap-eq-iff-eq-TickSwap*)

**corollary** *Sync*$_{Classic}$-*commute* : ‹P $[\![S]\!]_{✓Classic}$ Q = Q $[\![S]\!]_{✓Classic}$ P›
**by** (*fact* $Sync_{Classic}$.$Sync_{ptick}$-*commute*[*simplified*])

**lemma** ‹*RenamingTick* (P $_{lenL}[\![S]\!]_{✓lenR}$ Q) (λr-s. *drop lenL r-s* @ *take lenL r-s*) =
Q $_{lenR}[\![S]\!]_{✓lenL}$ P›
**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* ::
‹*nat* ⇒ ('*a*, '*r list*) $process_{ptick}$ ⇒ *bool*› (‹($length_{✓}$)-('(-'))›)
**where** ‹$length_{✓}$ *n*(P) ≡ ∀ *rs* ∈ ✓**s**(P). *length rs* = *n*›

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

**lemma** *is-ticks-lengthI* : ‹(⋀*rs*. *rs* ∈ ✓**s**(P) ⟹ *length rs* = *n*) ⟹ $length_{✓}$ *n*(P)›
**by** (*simp add*: *is-ticks-length-def*)

**lemma** *is-ticks-lengthD* : ‹$length_{✓}$ *n*(P) ⟹ *rs* ∈ ✓**s**(P) ⟹ *length rs* = *n*›
**by** (*simp add*: *is-ticks-length-def*)

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

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

**lemma** *nonempty-strict-ticks-of-imp-is-ticks-length-unique* :
‹$\checkmark\mathbf{s}(P) \neq \{\} \Longrightarrow length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, m(P) \Longrightarrow m = n$›
**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*] :
‹$length_{\checkmark}\, n(STOP)$› **by** (*simp add*: *empty-strict-ticks-of-imp-is-ticks-length*)

**lemma** *is-ticks-length-BOT* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(\bot)$› **by** (*simp add*: *empty-strict-ticks-of-imp-is-ticks-length*)

**lemma** *is-ticks-length-SKIP-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(SKIP\ rs) \longleftrightarrow length\ rs = n$›
**by** (*simp add*: *is-ticks-length-def*)

**lemma** *is-ticks-length-SKIPS-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark}\, n(SKIPS\ R) \longleftrightarrow (\forall\, rs \in R.\ length\ rs = n)$›
**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*]
:
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \sqcap Q)$›
**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*] :
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \mathbin{\Box} Q)$›
**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*] :
‹$length_{\checkmark}\, n(P) \Longrightarrow length_{\checkmark}\, n(Q) \Longrightarrow length_{\checkmark}\, n(P \rhd Q)$›
**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*] :
‹*length*$_{\checkmark}$ $n(P) \Longrightarrow$ *length*$_{\checkmark}$ $n(Q) \Longrightarrow$ *length*$_{\checkmark}$ $n(P \llbracket S \rrbracket Q)$›
**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*] :
‹*non-terminating* $P \vee$ *length*$_{\checkmark}$ $n(Q) \Longrightarrow$ *length*$_{\checkmark}$ $n(P ; Q)$›
**proof** (*elim disjE*)
**show** ‹*non-terminating* $P \Longrightarrow$ *length*$_{\checkmark}$ $n(P ; Q)$›
**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** ‹*length*$_{\checkmark}$ $n(Q) \Longrightarrow$ *length*$_{\checkmark}$ $n(P ; Q)$›
**by** (*auto simp add*: *is-ticks-length-def split*: *if-split-asm*)
**qed**

**lemma** *is-ticks-length-Hiding* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark}$ $n(P \setminus S)$› **if** ‹*length*$_{\checkmark}$ $n(P)$›
**proof** (*rule is-ticks-lengthI*)
**fix** *rs* **assume** ‹$rs \in \checkmark\mathbf{s}(P \setminus S)$›
**then obtain** $t\ t'$ **where** ‹$t = t' @ [\checkmark(rs)]$› ‹$t \in \mathcal{T}\ (P \setminus S)$› ‹$t \notin \mathcal{D}\ (P \setminus S)$›
**by** (*metis is-processT9 strict-ticks-of-memE*)
**from** *this*(*2*, *3*) **obtain** *u* **where** ‹$t =$ *trace-hide* $u$ ($ev$ ‘ $S$)› ‹$u \in \mathcal{T}\ P$›
**unfolding** *T-Hiding D-Hiding* **using** *F-T* **by** *fast*
**from** *this*(*1*) *this*(*2*)[*THEN T-imp-front-tickFree*] **obtain** $u'$ **where** ‹$u = u' @ [\checkmark(rs)]$›
**by** (*cases u rule*: *rev-cases*, *simp-all add*: ‹$t = t' @ [\checkmark(rs)]$› *split*: *if-split-asm*)
(*metis Hiding-tickFree front-tickFree-nonempty-append-imp list.distinct*(*1*) *non-tickFree-tick tickFree-append-iff*)
**from** ‹$t \notin \mathcal{D}\ (P \setminus S)$› *mem-D-imp-mem-D-Hiding*[*of u P S*]
**have** ‹$u \notin \mathcal{D}\ P$› **unfolding** ‹$t =$ *trace-hide* $u$ ($ev$ ‘ $S$)› **by** *blast*
**with** ‹$u \in \mathcal{T}\ P$› ‹$u = u' @ [\checkmark(rs)]$› **have** ‹$rs \in \checkmark\mathbf{s}(P)$›
**by** (*simp add*: *strict-ticks-of-memI*)
**with** *that* **show** ‹*length* $rs = n$› **by** (*simp add*: *is-ticks-lengthD*)
**qed**

**lemma** *is-ticks-length-Interrupt* [*is-ticks-length-intro*] :
‹*length*$_{\checkmark}$ $n(P) \Longrightarrow$ *length*$_{\checkmark}$ $n(Q) \Longrightarrow$ *length*$_{\checkmark}$ $n(P \triangle Q)$›
**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* :
‹$\checkmark\mathbf{s}(P \ \Theta\ a{\in}A.\ Q\ a) \subseteq \checkmark\mathbf{s}(P) \cup (\bigcup a{\in}A \cap \alpha(P).\ \checkmark\mathbf{s}(Q\ a))$›

**proof** (*rule subsetI*)
**fix** $r$ **assume** ‹$r \in \checkmark\mathbf{s}(P \ominus a{\in}A.\ Q\ a)$›
**then obtain** $t$ **where** ‹$t \mathbin{@} [\checkmark(r)] \in \mathcal{T}\ (P \ominus a{\in}A.\ Q\ a)$› ‹$t \mathbin{@} [\checkmark(r)] \notin \mathcal{D}\ (P \ominus a{\in}A.\ Q\ a)$›
**by** (*meson is-processT9 strict-ticks-of-memE*)
**then consider** ‹$t \mathbin{@} [\checkmark(r)] \in \mathcal{T}\ P$› ‹$t \mathbin{@} [\checkmark(r)] \notin \mathcal{D}\ P$›
| *t1 a t2* **where** ‹$t \mathbin{@} [\checkmark(r)] = t1 \mathbin{@} ev\ a \#\ t2$› ‹$t1 \mathbin{@} [ev\ a] \in \mathcal{T}\ P$›
‹$a \in A$› ‹$t2 \in \mathcal{T}\ (Q\ a)$› ‹$t2 \notin \mathcal{D}\ (Q\ a)$›
**by** (*simp add*: *Throw-projs*)
(*metis* (*no-types*, *lifting*) *append-T-imp-tickFree*
*front-tickFree-single is-processT9 not-Cons-self2*)
**thus** ‹$r \in \checkmark\mathbf{s}(P) \cup (\bigcup a{\in}A \cap \alpha(P).\ \checkmark\mathbf{s}(Q\ a))$›
**proof** *cases*
**show** ‹$t \mathbin{@} [\checkmark(r)] \in \mathcal{T}\ P \Longrightarrow t \mathbin{@} [\checkmark(r)] \notin \mathcal{D}\ P \Longrightarrow r \in \checkmark\mathbf{s}(P) \cup (\bigcup a{\in}A \cap \alpha(P).\ \checkmark\mathbf{s}(Q\ a))$›
**by** (*simp add*: *strict-ticks-of-memI*)
**next**
**show** ‹$[\![t \mathbin{@} [\checkmark(r)] = t1 \mathbin{@} ev\ a \#\ t2;\ t1 \mathbin{@} [ev\ a] \in \mathcal{T}\ P;\ a \in A;\ t2 \in \mathcal{T}\ (Q\ a);\ t2 \notin \mathcal{D}\ (Q\ a)]\!] \Longrightarrow r \in \checkmark\mathbf{s}(P) \cup (\bigcup a{\in}A \cap \alpha(P).\ \checkmark\mathbf{s}(Q\ a))$› **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*] :
‹$length_{\checkmark}{}_n(P \ominus a \in A.\ Q\ a)$›
**if** ‹$length_{\checkmark}{}_n(P)$› ‹$\bigwedge a.\ a \in \alpha(P) \Longrightarrow length_{\checkmark}{}_n(Q\ a)$›
**proof** −
**from** *that* **have** ‹$\forall\, rs{\in}\checkmark\mathbf{s}(P) \cup (\bigcup a{\in}A \cap \alpha(P).\ \checkmark\mathbf{s}(Q\ a)).\ length\ rs = n$›
**by** (*auto simp add*: *is-ticks-length-def*)
**with** *strict-ticks-Throw-subset* **show** ‹$length_{\checkmark}{}_n(P \ominus a \in A.\ Q\ a)$›
**unfolding** *is-ticks-length-def* **by** *fast*
**qed**

**lemma** *is-ticks-length-Renaming* [*is-ticks-length-intro*] :
‹$length_{\checkmark}{}_n(Renaming\ P\ f\ g)$ › **if** ‹$\bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow length\ (g\ r) = n$›
**proof** (*rule is-ticks-lengthI*)
**fix** *rs* **assume** ‹$rs \in \checkmark\mathbf{s}(Renaming\ P\ f\ g)$›
**then obtain** $t$ **where** ‹$t \mathbin{@} [\checkmark(rs)] \in \mathcal{T}\ (Renaming\ P\ f\ g)$›
‹$t \mathbin{@} [\checkmark(rs)] \notin \mathcal{D}\ (Renaming\ P\ f\ g)$›
**by** (*meson is-processT9 strict-ticks-of-memE*)
**then obtain** $u$ **where** $*$ : ‹$t \mathbin{@} [\checkmark(rs)] = map\ (map\text{-}event_{ptick}\ f\ g)\ u$› **and** ‹$u \in \mathcal{T}\ P$›
**by** (*auto simp add*: *Renaming-projs*)
**from** *this*(*1*) ‹$u \in \mathcal{T}\ P$› *append-T-imp-tickFree* **obtain** $u'\ r$
**where** ‹$rs = g\ r$› ‹$u = u' \mathbin{@} [\checkmark(r)]$› ‹$tF\ u'$›
**by** (*cases u rule*: *rev-cases*) (*auto simp add*: *tick-eq-map-event*$_{ptick}$*-iff*)
**from** $*$ ‹$t \mathbin{@} [\checkmark(rs)] \notin \mathcal{D}\ (Renaming\ P\ f\ g)$› *this*(*2*, *3*) *front-tickFree-Cons-iff*

**have** ‹$u' \notin \mathcal{D}\ P$› **by** (*auto simp add*: *D-Renaming*)
**moreover from** ‹$u \in \mathcal{T}\ P$› **have** ‹$u' @ [\checkmark(r)] \in \mathcal{T}\ P$›
**by** (*simp add*: ‹$u = u' @ [\checkmark(r)]$›)
**ultimately have** ‹$r \in \checkmark\mathbf{s}(P)$› **by** (*meson is-processT9 strict-ticks-of-memI*)
**with** *that* **have** ‹*length* $(g\ r) = n$› **by** *blast*
**thus** ‹*length rs* $= n$› **by** (*simp add*: ‹$rs = g\ r$›)
**qed**

**Architectural Operators** **lemma** *is-ticks-length-GlobalNdet* [*is-ticks-length-intro*]
:
‹$(\bigwedge a.\ a \in A \Longrightarrow length_{\checkmark} n(P\ a)) \Longrightarrow length_{\checkmark} n(\sqcap a \in A.\ P\ a)$›
**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*] :
‹$(\bigwedge a.\ a \in A \Longrightarrow length_{\checkmark} n(P\ a)) \Longrightarrow length_{\checkmark} n(\Box a \in A.\ P\ a)$›
**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*] :
‹$(\bigwedge m.\ m \in set\text{-}mset\ M \Longrightarrow length_{\checkmark} n(P\ m)) \Longrightarrow length_{\checkmark} n([\![S]\!]\ m \in\#\ M.\ P\ m)$›
**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*] :
‹$L \neq [] \Longrightarrow length_{\checkmark} n(P\ (last\ L)) \Longrightarrow length_{\checkmark} n(SEQ\ l \in@\ L.\ P\ l)$›
**by** (*induct L rule*: *rev-induct*)
(*simp-all add*: *is-ticks-length-Seq*)

**Communications** **lemma** *is-ticks-length-write0-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark} n(e \rightarrow P) \longleftrightarrow length_{\checkmark} n(P)$›
**by** (*simp add*: *is-ticks-length-def strict-ticks-of-write0*)

**lemma** *is-ticks-length-write-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark} n(c!e \rightarrow P) \longleftrightarrow length_{\checkmark} n(P)$›
**by** (*simp add*: *is-ticks-length-def strict-ticks-of-write*)

**lemma** *is-ticks-length-Mprefix-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark} n(\Box a \in A \rightarrow P\ a) = (\forall\, a \in A.\ length_{\checkmark} n(P\ a))$›
**by** (*auto simp add*: *is-ticks-length-def strict-ticks-of-Mprefix*)

**lemma** *is-ticks-length-read-iff* [*is-ticks-length-simp*] :
‹$length_{\checkmark} n(c?a{\in}A \rightarrow P\ a) = (\forall\, b \in c\ `\ A.\ length_{\checkmark} n(P\ (inv\text{-}into\ A\ c\ b)))$›
**by** (*simp add*: *read-def is-ticks-length-Mprefix-iff*)

**corollary** ‹*inj-on c A* $\Longrightarrow length_{\checkmark} n(c?a{\in}A \rightarrow P\ a) = (\forall\, a \in A.\ length_{\checkmark} n(P\ a))$›
**by** (*simp add*: *is-ticks-length-read-iff*)

**lemma** *is-ticks-length-Mndetprefix-iff* [*is-ticks-length-simp*] :

$\langle length_{\checkmark} n(\sqcap a \in A \to P\ a) = (\forall\, a \in A.\ length_{\checkmark} n(P\ a))\rangle$
**by** (*auto simp add*: *is-ticks-length-def strict-ticks-of-Mndetprefix*)

**lemma** *is-ticks-length-ndet-write-iff* [*is-ticks-length-simp*] :
$\langle length_{\checkmark} n(c!!a{\in}A \to P\ a) = (\forall\, b \in c\ `\ A.\ length_{\checkmark} n(P\ (inv\text{-}into\ A\ c\ b)))\rangle$
**by** (*simp add*: *ndet-write-def is-ticks-length-Mndetprefix-iff*)

**corollary** $\langle inj\text{-}on\ c\ A \Longrightarrow length_{\checkmark} n(c!!a{\in}A \to P\ a) = (\forall\, a \in A.\ length_{\checkmark} n(P\ a))\rangle$
**by** (*simp add*: *is-ticks-length-ndet-write-iff*)

**Generalizations** **lemma** *strict-ticks-of-Seq$_{ptick}$-subset* : $\langle \checkmark\mathbf{s}(P\ ;_{\checkmark}\ Q) \subseteq \bigcup \{\checkmark\mathbf{s}(Q\ r)\ |r.\ r \in \checkmark\mathbf{s}(P)\}\rangle$
**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* $\langle map\ (ev \circ of\text{-}ev)$ - @ -$\rangle$] *last-snoc*[*of* - $\langle\checkmark(-)\rangle$]
*butlast-snoc*[*of* $\langle map\ (ev \circ of\text{-}ev)$ - @ -$\rangle$] *butlast-snoc*[*of* - $\langle\checkmark(-)\rangle$]
*append.assoc*[*of* $\langle map\ (ev \circ of\text{-}ev)$ -$\rangle$ - $\langle$[-]$\rangle$] *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$_{ptick}$* :
$\langle P\ ;_{\checkmark}\ Q = RenamingTick\ P\ g\rangle$ **if** $\langle non\text{-}terminating\ P\rangle$
**proof** −
**from** $\langle non\text{-}terminating\ P\rangle$ **have** £ : $\langle \mathcal{D}\ P = \{\}\rangle$ $\langle t\ @\ [\checkmark(r)] \notin \mathcal{T}\ P\rangle$ **for** *t r*
**by** (*force simp add*: *non-terminating-is-right nonterminating-implies-div-free*)+
**show** $\langle P\ ;_{\checkmark}\ Q = RenamingTick\ P\ g\rangle$
**proof** (*rule Process-eq-optimizedI*)
**show** $\langle t \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q) \Longrightarrow t \in \mathcal{D}\ (RenamingTick\ P\ g)\rangle$
**and** $\langle t \in \mathcal{D}\ (RenamingTick\ P\ g) \Longrightarrow t \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q)\rangle$ **for** *t*
**by** (*simp-all add*: *Seq$_{ptick}$-projs Renaming-projs* £)
**next**
**fix** *t X* **assume** $\langle (t,\ X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q)\rangle$
**then obtain** *t′* **where** ∗ : $\langle t = map\ (ev \circ of\text{-}ev)\ t'\rangle$
$\langle (t',\ ref\text{-}Seq_{ptick}\ X) \in \mathcal{F}\ P\rangle$ $\langle tF\ t'\rangle$
**by** (*auto simp add*: *Seq$_{ptick}$-projs Renaming-projs* £)
**have** $ : $\langle t = map\ (map\text{-}event_{ptick}\ id\ g)\ t'\rangle$
**by** (*simp add*: ∗(*1*, *3*) *tickFree-map-map-event$_{ptick}$-is*)
**have** $$ : $\langle map\text{-}event_{ptick}\ id\ g\ -`\ X \subseteq ref\text{-}Seq_{ptick}\ X\rangle$
**by** (*simp add*: *subset-iff ref-Seq$_{ptick}$-def image-iff*)
(*metis Int-iff event$_{ptick}$.exhaust event$_{ptick}$.sel*(*1*) *event$_{ptick}$.simps*(*9*) *id-apply rangeI*)
**show** $\langle (t,\ X) \in \mathcal{F}\ (RenamingTick\ P\ g)\rangle$
**by** (*simp add*: *Renaming-projs*) (*metis* $ $$ ∗(*2*) *is-processT4*)
**next**
**fix** *t X* **assume** $\langle (t,\ X) \in \mathcal{F}\ (RenamingTick\ P\ g)\rangle$
**then obtain** *t′* **where** ∗ : $\langle t = map\ (map\text{-}event_{ptick}\ id\ g)\ t'\rangle$

‹$(t'$, *map-event*$_{ptick}$ *id* $g$ $-$‘ $X) \in \mathcal{F}\ P$›
**by** (*auto simp add*: *Renaming-projs* £)
**from** ∗(2) *F-T non-terminating-is-right* ‹*non-terminating* $P$› **have** ‹*tF* $t'$› **by** *blast*
**have** ‹$(t'$, *map-event*$_{ptick}$ *id* $g$ $-$‘ $X \cup$ *range tick*$) \in \mathcal{F}\ P$›
**by** (*rule is-processT5*[*OF* ∗(2)]) (*use* £(2) *F-T* **in** *blast*)
**moreover have** ‹*ref-Seq*$_{ptick}$ $X \subseteq$ *map-event*$_{ptick}$ *id* $g$ $-$‘ $X \cup$ *range tick*›
**by** (*auto simp add*: *ref-Seq*$_{ptick}$*-def*)
**ultimately have** ‹$(t'$, *ref-Seq*$_{ptick}$ $X) \in \mathcal{F}\ P$›
**by** (*metis is-processT4*)
**moreover have** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$›
**by** (*simp add*: ∗(1) ‹*tF* $t'$› *tickFree-map-map-event*$_{ptick}$*-is*)
**ultimately show** ‹$(t, X) \in \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$›
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs* ‹*tF* $t'$›)
**qed**
**qed**

**lemma** *is-ticks-length-Seq*$_{ptick}$ [*is-ticks-length-intro*] :
‹*non-terminating* $P \vee (\forall\, r{\in}\checkmark\mathbf{s}(P).\ length_{\checkmark}n(Q\ r)) \Longrightarrow length_{\checkmark}n(P\ ;_{\checkmark}\ Q)$›
**proof** (*elim disjE*)
**assume** ‹*non-terminating* $P$›
**hence** ‹$\checkmark\mathbf{s}(P) = \{\}$›
**by** (*metis* (*full-types*) *non-terminating-Seq strict-ticks-of-BOT*
*strict-ticks-of-Seq-subset subset-empty*)
**show** ‹*non-terminating* $P \Longrightarrow length_{\checkmark}n(P\ ;_{\checkmark}\ Q)$›
**by** (*subst non-terminating-Seq*$_{ptick}$, *assumption*)
(*rule is-ticks-length-Renaming*, *simp add*: *is-ticks-length-Renaming* ‹$\checkmark\mathbf{s}(P) = \{\}$›)
**next**
**from** *strict-ticks-of-Seq*$_{ptick}$*-subset*[*of* $P$ $Q$]
**show** ‹$\forall\, r{\in}\checkmark\mathbf{s}(P).\ length_{\checkmark}n(Q\ r) \Longrightarrow length_{\checkmark}n(P\ ;_{\checkmark}\ Q)$›
**by** (*auto simp add*: *is-ticks-length-def*)
**qed**

**lemma** *is-ticks-length-Sync*$_{ptick}$ :
‹*length*$_{\checkmark}n$(*Sync*$_{ptick}$*-locale.Sync*$_{ptick}$ *tick-join* $P$ $A$ $Q$)›
— We cannot work directly inside the locale since in this context the types of ticks $'t$ cannot be set to $'r$ *list*.
**if** ‹*Sync*$_{ptick}$*-locale tick-join*›
**and** ‹$\bigwedge r\ s.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow s \in \checkmark\mathbf{s}(Q) \Longrightarrow$
*case tick-join r s of* $\Diamond \Rightarrow$ *True* | $\lfloor r\text{-}s \rfloor \Rightarrow$ *length r-s* = $n$›
**proof** −
**interpret** *Sync*$_{ptick}$*-locale tick-join*
**by** (*fact* ‹*Sync*$_{ptick}$*-locale tick-join*›)
**show** ‹*length*$_{\checkmark}n(P\ [\![A]\!]_{\checkmark}\ Q)$›

**proof** (*rule is-ticks-lengthI*)
**fix** *rs* **assume** ‹*rs* ∈ ✓**s**(*P* ⟦*A*⟧✓ *Q*)›
**then obtain** *t* **where** ‹*t* @ [✓(*rs*)] ∈ $\mathcal{T}$ (*P* ⟦*A*⟧✓ *Q*)› ‹*t* @ [✓(*rs*)] ∉ $\mathcal{D}$ (*P* ⟦*A*⟧✓ *Q*)›
**by** (*meson is-processT9 strict-ticks-of-memE*)
**then obtain** *t-P t-Q* **where** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*›
**and** ∗ : ‹*t* @ [✓(*rs*)] *setinterleaves*✓*tick-join* ((*t-P*, *t-Q*), *A*)›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**with** ‹*t* @ [✓(*rs*)] ∉ $\mathcal{D}$ (*P* ⟦*A*⟧✓ *Q*)› **have** ‹*t-P* ∉ $\mathcal{D}$ *P*› ‹*t-Q* ∉ $\mathcal{D}$ *Q*›
**by** (*simp add*: *D-Sync*$_{ptick}$′, *use front-tickFree-Nil* **in** *blast*)+
**from** ∗ **obtain** *r s t-P*′ *t-Q*′
**where** ‹*tick-join r s* = ⌊*rs*⌋› ‹*t-P* = *t-P*′ @ [✓(*r*)]› ‹*t-Q* = *t-Q*′ @ [✓(*s*)]›
**by** (*blast elim*: *snoc-tick-setinterleaves*$_{ptick}$*E*)
**from** *this*(*2*, *3*) ‹*t-P* ∉ $\mathcal{D}$ *P*› ‹*t-Q* ∉ $\mathcal{D}$ *Q*› ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*›
**have** ‹*r* ∈ ✓**s**(*P*)› ‹*s* ∈ ✓**s**(*Q*)› **by** (*metis strict-ticks-of-memI*)+
**from** *that*(*2*)[*OF this*, *unfolded* ‹*tick-join r s* = ⌊*rs*⌋›] **show** ‹*length rs* = *n*› **by** *simp*
**qed**
**qed**

**lemma** *is-ticks-length-One-RenamingTick-singl* [*is-ticks-length-simp*] :
‹*length*✓*Suc 0*(*RenamingTick P* (λ*r*. [*r*]))›
**by** (*simp add*: *is-ticks-length-Renaming*)

**lemma** *is-ticks-length-Two-Sync*$_{Pairlist}$ [*is-ticks-length-simp*] :
‹*length*✓*Suc* (*Suc 0*)(*P* ⟦*S*⟧✓$_{Pairlist}$ *Q*)›
**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*] :
‹*length*✓*n*(*Q*) ⟹ *length*✓*Suc n*(*P* ⟦*S*⟧✓$_{Rlist}$ *Q*)›
**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** ‹⋀*P Q n*. *length*✓*Suc n*(*P* ⟦*S*⟧✓$_{Rlist}$ *Q*) ⟹ *length*✓*n*(*Q*)›
**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*] :
‹*length*✓*n*(*P*) ⟹ *length*✓*Suc n*(*P* ⟦*S*⟧✓$_{Llist}$ *Q*)›
**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*$_{ListslenL}$ [*is-ticks-length-intro*] :

$\langle length_{\checkmark} m(Q) \Longrightarrow length_{\checkmark} n + m(P\ n[\![S]\!]_{\checkmark ListslenL}\ Q)\rangle$
**by** (*rule is-ticks-length-Sync*$_{ptick}$[*OF Sync*$_{ListslenL}$.*Sync*$_{ptick}$*-locale-axioms*])
(*simp add*: *is-ticks-lengthD*)

**lemma** *is-ticks-length-sum-Sync*$_{ListslenR}$ [*is-ticks-length-intro*] :
$\langle length_{\checkmark} n(P) \Longrightarrow length_{\checkmark} n + m(P\ m[\![S]\!]_{\checkmark ListslenR}\ Q)\rangle$
**by** (*rule is-ticks-length-Sync*$_{ptick}$
[*OF Sync*$_{ListslenL}$.*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*] :
$\langle length_{\checkmark} n + m(P\ n[\![S]\!]_{\checkmark} m\ Q)\rangle$
**by** (*rule is-ticks-length-Sync*$_{ptick}$[*OF Sync*$_{Lists}$.*Sync*$_{ptick}$*-locale-axioms*]) *simp*

### 10.4.2 Conversions

**lemma** *Sync*$_{Pairlist}$*-to-Sync*$_{Rlist}$ :
$\langle P\ [\![S]\!]_{\checkmark Pairlist}\ Q = P\ [\![S]\!]_{\checkmark Rlist}\ RenamingTick\ Q\ (\lambda s.\ [s])\rangle$
**by** (*rule Sync*$_{Rlist}$.*inj-RenamingTick-Sync*$_{ptick}$*-inj-RenamingTick*
[*of id* ‹$\lambda s.\ [s]$›, *simplified*, *symmetric*])
(*auto intro*: *inj-onI*)

**lemma** *Sync*$_{Pairlist}$*-to-Sync*$_{Llist}$ :
$\langle P\ [\![S]\!]_{\checkmark Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ [\![S]\!]_{\checkmark Llist}\ Q\rangle$
**by** (*rule Sync*$_{Rlist}$.*Sync*$_{ptick}$*-comm-locale-sym.inj-RenamingTick-Sync*$_{ptick}$*-inj-RenamingTick*
[*of* ‹$\lambda r.\ [r]$› *id*, *simplified*, *symmetric*])
(*auto intro*: *inj-onI*)

**lemma** *Sync*$_{Rlist}$*-to-Sync*$_{ListslenL}$ :
$\langle P\ [\![S]\!]_{\checkmark Rlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ _{Suc\ 0}[\![S]\!]_{\checkmark ListslenL}\ Q\rangle$
**by** (*rule Sync*$_{ListslenL}$.*inj-RenamingTick-Sync*$_{ptick}$*-inj-RenamingTick*
[*of* ‹$\lambda r.\ [r]$› *id* ‹*Suc 0*›, *simplified*, *symmetric*])
(*auto intro*: *inj-onI*)

**lemma** *Sync*$_{Llist}$*-to-Sync*$_{ListslenR}$ :
$\langle P\ [\![S]\!]_{\checkmark Llist}\ Q = P\ _{Suc\ 0}[\![S]\!]_{\checkmark ListslenR}\ RenamingTick\ Q\ (\lambda s.\ [s])\rangle$
**by** (*rule Sync*$_{ListslenL}$.*Sync*$_{ptick}$*-comm-locale-sym.inj-RenamingTick-Sync*$_{ptick}$*-inj-RenamingTick*
[*of id* ‹$\lambda s.\ [s]$› ‹*Suc 0*›, *simplified*, *symmetric*])
(*auto intro*: *inj-onI*)

**lemma** *Sync*$_{ListslenL}$*-to-Sync*$_{Lists}$ :
$\langle length_{\checkmark} m(Q) \Longrightarrow P\ n[\![S]\!]_{\checkmark ListslenL}\ Q = P\ n[\![S]\!]_{\checkmark} m\ Q\rangle$
**by** (*auto intro*!: *Sync*$_{Lists}$.*Sync*$_{ptick}$*-same-tick-join-on-strict-ticks-of*
*Sync*$_{ListslenL}$.*Sync*$_{ptick}$*-locale-axioms*
*dest*: *is-ticks-lengthD*)

**lemma** *Sync*$_{ListslenR}$*-to-Sync*$_{Lists}$ :

‹$length_{\checkmark n}(P) \Longrightarrow P\ {}_m[\![S]\!]_{\checkmark ListslenR}\ Q = P\ {}_n[\![S]\!]_{\checkmark m}\ Q$›
**by** (*auto intro*!: $Sync_{Lists}$.$Sync_{ptick}$-*same-tick-join-on-strict-ticks-of* $Sync_{ListslenL}$.$Sync_{ptick}$-*comm-locale-sym*.$Sync_{ptick}$-*locale-axioms* *dest*: *is-ticks-lengthD*)

**corollary** $Sync_{ListslenL}$-*is-*$Sync_{ListslenR}$ :
‹$length_{\checkmark n}(P) \Longrightarrow length_{\checkmark m}(Q) \Longrightarrow P\ {}_n[\![S]\!]_{\checkmark ListslenL}\ Q = P\ {}_m[\![S]\!]_{\checkmark ListslenR}\ Q$›
**by** (*simp add*: $Sync_{ListslenL}$-*to-*$Sync_{Lists}$ $Sync_{ListslenR}$-*to-*$Sync_{Lists}$)

**corollary** $Sync_{Pairlist}$-*to-*$Sync_{ListslenL}$ :
‹$P\ [\![S]\!]_{\checkmark Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ {}_{Suc\ 0}[\![S]\!]_{\checkmark ListslenL}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
**by** (*simp add*: $Sync_{Pairlist}$-*to-*$Sync_{Rlist}$ $Sync_{Rlist}$-*to-*$Sync_{ListslenL}$)

**corollary** $Sync_{Pairlist}$-*to-*$Sync_{ListslenR}$ :
‹$P\ [\![S]\!]_{\checkmark Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ {}_{Suc\ 0}[\![S]\!]_{\checkmark ListslenR}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
**by** (*simp add*: $Sync_{Llist}$-*to-*$Sync_{ListslenR}$ $Sync_{Pairlist}$-*to-*$Sync_{Llist}$)

**corollary** $Sync_{Rlist}$-*to-*$Sync_{Lists}$ :
‹$length_{\checkmark m}(Q) \Longrightarrow P\ [\![S]\!]_{\checkmark Rlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ {}_{Suc\ 0}[\![S]\!]_{\checkmark m}\ Q$›
**by** (*simp add*: $Sync_{ListslenL}$-*to-*$Sync_{Lists}$ $Sync_{Rlist}$-*to-*$Sync_{ListslenL}$)

**corollary** $Sync_{Llist}$-*to-*$Sync_{Lists}$ :
‹$length_{\checkmark n}(P) \Longrightarrow P\ [\![S]\!]_{\checkmark Llist}\ Q = P\ {}_n[\![S]\!]_{\checkmark Suc\ 0}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
**by** (*simp add*: $Sync_{ListslenR}$-*to-*$Sync_{Lists}$ $Sync_{Llist}$-*to-*$Sync_{ListslenR}$)

**corollary** $Sync_{Pairlist}$-*to-*$Sync_{Lists}$ :
‹$P\ [\![S]\!]_{\checkmark Pairlist}\ Q = RenamingTick\ P\ (\lambda r.\ [r])\ {}_{Suc\ 0}[\![S]\!]_{\checkmark Suc\ 0}\ RenamingTick\ Q\ (\lambda s.\ [s])$›
**by** (*simp add*: $Sync_{Llist}$-*to-*$Sync_{Lists}$ $Sync_{Pairlist}$-*to-*$Sync_{Llist}$ *is-ticks-length-One-RenamingTick-singl*)

**lemma** $Sync_{Pair}$-*to-*$Sync_{Pairlist}$ :
‹$RenamingTick\ (P\ [\![S]\!]_{\checkmark Pair}\ Q)\ (\lambda(r,\ s).\ [r,\ s]) = P\ [\![S]\!]_{\checkmark Pairlist}\ Q$›
**by** (*rule* $Sync_{Pair}$.*inj-on-RenamingTick-*$Sync_{ptick}$ [*of* ‹$\lambda(r,\ s).\ [r,\ s]$›, *simplified*])
(*auto intro*: *inj-onI*)

**lemma** $Sync_{Pairlist}$-*to-*$Sync_{Pair}$ :
‹$RenamingTick\ (P\ [\![S]\!]_{\checkmark Pairlist}\ Q)\ (\lambda rs.\ (rs\ !\ 0,\ rs\ !\ Suc\ 0)) = P\ [\![S]\!]_{\checkmark Pair}\ Q$›
**by** (*rule* $Sync_{Pairlist}$.*inj-on-RenamingTick-*$Sync_{ptick}$ [*of* ‹$\lambda rs.\ (rs\ !\ 0,\ rs\ !\ Suc\ 0)$›, *simplified*])
(*auto intro*: *inj-onI*)

**lemma** *Sync$_{Pair}$-to-Sync$_{Rlist}$* :
‹*RenamingTick* (*P* ⟦*S*⟧$_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* # *s*) = *P* ⟦*S*⟧$_{✓Rlist}$ *Q*›
**by** (*rule Sync$_{Pair}$.inj-on-RenamingTick-Sync$_{ptick}$*
[*of* ‹λ(*r*, *s*). *r* # *s*›, *simplified*])
(*auto intro*: *inj-onI*)

**lemma** *Sync$_{Pair}$-to-Sync$_{Llist}$* :
‹*RenamingTick* (*P* ⟦*S*⟧$_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ [*s*]) = *P* ⟦*S*⟧$_{✓Llist}$ *Q*›
**by** (*rule Sync$_{Pair}$.inj-on-RenamingTick-Sync$_{ptick}$*
[*of* ‹λ(*r*, *s*). *r* @ [*s*]›, *simplified*])
(*auto intro*: *inj-onI*)

**lemma** *Sync$_{Pair}$-to-Sync$_{ListslenL}$* :
‹*RenamingTick* (*P* ⟦*S*⟧$_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ *s*) = *P* $_n$⟦*S*⟧$_{✓ListslenL}$ *Q*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*length*$_{✓n}$(*P*)›
**proof** −
**let** *?g* = ‹λ*rs*. (*take n rs*, *drop n rs*)›
**let** *?g′* = ‹λ(*r*, *s*). *r* @ *s*›
**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* ⟦*S*⟧$_{✓Pair}$ *Q*) *?g′*) *?g* = *P* ⟦*S*⟧$_{✓Pair}$ *Q*›
**proof** (*fold RenamingTick-comp*, *subst* (*2*) *RenamingTick-id*[*of* ‹*P* ⟦*S*⟧$_{✓Pair}$ *Q*›, *symmetric*])
**show** ‹*?RT* (*P* ⟦*S*⟧$_{✓Pair}$ *Q*) (*?g* ∘ *?g′*) = *?RT* (*P* ⟦*S*⟧$_{✓Pair}$ *Q*) *id*›
**proof** (*rule RenamingTick-is-restrictable-on-strict-ticks-of*)
**from** *that* **show** ‹*rs* ∈ ✓**s**(*P* ⟦*S*⟧$_{✓Pair}$ *Q*) ⟹ (*?g* ∘ (λ(*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* ⟦*S*⟧$_{✓Pair}$ *Q* =
*Sync$_{ptick}$-locale.Sync$_{ptick}$*
(λ*r s*. *case if length r* = *n then* ⌊*r* @ *s*⌋ *else* ◊
*of* ◊ ⇒ ◊ | ⌊*rs*⌋ ⇒ ⌊*?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-mp*[*OF* $Sync_{Pair}$.*strict-ticks-of-*$Sync_{ptick}$-*subset*] *is-ticks-lengthD*)
**also from** ‹*length*$_{✓n}(P)$› **have** ‹*?RT* (*?RT ?rhs ?g*) *?g'* = *?rhs*›
**by** (*fold RenamingTick-comp*, *subst* (*2*) *RenamingTick-id*[*of ?rhs*, *symmetric*])
(*auto simp del*: *RenamingTick-id intro*!: *RenamingTick-is-restrictable-on-strict-ticks-of*
*dest*!: *set-mp*[*OF* $Sync_{Pair}$.*strict-ticks-of-*$Sync_{ptick}$-*subset*] *is-ticks-lengthD*)
**finally show** ‹*?lhs* = *?rhs*› **.**
**qed**

**corollary** $Sync_{Pair}$-*to-*$Sync_{ListslenR}$ :
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ *s*) = *P* $_n[\![S]\!]_{✓ListslenR}$ *Q*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*length*$_{✓n}(Q)$›
**proof** −
**let** *?RT* = *RenamingTick*
**have** ‹*?RT* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*x*, *y*). *x* @ *y*) =
*?RT* (*Q* $[\![S]\!]_{✓Pair}$ *P*) ((λ(*x*, *y*). *x* @ *y*) ∘ *prod.swap*)›
**by** (*simp add*: *RenamingTick-comp subst* $Sync_{Pair}$.$Sync_{ptick}$-*commute*)
**also from** ‹*length*$_{✓n}(Q)$›
**have** ‹... = *?RT* (*Q* $[\![S]\!]_{✓Pair}$ *P*) ((λ*rs*. *drop n rs* @ *take n rs*) ∘ (λ(*x*, *y*). *x* @ *y*))›
**by** (*auto intro*!: *RenamingTick-is-restrictable-on-strict-ticks-of*
*dest*!: *set-mp*[*OF* $Sync_{Pair}$.*strict-ticks-of-*$Sync_{ptick}$-*subset*] *is-ticks-lengthD*)
**also have** ‹... = *?RT* (*Q* $_n[\![S]\!]_{✓ListslenL}$ *P*) (λ*rs*. *drop n rs* @ *take n rs*)›
**by** (*simp add*: *RenamingTick-comp* $Sync_{Pair}$-*to-*$Sync_{ListslenL}$[*OF* ‹*length*$_{✓n}(Q)$›])
**also have** ‹... = *P* $_n[\![S]\!]_{✓ListslenR}$ *Q*›
**by** (*fact* $Sync_{ListslenL}$.$Sync_{ptick}$-*commute*)
**finally show** ‹*?lhs* = *?rhs*› **.**
**qed**

**corollary** $Sync_{Pair}$-*to-*$Sync_{Lists}$ :
‹*RenamingTick* (*P* $[\![S]\!]_{✓Pair}$ *Q*) (λ(*r*, *s*). *r* @ *s*) = *P* $_n[\![S]\!]_{✓m}$ *Q*›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*length*$_{✓n}(P)$› **and** ‹*length*$_{✓m}(Q)$›
**by** (*subst* $Sync_{Pair}$-*to-*$Sync_{ListslenL}$[*OF* ‹*length*$_{✓n}(P)$›])
(*rule* $Sync_{ListslenL}$-*to-*$Sync_{Lists}$[*OF* ‹*length*$_{✓m}(Q)$›])

## 10.5 First Laws

**corollary** $Inter_{Classic}$-*STOP* [*simp*] :
‹*P* $|||_{✓Classic}$ *STOP* = *P* **;** *STOP*›
**by** (*simp add*: $Sync_{Classic}$.$Inter_{ptick}$-*STOP*[*of P id*])

**corollary** $Inter_{Pair}$-*STOP* :
‹*P* $|||_{✓Pair}$ *STOP* = *RenamingTick* (*P* **;** *STOP*) (λ*r*. (*r*, *g r*))›
**by** (*simp add*: $Sync_{Pair}$.$Inter_{ptick}$-*STOP*[*of P g*])

**corollary** $Inter_{Pairlist}$-*STOP* :
‹*P* $|||_{✓Pairlist}$ *STOP* = *RenamingTick* (*P* **;** *STOP*) (λ*r*. [*r*, *g r*])›
**by** (*simp add*: $Sync_{Pairlist}$.$Inter_{ptick}$-*STOP*[*of P g*])

**corollary** $Inter_{Rlist}$-$STOP$ :
‹$P$ $|||_{\checkmark Rlist}$ $STOP$ = $RenamingTick$ ($P$ ; $STOP$) ($\lambda r.$ $r$ # $g$ $r$)›
**by** (*simp add*: $Sync_{Rlist}$.$Inter_{ptick}$-$STOP$[*of* $P$ $g$])

**corollary** $Inter_{Llist}$-$STOP$ :
‹$P$ $|||_{\checkmark Llist}$ $STOP$ = $RenamingTick$ ($P$ ; $STOP$) ($\lambda r.$ $r$ @ [$g$ $r$])›
**by** (*simp add*: $Sync_{Rlist}$.$Sync_{ptick}$-*comm-locale-sym*.$Inter_{ptick}$-$STOP$[*of* $P$ $g$])

**corollary** $Inter_{ListslenL}$-$STOP$ :
‹$P$ $n|||_{\checkmark ListslenL}$ $STOP$ =
$RenamingTick$ ($P$ ; $STOP$) ($\lambda r.$ *if length* $r$ = $n$ *then* $r$ @ $g$ $r$ *else undefined*)›
**by** (*auto simp add*: $Sync_{ListslenL}$.$Inter_{ptick}$-$STOP$[*of* $n$ $P$ $g$] *option.the-def*
*intro*!: *arg-cong*[**where** $f$ = ‹$RenamingTick$ ($P$ ; $STOP$)›])

**corollary** $Inter_{ListslenR}$-$STOP$ :
‹$P$ $n|||_{\checkmark ListslenR}$ $STOP$ =
$RenamingTick$ ($P$ ; $STOP$) ($\lambda r.$ *if length* ($g$ $r$) = $n$ *then* $r$ @ $g$ $r$ *else undefined*)›
**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** $f$ = ‹$RenamingTick$ ($P$ ; $STOP$)›])

**corollary** $Inter_{Lists}$-$STOP$ :
‹$P$ $n|||_{\checkmark m}$ $STOP$ =
$RenamingTick$ ($P$ ; $STOP$) ($\lambda r.$ *if length* $r$ = $n$ ∧ *length* ($g$ $r$) = $m$ *then* $r$ @ $g$ $r$ *else undefined*)›
**by** (*auto simp add*: $Sync_{Lists}$.$Inter_{ptick}$-$STOP$[*of* $n$ $m$ $P$ $g$] *option.the-def*
*intro*!: *arg-cong*[**where** $f$ = ‹$RenamingTick$ ($P$ ; $STOP$)›])

**corollary** $STOP$-$Inter_{Classic}$ [*simp*] :
‹$STOP$ $|||_{\checkmark Classic}$ $Q$ = $Q$ ; $STOP$›
**by** (*simp add*: $Sync_{Classic}$.$STOP$-$Inter_{ptick}$[*of* $Q$ *id*])

**corollary** $STOP$-$Inter_{Pair}$ :
‹$STOP$ $|||_{\checkmark Pair}$ $Q$ = $RenamingTick$ ($Q$ ; $STOP$) ($\lambda s.$ ($g$ $s$, $s$))›
**by** (*simp add*: $Sync_{Pair}$.$STOP$-$Inter_{ptick}$[*of* $Q$ $g$])

**corollary** $STOP$-$Inter_{Pairlist}$ :
‹$STOP$ $|||_{\checkmark Pairlist}$ $Q$ = $RenamingTick$ ($Q$ ; $STOP$) ($\lambda s.$ [$g$ $s$, $s$])›
**by** (*simp add*: $Sync_{Pairlist}$.$STOP$-$Inter_{ptick}$[*of* $Q$ $g$])

**corollary** $STOP$-$Inter_{Rlist}$ :
‹$STOP$ $|||_{\checkmark Rlist}$ $Q$ = $RenamingTick$ ($Q$ ; $STOP$) ($\lambda s.$ $g$ $s$ # $s$)›
**by** (*simp add*: $Sync_{Rlist}$.$STOP$-$Inter_{ptick}$[*of* $Q$ $g$])

**corollary** $STOP$-$Inter_{Llist}$ :
‹$STOP$ $|||_{\checkmark Llist}$ $Q$ = $RenamingTick$ ($Q$ ; $STOP$) ($\lambda s.$ $g$ $s$ @ [$s$])›
**by** (*simp add*: $Sync_{Rlist}$.$Sync_{ptick}$-*comm-locale-sym*.$STOP$-$Inter_{ptick}$[*of* $Q$ $g$])

**corollary** *STOP-Inter*$_{ListslenL}$ :
‹*STOP* $_n|||_{\checkmark ListslenL}$ *Q* =
*RenamingTick* (*Q* ; *STOP*) (λ*r*. *if length* (*g r*) = *n then g r* @ *r else undefined*)›
**by** (*auto simp add*: *Sync*$_{ListslenL}$.*STOP-Inter*$_{ptick}$[*of n Q g*] *option.the-def*
*intro*!: *arg-cong*[**where** *f* = ‹*RenamingTick* (*Q* ; *STOP*)›])

**corollary** *STOP-Inter*$_{ListslenR}$ :
‹*STOP* $_n|||_{\checkmark ListslenR}$ *Q* =
*RenamingTick* (*Q* ; *STOP*) (λ*r*. *if length r* = *n then g r* @ *r else undefined*)›
**by** (*auto simp add*: *Sync*$_{ListslenL}$.*Sync*$_{ptick}$*-comm-locale-sym.STOP-Inter*$_{ptick}$[*of n Q g*] *option.the-def*
*intro*!: *arg-cong*[**where** *f* = ‹*RenamingTick* (*Q* ; *STOP*)›])

**corollary** *STOP-Inter*$_{Lists}$ :
‹*STOP* $_n|||_{\checkmark m}$ *Q* =
*RenamingTick* (*Q* ; *STOP*) (λ*r*. *if length* (*g r*) = *n* ∧ *length r* = *m then g r* @ *r else undefined*)›
**by** (*auto simp add*: *Sync*$_{Lists}$.*STOP-Inter*$_{ptick}$[*of n m Q g*] *option.the-def*
*intro*!: *arg-cong*[**where** *f* = ‹*RenamingTick* (*Q* ; *STOP*)›])

**corollary** *SKIP-Sync*$_{Classic}$*-SKIP* :
‹*SKIP r* ⟦*A*⟧$_{\checkmark Classic}$ *SKIP s* =
(*if r* = *s then SKIP r else STOP*)› **by** *simp*

**corollary** *SKIP-Sync*$_{Pair}$*-SKIP* :
‹*SKIP r* ⟦*A*⟧$_{\checkmark Pair}$ *SKIP s* = *SKIP* (*r*, *s*)› **by** *simp*

**corollary** *SKIP-Sync*$_{Pairlist}$*-SKIP* :
‹*SKIP r* ⟦*A*⟧$_{\checkmark Pairlist}$ *SKIP s* = *SKIP* [*r*, *s*]› **by** *simp*

**corollary** *SKIP-Sync*$_{Rlist}$*-SKIP* :
‹*SKIP r* ⟦*A*⟧$_{\checkmark Rlist}$ *SKIP s* = *SKIP* (*r* # *s*)› **by** *simp*

**corollary** *SKIP-Sync*$_{Llist}$*-SKIP* :
‹*SKIP r* ⟦*A*⟧$_{\checkmark Llist}$ *SKIP s* = *SKIP* (*r* @ [*s*])› **by** *simp*

**corollary** *SKIP-Sync*$_{ListslenL}$*-SKIP* :
‹*SKIP r* $_n$⟦*A*⟧$_{\checkmark ListslenL}$ *SKIP s* =
(*if length r* = *n then SKIP* (*r* @ *s*) *else STOP*)› **by** *simp*

**corollary** *SKIP-Sync*$_{ListslenR}$*-SKIP* :
‹*SKIP r* $_n$⟦*A*⟧$_{\checkmark ListslenR}$ *SKIP s* =
(*if length s* = *n then SKIP* (*r* @ *s*) *else STOP*)› **by** *simp*

**corollary** *SKIP-Sync*$_{Lists}$*-SKIP* :
‹*SKIP r* $_n$⟦*A*⟧$_{\checkmark m}$ *SKIP s* =

(*if length r = n ∧ length s = m then SKIP (r @ s) else STOP*)› **by** *simp*

## 10.6 Operational Laws

### 10.6.1 Classical Version

**locale** *After-Sync*$_{Classic}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹*λr s. if r = s then* ⌊*r*⌋ *else* ◊›
**begin**

— Just checking...
**lemma** ‹*Sync*$_{ptick}$ *P S Q* = *P* ⟦*S*⟧✓$_{Classic}$ *Q*› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync*$_{Classic}$*-locale* =
*AfterExt-Sync*$_{ptick}$*-locale* ‹*λr s. if r = s then* ⌊*r*⌋ *else* ◊›

**sublocale** *AfterExt-Sync*$_{Classic}$*-locale* ⊆ *After-Sync*$_{Classic}$*-locale*
**by** *unfold-locales*

**locale** *OpSemTransitions-Sync*$_{Classic}$*-locale* =
*OpSemTransitions-Sync*$_{ptick}$*-locale* ‹*λr s. if r = s then* ⌊*r*⌋ *else* ◊›

**sublocale** *OpSemTransitions-Sync*$_{Classic}$*-locale* ⊆ *AfterExt-Sync*$_{Classic}$*-locale*
**by** *unfold-locales*

### 10.6.2 Product Type

**locale** *After-Sync*$_{Pair}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹*λr s.* ⌊(*r*, *s*)⌋›
**begin**

— Just checking...
**lemma** ‹*Sync*$_{ptick}$ *P S Q* = *P* ⟦*S*⟧✓$_{Pair}$ *Q*› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync*$_{Pair}$*-locale* =
*AfterExt-Sync*$_{ptick}$*-locale* ‹*λr s.* ⌊(*r*, *s*)⌋›

**sublocale** *AfterExt-Sync*$_{Pair}$*-locale* ⊆ *After-Sync*$_{Pair}$*-locale*
**by** *unfold-locales*

**locale** *OpSemTransitions-Sync*$_{Pair}$*-locale* =
*OpSemTransitions-Sync*$_{ptick}$*-locale* ‹*λr s.* ⌊(*r*, *s*)⌋›

**sublocale** *OpSemTransitions-Sync*$_{Pair}$*-locale* ⊆ *AfterExt-Sync*$_{Pair}$*-locale*
**by** *unfold-locales*

### 10.6.3 List Type

#### Pair

**locale** *After-Sync*$_{Pairlist}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$›
**begin**

— Just checking...
**lemma** ‹*Sync*$_{ptick}$ $P\ S\ Q = P\ [\![S]\!]_{\checkmark Pairlist}\ Q$› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync*$_{Pairlist}$*-locale* =
*AfterExt-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$›

**sublocale** *AfterExt-Sync*$_{Pairlist}$*-locale* ⊆ *After-Sync*$_{Pairlist}$*-locale*
**by** *unfold-locales*

**locale** *OpSemTransitions-Sync*$_{Pairlist}$*-locale* =
*OpSemTransitions-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor[r,\ s]\rfloor$›

**sublocale** *OpSemTransitions-Sync*$_{Pairlist}$*-locale* ⊆ *AfterExt-Sync*$_{Pairlist}$*-locale*
**by** *unfold-locales*

#### Right List

**locale** *After-Sync*$_{Rlist}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor r\ \#\ s\rfloor$›
**begin**

— Just checking...
**lemma** ‹*Sync*$_{ptick}$ $P\ S\ Q = P\ [\![S]\!]_{\checkmark Rlist}\ Q$› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync*$_{Rlist}$*-locale* =
*AfterExt-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor r\ \#\ s\rfloor$›

**sublocale** *AfterExt-Sync*$_{Rlist}$*-locale* ⊆ *After-Sync*$_{Rlist}$*-locale*
**by** *unfold-locales*

**locale** *OpSemTransitions-Sync*$_{Rlist}$*-locale* =
*OpSemTransitions-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor r\ \#\ s\rfloor$›

**sublocale** *OpSemTransitions-Sync*$_{Rlist}$*-locale* ⊆ *AfterExt-Sync*$_{Rlist}$*-locale*
**by** *unfold-locales*

#### Left List

**locale** *After-Sync*$_{Llist}$*-locale* = *After-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.\ \lfloor r\ @\ [s]\rfloor$›
**begin**

— Just checking...
**lemma** ‹$Sync_{ptick}\ P\ S\ Q = P\ [\![S]\!]_{\checkmark Llist}\ Q$› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync$_{Llist}$-locale* =
  *AfterExt-Sync$_{ptick}$-locale* ‹$\lambda r\ s.\ \lfloor r\ @\ [s]\rfloor$›

**sublocale** *AfterExt-Sync$_{Llist}$-locale* ⊆ *After-Sync$_{Llist}$-locale*
  **by** *unfold-locales*

**locale** *OpSemTransitions-Sync$_{Llist}$-locale* =
  *OpSemTransitions-Sync$_{ptick}$-locale* ‹$\lambda r\ s.\ \lfloor r\ @\ [s]\rfloor$›

**sublocale** *OpSemTransitions-Sync$_{Llist}$-locale* ⊆ *AfterExt-Sync$_{Llist}$-locale*
  **by** *unfold-locales*

### Arbitrary Lists

**Control on left side** **locale** *After-Sync$_{ListslenL}$-locale* =
  *After-Sync$_{ptick}$-locale* ‹$\lambda r\ s.$ *if length* $r =$ *lenL then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
  **for** *lenL* :: *nat*
**begin**

— Just checking...
**lemma** ‹$Sync_{ptick}\ P\ S\ Q = P\ _{lenL}[\![S]\!]_{\checkmark ListslenL}\ Q$› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync$_{ListslenL}$-locale* =
  *AfterExt-Sync$_{ptick}$-locale* ‹$\lambda r\ s.$ *if length* $r =$ *lenL then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
  **for** *lenL* :: *nat*

**sublocale** *AfterExt-Sync$_{ListslenL}$-locale* ⊆ *After-Sync$_{ListslenL}$-locale*
  **by** *unfold-locales*

**locale** *OpSemTransitions-Sync$_{ListslenL}$-locale* =
  *OpSemTransitions-Sync$_{ptick}$-locale* ‹$\lambda r\ s.$ *if length* $r =$ *lenL then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
  **for** *lenL* :: *nat*

**sublocale** *OpSemTransitions-Sync$_{ListslenL}$-locale* ⊆ *AfterExt-Sync$_{ListslenL}$-locale*
  **by** *unfold-locales*

**Control on right side** **locale** *After-Sync$_{ListslenR}$-locale* =
  *After-Sync$_{ptick}$-locale* ‹$\lambda r\ s.$ *if length* $s =$ *lenR then* $\lfloor r\ @\ s\rfloor$ *else* $\Diamond$›
  **for** *lenR* :: *nat*
**begin**

— Just checking...
**lemma** ‹$Sync_{ptick}\ P\ S\ Q = P\ {}_{lenR}[\![S]\!]_{\checkmark Listslen R}\ Q$› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync*$_{ListslenR}$*-locale* =
*AfterExt-Sync*$_{ptick}$*-locale* ‹$\lambda r\ s.$ *if length* $s = lenR$ *then* $\lfloor r \mathbin{@} s \rfloor$ *else* $\Diamond$›
**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* ‹$\lambda r\ s.$ *if length* $r = lenL$ *then* $\lfloor r \mathbin{@} s \rfloor$ *else* $\Diamond$›
**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*
‹$\lambda r\ s.$ *if length* $r = lenL \wedge$ *length* $s = lenR$ *then* $\lfloor r \mathbin{@} s \rfloor$ *else* $\Diamond$›
**for** *lenL lenR* :: *nat*
**begin**

— Just checking...
**lemma** ‹$Sync_{ptick}\ P\ S\ Q = P\ {}_{lenL}[\![S]\!]_{\checkmark lenR}\ Q$› **by** (*fact refl*)

**end**

**locale** *AfterExt-Sync*$_{Lists}$*-locale* =
*AfterExt-Sync*$_{ptick}$*-locale*
‹$\lambda r\ s.$ *if length* $r = lenL \wedge$ *length* $s = lenR$ *then* $\lfloor r \mathbin{@} s \rfloor$ *else* $\Diamond$›
**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*
‹$\lambda r\ s.$ *if length* $r = lenL \wedge$ *length* $s = lenR$ *then* $\lfloor r \mathbin{@} s \rfloor$ *else* $\Diamond$›
**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** $MultiSeq_{ptick}$ :: ‹['b list, 'b ⇒ 'r ⇒ ('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
  **where** $MultiSeq_{ptick}$-Nil : ‹$MultiSeq_{ptick}$ [] P = SKIP›
  | MultiSeq$_{ptick}$-Cons : ‹$MultiSeq_{ptick}$ (l # L) P = (λr. P l r ;✓ $MultiSeq_{ptick}$ L P)›

**syntax** -$MultiSeq_{ptick}$ ::
  ‹[pttrn, 'b list, 'b ⇒ 'r ⇒ ('a, 'r) $process_{ptick}$, 'r] ⇒ ('a, 'r) $process_{ptick}$›
  (‹(3$SEQ_{✓}$ - ∈@ -./ -)› [78,78,77] 77)
**syntax-consts** -$MultiSeq_{ptick}$ ⇌ $MultiSeq_{ptick}$
**translations** $SEQ_{✓}$ p ∈@ L. P ⇌ CONST $MultiSeq_{ptick}$ L (λp. P)

### 11.1.2 First Properties

**lemma** ‹$SEQ_{✓}$ p ∈@ []. P p = SKIP›
  **and** ‹$SEQ_{✓}$ p ∈@ [a]. P p = (λr. P a r)›
  **and** ‹$SEQ_{✓}$ p ∈@ [a, b]. P p = (λr. P a r ;✓ P b)›
  **and** ‹$SEQ_{✓}$ p ∈@ [a, b, c]. P p = (λr. P a r ;✓ P b ;✓ P c)›
  **by** (*simp-all add*: $Seq_{ptick}$-assoc)

**lemma** ‹$SEQ_{✓}$ p ∈@ [1::int .. 3]. P p = (λr. P 1 r ;✓ P 2 ;✓ P 3)›
  **by** (*simp add*: upto.simps $Seq_{ptick}$-assoc)

**lemma** ‹($SEQ_{✓}$ p ∈@ []. P p) = SKIP› **by** (*fact* $MultiSeq_{ptick}$-Nil)

**lemma** ‹($SEQ_{✓}$ l ∈@ (a # L). P l) = (λr. P a r ;✓ $SEQ_{✓}$ l ∈@ L. P l)› **by** (*fact* $MultiSeq_{ptick}$-Cons)

**lemma** *MultiSeq$_{ptick}$-singl* [*simp*] : ‹*SEQ$_{\checkmark}$ l ∈@ [a]. P l = P a*› **by** *simp*

**lemma** *MultiSeq$_{ptick}$-snoc* : ‹*SEQ$_{\checkmark}$ l ∈@ (L @ [a]). P l = (λr. (SEQ$_{\checkmark}$ l ∈@ L. P l) r ;$_{\checkmark}$ P a)*›
**by** (*induct L*) (*simp-all add*: *Seq$_{ptick}$-assoc*)

**lemma** *mono-MultiSeq$_{ptick}$-eq*:
‹(⋀*l. l ∈ set L ⟹ P l = Q l*) ⟹ *SEQ$_{\checkmark}$ l ∈@ L. P l = SEQ$_{\checkmark}$ l ∈@ L. Q l*›
**by** (*induct L*) *fastforce+*

**lemma** *MultiSeq$_{ptick}$-const* [*simp*] :
‹(*SEQ$_{\checkmark}$ l ∈@ L. (λr. P l)*) =
(*if L = [] then SKIP else (λr. SEQ l ∈@ L. P l)*)›
**by** (*induct L rule*: *rev-induct*) (*auto simp add*: *MultiSeq$_{ptick}$-snoc*)

### 11.1.3 Behaviour with binary version

**lemma** *MultiSeq$_{ptick}$-append*:
‹*SEQ$_{\checkmark}$ l ∈@ (L1 @ L2). P l = (λr. (SEQ$_{\checkmark}$ l ∈@ L1. P l) r ;$_{\checkmark}$ SEQ$_{\checkmark}$ l ∈@ L2. P l)*›
**by** (*induct L1 rule*: *list.induct*, *simp-all*, *metis Seq$_{ptick}$-assoc*)

### 11.1.4 Other Properties

**lemma** *MultiSeq$_{ptick}$-SKIP-neutral*:
‹*P a = SKIP ⟹ SEQ$_{\checkmark}$ l ∈@ (L1 @ [a] @ L2). P l = SEQ$_{\checkmark}$ l ∈@ (L1 @ L2). P l*›
**by** (*simp add*: *MultiSeq$_{ptick}$-append*)

**lemma** *MultiSeq$_{ptick}$-BOT-absorb*:
‹*P a = ⊥ ⟹ SEQ$_{\checkmark}$ l ∈@ (L1 @ [a] @ L2). P l = (λr. (SEQ$_{\checkmark}$ l ∈@ L1. P l) r ;$_{\checkmark}$ ⊥)*›
**by** (*simp add*: *MultiSeq$_{ptick}$-append lambda-strict*)

**lemma** *MultiSeq$_{ptick}$-STOP-absorb*:
‹*P a = (λr. STOP) ⟹ SEQ$_{\checkmark}$ l ∈@ (L1 @ [a] @ L2). P l =*
*(λr. (SEQ$_{\checkmark}$ l ∈@ L1. P l) r ; STOP)*›
**by** (*simp add*: *MultiSeq$_{ptick}$-append*)

**lemma** *is-ticks-length-MultiSeq$_{ptick}$* [*is-ticks-length-intro*] :
‹*length$_{\checkmark}$n((SEQ$_{\checkmark}$ l ∈@ L. P l) r)*›
**if** ‹*L ≠ []*› **and** ‹⋀*r′. r′ ∈ ✓s((SEQ$_{\checkmark}$ l ∈@ (butlast L). P l) r) ⟹ length$_{\checkmark}$n(P (last L) r′)*›
**proof** −
**from** *that(1)* **obtain** *l L′* **where** ‹*L = L′ @ [l]*›
**by** (*cases L rule*: *rev-cases*) *auto*

**with** *that*(*2*) **have** ‹$r' \in$ ✓**s**((*SEQ*$_{✓}$ $l$ ∈@ $L'$. $P$ $l$) $r$) $\Longrightarrow$ *length*$_{✓n}$($P$ $l$ $r'$)› **for** $r'$ **by** *simp*
**thus** *?thesis*
**by** (*auto simp add*: ‹$L = L'$ @ [$l$]› *MultiSeq*$_{ptick}$*-snoc intro*: *is-ticks-length-Seq*$_{ptick}$)
**qed**

### 11.1.5 Behaviour with injectivity

**lemma** *inj-on-mapping-over-MultiSeq*$_{ptick}$:
‹*inj-on f* (*set L*) $\Longrightarrow$
*SEQ*$_{✓}$ $l$ ∈@ $L$. $P$ $l$ = *SEQ*$_{✓}$ $l$ ∈@ *map f L*. $P$ (*inv-into* (*set L*) *f l*)›
**proof** (*induct L*)
**show** ‹*inj-on f* (*set* []) $\Longrightarrow$ *MultiSeq*$_{ptick}$ [] $P$ =
*SEQ*$_{✓}$ $x$∈@*map f* []. $P$ (*inv-into* (*set* []) *f x*)› **by** *simp*
**next**
**case** (*Cons a L*)
**show** *?case*
**proof** (*rule ext*)
**fix** $r$
**have** ‹(*SEQ*$_{✓}$ $l$ ∈@ ($a$ # $L$). $P$ $l$) $r$ = $P$ $a$ $r$ ;$_{✓}$ *SEQ*$_{✓}$ $l$ ∈@ $L$. $P$ $l$› **by** *simp*
**also have** ‹*SEQ*$_{✓}$ $l$ ∈@ $L$. $P$ $l$ = *SEQ*$_{✓}$ $l$ ∈@ *map f L*. $P$ (*inv-into* (*set L*) *f l*)›
**using** *Cons.hyps Cons.prems* **by** *auto*
**also have** ‹... = *SEQ*$_{✓}$ $l$ ∈@ *map f L*. $P$ (*inv-into* (*set* ($a$ # $L$)) *f l*)›
**using** *Cons.prems* **by** (*auto intro*!: *mono-MultiSeq*$_{ptick}$*-eq*)
**finally show** ‹(*SEQ*$_{✓}$ $l$ ∈@ ($a$ # $L$). $P$ $l$) $r$ =
(*SEQ*$_{✓}$ $l$ ∈@ *map f* ($a$ # $L$). $P$ (*inv-into* (*set* ($a$ # $L$)) *f l*)) $r$›
**using** *Cons.prems* **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* ($P$ ⟦$A$⟧$_{✓}$ $Q$) ⊗✓⇒⊗✓$_{rev}$ = $Q$ ⟦$A$⟧$_{✓rev}$ $P$). We therefore define the architectural version on a list.

**fun** *MultiSync*$_{ptick}$ ::
‹['$a$ *set*, '$b$ *list*, '$b$ ⇒ ('$a$, '$r$) *process*$_{ptick}$] ⇒ ('$a$, '$r$ *list*) *process*$_{ptick}$›
**where** ‹*MultiSync*$_{ptick}$ $S$ [] $P$ = *STOP*›

| ‹$MultiSync_{ptick}$ $S$ [$l$] $P$ = $RenamingTick$ ($P$ $l$) (λ$r$. [$r$])›
| ‹$MultiSync_{ptick}$ $S$ ($l$ # $m$ # $L$) $P$ = $P$ $l$ $[\![S]\!]_{\checkmark Rlist}$ $MultiSync_{ptick}$ $S$ ($m$ # $L$) $P$›

**syntax** -$MultiSync_{ptick}$ ::
‹[$pttrn$, '$a$ $set$, '$b$ $list$, ('$a$, '$r$) $process_{ptick}$] ⇒ ('$a$, '$r$) $process_{ptick}$›
(‹(3$[\![$-$]\!]_{\checkmark}$ - ∈@ -./ -)› [78,78,78,77] 77)
**syntax-consts** -$MultiSync_{ptick}$ ⇌ $MultiSync_{ptick}$
**translations** $[\![S]\!]_{\checkmark}$ $l$ ∈@ $L$. $P$ ⇌ $CONST$ $MultiSync_{ptick}$ $S$ $L$ (λ$l$. $P$)

Special case of $MultiSync_{ptick}$ $S$ $P$ when $S = \{\}$.

**abbreviation** $MultiInter_{ptick}$ ::
‹['$b$ $list$, '$b$ ⇒ ('$a$, '$r$) $process_{ptick}$] ⇒ ('$a$, '$r$ $list$) $process_{ptick}$›
**where** ‹$MultiInter_{ptick}$ $L$ $P$ ≡ $MultiSync_{ptick}$ {} $L$ $P$›

**syntax** -$MultiInter_{ptick}$ ::
‹[$pttrn$, '$b$ $list$, ('$a$, '$r$) $process_{ptick}$] ⇒ ('$a$, '$r$) $process_{ptick}$›
(‹(3$|||_{\checkmark}$ -∈@-./ -)› [78,78,77] 77)
**syntax-consts** -$MultiInter_{ptick}$ ⇌ $MultiInter_{ptick}$
**translations** $|||_{\checkmark}$ $l$ ∈@ $L$. $P$ ⇌ $CONST$ $MultiInter_{ptick}$ $L$ (λ$l$. $P$)

Special case of $MultiSync_{ptick}$ $S$ $P$ when $S = UNIV$.

**abbreviation** $MultiPar_{ptick}$ ::
‹['$b$ $list$, '$b$ ⇒ ('$a$, '$r$) $process_{ptick}$] ⇒ ('$a$, '$r$ $list$) $process_{ptick}$›
**where** ‹$MultiPar_{ptick}$ $L$ $P$ ≡ $MultiSync_{ptick}$ $UNIV$ $L$ $P$›

**syntax** -$MultiPar_{ptick}$ ::
‹[$pttrn$, '$b$ $list$, ('$a$, '$r$) $process_{ptick}$] ⇒ ('$a$, '$r$) $process_{ptick}$›
(‹(3$||_{\checkmark}$ -∈@-./ -)› [78,78,77] 77)
**syntax-consts** -$MultiPar_{ptick}$ ⇌ $MultiPar_{ptick}$
**translations** $||_{\checkmark}$ $l$ ∈@ $L$. $P$ ⇌ $CONST$ $MultiPar_{ptick}$ $L$ (λ$l$. $P$)

### 11.2.2 First properties

**lemma** *is-ticks-length-*$MultiSync_{ptick}$ [*is-ticks-length-intro*] :
‹$length_{\checkmark}$ $length$ $L$($[\![S]\!]_{\checkmark}$ $l$ ∈@ $L$. $P$ $l$)›
**by** (*induct L rule*: *induct-list012*)
(*simp-all add*: *is-ticks-length-STOP is-ticks-length-Renaming*
*is-ticks-length-Suc-*$Sync_{Rlist}$)

**lemma** $MultiSync_{ptick}$*-Cons* :
‹$[\![S]\!]_{\checkmark}$ $m$ ∈@ ($l$ # $L$). $P$ $m$ =
( *if* $L = []$ *then* $RenamingTick$ ($P$ $l$) (λ$r$. [$r$])
*else* $P$ $l$ $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ $m$ ∈@ $L$. $P$ $m$)›
**by** (*cases L*) *simp-all*

**lemma** *mono-MultiSync$_{ptick}$-eq* :
‹($\bigwedge l.\ l \in set\ L \Longrightarrow P\ l = Q\ l$) $\Longrightarrow$ $[\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l = [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l$›
**by** (*induct L rule*: *induct-list012*) *simp-all*

**lemma** *mono-MultiSync$_{ptick}$-eq2*:
‹($\bigwedge l.\ l \in set\ L \Longrightarrow P\ (f\ l) = Q\ l$) $\Longrightarrow$ $[\![S]\!]_{\checkmark}\ l \in@\ map\ f\ L.\ P\ l = [\![S]\!]_{\checkmark}\ l \in@\ L.$
$Q\ l$›
**by** (*induct L rule*: *induct-list012*) *simp-all*

— Some tests
**lemma** ‹($[\![S]\!]_{\checkmark}\ l \in@\ [].\ P\ l$) = *STOP*›
**and** ‹($[\![S]\!]_{\checkmark}\ l \in@\ [a].\ P\ l$) = *RenamingTick* ($P\ a$) ($\lambda r.\ [r]$)›
**and** ‹($[\![S]\!]_{\checkmark}\ l \in@\ [a,\ b].\ P\ l$) = $P\ a\ [\![S]\!]_{\checkmark Rlist}$ *RenamingTick* ($P\ b$) ($\lambda r.\ [r]$)›
**and** ‹($[\![S]\!]_{\checkmark}\ l \in@\ [a,\ b,\ c].\ P\ l$) = $P\ a\ [\![S]\!]_{\checkmark Rlist}$ ($P\ b\ [\![S]\!]_{\checkmark Rlist}$ *RenamingTick*
($P\ c$) ($\lambda r.\ [r]$))›
**by** *simp-all*

### 11.2.3 Properties

**lemma** *MultiSync$_{ptick}$-is-BOT-iff*:
‹$[\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l = \bot \longleftrightarrow (\exists\, l \in set\ L.\ P\ l = \bot)$›
**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*:
‹$l \in set\ L \Longrightarrow P\ l = \bot \Longrightarrow [\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l = \bot$›
**using** *MultiSync$_{ptick}$-is-BOT-iff* **by** *blast*

**lemma** *MultiSync$_{ptick}$-SKIP-id* :
‹$[\![S]\!]_{\checkmark}\ r \in@\ L.\ SKIP\ r = (if\ L = []\ then\ STOP\ else\ SKIP\ L)$›
**by** (*induct L rule*: *induct-list012*) *simp-all*

### 11.2.4 Behaviour with binary version

**lemma** *MultiSync$_{ptick}$-append* :
‹*L1* $\neq []$ $\Longrightarrow$ *L2* $\neq []$ $\Longrightarrow$
$[\![S]\!]_{\checkmark}\ l \in@\ (L1\ @\ L2).\ P\ l =$
$[\![S]\!]_{\checkmark}\ l \in@\ L1.\ P\ l\ _{length\ L1}[\![S]\!]_{\checkmark length\ L2}\ [\![S]\!]_{\checkmark}\ l \in@\ L2.\ P\ l$›
**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* = ‹$\lambda P.$ *RenamingTick* $P$ ($\lambda r.\ [r]$)›
**case** (*cons l L1*)

**have** ‹$[\![S]\!]_{\checkmark}$ $l \in@ ((l \# L1) @ L2)$. $P\ l = P\ l\ [\![S]\!]_{\checkmark Rlist}\ [\![S]\!]_{\checkmark}\ l \in@ (L1 @ L2)$. $P\ l$›
**by** (*simp add*: $MultiSync_{ptick}$-*Cons* ‹$L1 \neq []$›)
**also have** ‹$\ldots = ?RT\ (P\ l)\ _{Suc\ 0}[\![S]\!]_{\checkmark length\ (L1 @ L2)}\ [\![S]\!]_{\checkmark}\ l \in@ (L1 @ L2)$. $P\ l$›
**by** (*intro* $Sync_{Rlist}$-*to-*$Sync_{Lists}$ *is-ticks-length-*$MultiSync_{ptick}$)
**also have** ‹$\ldots = ?RT\ (P\ l)\ _{Suc\ 0}[\![S]\!]_{\checkmark length\ L1\ +\ length\ L2}$
$([\![S]\!]_{\checkmark}\ l \in@ L1.\ P\ l\ _{length\ L1}[\![S]\!]_{\checkmark length\ L2}\ [\![S]\!]_{\checkmark}\ l \in@ L2.\ P\ l)$›
**using** *cons.hyps*(2) *cons.prems* **by** *simp*
**also have** ‹$\ldots = ?RT\ (P\ l)\ _{Suc\ 0}[\![S]\!]_{\checkmark length\ L1}\ [\![S]\!]_{\checkmark}\ l \in@ L1.\ P\ l$
$_{Suc\ 0\ +\ length\ L1}[\![S]\!]_{\checkmark length\ L2}\ [\![S]\!]_{\checkmark}\ l \in@ L2.\ P\ l$›
**by** (*simp add*: $Sync_{Lists}$-*assoc*)
**also have** ‹$?RT\ (P\ l)\ _{Suc\ 0}[\![S]\!]_{\checkmark length\ L1}\ [\![S]\!]_{\checkmark}\ l \in@ L1.\ P\ l =$
$P\ l\ [\![S]\!]_{\checkmark Rlist}\ [\![S]\!]_{\checkmark}\ l \in@ L1.\ P\ l$›
**by** (*intro* $Sync_{Rlist}$-*to-*$Sync_{Lists}$[*symmetric*] *is-ticks-length-*$MultiSync_{ptick}$)
**also have** ‹$\ldots = [\![S]\!]_{\checkmark}\ l \in@ (l \# L1).\ P\ l$›
**by** (*simp add*: $MultiSync_{ptick}$-*Cons* ‹$L1 \neq []$›)
**finally show** *?case* **by** *simp*
**qed**

### 11.2.5 Behaviour with injectivity

**lemma** $inj\text{-}on\text{-}mapping\text{-}over\text{-}MultiSync_{ptick}$:
‹*inj-on f* (*set L*) $\Longrightarrow$
$[\![S]\!]_{\checkmark}\ l \in@ L.\ P\ l = [\![S]\!]_{\checkmark}\ l \in@ map\ f\ L.\ P\ (inv\text{-}into\ (set\ L)\ f\ l)$›
**proof** (*induct L rule*: *induct-list012*)
**case** (*3 l' l'' L*)
**have** ‹$[\![S]\!]_{\checkmark}\ l \in@ (l' \# l'' \# L).\ P\ l =$
$P\ l'\ [\![S]\!]_{\checkmark Rlist}\ [\![S]\!]_{\checkmark}\ l \in@ (l'' \# L).\ P\ l$› **by** *simp*
**also have** ‹$[\![S]\!]_{\checkmark}\ l \in@ (l'' \# L).\ P\ l =$
$[\![S]\!]_{\checkmark}\ l \in@ map\ f\ (l'' \# L).\ P\ (inv\text{-}into\ (set\ (l'' \# L))\ f\ l)$›
**by** (*metis 3.hyps*(2) *3.prems inj-on-insert list.simps*(15))
**also have** ‹$\ldots = [\![S]\!]_{\checkmark}\ l \in@ map\ f\ (l'' \# L).\ P\ (inv\text{-}into\ (set\ (l' \# l'' \# L))\ f\ l)$›
**using** *3.prems* **by** (*auto intro*!: $mono\text{-}MultiSync_{ptick}$-*eq*)
**also have** ‹$P\ l' = P\ (inv\text{-}into\ (set\ (l' \# l'' \# L))\ f\ (f\ l'))$›
**using** *3.prems* **by** *auto*
**finally show** *?case* **by** *simp*
**qed** *simp-all*

### 11.2.6 Permuting the Sequence

#### A particular Case

**lemma** $MultiSync_{ptick}$-*snoc* :
‹$[\![S]\!]_{\checkmark}\ m \in@ (L @ [l]).\ P\ m =$
$(\ if\ L = []\ then\ RenamingTick\ (P\ l)\ (\lambda r.\ [r])$
$else\ [\![S]\!]_{\checkmark}\ m \in@ L.\ P\ m\ [\![S]\!]_{\checkmark Llist}\ P\ l)$›
**by** (*simp add*: $MultiSync_{ptick}$-*append*)

(*metis* (*lifting*) *ext* $Sync_{Llist}$*-to-*$Sync_{Lists}$ *is-ticks-length-*$MultiSync_{ptick}$)

At the beginning, we wanted to prove the following property.

**theorem** $MultiSync_{ptick}$*-rev* :
‹$[\![S]\!]_{\checkmark}$ *l* ∈@ (*rev L*). *P l* = *RenamingTick* ($[\![S]\!]_{\checkmark}$ *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** ‹$[\![S]\!]_{\checkmark}$ *m* ∈@ (*rev* (*l* # *L*)). *P m* = $[\![S]\!]_{\checkmark}$ *m* ∈@ (*rev L*). *P m* $[\![S]\!]_{\checkmark Llist}$ *P l*›
**by** (*simp add*: $MultiSync_{ptick}$*-snoc* ‹*L* ≠ []›)
**also have** ‹... = *?RT* ($[\![S]\!]_{\checkmark}$ *m* ∈@ *L*. *P m*) *rev* $[\![S]\!]_{\checkmark Llist}$ *P l*›
**by** (*simp only*: *Cons.hyps*)
**also have** ‹... = *?RT* ($[\![S]\!]_{\checkmark}$ *m* ∈@ *L*. *P m*) *rev* $[\![S]\!]_{\checkmark Llist}$ *?RT* (*P l*) *id*› **by** *simp*
**also have** ‹... = $Sync_{ptick}$*-locale.*$Sync_{ptick}$ (λ*r s*. *Some* (*rev r* @ [*s*]))
($[\![S]\!]_{\checkmark}$ *m* ∈@ *L*. *P m*) *S* (*P l*)›
**by** (*subst* $Sync_{Rlist}$*.*$Sync_{ptick}$*-comm-locale-sym.inj-RenamingTick-*$Sync_{ptick}$*-inj-RenamingTick*) *simp-all*
**also have** ‹... = *?RT* (*P l* $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *m* ∈@ *L*. *P m*) *rev*›
**proof** (*subst* $Sync_{Rlist}$*.inj-on-RenamingTick-*$Sync_{ptick}$)
**show** ‹*inj-on rev* $Sync_{Rlist}$*.range-tick-join*› **by** *simp*
**next**
**show** ‹$Sync_{ptick}$*-locale.*$Sync_{ptick}$ (λ*r s*. *Some* (*rev r* @ [*s*])) ($[\![S]\!]_{\checkmark}$ *m* ∈@ *L*. *P m*) *S* (*P l*) =
$Sync_{ptick}$*-locale.*$Sync_{ptick}$
(λ*r s*. *case Some* (*r* # *s*) *of None* ⇒ *None* | *Some r-s* ⇒ *Some* (*rev r-s*))
(*P l*) *S* ($MultiSync_{ptick}$ *S L P*)›
**by** (*subst* $Sync_{ptick}$*-locale.*$Sync_{ptick}$*-sym*, *simp-all*)
(*unfold-locales*, *blast*)
**qed**
**also have** ‹*P l* $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *m* ∈@ *L*. *P m* = $[\![S]\!]_{\checkmark}$ *m* ∈@ (*l* # *L*). *P m*›
**by** (*simp add*: $MultiSync_{ptick}$*-Cons* ‹*L* ≠ []›)
**finally show** *?case* **.**
**qed**
**qed**

This has just been established for *rev L*, which is a particular permutation of the list *L*. It turns out that it actually holds for any permutation. The rest of this file constitutes the proof.

### Arbitrary Permutation

**Some preliminary results** **lemma** *permute-list-transpose-eq-list-update* :
‹*i* < *length xs* ⟹ *j* < *length xs* ⟹
*permute-list* (*Transposition.transpose i j*) *xs* = *xs*[*i* := *xs*!*j*, *j* := *xs*!*i*]›
**by** (*auto simp add*: *permute-list-def transpose-def intro*: *nth-equalityI*)

**lemma** *inj-on-permute-list-transpose* :
‹*i* < *n* ⟹ *j* < *n* ⟹ *inj-on* (*permute-list* (*Transposition.transpose i j*)) {*xs*. *n* ≤ *length xs*}›
**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* :
‹*i* < *length L* ⟹ *j* < *length L* ⟹
*rev* (*permute-list* (*Transposition.transpose i j*) *L*) =
*permute-list* (*Transposition.transpose* (*length L* − *Suc i*) (*length L* − *Suc j*)) (*rev L*)›
**by** (*simp add*: *permute-list-transpose-eq-list-update rev-nth rev-update*)

**lemma** *permute-list-transpose-rev* :
‹*i* < *length L* ⟹ *j* < *length L* ⟹
*permute-list* (*Transposition.transpose i j*) (*rev L*) =
*rev* (*permute-list* (*Transposition.transpose* (*length L* − *Suc i*) (*length L* − *Suc j*)) *L*)›
**by** (*simp add*: *permute-list-transpose-eq-list-update rev-nth rev-update*)

**lemma** *tickFree-map-map-event*$_{ptick}$*-id-eq* :
‹*tF t* ⟹ *map* (*map-event*$_{ptick}$ *id g*) *t* = *t*›
**and** *tickFree-mem-T-RenamingTick-iff-mem-T* :
‹*tF t* ⟹ *t* ∈ 𝒯 (*RenamingTick P g*) ⟷ *t* ∈ 𝒯 *P*›
**and** *tickFree-mem-D-RenamingTick-iff-mem-D* :
‹*tF t* ⟹ *t* ∈ 𝒟 (*RenamingTick P g*) ⟷ *t* ∈ 𝒟 *P*›
**for** *P* :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** *g* :: ‹'*r* ⇒ '*r*›
— Necessarily here, antecedents and images for *g* share the same type.
**proof** −
**show** ∗ : ‹*tF t* ⟹ *map* (*map-event*$_{ptick}$ *id g*) *t* = *t*› **for** *t* :: ‹('*a*, '*r*) *trace*$_{ptick}$›
**by** (*induct t*) (*auto simp add*: *is-ev-def*)
**show** ‹*tF t* ⟹ *t* ∈ 𝒯 (*RenamingTick P g*) ⟷ *t* ∈ 𝒯 *P*›
**by** (*auto simp add*: *T-Renaming* ∗ *map-event*$_{ptick}$*-tickFree D-T is-processT7*)
**show** ‹*tF t* ⟹ *t* ∈ 𝒟 (*RenamingTick P g*) ⟷ *t* ∈ 𝒟 *P*›
**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 $P$ $[\![S]\!]_{\checkmark Rlist}$ $Q$.

**lemma** *Sync$_{Rlist}$-RenamingTick-permute-list-transpose* :
‹$P$ $[\![S]\!]_{\checkmark Rlist}$ *RenamingTick* $Q$ (*permute-list* (*Transposition.transpose* $i$ $j$)) =
*RenamingTick* ($P$ $[\![S]\!]_{\checkmark Rlist}$ $Q$) (*permute-list* (*Transposition.transpose* (*Suc* $i$) (*Suc* $j$)))›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹$i < n$› ‹$j < n$› ‹$\bigwedge rs.\ rs \in \checkmark\mathbf{s}(Q) \Longrightarrow n \leq$ *length* $rs$›
**proof** −
**let** *?τ* = *Transposition.transpose*
**let** *?pl-τ* = ‹$\lambda i\ j.$ *permute-list* (*?τ* $i$ $j$)›
**let** *?fun-evt* = ‹$\lambda i\ j.$ *map-event$_{ptick}$* *id* (*?pl-τ* $i$ $j$)›
**let** *?map-evt* = ‹$\lambda i\ j.$ *map* (*?fun-evt* $i$ $j$)›
**let** *?RT* = ‹$\lambda i\ j\ P.$ *RenamingTick* $P$ (*?pl-τ* $i$ $j$)›
**let** *?tj* = ‹$\lambda r\ s.\ \lfloor r \#\ s \rfloor$›
**note** *map-event$_{ptick}$-eq-iffs* [*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-τ-imp-eq* : ‹$r = r'$›
**if** ‹$n \leq$ *length* $r$› **and** ‹*?pl-τ* $i$ $j$ $r$ = *?pl-τ* $i$ $j$ $r'$› **for** $r$ $r'$ :: ‹'$r$ *list*›
**proof** −
**from** ‹$n \leq$ *length* $r$› **have** ‹$n \leq$ *length* (*?pl-τ* $i$ $j$ $r$)› **by** *simp*
**with** ‹*?pl-τ* $i$ $j$ $r$ = *?pl-τ* $i$ $j$ $r'$› **have** ‹$n \leq$ *length* $r'$› **by** *simp*
**have** ‹*inj-on* (*?pl-τ* $i$ $j$) {$r$. $n \leq$ *length* $r$}›
**by** (*simp add*: ‹$i < n$› ‹$j < n$› *inj-on-permute-list-transpose*)
**with** ‹$n \leq$ *length* $r$› ‹$n \leq$ *length* $r'$› ‹*?pl-τ* $i$ $j$ $r$ = *?pl-τ* $i$ $j$ $r'$›
**show** ‹$r = r'$› **by** (*auto dest*: *inj-onD*)
**qed**
**have** *pl-τ-pl-τ* : ‹$n \leq$ *length* $r \Longrightarrow$ *?pl-τ* $i$ $j$ (*?pl-τ* $i$ $j$ $r$) = $r$› **for** $r$ :: ‹'$r$ *list*›
**by** (*subst permute-list-compose*[*symmetric*])
(*metis lessThan-iff order-less-le-trans permutes-swap-id* ‹$i < n$› ‹$j < n$›, *simp*)
**have** *fun-evt-fun-evt* :
‹(*case* $e$ *of ev* $a \Rightarrow$ *True* | $\checkmark(r) \Rightarrow n \leq$ *length* $r$) $\Longrightarrow$ *?fun-evt* $i$ $j$ (*?fun-evt* $i$ $j$ $e$) = $e$›
**for** $e$ :: ‹('$a$, '$r$ *list*) *event$_{ptick}$*›
**by** (*cases* $e$) (*simp-all add*: *pl-τ-pl-τ*)
**have** *map-evt-map-evt* :
‹($\bigwedge e.\ e \in$ *set* $t \Longrightarrow$ *case* $e$ *of ev* $a \Rightarrow$ *True* | $\checkmark(r) \Rightarrow n \leq$ *length* $r$)
$\Longrightarrow$ *?map-evt* $i$ $j$ (*?map-evt* $i$ $j$ $t$) = $t$› **for** $t$ :: ‹('$a$, '$r$ *list*) *trace$_{ptick}$*›
**by** (*induct* $t$) (*simp-all add*: *fun-evt-fun-evt*)
**from** ‹$i < n$› ‹$j < n$› **have** *pl-τ-Cons* :
‹$n \leq$ *length* $s \Longrightarrow$ *?pl-τ* (*Suc* $i$) (*Suc* $j$) ($r \#\ s$) = $r \#$ *?pl-τ* $i$ $j$ $s$› **for** $r$ **and** $s$ :: ‹'$r$ *list*›
**by** (*simp add*: *list-update-append1 nth-append-left permute-list-transpose-eq-list-update*)
**show** ‹*?lhs* = *?rhs*›
**proof** (*rule Process-eq-optimizedI*)
**fix** $t$ **assume** ‹$t \in \mathcal{D}$ *?lhs*›
**then obtain** $u$ $v$ *t-P* *t-Q* **where** $*$ : ‹$t = u$ @ $v$› ‹*tF* $u$› ‹*ftF* $v$›
‹$u$ *setinterleaves*$_{\checkmark\ ?tj}$ ((*t-P*, *t-Q*), $S$)›
‹*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* $i$ $j$ $Q$)›
**unfolding** *Sync$_{Rlist}$.D-Sync$_{ptick}$* **by** *blast*

**from** *tickFree-setinterleaves$_{ptick}$-iff*[*THEN iffD1*, *OF* ∗(*4*, *2*)] **have** ‹*tF t-Q*›
..
**with** ∗(*5*) **have** ‹*t-P* ∈ $\mathcal{D}$ *P* ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨ *t-P* ∈ $\mathcal{T}$ *P* ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**by** (*simp add*: *tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-D*)
**with** ∗(*1*−*4*) **have** ‹*t* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓Rlist}$ *Q*)›
**by** (*auto simp add*: *Sync$_{Rlist}$.D-Sync$_{ptick}$*)
**thus** ‹*t* ∈ $\mathcal{D}$ *?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* ∈ $\mathcal{D}$ *?rhs*›
**then obtain** *t1 t2*
**where** ∗ : ‹*t* = *?map-evt* (*Suc i*) (*Suc j*) *t1* @ *t2*› ‹*tF t1*› ‹*ftF t2*› ‹*t1* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓Rlist}$ *Q*)›
**by** (*auto simp add*: *D-Renaming*)
**from** ∗(*1*, *2*) **have** ‹*t* = *t1* @ *t2*›
**by** (*simp add*: *tickFree-map-map-event$_{ptick}$-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* ∈ $\mathcal{D}$ *P* ∧ *t-Q* ∈ $\mathcal{T}$ *Q* ∨ *t-P* ∈ $\mathcal{T}$ *P* ∧ *t-Q* ∈ $\mathcal{D}$ *Q*›
**unfolding** *Sync$_{Rlist}$.D-Sync$_{ptick}$* **by** *blast*
**from** *tickFree-setinterleaves$_{ptick}$-iff*[*THEN iffD1*, *OF* ∗∗(*4*, *2*)] **have** ‹*tF t-Q*›
..
**with** ∗∗(*5*) **have** ‹*t-P* ∈ $\mathcal{D}$ *P* ∧ *t-Q* ∈ $\mathcal{T}$ (*?RT i j Q*) ∨ *t-P* ∈ $\mathcal{T}$ *P* ∧ *t-Q* ∈ $\mathcal{D}$ (*?RT i j Q*)›
**by** (*simp-all add*: *tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-*
**with** ∗∗(*1*−*4*) ∗(*2*, *3*) ‹*t* = *t1* @ *t2*› **show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**by** (*auto simp add*: *Sync$_{Rlist}$.D-Sync$_{ptick}$ intro*: *front-tickFree-append*)
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› ‹*t* ∉ $\mathcal{D}$ *?rhs*› ‹*t* ∉ $\mathcal{D}$ *?lhs*›
**then obtain** *t′* **where** ∗ : ‹*t* = *?map-evt* (*Suc i*) (*Suc j*) *t′*›
‹(*t′*, *?fun-evt* (*Suc i*) (*Suc j*) −‘ *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧$_{✓Rlist}$ *Q*)›
**unfolding** *Renaming-projs* **by** *blast*
**from** ∗(*2*) ‹*t* ∉ $\mathcal{D}$ *?rhs*› **have** ‹*t′* ∉ $\mathcal{D}$ (*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-event$_{ptick}$-tickFree*
*tickFree-map-map-event$_{ptick}$-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*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*t′ setinterleaves*$_{✓}$ *?tj* ((*t-P*, *t-Q*), *S*)›
‹*?fun-evt* (*Suc i*) (*Suc j*) −‘ *X* ⊆ *super-ref-Sync$_{ptick}$ ?tj X-P S X-Q*›
**unfolding** *Sync$_{Rlist}$.Sync$_{ptick}$-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*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**assume** ‹*tF t′*›
**have** ‹*?map-evt* (*Suc i*) (*Suc j*) *t′* = *t′*›

**by** (*simp add*: ‹*tF t'*› *tickFree-map-map-event*$_{ptick}$*-id-eq*)
**have** ‹*?map-evt i j t-Q* = *t-Q*›
**using** ∗∗(*3*) ‹*tF t'*› *tickFree-map-map-event*$_{ptick}$*-id-eq tickFree-setinterleaves*$_{ptick}$*-iff*
**by** *blast*
**define** *X-Q'* **where** ‹*X-Q'* ≡ *X-Q* ∩ (*range ev* ∪ {✓(*r*) |*r*. *n* ≤ *length r*})›
**define** *X'* **where** ‹*X'* ≡ *X* ∩ (*range ev* ∪ {✓(*rs*) |*rs*. *Suc n* ≤ *length rs*})›
**have** ‹*X-Q'* ⊆ *X-Q*› **unfolding** *X-Q'-def* **by** *blast*
**with** ∗∗(*2*) *is-processT4* **have** ‹(*t-Q*, *X-Q'*) ∈ $\mathcal{F}$ *Q*› **by** *blast*
**moreover have** ‹*?fun-evt i j* −‘ (*?fun-evt i j* ‘ *X-Q'*) = *X-Q'*›
**by** (*auto simp add*: *X-Q'-def*)
(*use length-ge-eq-pl-τ-imp-eq* **in** *blast*)+
**moreover have** ‹*?map-evt i j t-Q* = *t-Q*› **by** *fact*
**ultimately have** ‹(*t-Q*, *?fun-evt i j* ‘ *X-Q'*) ∈ $\mathcal{F}$ (*?RT i j Q*)›
**by** (*auto simp add*: *F-Renaming*)
**moreover have** ‹*e* ∈ *X'* ⟹ *e* ∈ *super-ref-Sync*$_{ptick}$ *?tj X-P S* (*?fun-evt i j* ‘ *X-Q'*)› **for** *e*
**using** ∗∗(*4*)[*THEN set-mp*, *of* ‹*?fun-evt* (*Suc i*) (*Suc j*) *e*›]
**unfolding** *X'-def X-Q'-def super-ref-Sync*$_{ptick}$*-def*
**by** (*auto simp add*: *image-iff pl-τ-Cons*) (*use pl-τ-pl-τ* **in** *force*)
**ultimately have** ‹(*t*, *X'*) ∈ $\mathcal{F}$ *?lhs*›
**by** (*simp add*: *Sync*$_{Rlist}$*.F-Sync*$_{ptick}$)
(*metis* ∗(*1*) ∗∗(*1*, *3*) ‹*?map-evt* (*Suc i*) (*Suc j*) *t'* = *t'*› *subsetI*)
**moreover have** ‹*Suc n* ≤ *length* (*rs*)› **if** ‹*t* @ [✓(*rs*)] ∈ $\mathcal{T}$ *?lhs*› **for** *rs*
**proof** −
**from** ‹*t* ∉ $\mathcal{D}$ *?lhs*› **have** ‹*t* @ [✓(*rs*)] ∉ $\mathcal{D}$ *?lhs*›
**by** (*meson is-processT9*)
**with** ‹*t* @ [✓(*rs*)] ∈ $\mathcal{T}$ *?lhs*›
**obtain** *t-P''* *t-Q''* **where** £ : ‹*t-P''* ∈ $\mathcal{T}$ *P*› ‹*t-Q''* ∈ $\mathcal{T}$ (*?RT i j Q*)›
‹*t* @ [✓(*rs*)] *setinterleaves*$_{✓}$$_{\lambda r\ s.\ \lfloor r\ \#\ s\rfloor}$ ((*t-P''*, *t-Q''*), *S*)›
**unfolding** *Sync*$_{Rlist}$*.Sync*$_{ptick}$*-projs* **by** *blast*
**from** £ **obtain** *t-P'''* *t-Q'''* *r s*
**where** ££ : ‹*rs* = *r* # *s*› ‹*t-P''* = *t-P'''* @ [✓(*r*)]› ‹*t-Q''* = *t-Q'''* @ [✓(*s*)]›
‹*t setinterleaves*$_{✓}$$_{\lambda r\ s.\ \lfloor r\ \#\ s\rfloor}$ ((*t-P'''*, *t-Q'''*), *S*)›
**by** (*auto elim*: *snoc-tick-setinterleaves*$_{ptick}$*E*)
**have** ‹*tF t-Q'''*› **using** £(*2*) ££(*3*) *append-T-imp-tickFree* **by** *blast*
**from** ‹*t* ∉ $\mathcal{D}$ *?lhs*› £(*1*) ££(*2*, *4*) **have** ‹*t-Q'''* ∉ $\mathcal{D}$ (*?RT i j Q*)›
**by** (*simp add*: *Sync*$_{Rlist}$*.D-Sync*$_{ptick}$*'*)
(*use front-tickFree-Nil is-processT3-TR-append* **in** *blast*)
**with** £(*2*) **obtain** *t-Q''''*
**where** ‹*?map-evt i j t-Q''''* = *t-Q'''* @ [✓(*s*)]› ‹*t-Q''''* ∈ $\mathcal{T}$ *Q*›
**by** (*simp add*: *Renaming-projs*)
(*metis* ££(*3*) ‹*t-Q'''* ∉ $\mathcal{D}$ (*?RT i j Q*)› *is-processT7 is-processT9*
*tickFree-map-map-event*$_{ptick}$*-id-eq tickFree-mem-D-RenamingTick-iff-mem-D*)
**then obtain** *s'* **where** £££ : ‹*s* = *?pl-τ i j s'*› ‹*t-Q'''* @ [✓(*s'*)] ∈ $\mathcal{T}$ *Q*›
**by** (*auto simp add*: *map-eq-append-conv Cons-eq-map-conv*
*append-T-imp-tickFree tickFree-map-map-event*$_{ptick}$*-id-eq*)
**have** ‹*s'* ∈ ✓**s**(*Q*)›
**by** (*meson* £££(*2*) ‹*tF t-Q'''*› ‹*t-Q'''* ∉ $\mathcal{D}$ (*?RT i j Q*)› *is-processT9*
*strict-ticks-of-memI tickFree-mem-D-RenamingTick-iff-mem-D*)

**with** ‹$\bigwedge rs.\ rs \in \checkmark\mathbf{s}(Q) \Longrightarrow n \leq length\ rs$› **have** ‹$n \leq length\ s'$› **.**
**with** ‹$s = ?pl\text{-}\tau\ i\ j\ s'$› **have** ‹$n \leq length\ s$› **by** *simp*
**with** ‹$rs = r\ \#\ s$› **show** ‹$Suc\ n \leq length\ rs$› **by** *simp*
**qed**
**ultimately have** ‹$(t,\ X' \cup (X \cap \{\checkmark(rs)\ |rs.\ \neg\ Suc\ n \leq length\ rs\})) \in \mathcal{F}\ ?lhs$›
**using** *is-processT5-S7'* **by** *blast*
**also have** ‹$X' \cup (X \cap \{\checkmark(rs)\ |rs.\ \neg\ Suc\ n \leq length\ rs\}) = X$›
**by** (*simp add*: *set-eq-iff X'-def image-iff*) (*meson* $event_{ptick}$*.exhaust*)
**finally show** ‹$(t,\ X) \in \mathcal{F}\ ?lhs$› **.**
**next**
**fix** $t''\ rs$ **assume** ‹$tF\ t''$› ‹$t' = t'' @ [\checkmark(rs)]$›
**from** $**(3)$ **obtain** *t-P' t-Q' r s*
**where** $*** : ‹r\ \#\ s = rs›$
‹$t''\ setinterleaves_{\checkmark}\ ?tj\ ((t\text{-}P',\ t\text{-}Q'),\ S)$›
‹$t\text{-}P = t\text{-}P' @ [\checkmark(r)]$› ‹$t\text{-}Q = t\text{-}Q' @ [\checkmark(s)]$›
**by** (*auto elim*: $snoc\text{-}tick\text{-}setinterleaves_{ptick}E$
*simp add*: ‹$t' = t'' @ [\checkmark(rs)]$› *split*: *if-split-asm*)
**have** ‹$n \leq length\ s$›
**proof** −
**from** $**(1)[THEN\ F\text{-}T]$ $**(3)$ ‹$t' \notin \mathcal{D}\ (P\ [\![S]\!]_{\checkmark Rlist}\ Q)$› **have** ‹$t\text{-}Q \notin \mathcal{D}\ Q$›
**by** (*simp add*: $Sync_{Rlist}.Sync_{ptick}\text{-}projs'$)
(*use front-tickFree-Nil* **in** *blast*)
**with** ‹$(t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ Q$›$[THEN\ F\text{-}T]$ **have** ‹$s \in \checkmark\mathbf{s}(Q)$›
**by** (*simp add*: $***(4)$ *strict-ticks-of-memI*)
**with** ‹$\bigwedge rs.\ rs \in \checkmark\mathbf{s}(Q) \Longrightarrow n \leq length\ rs$› **show** ‹$n \leq length\ s$› **.**
**qed**
**from** ‹$t''\ setinterleaves_{\checkmark}\ ?tj\ ((t\text{-}P',\ t\text{-}Q'),\ S)$›
**have** ‹$?map\text{-}evt\ (Suc\ i)\ (Suc\ j)\ t''\ setinterleaves_{\checkmark}\ ?tj\ ((t\text{-}P',\ ?map\text{-}evt\ i\ j\ t\text{-}Q'),\ S)$›
**by** (*metis* (*no-types*, *lifting*) ‹$tF\ t''$› $tickFree\text{-}map\text{-}map\text{-}event_{ptick}\text{-}id\text{-}eq$
$tickFree\text{-}setinterleaves_{ptick}\text{-}iff$)
**from** $setinterleaves_{ptick}\text{-}snoc\text{-}tick$
$[OF\ this,\ of\ r$ ‹$?pl\text{-}\tau\ i\ j\ s$› ‹$?pl\text{-}\tau\ (Suc\ i)\ (Suc\ j)\ rs$›$]$ ‹$n \leq length\ s$›
**have** ‹$?map\text{-}evt\ (Suc\ i)\ (Suc\ j)\ t'\ setinterleaves_{\checkmark}\ ?tj\ ((t\text{-}P,\ ?map\text{-}evt\ i\ j\ t\text{-}Q),\ S)$›
**by** (*simp add*: $***(1,\ 3,\ 4)$ ‹$t' = t'' @ [\checkmark(rs)]$›) (*metis* $***(1)$ *pl-τ-Cons*)
**moreover from** $**(2)[THEN\ F\text{-}T]$ **have** ‹$(?map\text{-}evt\ i\ j\ t\text{-}Q,\ UNIV) \in \mathcal{F}\ (?RT\ i\ j\ Q)$›
**by** (*simp add*: $***(4)$, *intro tick-T-F*) (*auto simp add*: *T-Renaming*)
**moreover have** ‹$(t\text{-}P,\ UNIV) \in \mathcal{F}\ P$›
**by** (*metis* $**(1)$ $***(3)$ *F-T tick-T-F*)
**moreover have** ‹$e \in X \Longrightarrow e \in super\text{-}ref\text{-}Sync_{ptick}\ ?tj\ UNIV\ S\ UNIV$› **for** *e*
**using** $**(4)[THEN\ set\text{-}mp,\ of$ ‹$?fun\text{-}evt\ (Suc\ i)\ (Suc\ j)\ e$›$]$
**by** (*cases e*) (*auto simp add*: $super\text{-}ref\text{-}Sync_{ptick}\text{-}def$)
**ultimately show** ‹$(t,\ X) \in \mathcal{F}\ ?lhs$›
**using** $*(1)$ **by** (*simp add*: $Sync_{Rlist}.F\text{-}Sync_{ptick}$) *blast*
**qed**
**next**

**fix** $t$ $X$ **assume** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*› ‹$t \notin \mathcal{D}$ *?lhs*›
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : ‹$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}\ P$› ‹$(t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}$ (*?RT i j Q*)›
‹$t$ *setinterleaves*$_{\checkmark\ ?tj}$ $((t\text{-}P,\ t\text{-}Q),\ S)$›
‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ *?tj X-P S X-Q*›
**unfolding** *Sync*$_{Rlist}$.*Sync*$_{ptick}$*-projs* **by** *force*
**from** $*(1,\ 3)$ ‹$t \notin \mathcal{D}$ *?lhs*› *F-T front-tickFree-Nil*
**have** ‹*t-Q* $\notin \mathcal{D}$ (*?RT i j Q*)› **unfolding** *Sync*$_{Rlist}$.*D-Sync*$_{ptick}{}'$ **by** *blast*
**with** $*(2)$ **obtain** *t-Q′* **where** $**$ : ‹*t-Q* = *?map-evt i j t-Q′*› ‹(*t-Q′*, *?fun-evt i j* −‘ *X-Q*) $\in \mathcal{F}\ Q$›
**unfolding** *Renaming-projs* **by** *blast*
**define** *X-Q′* **where** ‹*X-Q′* ≡ *X-Q* ∩ (*range ev* ∪ {✓(*rs*) |*rs*. *n* ≤ *length rs*})›
**define** *X′* **where** ‹*X′* ≡ *X* ∩ (*range ev* ∪ {✓(*rs*) |*rs*. *Suc n* ≤ *length rs*})›

**from** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›[*THEN F-T*] **consider** ‹*tF t*› | *t′ rs* **where** ‹*tF t′*› ‹*t* = *t′* @ [✓(*rs*)]›
**using** *T-nonTickFree-imp-decomp append-T-imp-tickFree* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}$ *?rhs*›
**proof** *cases*
**assume** ‹*tF t*›
**hence** ‹*?map-evt* (*Suc i*) (*Suc j*) *t* = *t*›
**by** (*simp add*: *tickFree-map-map-event*$_{ptick}$*-id-eq*)
**have** ‹*?map-evt i j t-Q′* = *t-Q′*›
**using** $*(3)$ $**(1)$ ‹*tF t*› *map-event*$_{ptick}$*-tickFree tickFree-map-map-event*$_{ptick}$*-id-eq*
*tickFree-setinterleaves*$_{ptick}$*-iff* **by** *blast*
**have** ‹(*t-Q*, *?fun-evt i j* −‘ *X-Q*) $\in \mathcal{F}\ Q$›
**by** (*simp add*: $**(1,\ 2)$ ‹*?map-evt i j t-Q′* = *t-Q′*›)
**hence** ‹(*t-Q*, *?fun-evt i j* −‘ *X-Q′*) $\in \mathcal{F}\ Q$›
**by** (*simp add*: *X-Q′-def is-processT4*)
**moreover have** ‹$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}\ P$› **by** (*fact* $*(1)$)
**moreover have** ‹$t$ *setinterleaves*$_{\checkmark\ ?tj}$ $((t\text{-}P,\ t\text{-}Q),\ S)$› **by** (*fact* $*(3)$)
**moreover have** ‹$e \in$ *?fun-evt* (*Suc i*) (*Suc j*) −‘ $X' \Longrightarrow$
$e \in$ *super-ref-Sync*$_{ptick}$ *?tj X-P S* (*?fun-evt i j* −‘ *X-Q*)› **for** $e$
**using** *set-mp*[*OF* $*(4)$, *of* ‹*?fun-evt* (*Suc i*) (*Suc j*) *e*›]
**by** (*auto simp add*: *super-ref-Sync*$_{ptick}$*-def X′-def pl-τ-Cons*)
**ultimately have** ‹$(t,$ *?fun-evt* (*Suc i*) (*Suc j*) −‘ $X') \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark Rlist}\ Q)$›
**by** (*unfold Sync*$_{Rlist}$.*F-Sync*$_{ptick}$, *clarify*)
(*metis* $**(1,\ 2)$ ‹*?map-evt i j t-Q′* = *t-Q′*› *subsetI*)
**moreover have** ‹*Suc n* ≤ *length* (*rs*)› **if** ‹$t$ @ [✓(*rs*)] $\in \mathcal{T}\ (P\ [\![S]\!]_{\checkmark Rlist}\ Q)$›
**for** *rs*
**proof** −
**from** ‹$t \notin \mathcal{D}$ *?lhs*› **have** ‹$t \notin \mathcal{D}\ (P\ [\![S]\!]_{\checkmark Rlist}\ Q)$›
**by** (*simp add*: *Sync*$_{Rlist}$.*D-Sync*$_{ptick}$)
(*metis tickFree-mem-D-RenamingTick-iff-mem-D*
*tickFree-mem-T-RenamingTick-iff-mem-T tickFree-setinterleaves*$_{ptick}$*-iff*)
**hence** ‹$t$ @ [✓(*rs*)] $\notin \mathcal{D}\ (P\ [\![S]\!]_{\checkmark Rlist}\ Q)$›
**by** (*meson is-processT9*)
**with** ‹$t$ @ [✓(*rs*)] $\in \mathcal{T}\ (P\ [\![S]\!]_{\checkmark Rlist}\ Q)$›
**obtain** *t-P″ t-Q″* **where** £ : ‹*t-P″* $\in \mathcal{T}\ P$› ‹*t-Q″* $\in \mathcal{T}\ Q$›

```
    ‹t @ [✓(rs)] setinterleaves✓ λr s. ⌊r # s⌋ ((t-P'', t-Q''), S)›
     unfolding Sync_Rlist.Sync_ptick-projs by blast
   from £ obtain t-P''' t-Q''' r s
    where ££ : ‹rs = r # s› ‹t-P'' = t-P''' @ [✓(r)]› ‹t-Q'' = t-Q''' @ [✓(s)]›
      ‹t setinterleaves✓ λr s. ⌊r # s⌋ ((t-P''', t-Q'''), S)›
     by (auto elim: snoc-tick-setinterleaves_ptick E)
   from ‹t ∉ 𝒟 (P ⟦S⟧✓Rlist Q)› have ‹t-Q'' ∉ 𝒟 Q›
     by (simp add: Sync_Rlist.D-Sync_ptick')
      (metis £(1) ££(2−4) append.right-neutral
       front-tickFree-Nil is-processT3-TR-append is-processT9)
   have ‹s ∈ ✓s(Q)›
     by (metis £(2) ££(3) ‹t-Q'' ∉ 𝒟 Q› strict-ticks-of-memI)
   with ‹⋀rs. rs ∈ ✓s(Q) ⟹ n ≤ length rs› have ‹n ≤ length s› .
   thus ‹Suc n ≤ length rs› by (simp add: ‹rs = r # s›)
  qed
  ultimately have ‹(t, ?fun-evt (Suc i) (Suc j) −‘ X' ∪
                    ?fun-evt (Suc i) (Suc j) −‘
                    (X ∩ {✓(rs) |rs. ¬ Suc n ≤ length rs})) ∈ ℱ (P ⟦S⟧✓Rlist
Q)›
   using is-processT5-S7' by force
  also have ‹?fun-evt (Suc i) (Suc j) −‘ X' ∪
          ?fun-evt (Suc i) (Suc j) −‘ (X ∩ {✓(rs) |rs. ¬ Suc n ≤ length rs}) =
           ?fun-evt (Suc i) (Suc j) −‘ X›
   by (auto simp add: X'-def image-iff) (metis event_ptick.exhaust)
  finally show ‹(t, X) ∈ ℱ ?rhs›
   using ‹?map-evt (Suc i) (Suc j) t = t› by (auto simp add: F-Renaming)
 next
  fix t' rs assume ‹tF t'› ‹t = t' @ [✓(rs)]›
  from ∗(3) obtain t-P'' t-Q'' r s
   where ∗∗∗ : ‹rs = r # s›
    ‹t' setinterleaves✓ ?tj ((t-P'', t-Q''), S)›
    ‹t-P = t-P'' @ [✓(r)]› ‹t-Q = t-Q'' @ [✓(s)]›
    ‹tF t-P''› ‹tF t-Q''›
   by (auto elim!: snoc-tick-setinterleaves_ptick E
      simp add: ‹t = t' @ [✓(rs)]› split: if-split-asm)
     (metis ‹tF t'› tickFree-setinterleaves_ptick-iff)
  have ‹t-Q'' ∉ 𝒟 Q›
   by (metis ∗(2) ∗∗∗(4) F-imp-front-tickFree ‹t-Q ∉ 𝒟 (?RT i j Q)›
      front-tickFree-append-iff is-processT7 non-tickFree-tick
      tickFree-Nil tickFree-mem-D-RenamingTick-iff-mem-D)
  from ∗∗(1) ∗∗∗(4) obtain s' where ‹s = ?pl-τ i j s'›
   by (auto simp add: ∗∗(2) append-eq-map-conv Cons-eq-map-conv)
  with ∗∗(1) ∗∗(2)[THEN F-T] ∗∗∗(4) have ‹t-Q'' @ [✓(s')] ∈ 𝒯 Q›
   by (simp add: ∗∗∗(4) append-eq-map-conv Cons-eq-map-conv)
  (metis ∗∗∗(6) ‹t-Q'' ∉ 𝒟 Q› is-processT9 length-permute-list map-event_ptick-tickFree
      pl-τ-pl-τ strict-ticks-of-memI that(3) tickFree-map-map-event_ptick-id-eq)
  with ‹t-Q'' ∉ 𝒟 Q› have ‹s' ∈ ✓s(Q)›
   by (metis is-processT9 strict-ticks-of-memI)
  with ‹⋀rs. rs ∈ ✓s(Q) ⟹ n ≤ length rs› have ‹n ≤ length s'› .
```

**with** ‹$s$ = *?pl-τ i j s'*› **have** ‹$n \leq$ *length s*› **by** *simp*
**hence** ‹*Suc n* ≤ *length rs*› **by** (*simp add*: ‹*rs* = *r* # *s*›)

**have** ‹*?map-evt* (*Suc i*) (*Suc j*) *t setinterleaves*$_{\checkmark}$ *?tj* ((*t-P*, *?map-evt i j t-Q*), *S*)›
**by** (*simp add*: ∗∗∗(*1−3*, *4*, *6*) ‹*t* = *t'* @ [✓(*rs*)]› ‹*n* ≤ *length s*› ‹*tF t'*› *pl-τ-Cons*
*setinterleaves*$_{ptick}$*-snoc-tick tickFree-map-map-event*$_{ptick}$*-id-eq*)
**moreover have** ‹*t-P* ∈ $\mathcal{T}$ *P*› **using** ∗(*1*) *F-T* **by** *blast*
**moreover from** ∗(*2*)[*THEN F-T*] ‹*n* ≤ *length s*› **have** ‹*?map-evt i j t-Q* ∈ $\mathcal{T}$ *Q*›
**by** (*auto simp add*: *T-Renaming* ∗∗∗(*4*, *6*) *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* ‹*t-Q''* ∉ $\mathcal{D}$ *Q*› *is-processT7 is-processT9*)
**ultimately have** ‹*?map-evt* (*Suc i*) (*Suc j*) *t* ∈ $\mathcal{T}$ (*P* ⟦*S*⟧$_{\checkmark Rlist}$ *Q*)›
**by** (*auto simp add*: *Sync*$_{Rlist}$*.T-Sync*$_{ptick}$)
**with** ‹*n* ≤ *length s*› **have** ‹*t* ∈ $\mathcal{T}$ *?rhs*›
**by** (*auto simp add*: *T-Renaming* ‹*t* = *t'* @ [✓(*rs*)]› *append-eq-map-conv Cons-eq-map-conv*)
(*metis* ∗∗∗(*1*) ‹*tF t'*› *length-permute-list pl-τ-Cons pl-τ-pl-τ tickFree-map-map-event*$_{ptick}$*-id-eq*)
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› **by** (*simp add*: ‹*t* = *t'* @ [✓(*rs*)]› *tick-T-F*)
**qed**
**qed**
**qed**

**lemma** *RenamingTick-permute-list-transpose-Sync*$_{ListslenL}$ :
‹*RenamingTick P* (*permute-list* (*Transposition.transpose i j*)) $_n$⟦*S*⟧$_{\checkmark ListslenL}$ *Q*
=
*RenamingTick* (*P* $_n$⟦*S*⟧$_{\checkmark ListslenL}$ *Q*) (*permute-list* (*Transposition.transpose i j*))›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*i* < *n*› ‹*j* < *n*› **for** *P* :: ‹('*a*, '*r list*) *process*$_{ptick}$›
**proof** −
**let** *?pl* = ‹*permute-list* (*Transposition.transpose i j*)›
**let** *?fun-evt* = ‹*map-event*$_{ptick}$ *id* (*permute-list* (*Transposition.transpose i j*))›
**let** *?map-evt* = ‹*map ?fun-evt*›
**and** *?RT* = ‹λ*P*. *RenamingTick P* (*permute-list* (*Transposition.transpose i j*))›
**and** *?tj* = ‹λ*r s*. *if length r* = *n then* ⌊*r* @ *s*⌋ *else* ◊›
**note** *map-event*$_{ptick}$*-eq-iffs* [*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-eq-pl-imp* : ‹*r* = *r'*› **if** ‹*n* ≤ *length r*› **and** ‹*?pl r* = *?pl r'*› **for** *r r'* :: ‹'*r list*›
**proof** −
**from** ‹*n* ≤ *length r*› **have** ‹*n* ≤ *length* (*?pl r*)› **by** *simp*
**with** ‹*?pl r* = *?pl r'*› **have** ‹*n* ≤ *length r'*› **by** *simp*
**have** ‹*inj-on ?pl* {*r*. *n* ≤ *length r*}›
**by** (*simp add*: ‹*i* < *n*› ‹*j* < *n*› *inj-on-permute-list-transpose*)

```
  with ‹n ≤ length r› ‹n ≤ length r'› ‹?pl r = ?pl r'›
  show ‹r = r'› by (auto dest: inj-onD)
qed
have pl-pl : ‹n ≤ length r ⟹ ?pl (?pl r) = r› for r :: ‹'r list›
  by (subst permute-list-compose[symmetric])
   (metis lessThan-iff order-less-le-trans permutes-swap-id ‹i < n› ‹j < n›, simp)
have fun-evt-fun-evt :
  ‹(case e of ev a ⇒ True | ✓(r) ⇒ n ≤ length r) ⟹ ?fun-evt (?fun-evt e) = e›
  for e :: ‹('a, 'r list) event_ptick›
  by (cases e) (simp-all add: pl-pl)
have map-evt-map-evt :
  ‹(⋀e. e ∈ set t ⟹ case e of ev a ⇒ True | ✓(r) ⇒ n ≤ length r)
   ⟹ ?map-evt (?map-evt t) = t› for t :: ‹('a, 'r list) trace_ptick›
  by (induct t) (simp-all add: fun-evt-fun-evt)
from ‹i < n› ‹j < n› have pl-append :
  ‹n ≤ length r ⟹ ?pl (r @ r') = ?pl r @ r'› for r r' :: ‹'r list›
 by (simp add: list-update-append1 nth-append-left permute-list-transpose-eq-list-update)

show ‹?lhs = ?rhs›
proof (rule Process-eq-optimizedI)
  fix t assume ‹t ∈ 𝒟 ?lhs›
  then obtain u v t-P t-Q where ∗ : ‹t = u @ v› ‹tF u› ‹ftF v›
    ‹u setinterleaves✓ ?tj ((t-P, t-Q), S)›
    ‹t-P ∈ 𝒟 (?RT P) ∧ t-Q ∈ 𝒯 Q ∨ t-P ∈ 𝒯 (?RT P) ∧ t-Q ∈ 𝒟 Q›
    unfolding Sync_ListslenL.D-Sync_ptick by blast
  from tickFree-setinterleaves_ptick-iff[THEN iffD1, OF ∗(4, 2)] have ‹tF t-P›
..
  with ∗(5) have ‹t-P ∈ 𝒟 P ∧ t-Q ∈ 𝒯 Q ∨ t-P ∈ 𝒯 P ∧ t-Q ∈ 𝒟 Q›
  by (simp-all add: tickFree-mem-T-RenamingTick-iff-mem-T tickFree-mem-D-RenamingTick-iff-mem-
  with ∗(1−4) have ‹t ∈ 𝒟 (P n⟦S⟧✓ListslenL Q)›
    by (auto simp add: Sync_ListslenL.D-Sync_ptick)
  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 t1 @ t2› ‹tF t1› ‹ftF t2› ‹t1 ∈ 𝒟 (P n⟦S⟧✓ListslenL
Q)›
    by (auto simp add: D-Renaming)
  from ∗(1, 2) have ‹t = t1 @ t2›
    by (simp add: tickFree-map-map-event_ptick-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 Sync_ListslenL.D-Sync_ptick by blast
  from tickFree-setinterleaves_ptick-iff[THEN iffD1, OF ∗∗(4, 2)] have ‹tF t-P›
..
  with ∗∗(5) have ‹t-P ∈ 𝒟 (?RT P) ∧ t-Q ∈ 𝒯 Q ∨ t-P ∈ 𝒯 (?RT P) ∧ t-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** $**(1-4)$ $*(2,\ 3)$ ‹$t = t1\ @\ t2$› **show** ‹$t \in \mathcal{D}$ *?lhs*›
**by** (*auto simp add*: $Sync_{ListslenL}$.*D-Sync*$_{ptick}$ *intro*: *front-tickFree-append*)
**next**
**fix** *t X* **assume** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*› ‹$t \notin \mathcal{D}$ *?lhs*›
**then obtain** *t-P t-Q X-P X-Q*
**where** $*$ : ‹$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}$ (*?RT P*)› ‹$(t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ Q$›
‹*t setinterleaves*$_{\checkmark}$ *?tj* $((t\text{-}P,\ t\text{-}Q),\ S)$›
‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ *?tj X-P S X-Q*›
**unfolding** $Sync_{ListslenL}$.*Sync*$_{ptick}$*-projs* **by** *force*
**from** $*(2,\ 3)$ ‹$t \notin \mathcal{D}$ *?lhs*› *F-T front-tickFree-Nil*
**have** ‹*t-P* $\notin \mathcal{D}$ (*?RT P*)› **unfolding** $Sync_{ListslenL}$.*D-Sync*$_{ptick}$′ **by** *blast*
**with** $*(1)$ **obtain** *t-P′* **where** $**$ : ‹*t-P* = *?map-evt t-P′*› ‹(*t-P′*, *?fun-evt* −‘ *X-P*) $\in \mathcal{F}\ P$›
**unfolding** *Renaming-projs* **by** *blast*
**from** ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›[*THEN F-T*] **consider** ‹*tF t*› | *t′ rs* **where** ‹*tF t′*› ‹*t* = *t′* @ [✓(*rs*)]›
**using** *T-nonTickFree-imp-decomp append-T-imp-tickFree* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}$ *?rhs*›
**proof** *cases*
**assume** ‹*tF t*›
**hence** ‹*?map-evt t* = *t*›
**by** (*simp add*: *tickFree-map-map-event*$_{ptick}$*-id-eq*)
**have** ‹*?map-evt t-P′* = *t-P′*›
**using** $*(3)$ $**(1)$ ‹*tF t*› *map-event*$_{ptick}$*-tickFree tickFree-map-map-event*$_{ptick}$*-id-eq*
*tickFree-setinterleaves*$_{ptick}$*-iff* **by** *blast*
**have** ‹(*t-P*, *?fun-evt* −‘ *X-P*) $\in \mathcal{F}\ P$›
**by** (*simp add*: $**(1,2)$ ‹*?map-evt t-P′* = *t-P′*›)
**moreover have** ‹$(t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ Q$› **by** (*fact* $*(2)$)
**moreover have** ‹*t setinterleaves*$_{\checkmark}$ *?tj* $((t\text{-}P,\ t\text{-}Q),\ S)$› **by** (*fact* $*(3)$)
**moreover have** ‹*e* ∈ *?fun-evt* −‘ *X* ⟹ *e* ∈ *super-ref-Sync*$_{ptick}$ *?tj* (*?fun-evt* −‘ *X-P*) *S X-Q*› **for** *e*
**using** $*(4)$[*THEN set-mp*, *of* ‹*?fun-evt e*›]
**by** (*cases e*, *auto simp add*: *super-ref-Sync*$_{ptick}$*-def split*: *if-split-asm*)
(*metis append-eq-append-conv dual-order.refl length-permute-list pl-append*,
*metis append-eq-append-conv dual-order.refl length-permute-list pl-append*,
*metis dual-order.refl length-permute-list pl-append*)
**ultimately have** ‹$(t,$ *?fun-evt* −‘ *X*) $\in \mathcal{F}$ ($P\ n[\![S]\!]_{\checkmark ListslenL}\ Q$)›
**by** (*simp add*: $Sync_{ListslenL}$.*F-Sync*$_{ptick}$) *blast*
**with** ‹*?map-evt t* = *t*› **show** ‹$(t,\ X) \in \mathcal{F}$ *?rhs*›
**by** (*auto simp add*: *F-Renaming*)
**next**
**fix** *t′ rs* **assume** ‹*tF t′*› ‹*t* = *t′* @ [✓(*rs*)]›
**from** $*(3)$ **obtain** *t-P″ t-Q″ r s*
**where** $***$ : ‹*length r* = *n*› ‹*r* @ *s* = *rs*›
‹*t′ setinterleaves*$_{\checkmark}$ *?tj* $((t\text{-}P'',\ t\text{-}Q''),\ S)$›
‹*t-P* = *t-P″* @ [✓(*r*)]› ‹*t-Q* = *t-Q″* @ [✓(*s*)]›
**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-event_ptick-id-eq
        tickFree-setinterleaves_ptick-iff)
  have ‹case e of ev a ⇒ True | ✓(r) ⇒ n ≤ length r› if ‹e ∈ set t-P'› for e
  proof −
    have ‹tF t-P''›
      using ***(3) ‹tF t'› tickFree-setinterleaves_ptick-iff by blast
    hence ‹e ∈ set t-P'' ⟹ is-ev e› for e
      by (metis in-set-conv-decomp tickFree-Cons-iff tickFree-append-iff)
    moreover from ‹e ∈ set t-P'› have ‹?fun-evt e ∈ set t-P›
      by (simp add: **(1))
    ultimately show ‹case e of ev a ⇒ True | ✓(r) ⇒ n ≤ length r›
    using ***(1) by (cases e, auto simp add: ***(4)) (metis event_ptick.disc(2))
  qed
  with arg-cong[OF **(1), where f = ?map-evt] map-evt-map-evt
  have ‹?map-evt t-P = t-P'› by presburger
  with ** have ‹?map-evt t-P ∈ 𝒯 P› by (simp add: F-T)
  moreover have ‹t-Q ∈ 𝒯 Q› using *(2) F-T by blast
  moreover from ‹?map-evt t' setinterleaves✓ ?tj ((?map-evt t-P'', t-Q''), S)›
  have ‹?map-evt t setinterleaves✓ ?tj ((?map-evt t-P, t-Q), S)›
    by (simp add: ‹t = t' @ [✓(rs)]› ***(1, 4, 5)
        ‹?pl r @ s = ?pl rs› setinterleaves_ptick-snoc-tick)
  ultimately have ‹?map-evt t ∈ 𝒯 (P n⟦S⟧✓ListslenL Q)›
    unfolding Sync_ListslenL.T-Sync_ptick by blast
  hence ‹?map-evt (?map-evt t) ∈ 𝒯 ?rhs›
    by (auto simp add: T-Renaming)
  also have ‹?map-evt (?map-evt t) = t›
    by (simp add: ‹t = t' @ [✓(rs)]› )
      (metis ***(1, 2) ‹tF t'› dual-order.refl length-permute-list
        list.map-comp pl-append pl-pl tickFree-map-map-event_ptick-id-eq)
  finally show ‹(t, X) ∈ ℱ ?rhs› by (simp add: ‹t = t' @ [✓(rs)]› tick-T-F)
qed
next
  fix t X assume ‹(t, X) ∈ ℱ ?rhs› ‹t ∉ 𝒟 ?rhs› ‹t ∉ 𝒟 ?lhs›
  then obtain t' where * : ‹t = ?map-evt t'›
    ‹(t', ?fun-evt −‘ X) ∈ ℱ (P n⟦S⟧✓ListslenL Q)›
    unfolding Renaming-projs by blast
  from *(2) ‹t ∉ 𝒟 ?rhs› have ‹t' ∉ 𝒟 (P n⟦S⟧✓ListslenL 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-event_ptick-tickFree
    tickFree-map-map-event_ptick-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 −‘ X ⊆ super-ref-Sync_ptick ?tj X-P S X-Q›
    unfolding Sync_ListslenL.Sync_ptick-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 t' setinterleaves✓ ?tj ((?map-evt t-P, t-Q), S)*›
**by** (*metis* (*lifting*) $**(3)$ ‹*tF t'*› *tickFree-map-map-event$_{ptick}$-id-eq tickFree-setinterleaves$_{ptick}$-iff*)

**define** *X-P'* **where** ‹*X-P' ≡ X-P ∩ (range ev ∪ {✓(r) |r. length r = n})*›
**define** *X'* **where** ‹*X' ≡ X ∩ (range ev ∪ {✓(rs) |rs. n ≤ length rs})*›
**have** ‹*X-P' ⊆ X-P*› **unfolding** *X-P'-def* **by** *blast*
**with** $**(1)$ *is-processT4* **have** ‹*(t-P, X-P') ∈ ℱ P*› **by** *blast*
**moreover have** ‹*?fun-evt −‘ (?fun-evt ‘ X-P') = X-P'*›
**by** (*auto simp add*: *X-P'-def*) (*use length-eq-pl-imp* **in** *blast*)+
**moreover have** ‹*?map-evt t-P = t-P*›
**using** $**(3)$ ‹*tF t'*› *tickFree-map-map-event$_{ptick}$-id-eq tickFree-setinterleaves$_{ptick}$-iff*
**by** *blast*
**ultimately have** ‹*(t-P, ?fun-evt ‘ X-P') ∈ ℱ (?RT P)*›
**by** (*auto simp add*: *F-Renaming*)
**moreover have** ‹*e ∈ X' ⟹ e ∈ super-ref-Sync$_{ptick}$ ?tj (?fun-evt ‘ X-P') S X-Q*› **if** ‹*e ∈ X'*› **for** *e*
**using** $**(4)$[*THEN set-mp, of* ‹*?fun-evt e*›] *fun-evt-fun-evt*[*of e*]
**unfolding** *X'-def X-P'-def super-ref-Sync$_{ptick}$-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** ‹*(t, X') ∈ ℱ ?lhs*›
**by** (*simp add*: *Sync$_{ListslenL}$.F-Sync$_{ptick}$*)
(*metis* (*lifting*) $*(1)$ $**(2, 3)$ ‹*tF t'*› *subsetI tickFree-map-map-event$_{ptick}$-id-eq*)
**moreover from** ‹*t ∉ 𝒟 ?lhs*› **have** ‹*t @ [✓(rs)] ∈ 𝒯 ?lhs ⟹ n ≤ length (rs)*› **for** *rs*
**by** (*auto simp add*: *Sync$_{ListslenL}$.Sync$_{ptick}$-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** ‹*(t, X' ∪ (X ∩ {✓(rs) |rs. ¬ n ≤ length rs})) ∈ ℱ ?lhs*›
**using** *is-processT5-S7'* **by** *fastforce*
**also have** ‹*X' ∪ (X ∩ {✓(rs) |rs. ¬ n ≤ length rs}) = X*›
**by** (*simp add*: *set-eq-iff X'-def image-iff*) (*meson event$_{ptick}$.exhaust*)
**finally show** ‹*(t, X) ∈ ℱ ?lhs*› **.**
**next**
**fix** *t'' rs* **assume** ‹*tF t''*› ‹*t' = t'' @ [✓(rs)]*›
**from** $**(3)$ **obtain** *t-P' t-Q' r s*
**where** $***$ : ‹*length r = n*› ‹*r @ s = rs*›
‹*t'' setinterleaves✓ ?tj ((t-P', t-Q'), S)*›
‹*t-P = t-P' @ [✓(r)]*› ‹*t-Q = t-Q' @ [✓(s)]*›
**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-event_ptick-id-eq
          tickFree-setinterleaves_ptick-iff)
    from setinterleaves_ptick-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-Sync_ptick ?tj UNIV S UNIV› for
e
      using ∗∗(4)[THEN set-mp, of ‹?fun-evt e›]
      by (cases e) (auto simp add: super-ref-Sync_ptick-def)
    ultimately show ‹(t, X) ∈ ℱ ?lhs›
      using ∗(1) by (simp add: Sync_ListslenL.F-Sync_ptick) 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.prems(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⟧✓Llist P l2) ?g›
        by (simp add: Sync_Rlist.Sync_ptick-comm-locale-sym.Sync_ptick-commute)
      also have ‹... = ?RT (?MS L) ?g›
        by (simp add: ‹L = [l1, l2]› MultiSync_ptick-snoc[of - ‹[l1]›, simplified])
      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)›
        from is-ticks-lengthD is-ticks-length-MultiSync_ptick 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.prems(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 ∗ ‹i ≠ 0›
      show ‹L[i := L ! j, j := L ! i] =
          l0 # (l1 # L′)[i − 1 := (l1 # L′) ! (j − 1), j − 1 := (l1 # L′) ! (i
− 1)]›
        by (cases i; cases j) (simp-all add: ‹L = l0 # l1 # L′› nat.case-eq-if)
    qed
   hence ‹?MS (?pl-τ i j L) = P l0 ⟦S⟧✓Rlist ?MS (?pl-τ (i − 1) (j − 1) (l1
# L′))›
      by (simp add: MultiSyncptick-Cons)
        (metis Zero-not-Suc length-Cons length-permute-list list.size(3))
    also have ‹?MS (?pl-τ (i − 1) (j − 1) (l1 # L′)) =
          ?RT (?MS (l1 # L′)) (?pl-τ (i − 1) (j − 1))›
      by (subst Suc.hyps)
        (use ∗ ‹Suc n = length L› ‹L = l0 # l1 # L′› in simp-all)
   also have ‹P l0 ⟦S⟧✓Rlist ?RT (?MS (l1 # L′)) (?pl-τ (i − 1) (j − 1)) =
          ?RT (P l0 ⟦S⟧✓Rlist ?MS (l1 # L′)) (?pl-τ (Suc (i − 1)) (Suc (j
− 1)))›
      by (rule SyncRlist-RenamingTick-permute-list-transpose[OF ∗(2, 3)])
        (metis is-ticks-lengthD is-ticks-length-MultiSyncptick order-le-less)
    also have ‹(Suc (i − 1)) = i› using ‹i ≠ 0› by simp
    also have ‹(Suc (j − 1)) = j› using ∗(1) by linarith
    also have ‹P l0 ⟦S⟧✓Rlist ?MS (l1 # L′) = ?MS L›
      by (simp add: ‹L = l0 # l1 # L′›)
    finally have ‹?MS (?pl-τ i j L) = ?RT (?MS L) (?pl-τ i j)› .
  } note £ = this

  consider ‹i ≠ 0› | ‹j ≠ n› | ‹i = 0› ‹j = n› by argo
  thus ?case
  proof cases
    assume ‹i ≠ 0›
    from Suc.hyps(1, 3) obtain l0 l1 L′ where ‹L = l0 # l1 # L′›
      by (cases L; cases ‹tl L›) simp-all
    from £ ‹i ≠ 0› ‹i < j› Suc.prems(2, 3) Suc.hyps(3) this show ?case .
  next
    assume ‹j ≠ n›
    from Suc.hyps(1, 3) obtain l0 l1 L′ where ‹L = L′ @ [l1] @ [l0]›
      by (cases L rule: rev-cases; cases ‹butlast L› rule: rev-cases) simp-all
    hence ‹rev L = l0 # l1 # rev L′› by simp
    have ‹Suc n = length (rev L)› by (simp add: Suc.hyps(3))

   have ‹?MS (?pl-τ i j L) = ?MS (rev (?pl-τ (length L − Suc i) (length L −
Suc j) (rev L)))›
      by (subst rev-rev-ident[of L, symmetric], subst permute-list-transpose-rev)
        (simp-all add: Suc.prems(2, 3))
    also have ‹... = ?RT (?MS (?pl-τ (length L − Suc i) (length L − Suc j)
(rev L))) rev›
      by (fact MultiSyncptick-rev)
    also have ‹?pl-τ (length L − Suc i) (length L − Suc j) =
```

```
        ?pl-τ (length L − Suc j) (length L − Suc i)›
    by (simp add: transpose-commute)
  also have ‹?MS (?pl-τ (length L − Suc j) (length L − Suc i) (rev L)) =
        ?RT (?MS (rev L)) (?pl-τ (length L − Suc j) (length L − Suc i))›
    by (rule £) (use Suc.hyps(3) Suc.prems(3) ‹j ≠ n› ‹i < j›
      ‹rev L = l0 # l1 # rev L'› in auto)
  also have ‹?MS (rev L) = ?RT (?MS L) rev›
    by (fact MultiSync_ptick-rev)
  also have ‹?RT (?RT (?RT (?MS L) rev) (?pl-τ (length L − Suc j) (length
L − Suc i))) rev =
        ?RT (?MS L) (rev ∘ (?pl-τ (length L − Suc j) (length L − Suc i))
∘ rev)›
    by (simp add: RenamingTick-comp)
  also have ‹... = ?RT (?MS L) (?pl-τ j i)›
  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-MultiSync_ptick by blast
    thus ‹(rev ∘ (?pl-τ (length L − Suc j) (length L − Suc i)) ∘ rev) rs =
?pl-τ j i rs›
      by (unfold comp-def, subst rev-permute-list-transpose)
        (use Suc.prems(2, 3) in auto)
  qed
  also have ‹... = ?RT (?MS L) (?pl-τ i j)›
    by (simp add: transpose-commute)
  finally show ?case .
next
  let ?g1 = ‹λrs. if rs = [] then [] else last rs # butlast rs›
  let ?g2 = ‹λrs. drop (Suc (Suc 0)) rs @ take (Suc (Suc 0)) rs›
  let ?g3 = ‹λrs. case rs of r # s ⇒ r # (if s = [] then [] else last s # butlast
s)›
  let ?tj = ‹λr s. ⌊id r # (if s = [] then [] else last s # butlast s)⌋›
  assume ‹i = 0› ‹j = n›
  from Suc.hyps(1, 3) obtain l0 l1 L'
    where ‹L = l0 # L' @ [l1]› ‹L' ≠ []›
    by (cases L; cases ‹tl L› rule: rev-cases; force)
  have ‹?pl-τ i j L = l1 # L' @ [l0]›
    by (subst permute-list-transpose-eq-list-update)
      (use Suc.prems(3) Suc.hyps(3) ‹L = l0 # L' @ [l1]›
        in ‹auto simp add: ‹i = 0› ‹j = n››)
  hence ‹?MS (?pl-τ i j L) = P l1 ⟦S⟧✓Rlist (?MS L' ⟦S⟧✓Llist P l0)›
    by (simp add: MultiSync_ptick-Cons MultiSync_ptick-snoc ‹L' ≠ []›)
  also have ‹... = ?RT (?MS L' ⟦S⟧✓Llist P l0 ⟦S⟧✓Llist P l1) ?g1›
    by (simp only: Sync_Rlist.Sync_ptick-comm-locale-sym.Sync_ptick-commute)
  also have ‹?MS L' ⟦S⟧✓Llist P l0 ⟦S⟧✓Llist P l1 =
        (?MS L' Suc (Suc 0)⟦S⟧✓ListslenR (P l0 ⟦S⟧✓Pairlist P l1))›
    by (simp only: Sync_ListslenR-Sync_Pairlist-assoc)
  also have ‹... = ?RT (P l0 ⟦S⟧✓Pairlist P l1 Suc (Suc 0)⟦S⟧✓ListslenL
?MS L') ?g2›
```

**by** (*simp only*: $Sync_{ListslenL}.Sync_{ptick}$-*commute*)
**also have** ‹*P l0* $[\![S]\!]_{\checkmark Pairlist}$ *P l1* $_{Suc\ (Suc\ 0)}[\![S]\!]_{\checkmark ListslenL}$ *?MS L′* =
*P l0* $[\![S]\!]_{\checkmark Rlist}$ (*P l1* $[\![S]\!]_{\checkmark Rlist}$ *?MS L′*)›
**by** (*simp only*: $Sync_{Rlist}$-$Sync_{Rlist}$-*assoc*)
**also have** ‹... = *P l0* $[\![S]\!]_{\checkmark Rlist}$ *?RT* (*?MS L′* $[\![S]\!]_{\checkmark Llist}$ *P l1*) *?g1*›
**by** (*simp only*: $Sync_{Rlist}.Sync_{ptick}$-*comm-locale-sym*.$Sync_{ptick}$-*commute*)
**also have** ‹... = *?RT* (*P l0*) *id* $[\![S]\!]_{\checkmark Rlist}$ *?RT* (*?MS L′* $[\![S]\!]_{\checkmark Llist}$ *P l1*) *?g1*› **by** *simp*
**also have** ‹... = $Sync_{ptick}$-*locale*.$Sync_{ptick}$ *?tj* (*P l0*) *S* (*?MS L′* $[\![S]\!]_{\checkmark Llist}$ *P l1*)›
**proof** (*rule* $Sync_{Rlist}$.*inj-RenamingTick-*$Sync_{ptick}$-*inj-RenamingTick*)
**show** ‹*inj id*› ‹*inj* (λ*rs*. *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* $[\![S]\!]_{\checkmark Rlist}$ (*?MS L′* $[\![S]\!]_{\checkmark Llist}$ *P l1*)) *?g3*›
**by** (*subst* $Sync_{Rlist}$.*inj-on-RenamingTick-*$Sync_{ptick}$)
(*auto intro*!: *inj-onI split*: *if-split-asm*, *metis append-butlast-last-id*)
**also have** ‹*P l0* $[\![S]\!]_{\checkmark Rlist}$ (*?MS L′* $[\![S]\!]_{\checkmark Llist}$ *P l1*) = *?MS L*›
**by** (*simp add*: ‹*L* = *l0* # *L′* @ [*l1*]› $MultiSync_{ptick}$-*Cons* $MultiSync_{ptick}$-*snoc* ‹*L′* ≠ []›)
**also have** ‹*?RT* (*?RT* (*?RT* (*?MS L*) *?g3*) *?g2*) *?g1* = *?RT* (*?MS L*) (*?g1* ∘ *?g2* ∘ *?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-*$MultiSync_{ptick}$ **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* ∘ *?g2* ∘ *?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.prems*(*2*, *3*) ‹*length rs* = *length L*›)
**next**
**show** ‹(*?g1* ∘ *?g2* ∘ *?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 ‹i ≤ j› | ‹j ≤ i› by linarith
  thus ‹?RS (?pl-τ i j L) = ?RT (?RS L) (?pl-τ i j)› if ‹i < length L› ‹j < length
L›
  proof cases
    from that show ‹i ≤ j ⟹ ?RS (?pl-τ i j L) = ?RT (?RS L) (?pl-τ i j)› by
(fact ∗)
  next
    from that show ‹j ≤ i ⟹ ?RS (?pl-τ i j L) = ?RT (?RS L) (?pl-τ i j)›
      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 MultiSync_ptick-permute-list :
  ‹⟦S⟧✓ l ∈@ permute-list f L. P l =
   RenamingTick (⟦S⟧✓ l ∈@ L. P l) (permute-list f)›
  if f-permutes : ‹f permutes {..<length L}›
  using finite-lessThan f-permutes
proof (induct f rule: permutes-rev-induct)
  case id show ?case by (simp flip: id-def)
next
 let ?RT = RenamingTick and ?pl = permute-list and ?τ = Transposition.transpose
  case (swap i j f)
  have ‹?τ i j permutes {..<length L}›
    by (meson permutes-swap-id swap.hyps(1, 2))
  hence ‹⟦S⟧✓ l ∈@ ?pl (f ∘ ?τ i j) L. P l =
        ⟦S⟧✓ l ∈@ (?pl (?τ i j) (?pl f L)). P l›
    by (simp add: permute-list-compose)

  also have ‹... = ?RT (⟦S⟧✓ l ∈@ (?pl f L). P l) (?pl (?τ i j))›
    by (metis MultiSync_ptick-permute-list-transpose atLeast0LessThan
        atLeastLessThan-iff length-permute-list swap.hyps(1,2))
  also have ‹... = ?RT (?RT (⟦S⟧✓ l ∈@ L. P l) (?pl f)) (?pl (?τ i j))›
    unfolding swap.hyps(4) ..
  also have ‹... = ?RT (⟦S⟧✓ l ∈@ L. P l) (?pl (?τ i j) ∘ ?pl f)›
    by (simp flip: Renaming-comp)
  also have ‹... = ?RT (⟦S⟧✓ l ∈@ L. P l) (?pl (f ∘ (?τ i j)))›
  proof (rule RenamingTick-is-restrictable-on-strict-ticks-of)
    fix rs assume ‹rs ∈ ✓s(⟦S⟧✓ l ∈@ L. P l)›
    from is-ticks-lengthD is-ticks-length-MultiSync_ptick this
    have ‹length rs = length L› .
    with ‹?τ i j permutes {..<length L}›
    show ‹(?pl (?τ i j) ∘ ?pl f) rs = ?pl (f ∘ ?τ i j) rs›
      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* :
‹$(\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow t \notin \mathcal{D}\ P \Longrightarrow ev\ a \in set\ t \Longrightarrow tF\ t \Longrightarrow thesis) \Longrightarrow thesis$› **if** ‹$a \in \boldsymbol{\alpha}(P)$›
**proof** −
**from** ‹$a \in \boldsymbol{\alpha}(P)$› **obtain** $t$ **where** ‹$t \in \mathcal{T}\ P$› ‹$t \notin \mathcal{D}\ P$› ‹$ev\ a \in set\ t$›
**by** (*meson strict-events-of-memE*)
**have** ‹(*if tF t then t else butlast t*) $\in \mathcal{T}\ P$›
**by** *simp* (*metis* ‹$t \in \mathcal{T}\ P$› *append-butlast-last-id is-processT3-TR-append tickFree-Nil*)
**moreover have** ‹(*if tF t then t else butlast t*) $\notin \mathcal{D}\ P$›
**using** *T-imp-front-tickFree* ‹$t \in \mathcal{T}\ P$› ‹$t \notin \mathcal{D}\ P$› *div-butlast-when-non-tickFree-iff*
**by** *blast*
**moreover from** *T-nonTickFree-imp-decomp* ‹$ev\ a \in set\ t$› ‹$t \in \mathcal{T}\ P$›
**have** ‹$ev\ a \in set$ (*if tF t then t else butlast t*)› **by** *force*
**moreover from** *T-imp-front-tickFree* ‹$t \in \mathcal{T}\ P$› *front-tickFree-iff-tickFree-butlast*
**have** ‹*tF* (*if tF t then t else butlast t*)› **by** (*metis* (*full-types*))
**ultimately show** ‹$(\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow t \notin \mathcal{D}\ P \Longrightarrow ev\ a \in set\ t \Longrightarrow tF\ t \Longrightarrow thesis) \Longrightarrow thesis$› **by** *blast*
**qed**

**lemma** *events-of-memE-optimized-tickFree* :
‹$(\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow ev\ a \in set\ t \Longrightarrow tF\ t \Longrightarrow thesis) \Longrightarrow thesis$› **if** ‹$a \in \alpha(P)$›
**proof** −
**from** ‹$a \in \alpha(P)$› **obtain** $t$ **where** ‹$t \in \mathcal{T}\ P$› ‹$ev\ a \in set\ t$›
**by** (*meson events-of-memE*)
**have** ‹(*if tF t then t else butlast t*) $\in \mathcal{T}\ P$›
**by** *simp* (*metis* ‹$t \in \mathcal{T}\ P$› *append-butlast-last-id is-processT3-TR-append tickFree-Nil*)
**moreover from** *T-nonTickFree-imp-decomp* ‹$ev\ a \in set\ t$› ‹$t \in \mathcal{T}\ P$›
**have** ‹$ev\ a \in set$ (*if tF t then t else butlast t*)› **by** *force*
**moreover from** *T-imp-front-tickFree* ‹$t \in \mathcal{T}\ P$› *front-tickFree-iff-tickFree-butlast*
**have** ‹*tF* (*if tF t then t else butlast t*)› **by** (*metis* (*full-types*))

**ultimately show** ‹$(\bigwedge t.\ t \in \mathcal{T}\ P \Longrightarrow ev\ a \in set\ t \Longrightarrow tF\ t \Longrightarrow thesis) \Longrightarrow thesis$›
**by** *blast*
**qed**

## 12.2 Sequential Composition

### 12.2.1 Events

**lemma** *events-of-Seq*$_{ptick}$ : ‹$\alpha(P\ ;_{\checkmark}\ Q) = \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$›
**proof** (*intro subset-antisym subsetI*)
**show** ‹$a \in \alpha(P\ ;_{\checkmark}\ Q) \Longrightarrow a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$› **for** *a*
**proof** (*elim events-of-memE*)
**fix** *t* **assume** ‹$t \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)$› ‹$ev\ a \in set\ t$›
**from** *this*(*1*) **consider** (*T-P*) *t'* **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'$› ‹$t' \in \mathcal{T}\ P$› ‹$tF\ t'$›
| (*T-Q*) *t' r u* **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t' @ u$› ‹$t' @ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$tF\ t'$› ‹$u \in \mathcal{T}\ (Q\ r)$›
| (*D-P*) *t' u* **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t' @ u$› ‹$t' \in \mathcal{D}\ P$› ‹$tF\ t'$› ‹$ftF\ u$›
**unfolding** *Seq*$_{ptick}$*-projs* **by** *blast*
**thus** ‹$a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$›
**proof** *cases*
**case** *T-P*
**from** *T-P*(*1*, *3*) ‹$ev\ a \in set\ t$› **have** ‹$ev\ a \in set\ t'$›
**by** (*meson tickFree-map-ev-of-ev-eq-imp-ev-mem-iff*)
**with** *T-P*(*2*) **have** ‹$a \in \alpha(P)$› **by** (*rule events-of-memI*)
**thus** ‹$a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$› **by** *simp*
**next**
**case** *T-Q*
**have** ‹$r \in \checkmark\mathbf{s}(P) \vee \mathcal{D}\ P \neq \{\}$›
**by** (*metis T-Q*(*2*) *empty-iff strict-ticks-of-memI*)
**thus** ‹$a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$›
**proof** (*elim disjE*)
**from** *T-Q*
**show** ‹$r \in \checkmark\mathbf{s}(P) \Longrightarrow a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$›
**by** *simp* (*metis Un-iff* ‹$ev\ a \in set\ t$› *events-of-memI set-append tickFree-map-ev-of-ev-eq-imp-ev-mem-iff*)
**next**
**assume** ‹$\mathcal{D}\ P \neq \{\}$›
**hence** ‹$\alpha(P) = UNIV$› **by** (*simp add*: *events-of-is-strict-events-of-or-UNIV*)
**thus** ‹$a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$› **by** *simp*
**qed**
**next**
**case** *D-P*
**have** ‹$\alpha(P) = UNIV$›
**by** (*metis D-P*(*2*) *empty-iff events-of-is-strict-events-of-or-UNIV*)
**thus** ‹$a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r))$› **by** *simp*
**qed**
**qed**
**next**

**show** ‹$a \in \alpha(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\ \alpha(Q\ r)) \Longrightarrow a \in \alpha(P\ ;_{\checkmark}\ Q)$› **for** $a$
**proof** (*elim UnE UnionE events-of-memE*, *safe*)
**fix** $t$ **assume** ‹$t \in \mathcal{T}\ P$› ‹*ev a ∈ set t*›
**then obtain** $t'$ **where** ‹$t' \in \mathcal{T}\ P$› ‹*tF t'*› ‹*ev a ∈ set t'*›
**by** (*cases t rule*: *rev-cases*, *simp-all*)
(*metis prefixI ‹ev a ∈ set t› append-T-imp-tickFree event*$_{ptick}$*.disc(1) is-processT3-TR not-Cons-self2 tickFree-Cons-iff tickFree-Nil tickFree-append-iff*)
**thus** ‹$a \in \alpha(P\ ;_{\checkmark}\ Q)$› **by** (*auto simp add*: *Seq*$_{ptick}$*-projs rev-image-eqI intro*!: *events-of-memI*)
**next**
**fix** $a\ r$ **assume** ‹$a \in \alpha(Q\ r)$› ‹$r \in \checkmark\mathbf{s}(P)$›
**from** ‹$r \in \checkmark\mathbf{s}(P)$› **obtain** $t$ **where** ‹$t\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› **by** (*meson strict-ticks-of-memD*)
**moreover from** ‹$a \in \alpha(Q\ r)$› **obtain** $u$
**where** ‹$u \in \mathcal{T}\ (Q\ r)$› ‹*ev a ∈ set u*› **by** (*meson events-of-memD*)
**ultimately have** ‹*map* (*ev* ∘ *of-ev*) $t\ @\ u \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)$› ‹*ev a ∈ set* (*map* (*ev* ∘ *of-ev*) $t\ @\ u$)›
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs*) (*metis append-T-imp-tickFree not-Cons-self2*)
**thus** ‹$a \in \alpha(P\ ;_{\checkmark}\ Q)$› **by** (*simp add*: *events-of-memI*)
**qed**
**qed**

— Big approximation.
**lemma** *events-of-Seq*$_{ptick}$*-subset* : ‹$\alpha(P\ ;_{\checkmark}\ Q) \subseteq \alpha(P) \cup (\bigcup r.\ \alpha(Q\ r))$›
**by** (*auto simp add*: *events-of-Seq*$_{ptick}$)

— Big approximation.
**corollary** *events-of-Seq-subset* : ‹$\alpha(P\ ;\ Q) \subseteq \alpha(P) \cup \alpha(Q)$›
**by** (*simp add*: *events-of-Seq*)

**lemma** *strict-events-of-Seq*$_{ptick}$*-subset* : ‹$\boldsymbol{\alpha}(P\ ;_{\checkmark}\ Q) \subseteq \boldsymbol{\alpha}(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\boldsymbol{\alpha}(Q\ r))$›
**proof** (*rule subsetI*)
**show** ‹$a \in \boldsymbol{\alpha}(P\ ;_{\checkmark}\ Q) \Longrightarrow a \in \boldsymbol{\alpha}(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\boldsymbol{\alpha}(Q\ r))$› **for** $a$
**proof** (*elim strict-events-of-memE*)
**fix** $t$ **assume** ‹$t \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)$› ‹$t \notin \mathcal{D}\ (P\ ;_{\checkmark}\ Q)$› ‹*ev a ∈ set t*›
**from** *this*(*1*, *2*) **consider** (*T-P*) $t'$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'$› ‹$t' \in \mathcal{T}\ P$› ‹$t' \notin \mathcal{D}\ P$› ‹*tF t'*›
| (*T-Q*) $t'\ r\ u$ **where** ‹$t = map\ (ev \circ of\text{-}ev)\ t'\ @\ u$› ‹$t'\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$t' \notin \mathcal{D}\ P$› ‹*tF t'*› ‹$u \in \mathcal{T}\ (Q\ r)$› ‹$u \notin \mathcal{D}\ (Q\ r)$›
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs*) (*metis T-imp-front-tickFree*)
**thus** ‹$a \in \boldsymbol{\alpha}(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\boldsymbol{\alpha}(Q\ r))$›
**proof** *cases*
**case** *T-P*
**have** ‹*ev a ∈ set t'*›
**by** (*metis T-P*(*1*, *4*) ‹*ev a ∈ set t*› *tickFree-map-ev-of-ev-eq-imp-ev-mem-iff*)
**have** ‹$a \in \boldsymbol{\alpha}(P)$›

**by** (*meson T-P*(*2*, *3*) ‹*ev a ∈ set t'*› *strict-events-of-memI*)
**thus** ‹$a \in \boldsymbol{\alpha}(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\boldsymbol{\alpha}(Q\ r))$› **by** *simp*
**next**
**case** *T-Q*
**have** ‹$r \in \checkmark\mathbf{s}(P)$› **by** (*meson T-Q*(*2*, *3*) *is-processT9 strict-ticks-of-memI*)
**thus** ‹$a \in \boldsymbol{\alpha}(P) \cup (\bigcup r \in \checkmark\mathbf{s}(P).\boldsymbol{\alpha}(Q\ r))$›
**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-Seq*$_{ptick}$ :
‹$\checkmark s(P\ ;_{\checkmark}\ Q) = (\mathit{if}\ \mathcal{D}\ P = \{\}\ \mathit{then}\ (\bigcup r \in \checkmark\mathbf{s}(P).\ \checkmark s(Q\ r))\ \mathit{else}\ \mathit{UNIV})$›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$\mathcal{D}\ P \neq \{\} \Longrightarrow \checkmark s(P\ ;_{\checkmark}\ Q) = \mathit{UNIV}$›
**by** (*simp add*: *Seq*$_{ptick}$*-projs ticks-of-is-strict-ticks-of-or-UNIV*)
(*metis front-tickFree-Nil nonempty-divE*)
**next**
**show** ‹$\mathcal{D}\ P = \{\} \Longrightarrow \checkmark s(P\ ;_{\checkmark}\ Q) = (\bigcup r{\in}\checkmark\mathbf{s}(P).\ \checkmark s(Q\ r))$› **if** ‹$\mathcal{D}\ P = \{\}$›
**proof** (*intro subset-antisym subsetI*)
**from** ‹$\mathcal{D}\ P = \{\}$› *ticks-of-memI*[*of* - - ‹$Q$ -›]
**show** ‹$s \in \checkmark s(P\ ;_{\checkmark}\ Q) \Longrightarrow s \in (\bigcup r{\in}\checkmark\mathbf{s}(P).\ \checkmark s(Q\ r))$› **for** *s*
**by** (*auto simp add*: *Seq*$_{ptick}$*-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** ‹$s \in (\bigcup r{\in}\checkmark\mathbf{s}(P).\ \checkmark s(Q\ r)) \Longrightarrow s \in \checkmark s(P\ ;_{\checkmark}\ Q)$› **for** *s*
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs ticks-of-def elim*!: *strict-ticks-of-memE*)
(*meson append.assoc append-T-imp-tickFree not-Cons-self2*)
**qed**
**qed**

**lemma** ‹$\checkmark\mathbf{s}(P\ ;_{\checkmark}\ Q) \subseteq \bigcup\ \{\checkmark\mathbf{s}(Q\ r)\ |r.\ r \in \checkmark\mathbf{s}(P)\}$›
— Already proven earlier in the construction.
**by** (*fact strict-ticks-of-Seq*$_{ptick}$*-subset*)

## 12.3 Synchronization Product

### 12.3.1 Events

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *events-of-Sync*$_{ptick}$*-subset* : ‹$\alpha(P\ [\![S]\!]_{\checkmark}\ Q) \subseteq \alpha(P) \cup \alpha(Q)$›
**by** (*subst events-of-def*, *simp add*: *T-Sync*$_{ptick}$ *subset-iff*)
(*metis UNIV-I empty-iff events-of-is-strict-events-of-or-UNIV events-of-memI setinterleaves*$_{ptick}$*-preserves-ev-notin-set*)

**lemma** (**in** $Sync_{ptick}$-*locale*) *events-of-Inter*$_{ptick}$: ‹$\alpha(P \mid\mid\mid_{\checkmark} Q) = \alpha(P) \cup \alpha(Q)$›
**proof** (*rule subset-antisym*[*OF events-of-Sync*$_{ptick}$-*subset*])
**show** ‹$\alpha(P) \cup \alpha(Q) \subseteq \alpha(P \mid\mid\mid_{\checkmark} Q)$›
**proof** (*rule subsetI*, *elim UnE*)
**fix** *a* **assume** ‹$a \in \alpha(P)$›
**then obtain** *t-P* **where** ‹*tF t-P*› ‹*ev a* ∈ *set t-P*› ‹*t-P* ∈ $\mathcal{T}$ *P*›
**by** (*meson events-of-memE-optimized-tickFree*)
**have** ‹*map ev* (*map of-ev t-P*) *setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P*, []), {})›
**by** (*simp add*: ‹*tF t-P*› *setinterleaves*$_{ptick}$-*NilR-iff*)
**hence** ‹*map ev* (*map of-ev t-P*) ∈ $\mathcal{T}$ (*P* $\mid\mid\mid_{\checkmark}$ *Q*)›
**by** (*simp add*: *T-Sync*$_{ptick}$) (*metis* ‹*t-P* ∈ $\mathcal{T}$ *P*› *is-processT1-TR*)
**moreover from** ‹*ev a* ∈ *set t-P*› **have** ‹*ev a* ∈ *set* (*map ev* (*map of-ev t-P*))›
**by** *force*
**ultimately show** ‹$a \in \alpha(P \mid\mid\mid_{\checkmark} Q)$› **by** (*metis events-of-memI*)
**next**
**fix** *a* **assume** ‹$a \in \alpha(Q)$›
**then obtain** *t-Q* **where** ‹*tF t-Q*› ‹*ev a* ∈ *set t-Q*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*›
**by** (*meson events-of-memE-optimized-tickFree*)
**have** ‹*map ev* (*map of-ev t-Q*) *setinterleaves*$_{\checkmark\,tick\text{-}join}$ (([], *t-Q*), {})›
**by** (*simp add*: ‹*tF t-Q*› *setinterleaves*$_{ptick}$-*NilL-iff*)
**hence** ‹*map ev* (*map of-ev t-Q*) ∈ $\mathcal{T}$ (*P* $\mid\mid\mid_{\checkmark}$ *Q*)›
**by** (*simp add*: *T-Sync*$_{ptick}$) (*metis* ‹*t-Q* ∈ $\mathcal{T}$ *Q*› *is-processT1-TR*)
**moreover from** ‹*ev a* ∈ *set t-Q*› **have** ‹*ev a* ∈ *set* (*map ev* (*map of-ev t-Q*))›
**by** *force*
**ultimately show** ‹$a \in \alpha(P \mid\mid\mid_{\checkmark} Q)$› **by** (*metis events-of-memI*)
**qed**
**qed**

**lemma** (**in** $Sync_{ptick}$-*locale*) *strict-events-of-Sync*$_{ptick}$-*subset* :
‹$\boldsymbol{\alpha}(P \llbracket S \rrbracket_{\checkmark} Q) \subseteq \boldsymbol{\alpha}(P) \cup \boldsymbol{\alpha}(Q)$›
**proof** (*rule subsetI*)
**fix** *a* **assume** ‹$a \in \boldsymbol{\alpha}(P \llbracket S \rrbracket_{\checkmark} Q)$›
**then obtain** *t* **where** ‹$t \in \mathcal{T}$ (*P* $\llbracket S \rrbracket_{\checkmark}$ *Q*)› ‹*ev a* ∈ *set t*› ‹*tF t*› ‹$t \notin \mathcal{D}$ (*P* $\llbracket S \rrbracket_{\checkmark}$ *Q*)›
**by** (*blast elim*: *strict-events-of-memE-optimized-tickFree*)
**from** ‹$t \in \mathcal{T}$ (*P* $\llbracket S \rrbracket_{\checkmark}$ *Q*)› ‹$t \notin \mathcal{D}$ (*P* $\llbracket S \rrbracket_{\checkmark}$ *Q*)›
**obtain** *t-P t-Q* **where** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*›
**and** *setinter* : ‹*t setinterleaves*$_{\checkmark\,tick\text{-}join}$ ((*t-P*, *t-Q*), *S*)›
**unfolding** *Sync*$_{ptick}$-*projs* **by** *blast*
**from** *this*(3) *setinterleaves*$_{ptick}$-*preserves-ev-notin-set* ‹*ev a* ∈ *set t*›
**have** ‹*ev a* ∈ *set t-P* ∨ *ev a* ∈ *set t-Q*› **by** *metis*
**moreover have** ‹*t-P* ∉ $\mathcal{D}$ *P* ∧ *t-Q* ∉ $\mathcal{D}$ *Q*›
**proof** (*rule ccontr*)
**assume** ‹¬ (*t-P* ∉ $\mathcal{D}$ *P* ∧ *t-Q* ∉ $\mathcal{D}$ *Q*)›
**with** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› ‹*tF t*› *front-tickFree-Nil setinter*
**have** ‹*t* ∈ $\mathcal{D}$ (*P* $\llbracket S \rrbracket_{\checkmark}$ *Q*)› **unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**with** ‹*t* ∉ $\mathcal{D}$ (*P* $\llbracket S \rrbracket_{\checkmark}$ *Q*)› **show** *False* **..**

**qed**
**ultimately show** ‹$a \in \boldsymbol{\alpha}(P) \cup \boldsymbol{\alpha}(Q)$›
**by** (*meson UnCI* ‹*t-P* $\in \mathcal{T}$ *P*› ‹*t-Q* $\in \mathcal{T}$ *Q*› *strict-events-of-memI*)
**qed**

### 12.3.2 Ticks

**lemma** (**in** *Sync$_{ptick}$-locale*)
‹$\checkmark\mathbf{s}(P \llbracket S \rrbracket_{\checkmark} Q) \subseteq \{r\text{-}s \mid r\text{-}s\ r\ s.\ r \otimes\checkmark\ s = \mathit{Some}\ r\text{-}s \wedge r \in \checkmark\mathbf{s}(P) \wedge s \in \checkmark\mathbf{s}(Q)\}$›
— Already proven earlier in the construction.
**by** (*fact strict-ticks-of-Sync$_{ptick}$-subset*)

**lemma** (**in** *Sync$_{ptick}$-locale*) *ticks-of-no-div-Sync$_{ptick}$-subset* :
‹$\mathcal{D}\ (P \llbracket S \rrbracket_{\checkmark} Q) = \{\} \Longrightarrow$
$\checkmark s(P \llbracket S \rrbracket_{\checkmark} Q) \subseteq \{r\text{-}s \mid r\text{-}s\ r\ s.\ \mathit{tick\text{-}join}\ r\ s = \mathit{Some}\ r\text{-}s \wedge r \in \checkmark s(P) \wedge s \in \checkmark s(Q)\}$›
**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* :

‹$\alpha(\mathit{SEQ}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \alpha(P\ l))$›
**by** (*induct L rule*: *rev-induct*)
(*auto intro*!: *subset-trans*[*OF events-of-Seq-subset*])

**lemma** *events-of-MultiSeq$_{ptick}$-subset* :
‹$\alpha((\mathit{SEQ}_{\checkmark}\ l \in@\ L.\ P\ l)\ r) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \alpha(P\ l\ r))$›
**by** (*induct L arbitrary*: *r*)
(*auto intro*!: *subset-trans*[*OF events-of-Seq$_{ptick}$-subset*])

**lemma** *strict-events-of-MultiSeq-subset* :

‹$\boldsymbol{\alpha}(\mathit{SEQ}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \boldsymbol{\alpha}(P\ l))$›
**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* :
‹$\boldsymbol{\alpha}((\mathit{SEQ}_{\checkmark}\ l \in@\ L.\ P\ l)\ r) \subseteq (\bigcup l \in \mathit{set}\ L.\ \bigcup r.\ \boldsymbol{\alpha}(P\ l\ r))$›
**by** (*induct L arbitrary*: *r*, *simp*)
(*auto intro*!: *subset-trans*[*OF strict-events-of-Seq$_{ptick}$-subset*])

**lemma** *events-of-MultiSync$_{ptick}$-subset* :

$\langle \alpha([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in set\ L.\ \alpha(P\ l))\rangle$
**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* $Sync_{Rlist}$.$events\text{-}of\text{-}Sync_{ptick}$-*subset* **in** *fastforce*)

**lemma** $events\text{-}of\text{-}MultiInter_{ptick}$ :
$\langle \alpha(|||_{\checkmark}\ l \in@\ L.\ P\ l) = (\bigcup l \in set\ L.\ \alpha(P\ l))\rangle$
**by** (*induct L rule*: *induct-list012*, *simp-all add*: $Sync_{Rlist}$.$events\text{-}of\text{-}Inter_{ptick}$)
(*metis events-of-Renaming events-of-is-strict-events-of-or-UNIV id-apply image-id*)

**lemma** $strict\text{-}events\text{-}of\text{-}MultiSync_{ptick}$-*subset* :
$\langle \boldsymbol{\alpha}([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l) \subseteq (\bigcup l \in set\ L.\ \boldsymbol{\alpha}(P\ l))\rangle$
**by** (*induct L rule*: *induct-list012*, *simp-all add*: *strict-events-of-inj-on-Renaming*)
(*use* $Sync_{Rlist}$.$strict\text{-}events\text{-}of\text{-}Sync_{ptick}$-*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** $strict\text{-}ticks\text{-}of\text{-}MultiSeq_{ptick}$-*subset* :
$\langle \checkmark\mathbf{s}((SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r) \subseteq (if\ L = []\ then\ \{r\}\ else\ (\bigcup r.\ \checkmark\mathbf{s}(P\ (last\ L)\ r)))\rangle$
**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** $\langle \checkmark\mathbf{s}(\ldots) \subseteq \bigcup\ \{\checkmark\mathbf{s}((SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r')\ |r'.\ r' \in \checkmark\mathbf{s}(P\ l\ r)\}\rangle$
**by** (*fact strict-ticks-of-Seq*$_{ptick}$-*subset*)
**also have** $\langle \ldots \subseteq \bigcup\ \{if\ L = []\ then\ \{r'\}\ else \bigcup r.\ \checkmark\mathbf{s}(P\ (last\ L)\ r)\ |r'.\ r' \in \checkmark\mathbf{s}(P\ l\ r)\}\rangle$
**using** *Cons.hyps* **by** (*blast intro*: *Union-subsetI*)
**also have** $\langle \ldots \subseteq (if\ l\ \#\ L = []\ then\ \{r\}\ else \bigcup r.\ \checkmark\mathbf{s}(P\ (last\ (l\ \#\ L))\ r))\rangle$ **by** *auto*
**finally show** *?case* **.**
**qed**

**lemma** $strict\text{-}ticks\text{-}of\text{-}MultiSeq$-*subset* :
$\langle \checkmark\mathbf{s}(SEQ\ l \in@\ L.\ P\ l) \subseteq (if\ L = []\ then\ \{undefined\}\ else\ (\bigcup r.\ \checkmark\mathbf{s}(P\ (last\ L))))\rangle$
**using** $strict\text{-}ticks\text{-}of\text{-}MultiSeq_{ptick}$-*subset*[*of L* $\langle \lambda l\ r.\ P\ l\rangle$]
**unfolding** $MultiSeq_{ptick}$-*const* **by** *auto*

**lemma** $strict\text{-}ticks\text{-}of\text{-}MultiSync_{ptick}$-*subset* :
$\langle \checkmark\mathbf{s}([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l) \subseteq$
$\{l.\ length\ l = length\ L \wedge (\forall i < length\ L.\ l\ !\ i \in \checkmark\mathbf{s}(P\ (L\ !\ i)))\}\rangle$

**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** ‹$[\![S]\!]_{\checkmark}$ *l* ∈@ (*l0* # *l1* # *L*). *P l* = *P l0* $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l*› **by** *simp*
**also have** ‹✓**s**(...) ⊆ {*r* # *s* |*r s*. *r* ∈ ✓**s**(*P l0*) ∧ *s* ∈ ✓**s**($[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l*)}›
**by** (*rule subset-trans*[*OF* $Sync_{Rlist}$.*strict-ticks-of-*$Sync_{ptick}$-*subset*]) *blast*
**also have** ‹... ⊆
{*r* # *s* |*r s*. *r* ∈ ✓**s**(*P l0*) ∧
*s* ∈ {*l*. *length l* = *length* (*l1* # *L*) ∧
(∀ *i*<*length* (*l1* # *L*). *l* ! *i* ∈ ✓**s**(*P* ((*l1* # *L*) ! *i*)))}}›
**using** *3.hyps*(*2*) **by** *blast*
**also have** ‹... = {*l*. *length l* = *length* (*l0* # *l1* # *L*) ∧
(∀ *i*<*length* (*l0* # *l1* # *L*). *l* ! *i* ∈ ✓**s**(*P* ((*l0* # *l1* # *L*) ! *i*)))}›
(**is** ‹*?S1* = *?S2*›)
**proof** (*unfold set-eq-iff*, *intro allI*)
**show** ‹*l* ∈ *?S1* ⟷ *l* ∈ *?S2*› **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* : ‹$t \in$ *min-elems* ($\mathcal{D}$ $P$) $\Longrightarrow$ *tF* $t$›
**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*$_{ptick}$ : ‹$P$ ;$_{\checkmark}$ $R \sqsubseteq Q$ ;$_{\checkmark}$ $S$› **if** ‹$P \sqsubseteq Q$› **and** ‹$R \sqsubseteq S$›
**for** $P$ $Q$ :: ‹('$a$, '$r$) *process*$_{ptick}$› **and** $R$ $S$ :: ‹'$r \Rightarrow$ ('$a$, '$s$) *process*$_{ptick}$›
**proof** −
**let** *?S* = ‹$\lambda P\ R$. *map* (*ev* ∘ *of-ev*) ‘ *min-elems* ($\mathcal{D}$ $P$) ∪
{*map* (*ev* ∘ *of-ev*) $t$ @ $u$| $t\ r\ u$. $t$ @ [✓($r$)] ∈ $\mathcal{T}$ $P$ ∧ $t \notin \mathcal{D}$ $P$ ∧
*tF* $t$ ∧ $u \in$ *min-elems* ($\mathcal{D}$ ($R$ $r$))}›
{ **fix** $P$ **and** $R$ :: ‹'$r \Rightarrow$ ('$a$, '$s$) *process*$_{ptick}$› **and** $t$
**assume** ‹$t \in$ *min-elems* ($\mathcal{D}$ ($P$ ;$_{\checkmark}$ $R$))›
**hence** ∗ : ‹$t \in \mathcal{D}$ ($P$ ;$_{\checkmark}$ $R$)› **and** ∗∗ : ‹$\bigwedge t'$. $t' \in \mathcal{D}$ ($P$ ;$_{\checkmark}$ $R$) $\Longrightarrow \neg\ t' < t$›
**by** (*simp-all add*: *min-elems-def*)
**from** ∗ **consider** (*D-P*) $t'$ $u$ **where** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$ @ $u$› ‹$t' \in \mathcal{D}$ $P$› ‹*tF* $t'$› ‹*ftF* $u$›
| (*D-R*) $t'$ $r$ $u$ **where** ‹$t$ = *map* (*ev* ∘ *of-ev*) $t'$ @ $u$› ‹$t'$ @ [✓($r$)] ∈ $\mathcal{T}$ $P$› ‹$t' \notin \mathcal{D}$ $P$› ‹*tF* $t'$› ‹$u \in \mathcal{D}$ ($R$ $r$)›
**by** (*simp add*: *Seq*$_{ptick}$-*projs*) (*metis D-imp-front-tickFree*)
**hence** ‹$t \in$ *?S* $P$ $R$›
**proof** *cases*
**case** *D-P*
**from** *D-P*(*1*−*3*) ∗∗[*of* ‹*map* (*ev* ∘ *of-ev*) $t'$›] **have** ‹$u$ = []›
**by** (*simp add*: *Seq*$_{ptick}$-*projs*)
(*metis strict-prefixI' append.right-neutral front-tickFree-Nil neq-Nil-conv*)
**have** ‹$t' \in$ *min-elems* ($\mathcal{D}$ $P$)›
**proof** (*rule ccontr*)
**assume** ‹$t' \notin$ *min-elems* ($\mathcal{D}$ $P$)›
**with** *D-P*(*2*) **obtain** $t''$ **where** ‹$t'' \in \mathcal{D}$ $P$› ‹$t'' < t'$› **unfolding** *min-elems-def*

**by** *fast*
**with** *D-P*(*1*, *3*) ∗∗[*of* ‹*map* (*ev* ∘ *of-ev*) *t''*›] **show** *False*
**by** (*auto simp add*: *Seq*$_{ptick}$*-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*: *Seq*$_{ptick}$*-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** *Seq*$_{ptick}$*-projs* **by** *blast*
**next**
**from** *le-approx2*[*OF* ‹*P* ⊑ *Q*›] *le-approx2T*[*OF* ‹*P* ⊑ *Q*›]
**show** ‹*t* ∉ *D* (*P* ;$_{✓}$ *R*) ⟹ *X* ∈ $\mathcal{R}_a$ (*P* ;$_{✓}$ *R*) *t* ⟹ *X* ∈ $\mathcal{R}_a$ (*Q* ;$_{✓}$ *R*) *t*› **for** *t X*
**by** (*simp add*: *Seq*$_{ptick}$*-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* ∈ $\mathcal{R}_a$ (*Q* ;$_{✓}$ *R*) *t* ⟹ *X* ∈ $\mathcal{R}_a$ (*P* ;$_{✓}$ *R*) *t*› **for** *t X*
**by** (*simp add*: *subset-iff Seq*$_{ptick}$*-projs Refusals-after-def*)
(*metis D-T is-processT8*)
**next**

**show** ‹*t* ∈ *min-elems* (*D* (*P* ;$_{✓}$ *R*)) ⟹ *t* ∈ $\mathcal{T}$ (*Q* ;$_{✓}$ *R*)› **for** *t*
**proof** (*rule set-mp*[*OF* - $])
**from** *le-approx2T*[*OF* ‹*P* ⊑ *Q*›] *le-approx3*[*OF* ‹*P* ⊑ *Q*›] **show** ‹*?S P R* ⊆ $\mathcal{T}$ (*Q* ;$_{✓}$ *R*)›
**by** (*simp add*: *subset-iff Seq*$_{ptick}$*-projs*)
(*meson D-T elem-min-elems image-iff is-processT9 tickFree-mem-min-elems-D*)

**qed**
**qed**
**next**
**show** ‹$Q$ ;✓ $R \sqsubseteq Q$ ;✓ $S$›
**proof** (*unfold le-approx-def*, *safe*)
**from** *le-approx1*[*OF fun-belowD*[*OF* ‹$R \sqsubseteq S$›]]
**show** ‹$t \in \mathcal{D}\ (Q$ ;✓ $S) \Longrightarrow t \in \mathcal{D}\ (Q$ ;✓ $R)$› **for** $t$
**unfolding** $Seq_{ptick}$-*projs* **by** *blast*
**next**
**from** *proc-ord2a*[*OF fun-belowD*[*OF* ‹$R \sqsubseteq S$›]]
**show** ‹$t \notin \mathcal{D}\ (Q$ ;✓ $R) \Longrightarrow X \in \mathcal{R}_a\ (Q$ ;✓ $R)\ t \Longrightarrow X \in \mathcal{R}_a\ (Q$ ;✓ $S)\ t$›
‹$t \notin \mathcal{D}\ (Q$ ;✓ $R) \Longrightarrow X \in \mathcal{R}_a\ (Q$ ;✓ $S)\ t \Longrightarrow X \in \mathcal{R}_a\ (Q$ ;✓ $R)\ t$› **for** $t\ X$
**by** (*simp add*: $Seq_{ptick}$-*projs Refusals-after-def*, *metis*)+
**next**
**show** ‹$t \in$ *min-elems* $(\mathcal{D}\ (Q$ ;✓ $R)) \Longrightarrow t \in \mathcal{T}\ (Q$ ;✓ $S)$› **for** $t$
**proof** (*rule set-mp*[*OF* - $])
**from** *le-approx3*[*OF fun-belowD*[*OF* ‹$R \sqsubseteq S$›]] **show** ‹*?S Q R* $\subseteq \mathcal{T}\ (Q$ ;✓ $S)$›
**by** (*simp add*: *subset-iff* $Seq_{ptick}$-*projs*)
(*meson D-T elem-min-elems image-iff tickFree-mem-min-elems-D*)
**qed**
**qed**
**qed**
**qed**

### 13.1.2 Preliminaries

**context begin**

**private lemma** $chain\text{-}Seq_{ptick}$-*left*: ‹*chain* $Y \Longrightarrow$ *chain* $(\lambda i.\ Y\ i$ ;✓ $S)$›
**by** (*simp add*: $mono\text{-}Seq_{ptick}$ *po-class.chain-def*)

**private lemma** $chain\text{-}Seq_{ptick}$-*right*: ‹*chain* $Y \Longrightarrow$ *chain* $(\lambda i.\ S$ ;✓ $Y\ i)$›
**by** (*simp add*: $mono\text{-}Seq_{ptick}$ *po-class.chain-def*)

**private lemma** $cont\text{-}left\text{-}prem\text{-}Seq_{ptick}$ :
‹$(\bigsqcup i.\ Y\ i)$ ;✓ $S = (\bigsqcup i.\ Y\ i$ ;✓ $S)$› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*chain* $Y$›
— We have to add this hypothesis in the generalization.
**proof** (*rule Process-eq-optimizedI*)
**show** ‹$t \in \mathcal{D}$ *?lhs* $\Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** $t$
**by** (*simp add*: $Seq_{ptick}$-*projs limproc-is-thelub ch2ch-fun* ‹*chain* $Y$› *lub-fun* $chain\text{-}Seq_{ptick}$-*left LUB-projs*) *blast*
**next**
**have** ‹$t \in \mathcal{D}$ *?lhs*› **if** ‹$t \in \mathcal{D}$ *?rhs*› **and** ‹*tF t*› **for** $t$
**proof** (*cases* ‹*map* $(ev \circ of\text{-}ev)\ t \in \mathcal{D}\ (\bigsqcup i.\ Y\ i)$›)
**show** ‹*map* $(ev \circ of\text{-}ev)\ t \in \mathcal{D}\ (\bigsqcup i.\ Y\ i) \Longrightarrow t \in \mathcal{D}$ *?lhs*›
**by** (*simp add*: $Seq_{ptick}$-*projs*)
(*metis append.right-neutral front-tickFree-Nil* ‹*tF t*› *tickFree-map-ev-comp*

*tickFree-map-ev-of-ev-eq-iff*)
**next**
**define** *T1* **and** *T2*
**where** ‹*T1 i* ≡ {*t1*. ∃ *t2*. *t* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2* ∧ *t1* ∈ 𝒟 (*Y i*) ∧ *tF t1* ∧ *ftF t2*}›
**and** ‹*T2 i* ≡ {*t1*. ∃ *t2 r*. *t* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2* ∧ *t1* @ [✓(*r*)] ∈ 𝒯 (*Y i*) ∧ *tF t1* ∧ *t2* ∈ 𝒟 (*S r*)}› **for** *i*
**assume** ‹*map* (*ev* ∘ *of-ev*) *t* ∉ 𝒟 (⨆ *i*. *Y i*)›
**with** ‹*t* ∈ 𝒟 *?rhs*› **have** ‹*T1 i* ∪ *T2 i* ≠ {}› **for** *i*
**by** (*simp add*: *T1-def T2-def limproc-is-thelub chain-Seq*$_{ptick}$*-left* ‹*chain Y*›
*LUB-projs Seq*$_{ptick}$*-projs*) *fast*
**moreover have** ‹*finite* (*T1 0* ∪ *T2 0*)›
**unfolding** *T1-def T2-def*
**by** (*rule finite-subset*[*of* - ‹{*u*. *u* ≤ *map* (*ev* ∘ *of-ev*) *t*}›])
(*use tickFree-map-ev-of-ev-eq-iff* **in** ‹*force simp add*: *prefixes-fin*›)+
**moreover have** ‹*T1* (*Suc i*) ∪ *T2* (*Suc i*) ⊆ *T1 i* ∪ *T2 i*› **for** *i*
**unfolding** *T1-def T2-def* **by** (*intro allI subsetI*; *simp*)
(*metis* (*no-types*, *lifting*) ‹*chain Y*› *po-class.chainE le-approx-lemma-T le-approx1*
*subsetD*[*of* ‹𝒟 (*Y* (*Suc i*))› ‹𝒟 (*Y i*)›] *subsetD*[*of* ‹𝒯 (*Y* (*Suc i*))› ‹𝒯 (*Y i*)› ‹- @ [✓(-)]›])
**ultimately have** ‹(⋂ *i*. *T1 i* ∪ *T2 i*) ≠ {}› **by** (*rule Inter-nonempty-finite-chained-sets*)
**then obtain** *t1* **where** ∗ : ‹∀ *i*. *t1* ∈ *T1 i* ∪ *T2 i*› **by** *auto*
**then obtain** *t2* **where** ∗∗ : ‹*t* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2*› ‹*tF t1*› ‹*ftF t2*›
**by** (*auto simp add*: *T1-def T2-def dest*: *D-imp-front-tickFree*)
**show** ‹*t* ∈ 𝒟 *?lhs*›
**proof** (*cases* ‹∀ *i*. *t1* ∈ 𝒟 (*Y i*)›)
**from** ∗∗ **show** ‹∀ *i*. *t1* ∈ 𝒟 (*Y i*) ⟹ *t* ∈ 𝒟 *?lhs*›
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs limproc-is-thelub* ‹*chain Y*› *LUB-projs*)
**next**
**assume** ‹¬ (∀ *i*. *t1* ∈ 𝒟 (*Y i*))›
**then obtain** *j* **where** ∗∗∗ : ‹*j* ≤ *i* ⟹ *t1* ∉ 𝒟 (*Y i*)› **for** *i*
**by** (*meson* ‹*chain Y*› *in-mono le-approx-def po-class.chain-mono*)
**hence** ‹*j* ≤ *i* ⟹ *t1* ∉ *T1 i*› **for** *i* **by** (*simp add*: *T1-def*)
**with** ∗ **have** ‹*j* ≤ *i* ⟹ *t1* ∈ *T2 i*› **for** *i* **by** *blast*
**then obtain** *r* **where** ‹*t1* @ [✓(*r*)] ∈ 𝒯 (*Y j*)› ‹*t2* ∈ 𝒟 (*S r*)›
**unfolding** *T2-def* **by** (*auto simp add*: ∗∗(*1*))
**from** *this*(*1*) ‹*chain Y*› ∗∗∗ **have** ‹*j* ≤ *i* ⟹ *t1* @ [✓(*r*)] ∈ 𝒯 (*Y i*)› **for** *i*
**by** (*metis eq-imp-le is-processT9 le-approx2T po-class.chain-mono*)
**hence** ‹*t1* @ [✓(*r*)] ∈ 𝒯 (⨆ *i*. *Y i*)›
**by** (*meson* ∗∗∗ ‹*chain Y*› *dual-order.refl is-processT9 is-ub-thelub le-approx2T*)
**with** ‹*t2* ∈ 𝒟 (*S r*)› ∗∗(*1*, *2*) **show** ‹*t* ∈ 𝒟 *?lhs*›
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs*)
**qed**
**qed**
**thus** ‹*t* ∈ 𝒟 *?rhs* ⟹ *t* ∈ 𝒟 *?lhs*› **for** *t*
**by** (*meson D-imp-front-tickFree div-butlast-when-non-tickFree-iff front-tickFree-iff-tickFree-butlast*)
**next**
**show** ‹(*t*, *X*) ∈ ℱ *?lhs* ⟹ (*t*, *X*) ∈ ℱ *?rhs*› **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** ‹$(t, X) \in \mathcal{F}$ *?rhs*› ‹$t \notin \mathcal{D}$ *?rhs*›
**from** ‹$t \notin \mathcal{D}$ *?rhs*› **obtain** *j* **where** ‹$t \notin \mathcal{D}$ $(Y\ j\ ;_{\checkmark}\ S)$›
**by** (*auto simp add*: *limproc-is-thelub chain-*$Seq_{ptick}$*-left* ‹*chain Y*› *LUB-projs*)
**moreover from** ‹$(t, X) \in \mathcal{F}$ *?rhs*› **have** ‹$(t, X) \in \mathcal{F}$ $(Y\ j\ ;_{\checkmark}\ S)$›
**by** (*simp add*: *limproc-is-thelub chain-*$Seq_{ptick}$*-left* ‹*chain Y*› *F-LUB*)
**ultimately show** ‹$(t, X) \in \mathcal{F}$ *?lhs*›
**by** (*fact le-approx2*[*OF mono-*$Seq_{ptick}$[*OF is-ub-thelub*[*OF* ‹*chain Y*›] *below-refl*], *THEN iffD2*])
**qed**

**lemma** ‹*finite R* $\Longrightarrow$ *chain Y* $\Longrightarrow$ $\sqcap r \in R.\ (\bigsqcup i.\ Y\ i\ r) = (\bigsqcup i.\ \sqcap r \in R.\ Y\ i\ r)$›
**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* : ‹$Y \equiv \lambda i\ r :: nat.$ *if* $r \leq i$ *then STOP else* $\bot :: (nat, nat)$ $process_{ptick}$›
**shows** ‹*chain Y*› ‹$\sqcap r \in UNIV.\ (\bigsqcup i.\ Y\ i\ r) \neq (\bigsqcup i.\ \sqcap r \in UNIV.\ Y\ i\ r)$›
**proof** −
**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** ‹$(\bigsqcup i.\ Y\ i) = (\lambda r.\ STOP)$›
**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** \$ : ‹$\sqcap r \in UNIV.\ (\bigsqcup i.\ Y\ i\ r) = STOP$›
**by** (*simp add*: *GlobalNdet-is-STOP-iff* $*$ *lub-fun*)

**have** ‹$\sqcap r \in UNIV.\ Y\ i\ r = \bot$› **for** *i*
**by** (*simp add*: *BOT-iff-Nil-D D-GlobalNdet Y-def D-BOT*)
(*use Suc-n-not-le-n* **in** *blast*)
**hence** \$\$ : ‹$(\bigsqcup i.\ \sqcap r \in UNIV.\ Y\ i\ r) = \bot$› **by** *simp*

**from** \$ \$\$ **show** ‹$\sqcap r \in UNIV.\ (\bigsqcup i.\ Y\ i\ r) \neq (\bigsqcup i.\ \sqcap r \in UNIV.\ Y\ i\ r)$› **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* : ‹$P \equiv SKIPS\ UNIV :: (nat, nat)\ process_{ptick}$›
**and** *Y-def* : ‹$Y \equiv \lambda i\ r :: nat.$ *if* $r \leq i$ *then STOP else* $\bot :: (nat, nat)\ process_{ptick}$›
**shows** ‹*chain Y*› ‹$P\ ;_{\checkmark}\ (\bigsqcup i.\ Y\ i) \neq (\bigsqcup i.\ P\ ;_{\checkmark}\ Y\ i)$›

**proof** −
**show** ∗ : ‹*chain Y*› **unfolding** *Y-def* **by** (*auto intro*!: *chainI fun-belowI*)
**have** ‹*P* ;✓ (⨆ *i*. *Y i*) = ⊓*r* ∈ *UNIV*. (⨆ *i*. *Y i r*)›
**by** (*simp add*: *P-def* ∗ *lub-fun*)
**also have** ‹. . . ≠ (⨆ *i*. ⊓*r* ∈ *UNIV*. *Y i r*)›
**unfolding** *P-def Y-def* **by** (*fact infinite-GlobalNdet-not-cont*(*2*))
**also have** ‹(⨆ *i*. ⊓*r* ∈ *UNIV*. *Y i r*) = (⨆ *i*. *P* ;✓ *Y i*)›
**by** (*simp add*: *P-def*)
**finally show** ‹*P* ;✓ (⨆ *i*. *Y i*) ≠ (⨆ *i*. *P* ;✓ *Y i*)› .
**qed**

We must therefore find a condition under which $Seq_{ptick}$ is continuous.

**private lemma** *cont-right-prem-Seq*$_{ptick}$ :
‹*S* ;✓ (⨆ *i*. *Y i*) = (⨆ *i*. *S* ;✓ *Y i*)› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*chain Y*› **and** ‹$\mathbb{F}_{\checkmark}$(*S*)›
— We have to add this hypothesis in the generalization.
**proof** (*rule Process-eq-optimizedI*)
**show** ‹*t* ∈ $\mathcal{D}$ *?lhs* ⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **for** *t*
**by** (*simp add*: *Seq*$_{ptick}$*-projs limproc-is-thelub ch2ch-fun* ‹*chain Y*› *lub-fun chain-Seq*$_{ptick}$*-right D-LUB*) *blast*
**next**
**have** ‹*t* ∈ $\mathcal{D}$ *?lhs*› **if** ‹*t* ∈ $\mathcal{D}$ *?rhs*› **and** ‹*tF t*› **for** *t*
**proof** (*cases* ‹*map* (*ev* ∘ *of-ev*) *t* ∈ $\mathcal{D}$ *S*›)
**show** ‹*map* (*ev* ∘ *of-ev*) *t* ∈ $\mathcal{D}$ *S* ⟹ *t* ∈ $\mathcal{D}$ *?lhs*›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*)
(*metis append.right-neutral front-tickFree-Nil* ‹*tF t*› *tickFree-map-ev-comp tickFree-map-ev-of-ev-eq-iff*)
**next**
**define** *T* **where** ‹*T i* ≡ {*t1*. ∃ *t2 r*. *t* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2* ∧ *t1* @ [✓(*r*)] ∈ $\mathcal{T}$ *S* ∧ *tF t1* ∧ *t2* ∈ $\mathcal{D}$ (*Y i r*)}› **for** *i*
**assume** ‹*map* (*ev* ∘ *of-ev*) *t* ∉ $\mathcal{D}$ *S*›
**with** ‹*t* ∈ $\mathcal{D}$ *?rhs*› **have** ‹*T i* ≠ {}› **for** *i*
**by** (*fastforce simp add*: *T-def limproc-is-thelub chain-Seq*$_{ptick}$*-right* ‹*chain Y*› *D-LUB Seq*$_{ptick}$*-projs is-processT7 tickFree-map-ev-of-ev-same-type-is*)
**moreover have** ‹*finite* (*T 0*)›
**unfolding** *T-def*
**by** (*rule finite-subset*[*of* - ‹{*u*. *u* ≤ *map* (*ev* ∘ *of-ev*) *t*}›])
(*use tickFree-map-ev-of-ev-eq-iff* **in** ‹*force simp add*: *prefixes-fin*›)+
**moreover have** ‹*T* (*Suc i*) ⊆ *T i*› **for** *i*
**unfolding** *T-def* **by** (*intro allI Un-mono subsetI*; *simp*)
(*metis* ‹*chain Y*› *fun-below-iff subset-iff*[*of* ‹$\mathcal{D}$ (*Y* (*Suc i*) -)› ‹$\mathcal{D}$ (*Y i* -)›]
*po-class.chainE le-approx1*)
**ultimately have** ‹(⋂ *i*. *T i*) ≠ {}› **by** (*rule Inter-nonempty-finite-chained-sets*)
**then obtain** *t1* **where** ‹∀ *i*. *t1* ∈ *T i*› **by** *auto*
**then obtain** *t2* **where** ∗ : ‹*t* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2*›
‹*tF t1*› ‹∀ *i*. ∃ *r*. *t1* @ [✓(*r*)] ∈ $\mathcal{T}$ *S* ∧ *t2* ∈ $\mathcal{D}$ (*Y i r*)›
**by** (*simp add*: *T-def*) *blast*
**have** ‹*t1* ∈ $\mathcal{T}$ *S*› **by** (*meson* ∗(*3*) *prefixI is-processT3-TR*)
**from** ∗(*1*, *2*) ‹*map* (*ev* ∘ *of-ev*) *t* ∉ $\mathcal{D}$ *S*›
**have** ‹*t1* ∉ $\mathcal{D}$ *S*› **using** *is-processT7 tickFree-map-ev-of-ev-eq-iff* **by** *fastforce*

**define** $U$ **where** ‹$U\ i \equiv \{r.\ t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S \wedge t2 \in \mathcal{D}\ (Y\ i\ r)\}$› **for** $i$
**from** $*(3)$ **have** ‹$U\ i \neq \{\}$› **for** $i$ **by** (*simp add*: *U-def*)
**moreover have** ‹*finite* $(U\ 0)$›
**proof** (*rule finite-subset*[*of* - ‹$\{r.\ t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S\}$›])
**show** ‹$U\ 0 \subseteq \{r.\ t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S\}$› **unfolding** *U-def* **by** *blast*
**next**
**show** ‹*finite* $\{r.\ t1\ @\ [\checkmark(r)] \in \mathcal{T}\ S\}$›
**by** (*simp add*: ‹$\mathbb{F}_{\checkmark}(S)$› ‹$t1 \notin \mathcal{D}\ S$› *finite-ticksD*)
**qed**
**moreover have** ‹$U\ (Suc\ i) \subseteq U\ i$› **for** $i$
**by** (*simp add*: *U-def subset-iff*)
(*meson fun-below-iff in-mono le-approx1 chainE* ‹*chain* $Y$›)
**ultimately have** ‹$(\bigcap i.\ U\ i) \neq \{\}$› **by** (*rule Inter-nonempty-finite-chained-sets*)
**then obtain** $r$ **where** $**$ : ‹$\forall i.\ r \in U\ i$› **by** *auto*
**with** $*$ **show** ‹$t \in \mathcal{D}$ *?lhs*›
**by** (*simp add*: $Seq_{ptick}$*-projs U-def* ‹*chain* $Y$› *ch2ch-fun limproc-is-thelub D-LUB lub-fun*) *blast*
**qed**
**thus** ‹$t \in \mathcal{D}$ *?rhs* $\Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** $t$
**by** (*meson D-imp-front-tickFree div-butlast-when-non-tickFree-iff front-tickFree-iff-tickFree-butlast*)
**next**
**show** ‹$(t, X) \in \mathcal{F}$ *?lhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?rhs*› **for** $t\ X$
**by** (*simp add*: $Seq_{ptick}$*-projs limproc-is-thelub ch2ch-fun* ‹*chain* $Y$› *lub-fun chain-*$Seq_{ptick}$*-right F-LUB*) *blast*
**next**
**fix** $t\ X$ **assume** ‹$(t, X) \in \mathcal{F}$ *?rhs*› ‹$t \notin \mathcal{D}$ *?rhs*›
**from** ‹$t \notin \mathcal{D}$ *?rhs*› **obtain** $j$ **where** ‹$t \notin \mathcal{D}\ (S\ ;_{\checkmark}\ Y\ j)$›
**by** (*auto simp add*: *limproc-is-thelub chain-*$Seq_{ptick}$*-right* ‹*chain* $Y$› *D-LUB*)
**moreover from** ‹$(t, X) \in \mathcal{F}$ *?rhs*› **have** ‹$(t, X) \in \mathcal{F}\ (S\ ;_{\checkmark}\ Y\ j)$›
**by** (*simp add*: *limproc-is-thelub chain-*$Seq_{ptick}$*-right* ‹*chain* $Y$› *F-LUB*)
**ultimately show** ‹$(t, X) \in \mathcal{F}$ *?lhs*›
**by** (*fact le-approx2*[*OF mono-*$Seq_{ptick}$[*OF below-refl is-ub-thelub*[*OF* ‹*chain* $Y$›]], *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** $Seq_{ptick}$*-cont* [*simp*] : ‹*cont* $(\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
**if** ‹*cont* $f$› **and** ‹*cont* $g$› **and** ‹$\mathbb{F}_{\checkmark\Rightarrow}(f)$›
**for** $g$ :: ‹- ⇒ - ⇒ $('a, 's)$ $process_{ptick}$›
**proof** (*rule cont-apply*[**where** $f$ = ‹$\lambda x\ y.\ f\ x\ ;_{\checkmark}\ y$›])
**show** ‹*cont* $g$› **by** (*fact* ‹*cont* $g$›)
**next**
**show** ‹*cont* $(\lambda x.\ f\ x\ ;_{\checkmark}\ y)$› **for** $y$ :: ‹- ⇒ $('a, 's)$ $process_{ptick}$›
**proof** (*rule contI2*)
**show** ‹*monofun* $(\lambda x.\ f\ x\ ;_{\checkmark}\ y)$› **by** (*simp add*: *cont2monofunE mono-*$Seq_{ptick}$ *monofunI* ‹*cont* $f$›)

**next**
**show** ‹*chain* $Y \Longrightarrow f\ (\bigsqcup i.\ Y\ i)\ ;_{\checkmark}\ y \sqsubseteq (\bigsqcup i.\ f\ (Y\ i)\ ;_{\checkmark}\ y)$› **for** $Y$
**by** (*simp add*: *ch2ch-cont cont2contlubE cont-left-prem-Seq*$_{ptick}$ ‹*cont f*›)
**qed**
**next**
**show** ‹*cont* $(\lambda y :: - \Rightarrow ('a, 's)\ process_{ptick}.\ f\ x\ ;_{\checkmark}\ y)$› **for** $x$
**proof** (*rule contI2*)
**show** ‹*monofun* $((;_{\checkmark})\ (f\ x))$› **by** (*simp add*: *mono-Seq*$_{ptick}$ *monofunI*)
**next**
**show** ‹*chain* $Y \Longrightarrow f\ x\ ;_{\checkmark}\ (\bigsqcup i.\ Y\ i) \sqsubseteq (\bigsqcup i.\ f\ x\ ;_{\checkmark}\ Y\ i)$›
**for** $Y :: ‹- \Rightarrow - \Rightarrow ('a, 's)\ process_{ptick}›$
**oops**
— Unfortunately here, we cannot use *cont-right-prem-Seq*$_{ptick}$ 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** *Seq*$_{ptick}$*-cont* [*simp*] : ‹*cont* $(\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
**if** ‹*cont f*› **and** ‹*cont g*› **and** ‹$\bigwedge x.\ \mathbb{F}_{\checkmark}(f\ x)$›
**for** $g :: ‹- \Rightarrow - \Rightarrow ('a, 's)\ process_{ptick}›$
**proof** (*rule cont-apply*[**where** $f = ‹\lambda x\ y.\ f\ x\ ;_{\checkmark}\ y›$])
**show** ‹*cont g*› **by** (*fact* ‹*cont g*›)
**next**
**show** ‹*cont* $(\lambda x.\ f\ x\ ;_{\checkmark}\ y)$› **for** $y :: ‹- \Rightarrow ('a, 's)\ process_{ptick}›$
**proof** (*rule contI2*)
**show** ‹*monofun* $(\lambda x.\ f\ x\ ;_{\checkmark}\ y)$› **by** (*simp add*: *cont2monofunE mono-Seq*$_{ptick}$ *monofunI* ‹*cont f*›)
**next**
**show** ‹*chain* $Y \Longrightarrow f\ (\bigsqcup i.\ Y\ i)\ ;_{\checkmark}\ y \sqsubseteq (\bigsqcup i.\ f\ (Y\ i)\ ;_{\checkmark}\ y)$› **for** $Y$
**by** (*simp add*: *ch2ch-cont cont2contlubE cont-left-prem-Seq*$_{ptick}$ ‹*cont f*›)
**qed**
**next**
**show** ‹*cont* $(\lambda y :: - \Rightarrow ('a, 's)\ process_{ptick}.\ f\ x\ ;_{\checkmark}\ y)$› **for** $x$
**proof** (*rule contI2*)
**show** ‹*monofun* $((;_{\checkmark})\ (f\ x))$› **by** (*simp add*: *mono-Seq*$_{ptick}$ *monofunI*)
**next**
**show** ‹*chain* $Y \Longrightarrow f\ x\ ;_{\checkmark}\ (\bigsqcup i.\ Y\ i) \sqsubseteq (\bigsqcup i.\ f\ x\ ;_{\checkmark}\ Y\ i)$›
**for** $Y :: ‹- \Rightarrow - \Rightarrow ('a, 's)\ process_{ptick}›$
**by** (*simp add*: *cont-right-prem-Seq*$_{ptick}$ ‹$\bigwedge x.\ \mathbb{F}_{\checkmark}(f\ x)$›)
**qed**
**qed**

**end**

**corollary** ‹*cont f* $\Longrightarrow$ *cont g* $\Longrightarrow$ *cont* $(\lambda x.\ f\ x\ ;_{\checkmark}\ g\ x)$›
**for** $f :: ‹'b :: cpo \Rightarrow ('a, 'r :: finite)\ process_{ptick}›$
**by** (*simp add*: *finite-ticks-simps*(*5*))

**lemma** *MultiSeq$_{ptick}$-cont*[*simp*]:
‹⟦⋀*l*. *l* ∈ *set L* ⟹ *cont* (*f l*); ⋀*l r x*. *l* ∈ *set* (*butlast L*) ⟹ 𝔽✓(*f l x r*)⟧
⟹ *cont* (λ*x*. (*SEQ*✓ *l* ∈@ *L*. *f l x*) *r*)›
**proof** (*induct L arbitrary*: *r*)
**show** ‹⋀*r*. *cont* (λ*x*. (*SEQ*✓ *l* ∈@ []. *f l x*) *r*)› **by** *simp*
**next**
**case** (*Cons l0 L*)
**show** ‹*cont* (λ*x*. (*SEQ*✓ *l* ∈@ (*l0* # *L*). *f l x*) *r*)›
**proof** (*cases* ‹*L* = []›)
**show** ‹*L* = [] ⟹ *cont* (λ*x*. (*SEQ*✓ *l* ∈@ (*l0* # *L*). *f l x*) *r*)›
**by** (*simp add*: *Cons.prems*(*1*) *cont2cont-fun*)
**next**
**show** ‹*cont* (λ*x*. (*SEQ*✓ *l* ∈@ (*l0* # *L*). *f l x*) *r*)› **if** ‹*L* ≠ []›
**proof** (*subst MultiSeq$_{ptick}$-Cons*, *intro cont2cont-lambda Seq$_{ptick}$-cont*)
**show** ‹*cont* (λ*x*. *f l0 x r*)› **by** (*simp add*: *Cons.prems*(*1*) *cont2cont-fun*)
**next**
**have** ‹*cont* (λ*x*. (*SEQ*✓ *l* ∈@ *L*. *f l x*))›
**by** (*rule cont2cont-lambda*, *rule Cons.hyps*)
(*simp-all add*: *Cons.prems*(*1*, *2*) ‹*L* ≠ []›)
**thus** ‹*cont* (λ*x*. (*SEQ*✓ *l* ∈@ *L*. *f l x*) *y*)› **for** *y*
**by** (*fact cont2cont-fun*)
**next**
**show** ‹𝔽✓(*f l0 x r*)› **for** *x* **by** (*simp add*: *Cons.prems*(*2*) *that*)
**qed**
**qed**
**qed**

## 13.2 Synchronization Product

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

### 13.2.1 Monotonicity

**lemma** *mono-Sync$_{ptick}$* : ‹*P* ⟦*A*⟧✓ *Q* ⊑ *P'* ⟦*A*⟧✓ *Q'*› **if** ‹*P* ⊑ *P'*› **and** ‹*Q* ⊑ *Q'*›
**proof** (*unfold le-approx-def Refusals-after-def*, *safe*)
**from** *le-approx1*[*OF* ‹*P* ⊑ *P'*›] *le-approx-lemma-T*[*OF* ‹*P* ⊑ *P'*›]
*le-approx1*[*OF* ‹*Q* ⊑ *Q'*›] *le-approx-lemma-T*[*OF* ‹*Q* ⊑ *Q'*›]
**show** ‹*t* ∈ 𝒟 (*P'* ⟦*A*⟧✓ *Q'*) ⟹ *t* ∈ 𝒟 (*P* ⟦*A*⟧✓ *Q*)› **for** *t*
**by** (*simp add*: *D-Sync$_{ptick}$*) *fast*
**next**
**from** *le-approx2*[*OF* ‹*P* ⊑ *P'*›] *le-approx2*[*OF* ‹*Q* ⊑ *Q'*›]
**show** ‹*t* ∉ 𝒟 (*P* ⟦*A*⟧✓ *Q*) ⟹ (*t*, *X*) ∈ ℱ (*P* ⟦*A*⟧✓ *Q*) ⟹

$(t, X) \in \mathcal{F}\ (P' \llbracket A \rrbracket_{\checkmark}\ Q')\rangle$ **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** $\langle t \notin \mathcal{D}\ (P \llbracket A \rrbracket_{\checkmark}\ Q) \Longrightarrow (t, X) \in \mathcal{F}\ (P' \llbracket A \rrbracket_{\checkmark}\ Q') \Longrightarrow$
$(t, X) \in \mathcal{F}\ (P \llbracket A \rrbracket_{\checkmark}\ Q)\rangle$ **for** $t\ X$
**by** (*simp add*: $Sync_{ptick}$*-projs subset-iff*, *elim disjE*) *metis+*
**next**
**fix** $t$ **assume** $\langle t \in$ *min-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-P t-Q*
**where** $* : \langle t = u\ @\ v\rangle\ \langle tF\ u\rangle\ \langle ftF\ v\rangle$
$\langle u$ *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** $D\text{-}Sync_{ptick}$ **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*: $D\text{-}Sync_{ptick}$) (*use front-tickFree-Nil* **in** *blast*)
**ultimately show** *False*
**using** $\langle t \in$ *min-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-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-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** $\$ : \langle u' < u\rangle\ \langle t\text{-}Q' \leq t\text{-}Q\rangle$
$\langle u'$ *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*: $D\text{-}Sync_{ptick}{}'$)
(*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-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** ‹*t-Q* ∈ *min-elems* (*D Q*)› **if** ‹*t-Q* ∈ *D Q*›
**proof** (*rule ccontr*)
**assume** ‹*t-Q* ∉ *min-elems* (*D Q*)›
**with** ‹*t-Q* ∈ *D Q*› **obtain** *t-Q'* **where** ‹*t-Q'* < *t-Q*› ‹*t-Q'* ∈ *D Q*›
**by** (*metis antisym-conv2 elem-min-elems min-elems5*)
**from** *setinterleaves*$_{ptick}$*-less-prefixR*[*OF* ∗(*4*) ‹*t-Q'* < *t-Q*›]
**obtain** *u' t-P'*
**where** $ : ‹*u'* < *u*› ‹*t-P'* ≤ *t-P*›
‹*u' setinterleaves*$_{✓}$*tick-join* ((*t-P'*, *t-Q'*), *A*)› **by** *blast*
**from** ∗(*5*) *D-T* **have** ‹*t-P* ∈ *T P*› **by** *blast*
**with** $(*2*,*3*) ‹*t-Q'* ∈ *D Q*› **have** ‹*u'* ∈ *D* (*P* ⟦*A*⟧$_{✓}$ *Q*)›
**by** (*simp add*: *D-Sync*$_{ptick}$*'*)
(*metis append.right-neutral front-tickFree-Nil is-processT3-TR*)
**moreover from** ‹*u'* < *u*› **have** ‹*u'* < *t*›
**by** (*simp add*: ∗(*1*)) (*meson Prefix-Order.prefixI dual-order.strict-trans1*)
**ultimately show** *False*
**using** ‹*t* ∈ *min-elems* (*D* (*P* ⟦*A*⟧$_{✓}$ *Q*))› *min-elems-no nless-le* **by** *blast*
**qed**
**with** ∗(*5*) **have** ‹*t-Q* ∈ *T Q'*›
**by** (*meson in-mono le-approx2T le-approx3* ‹*Q* ⊑ *Q'*›)

**from** ‹*t-P* ∈ *T P'*› ‹*t-Q* ∈ *T Q'*› ∗(*4*) **show** ‹*t* ∈ *T* (*P'* ⟦*A*⟧$_{✓}$ *Q'*)›
**by** (*auto simp add*: ∗(*1*) ‹*v* = []› *T-Sync*$_{ptick}$)
**qed**

### 13.2.2 Preliminaries

**lemma** *chain-Sync*$_{ptick}$*-left* : ‹*chain Y* ⟹ *chain* (λ*i*. *Y i* ⟦*A*⟧$_{✓}$ *Q*)›
**and** *chain-Sync*$_{ptick}$*-right* : ‹*chain Z* ⟹ *chain* (λ*i*. *P* ⟦*A*⟧$_{✓}$ *Z i*)›
**by** (*simp-all add*: *chain-def mono-Sync*$_{ptick}$)

**lemma** *cont-left-prem-Sync*$_{ptick}$ :
‹(⨆ *i*. *Y i*) ⟦*A*⟧$_{✓}$ *Q* = (⨆ *i*. *Y i* ⟦*A*⟧$_{✓}$ *Q*)› **if** *chain*: ‹*chain Y*›
**proof** (*rule Process-eq-optimizedI*)
**show** ‹*t* ∈ *D* ((⨆ *i*. *Y i*) ⟦*A*⟧$_{✓}$ *Q*) ⟹ *t* ∈ *D* (⨆ *i*. *Y i* ⟦*A*⟧$_{✓}$ *Q*)› **for** *t*
**by** (*simp add*: *limproc-is-thelub chain chain-Sync*$_{ptick}$*-left D-Sync*$_{ptick}$ *D-LUB T-LUB*) *blast*
**next**
**show** ‹(*t*, *X*) ∈ *F* ((⨆ *i*. *Y i*) ⟦*A*⟧$_{✓}$ *Q*) ⟹ (*t*, *X*) ∈ *F* (⨆ *i*. *Y i* ⟦*A*⟧$_{✓}$ *Q*)› **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** ‹*t* ∈ *D* (⨆ *i*. *Y i* ⟦*A*⟧$_{✓}$ *Q*)›
**define** *S*
**where** ‹*S i* ≡ {(*t-Y*, *t-Q*, *u*). ∃ *v*. *tF u* ∧ *ftF v* ∧ *t* = *u* @ *v* ∧

$u$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((t\text{-}Y,\ t\text{-}Q),\ A) \wedge$
$(t\text{-}Y \in \mathcal{D}\ (Y\ i) \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}Y \in \mathcal{T}\ (Y\ i) \wedge$
$t\text{-}Q \in \mathcal{D}\ Q)\}$› **for** $i$
**from** ‹$t \in \mathcal{D}\ (\bigsqcup i.\ Y\ i\ [\![A]\!]_{\checkmark}\ Q)$› **have** ‹$S\ i \neq \{\}$› **for** $i$
**by** (*simp add*: *S-def limproc-is-thelub chain chain-Sync*$_{ptick}$*-left D-Sync*$_{ptick}$ *D-LUB*) *fast*
**moreover have** ‹*finite* $(S\ 0)$›
**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** ‹$S\ (Suc\ i) \subseteq S\ i$› **for** $i$ **by** (*simp add*: *S-def*) *blast*
**ultimately have** ‹$(\bigcap i.\ S\ i) \neq \{\}$› **by** (*rule Inter-nonempty-finite-chained-sets*)
**then obtain** $t\text{-}Y\ t\text{-}Q\ u$ **where** ‹$(t\text{-}Y,\ t\text{-}Q,\ u) \in (\bigcap i.\ S\ i)$› **by** *auto*
**hence** ‹$tF\ u \wedge ftF\ (drop\ (length\ u)\ t) \wedge$
$t = u\ @\ drop\ (length\ u)\ t \wedge u$ *setinterleaves*$_{\checkmark tick\text{-}join}$ $((t\text{-}Y,\ t\text{-}Q),\ A) \wedge$
$((\forall i.\ t\text{-}Y \in \mathcal{D}\ (Y\ i)) \wedge t\text{-}Q \in \mathcal{T}\ Q \vee (\forall i.\ t\text{-}Y \in \mathcal{T}\ (Y\ i)) \wedge t\text{-}Q \in \mathcal{D}\ Q)$›
**by** (*auto simp add*: *S-def*) (*meson chain-lemma le-approx1 le-approx-lemma-T subsetD chain*)
**show** ‹$t \in \mathcal{D}\ ((\bigsqcup i.\ Y\ i)\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*simp add*: *limproc-is-thelub chain D-Sync*$_{ptick}$ *T-LUB D-LUB*)
(*use* ‹*?this*› **in** *blast*)
**next**
**fix** $t\ X$ **assume** ‹$(t,\ X) \in \mathcal{F}\ (\bigsqcup i.\ Y\ i\ [\![A]\!]_{\checkmark}\ Q)$› ‹$t \notin \mathcal{D}(\bigsqcup i.\ Y\ i\ [\![A]\!]_{\checkmark}\ Q)$›
**have** ‹$Y\ i \sqsubseteq (\bigsqcup i.\ Y\ i)$› **for** $i$ **by** (*simp add*: *is-ub-thelub* ‹*chain Y*›)
**moreover from** ‹$t \notin \mathcal{D}(\bigsqcup i.\ Y\ i\ [\![A]\!]_{\checkmark}\ Q)$› **obtain** $j$ **where** ‹$t \notin \mathcal{D}\ (Y\ j\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*auto simp add*: *limproc-is-thelub chain-Sync*$_{ptick}$*-left* ‹*chain Y*› *D-LUB*)
**moreover from** ‹$(t,\ X) \in \mathcal{F}\ (\bigsqcup i.\ Y\ i\ [\![A]\!]_{\checkmark}\ Q)$› **have** ‹$(t,\ X) \in \mathcal{F}\ (Y\ j\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*simp add*: *limproc-is-thelub chain-Sync*$_{ptick}$*-left* ‹*chain Y*› *F-LUB*)
**ultimately show** ‹$(t,\ X) \in \mathcal{F}\ ((\bigsqcup i.\ Y\ i)\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*metis* (*mono-tags*, *lifting*) *below-refl le-approx2 mono-Sync*$_{ptick}$)
**qed**

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *cont-right-prem-Sync*$_{ptick}$ :
‹$P\ [\![A]\!]_{\checkmark}\ (\bigsqcup i.\ Z\ i) = (\bigsqcup i.\ P\ [\![A]\!]_{\checkmark}\ Z\ i)$› **if** ‹*chain Z*›
**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*]: ‹*cont* $(\lambda x.\ f\ x\ [\![A]\!]_{\checkmark}\ g\ x)$› **if** ‹*cont f*› ‹*cont g*›
**proof** (*rule cont-apply*[**where** $f$ = ‹$\lambda x\ y.\ y\ [\![A]\!]_{\checkmark}\ g\ x$›])
**from** ‹*cont f*› **show** ‹*cont f*› **.**
**next**
**show** ‹*cont* $(\lambda y.\ y\ [\![A]\!]_{\checkmark}\ g\ x)$› **for** $x$
**proof** (*rule contI2*)

```
    show ‹monofun (λy. y ⟦A⟧✓ g x)› by (simp add: monofunI mono-Sync_ptick)
  next
    show ‹chain Y ⟹ (⨆ i. Y i) ⟦A⟧✓ g x ⊑ (⨆ i. Y i ⟦A⟧✓ g x)› for Y
      by (simp add: cont-left-prem-Sync_ptick)
  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 (Sync_ptick y A)› by (simp add: monofunI mono-Sync_ptick)
    next
      show ‹chain Z ⟹ y ⟦A⟧✓ (⨆ i. Z i) ⊑ (⨆ i. y ⟦A⟧✓ Z i)› for Z
        by (simp add: cont-right-prem-Sync_ptick)
    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* : $\langle P \sqsubseteq_{FD} P' \Longrightarrow (\bigwedge r.\ Q\ r \sqsubseteq_{FD} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{FD} P'\ ;_{\checkmark}\ Q' \rangle$
**proof** (*rule trans-FD*[*of* - $\langle P'\ ;_{\checkmark}\ Q \rangle$])
  **show** $\langle P \sqsubseteq_{FD} P' \Longrightarrow (\bigwedge r.\ Q\ r \sqsubseteq_{FD} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{FD} P'\ ;_{\checkmark}\ Q \rangle$
    **unfolding** *refine-defs Seq$_{ptick}$-projs*
    **by** (*auto simp add*: *subset-iff T-F-spec*[*symmetric*])
**next**
  **show** $\langle P \sqsubseteq_{FD} P' \Longrightarrow (\bigwedge r.\ Q\ r \sqsubseteq_{FD} Q'\ r) \Longrightarrow P'\ ;_{\checkmark}\ Q \sqsubseteq_{FD} P'\ ;_{\checkmark}\ Q' \rangle$
    **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* : $\langle P \sqsubseteq_{DT} P' \Longrightarrow (\bigwedge r.\ Q\ r \sqsubseteq_{DT} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{DT} P'\ ;_{\checkmark}\ Q' \rangle$
**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** $\langle s \in \mathcal{D}\ (P'\ ;_{\checkmark}\ Q) \Longrightarrow s \in \mathcal{D}\ (P\ ;_{\checkmark}\ Q) \rangle$ **for** $s$
      **by** (*auto simp add*: *refine-defs Seq$_{ptick}$-projs*)
  **next**
    **from** $\langle P \sqsubseteq_{DT} P' \rangle$ **show** $\langle s \in \mathcal{T}\ (P'\ ;_{\checkmark}\ Q) \Longrightarrow s \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q) \rangle$ **for** $s$
      **by** (*auto simp add*: *Seq$_{ptick}$-projs refine-defs*)
  **qed**
**next**
  **show** $\langle (\bigwedge r.\ Q\ r \sqsubseteq_{DT} Q'\ r) \Longrightarrow P'\ ;_{\checkmark}\ Q \sqsubseteq_{DT} P'\ ;_{\checkmark}\ Q' \rangle$
    **by** (*simp add*: *refine-defs Seq$_{ptick}$-projs*) *blast*
**qed**

**lemma** *mono-Seq$_{ptick}$-F-right* : $\langle (\bigwedge r.\ Q\ r \sqsubseteq_{F} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{F} P\ ;_{\checkmark}\ Q' \rangle$
  **by** (*auto simp add*: *failure-refine-def Seq$_{ptick}$-projs*) *blast*

**lemma** *mono-Seq$_{ptick}$-D-right* : $\langle (\bigwedge r.\ Q\ r \sqsubseteq_{D} Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_{D} P\ ;_{\checkmark}\ Q' \rangle$

**by** (*simp add*: *divergence-refine-def* $Seq_{ptick}$*-projs*) *blast*

**lemma** *mono-*$Seq_{ptick}$*-T-right* : $\langle(\bigwedge r.\ Q\ r \sqsubseteq_T Q'\ r) \Longrightarrow P\ ;_{\checkmark}\ Q \sqsubseteq_T P\ ;_{\checkmark}\ Q'\rangle$
**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** $monos\text{-}Seq_{ptick}$ = $mono\text{-}Seq_{ptick}$ $mono\text{-}Seq_{ptick}$*-FD* $mono\text{-}Seq_{ptick}$*-DT* $mono\text{-}Seq_{ptick}$*-F-right* $mono\text{-}Seq_{ptick}$*-D-right* $mono\text{-}Seq_{ptick}$*-T-right*

### 14.0.2 Multiple Sequential Composition

**lemma** $mono\text{-}MultiSeq_{ptick}$ :
$\langle(\bigwedge x\ r.\ x \in set\ L \Longrightarrow P\ x\ r \sqsubseteq Q\ x\ r) \Longrightarrow$
$(SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r \sqsubseteq (SEQ_{\checkmark}\ l \in@\ L.\ Q\ l)\ r\rangle$
**by** (*induct L arbitrary*: *r*, *simp-all add*: *fun-belowI* $mono\text{-}Seq_{ptick}$)

**lemma** $mono\text{-}MultiSeq_{ptick}$*-FD* :
$\langle(\bigwedge x\ r.\ x \in set\ L \Longrightarrow P\ x\ r \sqsubseteq_{FD} Q\ x\ r) \Longrightarrow$
$(SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r \sqsubseteq_{FD} (SEQ_{\checkmark}\ l \in@\ L.\ Q\ l)\ r\rangle$
**and** $mono\text{-}MultiSeq_{ptick}$*-DT* :
$\langle(\bigwedge x\ r.\ x \in set\ L \Longrightarrow P\ x\ r \sqsubseteq_{DT} Q\ x\ r) \Longrightarrow$
$(SEQ_{\checkmark}\ l \in@\ L.\ P\ l)\ r \sqsubseteq_{DT} (SEQ_{\checkmark}\ l \in@\ L.\ Q\ l)\ r\rangle$
**by** (*induct L arbitrary*: *r*, *simp-all add*: $monos\text{-}Seq_{ptick}$)

**lemmas** $monos\text{-}MultiSeq_{ptick}$ =
$mono\text{-}MultiSeq_{ptick}$ $mono\text{-}MultiSeq_{ptick}$*-FD* $mono\text{-}MultiSeq_{ptick}$*-FD*

### 14.0.3 Synchronization Product

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

**lemma** $mono\text{-}Sync_{ptick}$*-DT* :
$\langle P \sqsubseteq_{DT} P' \Longrightarrow Q \sqsubseteq_{DT} Q' \Longrightarrow P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_{DT} P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$
**by** (*simp add*: *refine-defs* $T\text{-}Sync_{ptick}$ $D\text{-}Sync_{ptick}$) *blast*

**lemma** $mono\text{-}Sync_{ptick}$*-FD* : $\langle P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_{FD} P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$
**if** $\langle P \sqsubseteq_{FD} P'\rangle$ **and** $\langle Q \sqsubseteq_{FD} Q'\rangle$
**proof** −
**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** $mono\text{-}Sync_{ptick}$*-DT* **have** $\langle P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_{DT} P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$ **by** *blast*
**hence** $*$ : $\langle P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_D P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$ **by** (*simp add*: *leDT-imp-leD*)
**show** $\langle P\ [\![A]\!]_{\checkmark}\ Q \sqsubseteq_{FD} P'\ [\![A]\!]_{\checkmark}\ Q'\rangle$
**proof** (*rule leF-leD-imp-leFD*[*OF* - $*$],
*unfold failure-refine-def*, *safe*)
**fix** *t X* **assume** $\langle(t, X) \in \mathcal{F}\ (P'\ [\![A]\!]_{\checkmark}\ Q')\rangle$
**then consider** $\langle t \in \mathcal{D}\ (P'\ [\![A]\!]_{\checkmark}\ Q')\rangle$
| (*fail*) *t-P t-Q X-P X-Q*
**where** $\langle(t\text{-}P, X\text{-}P) \in \mathcal{F}\ P'\rangle$ $\langle(t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ Q'\rangle$

‹$t$ *setinterleaves*$_{\checkmark\, tick\text{-}join}$ (($t$-$P$, $t$-$Q$), $A$)›
‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ *tick-join X-P A X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ [\![A]\!]_{\checkmark}\ Q)$›
**proof** *cases*
**show** ‹$t \in \mathcal{D}\ (P'\ [\![A]\!]_{\checkmark}\ Q') \Longrightarrow (t,\ X) \in \mathcal{F}\ (P\ [\![A]\!]_{\checkmark}\ Q)$›
**using** $*$ *D-F* **unfolding** *divergence-refine-def* **by** *blast*
**next**
**case** *fail*
**from** *fail*(*1*, *2*) ‹$P \sqsubseteq_{FD}\ P'$› ‹$Q \sqsubseteq_{FD}\ Q'$›
**have** ‹($t$-$P$, $X$-$P$) $\in \mathcal{F}\ P$› ‹($t$-$Q$, $X$-$Q$) $\in \mathcal{F}\ Q$›
**unfolding** *refine-defs* **by** *auto*
**with** *fail*(*3*, *4*) **show** ‹$(t,\ X) \in \mathcal{F}\ (P\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*auto simp add*: *F-Sync*$_{ptick}$)
**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*$_{ptick}$ :
‹($\bigwedge l.\ l \in set\ L \Longrightarrow P\ l \sqsubseteq Q\ l$) $\Longrightarrow$ $[\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l \sqsubseteq [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l$›
**by** (*induct L rule*: *induct-list012*)
(*simp-all add*: *Sync*$_{Rlist}$*.mono-Sync*$_{ptick}$ *mono-Renaming*)

**lemma** *mono-MultiSync*$_{ptick}$*-FD* :
‹($\bigwedge l.\ l \in set\ L \Longrightarrow P\ l \sqsubseteq_{FD}\ Q\ l$) $\Longrightarrow$ $[\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l \sqsubseteq_{FD} [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l$›
**by** (*induct L rule*: *induct-list012*)
(*simp-all add*: *Sync*$_{Rlist}$*.mono-Sync*$_{ptick}$*-FD mono-Renaming-FD*)

**lemma** *mono-MultiSync*$_{ptick}$*-DT* :
‹($\bigwedge l.\ l \in set\ L \Longrightarrow P\ l \sqsubseteq_{DT}\ Q\ l$) $\Longrightarrow$ $[\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l \sqsubseteq_{DT} [\![S]\!]_{\checkmark}\ l \in@\ L.\ Q\ l$›
**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** ‹*tF u*› ‹*ftF v*› ‹*t-P* ∈ $\mathcal{D}$ (*P* ↓ *n*)› ‹*t-Q* ∈ $\mathcal{T}$ (*Q* ↓ *n*)›
‹*u setinterleaves*$_{✓}$ *tick-join* ((*t-P*, *t-Q*), *A*)›
**shows** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**proof** (*insert assms*(3, 4), *elim D-restriction-process*$_{ptick}$*E T-restriction-process*$_{ptick}$*E*)
**from** *assms*(1, 2, 5) **show** ‹*t-P* ∈ $\mathcal{D}$ *P* ⟹ *t-Q* ∈ $\mathcal{T}$ *Q* ⟹ *u* @ *v* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**by** (*auto simp add*: *D-restriction-process*$_{ptick}$ *D-Sync*$_{ptick}$)
**next**
**fix** *t-Q′ t-Q″*
**assume** ∗ : ‹*t-P* ∈ $\mathcal{D}$ *P*› ‹*length t-P* ≤ *n*› ‹*t-Q* = *t-Q′* @ *t-Q″*›
‹*t-Q′* ∈ $\mathcal{T}$ *Q*› ‹*length t-Q′* = *n*› ‹*tF t-Q′*› ‹*ftF t-Q″*›
**from** ‹*t-Q* = *t-Q′* @ *t-Q″*› **have** ‹*t-Q′* ≤ *t-Q*› **by** *simp*
**from** *setinterleaves*$_{ptick}$-*le-prefixR*[*OF assms*(5) *this*]
**obtain** *t-P′ t-P″ u′ u″*
**where** ∗∗ : ‹*u* = *u′* @ *u″*› ‹*t-P* = *t-P′* @ *t-P″*›
‹*u′ setinterleaves*$_{✓}$ *tick-join* ((*t-P′*, *t-Q′*), *A*)›
**by** (*meson Prefix-Order.prefixE*)
**from** *assms*(1) ‹*u* = *u′* @ *u″*› **have** ‹*tF u′*› **by** *auto*
**moreover from** ∗(1,4) ∗∗(2,3) **have** ‹*u′* ∈ $\mathcal{T}$ (*P* $[\![A]\!]_{✓}$ *Q*)›
**by** (*simp add*: *T-Sync*$_{ptick}$) (*metis D-T is-processT3-TR-append*)
**moreover have** ‹*length t-Q′* ≤ *length u′*›
**using** ∗∗(3) *setinterleaves*$_{ptick}$-*imp-lengthLR-le* **by** *blast*
**ultimately have** ‹*u′* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**by** (*metis* ∗(5) *D-restriction-process*$_{ptick}$*I nless-le*)
**with** ∗∗(1) *assms*(1, 2) **show** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›

**by** (*metis is-processT7 tickFree-append-iff tickFree-imp-front-tickFree*)
**next**
**fix** *t-P′ t-P″*
**assume** ∗ : ‹*t-P* = *t-P′* @ *t-P″*› ‹*t-P′* ∈ $\mathcal{T}$ *P*› ‹*length t-P′* = *n*›
‹*tF t-P′*› ‹*ftF t-P″*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› ‹*length t-Q* ≤ *n*›
**from** ‹*t-P* = *t-P′* @ *t-P″*› **have** ‹*t-P′* ≤ *t-P*› **by** *simp*
**from** *setinterleaves*$_{ptick}$*-le-prefixL*[*OF assms*(*5*) *this*]
**obtain** *t-Q′ t-Q″ u′ u″*
**where** ∗∗ : ‹*u* = *u′* @ *u″*› ‹*t-Q* = *t-Q′* @ *t-Q″*›
‹*u′ setinterleaves*$_{✓}$ *tick-join* ((*t-P′*, *t-Q′*), *A*)›
**by** (*meson Prefix-Order.prefixE*)
**from** *assms*(*1*) ‹*u* = *u′* @ *u″*› **have** ‹*tF u′*› **by** *auto*
**moreover from** ∗(*2*,*6*) ∗∗(*2*,*3*) **have** ‹*u′* ∈ $\mathcal{T}$ (*P* $[\![A]\!]_{✓}$ *Q*)›
**by** (*simp add*: *T-Sync*$_{ptick}$) (*metis is-processT3-TR-append*)
**moreover have** ‹*length t-P′* ≤ *length u′*›
**using** ∗∗(*3*) *setinterleaves*$_{ptick}$*-imp-lengthLR-le* **by** *blast*
**ultimately have** ‹*u′* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**by** (*metis* ∗(*3*) *D-restriction-process*$_{ptick}$*I nless-le*)
**with** ∗∗(*1*) *assms*(*1*, *2*) **show** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**by** (*metis is-processT7 tickFree-append-iff tickFree-imp-front-tickFree*)
**next**
**fix** *t-P′ t-P″ t-Q′ t-Q″*
**assume** $ : ‹*t-P* = *t-P′* @ *t-P″*› ‹*t-P′* ∈ $\mathcal{T}$ *P*› ‹*length t-P′* = *n*›
‹*tF t-P′*› ‹*ftF t-P″*› ‹*t-Q* = *t-Q′* @ *t-Q″*› ‹*t-Q′* ∈ $\mathcal{T}$ *Q*›
‹*length t-Q′* = *n*› ‹*tF t-Q′*› ‹*ftF t-Q″*›
**from** $(*1*, *6*) **have** ‹*t-P′* ≤ *t-P*› ‹*t-Q′* ≤ *t-Q*› **by** *simp-all*
**from** *setinterleaves*$_{ptick}$*-le-prefixLR*[*OF assms*(*5*) *this*]
**show** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**proof** (*elim disjE conjE exE*)
**fix** *u′ t-Q‴* **assume** $$ : ‹*u′* ≤ *u*› ‹*t-Q‴* ≤ *t-Q′*›
‹*u′ setinterleaves*$_{✓}$ *tick-join* ((*t-P′*, *t-Q‴*), *A*)›
**from** $(*7*) $$(*2*) *is-processT3-TR* **have** ‹*t-Q‴* ∈ $\mathcal{T}$ *Q*› **by** *blast*
**with** $$(*3*) ‹*t-P′* ∈ $\mathcal{T}$ *P*› **have** ‹*u′* ∈ $\mathcal{T}$ (*P* $[\![A]\!]_{✓}$ *Q*)›
**by** (*auto simp add*: *T-Sync*$_{ptick}$)
**moreover have** ‹*n* ≤ *length u′*›
**using** $(*3*) $$(*3*) *setinterleaves*$_{ptick}$*-imp-lengthLR-le* **by** *blast*
**ultimately have** ‹*u′* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**by** (*metis* $$(*1*) *D-restriction-process*$_{ptick}$*I Prefix-Order.prefixE*
*assms*(*1*) *nless-le tickFree-append-iff*)
**thus** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{✓}$ *Q* ↓ *n*)›
**by** (*metis* $$(*1*) *Prefix-Order.prefixE assms*(*1*,*2*) *is-processT7*
*tickFree-append-iff tickFree-imp-front-tickFree*)
**next**
**fix** *u′ t-P‴* **assume** $$ : ‹*u′* ≤ *u*› ‹*t-P‴* ≤ *t-P′*›
‹*u′ setinterleaves*$_{✓}$ *tick-join* ((*t-P‴*, *t-Q′*), *A*)›
**from** $(*2*) $$(*2*) *is-processT3-TR* **have** ‹*t-P‴* ∈ $\mathcal{T}$ *P*› **by** *blast*
**with** $$(*3*) ‹*t-Q′* ∈ $\mathcal{T}$ *Q*› **have** ‹*u′* ∈ $\mathcal{T}$ (*P* $[\![A]\!]_{✓}$ *Q*)›
**by** (*auto simp add*: *T-Sync*$_{ptick}$)
**moreover have** ‹*n* ≤ *length u′*›

**using** $(3) $$(3) *setinterleaves*$_{ptick}$*-imp-lengthLR-le* **by** *blast*
**ultimately have** ‹$u' \in \mathcal{D}\ (P\ [\![A]\!]_{\checkmark}\ Q \downarrow n)$›
**by** (*metis* $$(1) *D-restriction-process*$_{ptick}$*I Prefix-Order.prefixE assms(1) nless-le tickFree-append-iff*)
**thus** ‹$u \mathbin{@} v \in \mathcal{D}\ (P\ [\![A]\!]_{\checkmark}\ Q \downarrow n)$›
**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* :
‹$P\ [\![A]\!]_{\checkmark}\ Q \downarrow n \sqsubseteq_{FD} (P \downarrow n)\ [\![A]\!]_{\checkmark}\ (Q \downarrow n)$› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
**proof** (*unfold refine-defs*, *safe*)
**show** ‹$t \in \mathcal{D}$ *?rhs* $\Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** $t$
**by** (*unfold D-Sync*$_{ptick}$, *safe*)
(*solves* ‹*simp add*: *restriction-process*$_{ptick}$*-Sync*$_{ptick}$*-FD-div-oneside*›,
*metis Sync*$_{ptick}$*-locale-sym.restriction-process*$_{ptick}$*-Sync*$_{ptick}$*-FD-div-oneside Sync*$_{ptick}$*-sym setinterleaves*$_{ptick}$*-sym*)
**thus** ‹$(t, X) \in \mathcal{F}\ ((P \downarrow n)\ [\![A]\!]_{\checkmark}\ (Q \downarrow n)) \Longrightarrow (t, X) \in \mathcal{F}\ (P\ [\![A]\!]_{\checkmark}\ Q \downarrow n)$› **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*$_{ptick}$ :
‹$P\ [\![A]\!]_{\checkmark}\ Q \downarrow n = (P \downarrow n)\ [\![A]\!]_{\checkmark}\ (Q \downarrow n)$› (**is** ‹*?lhs* = *?rhs*›)
**if** ‹$\boldsymbol{\alpha}(P) \subseteq A \vee \boldsymbol{\alpha}(Q) \subseteq A$›
**proof** (*rule FD-antisym*)
**show** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*› **by** (*fact restriction-process*$_{ptick}$*-Sync*$_{ptick}$*-FD*)
**next**
**have** *div* : ‹$t \in \mathcal{D}\ (P\ [\![A]\!]_{\checkmark}\ Q) \Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** $t$
**by** (*auto simp add*: *D-Sync*$_{ptick}$ *restriction-process*$_{ptick}$*-projs*)

**{ fix** $t\ u\ v$ **assume** ‹$t = u \mathbin{@} v$› ‹$u \in \mathcal{T}\ (P\ [\![A]\!]_{\checkmark}\ Q)$› ‹*length* $u = n$› ‹*tF* $u$› ‹*ftF* $v$›
**from** *this*(2) **consider** ‹$u \in \mathcal{D}\ (P\ [\![A]\!]_{\checkmark}\ Q)$›
| *t-P t-Q* **where** ‹*t-P* $\in \mathcal{T}$ $P$› ‹*t-Q* $\in \mathcal{T}$ $Q$›
‹$u$ *setinterleaves*$_{\checkmark}$*$_{tick\text{-}join}$* ((*t-P*, *t-Q*), $A$)›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**hence** ‹$t \in \mathcal{D}$ *?rhs*›
**proof** *cases*
**show** ‹$u \in \mathcal{D}\ (P\ [\![A]\!]_{\checkmark}\ Q) \Longrightarrow t \in \mathcal{D}$ *?rhs*›
**by** (*simp add*: ‹*ftF* $v$› ‹$t = u \mathbin{@} v$› ‹*tF* $u$› *div is-processT7*)
**next**
**fix** *t-P t-Q* **assume** ‹*t-P* $\in \mathcal{T}$ $P$› ‹*t-Q* $\in \mathcal{T}$ $Q$›

**and** *setinter* : ‹*u setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P*, *t-Q*), *A*)›
**consider** ‹*t-P* ∈ $\mathcal{D}$ *P* ∨ *t-Q* ∈ $\mathcal{D}$ *Q*› | ‹*t-P* ∉ $\mathcal{D}$ *P*› ‹*t-Q* ∉ $\mathcal{D}$ *Q*› **by** *blast*
**thus** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**proof** *cases*
**assume** ‹*t-P* ∈ $\mathcal{D}$ *P* ∨ *t-Q* ∈ $\mathcal{D}$ *Q*›
**with** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› *setinter* ‹*ftF v*› ‹*t* = *u* @ *v*› ‹*tF u*›
**have** ‹*t* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{\checkmark}$ *Q*)› **by** (*auto simp add*: *D-Sync*$_{ptick}$)
**thus** ‹*t* ∈ $\mathcal{D}$ *?rhs*› **by** (*fact div*)
**next**
**assume** ‹*t-P* ∉ $\mathcal{D}$ *P*› ‹*t-Q* ∉ $\mathcal{D}$ *Q*›
**with** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› ‹$\boldsymbol{\alpha}$(*P*) ⊆ *A* ∨ $\boldsymbol{\alpha}$(*Q*) ⊆ *A*›
**have** ‹{*a*. *ev a* ∈ *set t-P*} ⊆ *A* ∨ {*a*. *ev a* ∈ *set t-Q*} ⊆ *A*›
**by** (*auto dest*: *subsetD intro*: *strict-events-of-memI*)
**with** *setinterleaves*$_{ptick}$-*subsetL*[*OF* ‹*tF u*› - *setinter*]
*setinterleaves*$_{ptick}$-*subsetR*[*OF* ‹*tF u*› - *setinter*]
**have** ‹*u* = *map ev* (*map of-ev t-P*) ∨ *u* = *map ev* (*map of-ev t-Q*)› **by** *blast*
**with** ‹*length u* = *n*› **have** ‹*length t-P* = *n* ∨ *length t-Q* = *n*› **by** *auto*
**moreover from** ‹*tF u*› *tickFree-setinterleaves*$_{ptick}$-*iff*[*OF setinter*]
**have** ‹*tF t-P*› ‹*tF t-Q*› **by** *simp-all*
**ultimately have** ‹*t-P* ∈ $\mathcal{D}$ (*P* ↓ *n*) ∨ *t-Q* ∈ $\mathcal{D}$ (*Q* ↓ *n*)›
**using** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› **by** (*metis D-restriction-process*$_{ptick}$*I*)
**moreover from** ‹*t-P* ∈ $\mathcal{T}$ *P*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*›
**have** ‹*t-P* ∈ $\mathcal{T}$ (*P* ↓ *n*)› ‹*t-Q* ∈ $\mathcal{T}$ (*Q* ↓ *n*)›
**by** (*simp-all add*: *T-restriction-process*$_{ptick}$*I*)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**using** ‹*ftF v*› ‹*t* = *u* @ *v*› ‹*tF u*› *setinter* **by** (*auto simp add*: *D-Sync*$_{ptick}$)
**qed**
**qed**
} **note** ∗ = *this*

**show** ‹*?rhs* $\sqsubseteq_{FD}$ *?lhs*›
**proof** (*unfold refine-defs*, *safe*)
**show** ‹*t* ∈ $\mathcal{D}$ *?lhs* ⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **for** *t*
**proof** (*elim D-restriction-process*$_{ptick}$*E*)
**show** ‹*t* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{\checkmark}$ *Q*) ⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **by** (*fact div*)
**next**
**show** ‹⟦*t* = *u* @ *v*; *u* ∈ $\mathcal{T}$ (*P* $[\![A]\!]_{\checkmark}$ *Q*); *length u* = *n*; *tF u*; *ftF v*⟧
⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **for** *u v* **by** (*fact* ∗)
**qed**
**next**
**show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› **for** *t X*
**proof** (*elim F-restriction-process*$_{ptick}$*E*)
**assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* $[\![A]\!]_{\checkmark}$ *Q*)›
**then consider** ‹*t* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{\checkmark}$ *Q*)›
| (*fail*) *t-P t-Q X-P X-Q* **where** ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
‹*t setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P*, *t-Q*), *A*)›
‹*X* ⊆ *super-ref-Sync*$_{ptick}$ *tick-join X-P A X-Q*›
**unfolding** *Sync*$_{ptick}$-*projs* **by** *blast*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs*›

**proof** *cases*
**from** *div D-F* **show** ‹$t \in \mathcal{D}\ (P\ [\![A]\!]_{\checkmark}\ Q) \Longrightarrow (t,\ X) \in \mathcal{F}$ *?rhs*› **by** *blast*
**next**
**case** *fail*
**thus** ‹$(t,\ X) \in \mathcal{F}$ *?rhs*›
**by** (*auto simp add*: *F-Sync*$_{ptick}$ *F-restriction-process*$_{ptick}$)
**qed**
**next**
**show** ‹$[\![t = u\ @\ v;\ u \in \mathcal{T}\ (P\ [\![A]\!]_{\checkmark}\ Q);$ *length* $u = n;$ *tF* $u;$ *ftF* $v]\!]$
$\Longrightarrow (t,\ X) \in \mathcal{F}$ *?rhs*› **for** $u\ v$ **by** (*simp add*: $*$ *is-processT8*)
**qed**
**qed**
**qed**

**corollary** *restriction-process*$_{ptick}$*-MultiSync*$_{ptick}$*-FD* :
‹$[\![A]\!]_{\checkmark}\ l \in@\ L.\ P\ l \downarrow n \sqsubseteq_{FD} [\![A]\!]_{\checkmark}\ l \in@\ L.\ (P\ l \downarrow n)$›
**proof** (*induct L rule*: *induct-list012*)
**show** ‹$[\![A]\!]_{\checkmark}\ l \in@\ [].\ P\ l \downarrow n \sqsubseteq_{FD} [\![A]\!]_{\checkmark}\ l \in@\ [].\ (P\ l \downarrow n)$› **by** *simp*
**next**
**show** ‹$[\![A]\!]_{\checkmark}\ l \in@\ [l1].\ P\ l \downarrow n \sqsubseteq_{FD} [\![A]\!]_{\checkmark}\ l \in@\ [l1].\ (P\ l \downarrow n)$› **for** *l1*
**by** (*simp add*: *restriction-process*$_{ptick}$*-Renaming*)
**next**
**fix** *l1 l2 L*
**assume** *hyp* : ‹$[\![A]\!]_{\checkmark}\ l \in@\ (l2\ \#\ L).\ P\ l \downarrow n \sqsubseteq_{FD} [\![A]\!]_{\checkmark}\ l \in@\ (l2\ \#\ L).\ (P\ l \downarrow n)$›
**show** ‹$[\![A]\!]_{\checkmark}\ l \in@\ (l1\ \#\ l2\ \#\ L).\ P\ l \downarrow n \sqsubseteq_{FD} [\![A]\!]_{\checkmark}\ l \in@\ (l1\ \#\ l2\ \#\ L).\ (P\ l \downarrow n)$›
**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 $\boldsymbol{\alpha}(P) \subseteq A \vee \boldsymbol{\alpha}(Q) \subseteq A \Longrightarrow P\ [\![A]\!]_{\checkmark}\ Q \downarrow n = (P \downarrow n)\ [\![A]\!]_{\checkmark}\ (Q \downarrow n)$ 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*$_{ptick}$ :
‹$[\![A]\!]_{\checkmark}\ l \in@\ L.\ P\ l \downarrow n = (\mathit{if}\ n = 0\ \mathit{then}\ \bot\ \mathit{else}\ [\![A]\!]_{\checkmark}\ l \in@\ L.\ (P\ l \downarrow n))$›
— *if* $n = 0$ *then* $\bot$ *else* - is necessary because we can have $L = []$.
**if** ‹$\bigwedge l.\ l \in$ *set* $(tl\ L) \Longrightarrow \boldsymbol{\alpha}(P\ l) \subseteq A$›
**proof** (*split if-split*, *intro conjI impI*)
**show** ‹$n = 0 \Longrightarrow [\![A]\!]_{\checkmark}\ l \in@\ L.\ P\ l \downarrow n = \bot$› **by** *simp*
**next**
**from** *that* **show** ‹$[\![A]\!]_{\checkmark}\ l \in@\ L.\ P\ l \downarrow n = [\![A]\!]_{\checkmark}\ l \in@\ L.\ (P\ l \downarrow n)$› **if** ‹$n \neq 0$›
**proof** (*induct L rule*: *induct-list012*)
**case** *1* **show** *?case* **by** (*simp add*: ‹$n \neq 0$›)
**next**

**case** (*2 l1*) **show** *?case* **by** (*simp add*: *restriction-process*$_{ptick}$*-Renaming*)
**next**
**case** (*3 l1 l2 L*)
**from** *3.prems* **have** $*$ : ‹$\alpha$(*MultiSync*$_{ptick}$ *A* (*l2* # *L*) *P*) $\subseteq$ *A*›
**by** (*intro subset-trans*[*OF strict-events-of-MultiSync*$_{ptick}$*-subset*]) *auto*
**have** ‹$[\![A]\!]_{\checkmark}$ *l* ∈@ (*l1* # *l2* # *L*). *P l* ↓ *n* =
*P l1* $[\![A]\!]_{\checkmark Rlist}$ $[\![A]\!]_{\checkmark}$ *l* ∈@ (*l2* # *L*). *P l* ↓ *n*› **by** *simp*
**also have** ‹... = (*P l1* ↓ *n*) $[\![A]\!]_{\checkmark Rlist}$ ($[\![A]\!]_{\checkmark}$ *l* ∈@ (*l2* # *L*). *P l* ↓ *n*)›
**by** (*simp add*: *Sync*$_{Rlist}$*.strict-events-of-subset-restriction-process*$_{ptick}$*-Sync*$_{ptick}$
$*$)
**also have** ‹... = (*P l1* ↓ *n*) $[\![A]\!]_{\checkmark Rlist}$ $[\![A]\!]_{\checkmark}$ *l* ∈@ (*l2* # *L*). (*P l* ↓ *n*)›
**using** *3.hyps*(*2*) *3.prems* **by** *auto*
**also have** ‹... = $[\![A]\!]_{\checkmark}$ *l* ∈@ (*l1* # *l2* # *L*). (*P l* ↓ *n*)› **by** *simp*
**finally show** *?case* **.**
**qed**
**qed**

**corollary** (**in** *Sync*$_{ptick}$*-locale*) *restriction-process*$_{ptick}$*-Par*$_{ptick}$ :
‹*P* $||_{\checkmark}$ *Q* ↓ *n* = (*P* ↓ *n*) $||_{\checkmark}$ (*Q* ↓ *n*)›
**by** (*simp add*: *strict-events-of-subset-restriction-process*$_{ptick}$*-Sync*$_{ptick}$)

**corollary** *restriction-process*$_{ptick}$*-MultiPar*$_{ptick}$ :
‹$||_{\checkmark}$ *l* ∈@ *L*. *P l* ↓ *n* = (*if n* = *0 then* ⊥ *else* $||_{\checkmark}$ *l* ∈@ *L*. (*P l* ↓ *n*))›
**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* :
‹*non-destructive* (λ(*P*, *Q*). *P* $[\![A]\!]_{\checkmark}$ *Q*)›
**proof** (*rule order-non-destructiveI*, *clarify*)
**fix** *P P'* :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** *Q Q'* :: ‹('*a*, '*s*) *process*$_{ptick}$› **and** *n*
**assume** ‹(*P*, *Q*) ↓ *n* = (*P'*, *Q'*) ↓ *n*›
**hence** ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*Q* ↓ *n* = *Q'* ↓ *n*›
**by** (*simp-all add*: *restriction-prod-def*)
**show** ‹*P* $[\![A]\!]_{\checkmark}$ *Q* ↓ *n* $\sqsubseteq_{FD}$ *P'* $[\![A]\!]_{\checkmark}$ *Q'* ↓ *n*›
**proof** (*rule leFD-restriction-process*$_{ptick}$*I*)
**show** ‹*t* ∈ $\mathcal{D}$ (*P'* $[\![A]\!]_{\checkmark}$ *Q'*) ⟹ *t* ∈ $\mathcal{D}$ (*P* $[\![A]\!]_{\checkmark}$ *Q* ↓ *n*)› **for** *t*
**by** (*metis* (*no-types*, *lifting*) ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*Q* ↓ *n* = *Q'* ↓ *n*› *in-mono*
*le-ref1 mono-Sync*$_{ptick}$*-FD*
*restriction-process*$_{ptick}$*-FD-self restriction-process*$_{ptick}$*-Sync*$_{ptick}$*-FD*)
**next**
**show** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P'* $[\![A]\!]_{\checkmark}$ *Q'*) ⟹ (*t*, *X*) ∈ $\mathcal{F}$ (*P* $[\![A]\!]_{\checkmark}$ *Q* ↓ *n*)› **for** *t X*
**by** (*metis* (*no-types*, *lifting*) ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*Q* ↓ *n* = *Q'* ↓ *n*› *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** $Sync_{ptick}$-*locale*) $Sync_{ptick}$-*restriction-shift-process*$_{ptick}$
[*restriction-shift-process*$_{ptick}$-*simpset*, *simp*] :
‹*non-destructive f* $\Longrightarrow$ *non-destructive g* $\Longrightarrow$ *non-destructive* ($\lambda x$. *f x* $[\![S]\!]_{\checkmark}$ *g x*)›
‹*constructive f* $\Longrightarrow$ *constructive g* $\Longrightarrow$ *constructive* ($\lambda x$. *f x* $[\![S]\!]_{\checkmark}$ *g x*)›
**by** (*fact non-destructive-comp-non-destructive*
[*OF* $Sync_{ptick}$-*non-destructive non-destructive-prod-codomain*, *simplified*])
(*fact non-destructive-comp-constructive*
[*OF* $Sync_{ptick}$-*non-destructive constructive-prod-codomain*, *simplified*])

**lemma** *MultiSync*$_{ptick}$-*restriction-shift-process*$_{ptick}$
[*restriction-shift-process*$_{ptick}$-*simpset*, *simp*] :
‹($\bigwedge l$. *l* $\in$ *set L* $\Longrightarrow$ *non-destructive* (*f l*)) $\Longrightarrow$ *non-destructive* ($\lambda x$. $[\![S]\!]_{\checkmark}$ *l* $\in@$ *L*. *f l x*)›
‹($\bigwedge l$. *l* $\in$ *set L* $\Longrightarrow$ *constructive* (*f l*)) $\Longrightarrow$ *constructive* ($\lambda x$. $[\![S]\!]_{\checkmark}$ *l* $\in@$ *L*. *f l x*)›
**by** (*induct L rule*: *induct-list012*; *simp*)+

**corollary** *MultiSync*$_{ptick}$-*non-destructive* : ‹*non-destructive* ($\lambda P$. $[\![S]\!]_{\checkmark}$ *l* $\in@$ *L*. *P l*)›
**by** (*rule MultiSync*$_{ptick}$-*restriction-shift-process*$_{ptick}$(*1*)[*of L* ‹$\lambda m$ *x*. *x m*›]) *simp*

# Chapter 16

# Other Laws

**declare** [[*metis-instantiate*]]

## 16.1 Laws of Renaming

### 16.1.1 Renaming and Sequential Composition

**lemma** *FD-Renaming-Seq*$_{ptick}$ :
‹*Renaming P f g* ;$_{\checkmark}$ (λ*g-r*. ⊓*r* ∈ {*r* ∈ ✓**s**(*P*). *g-r* = *g r*}. *Renaming* (*Q r*) *f g*′)
⊑$_{FD}$ *Renaming* (*P* ;$_{\checkmark}$ *Q*) *f g*′› (**is** ‹*?lhs* ⊑$_{FD}$ *?rhs*›)
**proof** (*rule failure-divergence-refine-optimizedI*)
**fix** *s* **assume** ‹*s* ∈ 𝒟 *?rhs*›
**then obtain** *s1 s2* **where** ∗ : ‹*s* = *map* (*map-event*$_{ptick}$ *f g*′) *s1* @ *s2*› ‹*tF s1*›
‹*ftF s2*› ‹*s1* ∈ 𝒟 (*P* ;$_{\checkmark}$ *Q*)›
**unfolding** *D-Renaming* **by** *blast*
**from** ∗(*4*) **consider** (*D-P*) *t1 t2* **where** ‹*s1* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2*› ‹*t1* ∈ 𝒟 *P*› ‹*tF t1*› ‹*ftF t2*›
| (*D-Q*) *t1 r t2* **where** ‹*s1* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2*› ‹*t1* @ [✓(*r*)] ∈ 𝒯 *P*› ‹*t1* ∉ 𝒟 *P*› ‹*tF t1*› ‹*t2* ∈ 𝒟 (*Q r*)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*) (*metis D-imp-front-tickFree*)
**thus** ‹*s* ∈ 𝒟 *?lhs*›
**proof** *cases*
**case** *D-P*
**from** *D-P*(*2*, *3*) **have** ‹*map* (*map-event*$_{ptick}$ *f g*) *t1* ∈ 𝒟 (*Renaming P f g*)›
**by** (*auto simp add*: *D-Renaming intro*: *front-tickFree-Nil*)
**hence** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) ∈ 𝒟 *?lhs*›
**unfolding** *Seq*$_{ptick}$*-projs*
**by** (*metis* (*mono-tags*, *lifting*) *front-tickFree-Nil D-P*(*3*)
*map-event*$_{ptick}$*-tickFree append.right-neutral mem-Collect-eq Un-iff*)
**also have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) =
*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1*)›
**by** (*simp add*: ‹*tF t1*› *tickFree-map-map-event*$_{ptick}$*-is*)
**finally show** ‹*s* ∈ 𝒟 *?lhs*›

**by** (*auto simp add*: ∗(*1*) *D-P*(*1*) *intro*!: *is-processT7*)
(*metis list.map-comp map-event*$_{ptick}$*-tickFree tickFree-map-ev-comp*,
*use* ∗(*2*, *3*) *D-P*(*1*) *front-tickFree-append map-event*$_{ptick}$*-tickFree tick-Free-append-iff* **in** *blast*)
**next**
**case** *D-Q*
**from** ∗(*2*) *D-Q*(*1*, *5*) **have** ‹*map* (*map-event*$_{ptick}$ *f g*′) *t2* ∈ $\mathcal{D}$ (*Renaming* (*Q r*) *f g*′)›
**by** (*auto simp add*: *D-Renaming intro*: *front-tickFree-Nil*)
**hence** ‹*map* (*map-event*$_{ptick}$ *f g*′) *t2* ∈ $\mathcal{D}$ (⊓*r*′ ∈ {*r*′ ∈ ✓**s**(*P*). *g r* = *g r*′}. *Renaming* (*Q r*′) *f g*′)›
**by** (*simp add*: *D-GlobalNdet*)
(*metis D-Q*(*2*, *3*) *is-processT9 strict-ticks-of-memI*)
**moreover from** *D-Q*(*2*) **have** ‹*map* (*map-event*$_{ptick}$ *f g*) *t1* @ [✓(*g r*)] ∈ $\mathcal{T}$ (*Renaming P f g*)›
**by** (*auto simp add*: *T-Renaming*)
**moreover have** ‹*tF* (*map* (*map-event*$_{ptick}$ *f g*) *t1*)›
**by** (*simp add*: *D-Q*(*4*) *map-event*$_{ptick}$*-tickFree*)
**ultimately have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) @
*map* (*map-event*$_{ptick}$ *f g*′) *t2* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *Seq*$_{ptick}$*-projs* **by** *blast*
**with** ∗(*2*, *3*) **have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) @
*map* (*map-event*$_{ptick}$ *f g*′) *t2* @ *s2* ∈ $\mathcal{D}$ *?lhs*›
**by** (*auto simp add*: *D-Q*(*1*) *comp-assoc map-event*$_{ptick}$*-tickFree*
*intro*!: *is-processT7*[*of* ‹- @ ->, *simplified*])
**also from** *D-Q*(*4*) **have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) @
*map* (*map-event*$_{ptick}$ *f g*′) *t2* @ *s2* = *s*›
**by** (*simp add*: ∗(*1*) *D-Q*(*1*))
(*metis event*$_{ptick}$*.map-sel*(*1*) *in-set-conv-decomp tickFree-Cons-iff tick-Free-append-iff*)
**finally show** ‹*s* ∈ $\mathcal{D}$ *?lhs*› **.**
**qed**
**next**
**assume** *subset-div* : ‹$\mathcal{D}$ *?rhs* ⊆ $\mathcal{D}$ *?lhs*›
**fix** *s X* **assume** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**then consider** ‹*s* ∈ $\mathcal{D}$ *?rhs*›
| (*fail*) *s1* **where** ‹*s* = *map* (*map-event*$_{ptick}$ *f g*′) *s1*›
‹(*s1*, *map-event*$_{ptick}$ *f g*′ −‘ *X*) ∈ $\mathcal{F}$ (*P* ;✓ *Q*)› ‹*s1* ∉ $\mathcal{D}$ (*P* ;✓ *Q*)›
**by** (*simp add*: *Renaming-projs*)
(*metis* (*no-types*, *opaque-lifting*) *front-tickFree-Nil front-tickFree-iff-tickFree-butlast*
*front-tickFree-Cons-iff*[*of* ‹*last s*› ‹[]›] *map-butlast*[*of* ‹*map-event*$_{ptick}$ *f g*′›]
*map-is-Nil-conv*[*of* ‹*map-event*$_{ptick}$ *f g*′› ‹[]›] *map-is-Nil-conv*[*of* ‹*map-event*$_{ptick}$ *f g*′›]
*append-self-conv*[*of* ‹*map* (*map-event*$_{ptick}$ *f g*′) -› ‹[]›] *F-imp-front-tickFree*
*snoc-eq-iff-butlast*[*of* ‹*butlast s*› ‹*last s*› *s*]
*div-butlast-when-non-tickFree-iff non-tickFree-imp-not-Nil*)
**thus** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**from** *subset-div D-F* **show** ‹*s* ∈ $\mathcal{D}$ *?rhs* ⟹ (*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** *blast*

**next**
**case** *fail*
**from** *fail*(2, 3)
**consider** (*F-P*) *t1* **where** ‹*s1* = *map* (*ev* ∘ *of-ev*) *t1*› ‹(*t1*, *ref-Seq*$_{ptick}$ (*map-event*$_{ptick}$ *f g*′ −‘ *X*)) ∈ $\mathcal{F}$ *P*› ‹*tF t1*›
| (*F-Q*) *t1 r t2* **where** ‹*s1* = *map* (*ev* ∘ *of-ev*) *t1* @ *t2*› ‹*t1* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*› ‹*tF t1*› ‹*t1* ∉ $\mathcal{D}$ *P*› ‹(*t2*, *map-event*$_{ptick}$ *f g*′ −‘ *X*) ∈ $\mathcal{F}$ (*Q r*)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*) (*metis F-imp-front-tickFree*)
**thus** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**case** *F-P*
**have** ‹*map-event*$_{ptick}$ *f g* −‘ (*ref-Seq*$_{ptick}$ *X*) = *ref-Seq*$_{ptick}$ (*map-event*$_{ptick}$ *f g*′ −‘ *X*)› **for** *X*
**proof** (*rule set-eqI*)
**show** ‹*e* ∈ *map-event*$_{ptick}$ *f g* −‘ (*ref-Seq*$_{ptick}$ *X*) ⟷
*e* ∈ *ref-Seq*$_{ptick}$ (*map-event*$_{ptick}$ *f g*′ −‘ *X*)› **for** *e*
**by** (*cases e*, *auto simp add*: *ref-Seq*$_{ptick}$*-def image-iff*)
(*metis Int-iff event*$_{ptick}$*.sel*(1) *event*$_{ptick}$*.simps*(9) *rangeI vimage-eq*,
*metis IntI UNIV-I event*$_{ptick}$*.sel*(1) *image-eqI*)
**qed**
**with** *F-P*(2) **have** ‹(*map* (*map-event*$_{ptick}$ *f g*) *t1*, *ref-Seq*$_{ptick}$ *X*) ∈ $\mathcal{F}$ (*Renaming P f g*)›
**by** (*auto simp add*: *F-Renaming*)
**with** *F-P*(3) **have** ‹(*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*), *X*) ∈ $\mathcal{F}$ *?lhs*›
**by** (*fastforce simp add*: *Seq*$_{ptick}$*-projs map-event*$_{ptick}$*-tickFree*)
**also have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) = *s*›
**by** (*simp add*: *fail*(1) *F-P*(1))
(*metis F-P*(3) *event*$_{ptick}$*.map-sel*(1) *in-set-conv-decomp tickFree-Cons-iff tickFree-append-iff*)
**finally show** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› .
**next**
**case** *F-Q*
**with** *F-Q*(4) **have** ‹(*map* (*map-event*$_{ptick}$ *f g*′) *t2*, *X*) ∈ $\mathcal{F}$ (*Renaming* (*Q r*) *f g*′)›
**by** (*auto simp add*: *F-Renaming*)
**hence** ‹(*map* (*map-event*$_{ptick}$ *f g*′) *t2*, *X*) ∈
$\mathcal{F}$ (⊓*r*′ ∈ {*r*′ ∈ ✓**s**(*P*). *g r* = *g r*′}. *Renaming* (*Q r*′) *f g*′)›
**by** (*simp add*: *F-GlobalNdet*)
(*metis F-Q*(2, 4) *is-processT9 strict-ticks-of-memI*)
**moreover from** *F-Q*(2) **have** ‹*map* (*map-event*$_{ptick}$ *f g*) *t1* @ [✓(*g r*)] ∈ $\mathcal{T}$ (*Renaming P f g*)›
**by** (*auto simp add*: *T-Renaming*)
**moreover have** ‹*tF* (*map* (*map-event*$_{ptick}$ *f g*) *t1*)›
**by** (*simp add*: *F-Q*(3) *map-event*$_{ptick}$*-tickFree*)
**ultimately have** ‹(*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) @
*map* (*map-event*$_{ptick}$ *f g*′) *t2*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**unfolding** *Seq*$_{ptick}$*-projs* **by** *fast*
**also have** ‹*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*) @ *map* (*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** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› **.**
**qed**
**qed**
**qed**

**lemma** *inj-on-Renaming-Seq*$_{ptick}$ :
‹*Renaming* (*P* ;$_{\checkmark}$ *Q*) *f g*′ =
*Renaming P f g* ;$_{\checkmark}$ (λ*g-r*. *Renaming* (*Q* (*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*)) *f g*′)›
(**is** ‹*?lhs* = *?rhs*›) **if** ‹*inj-on g* ✓**s**(*P*)›
— This assumption is necessary, otherwise we cannot know which tick triggered *Q*.
**proof** (*rule FD-antisym*)
**show** ‹*?lhs* ⊑$_{FD}$ *?rhs*›
**proof** (*rule failure-divergence-refine-optimizedI*)
**fix** *s* **assume** ‹*s* ∈ $\mathcal{D}$ *?rhs*›
**then consider** (*D-P*) *s1 s2* **where** ‹*s* = *map* (*ev* ∘ *of-ev*) *s1* @ *s2*› ‹*s1* ∈ $\mathcal{D}$ (*Renaming P f g*)› ‹*tF s1*› ‹*ftF s2*›
| (*D-Q*) *s1 g-r s2* **where** ‹*s* = *map* (*ev* ∘ *of-ev*) *s1* @ *s2*› ‹*s1* @ [✓(*g-r*)] ∈ $\mathcal{T}$ (*Renaming P f g*)›
‹*s1* ∉ $\mathcal{D}$ (*Renaming P f g*)› ‹*tF s1*› ‹*s2* ∈ $\mathcal{D}$ (*Renaming* (*Q* (*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*)) *f g*′)›
**by** (*simp add*: $Seq_{ptick}$-*projs*) (*use D-imp-front-tickFree* **in** *blast*)
**thus** ‹*s* ∈ $\mathcal{D}$ *?lhs*›
**proof** *cases*
**case** *D-P*
**from** *D-P*(*2*) **obtain** *t1 t2*
**where** ∗ : ‹*s1* = *map* (*map-event*$_{ptick}$ *f g*) *t1* @ *t2*› ‹*tF t1*› ‹*ftF t2*› ‹*t1* ∈ $\mathcal{D}$ *P*›
**unfolding** *D-Renaming* **by** *blast*
**from** ∗(*2*, *4*) **have** ‹*map* (*ev* ∘ *of-ev*) *t1* ∈ $\mathcal{D}$ (*P* ;$_{\checkmark}$ *Q*)›
**by** (*auto simp add*: $Seq_{ptick}$-*projs intro*: *front-tickFree-Nil*)
**hence** ‹*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1*) ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming mem-Collect-eq*
**by** (*metis* (*mono-tags*, *lifting*) *front-tickFree-Nil tickFree-map-ev-comp append.right-neutral*)
**also have** ‹*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1*) =
*map* (*ev* ∘ *of-ev*) (*map* (*map-event*$_{ptick}$ *f g*) *t1*)›
**by** *simp* (*metis* ∗(*2*) $event_{ptick}$.*map-sel*(*1*) *in-set-conv-decomp tickFree-Cons-iff tickFree-append-iff*)
**finally show** ‹*s* ∈ $\mathcal{D}$ *?lhs*›
**by** (*auto simp add*: *D-P*(*1*, *4*) ∗(*1*) *front-tickFree-append comp-assoc intro*!: *is-processT7*)
**next**

**case** *D-Q*
**have** ‹*s1* @ [✓(*g-r*)] ∉ $\mathcal{D}$ (*Renaming P f g*)› **by** (*meson D-Q*(*3*) *is-processT9*)
**with** *D-Q*(*2−4*) **obtain** *t1 r*
**where** ∗ : ‹*g-r* = *g r*› ‹*r* ∈ ✓**s**(*P*)› ‹*s1* = *map* (*map-event*$_{ptick}$ *f g*) *t1*› ‹*t1* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*›
**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*) ‹*inj-on g* ✓**s**(*P*)› **have** ‹(*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*) = *r*›
**by** (*auto dest*: *inj-onD*)
**with** *D-Q*(*5*) **have** ‹*s2* ∈ $\mathcal{D}$ (*Renaming* (*Q r*) *f g*′)› **by** *simp*
**then obtain** *t2 t3*
**where** ∗∗ : ‹*s2* = *map* (*map-event*$_{ptick}$ *f g*′) *t2* @ *t3*› ‹*tF t2*› ‹*ftF t3*› ‹*t2* ∈ $\mathcal{D}$ (*Q r*)›
**unfolding** *D-Renaming* **by** *blast*
**from** ∗(*4*) ∗∗(*4*) **have** ‹*map* (*ev* ∘ *of-ev*) *t1* @ *t2* ∈ $\mathcal{D}$ (*P* ;✓ *Q*)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*) (*metis append-T-imp-tickFree not-Cons-self*)
**with** ∗∗(*2*) **have** ‹*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1* @ *t2*) ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming mem-Collect-eq*
**by** (*metis append.right-neutral front-tickFree-Nil tickFree-append-iff tick-Free-map-ev-comp*)
**moreover have** ‹*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1* @ *t2*) @ *t3* = *s*›
**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** ‹*s* ∈ $\mathcal{D}$ *?lhs*›
**by** (*auto simp add*: ∗∗(*3*) *intro*!: *is-processT7*[*of* ‹- @ -›, *simplified*])
(*use* ∗∗(*1*) *D-Q*(*1*) **in** *force*, *use* ∗∗(*2*) *map-event*$_{ptick}$*-tickFree* **in** *blast*)
**qed**
**next**
**assume** *subset-div* : ‹$\mathcal{D}$ *?rhs* ⊆ $\mathcal{D}$ *?lhs*›
**fix** *s X* **assume** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?rhs*›
**then consider** ‹*s* ∈ $\mathcal{D}$ *?rhs*›
| (*F-P*) *s1* **where** ‹*s* = *map* (*ev* ∘ *of-ev*) *s1*› ‹(*s1*, *ref-Seq*$_{ptick}$ *X*) ∈ $\mathcal{F}$ (*Renaming P f g*)› ‹*s1* ∉ $\mathcal{D}$ (*Renaming P f g*)› ‹*tF s1*›
| (*F-Q*) *s1 g-r s2* **where** ‹*s* = *map* (*ev* ∘ *of-ev*) *s1* @ *s2*› ‹*s1* @ [✓(*g-r*)] ∈ $\mathcal{T}$ (*Renaming P f g*)›
‹*s1* ∉ $\mathcal{D}$ (*Renaming P f g*)› ‹*tF s1*› ‹(*s2*, *X*) ∈ $\mathcal{F}$ (*Renaming* (*Q* (*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*)) *f g*′)›
‹*s2* ∉ $\mathcal{D}$ (*Renaming* (*Q* (*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*)) *f g*′)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*)
(*metis* (*no-types*, *lifting*) *F-imp-front-tickFree front-tickFree-charn self-append-conv*)
**thus** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*›
**proof** *cases*
**from** *subset-div D-F* **show** ‹*s* ∈ $\mathcal{D}$ *?rhs* ⟹ (*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** *blast*
**next**
**case** *F-P*

**from** *F-P*(*2*, *3*) **obtain** *t1*
**where** ∗ : ‹*s1* = *map* (*map-event*$_{ptick}$ *f g*) *t1*› ‹(*t1*, *map-event*$_{ptick}$ *f g* −‘ *ref-Seq*$_{ptick}$ *X*) ∈ $\mathcal{F}$ *P*›
**unfolding** *Renaming-projs* **by** *blast*
**have** ‹*map-event*$_{ptick}$ *f g* −‘ (*ref-Seq*$_{ptick}$ *X*) = *ref-Seq*$_{ptick}$ (*map-event*$_{ptick}$ *f g*′ −‘ *X*)› **for** *X*
**proof** (*rule set-eqI*)
**show** ‹*e* ∈ *map-event*$_{ptick}$ *f g* −‘ (*ref-Seq*$_{ptick}$ *X*) ⟷
*e* ∈ *ref-Seq*$_{ptick}$ (*map-event*$_{ptick}$ *f g*′ −‘ *X*)› **for** *e*
**by** (*cases e*, *auto simp add*: *ref-Seq*$_{ptick}$*-def image-iff*)
(*metis Int-iff event*$_{ptick}$*.sel*(*1*) *event*$_{ptick}$*.simps*(*9*) *rangeI vimage-eq*,
*metis IntI UNIV-I event*$_{ptick}$*.sel*(*1*) *image-eqI*)
**qed**
**with** ∗(*2*) **have** ‹(*t1*, *ref-Seq*$_{ptick}$ (*map-event*$_{ptick}$ *f g*′ −‘ *X*)) ∈ $\mathcal{F}$ *P*› **by** *simp*
**hence** ‹(*map* (*ev* ∘ *of-ev*) *t1*, *map-event*$_{ptick}$ *f g*′ −‘ *X*) ∈ $\mathcal{F}$ (*P* ;$_{\checkmark}$ *Q*)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*)
(*metis* ∗(*1*) *F-P*(*4*) *map-event*$_{ptick}$*-tickFree*)
**hence** ‹(*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1*), *X*) ∈ $\mathcal{F}$ *?lhs*›
**unfolding** *F-Renaming* **by** *blast*
**also have** ‹*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1*) = *s*›
**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** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› **.**
**next**
**case** *F-Q*
**have** ‹*s1* @ [✓(*g-r*)] ∉ $\mathcal{D}$ (*Renaming P f g*)› **by** (*meson F-Q*(*3*) *is-processT9*)
**with** *F-Q*(*2*−*4*) **obtain** *t1 r*
**where** ∗ : ‹*g-r* = *g r*› ‹*r* ∈ ✓**s**(*P*)› ‹*s1* = *map* (*map-event*$_{ptick}$ *f g*) *t1*› ‹*t1* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*›
**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*) ‹*inj-on g* ✓**s**(*P*)› **have** ‹(*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*) = *r*›
**by** (*auto dest*: *inj-onD*)
**with** *F-Q*(*5*, *6*) **have** ‹(*s2*, *X*) ∈ $\mathcal{F}$ (*Renaming* (*Q r*) *f g*′)›
‹*s2* ∉ $\mathcal{D}$ (*Renaming* (*Q r*) *f g*′)› **by** *simp-all*
**then obtain** *t2* **where** ∗∗ : ‹*s2* = *map* (*map-event*$_{ptick}$ *f g*′) *t2*› ‹(*t2*, *map-event*$_{ptick}$ *f g*′ −‘ *X*) ∈ $\mathcal{F}$ (*Q r*)›
**unfolding** *Renaming-projs* **by** *blast*
**from** ∗(*4*) ∗∗(*2*) **have** ‹(*map* (*ev* ∘ *of-ev*) *t1* @ *t2*, *map-event*$_{ptick}$ *f g*′ −‘ *X*) ∈ $\mathcal{F}$ (*P* ;$_{\checkmark}$ *Q*)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*) (*metis append-T-imp-tickFree not-Cons-self2*)
**hence** ‹(*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1* @ *t2*), *X*) ∈ $\mathcal{F}$ *?lhs*›
**unfolding** *F-Renaming* **by** *blast*
**also have** ‹*map* (*map-event*$_{ptick}$ *f g*′) (*map* (*ev* ∘ *of-ev*) *t1* @ *t2*) = *s*›
**by** (*simp add*: *F-Q*(*1*) ∗(*3*) ∗∗(*1*))
(*metis* ∗(*3*) *F-Q*(*4*) *event*$_{ptick}$*.map-sel*(*1*) *in-set-conv-decomp*

*map-event*$_{ptick}$*-tickFree tickFree-Cons-iff tickFree-append-iff*)
**finally show** ‹(*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› **.**
**qed**
**qed**
**next**

**have** ‹*?rhs* = *Renaming P f g* ;$_{\checkmark}$ (λ*g-r*. ⊓*r* ∈ {*r* ∈ ✓**s**(*P*). *g-r* = *g r*}. *Renaming* (*Q r*) *f g*′)›
**proof** (*rule mono-Seq*$_{ptick}$*-eq*)
**show** ‹*Renaming P f g* = *Renaming P f g*› **..**
**next**
**fix** *g-r* **assume** ‹*g-r* ∈ ✓**s**(*Renaming P f g*)›
**then obtain** *s s1* **where** ‹*s* @ [✓(*g-r*)] = *map* (*map-event*$_{ptick}$ *f g*) *s1*› ‹*s1* ∈ $\mathcal{T}$ *P*› ‹*s1* ∉ $\mathcal{D}$ *P*›
**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*′ *r* **where** ‹*g-r* = *g r*› ‹*s1* = *s1*′ @ [✓(*r*)]›
**by** (*cases s1 rule*: *rev-cases*) (*auto simp add*: *tick-eq-map-event*$_{ptick}$*-iff*)
**with** ‹*s1* ∈ $\mathcal{T}$ *P*› ‹*s1* ∉ $\mathcal{D}$ *P*› **have** ‹*s1*′ @ [✓(*r*)] ∈ $\mathcal{T}$ *P*› ‹*s1*′ @ [✓(*r*)] ∉ $\mathcal{D}$ *P*› **by** *simp-all*
**hence** ‹*r* ∈ ✓**s**(*P*)› **unfolding** *strict-ticks-of-def* **by** *blast*
**have** ‹{*r* ∈ ✓**s**(*P*). *g-r* = *g r*} = {*r*}›
**by** (*auto simp add*: ‹*r* ∈ ✓**s**(*P*)› ‹*g-r* = *g r*› *intro*: *inj-onD*[*OF* ‹*inj-on g* ✓**s**(*P*)›])
**moreover have** ‹(*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*) = *r*›
**using** *calculation* **by** *blast*
**ultimately have** ‹*Q* (*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*) = *GlobalNdet* {*r* ∈ ✓**s**(*P*). *g-r* = *g r*} *Q*› **by** *simp*
**thus** ‹*Renaming* (*Q* (*THE r*. *r* ∈ ✓**s**(*P*) ∧ *g-r* = *g r*)) *f g*′ = (⊓*r* ∈ {*r* ∈ ✓**s**(*P*). *g-r* = *g r*}. *Renaming* (*Q r*) *f g*′)›
**by** (*simp flip*: *Renaming-distrib-GlobalNdet*)
**qed**
**thus** ‹*?rhs* ⊑$_{FD}$ *?lhs*› **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** ‹*Renaming* (*P* ;$_{\checkmark}$ *Q*) *f g* = *Renaming P f g* ;$_{\checkmark}$ (λ*r*. *Renaming* (*Q* ()) *f g*)›
**by** (*subst inj-on-Renaming-Seq*$_{ptick}$[**where** *g* = *g*]) (*auto intro*: *inj-onI*)

**lemma** *TickSwap-Seq*$_{ptick}$ [*simp*] :
‹*TickSwap* (*P* ;$_{\checkmark}$ *Q*) = *TickSwap P* ;$_{\checkmark}$ (λ(*s*, *r*). *TickSwap* (*Q* (*r*, *s*)))› (**is** ‹*?lhs* = *?rhs*›)
**proof** −
**have** ‹*?lhs* = *Renaming* (*P* ;$_{\checkmark}$ *Q*) *id prod.swap*› **by** (*simp add*: *TickSwap-is-Renaming*)
**also have** ‹... = *Renaming P id prod.swap* ;$_{\checkmark}$

(λs-r. Renaming (Q (THE r-s. r-s ∈ strict-ticks-of P ∧
s-r = prod.swap r-s)) id prod.swap)›
(is ‹- = - ;✓ ?rhs'›) by (simp add: inj-on-Renaming-Seq$_{ptick}$)
also have ‹... = ?rhs›
proof (rule mono-Seq$_{ptick}$-eq, unfold TickSwap-is-Renaming)
show ‹Renaming P id prod.swap = Renaming P id prod.swap› ..
next
fix s-r assume ‹s-r ∈ strict-ticks-of (Renaming P id prod.swap)›
then obtain r s where ‹(r, s) ∈ strict-ticks-of P› ‹s-r = (s, r)›
by (auto simp flip: TickSwap-is-Renaming)
hence ‹(THE r-s. r-s ∈ strict-ticks-of P ∧ s-r = prod.swap r-s) = (r, s)› by auto
thus ‹Renaming (Q (THE r-s. r-s ∈ strict-ticks-of P ∧ s-r = prod.swap r-s)) id prod.swap =
(case s-r of (s, r) ⇒ Renaming (Q (r, s)) id prod.swap)›
by (simp add: ‹s-r = (s, r)›)
qed
finally show ‹?lhs = ?rhs› .
qed

lemma TickSwap-is-Seq$_{ptick}$-iff [simp] :
‹TickSwap P = Q ;✓ R ⟷ P = TickSwap Q ;✓ (λ(r, s). TickSwap (R (s, r)))›
by (simp add: TickSwap-eq-iff-eq-TickSwap)

### 16.1.2 Renaming and Synchronization Product

theorem (in Sync$_{ptick}$-locale) inj-RenamingEv-Sync$_{ptick}$ :
‹RenamingEv (P ⟦S⟧✓ Q) f = RenamingEv P f ⟦f ‘ S⟧✓ RenamingEv Q f›
(is ‹?lhs = ?rhs›) if ‹inj f›
proof −
let ?fun = ‹map-event$_{ptick}$ f id›
let ?map = ‹map ?fun›
let ?R = ‹λP. RenamingEv P f›
show ‹?lhs = ?rhs›
proof (rule Process-eq-optimizedI)
fix t assume ‹t ∈ 𝒟 ?lhs›
then obtain t1 t2 where ∗ : ‹t = ?map t1 @ t2›
‹tF t1› ‹ftF t2› ‹t1 ∈ 𝒟 (P ⟦S⟧✓ Q)› unfolding D-Renaming by blast
from ∗(4) obtain u v t-P t-Q where ∗∗ : ‹t1 = u @ v› ‹tF u› ‹ftF v›
‹u setinterleaves✓(⊗✓) ((t-P, t-Q), S)›
‹t-P ∈ 𝒟 P ∧ t-Q ∈ 𝒯 Q ∨ t-P ∈ 𝒯 P ∧ t-Q ∈ 𝒟 Q›
unfolding D-Sync$_{ptick}$ by blast
from setinterleaves$_{ptick}$-inj-map-map-event$_{ptick}$-iff-weak [THEN iffD2, OF ‹inj f› ∗∗(4)]
have ‹?map u setinterleaves✓(⊗✓) ((?map t-P, ?map t-Q), f ‘ S)› .
moreover from ∗∗(5) have ‹?map t-P ∈ 𝒟 (?R P) ∧ ?map t-Q ∈ 𝒯 (?R Q)
∨
?map t-P ∈ 𝒯 (?R P) ∧ ?map t-Q ∈ 𝒟 (?R Q)›
by (auto simp add: Renaming-projs dest: D-T)

```
    (metis **(2,4) append-self-conv front-tickFree-map-tick-iff
      list.map-disc-iff tickFree-setinterleaves_ptick-iff)+
  moreover have ‹t = ?map u @ (?map v @ t2)› by (simp add: *(1) **(1))
  moreover have ‹tF (?map u)› by (simp add: **(2) map-event_ptick-tickFree)
  moreover from *(2,3) **(1) have ‹ftF (?map v @ t2)›
    using front-tickFree-append map-event_ptick-tickFree tickFree-append-iff by
blast
  ultimately show ‹t ∈ 𝒟 ?rhs› unfolding D-Sync_ptick by blast
 next
  fix t assume ‹t ∈ 𝒟 ?rhs›
  then obtain u v t-P t-Q where * : ‹t = u @ v› ‹tF u› ‹ftF v›
   ‹u setinterleaves✓(⊗✓) ((t-P, t-Q), f ‘ S)›
   ‹t-P ∈ 𝒟 (?R P) ∧ t-Q ∈ 𝒯 (?R Q) ∨ t-P ∈ 𝒯 (?R P) ∧ t-Q ∈ 𝒟 (?R Q)›
   unfolding D-Sync_ptick by blast
  from *(5) show ‹t ∈ 𝒟 ?lhs›
  proof (elim disjE conjE)
   assume ‹t-P ∈ 𝒟 (?R P)› ‹t-Q ∈ 𝒯 (?R Q)›
   from ‹t-P ∈ 𝒟 (?R P)› obtain t-P1 t-P2
     where ** : ‹t-P = ?map t-P1 @ t-P2› ‹tF t-P1› ‹ftF t-P2› ‹t-P1 ∈ 𝒟 P›
     unfolding D-Renaming by blast
    from ‹t-Q ∈ 𝒯 (?R Q)› consider (T-Q) t-Q1 where ‹t-Q = ?map t-Q1›
‹t-Q1 ∈ 𝒯 Q›
     | (D-Q) t-Q1 t-Q2 where ‹t-Q = ?map t-Q1 @ t-Q2› ‹tF t-Q1› ‹ftF t-Q2›
‹t-Q1 ∈ 𝒟 Q›
     unfolding T-Renaming by blast
   thus ‹t ∈ 𝒟 ?lhs›
   proof cases
    case T-Q
    from *(4)[unfolded **(1) T-Q(1), THEN setinterleaves_ptick-appendL]
    obtain u1 u2 t-Q11 t-Q12 where *** : ‹u = u1 @ u2› ‹?map t-Q1 = t-Q11
@ t-Q12›
      ‹u1 setinterleaves✓(⊗✓) ((?map t-P1, t-Q11), f ‘ S)› by blast
     obtain t-Q11' where ‹t-Q11' ≤ t-Q1› ‹t-Q11 = ?map t-Q11'›
      by (metis ***(2) map-eq-append-conv Prefix-Order.prefixI)
     from setinterleaves_ptick-inj-map-map-event_ptick-iff-strong
      [THEN iffD1, OF ‹inj f› ***(3)[unfolded this]]
     obtain u1' where **** : ‹u1 = ?map u1'›
      ‹u1' setinterleaves✓(⊗✓) ((t-P1, t-Q11'), S)› by blast
     from *(2) ***(1) ****(1) map-event_ptick-tickFree
      T-Q(2) ‹t-Q11' ≤ t-Q1› is-processT3-TR
     have ‹u1' = u1' @ []› ‹tF u1'› ‹ftF []› ‹t-Q11' ∈ 𝒯 Q› by simp-all blast
     with ****(2) **(4) have ‹u1' ∈ 𝒟 (P ⟦S⟧✓ Q)›
      unfolding D-Sync_ptick by blast
     moreover have ‹t = ?map u1' @ (u2 @ v)› by (simp add: *(1) ***(1)
****(1))
     moreover have ‹ftF (u2 @ v)›
      using *(2,3) ***(1) front-tickFree-append tickFree-append-iff by blast
     ultimately show ‹t ∈ 𝒟 ?lhs› unfolding D-Renaming using ‹tF u1'› by
blast
```

**next**
**case** *D-Q*
**have** ‹*?map t-P1* ≤ *t-P*› ‹*?map t-Q1* ≤ *t-Q*›
**by** (*simp-all add*: ∗∗(*1*) *D-Q*(*1*))
**from** *setinterleaves*$_{ptick}$*-le-prefixLR*[*OF* ∗(*4*) *this*] **show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**proof** (*elim disjE exE conjE*)
**fix** *u1 t-Q1′* **assume** ∗∗∗ : ‹*u1* ≤ *u*› ‹*t-Q1′* ≤ *?map t-Q1*›
‹*u1 setinterleaves*$_{✓(⊗✓)}$ ((*?map t-P1*, *t-Q1′*), *f* ‘ *S*)›
**obtain** *u2* **where** ‹*u* = *u1* @ *u2*› **using** ∗∗∗(*1*) *prefixE* **by** *blast*
**obtain** *t-Q1″* **where** ‹*t-Q1′* = *?map t-Q1″*› ‹*t-Q1″* ≤ *t-Q1*›
**by** (*metis* ∗∗∗(*2*) *prefixE prefixI map-eq-append-conv*)
**from** *setinterleaves*$_{ptick}$*-inj-map-map-event*$_{ptick}$*-iff-strong*
[*THEN iffD1*, *OF* ‹*inj f*› ∗∗∗(*3*)[*unfolded* ‹*t-Q1′* = *?map t-Q1″*›]]
**obtain** *u1′* **where** ∗∗∗∗ : ‹*u1* = *?map u1′*›
‹*u1′ setinterleaves*$_{✓(⊗✓)}$ ((*t-P1*, *t-Q1″*), *S*)› **by** *blast*
**have** ‹*u1′* = *u1′* @ []› ‹*ftF* []› **by** *simp-all*
**moreover from** ∗∗(*2*) ∗∗∗∗(*2*) *setinterleaves*$_{ptick}$*-tickFree-imp*
**have** ‹*tF u1′*› **by** *blast*
**moreover from** *D-Q*(*4*) *D-T* ‹*t-Q1″* ≤ *t-Q1*› *is-processT3-TR*
**have** ‹*t-Q1″* ∈ $\mathcal{T}$ *Q*› **by** *blast*
**ultimately have** ‹*u1′* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓}$ *Q*)›
**unfolding** *D-Sync*$_{ptick}$ **using** ‹*t-P1* ∈ $\mathcal{D}$ *P*› ∗∗∗∗(*2*) **by** *blast*
**moreover from** ∗(*1*−*3*) ∗∗∗∗(*1*)
**have** ‹*t* = *?map u1′* @ (*u2* @ *v*)› ‹*ftF* (*u2* @ *v*)›
**by** (*auto simp add*: ‹*u* = *u1* @ *u2*› *front-tickFree-append*)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming* **using** ‹*tF u1′*› **by** *blast*
**next**
**fix** *u1 t-P1′* **assume** ∗∗∗ : ‹*u1* ≤ *u*› ‹*t-P1′* ≤ *?map t-P1*›
‹*u1 setinterleaves*$_{✓(⊗✓)}$ ((*t-P1′*, *?map t-Q1*), *f* ‘ *S*)›
**obtain** *u2* **where** ‹*u* = *u1* @ *u2*› **using** ∗∗∗(*1*) *prefixE* **by** *blast*
**obtain** *t-P1″* **where** ‹*t-P1′* = *?map t-P1″*› ‹*t-P1″* ≤ *t-P1*›
**by** (*metis* ∗∗∗(*2*) *prefixE prefixI map-eq-append-conv*)
**from** *setinterleaves*$_{ptick}$*-inj-map-map-event*$_{ptick}$*-iff-strong*
[*THEN iffD1*, *OF* ‹*inj f*› ∗∗∗(*3*)[*unfolded* ‹*t-P1′* = *?map t-P1″*›]]
**obtain** *u1′* **where** ∗∗∗∗ : ‹*u1* = *?map u1′*›
‹*u1′ setinterleaves*$_{✓(⊗✓)}$ ((*t-P1″*, *t-Q1*), *S*)› **by** *blast*
**have** ‹*u1′* = *u1′* @ []› ‹*ftF* []› **by** *simp-all*
**moreover from** *D-Q*(*2*) ∗∗∗∗(*2*) *setinterleaves*$_{ptick}$*-tickFree-imp*
**have** ‹*tF u1′*› **by** *blast*
**moreover from** ∗∗(*4*) *D-T* ‹*t-P1″* ≤ *t-P1*› *is-processT3-TR*
**have** ‹*t-P1″* ∈ $\mathcal{T}$ *P*› **by** *blast*
**ultimately have** ‹*u1′* ∈ $\mathcal{D}$ (*P* ⟦*S*⟧$_{✓}$ *Q*)›
**unfolding** *D-Sync*$_{ptick}$ **using** ‹*t-Q1* ∈ $\mathcal{D}$ *Q*› ∗∗∗∗(*2*) **by** *blast*
**moreover from** ∗(*1*−*3*) ∗∗∗∗(*1*)
**have** ‹*t* = *?map u1′* @ (*u2* @ *v*)› ‹*ftF* (*u2* @ *v*)›
**by** (*auto simp add*: ‹*u* = *u1* @ *u2*› *front-tickFree-append*)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming* **using** ‹*tF u1′*› **by** *blast*

```
      qed
    qed
  next
    assume ‹t-Q ∈ 𝒟 (?R Q)› ‹t-P ∈ 𝒯 (?R P)›
    from ‹t-Q ∈ 𝒟 (?R Q)› obtain t-Q1 t-Q2
      where ** : ‹t-Q = ?map t-Q1 @ t-Q2› ‹tF t-Q1› ‹ftF t-Q2› ‹t-Q1 ∈ 𝒟 Q›
      unfolding D-Renaming by blast
     from ‹t-P ∈ 𝒯 (?R P)› consider (T-P) t-P1 where ‹t-P = ?map t-P1›
‹t-P1 ∈ 𝒯 P›
      | (D-P) t-P1 t-P2 where ‹t-P = ?map t-P1 @ t-P2› ‹tF t-P1› ‹ftF t-P2›
‹t-P1 ∈ 𝒟 P›
      unfolding T-Renaming by blast
    thus ‹t ∈ 𝒟 ?lhs›
    proof cases
      case T-P
      from *(4)[unfolded **(1) T-P(1), THEN setinterleaves_ptick-appendR]
     obtain u1 u2 t-P11 t-P12 where *** : ‹u = u1 @ u2› ‹?map t-P1 = t-P11
@ t-P12›
        ‹u1 setinterleaves✓(⊗✓) ((t-P11, ?map t-Q1), f ‘ S)› by blast
      obtain t-P11' where ‹t-P11' ≤ t-P1› ‹t-P11 = ?map t-P11'›
        by (metis ***(2) map-eq-append-conv Prefix-Order.prefixI)
      from setinterleaves_ptick-inj-map-map-event_ptick-iff-strong
        [THEN iffD1, OF ‹inj f› ***(3)[unfolded this]]
      obtain u1' where **** : ‹u1 = ?map u1'›
        ‹u1' setinterleaves✓(⊗✓) ((t-P11', t-Q1), S)› by blast
      from *(2) ***(1) ****(1) map-event_ptick-tickFree
        T-P(2) ‹t-P11' ≤ t-P1› is-processT3-TR
      have ‹u1' = u1' @ []› ‹tF u1'› ‹ftF []› ‹t-P11' ∈ 𝒯 P› by simp-all blast
      with ****(2) **(4) have ‹u1' ∈ 𝒟 (P ⟦S⟧✓ Q)›
        unfolding D-Sync_ptick by blast
       moreover have ‹t = ?map u1' @ (u2 @ v)› by (simp add: *(1) ***(1)
****(1))
      moreover have ‹ftF (u2 @ v)›
        using *(2,3) ***(1) front-tickFree-append tickFree-append-iff by blast
      ultimately show ‹t ∈ 𝒟 ?lhs› unfolding D-Renaming using ‹tF u1'› by
blast
    next
      case D-P
      have ‹?map t-P1 ≤ t-P› ‹?map t-Q1 ≤ t-Q›
        by (simp-all add: **(1) D-P(1))
      from setinterleaves_ptick-le-prefixLR[OF *(4) this] show ‹t ∈ 𝒟 ?lhs›
      proof (elim disjE exE conjE)
        fix u1 t-Q1' assume *** : ‹u1 ≤ u› ‹t-Q1' ≤ ?map t-Q1›
          ‹u1 setinterleaves✓(⊗✓) ((?map t-P1, t-Q1'), f ‘ S)›
        obtain u2 where ‹u = u1 @ u2› using ***(1) prefixE by blast
        obtain t-Q1'' where ‹t-Q1' = ?map t-Q1''› ‹t-Q1'' ≤ t-Q1›
          by (metis ***(2) prefixE prefixI map-eq-append-conv)
        from setinterleaves_ptick-inj-map-map-event_ptick-iff-strong
          [THEN iffD1, OF ‹inj f› ***(3)[unfolded ‹t-Q1' = ?map t-Q1''›]]
```

**obtain** *u1'* **where** $****$ : ‹*u1* = *?map u1'*›
‹*u1'* *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P1*, *t-Q1''*), *S*)› **by** *blast*
**have** ‹*u1'* = *u1'* @ []› ‹*ftF* []› **by** *simp-all*
**moreover from** *D-P*(*2*) $****$(*2*) *setinterleaves*$_{ptick}$*-tickFree-imp*
**have** ‹*tF u1'*› **by** *blast*
**moreover from** $**$(*4*) *D-T* ‹*t-Q1''* ≤ *t-Q1*› *is-processT3-TR*
**have** ‹*t-Q1''* ∈ $\mathcal{T}$ *Q*› **by** *blast*
**ultimately have** ‹*u1'* ∈ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**unfolding** *D-Sync*$_{ptick}$ **using** ‹*t-P1* ∈ $\mathcal{D}$ *P*› $****$(*2*) **by** *blast*
**moreover from** $*$(*1*−*3*) $****$(*1*)
**have** ‹*t* = *?map u1'* @ (*u2* @ *v*)› ‹*ftF* (*u2* @ *v*)›
**by** (*auto simp add*: ‹*u* = *u1* @ *u2*› *front-tickFree-append*)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming* **using** ‹*tF u1'*› **by** *blast*
**next**
**fix** *u1 t-P1'* **assume** $***$ : ‹*u1* ≤ *u*› ‹*t-P1'* ≤ *?map t-P1*›
‹*u1* *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P1'*, *?map t-Q1*), *f* ‘ *S*)›
**obtain** *u2* **where** ‹*u* = *u1* @ *u2*› **using** $***$(*1*) *prefixE* **by** *blast*
**obtain** *t-P1''* **where** ‹*t-P1'* = *?map t-P1''*› ‹*t-P1''* ≤ *t-P1*›
**by** (*metis* $***$(*2*) *prefixE prefixI map-eq-append-conv*)
**from** *setinterleaves*$_{ptick}$*-inj-map-map-event*$_{ptick}$*-iff-strong*
[*THEN iffD1*, *OF* ‹*inj f*› $***$(*3*)[*unfolded* ‹*t-P1'* = *?map t-P1''*›]]
**obtain** *u1'* **where** $****$ : ‹*u1* = *?map u1'*›
‹*u1'* *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P1''*, *t-Q1*), *S*)› **by** *blast*
**have** ‹*u1'* = *u1'* @ []› ‹*ftF* []› **by** *simp-all*
**moreover from** $**$(*2*) $****$(*2*) *setinterleaves*$_{ptick}$*-tickFree-imp*
**have** ‹*tF u1'*› **by** *blast*
**moreover from** *D-P*(*4*) *D-T* ‹*t-P1''* ≤ *t-P1*› *is-processT3-TR*
**have** ‹*t-P1''* ∈ $\mathcal{T}$ *P*› **by** *blast*
**ultimately have** ‹*u1'* ∈ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**unfolding** *D-Sync*$_{ptick}$ **using** ‹*t-Q1* ∈ $\mathcal{D}$ *Q*› $****$(*2*) **by** *blast*
**moreover from** $*$(*1*−*3*) $****$(*1*)
**have** ‹*t* = *?map u1'* @ (*u2* @ *v*)› ‹*ftF* (*u2* @ *v*)›
**by** (*auto simp add*: ‹*u* = *u1* @ *u2*› *front-tickFree-append*)
**ultimately show** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**unfolding** *D-Renaming* **using** ‹*tF u1'*› **by** *blast*
**qed**
**qed**
**qed**
**next**
**fix** *t X* **assume** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› ‹*t* ∉ $\mathcal{D}$ *?lhs*›
**then obtain** *t'* **where** ‹*t* = *?map t'*›
**and** $*$ : ‹(*t'*, *?fun* −‘ *X*) ∈ $\mathcal{F}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**unfolding** *Renaming-projs* **by** *blast*
**have** ‹*t'* ∉ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**proof** (*rule notI*)
**assume** ‹*t'* ∈ $\mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)›
**hence** ‹*t* ∈ $\mathcal{D}$ *?lhs*›
**by** (*simp add*: ‹*t* = *?map t'*› *D-Sync*$_{ptick}$ *D-Renaming*)

(*metis* (*no-types*) *append-Nil2 front-tickFree-Nil map-append map-event*$_{ptick}$*-front-tickFree*)
**with** ‹$t \notin \mathcal{D}$ *?lhs*› **show** *False* **..**
**qed**
**with** $*$ **obtain** *t-P t-Q X-P X-Q*
**where** $**$ : ‹(*t-P*, *X-P*) $\in \mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) $\in \mathcal{F}$ *Q*›
‹$t'$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), *S*)›
‹*?fun* −‘ *X* ⊆ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark$) *X-P S X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *fast*
**from** $**$(*2*, *3*) *F-T* ‹$t' \notin \mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)› *append-Nil2 front-tickFree-Nil*
**have** ‹*t-P* $\notin \mathcal{D}$ *P*› **unfolding** *D-Sync*$_{ptick}{}'$ **by** *blast*
**from** $**$(*1*, *3*) *F-T* ‹$t' \notin \mathcal{D}$ (*P* $[\![S]\!]_{\checkmark}$ *Q*)› *append-Nil2 front-tickFree-Nil*
**have** ‹*t-Q* $\notin \mathcal{D}$ *Q*› **unfolding** *D-Sync*$_{ptick}{}'$ **by** *blast*
**have** $***$ : ‹*?fun* −‘ *?fun* ‘ *X-P* = *X-P*› ‹*?fun* −‘ *?fun* ‘ *X-Q* = *X-Q*›
**by** (*simp add*: *set-eq-iff image-iff*,
*metis* (*mono-tags*, *opaque-lifting*) *event*$_{ptick}$*.inj-map-strong id-apply injD* ‹*inj f*›)+
**from** $**$(*1*) **have** ‹(*?map t-P*, *?fun* ‘ *X-P*) $\in \mathcal{F}$ (*?R P*)›
**by** (*subst* (*asm*) $***$(*1*)[*symmetric*]) (*auto simp add*: *F-Renaming*)
**moreover** **{**
**fix** *a* **assume** ‹*?map t-P* @ [*ev a*] $\in \mathcal{T}$ (*?R P*)› ‹*a* $\notin$ *range f*›
**then consider** *t-P1* **where** ‹*?map t-P* @ [*ev a*] = *?map t-P1*› ‹*t-P1* $\in \mathcal{T}$ *P*›
| *t-P1 t-P2* **where** ‹*?map t-P* @ [*ev a*] = *?map t-P1* @ *t-P2*› ‹*tF t-P1*›
‹*t-P1* $\in \mathcal{D}$ *P*›
**unfolding** *T-Renaming* **by** *blast*
**hence** *False*
**proof** *cases*
**from** ‹*a* $\notin$ *range f*› **show** ‹*?map t-P* @ [*ev a*] = *?map t-P1* $\Longrightarrow$ *False*› **for** *t-P1*
**by** (*auto simp add*: *append-eq-map-conv ev-eq-map-event*$_{ptick}$*-iff*)
**next**
**fix** *t-P1 t-P2* **assume** ‹*?map t-P* @ [*ev a*] = *?map t-P1* @ *t-P2*› ‹*tF t-P1*›
‹*t-P1* $\in \mathcal{D}$ *P*›
**from** *this*(*1*) ‹*a* $\notin$ *range f*› **have** ‹*t-P1* $\leq$ *t-P*›
**by** (*cases t-P2 rule*: *rev-cases*, *auto simp add*: *append-eq-map-conv ev-eq-map-event*$_{ptick}$*-iff*)
(*metis prefixI event*$_{ptick}$*.inj-map inj-map-eq-map inj-on-id map-eq-append-conv* ‹*inj f*›)
**with** ‹*t-P1* $\in \mathcal{D}$ *P*› **have** ‹*t-P* $\in \mathcal{D}$ *P*›
**by** (*metis* $**$(*1*) *F-imp-front-tickFree prefixE* ‹*tF t-P1*› *front-tickFree-append-iff*
*is-processT7 tickFree-Nil tickFree-imp-front-tickFree*)
**with** ‹*t-P* $\notin \mathcal{D}$ *P*› **show** *False* **..**
**qed**
**}**
**ultimately have** $ : ‹(*?map t-P*, *?fun* ‘ *X-P* ∪ {*ev a* | *a*. *a* $\notin$ *range f*}) $\in \mathcal{F}$ (*?R P*)›
**using** *is-processT5-S7'* **by** *blast*
**from** $**$(*2*) **have** ‹(*?map t-Q*, *?fun* ‘ *X-Q*) $\in \mathcal{F}$ (*?R Q*)›
**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-event*$_{ptick}$*-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-event*$_{ptick}$*-iff*)
(*metis prefixI event*$_{ptick}$*.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′*› *setinterleaves*$_{ptick}$*-inj-map-map-event*$_{ptick}$*-iff-weak* ‹*inj f*›)
**have** ‹*e ∈ X ⟹ e ∈ super-ref-Sync*$_{ptick}$ *(⊗✓) (?fun ‘ X-P ∪ {ev a | a. a ∉ range f}) (f ‘ S)*
*(?fun ‘ X-Q ∪ {ev a | a. a ∉ range f})*› **for** *e*
**using** ∗∗(*4*)[*THEN set-mp*, *of* ‹*map-event*$_{ptick}$ *(inv f) id e*›] ‹*inj f*›
**unfolding** *super-ref-Sync*$_{ptick}$*-def*
**by** (*cases e*, *simp-all add*: *image-iff tick-eq-map-event*$_{ptick}$*-iff*) *force*
**hence** $$$$ : ‹*X ⊆ super-ref-Sync*$_{ptick}$ *(⊗✓) (?fun ‘ X-P ∪ {ev a | a. a ∉ range f}) (f ‘ S)*
*(?fun ‘ X-Q ∪ {ev a | a. a ∉ range f})*› **by** *blast*
**from** $ $$ $$$ $$$$ **show** ‹(*t*, *X*) ∈ ℱ *?rhs*› **unfolding** *F-Sync*$_{ptick}$ **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-Sync*$_{ptick}$ *(⊗✓) X-P (f ‘ S) X-Q*›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**have** ‹¬ (*t-P ∈ 𝒟 (?R P) ∨ t-Q ∈ 𝒟 (?R Q)*)›

**proof** (*rule notI*)
**assume** ‹*t-P* ∈ $\mathcal{D}$ (*?R P*) ∨ *t-Q* ∈ $\mathcal{D}$ (*?R Q*)›
**hence** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
**by** (*simp add*: *D-Sync*$_{ptick}$′)
(*metis* ∗(*1*−*3*) *F-T append-Nil2 front-tickFree-Nil*)
**with** ‹*t* ∉ $\mathcal{D}$ *?rhs*› **show** *False* **..**
**qed**
**with** ∗(*1*, *2*) **obtain** *t-P*′ *t-Q*′
**where** ∗∗ : ‹*t-P* = *?map t-P*′› ‹(*t-P*′, *?fun* −‘ *X-P*) ∈ $\mathcal{F}$ *P*›
‹*t-Q* = *?map t-Q*′› ‹(*t-Q*′, *?fun* −‘ *X-Q*) ∈ $\mathcal{F}$ *Q*›
**unfolding** *Renaming-projs* **by** *blast*
**from** *setinterleaves*$_{ptick}$-*inj-map-map-event*$_{ptick}$-*iff-strong*
[*THEN iffD1*, *OF* ‹*inj f*› ∗(*3*)[*unfolded* ∗∗(*1*, *3*)]] **obtain** *t*′
**where** ∗∗∗ : ‹*t* = *?map t*′› ‹*t*′ *setinterleaves*$_{✓(⊗✓)}$ ((*t-P*′, *t-Q*′), *S*)› **by** *blast*
**have** ‹*e* ∈ *?fun* −‘ *X* ⟹ *e* ∈ *super-ref-Sync*$_{ptick}$ (⊗✓) (*?fun* −‘ *X-P*) *S* (*?fun* −‘ *X-Q*)› **for** *e*
**using** ∗(*4*)[*THEN set-mp*, *of* ‹*?fun e*›]
**by** (*cases e*) (*auto simp add*: *super-ref-Sync*$_{ptick}$-*def dest*: *injD*[*OF* ‹*inj f*›])
**hence** ‹*?fun* −‘ *X* ⊆ *super-ref-Sync*$_{ptick}$ (⊗✓) (*?fun* −‘ *X-P*) *S* (*?fun* −‘ *X-Q*)› **by** *blast*
**with** ∗∗(*2*, *4*) ∗∗∗(*2*) **have** ‹(*t*′, *?fun* −‘ *X*) ∈ $\mathcal{F}$ (*P* ⟦*S*⟧$_{✓}$ *Q*)›
**unfolding** *F-Sync*$_{ptick}$ **by** *auto*
**thus** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** (*auto simp add*: ∗∗∗(*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 $\mathbb{F}_{✓}(P)$ is not satisfied.

**notepad begin**
**define** *Q* :: ‹*nat* ⇒ (′*a*, ′*r*) *process*$_{ptick}$›
**where** ‹*Q r* ≡ (((→) *undefined*) ⌢⌢ *r*) *STOP*› **for** *r*
**have** ‹*SKIPS UNIV* \ {*undefined*} = (*SKIPS UNIV* :: (′*a*, *nat*) *process*$_{ptick}$)›
**by** (*simp add*: *Hiding-SKIPS*)
**moreover have** ‹*Q r* \ {*undefined*} = *STOP*› **for** *r*
**by** (*induct r*) (*simp-all add*: *Q-def Hiding-write0-non-disjoint*)
**ultimately have** ∗ : ‹(*SKIPS UNIV* \ {*undefined*}) ;$_{✓}$ (λ*r*. *Q r* \ {*undefined*}) = *STOP*›
**by** (*simp only*: *SKIPS-Seq*$_{ptick}$) *simp*

**have** ‹*SKIPS UNIV* ;$_{✓}$ *Q* = ⊓*r* ∈ *UNIV*. *Q r*› **by** *simp*
**moreover have** ‹[] ∈ $\mathcal{D}$ (... \ {*undefined*})›
**proof** (*rule D-Hiding-seqRunI*)
**show** ‹*ftF* []› ‹*tF* []› ‹[] = *trace-hide* [] (*ev* ‘ {*undefined*}) @ []› **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..<r]›
        by (simp add: map-replicate-trivial)
      finally have ‹map (λi. ev undefined) [0..<r] ∈ 𝒯 (Q r)› .
    }
    hence ‹∃ r. 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) process_ptick) 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: Seq_ptick-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: Seq_ptick-projs intro: front-tickFree-Nil)
      with mem-D-imp-mem-D-Hiding show ‹?th (?map u') ∈ 𝒟 ?lhs› .
    next
```

**assume** ‹*isInfHidden-seqRun x P S u′*›
**from** *this isInfHidden-seqRun-imp-tickFree-seqRun*[*OF this*]
**have** ‹*isInfHidden-seqRun* (*ev* ∘ *of-ev* ∘ *x*) (*P* ;✓ *Q*) *S* (*?map u′*)›
**by** (*simp add*: $Seq_{ptick}$*-projs image-iff*)
(*metis* $event_{ptick}$*.sel*(*1*) *list.map-comp map-append seqRun-def*)
**thus** ‹*?th* (*?map u′*) ∈ 𝒟 *?lhs*›
**by** (*simp add*: *D-Hiding-seqRun*)
(*metis* (*no-types*) *append.right-neutral comp-apply*
*front-tickFree-Nil tickFree-map-ev-comp*)
**qed**
**also have** ‹*?th* (*?map u′*) = *?map* (*?th u′*)›
**by** (*fact tickFree-trace-hide-map-ev-comp-of-ev*[*OF* ‹*tF u′*›])
**finally show** ‹*t* ∈ 𝒟 *?lhs*›
**by** (*simp add*: *D-P*(*1*, *4*) ∗(*1*) *front-tickFree-append is-processT7*)
**next**
**case** *D-Q*
**from** *D-Q*(*2*, *3*) **obtain** *u′* **where** ‹*u* @ [✓(*r*)] = *?th u′*› ‹(*u′*, *ev* ‘ *S*) ∈ ℱ *P*›
**unfolding** *T-Hiding D-Hiding* **by** *fast*
**then obtain** *u′* **where** ‹*u* = *?th u′*› ‹(*u′* @ [✓(*r*)], *ev* ‘ *S*) ∈ ℱ *P*›
**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** *D-Q*(*5*) **obtain** *v′ w′ x* **where** ∗ : ‹*v* = *?th v′* @ *w′*› ‹*tF v′*› ‹*ftF w′*›
‹*v′* ∈ 𝒟 (*Q r*) ∨ *isInfHidden-seqRun x* (*Q r*) *S v′*›
**by** (*blast elim*: *D-Hiding-seqRunE*)
**from** ∗(*4*) **have** ‹*?th* (*?map u′* @ *v′*) ∈ 𝒟 *?lhs*›
**proof** (*elim disjE*)
**assume** ‹*v′* ∈ 𝒟 (*Q r*)›
**hence** ‹*?map u′* @ *v′* ∈ 𝒟 (*P* ;✓ *Q*)›
**by** (*simp add*: $Seq_{ptick}$*-projs*)
(*metis F-T* ‹(*u′* @ [✓(*r*)], *ev* ‘ *S*) ∈ ℱ *P*› *append-T-imp-tickFree*
*not-Cons-self*)
**with** *mem-D-imp-mem-D-Hiding* **show** ‹*?th* (*?map u′* @ *v′*) ∈ 𝒟 *?lhs*› **.**
**next**
**assume** ‹*isInfHidden-seqRun x* (*Q r*) *S v′*›
**hence** ‹*isInfHidden-seqRun x* (*P* ;✓ *Q*) *S* (*?map u′* @ *v′*)›
**by** (*simp add*: *seqRun-def image-iff* $Seq_{ptick}$*-projs*)
(*metis F-T* ‹(*u′* @ [✓(*r*)], *ev* ‘ *S*) ∈ ℱ *P*› *append-T-imp-tickFree list.discI*)
**thus** ‹*?th* (*?map u′* @ *v′*) ∈ 𝒟 *?lhs*›
**by** (*simp add*: *D-Hiding-seqRun*)
(*metis append.right-neutral filter-append*
*front-tickFree-Nil isInfHidden-seqRun-imp-tickFree*)
**qed**
**also have** ‹*?th* (*?map u′* @ *v′*) = *?map* (*?th u′*) @ *?th v′*›
**using** *D-Q*(*4*) *Hiding-tickFree* ‹*u* = *?th u′*› *tickFree-trace-hide-map-ev-comp-of-ev*
**by** *auto*
**finally have** ‹*?map* (*?th u′*) @ *?th v′* ∈ 𝒟 *?lhs*› **.**
**moreover have** ‹*tF* (*?map* (*?th u′*) @ *?th v′*)›
**by** (*simp add*: ∗(*2*) *Hiding-tickFree*)

**ultimately show** ‹$t \in \mathcal{D}$ *?lhs*›
**unfolding** $*(1)$ *D-Q*$(1)$ ‹$u =$ *?th* $u'$› **using** ‹*ftF* $w'$›
**by** (*metis append.assoc is-processT7*)
**qed**
**next**

**assume** *subset-div* : ‹$\mathcal{D}$ *?rhs* $\subseteq \mathcal{D}$ *?lhs*›
**let** *?th* = ‹$\lambda t.$ *trace-hide* $t$ (*ev* ‘ $S$)› **and** *?map* = ‹$\lambda t.$ *map* (*ev* $\circ$ *of-ev*) $t$›
**fix** $t$ $X$ **assume** ‹$(t, X) \in \mathcal{F}$ *?rhs*›
**then consider** (*div*) ‹$t \in \mathcal{D}$ *?rhs*›
| (*F-P*) $u$ **where** ‹$t =$ *?map* $u$› ‹($u$, *ref-Seq*$_{ptick}$ $X$) $\in \mathcal{F}$ ($P \setminus S$)› ‹$u \notin \mathcal{D}$ ($P \setminus S$)› ‹*tF* $u$›
| (*F-Q*) $u$ $r$ $v$ **where** ‹$t =$ *?map* $u$ @ $v$› ‹$u$ @ [✓($r$)] $\in \mathcal{T}$ ($P \setminus S$)› ‹$u$ @ [✓($r$)] $\notin \mathcal{D}$ ($P \setminus S$)›
‹*tF* $u$› ‹($v$, $X$) $\in \mathcal{F}$ ($Q$ $r \setminus S$)› ‹$v \notin \mathcal{D}$ ($Q$ $r \setminus S$)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*)
(*metis F-T T-imp-front-tickFree front-tickFree-Nil is-processT9 self-append-conv*)
**thus** ‹($t$, $X$) $\in \mathcal{F}$ *?lhs*›
**proof** *cases*
**case** *div* **with** *subset-div* **show** ‹($t$, $X$) $\in \mathcal{F}$ *?lhs*›
**by** (*simp add*: *in-mono is-processT8*)
**next**
**case** *F-P*
**from** *F-P*$(2, 3)$ **obtain** $u'$ **where** $*$ : ‹$u =$ *?th* $u'$› ‹($u'$, *ref-Seq*$_{ptick}$ $X \cup$ *ev* ‘ $S$) $\in \mathcal{F}$ $P$›
**unfolding** *F-Hiding D-Hiding* **by** *fast*
**have** ‹*tF* $u'$› **using** $*(1)$ *F-P*$(4)$ *Hiding-tickFree* **by** *blast*
**have** $: ‹*ref-Seq*$_{ptick}$ ($X \cup$ *ev* ‘ $S$) = *ref-Seq*$_{ptick}$ $X \cup$ *ev* ‘ $S$›
**by** (*auto simp add*: *image-iff ref-Seq*$_{ptick}$*-def*)
(*metis Int-iff Un-iff event*$_{ptick}$*.sel*$(1)$ *image-eqI rangeI*)
**from** ‹*tF* $u'$› $*(2)$ **have** ‹(*?map* $u'$, $X \cup$ *ev* ‘ $S$) $\in \mathcal{F}$ ($P$ ;✓ $Q$)›
**by** (*auto simp add*: *Seq*$_{ptick}$*-projs* $)
**thus** ‹($t$, $X$) $\in \mathcal{F}$ *?lhs*›
**by** (*simp add*: *F-Hiding*)
(*metis* $*(1)$ *F-P*$(1)$ ‹*tF* $u'$› *tickFree-trace-hide-map-ev-comp-of-ev*)
**next**
**case** *F-Q*
**from** *F-Q*$(2, 3)$ **obtain** $u'$ **where** ‹$u$ @ [✓($r$)] = *?th* $u'$› ‹($u'$, *ev* ‘ $S$) $\in \mathcal{F}$ $P$›
**unfolding** *T-Hiding D-Hiding* **by** *fast*
**then obtain** $u'$ **where** $*$ : ‹$u =$ *?th* $u'$› ‹($u'$ @ [✓($r$)], *ev* ‘ $S$) $\in \mathcal{F}$ $P$›
**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-Q*$(5, 6)$ **obtain** $v'$ **where** $**$ : ‹$v =$ *?th* $v'$› ‹($v'$, $X \cup$ *ev* ‘ $S$) $\in \mathcal{F}$ ($Q$ $r$)›
**unfolding** *F-Hiding D-Hiding* **by** *blast*
**have** ‹(*?map* $u'$ @ $v'$, $X \cup$ *ev* ‘ $S$) $\in \mathcal{F}$ ($P$ ;✓ $Q$)›
**by** (*simp add*: *Seq*$_{ptick}$*-projs*)
(*metis* $*(2)$ $**(2)$ *F-T append-T-imp-tickFree list.distinct*$(1)$)

**with** *F-Q(4)* **show** ‹$(t,\ X) \in \mathcal{F}$ *?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** *setinterleaves*$_{ptick}$*-imp-superset-ev* :
‹*t setinterleaves*$_{✓\,tick\text{-}join}$ $((u,\ v),\ A) \Longrightarrow$
$\{ev\ a\ |a.\ ev\ a \in set\ u\} \cup \{ev\ a\ |a.\ ev\ a \in set\ v\} \subseteq \{ev\ a\ |a.\ ev\ a \in set\ t\}$›
**proof** (*induct t arbitrary*: *u v*)
**case** *Nil* **thus** *?case* **by** (*auto dest*: *Nil-setinterleaves*$_{ptick}$)
**next**
**case** (*Cons e t*)
**from** *Cons.prems* **consider** (*mv-left*) *a u'* **where** ‹$e = ev\ a$› ‹$u = ev\ a \#\ u'$›
‹*t setinterleaves*$_{✓\,tick\text{-}join}$ $((u',\ v),\ A)$›
| (*mv-right*) *a v'* **where** ‹$e = ev\ a$› ‹$v = ev\ a \#\ v'$›
‹*t setinterleaves*$_{✓\,tick\text{-}join}$ $((u,\ v'),\ A)$›
| (*mv-both-ev*) *a u' v'* **where** ‹$e = ev\ a$› ‹$u = ev\ a \#\ u'$› ‹$v = ev\ a \#\ v'$›
‹*t setinterleaves*$_{✓\,tick\text{-}join}$ $((u',\ v'),\ A)$›
| (*mv-both-tick*) *r s u' v'* **where** ‹$u = ✓(r) \#\ u'$› ‹$v = ✓(s) \#\ v'$›
‹*t setinterleaves*$_{✓\,tick\text{-}join}$ $((u',\ v'),\ A)$›
**by** (*cases e*) (*auto elim*: *Cons-ev-setinterleaves*$_{ptick}$*E Cons-tick-setinterleaves*$_{ptick}$*E*)
**thus** *?case* **by** *cases* (*auto dest*!: *Cons.hyps*)
**qed**

**lemma** (**in** *Sync*$_{ptick}$*-locale*) *disjoint-isInfHidden-seqRunL-to-Sync*$_{ptick}$ :
**assumes** ‹$A \cap S = \{\}$› **and** ‹*isInfHidden-seqRun x P A t-P*›
**and** ‹$t\text{-}Q \in \mathcal{T}\ Q$› **and** ‹*t setinterleaves*$_{✓(\otimes ✓)}$ $((t\text{-}P,\ t\text{-}Q),\ S)$›
**shows** ‹*isInfHidden-seqRun* $(ev \circ of\text{-}ev \circ x)$ $(P\ [\![S]\!]_{✓}\ Q)$ *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* $(ev \circ of\text{-}ev)$ (*map x* [*0*..<*i*])› **for** *i*
**from** *assms(1, 2)* **have** ‹$\{a.\ ev\ a \in set\ (map\ x\ [0..<i])\} \cap S = \{\}$› **for** *i*
**by** (*simp add*: *disjoint-iff image-iff*) (*metis event*$_{ptick}$*.inject(1)*)
**from** *tickFree-disjoint-setinterleaves*$_{ptick}$*-append-tailL*[*OF tF-x this assms(4)*]
**have** ‹*seqRun t* $(ev \circ of\text{-}ev \circ x)$ *i setinterleaves*$_{✓(\otimes ✓)}$ $((seqRun\ t\text{-}P\ x\ i,\ t\text{-}Q),\ S)$›
**for** *i*
**by** (*simp add*: *seqRun-def*)
**moreover have** ‹*of-ev* $(x\ i) \in A$› **for** *i*
**by** (*metis assms(2) event*$_{ptick}$*.sel(1) image-iff*)
**ultimately show** ‹*isInfHidden-seqRun* $(ev \circ of\text{-}ev \circ x)$ $(P\ [\![S]\!]_{✓}\ Q)$ *A t*›
**using** *assms(2, 3)* **by** (*auto simp add*: *T-Sync*$_{ptick}$)
**qed**

**lemma** (**in** $Sync_{ptick}$-*locale*) *disjoint-isInfHidden-seqRunR-to-Sync*$_{ptick}$ :
‹⟦$A \cap S = \{\}$; *isInfHidden-seqRun x Q A t-Q*; *t-P* $\in \mathcal{T}$ *P*;
*t setinterleaves*$_{✓(\otimes✓)}$ ((*t-P*, *t-Q*), *S*)⟧ ⟹
*isInfHidden-seqRun* (*ev* ∘ *of-ev* ∘ *x*) (*P* ⟦*S*⟧$_{✓}$ *Q*) *A t*›
**by** (*fold Sync*$_{ptick}$*-sym*, *rule Sync*$_{ptick}$*-locale.disjoint-isInfHidden-seqRunL-to-Sync*$_{ptick}$)
(*use setinterleaves*$_{ptick}$*-sym* **in** ‹*blast intro*: *Sync*$_{ptick}$*-locale-sym.Sync*$_{ptick}$*-locale-axioms*›)+

**lemma** (**in** $Sync_{ptick}$-*locale*) *disjoint-Hiding-Sync*$_{ptick}$*-FD-Sync*$_{ptick}$*-Hiding-aux* :
— This lemma avoids duplication of the proof work.
**assumes** ‹$A \cap S = \{\}$› ‹*tF u*› ‹*ftF v*› ‹*t-P* $\in \mathcal{D}$ (*P* $\setminus$ *A*)› ‹*t-Q* $\in \mathcal{T}$ (*Q* $\setminus$ *A*)›
**and** ∗ : ‹*u setinterleaves*$_{✓(\otimes✓)}$ ((*t-P*, *t-Q*), *S*)›
**shows** ‹*u* @ *v* $\in \mathcal{D}$ (*P* ⟦*S*⟧$_{✓}$ *Q* $\setminus$ *A*)›
**proof** −
**let** *?th-A* = ‹$\lambda$*t*. *trace-hide t* (*ev* ‘ *A*)›
**from** ‹*t-P* $\in \mathcal{D}$ (*P* $\setminus$ *A*)› **obtain** *t-P1 t-P2*
**where** *D-P* : ‹*tF t-P1*› ‹*ftF t-P2*› ‹*t-P* = *?th-A t-P1* @ *t-P2*›
‹*t-P1* $\in \mathcal{D}$ *P* ∨ (∃ *t-P-x*. *isInfHidden-seqRun-strong t-P-x P A t-P1*)›
**by** (*blast elim*: *D-Hiding-seqRunE*)
**from** *setinterleaves*$_{ptick}$*-appendL*[*OF* ∗[*unfolded D-P*(*3*)]] **obtain** *u1 u2 t-Q1 t-Q2*
**where** ∗∗ : ‹*u* = *u1* @ *u2*› ‹*t-Q* = *t-Q1* @ *t-Q2*›
‹*u1 setinterleaves*$_{✓(\otimes✓)}$ ((*?th-A t-P1*, *t-Q1*), *S*)›
‹*u2 setinterleaves*$_{✓(\otimes✓)}$ ((*t-P2*, *t-Q2*), *S*)› **by** *blast*
**from** ‹*t-Q* $\in \mathcal{T}$ (*Q* $\setminus$ *A*)› **consider** *t-Q1'* **where** ‹*t-Q* = *?th-A t-Q1'*› ‹(*t-Q1'*, *ev* ‘ *A*) $\in \mathcal{F}$ *Q*›
| (*D-Q*) *t-Q1' t-Q2'* **where** ‹*tF t-Q1'*› ‹*ftF t-Q2'*› ‹*t-Q* = *?th-A t-Q1'* @ *t-Q2'*›
‹*t-Q1'* $\in \mathcal{D}$ *Q* ∨ (∃ *t-Q-x*. *isInfHidden-seqRun-strong t-Q-x Q A t-Q1'*)›
**by** (*elim T-Hiding-seqRunE*)
**thus** ‹*u* @ *v* $\in \mathcal{D}$ (*P* ⟦*S*⟧$_{✓}$ *Q* $\setminus$ *A*)›
**proof** *cases*
**fix** *t-Q1'* **assume** ‹*t-Q* = *?th-A t-Q1'*› ‹(*t-Q1'*, *ev* ‘ *A*) $\in \mathcal{F}$ *Q*›
**from** ‹*t-Q* = *?th-A t-Q1'*› ∗∗(*2*) **obtain** *t-Q1''*
**where** ‹*t-Q1* = *?th-A t-Q1''*› ‹*t-Q1''* ≤ *t-Q1'*›
**by** (*metis Prefix-Order.prefixI le-trace-hide*)
**from** *F-T* ‹(*t-Q1'*, *ev* ‘ *A*) $\in \mathcal{F}$ *Q*› ‹*t-Q1''* ≤ *t-Q1'*› *is-processT3-TR*
**have** ‹*t-Q1''* $\in \mathcal{T}$ *Q*› **by** *blast*
**from** ∗∗(*3*)[*unfolded* ‹*t-Q1* = *?th-A t-Q1''*›,
*THEN disjoint-trace-hide-setinterleaves*$_{ptick}$[*OF* ‹$A \cap S = \{\}$›]]
**obtain** *u1'* **where** ‹*u1* = *?th-A u1'*› ‹*u1' setinterleaves*$_{✓(\otimes✓)}$ ((*t-P1*, *t-Q1''*), *S*)› **by** *blast*
**from** *D-P*(*4*) **show** ‹*u* @ *v* $\in \mathcal{D}$ (*P* ⟦*S*⟧$_{✓}$ *Q* $\setminus$ *A*)›
**proof** (*elim disjE exE*)
**assume** ‹*t-P1* $\in \mathcal{D}$ *P*›
**with** ‹*u1' setinterleaves*$_{✓(\otimes✓)}$ ((*t-P1*, *t-Q1''*), *S*)› *D-P*(*1*) *setinterleaves*$_{ptick}$*-tickFree-imp*
**have** ‹*u1'* = *u1'* @ []› ‹*tF u1'*› ‹*ftF* []›

```
    ‹u1′ setinterleaves✓(⊗✓) ((t-P1, t-Q1′′), S)› ‹t-P1 ∈ 𝒟 P›
    by simp-all (blast intro: is-processT3-TR)+
   with ‹t-Q1′′ ∈ 𝒯 Q› have ‹u1′ ∈ 𝒟 (P ⟦S⟧✓ Q)› unfolding D-Sync_ptick
by blast
   moreover have ‹u @ v = ?th-A u1′ @ (u2 @ v)›
    by (simp add: ∗(1) ∗∗(1) ‹u1 = ?th-A u1′›)
   ultimately show ‹u @ v ∈ 𝒟 (P ⟦S⟧✓ Q \ A)›
    unfolding D-Hiding using ‹tF u› ‹ftF v› ∗∗(1) ‹tF u1′›
    by (auto intro: front-tickFree-append)
  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 u1′›
    by (intro disjoint-isInfHidden-seqRunL-to-Sync_ptick
        [OF ‹A ∩ S = {}› - ‹t-Q1′′ ∈ 𝒯 Q› ‹u1′ setinterleaves✓(⊗✓) ((t-P1,
t-Q1′′), S)›]) simp
   with ∗∗(1) ‹u1 = ?th-A u1′› assms(2, 3) show ‹u @ v ∈ 𝒟 (P ⟦S⟧✓ Q \
A)›
    unfolding D-Hiding-seqRun by clarify
     (metis append-eq-append-conv2[of u1 ‹u2 @ v› ‹u1 @ u2› v]
       isInfHidden-seqRun-imp-tickFree[of u1′ ‹ev ∘ of-ev ∘ t-P-x› ‹P ⟦S⟧✓ Q›
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 ‹?th-A t-P1› ‹?th-A t-Q1′›]
  consider (left) u′ t-Q1′′ where ‹u′ ≤ u› ‹t-Q1′′ ≤ t-Q1′›
   ‹u′ setinterleaves✓(⊗✓) ((?th-A t-P1, ?th-A t-Q1′′), S)›
  | (right) u′ t-P1′ where ‹u′ ≤ u› ‹t-P1′ ≤ t-P1›
   ‹u′ setinterleaves✓(⊗✓) ((?th-A t-P1′, ?th-A t-Q1′), S)›
   by (auto dest!: le-trace-hide)
  thus ‹u @ v ∈ 𝒟 (P ⟦S⟧✓ Q \ A)›
  proof cases
   case left
    have ‹t-Q1′′ ∈ 𝒯 Q› by (meson D-Q(4) D-T is-processT3-TR left(2)
t-le-seqRun)
   from disjoint-trace-hide-setinterleaves_ptick[OF ‹A ∩ S = {}› left(3)]
   obtain u′′ where $ : ‹u′ = ?th-A u′′›
    ‹u′′ setinterleaves✓(⊗✓) ((t-P1, t-Q1′′), S)› by blast
   from D-P(4) show ‹u @ v ∈ 𝒟 (P ⟦S⟧✓ Q \ A)›
   proof (elim disjE exE)
    assume ‹t-P1 ∈ 𝒟 P›
    hence ‹u′′ ∈ 𝒟 (P ⟦S⟧✓ Q)›
     by (simp add: D-Sync_ptick)
      (metis $(2) D-P(1) ‹t-Q1′′ ∈ 𝒯 Q› append.right-neutral
       front-tickFree-Nil setinterleaves_ptick-tickFree-imp)
    with left(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-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-Sync_ptick
         [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-setinterleaves_ptick[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-Sync_ptick)
      (metis $(2) D-Q(1) ‹t-P1' ∈ 𝒯 P› append.right-neutral
       front-tickFree-Nil setinterleaves_ptick-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-Sync_ptick
         [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

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** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**from** *this* **obtain** $u\ v$ *t-P t-Q*
**where** $*$ : ‹$t = u\ @\ v$› ‹*tF u*› ‹*ftF v*›
‹$u$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ S)$›
‹$t\text{-}P \in \mathcal{D}\ (P \setminus A) \wedge t\text{-}Q \in \mathcal{T}\ (Q \setminus A) \vee t\text{-}P \in \mathcal{T}\ (P \setminus A) \wedge t\text{-}Q \in \mathcal{D}\ (Q \setminus A)$›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**from** $*(5)$ **show** ‹$t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**proof** (*elim disjE conjE*)
**show** ‹$t\text{-}P \in \mathcal{D}\ (P \setminus A) \Longrightarrow t\text{-}Q \in \mathcal{T}\ (Q \setminus A) \Longrightarrow t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**by** (*simp add*: $*(1\text{–}4)$ *disjoint-Hiding-Sync*$_{ptick}$*-FD-Sync*$_{ptick}$*-Hiding-aux* ‹$A \cap S = \{\}$›)
**next**
**assume** ‹$t\text{-}P \in \mathcal{T}\ (P \setminus A)$› ‹$t\text{-}Q \in \mathcal{D}\ (Q \setminus A)$›
**have** ‹$u$ *setinterleaves*$_{\checkmark\lambda s\ r.\ r\ \otimes\checkmark\ s}$ $((t\text{-}Q,\ t\text{-}P),\ S)$›
**using** $*(4)$ *setinterleaves*$_{ptick}$*-sym* **by** *blast*
**from** *Sync*$_{ptick}$*-locale-sym.disjoint-Hiding-Sync*$_{ptick}$*-FD-Sync*$_{ptick}$*-Hiding-aux*
[*OF* ‹$A \cap S = \{\}$› $*(2,\ 3)$ ‹$t\text{-}Q \in \mathcal{D}\ (Q \setminus A)$› ‹$t\text{-}P \in \mathcal{T}\ (P \setminus A)$› *this*]
**have** ‹$u\ @\ v \in \mathcal{D}$ (*Sync*$_{ptick}$*-locale-sym.Sync*$_{ptick}$ $Q\ S\ P \setminus A)$› **.**
**also have** ‹*Sync*$_{ptick}$*-locale-sym.Sync*$_{ptick}$ $Q\ S\ P = P\ [\![S]\!]_{\checkmark}\ Q$› **by** (*fact Sync*$_{ptick}$*-sym*)
**finally show** ‹$t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$› **unfolding** $*(1)$ **.**
**qed**
**next**
**fix** $t\ X$ **assume** ‹$(t,\ X) \in \mathcal{F}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**and** *subset-div* : ‹$\mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A)) \subseteq \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**from** *this*$(1)$ **consider** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
| (*fail-Sync*) *t-P t-Q X-P X-Q* **where** ‹$(t\text{-}P,\ X\text{-}P) \in \mathcal{F}\ (P \setminus A)$› ‹$(t\text{-}Q,\ X\text{-}Q) \in \mathcal{F}\ (Q \setminus A)$›
‹$t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ S)$› ‹$X \subseteq$ *super-ref-Sync*$_{ptick}$ $(\otimes\checkmark)\ X\text{-}P\ S\ X\text{-}Q$›
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**proof** *cases*
**from** *subset-div* **show** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A)) \Longrightarrow (t,\ X) \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**by** (*simp add*: *in-mono is-processT8*)
**next**
**case** *fail-Sync*
**from** *fail-Sync*$(1,\ 2)$ **consider** ‹$t\text{-}P \in \mathcal{D}\ (P \setminus A) \vee t\text{-}Q \in \mathcal{D}\ (Q \setminus A)$›
| (*fail-Hiding*) *t-P′ t-Q′* **where**
‹$t\text{-}P = \mathit{trace\text{-}hide}\ t\text{-}P'\ (\mathit{ev}\ `\ A)$› ‹$(t\text{-}P',\ X\text{-}P \cup \mathit{ev}\ `\ A) \in \mathcal{F}\ P$›
‹$t\text{-}Q = \mathit{trace\text{-}hide}\ t\text{-}Q'\ (\mathit{ev}\ `\ A)$› ‹$(t\text{-}Q',\ X\text{-}Q \cup \mathit{ev}\ `\ A) \in \mathcal{F}\ Q$›
**unfolding** *F-Hiding D-Hiding* **by** *blast*
**thus** ‹$(t,\ X) \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**proof** *cases*
**assume** ‹$t\text{-}P \in \mathcal{D}\ (P \setminus A) \vee t\text{-}Q \in \mathcal{D}\ (Q \setminus A)$›
**with** *fail-Sync*$(1\text{–}3)$ **have** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**by** (*simp add*: *D-Sync*$_{ptick}{}'$)

```
      (metis F-T append-self-conv front-tickFree-Nil)
    with subset-div show ‹(t, X) ∈ ℱ (P ⟦S⟧✓ Q \ A)›
     by (simp add: in-mono is-processT8)
   next
    case fail-Hiding
    from disjoint-trace-hide-setinterleaves_ptick
     [OF ‹A ∩ S = {}› fail-Sync(3)[unfolded fail-Hiding(1, 3)]]
    obtain t' where ∗ : ‹t = trace-hide t' (ev ‘ A)›
     ‹t' setinterleaves✓(⊗✓) ((t-P', t-Q'), S)› by blast
    from fail-Sync(4) have ‹X ∪ ev ‘ A ⊆ super-ref-Sync_ptick (⊗✓) (X-P ∪ ev
‘ A) S (X-Q ∪ ev ‘ A)›
     by (auto simp add: super-ref-Sync_ptick-def image-iff)
    with ∗(2) fail-Hiding(2, 4) have ‹(t', X ∪ ev ‘ A) ∈ ℱ (P ⟦S⟧✓ Q)›
     by (auto simp add: F-Sync_ptick)
    with ∗(1) show ‹(t, X) ∈ ℱ (P ⟦S⟧✓ Q \ A)› unfolding F-Hiding by blast
   qed
  qed
qed

theorem (in Sync_ptick-locale) disjoint-finite-Hiding-Sync_ptick :
  ‹P ⟦S⟧✓ Q \ A = (P \ A) ⟦S⟧✓ (Q \ A)› if ‹A ∩ S = {}› and ‹finite A›
  — Monster theorem!
proof (rule FD-antisym)
  from disjoint-Hiding-Sync_ptick-FD-Sync_ptick-Hiding[OF ‹A ∩ S = {}›]
  show ‹P ⟦S⟧✓ Q \ A ⊑_FD (P \ A) ⟦S⟧✓ (Q \ A)› .
next
  let ?th-A = ‹λt. trace-hide t (ev ‘ A)›
  show ‹(P \ A) ⟦S⟧✓ (Q \ A) ⊑_FD P ⟦S⟧✓ Q \ A›
  proof (rule failure-divergence-refine-optimizedI)
    fix t X assume ‹(t, X) ∈ ℱ (P ⟦S⟧✓ Q \ A)›
     and subset-div : ‹𝒟 (P ⟦S⟧✓ Q \ A) ⊆ 𝒟 ((P \ A) ⟦S⟧✓ (Q \ A))›
    from this(1) consider ‹t ∈ 𝒟 (P ⟦S⟧✓ Q \ A)›
     | t' where ‹t = ?th-A t'› ‹(t', X ∪ ev ‘ A) ∈ ℱ (P ⟦S⟧✓ Q)›
     unfolding F-Hiding D-Hiding by blast
    thus ‹(t, X) ∈ ℱ ((P \ A) ⟦S⟧✓ (Q \ A))›
    proof cases
     show ‹t ∈ 𝒟 (P ⟦S⟧✓ Q \ A) ⟹ (t, X) ∈ ℱ ((P \ A) ⟦S⟧✓ (Q \ A))›
      using subset-div is-processT8 by blast
    next
     fix t' assume ‹t = ?th-A t'› ‹(t', X ∪ ev ‘ A) ∈ ℱ (P ⟦S⟧✓ Q)›
     from this(2) consider ‹t' ∈ 𝒟 (P ⟦S⟧✓ Q)›
      | (fail) t-P X-P t-Q X-Q where ‹(t-P, X-P) ∈ ℱ P› ‹(t-Q, X-Q) ∈ ℱ Q›
       ‹t' setinterleaves✓(⊗✓) ((t-P, t-Q), S)›
       ‹X ∪ ev ‘ A ⊆ super-ref-Sync_ptick (⊗✓) X-P S X-Q›
      unfolding Sync_ptick-projs by auto
     thus ‹(t, X) ∈ ℱ ((P \ A) ⟦S⟧✓ (Q \ A))›
     proof cases
```

**assume** ‹$t' \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)$›
**with** ‹$t = ?th\text{-}A\ t'$› **have** ‹$t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**by** (*metis mem-D-imp-mem-D-Hiding*)
**with** *subset-div is-processT8* **show** ‹$(t, X) \in \mathcal{F}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**by** *blast*
**next**
**case** *fail*
**from** ‹$A \cap S = \{\}$› *fail*(*4*) **have** ‹$X\text{-}P = X\text{-}P \cup ev\ `\ A$› ‹$X\text{-}Q = X\text{-}Q \cup ev\ `\ A$›
— i.e. $ev\ `\ A \subseteq X\text{-}P$ and $ev\ `\ A \subseteq X\text{-}Q$
**by** (*auto simp add*: *super-ref-Sync*$_{ptick}$*-def*)
**with** *fail*(*1*, *2*) **have** ‹$(?th\text{-}A\ t\text{-}P, X\text{-}P) \in \mathcal{F}\ (P \setminus A)$›
‹$(?th\text{-}A\ t\text{-}Q, X\text{-}Q) \in \mathcal{F}\ (Q \setminus A)$›
**by** (*auto simp add*: *F-Hiding*)
**moreover from** *fail*(*3*) **have** ‹$t$ *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((?th\text{-}A\ t\text{-}P,\ ?th\text{-}A\ t\text{-}Q),\ S)$›
**unfolding** ‹$t = ?th\text{-}A\ t'$› **by** (*fact setinterleaves*$_{ptick}$*-trace-hide*)
**ultimately show** ‹$(t, X) \in \mathcal{F}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**using** *fail*(*4*) **unfolding** *F-Sync*$_{ptick}$ **by** *fast*
**qed**
**qed**
**next**
**fix** $t$ **assume** ‹$t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$›
**then obtain** $u\ v$ **where** $*$ : ‹*ftF v*› ‹*tF u*› ‹$t = ?th\text{-}A\ u\ @\ v$›
‹$u \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q) \vee (\exists x.\ isInfHidden\text{-}seqRun\text{-}strong\ x\ (P\ [\![S]\!]_{\checkmark}\ Q)\ A\ u)$›
**by** (*blast elim*: *D-Hiding-seqRunE*)
**from** $*(4)$ **show** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**proof** (*elim disjE exE*)
**assume** ‹$u \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q)$›
**then obtain** *u1 u2 t-P t-Q* **where** $**$ : ‹$u = u1\ @\ u2$› ‹*tF u1*› ‹*ftF u2*›
‹*u1* *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((t\text{-}P,\ t\text{-}Q),\ S)$›
‹$t\text{-}P \in \mathcal{D}\ P \wedge t\text{-}Q \in \mathcal{T}\ Q \vee t\text{-}P \in \mathcal{T}\ P \wedge t\text{-}Q \in \mathcal{D}\ Q$›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**have** ‹$t = ?th\text{-}A\ u1\ @\ (?th\text{-}A\ u2\ @\ v)$›
**by** (*simp add*: $*(3)$ $**(1)$)
**moreover from** $**(2)$ **have** ‹*tF* (*?th-A u1*)› **using** *Hiding-tickFree* **by** *blast*
**moreover have** ‹*ftF* (*?th-A u2* @ *v*)›
**by** (*metis D-imp-front-tickFree* ‹$t \in \mathcal{D}\ (P\ [\![S]\!]_{\checkmark}\ Q \setminus A)$› *calculation*(*1*)
*front-tickFree-append-iff front-tickFree-charn*)
**moreover from** $**(4)$ **have** ‹*?th-A u1* *setinterleaves*$_{\checkmark(\otimes\checkmark)}$ $((?th\text{-}A\ t\text{-}P,\ ?th\text{-}A\ t\text{-}Q),\ S)$›
**by** (*fact setinterleaves*$_{ptick}$*-trace-hide*)
**moreover from** $**(5)$ **have** ‹$?th\text{-}A\ t\text{-}P \in \mathcal{D}\ (P \setminus A) \wedge ?th\text{-}A\ t\text{-}Q \in \mathcal{T}\ (Q \setminus A) \vee$
$?th\text{-}A\ t\text{-}P \in \mathcal{T}\ (P \setminus A) \wedge ?th\text{-}A\ t\text{-}Q \in \mathcal{D}\ (Q \setminus A)$›
**by** (*metis mem-D-imp-mem-D-Hiding mem-T-imp-mem-T-Hiding*)
**ultimately show** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$›
**unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**next**

**fix** $x$ **assume** $**$ : ‹*isInfHidden-seqRun-strong* $x$ ($P$ $[\![S]\!]_{\checkmark}$ $Q$) $A$ $u$›
**from** $**$ **have** $***$ : ‹$\exists$ *t-P t-Q*. *t-P* $\in \mathcal{T}$ $P$ $\wedge$ *t-Q* $\in \mathcal{T}$ $Q$ $\wedge$
*seqRun u x i setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), $S$)› **for** $i$
**unfolding** *Sync*$_{ptick}$*-projs* **by** *blast*

**define** *t-P-t-Q* **where** ‹*t-P-t-Q* $i \equiv$ *SOME* (*t-P*, *t-Q*). *t-P* $\in \mathcal{T}$ $P$ $\wedge$ *t-Q* $\in \mathcal{T}$ $Q$ $\wedge$
*seqRun u x i setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*t-P*, *t-Q*), $S$)› **for** $i$
**define** *t-P* **where** ‹*t-P* $\equiv$ *fst* $\circ$ *t-P-t-Q*›
**define** *t-Q* **where** ‹*t-Q* $\equiv$ *snd* $\circ$ *t-P-t-Q*›
**have** $****$ : ‹*t-P* $i \in \mathcal{T}$ $P$› ‹*t-Q* $i \in \mathcal{T}$ $Q$›
‹*seqRun u x i setinterleaves*$_{\checkmark(\otimes\checkmark)}$ ((*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*) $\subseteq$ {*ev a* |*a*. *ev a* $\in$ *set u*} $\cup$ *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* $\in$ *set* (*t-P i*)} $\cup$ {*ev a* |*a*. *ev a* $\in$ *set* (*t-Q i*)} $\subseteq$
{*ev a* |*a*. *ev a* $\in$ *set u*} $\cup$ *ev* ‘ $A$› **for** $i$
**by** (*rule subset-trans*[*OF setinterleaves*$_{ptick}$*-imp-superset-ev*[*OF* $****(3)$]])
(*use* ‹*set* (*seqRun u x i*) $\subseteq$ {*ev a* |*a*. *ev a* $\in$ *set u*} $\cup$ *ev* ‘ $A$› **in** *blast*)
**have** $******$ : ‹*tF* (*t-P i*)› ‹*tF* (*t-Q i*)› **for** $i$
**using** *tickFree-setinterleaves*$_{ptick}$*-iff*[*OF* $****(3)$[*of* $i$]]
**by** (*metis* $*(2)$ $**$ *event*$_{ptick}$*.disc*(1) *imageE tickFree-seqRun-iff*)+

**{ fix** $i$
**have** ‹{*w*. *tF w* $\wedge$ {*ev a* |*a*. *ev a* $\in$ *set w*} $\subseteq$ *set u* $\cup$ *ev* ‘ $A$ $\wedge$ *length w* $\leq i$} $\subseteq$
*map* (*ev* $\circ$ *of-ev*) ‘ {*w*. *set w* $\subseteq$ *set u* $\cup$ *ev* ‘ $A$ $\wedge$ *length w* $\leq i$}›
(**is** ‹*?S1* $\subseteq$ *map* (*ev* $\circ$ *of-ev*) ‘ *?S2*›)
**proof** (*rule subsetI*)
**fix** $w$ **assume** ‹$w \in$ *?S1*›
**hence** ‹*map* (*ev* $\circ$ *of-ev*) (*map* (*ev* $\circ$ *of-ev*) $w$) = $w$›
**by** (*induct w*) (*auto simp add*: *subset-iff*)
**moreover from** ‹$w \in$ *?S1*› **have** ‹*map* (*ev* $\circ$ *of-ev*) $w \in$ *?S2*›
**by** (*induct w*) (*auto simp add*: *subset-iff*)
**ultimately show** ‹$w \in$ *map* (*ev* $\circ$ *of-ev*) ‘ *?S2*›
**by** (*metis* (*lifting*) *image-eqI*)
**qed**
**moreover have** ‹*finite* {*w*. *set w* $\subseteq$ *set u* $\cup$ *ev* ‘ $A$ $\wedge$ *length w* $\leq i$}›
**by** (*rule finite-lists-length-le*) (*simp add*: ‹*finite* $A$›)
**ultimately have** ‹*finite* {*w*. *tF w* $\wedge$ {*ev a* |*a*. *ev a* $\in$ *set w*} $\subseteq$ *set u* $\cup$ *ev* ‘ $A$ $\wedge$ *length w* $\leq i$}›
**using** *finite-subset*[*OF - finite-imageI*] **by** *blast*
**}** **note** $£ =$ *this*

**have** ‹*inj t-P-t-Q*›

**proof** (*rule injI*)
**fix** *i j* **assume** ‹*t-P-t-Q i = t-P-t-Q j*›
**with** ****(*3*) **have** ‹*seqRun u x i setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P i*, *t-Q i*), *S*)
∧
*seqRun u x j setinterleaves*$_{\checkmark\, tick\text{-}join}$ ((*t-P i*, *t-Q i*), *S*)›
**unfolding** *t-P-t-Q-def t-P-def t-Q-def* **by** *fastforce*
**with** *setinterleaves*$_{ptick}$*-eq-length*
**have** ‹*length* (*seqRun u x i*) = *length* (*seqRun u x j*)› **by** *blast*
**thus** ‹*i* = *j*› **by** *simp*
**qed**
**hence** ‹*infinite* (*range t-P-t-Q*)› **using** *finite-imageD* **by** *blast*
**moreover have** ‹*range t-P-t-Q* ⊆ *range t-P* × *range t-Q*›
**by** (*simp add*: *t-P-def t-Q-def subset-iff image-iff*) (*metis fst-conv snd-conv*)
**ultimately have** ‹*infinite* (*range t-P*) ∨ *infinite* (*range t-Q*)›
**by** (*meson finite-SigmaI infinite-super*)

**thus** ‹*t* ∈ $\mathcal{D}$ ((*P* \ *A*) ⟦*S*⟧$_{\checkmark}$ (*Q* \ *A*))›
**proof** (*elim disjE*)
**assume** ‹*infinite* (*range t-P*)›
**have** ‹*finite* {*w*. ∃ *t'*∈*range t-P*. *w* = *take i t'*}› **for** *i*
**using** ******(*1*) *****
**by** (*auto intro*!: *finite-subset*[*OF - £*[*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** ‹*infinite* (*range t-P*)› **obtain** *t-P'* :: ‹*nat* ⇒ -›
**where** $ : ‹*strict-mono t-P'*› ‹*range t-P'* ⊆ {*w*. ∃ *t'*∈*range t-P*. *w* ≤ *t'*}›
**using** *KoenigLemma* **by** *blast*
**from** $(*2*) ****(*1*) *is-processT3-TR* **have** ‹*range t-P'* ⊆ $\mathcal{T}$ *P*› **by** *blast*
**define** *t-P''* **where** ‹*t-P'' i* ≡ *t-P'* (*i* + *length u*)› **for** *i*
**from** ‹*range t-P'* ⊆ $\mathcal{T}$ *P*› **have** ‹*range t-P''* ⊆ $\mathcal{T}$ *P*› **and** ‹*strict-mono t-P''*›
**by** (*auto simp add*: *t-P''-def* $(*1*) *strict-monoD strict-monoI*)
**have** $$ : ‹*?th-A* (*t-P'' i*) = *?th-A* (*t-P'' 0*)› **for** *i*
**proof** −
**have** ‹*length u* ≤ *length* (*t-P'' 0*)›
**by** (*metis* $(*1*) *add-0 add-leD1 t-P''-def length-strict-mono*)
**obtain** *t'* **where** ‹*t-P'' i* = *t-P'' 0* @ *t'*›
**by** (*meson prefixE* ‹*strict-mono t-P''*› *strict-mono-less-eq zero-order*(*1*))
**moreover from** $(*2*) **obtain** *j* **where** ‹*t-P'' i* ≤ *t-P j*› **by** (*auto simp add*: *t-P''-def*)
**ultimately obtain** *t''* **where** ‹*t-P j* = *t-P'' 0* @ *t'* @ *t''*› **by** (*metis prefixE append.assoc*)

**have** ‹*tF* (*t'* @ *t''*)›
**by** (*metis* ******(*1*) ‹*t-P j* = *t-P'' 0* @ *t'* @ *t''*› *tickFree-append-iff*)
**with** *setinterleaves*$_{ptick}$*-set-subsetL*
[*OF* ****(*3*)[*of j*], **where** *n* = ‹*length* (*t-P'' 0*)›, *unfolded* ‹*t-P j* = *t-P'' 0* @ *t'* @ *t''*›]
**have** ‹*e* ∈ *set* (*t'* @ *t''*) ⟹ *e* ∈ {*ev a* |*a*. *ev a* ∈ *set* (*drop* (*length* (*t-P'' 0*)) (*seqRun u x j*))}› **for** *e*

**by** (*cases e*) (*auto simp add*: *tickFree-def*)
**moreover have** ‹{*a*. *ev a* ∈ *set* (*drop* (*length* (*t-P″ 0*)) (*seqRun u x j*))}
⊆
{*a*. *ev a* ∈ *set* (*drop* (*length u*) (*seqRun u x j*))}›
**by** (*simp add*: *subset-iff*)
(*meson* ‹*length u* ≤ *length* (*t-P″ 0*)› *in-mono set-drop-subset-set-drop*)
**moreover from** ∗∗ **have** ‹*set* (*drop* (*length u*) (*seqRun u x j*)) ⊆ *ev* ‘ *A*›
**by** (*auto simp add*: *seqRun-def*)
**ultimately have** ‹*set* (*t′* @ *t″*) ⊆ *ev* ‘ *A*› **by** *blast*
**thus** ‹*?th-A* (*t-P″ i*) = *?th-A* (*t-P″ 0*)›
**by** (*simp add*: ‹*t-P″ i* = *t-P″ 0* @ *t′*› *subset-iff*)
**qed**
**from** $(*2*) **obtain** *i* **where** ‹*t-P″ 0* ≤ *t-P i*› **by** (*auto simp add*: *t-P″-def*)
**with** *prefixE* **obtain** *w* **where** ‹*t-P i* = *t-P″ 0* @ *w*› **by** *blast*
**have** ‹*ftF v*› **by** (*fact* ∗(*1*))
**moreover have** ‹*tF* (*?th-A* (*seqRun u x i*))›
**by** (*metis* ∗(*2*) ∗∗ *Hiding-tickFree trace-hide-seqRun-eq-iff*)
**moreover have** ‹*t* = *?th-A* (*seqRun u x i*) @ *v*›
**by** (*metis* ∗(*3*) ∗∗ *trace-hide-seqRun-eq-iff*)
**moreover have** ‹*?th-A* (*seqRun u x i*) *setinterleaves*$_{✓(⊗✓)}$ ((*?th-A* (*t-P i*), *?th-A* (*t-Q i*)), *S*)›
**by** (*intro setinterleaves*$_{ptick}$*-trace-hide* ∗∗∗∗(*3*))
**moreover have** ‹*?th-A* (*t-P i*) ∈ $\mathcal{D}$ (*P* \ *A*)›
**proof** (*unfold D-Hiding*, *clarify*, *intro exI conjI*)
**show** ‹*ftF* (*?th-A w*)›
**by** (*metis* ∗∗∗∗∗∗(*1*) *Hiding-front-tickFree* ‹*t-P i* = *t-P″ 0* @ *w*›
*tickFree-append-iff tickFree-imp-front-tickFree*)
**next**
**show** ‹*tF* (*t-P″ 0*)›
**by** (*metis* ∗∗∗∗∗∗(*1*) ‹*t-P i* = *t-P″ 0* @ *w*› *tickFree-append-iff*)
**next**
**show** ‹*?th-A* (*t-P i*) = *?th-A* (*t-P″ 0*) @ *?th-A w*›
**by** (*simp add*: ‹*t-P i* = *t-P″ 0* @ *w*›)
**next**
**show** ‹*t-P″ 0* ∈ $\mathcal{D}$ *P* ∨ (∃ *f*. *isInfHiddenRun f P A* ∧ *t-P″ 0* ∈ *range f*)›
**using** $$ ‹*range t-P″* ⊆ $\mathcal{T}$ *P*› ‹*strict-mono t-P″*› **by** *blast*
**qed**
**moreover have** ‹*?th-A* (*t-Q i*) ∈ $\mathcal{T}$ (*Q* \ *A*)›
**proof** (*cases* ‹∃ *t′*. *?th-A t′* = *?th-A* (*t-Q i*) ∧ (*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *Q*›)
**assume** ‹∃ *t′*. *?th-A t′* = *?th-A* (*t-Q i*) ∧ (*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *Q*›
**then obtain** *t′* **where** ‹*?th-A* (*t-Q i*) = *?th-A t′*› ‹(*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *Q*› **by** *metis*
**thus** ‹*?th-A* (*t-Q i*) ∈ $\mathcal{T}$ (*Q* \ *A*)› **unfolding** *T-Hiding* **by** *blast*
**next**
**assume** ‹∄ *t′*. *?th-A t′* = *?th-A* (*t-Q i*) ∧ (*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *Q*›
**with** *inf-hidden*[*OF* - ∗∗∗∗(*2*)] **obtain** *t-Q′ j*
**where** ‹*isInfHiddenRun t-Q′ Q A*› ‹*t-Q i* = *t-Q′ j*› **by** *blast*
**thus** ‹*?th-A* (*t-Q i*) ∈ $\mathcal{T}$ (*Q* \ *A*)›
**unfolding** *T-Hiding* **using** ∗∗∗∗∗∗(*2*) *front-tickFree-Nil* **by** *blast*

**qed**
**ultimately show** ‹$t \in \mathcal{D}\ ((P \setminus A)\ [\![S]\!]_{\checkmark}\ (Q \setminus A))$› **unfolding** $D\text{-}Sync_{ptick}$
**by** *blast*
**next**
**assume** ‹*infinite* (*range t-Q*)›
**have** ‹*finite* $\{w.\ \exists\, t' \in range\ t\text{-}Q.\ w = take\ i\ t'\}$› **for** *i*
**using** ******(2) *****
**by** (*auto intro*!: *finite-subset*[*OF - £*[*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** ‹*infinite* (*range t-Q*)› **obtain** *t-Q'* :: ‹*nat* ⇒ -›
**where** $ : ‹*strict-mono t-Q'*› ‹*range t-Q'* ⊆ $\{w.\ \exists\, t' \in range\ t\text{-}Q.\ w \le t'\}$›
**using** *KoenigLemma* **by** *blast*
**from** $(2) ****(2) *is-processT3-TR* **have** ‹*range t-Q'* ⊆ $\mathcal{T}$ *Q*› **by** *blast*
**define** *t-Q''* **where** ‹*t-Q'' i* ≡ *t-Q'* (*i* + *length u*)› **for** *i*
**from** ‹*range t-Q'* ⊆ $\mathcal{T}$ *Q*› **have** ‹*range t-Q''* ⊆ $\mathcal{T}$ *Q*› **and** ‹*strict-mono t-Q''*›
**by** (*auto simp add*: *t-Q''-def* $(1) *strict-monoD strict-monoI*)
**have** $$ : ‹*?th-A* (*t-Q'' i*) = *?th-A* (*t-Q'' 0*)› **for** *i*
**proof** −
**have** ‹*length u* ≤ *length* (*t-Q'' 0*)›
**by** (*metis* $(1) *add-0 add-leD1 t-Q''-def length-strict-mono*)
**obtain** *t'* **where** ‹*t-Q'' i* = *t-Q'' 0* @ *t'*›
**by** (*meson prefixE* ‹*strict-mono t-Q''*› *strict-mono-less-eq zero-order*(1))
**moreover from** $(2) **obtain** *j* **where** ‹*t-Q'' i* ≤ *t-Q j*› **by** (*auto simp add*: *t-Q''-def*)
**ultimately obtain** *t''* **where** ‹*t-Q j* = *t-Q'' 0* @ *t'* @ *t''*› **by** (*metis prefixE append.assoc*)
**have** ‹*tF* (*t'* @ *t''*)›
**by** (*metis* ******(2) ‹*t-Q j* = *t-Q'' 0* @ *t'* @ *t''*› *tickFree-append-iff*)
**with** $setinterleaves_{ptick}$*-set-subsetR*
[*OF* ****(3)[*of j*], **where** *n* = ‹*length* (*t-Q'' 0*)›, *unfolded* ‹*t-Q j* = *t-Q'' 0* @ *t'* @ *t''*›]
**have** ‹*e* ∈ *set* (*t'* @ *t''*) ⟹ *e* ∈ $\{ev\ a\ |a.\ ev\ a \in set\ (drop\ (length\ (t\text{-}Q''\ 0))\ (seqRun\ u\ x\ j))\}$› **for** *e*
**by** (*cases e*) (*auto simp add*: *tickFree-def*)
**moreover have** ‹$\{a.\ ev\ a \in set\ (drop\ (length\ (t\text{-}Q''\ 0))\ (seqRun\ u\ x\ j))\}$ ⊆
$\{a.\ ev\ a \in set\ (drop\ (length\ u)\ (seqRun\ u\ x\ j))\}$›
**by** (*simp add*: *subset-iff*)
(*meson* ‹*length u* ≤ *length* (*t-Q'' 0*)› *in-mono set-drop-subset-set-drop*)
**moreover from** ** **have** ‹*set* (*drop* (*length u*) (*seqRun u x j*)) ⊆ *ev* ‘ *A*›
**by** (*auto simp add*: *seqRun-def*)
**ultimately have** ‹*set* (*t'* @ *t''*) ⊆ *ev* ‘ *A*› **by** *blast*
**thus** ‹*?th-A* (*t-Q'' i*) = *?th-A* (*t-Q'' 0*)›
**by** (*simp add*: ‹*t-Q'' i* = *t-Q'' 0* @ *t'*› *subset-iff*)
**qed**
**from** $(2) **obtain** *i* **where** ‹*t-Q'' 0* ≤ *t-Q i*› **by** (*auto simp add*: *t-Q''-def*)
**with** *prefixE* **obtain** *w* **where** ‹*t-Q i* = *t-Q'' 0* @ *w*› **by** *blast*
**have** ‹*ftF v*› **by** (*fact* *(1))

**moreover have** ‹*tF* (*?th-A* (*seqRun u x i*))›
**by** (*metis* ∗(*2*) ∗∗ *Hiding-tickFree trace-hide-seqRun-eq-iff*)
**moreover have** ‹*t* = *?th-A* (*seqRun u x i*) @ *v*›
**by** (*metis* ∗(*3*) ∗∗ *trace-hide-seqRun-eq-iff*)
**moreover have** ‹*?th-A* (*seqRun u x i*) *setinterleaves*$_{✓(⊗✓)}$ ((*?th-A* (*t-P i*), *?th-A* (*t-Q i*)), *S*)›
**by** (*intro setinterleaves*$_{ptick}$*-trace-hide* ∗∗∗∗(*3*))
**moreover have** ‹*?th-A* (*t-Q i*) ∈ $\mathcal{D}$ (*Q* \ *A*)›
**proof** (*unfold D-Hiding*, *clarify*, *intro exI conjI*)
**show** ‹*ftF* (*?th-A w*)›
**by** (*metis* ∗∗∗∗∗∗(*2*) *Hiding-front-tickFree* ‹*t-Q i* = *t-Q′′ 0* @ *w*› *tickFree-append-iff tickFree-imp-front-tickFree*)
**next**
**show** ‹*tF* (*t-Q′′ 0*)›
**by** (*metis* ∗∗∗∗∗∗(*2*) ‹*t-Q i* = *t-Q′′ 0* @ *w*› *tickFree-append-iff*)
**next**
**show** ‹*?th-A* (*t-Q i*) = *?th-A* (*t-Q′′ 0*) @ *?th-A w*›
**by** (*simp add*: ‹*t-Q i* = *t-Q′′ 0* @ *w*›)
**next**
**show** ‹*t-Q′′ 0* ∈ $\mathcal{D}$ *Q* ∨ (∃ *f*. *isInfHiddenRun f Q A* ∧ *t-Q′′ 0* ∈ *range f*)›
**using** $$ ‹*range t-Q′′* ⊆ $\mathcal{T}$ *Q*› ‹*strict-mono t-Q′′*› **by** *blast*
**qed**
**moreover have** ‹*?th-A* (*t-P i*) ∈ $\mathcal{T}$ (*P* \ *A*)›
**proof** (*cases* ‹∃ *t′*. *?th-A t′* = *?th-A* (*t-P i*) ∧ (*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *P*›)
**assume** ‹∃ *t′*. *?th-A t′* = *?th-A* (*t-P i*) ∧ (*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *P*›
**then obtain** *t′* **where** ‹*?th-A* (*t-P i*) = *?th-A t′*› ‹(*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *P*› **by** *metis*
**thus** ‹*?th-A* (*t-P i*) ∈ $\mathcal{T}$ (*P* \ *A*)› **unfolding** *T-Hiding* **by** *blast*
**next**
**assume** ‹∄ *t′*. *?th-A t′* = *?th-A* (*t-P i*) ∧ (*t′*, *ev* ‘ *A*) ∈ $\mathcal{F}$ *P*›
**with** *inf-hidden*[*OF* - ∗∗∗∗(*1*)] **obtain** *t-P′ j*
**where** ‹*isInfHiddenRun t-P′ P A*› ‹*t-P i* = *t-P′ j*› **by** *blast*
**thus** ‹*?th-A* (*t-P i*) ∈ $\mathcal{T}$ (*P* \ *A*)›
**unfolding** *T-Hiding* **using** ∗∗∗∗∗∗(*1*) *front-tickFree-Nil* **by** *blast*
**qed**
**ultimately show** ‹*t* ∈ $\mathcal{D}$ ((*P* \ *A*) ⟦*S*⟧$_{✓}$ (*Q* \ *A*))› **unfolding** *D-Sync*$_{ptick}$ **by** *blast*
**qed**
**qed**
**qed**
**qed**

**lemma** *disjoint-Hiding-MultiSync*$_{ptick}$*-FD-MultiSync*$_{ptick}$*-Hiding* :
‹⟦*S*⟧$_{✓}$ *l* ∈@ *L*. *P l* \ *A* ⊑$_{FD}$ ⟦*S*⟧$_{✓}$ *l* ∈@ *L*. (*P l* \ *A*)› **if** ‹*A* ∩ *S* = {}›
**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** ‹$[\![S]\!]_{\checkmark}$ *l* ∈@ (*l0* # *l1* # *L*). *P l* \ *A* =
*P l0* $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l* \ *A*› **by** *simp*
**also have** ‹... $\sqsubseteq_{FD}$ (*P l0* \ *A*) $[\![S]\!]_{\checkmark Rlist}$ ($[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l* \ *A*)›
**by** (*simp add*: $Sync_{Rlist}$.*disjoint-Hiding-*$Sync_{ptick}$*-FD-*$Sync_{ptick}$*-Hiding* ‹*A* ∩ *S* = {}›)
**also have** ‹... $\sqsubseteq_{FD}$ (*P l0* \ *A*) $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). (*P l* \ *A*)›
**by** (*simp add*: *3.hyps*(*2*) $Sync_{Rlist}$.*mono-*$Sync_{ptick}$*-FD*)
**also have** ‹... = $[\![S]\!]_{\checkmark}$ *l* ∈@ (*l0* # *l1* # *L*). (*P l* \ *A*)› **by** *simp*
**finally show** *?case* **.**
**qed**

**lemma** *disjoint-finite-Hiding-*$MultiSync_{ptick}$ :
‹$[\![S]\!]_{\checkmark}$ *l* ∈@ *L*. *P l* \ *A* = $[\![S]\!]_{\checkmark}$ *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** ‹$[\![S]\!]_{\checkmark}$ *l* ∈@ (*l0* # *l1* # *L*). *P l* \ *A* =
*P l0* $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l* \ *A*› **by** *simp*
**also have** ‹... = (*P l0* \ *A*) $[\![S]\!]_{\checkmark Rlist}$ ($[\![S]\!]_{\checkmark}$ *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*) $[\![S]\!]_{\checkmark Rlist}$ $[\![S]\!]_{\checkmark}$ *l* ∈@ (*l1* # *L*). (*P l* \ *A*)›
**by** (*simp add*: *3.hyps*(*2*) $Sync_{Rlist}$.*mono-*$Sync_{ptick}$*-FD*)
**also have** ‹... = $[\![S]\!]_{\checkmark}$ *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*$_{\checkmark}$*tick-join* ((*u*, *v*), *S*) ⟷
*t setinterleaves*$_{\checkmark}$*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* :

$\langle P \ [\![S]\!]_{\checkmark} \ Q = P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q\rangle$
**proof** −
**have** $*$ : $\langle t\text{-}P \in \mathcal{D} \ P \wedge t\text{-}Q \in \mathcal{T} \ Q \vee t\text{-}P \in \mathcal{T} \ P \wedge t\text{-}Q \in \mathcal{D} \ Q \Longrightarrow$
$\{a.\ ev\ a \in set\ t\text{-}P \vee ev\ a \in set\ t\text{-}Q\} \subseteq \alpha(P) \cup \alpha(Q)\rangle$
$\langle (t\text{-}P,\ X\text{-}P) \in \mathcal{F} \ P \Longrightarrow (t\text{-}Q,\ X\text{-}Q) \in \mathcal{F} \ Q \Longrightarrow$
$\{a.\ ev\ a \in set\ t\text{-}P \vee ev\ a \in set\ t\text{-}Q\} \subseteq \alpha(P) \cup \alpha(Q)\rangle$ **for** $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$
**by** (*auto intro*: *events-of-memI dest*: *F-T D-T*)
**show** $\langle P \ [\![S]\!]_{\checkmark} \ Q = P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q\rangle$
**proof** (*rule Process-eq-optimizedI*)
**show** $\langle t \in \mathcal{D} \ (P \ [\![S]\!]_{\checkmark} \ Q) \Longrightarrow t \in \mathcal{D} \ (P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q)\rangle$ **for** $t$
**using** $setinterleaves_{ptick}$-*is-restrictable-on-superset-events-of*[*OF* $*(1)$]
**unfolding** $D\text{-}Sync_{ptick}$ **by** *blast*
**next**
**show** $\langle t \in \mathcal{D} \ (P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q) \Longrightarrow t \in \mathcal{D} \ (P \ [\![S]\!]_{\checkmark} \ Q)\rangle$ **for** $t$
**using** $setinterleaves_{ptick}$-*is-restrictable-on-superset-events-of*[*OF* $*(1)$]
**unfolding** $D\text{-}Sync_{ptick}$ **by** *blast*
**next**
**fix** $t\ X$ **assume** $\langle (t,\ X) \in \mathcal{F} \ (P \ [\![S]\!]_{\checkmark} \ Q)\rangle$ $\langle t \notin \mathcal{D} \ (P \ [\![S]\!]_{\checkmark} \ Q)\rangle$
**then obtain** $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$
**where** \$ : $\langle (t\text{-}P,\ X\text{-}P) \in \mathcal{F} \ P\rangle$ $\langle (t\text{-}Q,\ X\text{-}Q) \in \mathcal{F} \ Q\rangle$
$\langle t\ setinterleaves_{\checkmark}\ tick\text{-}join \ ((t\text{-}P,\ t\text{-}Q),\ S)\rangle$
$\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick} \ tick\text{-}join \ X\text{-}P \ S \ X\text{-}Q\rangle$
**unfolding** $Sync_{ptick}$-*projs* **by** *blast*
**from** \$(*1*) **have** $\langle (t\text{-}P,\ X\text{-}P \cup \{ev\ a \mid a.\ a \notin \alpha(P)\}) \in \mathcal{F} \ P\rangle$
**by** (*rule is-processT5*) (*auto intro*: *events-of-memI dest*!: *F-T*)
**moreover from** \$(*2*) **have** $\langle (t\text{-}Q,\ X\text{-}Q \cup \{ev\ a \mid a.\ a \notin \alpha(Q)\}) \in \mathcal{F} \ Q\rangle$
**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** $\langle t\ setinterleaves_{\checkmark}\ tick\text{-}join \ ((t\text{-}P,\ t\text{-}Q),\ S \cap (\alpha(P) \cup \alpha(Q)))\rangle$ **by** *blast*
**moreover from** \$(*4*)
**have** $\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick}$
$tick\text{-}join \ (X\text{-}P \cup \{ev\ a \mid a.\ a \notin \alpha(P)\})$
$(S \cap (\alpha(P) \cup \alpha(Q))) \ (X\text{-}Q \cup \{ev\ a \mid a.\ a \notin \alpha(Q)\})\rangle$
**by** (*auto simp add*: $super\text{-}ref\text{-}Sync_{ptick}$-*def*)
**ultimately show** $\langle (t,\ X) \in \mathcal{F} \ (P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q)\rangle$
**by** (*auto simp add*: $F\text{-}Sync_{ptick}$)
**next**
**fix** $t\ X$ **assume** $\langle (t,\ X) \in \mathcal{F} \ (P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q)\rangle$
$\langle t \notin \mathcal{D} \ (P \ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark} \ Q)\rangle$
**then obtain** $t\text{-}P\ t\text{-}Q\ X\text{-}P\ X\text{-}Q$
**where** \$ : $\langle (t\text{-}P,\ X\text{-}P) \in \mathcal{F} \ P\rangle$ $\langle (t\text{-}Q,\ X\text{-}Q) \in \mathcal{F} \ Q\rangle$
$\langle t\ setinterleaves_{\checkmark}\ tick\text{-}join \ ((t\text{-}P,\ t\text{-}Q),\ S \cap (\alpha(P) \cup \alpha(Q)))\rangle$
$\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick} \ tick\text{-}join \ X\text{-}P \ (S \cap (\alpha(P) \cup \alpha(Q))) \ X\text{-}Q\rangle$
**unfolding** $Sync_{ptick}$-*projs* **by** *blast*
**from** $setinterleaves_{ptick}$-*is-restrictable-on-superset-events-of*
[*OF* $*(2)$[*OF* \$(*1*, *2*)]] \$(*3*)
**have** $\langle t\ setinterleaves_{\checkmark}\ tick\text{-}join \ ((t\text{-}P,\ t\text{-}Q),\ S)\rangle$ **by** *blast*
**moreover from** \$(*4*) **have** $\langle X \subseteq super\text{-}ref\text{-}Sync_{ptick} \ tick\text{-}join \ X\text{-}P \ S \ X\text{-}Q\rangle$

**by** (*meson Int-lower1 in-mono subsetI super-ref-Sync$_{ptick}$-mono*)
**ultimately show** ‹$(t, X) \in \mathcal{F}\ (P\ [\![S]\!]_{\checkmark}\ Q)$›
**using** $(*1*, *2*) **by** (*auto simp add*: *F-Sync$_{ptick}$*)
**qed**
**qed**

**corollary** (**in** *Sync$_{ptick}$-locale*) *Sync$_{ptick}$-is-restrictable-on-superset-events-of* :
‹$P\ [\![S]\!]_{\checkmark}\ Q = P\ [\![S \cap A]\!]_{\checkmark}\ Q$› **if** ‹$\alpha(P) \cup \alpha(Q) \subseteq A$›
**proof** (*rule trans*[*OF Sync$_{ptick}$-is-restrictable-on-events-of*],
*rule trans*[*OF - Sync$_{ptick}$-is-restrictable-on-events-of*[*symmetric*]])
**show** ‹$P\ [\![S \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark}\ Q = P\ [\![S \cap A \cap (\alpha(P) \cup \alpha(Q))]\!]_{\checkmark}\ Q$›
**using** ‹$\alpha(P) \cup \alpha(Q) \subseteq A$› **by** (*auto intro*: *arg-cong*[**where** $f = ‹\lambda S.\ P\ [\![S]\!]_{\checkmark}\ Q›$])
**qed**

**lemma** ‹$tF\ t \Longrightarrow \{a.\ ev\ a \in set\ u\} \cap S = \{\} \Longrightarrow a \in S \Longrightarrow$
$\neg\ t\ setinterleaves_{\checkmark\ tick\text{-}join}\ ((u, ev\ a\ \#\ v), S)$›
**proof** (*induct* ‹(*tick-join*, *u*, *S*, *v*)› *arbitrary*: *t u v*)
**case** (*ev-setinterleaving$_{ptick}$-Nil a u*)
**thus** *?case* **by** (*simp add*: *disjoint-iff*) (*meson tickFree-Cons-iff*)
**next**
**case** (*ev-setinterleaving$_{ptick}$-ev a u b v*)
**then show** *?case* **by** (*simp add*: *disjoint-iff*) (*meson tickFree-Cons-iff*)
**next**
**case** (*ev-setinterleaving$_{ptick}$-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-setinterleaving$_{ptick}$*)
**qed** *simp-all*

### 16.5.2 Some Refinements

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

**lemma** *Mndetprefix-Sync$_{ptick}$-Det-distr-FD* :
‹$(\sqcap\ a \in A \to (P\ a\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \to Q\ b))) \Box$
$(\sqcap\ b \in B \to ((\sqcap\ a \in A \to P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ Q\ b))$
$\sqsubseteq_{FD}\ (\sqcap\ a \in A \to P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \to Q\ b)$›
(**is** ‹*?lhs1* $\Box$ *?lhs2* $\sqsubseteq_{FD}$ *?rhs*›)
**if** ‹$A \neq \{\}$› ‹$B \neq \{\}$› ‹$A \cap C = \{\}$› ‹$B \cap C = \{\}$›
**proof** −
**have** ‹*?lhs1* $= \sqcap\ b{\in}B.\ \sqcap\ a{\in}A.\ (a \to (P\ a\ [\![C]\!]_{\checkmark}\ (b \to Q\ b)))$› (**is** ‹- = *?lhs1'*›)
**by** (*simp add*: ‹$A \neq \{\}$› ‹$B \neq \{\}$› *Mndetprefix-GlobalNdet*
*Sync$_{ptick}$-distrib-GlobalNdet-left Sync$_{ptick}$-distrib-GlobalNdet-right*
*write0-def GlobalNdet-Mprefix-distr*[*OF* ‹$B \neq \{\}$›, *symmetric*])
(*subst GlobalNdet-sets-commute*, *simp*)
**moreover have** ‹*?lhs2* $= \sqcap\ b{\in}B.\ \sqcap\ a{\in}A.\ (b \to (a \to P\ a\ [\![C]\!]_{\checkmark}\ Q\ b))$› (**is** ‹- =

*?lhs2'*›)
**by** (*simp add*: ‹$A \neq \{\}$› ‹$B \neq \{\}$› *Mndetprefix-GlobalNdet*
*Sync$_{ptick}$-distrib-GlobalNdet-left Sync$_{ptick}$-distrib-GlobalNdet-right*
*write0-def GlobalNdet-Mprefix-distr*[*OF* ‹$A \neq \{\}$›, *symmetric*])
**ultimately have** ‹*?lhs1* $\Box$ *?lhs2* = *?lhs1'* $\Box$ *?lhs2'*› **by** *simp*
**moreover have** ‹*?lhs1'* $\Box$ *?lhs2'* $\sqsubseteq_{FD}$ $\sqcap$ $b{\in}B.$ $\sqcap$ $a{\in}A.$ $(a \rightarrow (P\ a\ [\![C]\!]_{\checkmark}\ (b \rightarrow Q\ b)))$
$\Box\ (b \rightarrow ((a \rightarrow P\ a)\ [\![C]\!]_{\checkmark}\ Q\ b))$›
**by** (*auto simp add*: ‹$A \neq \{\}$› ‹$B \neq \{\}$› *refine-defs GlobalNdet-projs Det-projs write0-def*)
**moreover have** ‹$\ldots = \sqcap\ b{\in}B.\ \sqcap\ a{\in}A.\ ((a \rightarrow P\ a)\ [\![C]\!]_{\checkmark}\ (b \rightarrow Q\ b))$›
**by** (*rule mono-GlobalNdet-eq*, *rule mono-GlobalNdet-eq*,
*simp add*: *write0-def*, *subst Mprefix-Sync$_{ptick}$-Mprefix-indep*)
(*use* ‹$A \cap C = \{\}$› ‹$B \cap C = \{\}$› **in** *auto*)
**moreover have** ‹$\ldots$ = *?rhs*›
**by** (*simp add*: ‹$A \neq \{\}$› ‹$B \neq \{\}$› *Mndetprefix-GlobalNdet*
*Sync$_{ptick}$-distrib-GlobalNdet-left Sync$_{ptick}$-distrib-GlobalNdet-right*)
**ultimately show** ‹*?lhs1* $\Box$ *?lhs2* $\sqsubseteq_{FD}$ *?rhs*› **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* :
‹$[\![A \neq \{\};\ B \neq \{\};\ A \cap C = \{\};\ B \cap C = \{\}]\!] \Longrightarrow$
$(\sqcap\ a \in A \rightarrow (P\ a\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \rightarrow Q\ b)))\ \Box$
$(\sqcap\ b \in B \rightarrow ((\sqcap\ a \in A \rightarrow P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ Q\ b))$
$\sqsubseteq_{DT}\ (\sqcap\ a \in A \rightarrow P\ a)\ [\![\ C\ ]\!]_{\checkmark}\ (\sqcap\ b \in B \rightarrow Q\ b)$›
**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 $\mathit{lifelock\text{-}free}_{SKIPS}\ P = (\mathcal{D}\ P = \{\})$.

#### Sequential Composition

**lemma** ‹*non-terminating* $P \Longrightarrow P\ ;_{\checkmark}\ Q = \mathit{RenamingTick}\ P\ g$›
— Already proven earlier.
**by** (*fact* $\mathit{non\text{-}terminating\text{-}Seq}_{ptick}$)

#### Synchronization Product

**lemma** (**in** $\mathit{Sync}_{ptick}$-*locale*) $\mathit{non\text{-}terminating\text{-}Sync}_{ptick}$ :
‹*non-terminating* $P \Longrightarrow \mathit{lifelock\text{-}free}_{SKIPS}\ Q \Longrightarrow$ *non-terminating* $(P\ [\![A]\!]_{\checkmark}\ Q)$›
‹$\mathit{lifelock\text{-}free}_{SKIPS}\ P \Longrightarrow$ *non-terminating* $Q \Longrightarrow$ *non-terminating* $(P\ [\![A]\!]_{\checkmark}\ Q)$›
**by** (*simp add*: $\mathit{lifelock\text{-}free}_{SKIPS}$-*iff-div-free* $\mathit{T\text{-}Sync}_{ptick}$
*non-terminating-is-right nonterminating-implies-div-free*,
*use* $\mathit{setinterleaves}_{ptick}$-*tickFree-imp* **in** *blast*)+

### 17.1.2 Deadlock Free

#### Sequential Composition

**lemma** ‹*deadlock-free* $P \Longrightarrow$ *deadlock-free* $(P\ ;_{\checkmark}\ Q)$›
**by** (*metis deadlock-free-imp-deadlock-free-Renaming deadlock-free-implies-lifelock-free*
*lifelock-free-is-non-terminating* $\mathit{non\text{-}terminating\text{-}Seq}_{ptick}$)

The next lemma is of course more interesting.

**lemma** $\mathit{deadlock\text{-}free}_{SKIPS}$-$\mathit{Seq}_{ptick}$ :
‹$\mathit{deadlock\text{-}free}_{SKIPS}\ (P\ ;_{\checkmark}\ Q)$›
**if** *df-assms* : ‹$\mathit{deadlock\text{-}free}_{SKIPS}\ P$› ‹$\bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow \mathit{deadlock\text{-}free}_{SKIPS}\ (Q\ r)$›

**proof** (*unfold deadlock-free$_{SKIPS}$-is-right*, *intro ballI impI*)
**show** ‹$t \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q) \Longrightarrow tF\ t \Longrightarrow (t,\ UNIV) \notin \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$› **for** $t$
**proof** (*induct t rule*: *rev-induct*)
**from** *df-assms* **show** ‹$([],\ UNIV) \notin \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$›
**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** ‹$\mathcal{D}\ P = \{\}$›
**by** (*simp add*: *deadlock-free$_{SKIPS}$-implies-div-free*)
**fix** $t\ e$ **assume** *hyp* : ‹$t \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q) \Longrightarrow tF\ t \Longrightarrow (t,\ UNIV) \notin \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$›
**assume** ‹$t\ @\ [e] \in \mathcal{T}\ (P\ ;_{\checkmark}\ Q)$› ‹$tF\ (t\ @\ [e])$›
**then consider** $u\ v$ **where** ‹$t\ @\ [e] = map\ (ev \circ of\text{-}ev)\ u$› ‹$u \in \mathcal{T}\ P$› ‹$tF\ u$›
| $u\ r\ v$ **where** ‹$t\ @\ [e] = map\ (ev \circ of\text{-}ev)\ u\ @\ v$› ‹$u\ @\ [\checkmark(r)] \in \mathcal{T}\ P$› ‹$tF\ u$› ‹$v \in \mathcal{T}\ (Q\ r)$›
**by** (*auto simp add*: *Seq$_{ptick}$-projs* ‹$\mathcal{D}\ P = \{\}$›)
**thus** ‹$(t\ @\ [e],\ UNIV) \notin \mathcal{F}\ (P\ ;_{\checkmark}\ Q)$›
**by** (*cases*; *simp-all add*: *Seq$_{ptick}$-projs ref-Seq$_{ptick}$-UNIV*)
(*metis* (*no-types*) *F-T* ‹$\mathcal{D}\ P = \{\}$› ‹$tF\ (t\ @\ [e])$› *deadlock-free$_{SKIPS}$-is-right empty-iff strict-ticks-of-memI that tickFree-append-iff*)+
**qed**
**qed**

**corollary** *deadlock-free-Seq$_{ptick}$* :
‹$[\![deadlock\text{-}free_{SKIPS}\ P;\ \bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow deadlock\text{-}free\ (Q\ r)]\!]$
$\Longrightarrow deadlock\text{-}free\ (P\ ;_{\checkmark}\ Q)$›
**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* :
‹$P = STOP \wedge Q \neq STOP \vee deadlock\text{-}free\ P \Longrightarrow$
$Q = STOP \wedge P \neq STOP \vee deadlock\text{-}free\ Q \Longrightarrow deadlock\text{-}free\ (P\ \Box\ Q)$›
**using** *deadlock-free-Det* **by** *auto*

**lemma** *deadlock-free-Mprefix-Sync$_{ptick}$-Mprefix* :
**assumes** *not-all-empty*: ‹$\neg\ A \subseteq S \vee \neg\ B \subseteq S \vee A \cap B \cap S \neq \{\}$›
**and** ‹$\bigwedge a.\ a \in A - S \Longrightarrow deadlock\text{-}free\ (P\ a\ [\![S]\!]_{\checkmark}\ \Box b{\in}B \rightarrow Q\ b)$›
**and** ‹$\bigwedge b.\ b \in B - S \Longrightarrow deadlock\text{-}free\ (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ Q\ b)$›
**and** ‹$\bigwedge x.\ x \in A \cap B \cap S \Longrightarrow deadlock\text{-}free\ (P\ x\ [\![S]\!]_{\checkmark}\ Q\ x)$›
**shows** ‹$deadlock\text{-}free\ (\Box a{\in}A \rightarrow P\ a\ [\![S]\!]_{\checkmark}\ \Box b \in B \rightarrow Q\ b)$›
**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* :
‹⟦$A \subseteq S$; $B \subseteq S$; $A \cap B \neq \{\}$;
$\bigwedge x.\ x \in A \cap B \cap S \Longrightarrow$ *deadlock-free* (*P x* ⟦*S*⟧$_{\checkmark}$ *Q x*)⟧
$\Longrightarrow$ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ □*b*∈*B* → *Q b*)›
**and** *deadlock-free-Mprefix-Sync*$_{ptick}$*-Mprefix-indep* :
‹⟦$A \cap S = \{\}$; $B \cap S = \{\}$; $A \neq \{\} \vee B \neq \{\}$;
$\bigwedge a.\ a \in A - S \Longrightarrow$ *deadlock-free* (*P a* ⟦*S*⟧$_{\checkmark}$ □*b*∈*B* → *Q b*);
$\bigwedge b.\ b \in B - S \Longrightarrow$ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *Q b*)⟧
$\Longrightarrow$ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ □*b*∈*B* → *Q b*)›
**and** *deadlock-free-Mprefix-Sync*$_{ptick}$*-Mprefix-right* :
‹⟦$A \subseteq S$; $B \cap S = \{\}$; $B \neq \{\}$;
$\bigwedge b.\ b \in B - S \Longrightarrow$ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ *Q b*)⟧
$\Longrightarrow$ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ □*b*∈*B* → *Q b*)›
**and** *deadlock-free-Mprefix-Sync*$_{ptick}$*-Mprefix-left* :
‹⟦$A \cap S = \{\}$; $B \subseteq S$; $A \neq \{\}$;
$\bigwedge a.\ a \in A - S \Longrightarrow$ *deadlock-free* (*P a* ⟦*S*⟧$_{\checkmark}$ □*b*∈*B* → *Q b*)⟧
$\Longrightarrow$ *deadlock-free* (□*a*∈*A* → *P a* ⟦*S*⟧$_{\checkmark}$ □*b*∈*B* → *Q b*)›
**by** (*auto intro*!: *deadlock-free-Mprefix-Sync*$_{ptick}$*-Mprefix*)

**end**

## 17.2 Renaming and reference Processes

**lemma** *DF-empty* [*simp*] : ‹*DF* {} = *STOP*›
**and** *DF*$_{SKIPS}$*-empty* [*simp*] : ‹*DF*$_{SKIPS}$ {} {} = *STOP*›
**and** *RUN-empty* [*simp*] : ‹*RUN* {} = *STOP*›
**and** *CHAOS-empty* [*simp*] : ‹*CHAOS* {} = *STOP*›
**and** *CHAOS*$_{SKIPS}$*-empty* [*simp*] : ‹*CHAOS*$_{SKIPS}$ {} {} = *STOP*›
**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*) ⊑$_{FD}$ *Renaming* (*DF A*) *f g*, but the other refinement is requiring *finitary* assumptions (⟦*finitary f*; *finitary g*⟧ ⟹ *Renaming* (*DF A*) *f g* ⊑$_{FD}$ *DF* (*f* ‘ *A*)).

**lemma** *DF-restriction-fix-def* : ‹*DF A* = (*ν X*. □*a* ∈ *A* → *X*)›
**unfolding** *DF-def* **by** (*rule restriction-fix-is-fix*[*symmetric*]) *simp-all*

**lemma** *DF*$_{SKIPS}$*-restriction-fix-def* : ‹*DF*$_{SKIPS}$ *A R* = (*ν X*. (□*a* ∈ *A* → *X*) □ *SKIPS R*)›
**unfolding** *DF*$_{SKIPS}$*-def* **by** (*rule restriction-fix-is-fix*[*symmetric*]) *simp-all*

**lemma** *RUN-restriction-fix-def* : ‹*RUN A* = (*υ X*. □*a* ∈ *A* → *X*)›
**unfolding** *RUN-def* **by** (*rule restriction-fix-is-fix*[*symmetric*]) *simp-all*

**lemma** *CHAOS-restriction-fix-def* : ‹*CHAOS A* = (*υ X*. *STOP* ⊓ (□*a* ∈ *A* → *X*))›
**unfolding** *CHAOS-def* **by** (*rule restriction-fix-is-fix*[*symmetric*]) *simp-all*

**lemma** *CHAOS*$_{SKIPS}$*-restriction-fix-def* : ‹*CHAOS*$_{SKIPS}$ *A R* = (*υ X*. *SKIPS R* ⊓ *STOP* ⊓ (□*a* ∈ *A* → *X*))›
**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* : ‹*Renaming* (*DF A*) *f g* = *DF* (*f* ‘ *A*)›
**proof** (*unfold DF-restriction-fix-def*, *induct rule*: *parallel-restriction-fix-ind*)
**show** ‹*Renaming STOP f g* = *STOP*› **by** *simp*
**qed** (*auto simp add*: *Renaming-Mndetprefix intro*!: *mono-Mndetprefix-eq*)

**lemma** *Renaming-DF*$_{SKIPS}$ : ‹*Renaming* (*DF*$_{SKIPS}$ *A R*) *f g* = *DF*$_{SKIPS}$ (*f* ‘ *A*) (*g* ‘ *R*)›
**proof** (*unfold DF*$_{SKIPS}$*-restriction-fix-def*, *induct rule*: *parallel-restriction-fix-ind*)
**show** ‹*Renaming STOP f g* = *STOP*› **by** *simp*
**qed** (*auto simp add*: *Renaming-Mndetprefix Renaming-Ndet*
*intro*!: *mono-Mndetprefix-eq arg-cong2*[**where** *f* = ‹(⊓)›])

**lemma** *Renaming-RUN* : ‹*Renaming* (*RUN A*) *f g* = *RUN* (*f* ‘ *A*)›
**proof** (*unfold RUN-restriction-fix-def*, *induct rule*: *parallel-restriction-fix-ind*)
**show** ‹*Renaming STOP f g* = *STOP*› **by** *simp*
**qed** (*auto simp add*: *Renaming-Mprefix intro*!: *mono-Mprefix-eq*)

**lemma** *Renaming-CHAOS* : ‹*Renaming* (*CHAOS A*) *f g* = *CHAOS* (*f* ‘ *A*)›
**proof** (*unfold CHAOS-restriction-fix-def*, *induct rule*: *parallel-restriction-fix-ind*)
**show** ‹*Renaming STOP f g* = *STOP*› **by** *simp*
**qed** (*auto simp add*: *Renaming-Mprefix Renaming-Ndet*
*intro*!: *mono-Mprefix-eq arg-cong2*[**where** *f* = ‹(⊓)›])

**lemma** *Renaming-CHAOS*$_{SKIPS}$ : ‹*Renaming* (*CHAOS*$_{SKIPS}$ *A R*) *f g* = *CHAOS*$_{SKIPS}$ (*f* ‘ *A*) (*g* ‘ *R*)›
**proof** (*unfold CHAOS*$_{SKIPS}$*-restriction-fix-def*, *induct rule*: *parallel-restriction-fix-ind*)
**show** ‹*Renaming STOP f g* = *STOP*› **by** *simp*
**qed** (*auto simp add*: *Renaming-Mprefix Renaming-Ndet*
*intro*!: *mono-Mprefix-eq arg-cong2*[**where** *f* = ‹(⊓)›])

## 17.3 Data Independence

When working with the new interleaving $P \ [\![\{\}]\!]_{\checkmark} \ Q$, we intuitively expect it to be *deadlock-free* when both $P$ and $Q$ are. The purpose of this section

is to prove it.

### 17.3.1 An interesting equivalence

**lemma** (**in** $Sync_{ptick}$-*locale*) *deadlock-free-of-*$Sync_{ptick}$*-iff-DF-FD-DF-*$Sync_{ptick}$*-DF*:
‹(∀ *P Q*. *deadlock-free P* ⟶ *deadlock-free Q* ⟶ *deadlock-free* (*P* ⟦*S*⟧✓ *Q*))
⟷ *DF UNIV* ⊑$_{FD}$ (*DF UNIV* ⟦*S*⟧✓ *DF UNIV*)› (**is** ‹*?lhs* ⟷ *?rhs*›)
**proof** (*rule iffI*)
**assume** *?lhs*
**show** *?rhs* **by** (*fold deadlock-free-def*, *rule* ‹*?lhs*›[*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 $r = s \Longrightarrow (DF\ A \sqsubseteq_{FD} DF\ A\ [\![S]\!]\ SKIP\ r) = (A \cap S = \{\})$.

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

**lemma** (**in** $Sync_{ptick}$-*locale*) *DF-FD-DF-*$Sync_{ptick}$*-SKIPS-imp-disjoint* :
‹*A* ∩ *S* = {}› **if** ‹*DF A* ⊑$_{FD}$ *DF A* ⟦*S*⟧✓ *SKIPS R*›
**proof** (*rule ccontr*)
**assume** ‹*A* ∩ *S* ≠ {}›
**then obtain** *a* **where** ‹*a* ∈ *A*› **and** ‹*a* ∈ *S*› **by** *blast*
**have** ‹*DF A* ⟦*S*⟧✓ *SKIPS R* ⊑$_{FD}$ *DF* {*a*} ⟦*S*⟧✓ *SKIPS R*›
**by** (*intro mono-*$Sync_{ptick}$*-FD*[*OF - idem-FD*]) (*simp add*: *DF-subset* ‹*a* ∈ *A*›)
**also have** ‹... = *STOP*›
**by** (*subst DF-unfold*)
(*simp add*: ‹*a* ∈ *S*› *SKIPS-def* $Sync_{ptick}$*-distrib-GlobalNdet-left*
*write0-*$Sync_{ptick}$*-STOP write0-*$Sync_{ptick}$*-SKIP*)
**finally show** *False*
**by** (*metis that* ‹*a* ∈ *A*› *DF-Univ-freeness empty-iff non-deadlock-free-STOP trans-FD*)
**qed**

**lemma** *disjoint-imp-DF-eq-DF-*$Sync_{ptick}$*-SKIPS* :
‹*DF A* = *DF A* ⟦*S*⟧✓ *SKIPS R*› **if** ‹*A* ∩ *S* = {}›
**proof** (*subst DF-restriction-fix-def*, *induct rule*: *restriction-fix-ind*)
**show** ‹*X* = *DF A* ⟦*S*⟧✓ *SKIPS R* ⟹ ⊓*a*∈*A* → *X* = *DF A* ⟦*S*⟧✓ *SKIPS R*› **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* :
$\langle DF\ A \sqsubseteq_{FD} DF\ A\ [\![S]\!]_{\checkmark}\ STOP \Longrightarrow A \cap S = \{\}\rangle$
**and** *DF-FD-DF-Sync$_{ptick}$-SKIP-imp-disjoint* :
$\langle DF\ A \sqsubseteq_{FD} DF\ A\ [\![S]\!]_{\checkmark}\ SKIP\ r \Longrightarrow A \cap S = \{\}\rangle$
**and** *disjoint-imp-DF-eq-DF-Sync$_{ptick}$-STOP* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = DF\ A\ [\![S]\!]_{\checkmark}\ STOP\rangle$
**and** *disjoint-imp-DF-eq-DF-Sync$_{ptick}$-SKIP* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = DF\ A\ [\![S]\!]_{\checkmark}\ SKIP\ r\rangle$
**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* :
$\langle DF\ A \sqsubseteq_{FD} SKIPS\ R\ [\![S]\!]_{\checkmark}\ DF\ A \Longrightarrow A \cap S = \{\}\rangle$
**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* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = SKIPS\ R\ [\![S]\!]_{\checkmark}\ DF\ A\rangle$
**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* :
$\langle DF\ A \sqsubseteq_{FD} STOP\ [\![S]\!]_{\checkmark}\ DF\ A \Longrightarrow A \cap S = \{\}\rangle$
**and** *DF-FD-SKIP-Sync$_{ptick}$-DF-imp-disjoint* :
$\langle DF\ A \sqsubseteq_{FD} SKIP\ r\ [\![S]\!]_{\checkmark}\ DF\ A \Longrightarrow A \cap S = \{\}\rangle$
**and** *disjoint-imp-DF-eq-STOP-Sync$_{ptick}$-DF* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = STOP\ [\![S]\!]_{\checkmark}\ DF\ A\rangle$
**and** *disjoint-imp-DF-eq-SKIP-Sync$_{ptick}$-DF* :
$\langle A \cap S = \{\} \Longrightarrow DF\ A = SKIP\ r\ [\![S]\!]_{\checkmark}\ DF\ A\rangle$
**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* $(P \mid\mid\mid Q)$

**theorem** (**in** *Sync$_{ptick}$-locale*) *DF-F-DF-Sync$_{ptick}$-DF-weak* : $\langle DF\ (A \cup B) \sqsubseteq_F DF\ A\ [\![S]\!]_{\checkmark}\ DF\ B\rangle$
**if** *nonempty*: $\langle A \neq \{\}\rangle$ $\langle B \neq \{\}\rangle$
**and** *intersect-hyp*: $\langle B \cap S = \{\} \vee (\exists\, y.\ B \cap S = \{y\} \wedge A \cap S \subseteq \{y\})\rangle$

**proof** −
**have** ‹⟦(*u*, *X*) ∈ ℱ (*DF A*); (*v*, *Y*) ∈ ℱ (*DF B*); *t setinterleaves*$_{✓(⊗✓)}$ ((*u*, *v*), *S*)⟧
⟹ (*t*, *super-ref-Sync*$_{ptick}$ (⊗✓) *X S Y*) ∈ ℱ (*DF* (*A* ∪ *B*))› **for** *v t u X Y*
**proof** (*induct t arbitrary*: *u v*)
**case** *Nil*
**from** *Nil.prems*(*3*) **have** ‹*u* = []› ‹*v* = []› **by** (*simp-all add*: *Nil-setinterleaves*$_{ptick}$)
**from** *Nil.prems*(*1*) **obtain** *a* **where** ‹*a* ∈ *A*› ‹*ev a* ∉ *X*›
**by** (*subst* (*asm*) *F-DF*) (*auto simp add*: *nonempty* ‹*u* = []›)
**moreover from** *Nil.prems*(*2*) **obtain** *b* **where** ‹*b* ∈ *B*› ‹*ev b* ∉ *Y*›
**by** (*subst* (*asm*) *F-DF*) (*auto simp add*: *nonempty* ‹*v* = []›)
**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.prems*(*3*) **consider** (*mv-left*) *a u*′ **where** ‹*a* ∉ *S*› ‹*e* = *ev a*› ‹*u* = *ev a* # *u*′›
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*u*′, *v*), *S*)›
| (*mv-right*) *b v*′ **where** ‹*b* ∉ *S*› ‹*e* = *ev b*› ‹*v* = *ev b* # *v*′›
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*u*, *v*′), *S*)›
| (*mv-both-ev*) *a u*′ *v*′ **where** ‹*a* ∈ *S*› ‹*e* = *ev a*› ‹*u* = *ev a* # *u*′› ‹*v* = *ev a* # *v*′›
‹*t setinterleaves*$_{✓(⊗✓)}$ ((*u*′, *v*′), *S*)›
| (*mv-both-tick*) *r s r-s u*′ *v*′ **where** ‹*r* ⊗✓ *s* = *Some r-s*› ‹*e* = ✓(*r-s*)›
‹*u* = ✓(*r*) # *u*′› ‹*v* = ✓(*s*) # *v*′› ‹*t setinterleaves*$_{✓(⊗✓)}$ ((*u*′, *v*′), *S*)›
**by** (*cases e*) (*auto elim*: *Cons-ev-setinterleaves*$_{ptick}$*E Cons-tick-setinterleaves*$_{ptick}$*E*)
**thus** *?case*
**proof** *cases*
**case** *mv-left*
**from** *Cons.prems*(*1*) **have** ‹*a* ∈ *A*›
**by** (*subst* (*asm*) *F-DF*) (*simp add*: *mv-left*(*3*) *split*: *if-split-asm*)
**from** *Cons.prems*(*1*)[*unfolded mv-left*(*3*), *THEN Cons-F-DF*] **have** ‹(*u*′, *X*) ∈ ℱ (*DF A*)› **.**
**from** *Cons.hyps*[*OF this Cons.prems*(*2*) *mv-left*(*4*)] **show** *?thesis*
**by** (*subst F-DF*) (*simp add*: *nonempty* ‹*e* = *ev a*› ‹*a* ∈ *A*›)
**next**
**case** *mv-right*
**from** *Cons.prems*(*2*) **have** ‹*b* ∈ *B*›
**by** (*subst* (*asm*) *F-DF*) (*simp add*: *mv-right*(*3*) *split*: *if-split-asm*)
**from** *Cons.prems*(*2*)[*unfolded mv-right*(*3*), *THEN Cons-F-DF*] **have** ‹(*v*′, *Y*) ∈ ℱ (*DF B*)› **.**
**from** *Cons.hyps*[*OF Cons.prems*(*1*) *this mv-right*(*4*)] **show** *?thesis*
**by** (*subst F-DF*) (*simp add*: *nonempty* ‹*e* = *ev b*› ‹*b* ∈ *B*›)
**next**
**case** *mv-both-ev*
**from** *Cons.prems*(*1*) **have** ‹*a* ∈ *A*›

**by** (*subst* (*asm*) *F-DF*) (*simp add*: *mv-both-ev*(*3*) *split*: *if-split-asm*)
**from** *Cons.prems*(*1*)[*unfolded mv-both-ev*(*3*), *THEN Cons-F-DF*]
*Cons.prems*(*2*)[*unfolded mv-both-ev*(*4*), *THEN Cons-F-DF*]
**have** ‹(*u'*, *X*) ∈ $\mathcal{F}$ (*DF A*)› ‹(*v'*, *Y*) ∈ $\mathcal{F}$ (*DF B*)› **.**
**from** *Cons.hyps*[*OF this mv-both-ev*(*5*)] **show** *?thesis*
**by** (*subst F-DF*) (*simp add*: *nonempty* ‹*e* = *ev a*› ‹*a* ∈ *A*›)
**next**
**case** *mv-both-tick*
**from** *Cons.prems*(*1*) **have** *False*
**by** (*subst* (*asm*) *F-DF*) (*simp add*: *mv-both-tick*(*3*) *split*: *if-split-asm*)
**thus** *?thesis* **..**
**qed**
**qed**
**thus** ‹*DF* (*A* ∪ *B*) $\sqsubseteq_F$ *DF A* $[\![S]\!]_{\checkmark}$ *DF B*›
**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* :
‹*DF* (*A* ∪ *B*) $\sqsubseteq_F$ *DF A* $[\![S]\!]_{\checkmark}$ *DF B*› **if** ‹*A* ≠ {}› ‹*B* ≠ {}›
**and** ‹*A* ∩ *S* = {} ∨ (∃ *a*. *A* ∩ *S* = {*a*} ∧ *B* ∩ *S* ⊆ {*a*}) ∨
*B* ∩ *S* = {} ∨ (∃ *b*. *B* ∩ *S* = {*b*} ∧ *A* ∩ *S* ⊆ {*b*})›
**proof** −
**from** *that*(*3*) **consider** ‹*A* ∩ *S* = {} ∨ (∃ *a*. *A* ∩ *S* = {*a*} ∧ *B* ∩ *S* ⊆ {*a*})›
| ‹*B* ∩ *S* = {} ∨ (∃ *b*. *B* ∩ *S* = {*b*} ∧ *A* ∩ *S* ⊆ {*b*})› **by** *metis*
**thus** ‹*DF* (*A* ∪ *B*) $\sqsubseteq_F$ *DF A* $[\![S]\!]_{\checkmark}$ *DF B*›
**proof** *cases*
**from** *that*(*1*, *2*) **show** ‹*B* ∩ *S* = {} ∨ (∃ *b*. *B* ∩ *S* = {*b*} ∧ *A* ∩ *S* ⊆ {*b*}) ⟹
*DF* (*A* ∪ *B*) $\sqsubseteq_F$ *DF A* $[\![S]\!]_{\checkmark}$ *DF B*›
**by** (*rule DF-F-DF-Sync*$_{ptick}$-*DF-weak*)
**next**
**from** *that*(*1*, *2*) **show** ‹*A* ∩ *S* = {} ∨ (∃ *a*. *A* ∩ *S* = {*a*} ∧ *B* ∩ *S* ⊆ {*a*}) ⟹
*DF* (*A* ∪ *B*) $\sqsubseteq_F$ *DF A* $[\![S]\!]_{\checkmark}$ *DF B*›
**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* :
‹*DF* (*A* ∪ *B*) $\sqsubseteq_{FD}$ *DF A* $[\![S]\!]_{\checkmark}$ *DF B*› **if** ‹*A* ≠ {}› ‹*B* ≠ {}›
**and** ‹*A* ∩ *S* = {} ∨ (∃ *a*. *A* ∩ *S* = {*a*} ∧ *B* ∩ *S* ⊆ {*a*}) ∨
*B* ∩ *S* = {} ∨ (∃ *b*. *B* ∩ *S* = {*b*} ∧ *A* ∩ *S* ⊆ {*b*})›
**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*:

```
  ‹DF (A ∪ B) ⊑_FD DF A ⟦S⟧✓ DF B ⟷
   (     if A = {} then B ∩ S = {}
    else if B = {} then A ∩ S = {}
    else A ∩ S = {} ∨ (∃ a. A ∩ S = {a} ∧ B ∩ S ⊆ {a}) ∨
         B ∩ S = {} ∨ (∃ b. B ∩ S = {b} ∧ A ∩ S ⊆ {b}))›
  (is ‹?FD-ref ⟷ (     if A = {} then B ∩ S = {}
                  else if B = {} then A ∩ S = {}
                  else ?cases)›)
proof −
  { assume ‹A ≠ {}› and ‹B ≠ {}› and ?FD-ref and ‹¬ ?cases›
    from ‹¬ ?cases›[simplified]
    obtain a and b where ‹a ∈ A› ‹a ∈ S› ‹b ∈ B› ‹b ∈ S› ‹a ≠ b› by blast
    have ‹DF A ⟦S⟧✓ DF B ⊑_FD (a → DF A) ⟦S⟧✓ (b → DF B)›
     by (intro mono-Sync_ptick-FD; subst DF-unfold, meson Mndetprefix-FD-write0
‹a ∈ A› ‹b ∈ B›)
   also have ‹... = STOP› by (simp add: ‹a ∈ S› ‹a ≠ b› ‹b ∈ S› write0-Sync_ptick-write0-subset)
    finally have False
      by (metis DF-Univ-freeness Un-empty ‹A ≠ {}›
          trans-FD[OF ‹?FD-ref›] non-deadlock-free-STOP)
  } note ∗ = this
  show ?thesis
  proof (cases ‹A = {}›; cases ‹B = {}›)
    show ‹A = {} ⟹ B = {} ⟹ ?thesis› by simp
  next
    show ‹A = {} ⟹ B ≠ {} ⟹ ?thesis›
      by simp (metis DF-FD-STOP-Sync_ptick-DF-imp-disjoint
          disjoint-imp-DF-eq-STOP-Sync_ptick-DF order-refl)
  next
    show ‹A ≠ {} ⟹ B = {} ⟹ ?thesis›
      by simp (metis DF-FD-DF-Sync_ptick-STOP-imp-disjoint
          disjoint-imp-DF-eq-DF-Sync_ptick-STOP order-refl)
  next
    show ‹A ≠ {} ⟹ B ≠ {} ⟹ ?thesis›
      by simp (metis ∗ DF-FD-DF-Sync_ptick-DF)
  qed
qed



lemma DF-FD-DF-MultiSync_ptick-DF :
  ‹⟦⋀l. l ∈ set L ⟹ X l ≠ {}; ∃ s. (⋃ l ∈ set L. X l) ∩ S ⊆ {s}⟧
  ⟹ DF (⋃ l ∈ set L. X l) ⊑_FD ⟦S⟧✓ l ∈@ L. (DF (X l) :: ('a, 'r) process_ptick)›
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 ‹(DF (⋃ l ∈ set (l0 # l1 # L). X l) :: ('a, 'r list) process_ptick) =
```

*DF* (*X l0* ∪ (⋃ *l* ∈ *set* (*l1* # *L*). *X l*))› **by** *simp*
**also have** ‹... ⊑$_{FD}$ *DF* (*X l0*) ⟦*S*⟧$_{✓Rlist}$ *DF* (⋃ *l* ∈ *set* (*l1* # *L*). *X l*)›
**by** (*rule Sync*$_{Rlist}$*.DF-FD-DF-Sync*$_{ptick}$*-DF-iff*[*THEN iffD2*])
(*use 3.prems*(2) **in** ‹*simp add*: *3.prems*(1) *subset-singleton-iff*
*Int-Un-distrib2 Un-singleton-iff*, *safe*, *simp-all*›)
**also have** ‹... ⊑$_{FD}$ *DF* (*X l0*) ⟦*S*⟧$_{✓Rlist}$ ⟦*S*⟧$_{✓}$ *l*∈@(*l1* # *L*). (*DF* (*X l*))›
**by** (*intro Sync*$_{Rlist}$*.mono-Sync*$_{ptick}$*-FD*[*OF idem-FD*] *3.hyps*(2))
(*use 3.prems* **in** *auto*)
**also have** ‹... = ⟦*S*⟧$_{✓}$ *l*∈@(*l0* # *l1* # *L*). *DF* (*X l*)› **by** *simp*
**finally show** *?case* **.**
**qed**

**lemma** (**in** *Sync*$_{ptick}$*-locale*) ‹*DF* {*a*} = *DF* {*a*} ⟦*S*⟧$_{✓}$ *STOP* ⟷ *a* ∉ *S*›
**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*) ‹*DF* {*a*} ⟦*S*⟧$_{✓}$ *STOP* = *STOP* ⟷ *a* ∈ *S*›
**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* : ‹*DF A* ⊑$_{FD}$ *DF A* |||$_{✓}$ *DF A*›
**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*:
‹*DF UNIV* ⊑$_{FD}$ *DF UNIV* |||$_{✓}$ *DF UNIV*›
**by** (*fact DF-FD-DF-Inter*$_{ptick}$*-DF*)

**corollary** (**in** *Sync*$_{ptick}$*-locale*) *Inter*$_{ptick}$*-deadlock-free* :
‹*deadlock-free P* ⟹ *deadlock-free Q* ⟹ *deadlock-free* (*P* |||$_{✓}$ *Q*)›
**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* :
‹⟦*L* ≠ []; ⋀*l*. *l* ∈ *set L* ⟹ *deadlock-free* (*P l*)⟧ ⟹
*deadlock-free* (|||$_{✓}$ *l* ∈@ *L*. *P l*)›
**proof** (*induct L rule*: *induct-list012*)
**case** *1*
**from** *1.prems*(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** ‹$|||_{\checkmark}$ *l* ∈@ (*l0* # *l1* # *L*). *P l* = *P l0* $|||_{\checkmark Rlist}$ $|||_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l*›
**by** *simp*
**moreover have** ‹*deadlock-free* (*P l0*)› **by** (*simp add*: *3.prems(2)*)
**moreover have** ‹*deadlock-free* ($|||_{\checkmark}$ *l* ∈@ (*l1* # *L*). *P l*)›
**by** (*rule 3.hyps(2)*) (*simp-all add*: *3.prems(2)*)
**ultimately show** *?case*
**by** (*simp add*: $Sync_{Rlist}$.$Inter_{ptick}$-*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 $('a, 'r)\ process_{ptick} \Rightarrow ('r \Rightarrow ('a, 's)\ process_{ptick}) \Rightarrow ('a, 's)\ process_{ptick}$, so that the process on the right-hand side is now parameterized with the value returned by the process on the left-hand side. The main motivation for this generalization was to have *SKIP* as neutral element. This is now the case.

$$P \mathbin{;\!\!\checkmark} SKIP = P \qquad SKIP\ r \mathbin{;\!\!\checkmark} Q = Q\ r$$

Additionally, with the following associativity property :

$$P \mathbin{;\!\!\checkmark} (\lambda r.\ Q\ r \mathbin{;\!\!\checkmark} R) = P \mathbin{;\!\!\checkmark} Q \mathbin{;\!\!\checkmark} R$$

we can conclude that this generalized sequential composition fulfills the monad laws.
Unsurprisingly, the correspondence with classical version is very intuitive.

$$P \mathbin{;\!\!\checkmark} (\lambda r.\ Q) = P \mathbin{;} Q$$

The expected step law has also been established.

$$\Box a{\in}A \rightarrow P\ a \mathbin{;\!\!\checkmark} Q = \Box a{\in}A \rightarrow (P\ a \mathbin{;\!\!\checkmark} Q)$$

Additionally, in the same way as described in [4], operational laws have been derived.

$$\frac{P\ {}_{\alpha}{\rightsquigarrow}_{\tau}\ P'}{P \mathbin{;\!\!\checkmark} Q\ {}_{\beta}{\rightsquigarrow}_{\tau}\ P' \mathbin{;\!\!\checkmark} Q} \qquad \frac{a\ {}_{\alpha}{\rightsquigarrow}_{P}\ P'}{a \mathbin{;\!\!\checkmark} Q\ {}_{\beta}{\rightsquigarrow}_{P}\ P' \mathbin{;\!\!\checkmark} Q}$$

$$\frac{r\ {}_{\alpha}{\rightsquigarrow}\checkmark_{P}\ P' \qquad Q\ P\ {}_{\beta}{\rightsquigarrow}_{\tau}\ Q'}{r \mathbin{;\!\!\checkmark} Q\ {}_{\beta}{\rightsquigarrow}_{\tau}\ Q'}$$

The continuity has only be obtained under a kind of finiteness assumption, but non-destructiveness holds in general.
Finally, an architectural version is defined. It satisfies the following property.

$$SEQ\checkmark\ l \in@\ (L1\ @\ L2).\ P\ l = (\lambda r.\ (SEQ\checkmark\ l \in@\ L1.\ P\ l)\ r \mathbin{;\!\!\checkmark} SEQ\checkmark\ l \in@\ L2.\ P\ l)$$

### 18.2.3 Synchronization Product

The main motivation for generalizing the synchronization product was to have a satisfying handling of the synchronization of two terminations. Indeed, with the *Sync* operator inherited from `HOL-CSP`, the returned values were lost (most of the time).

$$SKIP\ r\ [\![A]\!]\ SKIP\ s = (\mathsf{if}\ r = s\ \mathsf{then}\ SKIP\ r\ \mathsf{else}\ STOP)$$

With the new definition, this is not the case anymore.

$$SKIP\ r\ [\![A]\!]_{\checkmark}\ SKIP\ s = (\mathsf{case}\ r\ \otimes\checkmark\ s\ \mathsf{of}\ None \Rightarrow STOP \mid Some\ r\text{-}s \Rightarrow SKIP\ r\text{-}s)$$

This law is directly extracted from the core of the construction, which is done in a very abstract way through a **locale** specification. The operator is then declined in several variations, leading to the following rules.

$$SKIP\ r\ [\![A]\!]_{\checkmark Pair}\ SKIP\ s = SKIP\ (r,\ s)$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Pairlist}\ SKIP\ s = SKIP\ [r,\ s]$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Rlist}\ SKIP\ s = SKIP\ (r \cdot s)$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Llist}\ SKIP\ s = SKIP\ (r\ @\ [s])$$
$$SKIP\ r\ {}_n[\![A]\!]_{\checkmark ListslenL}\ SKIP\ s = (\mathsf{if}\ |r| = n\ \mathsf{then}\ SKIP\ (r\ @\ s)\ \mathsf{else}\ STOP)$$
$$SKIP\ r\ {}_n[\![A]\!]_{\checkmark ListslenR}\ SKIP\ s = (\mathsf{if}\ |s| = n\ \mathsf{then}\ SKIP\ (r\ @\ s)\ \mathsf{else}\ STOP)$$
$$SKIP\ r\ {}_n[\![A]\!]_{\checkmark m}\ SKIP\ s = (\mathsf{if}\ |r| = n \wedge |s| = m\ \mathsf{then}\ SKIP\ (r\ @\ s)\ \mathsf{else}\ STOP)$$
$$SKIP\ r\ [\![A]\!]_{\checkmark Classic}\ SKIP\ s = (\mathsf{if}\ r = s\ \mathsf{then}\ SKIP\ r\ \mathsf{else}\ STOP)$$

Moreover, the last declension is proved to be equal to the old version, ensuring that this work is actually a generalization.

$$P\ [\![A]\!]_{\checkmark Classic}\ Q = P\ [\![A]\!]\ Q$$

We also established commutativity and associativity, modulo renaming the ticks. The underlying abstract setup is quite obscure, so we will only display here the pair versions.

$$RenamingTick\ (P\ [\![A]\!]_{\checkmark Pair}\ Q)\ prod.swap = Q\ [\![A]\!]_{\checkmark Pair}\ P$$
$$P\ [\![A]\!]_{\checkmark Pair}\ (Q\ [\![A]\!]_{\checkmark Pair}\ R) =$$
$$RenamingTick\ (P\ [\![A]\!]_{\checkmark Pair}\ Q\ [\![A]\!]_{\checkmark Pair}\ R)\ (\lambda((r,\ s),\ t).\ (r,\ s,\ t))$$

Again, the expected step law has been established.

$$\Box a{\in}A \rightarrow P\ a\ ;\!\!\checkmark\ Q = \Box a{\in}A \rightarrow (P\ a\ ;\!\!\checkmark\ Q)$$

In this abstract setup, the operational laws have also been derived.

$$\frac{P\ {}_{lhs}\!\!\rightsquigarrow_{\tau}\ P'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ P'\ [\![A]\!]_{\checkmark}\ Q} \qquad \frac{Q\ {}_{rhs}\!\!\rightsquigarrow_{\tau}\ Q'}{A\ [\![P]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ A\ [\![P]\!]_{\checkmark}\ Q'}$$

$$\frac{a \notin A \qquad P\ {}_{lhs}\!\!\rightsquigarrow_{a}\ P'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{a}\ P'\ [\![A]\!]_{\checkmark}\ Q} \qquad \frac{a \notin A \qquad Q\ {}_{rhs}\!\!\rightsquigarrow_{a}\ Q'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{a}\ P\ [\![A]\!]_{\checkmark}\ Q'}$$

$$\frac{a \in A \qquad P\ {}_{lhs}\!\!\rightsquigarrow_{a}\ P' \qquad Q\ {}_{rhs}\!\!\rightsquigarrow_{a}\ Q'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{a}\ P'\ [\![A]\!]_{\checkmark}\ Q'}$$

$$\frac{P\ {}_{lhs}\!\!\rightsquigarrow_{\checkmark r}\ P'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ SKIP\ r\ [\![A]\!]_{\checkmark}\ Q}$$

$$\frac{Q\ {}_{rhs}\!\!\rightsquigarrow_{\checkmark s}\ Q'}{P\ [\![A]\!]_{\checkmark}\ Q\ {}_{ptick}\!\!\rightsquigarrow_{\tau}\ P\ [\![A]\!]_{\checkmark}\ SKIP\ s}$$

$$\frac{r \otimes\!\checkmark\ s = Some\ r\text{-}s}{SKIP\ r\ [\![A]\!]_{\checkmark}\ SKIP\ s\ {}_{ptick}\!\!\rightsquigarrow_{\checkmark r\text{-}s}\ \Omega_{ptick}\ (SKIP\ r\text{-}s)\ r\text{-}s}$$

Continuity and non-destructiveness hold in general, and an architectural version is defined. It satisfies the following property.

$$\frac{L1 \neq [] \qquad L2 \neq []}{[\![S]\!]_{\checkmark}\ l \in@\ (L1\ @\ L2).\ P\ l = [\![S]\!]_{\checkmark}\ l \in@\ L1.\ P\ l\ {}_{|L1|}[\![S]\!]_{\checkmark|L2|}\ [\![S]\!]_{\checkmark}\ l \in@\ L2.\ P\ l}$$

It is defined on a list (while its counterpart *MultiSync* based on the *Sync* operator is defined on a multiset) because the order of appearance of the ticks matters. However, as long as we keep track of the positions, we can permute the list. This is summarized by the following theorem.

$$\frac{f\ permutes\ \{..<|L|\}}{[\![S]\!]_{\checkmark}\ l \in@\ permute\text{-}list\ f\ L.\ P\ l = RenamingTick\ ([\![S]\!]_{\checkmark}\ l \in@\ L.\ P\ l)\ (permute\text{-}list\ f)}$$

# Bibliography


[1] B. Ballenghien, S. Taha, and B. Wolff. Hol-cspm - architectural operators for hol-csp. *Archive of Formal Proofs*, December 2023. https://isa-afp.org/entries/HOL-CSPM.html, Formal proof development.

[2] B. Ballenghien, S. Taha, B. Wolff, and L. Ye. Hol-csp version 2.0. *Archive of Formal Proofs*, April 2019. https://isa-afp.org/entries/HOL-CSP.html, Formal proof development.

[3] B. Ballenghien and B. Wolff. Operational semantics formally proven in hol-csp. *Archive of Formal Proofs*, December 2023. https://isa-afp.org/entries/HOL-CSP_OpSem.html, Formal proof development.

[4] B. Ballenghien and B. Wolff. An Operational Semantics in Isabelle/HOL-CSP. In Y. Bertot, T. Kutsia, and M. Norrish, editors, *15th International Conference on Interactive Theorem Proving (ITP 2024)*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] B. Ballenghien and B. Wolff. Csp semantics over restriction spaces. *Archive of Formal Proofs*, May 2025. https://isa-afp.org/entries/HOL-CSP_RS.html, Formal proof development.