

HOL-CSP_Proc-Omata: A Bridge between CSP Processes and Functional Automata

Benoît Ballenghien Burkhart Wolff

April 14, 2026

Abstract

This entry develops the Proc-Omata framework on top of **HOL-CSP** and its extensions. Proc-Omata are defined from functional automata and come in four variants: deterministic, terminating deterministic, non-deterministic, and terminating non-deterministic. This subclass of processes combines the expressiveness of CSP with automata-like structure (reachability, enableness), making it particularly amenable to invariant-based reasoning.

We lift sequential composition and synchronization product to the automata level through combination functions and prove compactification theorems that enable reasoning over large process architectures. An essential ingredient is the use of restriction spaces, which guarantees well-defined fixed points even in the non-deterministic setting. Finally, we illustrate the applicability of the framework with the Dining Philosophers, where compactification yields proofs that scale to an arbitrary finite but unbounded number of participants in this parameterized process architecture.

Contents

1	Introduction	9
2	An Excursion into Determinism	11
2.1	Accepts initials	11
2.1.1	Definition	11
2.1.2	First properties	12
2.1.3	Monotonicity	13
2.1.4	Behaviour on Operators	13
2.1.5	Characterizations with After	18
2.2	Deterministic process	22
2.2.1	Definition	22
2.2.2	Monotonicity	23
2.2.3	Characterization as Maximal	23
2.2.4	Characterization with After	29
2.2.5	Operators preserving Determinism	29
2.2.6	Operators not (always) preserving Determinism	34
2.3	Application to Operational Semantics	34
3	ProcOmata: Functional Automata embedded into CSP Processes	37
3.1	Definitions	39
3.1.1	Non-deterministic and deterministic Automata	39
3.1.2	Enableness	39
3.1.3	States allowing Termination	39
3.1.4	Reachability	40
3.1.5	Morphisms	40
3.1.6	Generic update Functions	42
3.1.7	Assumptions on Automata	42
3.2	First Properties	46
3.2.1	ε , ρ and ω first equalities	46
3.2.2	Properties of our morphisms	47
3.2.3	Reachability results (for \mathcal{R}_d and \mathcal{R}_{nd})	54
3.3	Normalization	56

3.3.1	Non-deterministic Case	56
3.3.2	Deterministic Case	59
3.3.3	Link between deterministic and non-deterministic ProcOmata	62
3.3.4	Prove Equality between ProcOmata	62
4	Advanced Properties of ProcOmata	67
5	Combining Automata for Synchronization Product	69
5.1	Definitions	69
5.1.1	General Patterns	69
5.1.2	Specializations	70
5.2	First Properties	72
5.3	Reachability	73
5.4	Normalization	75
6	Compactification of Synchronization Product	79
6.1	Iterated Combine	79
6.1.1	Definitions	79
6.1.2	First Results	80
6.1.3	Reachability	81
6.1.4	Transmission of Properties	82
6.1.5	Normalization	84
6.2	Compactification Theorems	86
6.2.1	Binary	86
6.2.2	Rlist	92
6.2.3	ListslenL	93
6.2.4	Multiple	95
6.3	Derived Versions	100
6.4	More on Iterated Combine	103
6.5	More on Events	107
7	Combining Automata for Generalized Synchronization Product	111
7.1	Definitions	111
7.1.1	Specializations	111
7.2	First Properties	112
7.3	Transitions are unchanged in the Generalization	114
7.4	Reachability	114
7.5	Normalization	115
8	Compactification of Synchronization Product Generalized	119
8.1	Iterated Combine	119
8.1.1	Definitions	119

8.1.2	First Results	119
8.1.3	Transmission of Properties	120
8.1.4	Normalization	122
8.2	Compactification Theorems	124
8.2.1	Binary	124
8.2.2	Rlist	130
8.2.3	ListslenL	132
8.2.4	Multiple	134
8.3	Derived Versions	136
8.4	More on Iterated Combine and Events	139
9	Combining Automata for Sequential Composition Generalized	141
9.1	Definitions	141
9.2	General Patterns	141
9.3	Specializations	142
9.4	First Properties	143
9.4.1	Reachability	144
9.5	Normalization	147
10	Compactification of Sequential Composition Generalized	151
10.1	Iterated Combine	151
10.1.1	Definitions	151
10.1.2	First Results	151
10.1.3	Reachability	152
10.1.4	Transmission of Properties	153
10.1.5	Normalization	153
10.2	Compactification Theorems	155
10.2.1	Binary	155
10.2.2	ListslenL	158
10.2.3	Multiple	159
11	Application : May Philosophers dine ?	163
11.1	Preliminaries	163
11.1.1	Preliminary lemmas for proof automation	163
11.2	The dining processes definition	163
11.2.1	Unfolding rules	164
11.3	Translation into normal form	164
11.3.1	<i>FORK</i> , <i>LPHILO</i> and <i>RPHIL</i> are normalizable	164
11.3.2	<i>FORKS</i> is normalizable	166
11.3.3	<i>PHILS</i> is normalizable	167
11.3.4	The complete process <i>DINING</i> is normalizable	167
11.4	And finally: Philosophers may dine ! Always !	168
11.4.1	Construction of an invariant for the dining automaton	168

11.4.2	The invariant <i>inv-dining</i> implies that <i>DINING</i> is <i>deadlock-free</i>	171
11.4.3	Conclusion	172
11.5	Alternative version with only right-handed philosophers (in order to show that it's not <i>deadlock-free</i>)	173
11.5.1	Setup	173
11.5.2	Normalization	173
11.5.3	Correspondance between our normalized processes and the previous definitions	173
11.5.4	Proof that we have a deadlock in the state (<i>replicate N 1, replicate N 1</i>)	174
11.5.5	Proof that this state is reachable from our initial state, i.e. (<i>replicate N 1, replicate N 1</i>) $\in \mathcal{R}_d$ <i>ARD</i> (<i>replicate N 0, replicate N 0</i>)	174
12	Other Results similar to Compactification	177
12.1	Some preliminary Results	177
12.2	Results for <i>Det</i>	178
12.3	Results for <i>Ndet</i>	179
12.4	Other Operators	179
12.4.1	<i>initials</i>	179
12.4.2	<i>Throw</i>	180
12.4.3	(Δ)	180
12.4.4	<i>After</i>	181
12.5	<i>OpSem</i>	183
13	Conclusion	185
13.1	Entry Point	185
13.2	Conclusion	185

Chapter 1

Introduction

Communicating Sequential Processes (CSP) offers a rich and expressive framework for modeling and reasoning about concurrent systems. Its denotational, operational, and algebraic facets are covered by the sessions `HOL-CSP` [3], `HOL-CSPM` [2], `HOL-CSP_OpSem` [4], `HOL-CSP_RS` [6], and `HOL-CSP_PTick`. These developments, initially following Roscoes presentation [7], have since evolved considerably to admit arbitrary types, infinite sets, parameterized termination, and more.

However, this expressiveness comes with a cost: proofs about complex or parametric process architectures often become intricate and hard to scale. Proc-Omata address this issue by slightly constraining the class of processes in order to benefit from more powerful proof techniques. First sketched in [8] and properly conceptualized in [5], the Proc-Omata framework consists in embedding functional automata into CSP. The resulting subclass of processes combines the expressive and compositional features of CSP with automata-like properties (reachability, enableness, absence of divergences), making it particularly amenable to invariant reasoning.

In this entry we start by formalizing the basic notions of functional automata such as reachability and enableness, before introducing the definitions of Proc-Omata themselves. For synchronization product and sequential composition, we then provide combination functions that realize the effect of CSP operators at the level of the underlying automata. These translations are formally proved correct, and culminate in compactification theorems, which generalize the constructions inductively to architectural operators.

Chapter 2

An Excursion into Determinism

This chapter is a preliminary work. Indeed, later in the construction, we will define the notion of Procomata which comes in different flavours, in particular deterministic ones. We will establish then that such ProcOmata produce deterministic processes, a classical notion in CSP that we formalize below.

In a word, a deterministic process cannot refuse an event in which it can engage. More formally, if $s @ [e] \in \mathcal{T} P$, then $(s, \{e\}) \notin \mathcal{F} P$. In this theory, we follow the proof sketch given in [7] for characterizing deterministic processes as maximal elements for the failure-divergence refinement (\sqsubseteq_{FD}). Other lemmas are proved with respect to CSP operators.

2.1 Accepts initials

This notion is a weak version of determinism. It captures the idea of being deterministic for one step.

2.1.1 Definition

unbundle *option-type-syntax*

definition *accepts-initials* :: $\langle ('a, 'r) \text{ process}_{ptick} \Rightarrow \text{bool} \rangle \langle \text{determ}^0 \rangle$
where $\langle \text{determ}^0 P \equiv \forall e \in P^0. \{e\} \notin \mathcal{R} P \rangle$

lemma *accepts-initialsI* : $\langle (\bigwedge e. e \in P^0 \Longrightarrow \{e\} \notin \mathcal{R} P) \Longrightarrow \text{determ}^0 P \rangle$
and *accepts-initialsD* : $\langle \text{determ}^0 P \Longrightarrow e \in P^0 \Longrightarrow \{e\} \notin \mathcal{R} P \rangle$
by (*simp-all add: accepts-initials-def*)

lemma *accepts-initials-def-bis*:

$\langle \text{determ}^0 P \longleftrightarrow (\forall e \in P^0. \forall X \in \mathcal{R} P. e \notin X) \rangle$

by (*auto simp add: accepts-initials-def*)

(*metis Refusals-iff Un-upper1 insert-Diff insert-is-Un is-processT4*)

lemma *accepts-initialsI-bis* : $\langle (\bigwedge e X. e \in P^0 \implies X \in \mathcal{R} P \implies e \notin X) \implies \text{determ}^0 P \rangle$

and *accepts-initialsD-bis* : $\langle \text{determ}^0 P \implies e \in P^0 \implies X \in \mathcal{R} P \implies e \notin X \rangle$

by (*simp-all add: accepts-initials-def-bis*)

2.1.2 First properties

lemma *accepts-initials-STOP* [*simp*] : $\langle \text{determ}^0 \text{STOP} \rangle$

by (*simp add: accepts-initials-def*)

lemma *accepts-initials-SKIP* [*simp*] : $\langle \text{determ}^0 (\text{SKIP } r) \rangle$

by (*simp add: accepts-initials-def Refusals-iff F-SKIP*)

lemma *not-accepts-initials-BOT* [*simp*] : $\langle \neg \text{determ}^0 \perp \rangle$

by (*simp add: accepts-initials-def Refusals-iff F-BOT*)

lemma *accepts-initials-imp-initial-tick-iff-is-SKIP*:

$\langle \text{determ}^0 P \implies \checkmark(r) \in P^0 \longleftrightarrow P = \text{SKIP } r \rangle$

proof (*rule iffI*)

show $\langle P = \text{SKIP } r \implies \checkmark(r) \in P^0 \rangle$ **by** *simp*

next

assume *assms* : $\langle \text{determ}^0 P \rangle \langle \checkmark(r) \in P^0 \rangle$

hence *initials-is* : $\langle P^0 = \{\checkmark(r)\} \rangle$

by (*auto simp add: accepts-initials-def initials-def Refusals-iff subset-iff*)

(*metis append-self-conv2 is-processT6-TR-notin singletonD*)

show $\langle P = \text{SKIP } r \rangle$

proof (*subst SKIP-F-iff[symmetric], unfold failure-refine-def, safe*)

from *assms* **show** $\langle (s, X) \in \mathcal{F} P \implies (s, X) \in \mathcal{F} (\text{SKIP } r) \rangle$ **for** $s X$

by (*cases s, auto simp add: F-SKIP accepts-initials-def-bis Refusals-iff dest!: F-T*)

(*metis initials-is initials-memI singletonD,*

metis T-imp-front-tickFree event_{ptick}.disc(2) front-tickFree-Cons-iff

initials-is initials-memI singletonD)

qed

qed

lemma *accepts-initials-imp-not-initial-tick-iff-is-STOP-or-some-initial-ev*:

$\langle \text{determ}^0 P \implies (\text{range tick} \cap P^0 = \{\}) \longleftrightarrow P = \text{STOP} \vee (\exists e. \text{ev } e \in P^0) \rangle$

by (*simp add: disjoint-iff image-iff*)

(*metis accepts-initials-imp-initial-tick-iff-is-SKIP all-not-in-conv event_{ptick}.exhaust event_{ptick}.sims(3) initials-SKIP initials-empty-iff-STOP singletonD*)

2.1.3 Monotonicity

lemma *mono-accepts-initials-F*: $\langle P \sqsubseteq_F Q \implies \text{determ}^0 P \implies \text{determ}^0 Q \rangle$
by (*frule anti-mono-initials-F*)
(auto simp add: failure-refine-def accepts-initials-def Refusals-iff subset-iff)

lemma *mono-accepts-initials-FD*: $\langle P \sqsubseteq_{FD} Q \implies \text{determ}^0 P \implies \text{determ}^0 Q \rangle$
using *leFD-imp-leF mono-accepts-initials-F* **by** *blast*

lemma *mono-accepts-initials*: $\langle P \sqsubseteq Q \implies \text{determ}^0 P \implies \text{determ}^0 Q \rangle$
by (*drule le-approx-lemma-F, fold failure-refine-def*) (*rule mono-accepts-initials-F*)

lemma *restriction-adm-accepts-initials* [*restriction-adm-process_{ptick}-simpset, simp*]

:

$\langle \text{adm}_\downarrow (\lambda x. \text{determ}^0 (f x)) \rangle$ **if** $\langle \text{cont}_\downarrow f \rangle$
for $f :: \langle 'b :: \text{restriction} \Rightarrow ('a, 'r) \text{process}_{\text{ptick}} \rangle$
proof (*rule restriction-adm-subst*)
from $\langle \text{cont}_\downarrow f \rangle$ **show** $\langle \text{cont}_\downarrow f \rangle$.
next
show $\langle \text{adm}_\downarrow (\text{determ}^0 :: ('a, 'r) \text{process}_{\text{ptick}} \Rightarrow \text{bool}) \rangle$
proof (*rule restriction-admI*)
fix σ **and** $\Sigma :: \langle ('a, 'r) \text{process}_{\text{ptick}} \rangle$
assume $\langle \sigma \dashv \rightarrow \Sigma \rangle \langle \text{determ}^0 (\sigma n) \rangle$ **for** n
show $\langle \text{determ}^0 \Sigma \rangle$
proof (*rule accepts-initialsI*)
fix e **assume** $\langle e \in \Sigma^0 \rangle$
from $\langle \sigma \dashv \rightarrow \Sigma \rangle$ **obtain** $n0$
where $*$: $\langle \Sigma \downarrow \text{Suc} (\text{Suc } 0) = \sigma n0 \downarrow \text{Suc} (\text{Suc } 0) \rangle$
by (*blast dest: restriction-tendstoD*)
with $\langle e \in \Sigma^0 \rangle$ **have** $\langle e \in (\sigma n0)^0 \rangle$
by (*metis initials-restriction-process_{ptick} nat.distinct(1)*)
with $\langle \text{determ}^0 (\sigma n0) \rangle$ **have** $\langle \{e\} \notin \mathcal{R} (\sigma n0) \rangle$
by (*fact accepts-initialsD*)
with $*$ **show** $\langle \{e\} \notin \mathcal{R} \Sigma \rangle$
by (*metis Refusals-iff F-restriction-process_{ptick}-Suc-length-iff-F list.size(3) restriction-related-pred*)

qed

qed

qed

2.1.4 Behaviour on Operators

lemma *accepts-initials-Mprefix* [*simp*]: $\langle \text{determ}^0 (\Box a \in A \rightarrow P a) \rangle$
by (*simp add: accepts-initials-def initials-Mprefix Refusals-iff F-Mprefix*)

lemma *accepts-initials-write0* [*simp*]: $\langle \text{determ}^0 (a \rightarrow P) \rangle$
by (*simp add: write0-def*)

lemma *accepts-initials-write* [*simp*]: $\langle \text{determ}^0 (c!a \rightarrow P) \rangle$

by (simp add: write-def)

lemma *accepts-initials-read* [simp] : $\langle \text{determ}^0 (c?a \in A \rightarrow P a) \rangle$
 by (simp add: read-def)

lemma *accepts-initials-Ndet-iff*:

$\langle \text{determ}^0 (P \sqcap Q) \longleftrightarrow \text{determ}^0 P \wedge \text{determ}^0 Q \wedge P^0 = Q^0 \rangle$,

by (auto simp add: accepts-initials-def initials-Ndet Refusals-iff F-Ndet)

(metis CollectI F-T Un-iff append-Nil initials-def is-processT1 is-processT5-S7 singletonD)+

lemma *accepts-initials-GlobalNdet-iff*:

$\langle \text{determ}^0 (\sqcap a \in A. P a) \longleftrightarrow$

$(\forall a \in A. \text{determ}^0 (P a) \wedge (\forall b \in A. (P a)^0 = (P b)^0)) \rangle$

by (auto simp add: accepts-initials-def initials-GlobalNdet Refusals-iff F-GlobalNdet)

(metis CollectI append-Nil initials-def is-processT1-TR is-processT5-S7 singletonD)+

lemma *accepts-initials-Mndetprefix-iff*:

$\langle \text{determ}^0 (\sqcap a \in A \rightarrow P a) \longleftrightarrow (\exists a. A \subseteq \{a\}) \rangle$

by (simp add: Mndetprefix-GlobalNdet accepts-initials-GlobalNdet-iff initials-write0) blast

lemma *accepts-initials-ndet-write-iff*:

$\langle \text{determ}^0 (c!a \in A \rightarrow P a) \longleftrightarrow (\exists b. c \cdot A \subseteq \{b\}) \rangle$

by (simp add: ndet-write-def accepts-initials-Mndetprefix-iff)

lemma *accepts-initials-SKIPS-iff* :

$\langle \text{determ}^0 (SKIPS R) \longleftrightarrow R = \{\} \vee (\exists r. R = \{r\}) \rangle$

by (simp add: SKIPS-def accepts-initials-GlobalNdet-iff) blast

lemma *accepts-initials-Det* :

$\langle \text{determ}^0 (P \sqcap Q) \longleftrightarrow P = STOP \vee Q = STOP \vee \text{range tick} \cap P^0 \cap Q^0 \neq \{\}$

\vee

$\text{range tick} \cap (P^0 \cup Q^0) = \{\} \rangle$

(is $\langle - \longleftrightarrow ?rhs \rangle$ if *accepts-initials* : $\langle \text{determ}^0 P \rangle \langle \text{determ}^0 Q \rangle$)

proof (cases $\langle P = \perp \rangle$; cases $\langle Q = \perp \rangle$)

show $\langle \text{accepts-initials} (P \sqcap Q) \longleftrightarrow ?rhs \rangle$ if *non-BOT* : $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$

proof (rule iffI)

show $\langle ?rhs \implies \text{determ}^0 (P \sqcap Q) \rangle$

proof (elim disjE)

show $\langle P = STOP \implies \text{determ}^0 (P \sqcap Q) \rangle \langle Q = STOP \implies \text{determ}^0 (P \sqcap Q) \rangle$

by (simp-all add: accepts-initials)

next

from *non-BOT accepts-initials*
show $\langle \text{range tick} \cap P^0 \cap Q^0 \neq \{\} \implies \text{determ}^0 (P \sqcap Q) \rangle$
 $\langle \text{range tick} \cap (P^0 \cup Q^0) = \{\} \implies \text{determ}^0 (P \sqcap Q) \rangle$
by (*simp-all add: accepts-initials-def initials-def Refusals-iff*
Det-projs BOT-iff-Nil-D disjoint-iff)
(metis append-Nil is-processT6-TR-notin singletonD)
qed
next
have $*$: $\langle [Q \neq \text{STOP}; \checkmark(r) \in P^0; \checkmark(r) \notin Q^0; \text{determ}^0 P; \text{determ}^0 (P \sqcap Q)] \implies \text{False} \rangle$
for $P Q$:: $\langle ('a, 'r) \text{ process}_{\text{ptick}} \rangle$ **and** r
by (*metis Un-iff accepts-initials-imp-initial-tick-iff-is-SKIP ex-in-conv*
initials-Det initials-SKIP initials-empty-iff-STOP singletonD)
show $\langle \text{determ}^0 (P \sqcap Q) \implies ?\text{rhs} \rangle$
by (*simp add: disjoint-iff*) (*metis * Det-commute accepts-initials rangeE*)
qed
qed (*use not-accepts-initials-BOT accepts-initials in auto*)

lemma *accepts-initials-GlobalDet* :
 $\langle \text{determ}^0 (\sqcap a \in A. P a) \rangle$ **if** $\langle \bigwedge a. a \in A \implies \text{determ}^0 (P a) \rangle$
 $\langle \text{range tick} \cap (\bigcap a \in A. (P a)^0) \neq \{\} \vee \text{range tick} \cap (\bigcup a \in A. (P a)^0) = \{\} \rangle$
proof (*use that(2) in <elim disjE>*)
from *that(1)* **show** $\langle \text{range tick} \cap (\bigcap a \in A. (P a)^0) \neq \{\} \implies \text{determ}^0 (\sqcap a \in A. P a) \rangle$
by (*auto simp add: accepts-initials-def initials-GlobalDet Refusals-iff F-GlobalDet*)
(meson is-processT8,
metis accepts-initials-imp-initial-tick-iff-is-SKIP
initials-SKIP initials-memI singleton-iff that(1))
next
from *that(1)* **show** $\langle \text{range tick} \cap (\bigcup a \in A. (P a)^0) = \{\} \implies \text{determ}^0 (\sqcap a \in A. P a) \rangle$
by (*auto simp add: accepts-initials-def initials-GlobalDet Refusals-iff F-GlobalDet*)
(blast, metis BOT-iff-Nil-D not-accepts-initials-BOT that(1),
metis UN-I disjoint-iff initials-memI rangeI)
qed

lemma *accepts-initials-Seq_{ptick}* :
 $\langle \text{determ}^0 (P ; \checkmark Q) \iff (\forall r. \checkmark(r) \in P^0 \longrightarrow \text{determ}^0 (Q r)) \rangle$ **if** $\langle \text{determ}^0 P \rangle$
proof (*intro iffI allI impI*)
show $\langle \text{determ}^0 (P ; \checkmark Q) \implies \checkmark(r) \in P^0 \implies \text{determ}^0 (Q r) \rangle$ **for** r
by (*simp add: accepts-initials-imp-initial-tick-iff-is-SKIP <determ⁰ P>*)
next
have $\langle P \neq \perp \rangle$ **using** *not-accepts-initials-BOT* **that** **by** *blast*
show $\langle \text{determ}^0 (P ; \checkmark Q) \rangle$ **if** $*$: $\langle \forall r. \checkmark(r) \in P^0 \longrightarrow \text{determ}^0 (Q r) \rangle$
proof (*rule accepts-initialsI*)
fix e **assume** $\langle e \in (P ; \checkmark Q)^0 \rangle$
then consider a **where** $\langle e = \text{ev } a \rangle \langle \text{ev } a \in P^0 \rangle \mid r$ **where** $\langle \checkmark(r) \in P^0 \rangle \langle e \in$

$(Q\ r)^0$,
by (*simp add: initials-Seq_{ptick} $\langle P \neq \perp \rangle$ blast*)
thus $\langle \{e\} \notin \mathcal{R} (P ; \checkmark Q) \rangle$
proof cases
from $\langle \text{determ}^0 P \rangle \langle P \neq \perp \rangle$ **show** $\langle e = \text{ev } a \implies \text{ev } a \in P^0 \implies \{e\} \notin \mathcal{R} (P ; \checkmark Q) \rangle$ **for** a
unfolding *accepts-initials-def-bis Refusals-def-bis*
by (*simp add: Seq_{ptick}-projs BOT-iff-Nil-D ref-Seq_{ptick}-def*)
(metis $\langle \text{determ}^0 P \rangle$ accepts-initials-imp-initial-tick-iff-is-SKIP event_{ptick}.distinct(1) initials-SKIP initials-memI insertI1 singletonD)
next
show $\langle \checkmark(r) \in P^0 \implies e \in (Q\ r)^0 \implies \{e\} \notin \mathcal{R} (P ; \checkmark Q) \rangle$ **for** r
by (*simp add: $\langle \text{determ}^0 P \rangle$ accepts-initialsD accepts-initials-imp-initial-tick-iff-is-SKIP*)
*)
qed
qed
qed

corollary *accepts-initials-Seq* :
 $\langle \text{determ}^0 (P ; Q) \longleftrightarrow (P^0 \cap \text{range tick} = \{\}) \vee \text{determ}^0 Q \rangle$ **if** $\langle \text{determ}^0 P \rangle$
by (*fold Seq_{ptick}-const, unfold accepts-initials-Seq_{ptick}[OF that] fast*)

lemma (*in Sync_{ptick}-locale*) *accepts-initials-Sync_{ptick}* :
 $\langle \text{determ}^0 (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **if** $\langle \text{determ}^0 P \rangle \langle \text{determ}^0 Q \rangle$
proof (*rule accepts-initialsI*)
from $\langle \text{determ}^0 P \rangle \langle \text{determ}^0 Q \rangle$ **have** $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$ **by** *auto*
fix e **assume** $\langle e \in (P \llbracket S \rrbracket_{\checkmark} Q)^0 \rangle$
with $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$ **consider**
 a **where** $\langle e = \text{ev } a \rangle \langle a \in S \wedge \text{ev } a \in P^0 \wedge \text{ev } a \in Q^0 \vee a \notin S \wedge (\text{ev } a \in P^0 \vee \text{ev } a \in Q^0) \rangle$
 $|$ $r\text{-s } r\ s$ **where** $\langle e = \checkmark(r\text{-s}) \rangle \langle r \otimes \checkmark s = \llbracket r\text{-s} \rrbracket \rangle \langle \checkmark(r) \in P^0 \rangle \langle \checkmark(s) \in Q^0 \rangle$
by (*auto simp add: initials-Sync_{ptick} split: if-split-asm*)
thus $\langle \{e\} \notin \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
proof cases
fix a **assume** $\langle e = \text{ev } a \rangle \langle a \in S \wedge \text{ev } a \in P^0 \wedge \text{ev } a \in Q^0 \vee a \notin S \wedge (\text{ev } a \in P^0 \vee \text{ev } a \in Q^0) \rangle$
from *this(2)* **show** $\langle \{e\} \notin \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
proof (*elim disjE conjE*)
show $\langle \{e\} \notin \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ **if** $\langle \text{ev } a \in P^0 \rangle \langle \text{ev } a \in Q^0 \rangle$
proof (*rule notI*)
assume $\langle \{e\} \in \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
with $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$ **obtain** $X\text{-P } X\text{-Q}$
where $\langle (\llbracket, X\text{-P} \rrbracket) \in \mathcal{F} P \rangle \langle (\llbracket, X\text{-Q} \rrbracket) \in \mathcal{F} Q \rangle \langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \checkmark) X\text{-P } S\ X\text{-Q} \rangle$
by (*auto simp add: Refusals-iff F-Sync_{ptick} BOT-iff-Nil-D dest: Nil-setinterleaves_{ptick}*)
from *this(3)* **have** $\langle \text{ev } a \in X\text{-P} \vee \text{ev } a \in X\text{-Q} \rangle$
by (*auto simp add: $\langle e = \text{ev } a \rangle$ super-ref-Sync_{ptick}-def*)
with $\langle \text{ev } a \in P^0 \rangle \langle \text{ev } a \in Q^0 \rangle \langle (\llbracket, X\text{-P} \rrbracket) \in \mathcal{F} P \rangle \langle (\llbracket, X\text{-Q} \rrbracket) \in \mathcal{F} Q \rangle$ **show**

False
 by (fold Refusals-iff) (metis accepts-initialsD-bis $\langle \text{determ}^0 P \rangle \langle \text{determ}^0 Q \rangle$)
 qed
 next
 show $\langle \{e\} \notin \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ if $\langle a \notin S \rangle \langle \text{ev } a \in P^0 \rangle$
 proof (rule notI)
 assume $\langle \{e\} \in \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 with $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$ obtain $X-P X-Q$
 where $\langle (\llbracket \cdot \rrbracket, X-P) \in \mathcal{F} P \rangle \langle (\llbracket \cdot \rrbracket, X-Q) \in \mathcal{F} Q \rangle \langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \checkmark) \rangle$
X-P S X-Q
 by (auto simp add: Refusals-iff F-Sync_{ptick} BOT-iff-Nil-D dest: Nil-setinterleaves_{ptick})
 from this(3) have $\langle \text{ev } a \in X-P \rangle$
 by (simp add: $\langle e = \text{ev } a \rangle \langle a \notin S \rangle$ super-ref-Sync_{ptick}-def)
 with $\langle \text{ev } a \in P^0 \rangle \langle (\llbracket \cdot \rrbracket, X-P) \in \mathcal{F} P \rangle$ show *False*
 by (fold Refusals-iff) (metis accepts-initialsD-bis $\langle \text{determ}^0 P \rangle$)
 qed
 next
 show $\langle \{e\} \notin \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$ if $\langle a \notin S \rangle \langle \text{ev } a \in Q^0 \rangle$
 proof (rule notI)
 assume $\langle \{e\} \in \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 with $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$ obtain $X-P X-Q$
 where $\langle (\llbracket \cdot \rrbracket, X-P) \in \mathcal{F} P \rangle \langle (\llbracket \cdot \rrbracket, X-Q) \in \mathcal{F} Q \rangle \langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \checkmark) \rangle$
X-P S X-Q
 by (auto simp add: Refusals-iff F-Sync_{ptick} BOT-iff-Nil-D dest: Nil-setinterleaves_{ptick})
 from this(3) have $\langle \text{ev } a \in X-Q \rangle$
 by (simp add: $\langle e = \text{ev } a \rangle \langle a \notin S \rangle$ super-ref-Sync_{ptick}-def)
 with $\langle \text{ev } a \in Q^0 \rangle \langle (\llbracket \cdot \rrbracket, X-Q) \in \mathcal{F} Q \rangle$ show *False*
 by (fold Refusals-iff) (metis accepts-initialsD-bis $\langle \text{determ}^0 Q \rangle$)
 qed
 qed
 next
 fix $r-s$ r s assume $\langle e = \checkmark(r-s) \rangle \langle r \otimes \checkmark s = \llbracket r-s \rrbracket \rangle \langle \checkmark(r) \in P^0 \rangle \langle \checkmark(s) \in Q^0 \rangle$
 show $\langle \{e\} \notin \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 proof (rule notI)
 assume $\langle \{e\} \in \mathcal{R} (P \llbracket S \rrbracket_{\checkmark} Q) \rangle$
 with $\langle P \neq \perp \rangle \langle Q \neq \perp \rangle$ obtain $X-P X-Q$
 where $\langle (\llbracket \cdot \rrbracket, X-P) \in \mathcal{F} P \rangle \langle (\llbracket \cdot \rrbracket, X-Q) \in \mathcal{F} Q \rangle \langle e \in \text{super-ref-Sync}_{\text{ptick}} (\otimes \checkmark) \rangle$
X-P S X-Q
 by (auto simp add: Refusals-iff F-Sync_{ptick} BOT-iff-Nil-D dest: Nil-setinterleaves_{ptick})
 from this(3) have $\langle \checkmark(r) \in X-P \rangle$
 by (simp flip: $\langle r \otimes \checkmark s = \llbracket r-s \rrbracket \rangle$ add: $\langle e = \checkmark(r-s) \rangle$ super-ref-Sync_{ptick}-def)
 (metis Refusals-iff $\langle (\llbracket \cdot \rrbracket, X-Q) \in \mathcal{F} Q \rangle \langle \checkmark(s) \in Q^0 \rangle \langle r \otimes \checkmark s = \llbracket r-s \rrbracket \rangle$
 accepts-initials-def-bis inj-tick-join that(2))
 with $\langle \checkmark(r) \in P^0 \rangle \langle (\llbracket \cdot \rrbracket, X-P) \in \mathcal{F} P \rangle$ show *False*
 by (fold Refusals-iff) (metis accepts-initialsD-bis $\langle \text{determ}^0 P \rangle$)
 qed
 qed
 qed

corollary *accepts-initials-Sync*:

$\langle \text{determ}^0 P \implies \text{determ}^0 Q \implies \text{determ}^0 (P \llbracket S \rrbracket Q) \rangle$

by (*metis SyncClassic.accepts-initials-Sync_{ptick} SyncClassic-is-Sync*)

lemma *accepts-initials-Renaming* : $\langle \text{determ}^0 (\text{Renaming } P f g) \rangle$ **if** $\langle \text{determ}^0 P \rangle$

proof –

from $\langle \text{determ}^0 P \rangle$ **have** $\langle P \neq \perp \rangle$ **by** *auto*

with $\langle \text{determ}^0 P \rangle$ **show** $\langle \text{determ}^0 (\text{Renaming } P f g) \rangle$

by (*simp add: accepts-initials-def initials-Renaming Refusals-iff F-Renaming vimage-def BOT-iff-Nil-D*)

(*metis (full-types) accepts-initials-def Refusals-iff accepts-initials-def-bis mem-Collect-eq*)

qed

lemma *accepts-initials-Throw-iff* : $\langle \text{determ}^0 (P \Theta a \in A. Q a) \longleftrightarrow \text{determ}^0 P \rangle$

using *D-F* **by** (*auto simp add: accepts-initials-def initials-Throw Refusals-iff F-Throw*)

lemma *accepts-initials-Sliding*:

$\langle \text{determ}^0 P \implies \text{determ}^0 Q \implies \text{determ}^0 (P \triangleright Q) \longleftrightarrow$

$P = \text{STOP} \vee P^0 \subseteq Q^0 \wedge (\text{range tick} \cap P^0 \neq \{\}) \vee \text{range tick} \cap Q^0 = \{\}) \rangle$

by (*auto simp add: Sliding-def accepts-initials-Ndet-iff accepts-initials-Det initials-Det*)

2.1.5 Characterizations with After

context *After*

begin

Interesting results about the fact that we can express a process with *Mprefix* and (*after*)

lemma *leFD-SKIPS-Det-Mprefix-After*:

$\langle P \sqsubseteq_{FD} \text{SKIPS} \{r. \checkmark(r) \in P^0\} \sqcap (\sqcap a \in \{a. \text{ev } a \in P^0\} \rightarrow P \text{ after } a) \rangle$ (**is** $\langle P \sqsubseteq_{FD} ?rhs \rangle$)

proof (*unfold failure-divergence-refine-def failure-refine-def divergence-refine-def, safe*)

show $\langle s \in \mathcal{D} ?rhs \implies s \in \mathcal{D} P \rangle$ **for** *s*

by (*auto simp add: D-Det D-SKIPS D-Mprefix D-After*)

next

show $\langle (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} P \rangle$ **for** *s X*

proof (*cases s*)

show $\langle s = \square \implies (s, X) \in \mathcal{F} ?rhs \implies (s, X) \in \mathcal{F} P \rangle$

by (*simp add: F-Det SKIPS-projs STOP-projs F-Mprefix*

F-After disjoint-iff image-iff split: if-split-asm)

(*metis initials-memI append-Nil event_{ptick}.exhaust is-processT1-TR is-processT5-S7,*

$metis\ CollectD\ append.left\text{-}neutral\ initial\text{-}def\ is\text{-}process\ T6\text{-}TR\text{-}notin)$
next
show $\langle s = e \# s' \implies (s, X) \in \mathcal{F} \ ?rhs \implies (s, X) \in \mathcal{F} P \rangle$ **for** $e\ s'$
by $(auto\ simp\ add:\ F\text{-}Det\ SKIPS\text{-}projs\ STOP\text{-}projs\ F\text{-}Mprefix$
 $F\text{-}After\ disjoint\text{-}iff\ image\text{-}iff\ split:\ if\text{-}split\text{-}asm)$
 $(metis\ CollectD\ append\text{-}butlast\text{-}last\text{-}id\ initial\text{-}def\ last\text{-}ConsL\ list.distinct(1))$
tick-T-F
qed
qed

lemma *accepts-initials-imp-eq-Mprefix-After:*
 $\langle P = ($ *if* $\exists r. \checkmark(r) \in P^0$ *then* *SKIP* $(THE\ r. \checkmark(r) \in P^0)$
else $\Box a \in \{e. ev\ e \in P^0\} \rightarrow P\ after\ a) \rangle$ **(is** $\langle P = \ ?rhs \rangle$
if $\langle determ^0\ P \rangle$
proof –
from *not-accepts-initials-BOT* $\langle determ^0\ P \rangle$ **have** *non-BOT:* $\langle P \neq \perp \rangle$ **by** *blast*
note *initial-tick-iff-is-SKIP = accepts-initials-imp-initial-tick-iff-is-SKIP*[*OF* $\langle determ^0\ P \rangle$]

have $*$: $\langle \ ?rhs = (SKIPS\ \{r. \checkmark(r) \in P^0\}) \Box$
 $(\Box a \in \{e. ev\ e \in P^0\} \rightarrow P\ after\ a) \rangle$ **(is** $\langle \ ?rhs = \ ?rhs\text{-}bis \rangle$
by $(simp,\ rule\ impI,\ elim\ exE,\ simp\ add:\ initial\text{-}tick\text{-}iff\text{-}is\text{-}SKIP)$

show $\langle P = \ ?rhs \rangle$
proof $(rule\ FD\text{-}antisym)$
show $\langle P \sqsubseteq_{FD} \ ?rhs \rangle$ **by** $(unfold\ *)$ $(fact\ leFD\text{-}SKIPS\text{-}Det\text{-}Mprefix\text{-}After)$
next
show $\langle \ ?rhs \sqsubseteq_{FD} P \rangle$
proof $(unfold\ failure\text{-}divergence\text{-}refine\text{-}def\ failure\text{-}refine\text{-}def\ divergence\text{-}refine\text{-}def,$
safe)
from *non-BOT* **show** $\langle s \in \mathcal{D} P \implies s \in \mathcal{D} \ ?rhs \rangle$ **for** s
by $(cases\ s;\ simp\ add:\ D\text{-}Det\ D\text{-}SKIP\ D\text{-}STOP\ D\text{-}Mprefix\ BOT\text{-}iff\ Nil\text{-}D$
 $image\text{-}iff\ D\text{-}After\ initial\text{-}tick\text{-}iff\text{-}is\text{-}SKIP)$
 $(metis\ BOT\text{-}iff\text{-}tick\text{-}D\ initial\text{-}memI\ D\text{-}T\ D\text{-}imp\text{-}front\text{-}tickFree\ event_{ptick}.disc(2)$
 $event_{ptick}.exhaust\ front\text{-}tickFree\text{-}Cons\text{-}iff\ initial\text{-}SKIP\ non\text{-}BOT\ single\text{-}$
 $tonD)$
next
have $*$: $\langle \exists r. \checkmark(r) \in P^0 \implies \exists !r. \checkmark(r) \in P^0 \rangle$
by $(metis\ event_{ptick}.inject(2)\ initial\text{-}tick\text{-}iff\text{-}is\text{-}SKIP\ initial\text{-}SKIP\ single\text{-}$
 $tonD)$
show $\langle (t, X) \in \mathcal{F} P \implies (t, X) \in \mathcal{F} \ ?rhs \rangle$ **for** $t\ X$
proof $(cases\ t)$
from $*$ **show** $\langle t = \Box \implies (t, X) \in \mathcal{F} P \implies (t, X) \in \mathcal{F} \ ?rhs \rangle$
by $(simp\ add:\ add:\ F\text{-}Mprefix\ disjoint\text{-}iff\ image\text{-}iff\ initial\text{-}tick\text{-}iff\text{-}is\text{-}SKIP)$
 $(metis\ (lifting)\ \langle determ^0\ P \rangle\ Refusals\text{-}iff[of\ X\ P]$
 $accepts\text{-}initialsD\text{-}bis[of\ P - X]\ the\ equality[of\ \langle \lambda r. P = SKIP\ r \rangle])$
next
from $*$ **show** $\langle t = e \# t' \implies (t, X) \in \mathcal{F} P \implies (t, X) \in \mathcal{F} \ ?rhs \rangle$ **for** $e\ t'$

by (cases e , simp-all add: F -Mprefix F -After disjoint-iff image-iff initial-tick-iff-is-SKIP)
 (metis F -T event_{ptick}.distinct(1) initials-SKIP initials-memI singletonD,
 metis (mono-tags, lifting) F -T initial-tick-iff-is-SKIP initials-memI
 the-equality)
 qed
 qed
 qed
 qed

theorem *is-some-Mprefix-iff*:
 $\langle \exists A Q. P = \Box a \in A \rightarrow Q a \rangle \longleftrightarrow \text{range tick} \cap P^0 = \{\} \wedge \text{accepts-initials } P \rangle$
 for $P :: \langle ('a, 'r) \text{ process}_{ptick} \rangle$
proof (intro iffI exI)
 show $\langle \exists A Q. P = \text{Mprefix } A Q \implies \text{range tick} \cap P^0 = \{\} \wedge \text{accepts-initials } P \rangle$
 by (auto simp add: initials-Mprefix image-iff disjoint-iff)
next
 from *accepts-initials-imp-eq-Mprefix-After*
 show $\langle \text{range tick} \cap P^0 = \{\} \wedge \text{accepts-initials } P \implies$
 $P = \Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a \rangle$
 by (meson disjoint-iff rangeI)
 qed

lemma *tick-not-initial-imp-STOP-Ndet-Mndetprefix-After-FD*:
 $\langle \text{range tick} \cap P^0 = \{\} \implies \text{STOP} \sqcap (\Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a) \sqsubseteq_{FD} P \rangle$
 by (cases $\langle P = \perp \rangle$, solves $\langle \text{simp} \rangle$,
 auto simp add: refine-defs Ndet-projs F -STOP Mprefix-projs After-projs
 BOT-iff-Nil-D)
 (metis initials-memI F -T Int-iff empty-iff event_{ptick}.exhaust neq-Nil-conv rangeI,
 metis initials-memI D -T disjoint-iff event_{ptick}.exhaust neq-Nil-conv rangeI)

— With this we could obtain something about *CHAOS* and tF and $\mathcal{D} P = \{\}$ but we already have this.

lemma $\langle \text{lifelock-free } P \longleftrightarrow \mathcal{D} P = \{\} \wedge (\forall t \in \mathcal{T} P. tF t) \rangle$
 using *lifelock-free-is-non-terminating non-terminating-is-right nonterminating-implies-div-free*
 by *blast*

lemma *STOP-Ndet-SKIPS-Ndet-Mprefix-After-leF* :
 $\langle \text{STOP} \sqcap \text{SKIPS } \{r. \checkmark(r) \in P^0\} \sqcap (\Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a) \sqsubseteq_F P \rangle$
 (is $\langle - \sqcap ?lhs1 \sqcap ?lhs2 \sqsubseteq_F P \rangle$)
proof (unfold failure-refine-def, safe)
 fix $t X$ assume $\langle t, X \rangle \in \mathcal{F} P$
 then consider $\langle t = [] \rangle \mid r$ where $\langle t = [\checkmark(r)] \rangle \langle r \in \{r. \checkmark(r) \in P^0\} \rangle$

| $a t'$ **where** $\langle t = ev a \# t' \rangle \langle a \in \{a. ev a \in P^0\} \rangle$
by (*cases t, simp-all*) (*metis F-T F-imp-front-tickFree event_{ptick}.exhaust front-tickFree-Cons-iff initials-memI is-ev-def*)
thus $\langle (t, X) \in \mathcal{F} (STOP \sqcap ?lhs1 \sqcap ?lhs2) \rangle$
proof cases
show $\langle t = [] \implies (t, X) \in \mathcal{F} (STOP \sqcap ?lhs1 \sqcap ?lhs2) \rangle$ **by** (*simp add: F-Ndet F-STOP*)
next
fix r assume $\langle t = [\checkmark(r)] \rangle$
with $\langle (t, X) \in \mathcal{F} P \rangle$ **have** $\langle (t, X) \in \mathcal{F} (?lhs1) \rangle$
by (*auto simp add: F-SKIPS intro!: initials-memI' F-T*)
thus $\langle (t, X) \in \mathcal{F} (STOP \sqcap ?lhs1 \sqcap ?lhs2) \rangle$ **by** (*simp add: F-Ndet*)
next
fix a t' assume $\langle t = ev a \# t' \rangle$
with $\langle (t, X) \in \mathcal{F} P \rangle$ **have** $\langle (t, X) \in \mathcal{F} ?lhs2 \rangle$
by (*auto simp add: F-Mprefix F-After intro!: initials-memI F-T*)
thus $\langle (t, X) \in \mathcal{F} (STOP \sqcap ?lhs1 \sqcap ?lhs2) \rangle$ **by** (*simp add: F-Ndet*)
qed
qed

lemma non-BOT-imp-Mprefix-After-leD :
 $\langle \Box a \in \{e. ev e \in P^0\} \rightarrow P \text{ after } a \sqsubseteq_D P \rangle$ (**is** $\langle \cdot ?lhs \sqsubseteq_D P \rangle$) **if** $\langle P \neq \perp \rangle$
proof (*unfold divergence-refine-def, rule subsetI*)
fix t assume $\langle t \in \mathcal{D} P \rangle$
with $\langle P \neq \perp \rangle$ **obtain** $a t'$ **where** $\langle t = ev a \# t' \rangle$
by (*cases t, simp add: BOT-iff-Nil-D, simp add: BOT-iff-tick-D*)
(metis D-imp-front-tickFree event_{ptick}.exhaust front-tickFree-Cons-iff is-ev-def)
with $\langle t \in \mathcal{D} P \rangle$ **show** $\langle t \in \mathcal{D} ?lhs \rangle$
by (*auto simp add: D-Mprefix D-After intro: D-T initials-memI*)
qed

lemma non-BOT-imp-STOP-Ndet-SKIPS-Ndet-Mprefix-After-leFD :
 $\langle P \neq \perp \implies STOP \sqcap SKIPS \{r. \checkmark(r) \in P^0\} \sqcap (\Box a \in \{e. ev e \in P^0\} \rightarrow P \text{ after } a) \sqsubseteq_{FD} P \rangle$
by (*rule leF-leD-imp-leFD[OF STOP-Ndet-SKIPS-Ndet-Mprefix-After-leF]*)
(use non-BOT-imp-Mprefix-After-leD Ndet-D-self-right trans-D in blast)

theorem singl-initial-imp-equals-prefix-After:
 $\langle P = (\text{if } UNIV \notin \mathcal{R} P \text{ then } a \rightarrow P \text{ after } a \text{ else } STOP \sqcap (a \rightarrow P \text{ after } a)) \rangle$
if *initials-is* : $\langle \text{initials } P = \{ev a\} \rangle$
proof (*split if-split, intro conjI impI*)
assume *not-all-refusals* : $\langle UNIV \notin \mathcal{R} P \rangle$
have $\$: \langle e \neq ev a \implies [e] \notin \mathcal{T} P \rangle$ **for** e **using** *initials-is unfolding initials-def*
by *auto*
{ assume $\langle \{ev a\} \in \mathcal{R} P \rangle$
from *is-processT5[rule-format, of $\langle [] \rangle \langle \{ev a\} \rangle P \langle UNIV - \{ev a\} \rangle$, folded*

Refusals-iff,
simplified this T-F-spec, simplified, rule-format, OF \$, simplified]
have $\langle UNIV \in \mathcal{R} P \rangle$.
with *not-all-refusals* **have** *False* **by** *simp*
} **hence** *not-in-refusal*: $\langle \{ev\} a \notin \mathcal{R} P \rangle$ **by** *blast*
show $\langle P = a \rightarrow P \text{ after } a \rangle$
by (*unfold write0-def, subst accepts-initials-imp-eq-Mprefix-After*)
(solves simp add: accepts-initials-def initials-is not-in-refusal,
auto simp add: that(1) intro: mono-Mprefix-eq)
next
assume *all-refusals* : $\langle \neg UNIV \notin \mathcal{R} P \rangle$
from *tick-not-initial-imp-STOP-Ndet-Mndetprefix-After-FD*[*of P*]
have * : $\langle STOP \sqcap (a \rightarrow P \text{ after } a) \sqsubseteq_{FD} P \rangle$
by (*simp add: that(1) Mprefix-singl image-iff*)
from *leFD-SKIPS-Det-Mprefix-After*[*of P*] *all-refusals*
have ** : $\langle P \sqsubseteq_{FD} STOP \sqcap (a \rightarrow P \text{ after } a) \rangle$
by (*auto simp add: refine-defs that(1) write0-projs Mprefix-projs*
Ndet-projs STOP-projs Refusals-iff)
(meson is-processT4 subset-UNIV)
from ** * **show** $\langle P = STOP \sqcap (a \rightarrow P \text{ after } a) \rangle$ **by** (*fact FD-antisym*)
qed

lemma $\langle \{ev\} e \notin \mathcal{R} P \implies ev\ e \in P^0 \rangle$
by (*simp add: Refusals-iff initials-def*)
(metis is-processT1-TR is-processT5-S7 self-append-conv2 singletonD)

end

2.2 Deterministic process

2.2.1 Definition

definition *deterministic* :: $\langle ('a, 'r) \text{ process}_{ptick} \Rightarrow \text{bool} \rangle$ (*determ*)
where $\langle \text{determ } P \equiv \forall s\ e. s @ [e] \in \mathcal{T} P \longrightarrow (s, \{e\}) \notin \mathcal{F} (P) \rangle$

lemma *deterministicI* : $\langle (\bigwedge t\ e. t @ [e] \in \mathcal{T} P \implies (t, \{e\}) \notin \mathcal{F} (P)) \implies \text{determ } P \rangle$
and *deterministicD* : $\langle \text{determ } P \implies t @ [e] \in \mathcal{T} P \implies (t, \{e\}) \notin \mathcal{F} (P) \rangle$
by (*simp-all add: deterministic-def*)

lemma *deterministic-STOP* [*simp*] : $\langle \text{determ } STOP \rangle$
and *deterministic-SKIP* [*simp*] : $\langle \text{determ } (SKIP\ r) \rangle$
by (*simp-all add: deterministic-def T-STOP SKIP-projs*)

lemma *deterministic-div-free* : $\langle \text{determ } P \implies \mathcal{D} P = \{\} \rangle$
by (*auto simp add: deterministic-def*)
(metis D-T D-imp-front-tickFree append-butlast-last-id div-butlast-when-non-tickFree-iff
front-tickFree-single is-processT7 is-processT8 tickFree-Nil)

lemma *not-deterministic-BOT* [*simp*] : $\langle \neg \text{determ } \perp \rangle$
using *BOT-iff-Nil-D deterministic-div-free* **by** *blast*

2.2.2 Monotonicity

lemma *mono-deterministic-F*: $\langle P \sqsubseteq_F Q \implies \text{determ } P \implies \text{determ } Q \rangle$
by (*meson T-F-spec deterministic-def failure-refine-def subset-iff*)

lemma *mono-deterministic-FD*: $\langle P \sqsubseteq_{FD} Q \implies \text{determ } P \implies \text{determ } Q \rangle$
using *leFD-imp-leF mono-deterministic-F* **by** *blast*

lemma *mono-deterministic*: $\langle P \sqsubseteq Q \implies \text{determ } P \implies \text{determ } Q \rangle$
using *le-approx-imp-le-ref mono-deterministic-FD* **by** *auto*

lemma *restriction-adm-deterministic* [*restriction-adm-process_{ptick}-simpset, simp*]
 :

$\langle \text{adm}_\downarrow (\lambda x. \text{determ } (f x)) \rangle$ **if** $\langle \text{cont}_\downarrow f \rangle$
for $f :: \langle 'b :: \text{restriction} \Rightarrow ('a, 'r) \text{process}_{ptick} \rangle$
proof (*rule restriction-adm-subst*)
from $\langle \text{cont}_\downarrow f \rangle$ **show** $\langle \text{cont}_\downarrow f \rangle$.
next
show $\langle \text{adm}_\downarrow (\text{determ} :: ('a, 'r) \text{process}_{ptick} \Rightarrow \text{bool}) \rangle$
proof (*rule restriction-admI*)
fix σ **and** $\Sigma :: \langle ('a, 'r) \text{process}_{ptick} \rangle$
assume $\langle \sigma \dashv\rightarrow \Sigma \rangle$ $\langle \text{determ } (\sigma n) \rangle$ **for** n
show $\langle \text{determ } \Sigma \rangle$
proof (*rule deterministicI*)
fix $t e$ **assume** $\langle t @ [e] \in \mathcal{T} \Sigma \rangle$
from $\langle \sigma \dashv\rightarrow \Sigma \rangle$ **obtain** $n0$
where $*$: $\langle \Sigma \downarrow \text{Suc } (\text{length } (t @ [e])) = \sigma n0 \downarrow \text{Suc } (\text{length } (t @ [e])) \rangle$
by (*blast dest: restriction-tendstoD*)
with $\langle t @ [e] \in \mathcal{T} \Sigma \rangle$ **have** $\langle t @ [e] \in \mathcal{T} (\sigma n0) \rangle$
by (*metis T-restriction-process_{ptick}-Suc-length-iff-T*)
with $\langle \text{deterministic } (\sigma n0) \rangle$ **have** $\langle (t, \{e\}) \notin \mathcal{F} (\sigma n0) \rangle$
by (*fact deterministicD*)
with $*$ **show** $\langle (t, \{e\}) \notin \mathcal{F} \Sigma \rangle$
by (*metis F-restriction-process_{ptick}-Suc-length-iff-F length-append-singleton restriction-related-pred*)
qed
qed
qed

2.2.3 Characterization as Maximal

Some preliminary work

definition *is-process_T* :: $\langle ('a, 'r) \text{trace}_{ptick} \text{set} \Rightarrow \text{bool} \rangle$
where $\langle \text{is-process}_T T \equiv$

$$\begin{aligned} & \llbracket \in T \wedge (\forall t \in T. ftF t) \wedge (\forall t u. t @ u \in T \longrightarrow t \in T) \wedge \\ & (\forall t r e. t @ [\checkmark(r)] \in T \longrightarrow e \neq \checkmark(r) \longrightarrow t @ [e] \notin T) \end{aligned}$$

typedef $\langle 'a, 'r \rangle$ *process_T* = $\langle \{ T :: ('a, 'r) \text{ trace}_{ptick} \text{ set} . \text{is-process}_T T \} \rangle$
proof (*rule exI*)
show $\langle \{ \llbracket \} \in \{ T. \text{is-process}_T T \} \rangle$ **unfolding** *is-process_T-def* **by** *simp*
qed

setup-lifting *type-definition-process_T*

lift-definition *Traces_T* ::
 $\langle ('a, 'r) \text{ process}_T \Rightarrow ('a, 'r) \text{ trace}_{ptick} \text{ set} \rangle \langle \mathcal{T}_T \rangle$
is $\langle \lambda P. \text{Rep-process}_T P \rangle$.

lemma *Process_T-eq-spec* : $\langle T = U \longleftrightarrow \mathcal{T}_T T = \mathcal{T}_T U \rangle$
by (*simp add: Rep-process_T-inject Traces_T.rep-eq*)

lemma *is-process_T-1* : $\langle \llbracket \in \mathcal{T}_T P \rangle$
and *is-process_T-2* : $\langle s \in \mathcal{T}_T P \Longrightarrow ftF s \rangle$
and *is-process_T-3* : $\langle s @ t \in \mathcal{T}_T P \Longrightarrow s \in \mathcal{T}_T P \rangle$
and *is-process_T-4* : $\langle s @ [\checkmark(r)] \in \mathcal{T}_T P \Longrightarrow e \neq \checkmark(r) \Longrightarrow s @ [e] \notin \mathcal{T}_T P \rangle$
by (*transfer, meson Rep-process_T[simplified, unfolded is-process_T-def]*)⁺

lemmas *is-process_T-def-bis* = *is-process_T-def*[*of* $\langle \text{Rep-process}_T \text{-}, \text{folded Traces}_T.\text{rep-eq} \rangle$]

lift-definition *process_{ptick}-of-process_T* ::
 $\langle ('a, 'r) \text{ process}_T \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
is $\langle \lambda T. (\{ (s, X). s \in \mathcal{T}_T T \wedge X \subseteq - \{ e. s @ [e] \in \mathcal{T}_T T \} \}, \{ \}) \rangle$
by (*auto simp add: is-process-def FAILURES-def DIVERGENCES-def*
intro: is-process_T-1 is-process_T-2 is-process_T-3)
(meson is-process_T-4))

lemma *F-process_{ptick}-of-process_T* :
 $\langle \mathcal{F} (\text{process}_{ptick}\text{-of-process}_T T) = \{ (s, X). s \in \mathcal{T}_T T \wedge X \subseteq - \{ e. s @ [e] \in \mathcal{T}_T T \} \} \rangle$
and *D-process_{ptick}-of-process_T* :
 $\langle \mathcal{D} (\text{process}_{ptick}\text{-of-process}_T T) = \{ \} \rangle$
and *T-process_{ptick}-of-process_T* :
 $\langle \mathcal{T} (\text{process}_{ptick}\text{-of-process}_T T) = \mathcal{T}_T T \rangle$
by (*simp-all add: Failures-def FAILURES-def Divergences-def DIVERGENCES-def*
Traces-def TRACES-def process_{ptick}-of-process_T.rep-eq) *blast*

lemmas *process_{ptick}-of-process_T-projs* = *F-process_{ptick}-of-process_T*

$D\text{-process}_{ptick}\text{-of-process}_T$ $T\text{-process}_{ptick}\text{-of-process}_T$

Now the big results

lemma *bij-betw-det* :

$\langle \text{bij-betw } process_{ptick}\text{-of-process}_T \text{ UNIV } \{P :: ('a, 'r) process_{ptick}. \text{determ } P\} \rangle$
 (is $\langle \text{bij-betw } process_{ptick}\text{-of-process}_T \text{ ?S1 ?S2} \rangle$)

proof (intro *bij-betw-imageI*)

show $\langle \text{inj-on } process_{ptick}\text{-of-process}_T \text{ ?S1} \rangle$

by (rule *inj-onI*) (auto simp add: *Process-eq-spec process_{ptick}\text{-of-process}_T\text{-projs Process}_T\text{-eq-spec}*)

next

show $\langle process_{ptick}\text{-of-process}_T \text{ ' ?S1 = ?S2} \rangle$

proof (intro *subset-antisym subsetI*; clarify)

show $\langle \text{determ } (process_{ptick}\text{-of-process}_T P) \rangle$ **for** $P :: \langle ('a, 'r) process_T \rangle$

by (rule *deterministicI*) (simp add: *process_{ptick}\text{-of-process}_T\text{-projs}*)

next

fix $P :: \langle ('a, 'r) process_{ptick} \rangle$

assume *det* : $\langle \text{deterministic } P \rangle$

hence $*$: $\langle \text{Rep-process}_T (\text{Abs-process}_T (\mathcal{T} P)) = \mathcal{T} P \rangle$

by (intro *Abs-process}_T\text{-inverse}*)

(simp add: *deterministic-def is-process}_T\text{-def is-process}_T2\text{-TR,metis T-F-spec is-process}_T3 \text{is-process}_T6\text{-notin singletonD}*)

show $\langle P \in process_{ptick}\text{-of-process}_T \text{ ' ?S1} \rangle$

proof (subst *image-iff*, rule *beXI*)

show $\langle P = process_{ptick}\text{-of-process}_T (\text{Abs-process}_T (\mathcal{T} P)) \rangle$

by (auto intro!: *Process-eq-optimizedI* simp add: *process_{ptick}\text{-of-process}_T\text{-projs Traces}_T\text{-def} * *det deterministic-div-free subset-iff F-T*)*

(meson *det deterministic-def empty-subsetI insert-subset is-process}_T, use *is-process}_T5\text{-S7* in blast)*

next

show $\langle \text{Abs-process}_T (\mathcal{T} P) \in ?S1 \rangle$ **by** (simp add: *Traces}_T\text{-def} **)

qed

qed

qed

lemma *SKIPS-is-GlobalDet-SKIP* : $\langle \text{SKIPS } R = \square r \in R. \text{SKIP } r \rangle$

by (auto simp add: *Process-eq-spec SKIPS-projs GlobalDet-projs SKIP-projs*)

lemma *SKIP-Ndet-SKIP-is-SKIP-Det-SKIP* : $\langle \text{SKIP } r \sqcap \text{SKIP } s = \text{SKIP } r \sqcap \text{SKIP } s \rangle$

by (auto simp add: *Process-eq-spec Det-projs Ndet-projs SKIP-projs*)

theorem *P-FD-some-det* :

— In the generalization, since several terminations may occur after the same trace in the initial process, we have to specify a choice.

fixes *termination-choice* :: $\langle ('a, 'r) \text{ trace}_{ptick} \Rightarrow 'r \rangle$
assumes $\langle \bigwedge t. \exists r. t @ [\checkmark(r)] \in \mathcal{T} P \implies \text{termination-choice } t \in \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$
defines $\langle T \equiv \{t \in \mathcal{T} P. \forall t' < t. (\exists r. t' @ [\checkmark(r)] \in \mathcal{T} P) \longrightarrow t = t' @ [\checkmark(\text{termination-choice } t')]\} \rangle$
shows $\langle P \sqsubseteq_{FD} \text{process}_{ptick}\text{-of-process}_T (\text{Abs-process}_T T) \rangle$
proof (*unfold failure-divergence-refine-def failure-refine-def divergence-refine-def, intro conjI*)
show $\langle \mathcal{D} (\text{process}_{ptick}\text{-of-process}_T (\text{Abs-process}_T T)) \subseteq \mathcal{D} P \rangle$ **by** (*simp add: D-process_{ptick}-of-process_T*)
next
have $*$: $\langle T \in \{T. \text{is-process}_T T\} \rangle$
by (*auto simp add: T-def is-process_T-def T-imp-front-tickFree intro: is-process_{T3}-TR-append (metis prefix-prefix append-eq-first-pref-spec less-list-def nless-le self-append-conv, metis less-self)*)

show $\langle \mathcal{F} (\text{process}_{ptick}\text{-of-process}_T (\text{Abs-process}_T T)) \subseteq \mathcal{F} P \rangle$
proof *safe*
fix $s X$ **assume** $\langle (s, X) \in \mathcal{F} (\text{process}_{ptick}\text{-of-process}_T (\text{Abs-process}_T T)) \rangle$
hence $\langle s \in \mathcal{T} P \rangle$
by (*simp add: F-process_{ptick}-of-process_T Traces_T-def Abs-process_T-inverse[OF *]) (simp add: T-def)*)
show $\langle (s, X) \in \mathcal{F} P \rangle$
proof (*cases* $\langle \exists r. s @ [\checkmark(r)] \in \mathcal{T} P \rangle$)
assume $\langle \exists r. s @ [\checkmark(r)] \in \mathcal{T} P \rangle$
hence $\langle s @ [\checkmark(\text{termination-choice } s)] \in \mathcal{T} P \rangle$ **by** (*metis assms(1) mem-Collect-eq*)
with $\langle (s, X) \in \mathcal{F} (\text{process}_{ptick}\text{-of-process}_T (\text{Abs-process}_T T)) \rangle$ **have** $\langle \checkmark(\text{termination-choice } s) \notin X \rangle$
unfolding *F-process_{ptick}-of-process_T Traces_T.abs-eq Abs-process_T-inverse[OF *]*
by (*simp add: subset-iff T-def (metis prefix-snoc append-T-imp-tickFree nless-le non-tickFree-tick not-Cons-self2 tickFree-append-iff)*)
with $\langle s @ [\checkmark(\text{termination-choice } s)] \in \mathcal{T} P \rangle$ **show** $\langle (s, X) \in \mathcal{F} P \rangle$
by (*metis is-process_{T6}-TR-notin*)
next
assume $\langle \nexists r. s @ [\checkmark(r)] \in \mathcal{T} P \rangle$
with $\langle (s, X) \in \mathcal{F} (\text{process}_{ptick}\text{-of-process}_T (\text{Abs-process}_T T)) \rangle$ **have** $\langle X \subseteq -\{e. s @ [e] \in \mathcal{T} P\} \rangle$
unfolding *F-process_{ptick}-of-process_T Traces_T.abs-eq Abs-process_T-inverse[OF *]*
by (*simp add: subset-iff T-def (metis prefix-snoc append-T-imp-tickFree nless-le non-tickFree-tick not-Cons-self2 tickFree-append-iff)*)
with *is-process_{T5-S7}[OF* $\langle s \in \mathcal{T} P \rangle$ **show** $\langle (s, X) \in \mathcal{F} P \rangle$ **by** *blast*
qed
qed
qed

theorem *deterministic-iff-maximal-for-leFD*:
 $\langle \text{determ } P \longleftrightarrow (\forall Q. P \sqsubseteq_{FD} Q \longrightarrow P = Q) \rangle$ **for** $P :: \langle ('a, 'r) \text{ process}_{ptick} \rangle$
— see TPC, chapter 9)

proof (*intro iffI allI impI*)
fix Q **assume** $\langle \text{determ } P \rangle$ **and** $\langle P \sqsubseteq_{FD} Q \rangle$
from $\langle P \sqsubseteq_{FD} Q \rangle$ **have** $\text{no-div} : \langle \mathcal{D} P = \{\} \rangle$ $\langle \mathcal{D} Q = \{\} \rangle$ **and** $F\text{-subset} : \langle \mathcal{F} Q \subseteq \mathcal{F} P \rangle$
by (*simp-all add: $\langle \text{determ } P \rangle$ deterministic-div-free refine-defs*)

have $\text{same-T} : \langle \mathcal{T} P = \mathcal{T} Q \rangle$
proof (*rule subset-antisym*)
show $\langle \mathcal{T} P \subseteq \mathcal{T} Q \rangle$
proof (*rule ccontr*)
assume $\langle \neg \mathcal{T} P \subseteq \mathcal{T} Q \rangle$
then obtain $s\ e$ **where** $*$: $\langle s @ [e] \in \text{min-elems } (\mathcal{T} P - \mathcal{T} Q) \rangle$
by (*metis DiffD2 Diff-eq-empty-iff Nil-elem-T elem-min-elems min-elems4 rev-exhaust*)
hence $\langle s \in \mathcal{T} Q \rangle$ **unfolding** *min-elems-def*
by *simp (metis DiffI T-F-spec is-processT3 less-self)*
with $*$ **have** $\langle (s, \{e\}) \in \mathcal{F} Q \rangle$
by (*metis Diff-iff elem-min-elems is-processT5-S7 singletonD*)
from *set-mp[OF F-subset this]* **have** $\langle (s, \{e\}) \in \mathcal{F} P \rangle$.
moreover have $\langle (s, \{e\}) \notin \mathcal{F} P \rangle$ **by** (*metis * Diff-iff $\langle \text{determ } P \rangle$ deterministicD elem-min-elems*)
ultimately show *False* **by** *blast*
qed

next
show $\langle \mathcal{T} Q \subseteq \mathcal{T} P \rangle$ **by** (*simp add: F-subset F-subset-imp-T-subset*)
qed

have $\text{same-F} : \langle \mathcal{F} P = \mathcal{F} Q \rangle$
proof (*rule subset-antisym*)
show $\langle \mathcal{F} P \subseteq \mathcal{F} Q \rangle$
proof (*rule ccontr*)
assume $\langle \neg \mathcal{F} P \subseteq \mathcal{F} Q \rangle$
then obtain $s\ X$ **where** $*$: $\langle (s, X) \in \mathcal{F} P - \mathcal{F} Q \rangle$ $\langle X \neq \{\} \rangle$
by (*metis DiffI T-F-spec same-T subrelI*)

have $\langle e \in X \implies s @ [e] \notin \mathcal{T} P \rangle$ **for** e
by (*metis *(1) DiffD1 $\langle \text{determ } P \rangle$ deterministicD insert-Diff insert-is-Un is-processT4 sup-ge1*)
thus *False* **by** (*metis *(1) DiffE F-T is-processT5-S7 same-T*)
qed

next
show $\langle \mathcal{F} Q \subseteq \mathcal{F} P \rangle$ **by** (*fact F-subset*)
qed
show $\langle P = Q \rangle$ **by** (*simp add: Process-eq-spec failure-refine-def divergence-refine-def*)

no-div same-F)

next

define *termination-choice* **where** $\langle \text{termination-choice } s \equiv (\text{SOME } r. s @ [\checkmark(r)] \in \mathcal{T} P) \rangle$ **for** s

have $\$: \langle \exists r. s @ [\checkmark(r)] \in \mathcal{T} P \implies \text{termination-choice } s \in \{r. s @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$ **for** s

by (*simp add: termination-choice-def*) (*meson someI*)

define T **where** $\langle T \equiv \{s \in \mathcal{T} P. \forall s' < s. (\exists r. s' @ [\checkmark(r)] \in \mathcal{T} P) \longrightarrow s = s' @ [\checkmark(\text{termination-choice } s')]\} \rangle$

have $*$: $\langle T \in \{T. \text{is-process}_T T\} \rangle$

by (*auto simp add: T-def is-process_T-def T-imp-front-tickFree intro: is-process_{T3}-TR-append*)
(metis prefix-prefix append-eq-first-pref-spec less-list-def nless-le self-append-conv,
metis less-self)

assume *maximal* : $\langle \forall Q. P \sqsubseteq_{FD} Q \longrightarrow P = Q \rangle$

with $\$ P\text{-FD-some-det } T\text{-def}$ **have** $\langle P = \text{process}_{\text{ptick}}\text{-of-process}_T (\text{Abs-process}_T T) \rangle$ **by** *blast*

moreover **have** $\langle \text{Abs-process}_T T \in \{P. \forall s r e. s @ [\checkmark(r)] \in \mathcal{T}_T P \longrightarrow e \neq \checkmark(r) \longrightarrow s @ [e] \notin \mathcal{T}_T P\} \rangle$

by (*simp add: Traces_T-def Abs-process_T-inverse[OF *]*)
*(metis * is-process_T-def mem-Collect-eq)*

ultimately show $\langle \text{determ } P \rangle$ **using** *bij-betwE[OF bij-betw-det]* **by** *blast qed*

lemma $\langle \text{determ } P \implies X \in \mathcal{R} P \implies X \subseteq - P^0 \rangle$

unfolding *deterministic-def Refusals-iff initials-def*

by *auto (metis insert-Diff insert-is-Un process-charn self-append-conv2 sup-ge1)*

We have the immediate powerful corollaries.

corollary (*in After*) *deterministic-process-eq-SKIPS-Det-Mprefix-After* :

$\langle \text{determ } P \implies P = \text{SKIPS } \{r. \checkmark(r) \in P^0\} \square (\square a \in \{a. \text{ev } a \in P^0\} \rightarrow P \text{ after } a) \rangle$

by (*simp add: deterministic-iff-maximal-for-leFD leFD-SKIPS-Det-Mprefix-After*)

lemma *deterministic-imp-initial-tick-iff-eq-SKIP* [*simp*] :

$\langle \text{determ } P \implies \checkmark(r) \in P^0 \iff P = \text{SKIP } r \rangle$

by (*meson deterministic-iff-maximal-for-leFD dual-order.refl initial-tick-iff-FD-SKIP*)

lemma *deterministic-imp-constraints-on-initials* :

$\langle \text{determ } P \implies P^0 = \{\} \vee \{a. \text{ev } a \in P^0\} = \{\} \wedge (\exists r. P^0 = \{\checkmark(r)\}) \vee \{a. \text{ev } a \in P^0\} \neq \{\} \wedge \{r. \checkmark(r) \in P^0\} = \{\} \rangle$

by *auto (metis deterministic-imp-initial-tick-iff-eq-SKIP event_{ptick}.exhaust initials-SKIP)*

corollary (in *After*) *deterministic-process-eq-SKIP-or-Mprefix-After* :
 $\langle \text{determ } P \implies P = (\text{if } \exists r. \checkmark(r) \in P^0 \text{ then SKIP (THE } r. P^0 = \{\checkmark(r)\})$
 $\quad \text{else } \Box a \in \{a. \text{ev } a \in P^0\} \rightarrow P \text{ after } a) \rangle$
by (*subst deterministic-process-eq-SKIPS-Det-Mprefix-After*)
(auto simp add: inj-on-eq-iff[OF inj-SKIP])

2.2.4 Characterization with After

lemma (in *AfterExt*) *deterministic-iff-accepts-initials-After_{trace}*:

$\langle \text{determ } P \longleftrightarrow (\forall t \in \mathcal{T} P. tF t \longrightarrow \text{determ}^0 (P \text{ after}_{\mathcal{T}} t)) \rangle$

proof (*intro iffI ballI impI*)

show $\langle \text{determ } P \implies t \in \mathcal{T} P \implies tF t \implies \text{determ}^0 (P \text{ after}_{\mathcal{T}} t) \rangle$ **for** t

by (*rule accepts-initialsI*)

(simp add: initials-def Refusals-iff T-After_{trace}-eq F-After_{trace}-eq deterministic-def)

next

show $\langle \text{determ } P \rangle$ **if** $\langle \forall t \in \mathcal{T} P. tF t \longrightarrow \text{determ}^0 (P \text{ after}_{\mathcal{T}} t) \rangle$

proof (*rule deterministicI*)

fix $t e$ **assume** $\langle t @ [e] \in \mathcal{T} P \rangle$

have $\langle t \in \mathcal{T} P \rangle$ **and** $\langle tF t \rangle$

by (*meson prefixI* $\langle t @ [e] \in \mathcal{T} P \rangle$ *is-processT3-TR*)

(use $\langle t @ [e] \in \mathcal{T} P \rangle$ *append-T-imp-tickFree* **in** *blast*)

with $\langle t \in \mathcal{T} P \rangle$ **that**[*rule-format, OF this*] **show** $\langle t @ [e] \in \mathcal{T} P \implies (t, \{e\}) \notin \mathcal{F} P \rangle$

by (*simp add: accepts-initials-def Refusals-iff initials-def T-After_{trace}-eq F-After_{trace}-eq*)

qed

qed

2.2.5 Operators preserving Determinism

lemma *deterministic-Mprefix-iff* :

$\langle \text{determ } (\Box a \in A \rightarrow P a) \longleftrightarrow (\forall a \in A. \text{determ } (P a)) \rangle$

by (*auto simp add: deterministic-def Mprefix-projs*) (*metis append-Cons*)

corollary *deterministic-write0-iff* : $\langle \text{determ } (a \rightarrow P) \longleftrightarrow \text{determ } P \rangle$

unfolding *write0-def* **by** (*simp add: deterministic-Mprefix-iff*)

corollary *deterministic-write-iff* : $\langle \text{determ } (c!a \rightarrow P) \longleftrightarrow \text{determ } P \rangle$

unfolding *write-def* **by** (*simp add: deterministic-Mprefix-iff*)

corollary *deterministic-inj-on-read-iff* :

$\langle \text{inj-on } c A \implies \text{determ } (c?a \in A \rightarrow P a) \longleftrightarrow (\forall a \in A. \text{determ } (P a)) \rangle$

unfolding *read-def* **by** (*simp add: deterministic-Mprefix-iff*)

lemma *deterministic-inj-Renaming* :

$\langle \text{determ } (\text{Renaming } P f g) \rangle$ **if** $\langle \text{inj } f \rangle \langle \text{inj } g \rangle \langle \text{determ } P \rangle$

proof (*rule deterministicI*)

have $\$: \langle inj (map\text{-}event_{ptick} f g) \rangle$ **by** (*simp add: event_{ptick}.inj-map that(1,2)*)
fix $t e$
assume $\langle t @ [e] \in \mathcal{T} (Renaming P f g) \rangle$
then obtain $t1$ **where** $*$: $\langle t1 \in \mathcal{T} P \rangle$ $\langle t @ [e] = map (map\text{-}event_{ptick} f g) t1 \rangle$
by (*simp add: T-Renaming deterministic-div-free[OF <determ P>]*) *blast*
have $\langle (s1, map\text{-}event_{ptick} f g - \{e\}) \in \mathcal{F} P \implies t \neq map (map\text{-}event_{ptick} f g) s1 \rangle$ **for** $s1$
proof (*rule ccontr, clarify*)
assume *assms* : $\langle (s1, map\text{-}event_{ptick} f g - \{e\}) \in \mathcal{F} P \rangle$ $\langle t = map (map\text{-}event_{ptick} f g) s1 \rangle$
from *assms*(2) $*(2)$ **have** $\langle t1 = s1 @ [inv (map\text{-}event_{ptick} f g) e] \rangle$
by (*cases t1 rule: rev-cases; simp*)
(metis (mono-tags, opaque-lifting) \$ inj-map-eq-map inv-f-f)
from *deterministicD*[*OF <deterministic P> *(1)[unfolded this]*]
have $\langle (s1, \{inv (map\text{-}event_{ptick} f g) e\}) \notin \mathcal{F} P \rangle$.
also have $\langle \{inv (map\text{-}event_{ptick} f g) e\} = map\text{-}event_{ptick} f g - \{e\} \rangle$
using *inj-vimage-singleton*[*OF \$, of e*] $*(2)$
 $\langle t1 = s1 @ [inv (map\text{-}event_{ptick} f g) e] \rangle$ **by** (*auto simp add: \$*)
finally have $\langle (s1, map\text{-}event_{ptick} f g - \{e\}) \notin \mathcal{F} P \rangle$.
with *assms*(1) **show** *False* **by** *simp*
qed
thus $\langle (t, \{e\}) \notin \mathcal{F} (Renaming P f g) \rangle$
by (*auto simp add: F-Renaming deterministic-div-free[OF <determ P>]*)
qed

lemma *deterministic-bij-Renaming-iff* :
 $\langle determ (Renaming P f g) \iff determ P \rangle$ **if** $\langle bij f \rangle$ **and** $\langle bij g \rangle$
proof (*rule iffI*)
show $\langle determ (Renaming P f g) \implies determ P \rangle$
by (*metis Renaming-inv bij-betw-def deterministic-iff-maximal-for-leFD mono-Renaming-FD <bij f> <bij g>*)
next
show $\langle determ P \implies determ (Renaming P f g) \rangle$
by (*simp add: bij-is-inj deterministic-inj-Renaming <bij f> <bij g>*)
qed

lemma *deterministic-Throw* : $\langle determ (P \Theta a \in A. Q a) \rangle$
if $\langle determ P \rangle$ $\langle \bigwedge a. a \in A \implies a \in \alpha(P) \implies determ (Q a) \rangle$
proof (*subst Throw-is-restrictable-on-events-of*)
show $\langle determ (Throw P (A \cap \alpha(P)) Q) \rangle$
proof (*rule deterministicI*)
fix $t e$ **assume** $\langle t @ [e] \in \mathcal{T} (P \Theta a \in (A \cap \alpha(P)). Q a) \rangle$
moreover from $\langle determ P \rangle$ **have** $\langle \mathcal{D} P = \{\} \rangle$
by (*simp add: deterministic-div-free*)
ultimately consider
 $(traceL) \langle t @ [e] \in \mathcal{T} P \rangle$ $\langle e \notin ev \text{ ' } (A \cap \alpha(P)) \rangle$ $\langle set t \cap ev \text{ ' } (A \cap \alpha(P)) = \{\} \rangle$

| (*traceR*) $t1$ a $t2$ **where** $\langle t @ [e] = t1 @ ev a \# t2 \rangle \langle t1 @ [ev a] \in \mathcal{T} P \rangle$
 $\langle set\ t1 \cap ev\ ' (A \cap \alpha(P)) = \{\} \rangle \langle a \in A \rangle \langle a \in \alpha(P) \rangle \langle t2 \in \mathcal{T} (Q\ a) \rangle$
unfolding *T-Throw* **by** *auto*
thus $\langle (t, \{e\}) \notin \mathcal{F} (P \Theta a \in (A \cap \alpha(P)). Q\ a) \rangle$
proof *cases*
case *traceL*
from *traceL(1)* $\langle determ\ P \rangle$ **have** $\langle (t, \{e\}) \notin \mathcal{F} P \rangle$
by (*simp add: deterministicD*)
with *traceL(3)* **show** $\langle (t, \{e\}) \notin \mathcal{F} (P \Theta a \in (A \cap \alpha(P)). Q\ a) \rangle$
by (*auto simp add: F-Throw* $\langle \mathcal{D} P = \{\} \rangle$)
next
case *traceR*
from *traceR(1)* **consider** $\langle t2 = [] \rangle \langle t = t1 \rangle \langle e = ev\ a \rangle$
| $t2'$ **where** $\langle t2 = t2' @ [e] \rangle \langle t = t1 @ ev\ a \# t2' \rangle$
by (*cases t2 rule: rev-cases*) *simp-all*
thus $\langle (t, \{e\}) \notin \mathcal{F} (P \Theta a \in (A \cap \alpha(P)). Q\ a) \rangle$
proof *cases*
assume $\langle t2 = [] \rangle \langle t = t1 \rangle \langle e = ev\ a \rangle$
from $\langle determ\ P \rangle$ *traceR(2)* **have** $\langle (t1, \{ev\ a\}) \notin \mathcal{F} P \rangle$
by (*simp add: deterministicD*)
with *traceR(3)* **show** $\langle (t, \{e\}) \notin \mathcal{F} (P \Theta a \in (A \cap \alpha(P)). Q\ a) \rangle$
by (*auto simp add:* $\langle t = t1 \rangle \langle e = ev\ a \rangle \langle \mathcal{D} P = \{\} \rangle$ *F-Throw*)
next
fix $t2'$ **assume** $\langle t2 = t2' @ [e] \rangle \langle t = t1 @ ev\ a \# t2' \rangle$
from *traceR(4, 5)* **have** $\langle determ\ (Q\ a) \rangle$
by (*rule* $\langle \bigwedge a. a \in A \implies a \in \alpha(P) \implies determ\ (Q\ a) \rangle$)
with $\langle t2 = t2' @ [e] \rangle$ **have** $\langle (t2', \{e\}) \notin \mathcal{F} (Q\ a) \rangle$
using *traceR(6)* **by** (*simp add: deterministicD*)
with *traceR(3-5)* **show** $\langle (t, \{e\}) \notin \mathcal{F} (P \Theta a \in (A \cap \alpha(P)). Q\ a) \rangle$
by (*simp add:* $\langle t = t1 @ ev\ a \# t2' \rangle \langle \mathcal{D} P = \{\} \rangle$ *F-Throw Throw-T-third-clause-breaker*)
qed
qed
qed
qed

lemma *T-snoc-tick-imp-no-continuation-if-deterministic* :

$\langle u = [] \wedge e = \checkmark(r) \rangle$ **if** $\langle determ\ P \rangle \langle t @ u @ [e] \in \mathcal{T} P \rangle \langle t @ [\checkmark(r)] \in \mathcal{T} P \rangle$

proof –

have $*$: $\langle t @ [e] \in \mathcal{T} P \implies e = \checkmark(r) \rangle$ **for** e

by (*metis deterministicD is-processT6-TR-notin singletonD that(1, 3)*)

show $\langle u = [] \wedge e = \checkmark(r) \rangle$

proof (*cases u*)

from $*$ *that(2)* **show** $\langle u = [] \implies u = [] \wedge e = \checkmark(r) \rangle$ **by** *auto*

next

fix $e' u'$

assume $\langle u = e' \# u' \rangle$

with *that(2)* **have** $\langle t @ [e'] \in \mathcal{T} P \rangle$

by *simp* (*metis* *F-T T-F append.assoc append-Cons append-Nil is-processT3*)
 hence *False*
 by (*metis* * *T-imp-front-tickFree* $\langle u = e' \# u' \rangle$ *event_{ptick}.disc(2)* *front-tickFree-append-iff*
not-Cons-self snoc-eq-iff-butlast that(2) *tickFree-Cons-iff*)
 thus $\langle u = [] \wedge e = \checkmark(r) \rangle$ by *simp*
 qed
 qed

lemma *T-snoc-ev-imp-no-tick-continuation-if-deterministic* :
 $\langle u \neq [] \wedge \text{is-ev } (hd\ u) \vee \text{is-ev } e \rangle$ **if** $\langle \text{determ } P \rangle$ $\langle t @ u @ [e] \in \mathcal{T} P \rangle$ $\langle t @ [ev\ a] \in \mathcal{T} P \rangle$
proof –
 have $\langle t @ [e] \in \mathcal{T} P \implies \text{is-ev } e \rangle$ **for** *e*
 by (*metis* *T-snoc-tick-imp-no-continuation-if-deterministic append.left-neutral*
event_{ptick}.discI(1) *event_{ptick}.exhaust that(1,3)*)
 thus $\langle u \neq [] \wedge \text{is-ev } (hd\ u) \vee \text{is-ev } e \rangle$
 by (*metis* *T-imp-front-tickFree append-eq-appendI front-tickFree-append-iff*
list.exhaust-sel not-Cons-self snoc-eq-iff-butlast that(2) *tickFree-Cons-iff*)
 qed

lemma *deterministic-Seq_{ptick}* : $\langle \text{determ } (P ; \checkmark Q) \rangle$
if $\langle \text{determ } P \rangle$ $\langle \bigwedge r. r \in \checkmark s(P) \implies \text{determ } (Q\ r) \rangle$
proof (*rule deterministicI*)
fix *t e* **assume** $\langle t @ [e] \in \mathcal{T} (P ; \checkmark Q) \rangle$
moreover from $\langle \text{determ } P \rangle$ **have** $\langle \mathcal{D} P = \{ \} \rangle$
by (*simp add: deterministic-div-free*)
ultimately consider *a u* **where** $\langle e = ev\ a \rangle$ $\langle t = \text{map } (ev \circ of\ ev)\ u \rangle$ $\langle u @ [ev\ a] \in \mathcal{T} P \rangle$ $\langle tF\ u \rangle$
 $| u\ v\ r$ **where** $\langle t @ [e] = \text{map } (ev \circ of\ ev)\ u @ v \rangle$ $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$ $\langle tF\ u \rangle$
 $\langle v \in \mathcal{T} (Q\ r) \rangle$ $\langle v \neq [] \rangle$
by (*auto simp add: Seq_{ptick}-projs append-eq-map-conv append-eq-append-conv2*
Cons-eq-append-conv)
(metis Nil-is-append-conv append.right-neutral append-Nil not-Cons-self, blast,
metis Cons-eq-appendI append-eq-appendI eq-Nil-appendI event_{ptick}.collapse(1)
is-processT3-TR-append)
thus $\langle (t, \{e\}) \notin \mathcal{F} (P ; \checkmark Q) \rangle$
proof cases
show $\langle e = ev\ a \implies t = \text{map } (ev \circ of\ ev)\ u \implies u @ [ev\ a] \in \mathcal{T} P \implies tF\ u \implies (t, \{e\}) \notin \mathcal{F} (P ; \checkmark Q) \rangle$ **for** *a u*
by (*auto simp add: tickFree-map-ev-of-ev-inj* $\langle \mathcal{D} P = \{ \} \rangle$ *Seq_{ptick}-projs*
ref-Seq_{ptick}-def map-eq-append-conv)
(meson deterministicD empty-subsetI insertI1 insert-subset is-processT4
 $\langle \text{determ } P \rangle,$
meson T-snoc-tick-imp-no-continuation-if-deterministic event_{ptick}.distinct(1)
 $\langle \text{determ } P \rangle$)
next
fix *u v r* **assume** $\langle t @ [e] = \text{map } (ev \circ of\ ev)\ u @ v \rangle$ $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$ $\langle tF\ u \rangle$

$u \rangle \langle v \in \mathcal{T} (Q \ r) \rangle \langle v \neq [] \rangle$
from $\text{this}(1, 5)$ **obtain** v' **where** $\langle v = v' @ [e] \rangle \langle t = \text{map} (ev \circ \text{of-ev}) u @ v' \rangle$
by (cases v rule: *rev-cases*) *simp-all*
from $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$ *T-snoc-tick-imp-no-continuation-if-deterministic*[*OF*
 $\langle \text{determ} P \rangle$]
have $*$: $\langle \text{map} (ev \circ \text{of-ev}) u @ v' = \text{map} (ev \circ \text{of-ev}) w @ x \wedge w @ [\checkmark(s)] \in$
 $\mathcal{T} P \implies w = u \wedge s = r \rangle$ **for** $w \ x \ s$
by (*auto simp add: append-eq-append-conv2 map-eq-append-conv append-eq-map-conv*
append-T-imp-tickFree
 $\text{dest! : tickFree-map-ev-of-ev-inj}[\text{THEN iffD1, rotated 2}]$) *blast+*
have $\langle \text{determ} (Q \ r) \rangle$
by (*metis* $\langle \mathcal{D} P = \{\} \rangle \langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$ *empty-iff strict-ticks-of-memI*
that(2))
with $\langle v = v' @ [e] \rangle \langle v \in \mathcal{T} (Q \ r) \rangle$
have $\langle (v', \{e\}) \notin \mathcal{F} (Q \ r) \rangle$ **by** (*simp add: deterministicD*)
{ fix v'' **assume** $\langle (u @ v'', \text{ref-Seq}_{\text{ptick}} \{e\}) \in \mathcal{F} P \rangle \langle \text{tF } v'' \rangle \langle v' = \text{map} (ev \circ$
 $\text{of-ev}) v'' \rangle$
have $\langle v'' = [] \rangle$
by (*metis F-T F-imp-front-tickFree T-snoc-tick-imp-no-continuation-if-deterministic*
 $\langle (u @ v'', \text{ref-Seq}_{\text{ptick}} \{e\}) \in \mathcal{F} P \rangle$
 $\langle \text{tF } v'' \rangle \langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$ *front-tickFree-charn front-tickFree-nonempty-append-imp*
non-tickFree-tick $\langle \text{determ} P \rangle$)
with $\langle (u @ v'', \text{ref-Seq}_{\text{ptick}} \{e\}) \in \mathcal{F} P \rangle$ **have** $\langle (u, \{\checkmark(r)\}) \in \mathcal{F} P \rangle$
by (*simp add: ref-Seq_{ptick}-def*)
(meson UNIV-I UnCI empty-subsetI insert-subset is-processT4 rev-image-eqI)
hence *False* **by** (*simp add:* $\langle u @ [\checkmark(r)] \in \mathcal{T} P \rangle$ *deterministicD* $\langle \text{determ} P \rangle$)
}
with $*$ $\langle (v', \{e\}) \notin \mathcal{F} (Q \ r) \rangle$ **show** $\langle (t, \{e\}) \notin \mathcal{F} (P ; \checkmark Q) \rangle$
by (*auto simp add: Seq_{ptick}-projs* $\langle \mathcal{D} P = \{\} \rangle \langle t = \text{map} (ev \circ \text{of-ev}) u @ v' \rangle$
append-eq-map-conv $\langle \text{tF } u \rangle$
 $\text{dest! : tickFree-map-ev-of-ev-inj}[\text{THEN iffD1, rotated 2}])$ +)

qed
qed

corollary *deterministic-Seq* : $\langle \text{determ} P \implies \text{determ} Q \implies \text{determ} (P ; Q) \rangle$
by (*metis Seq_{ptick}-const deterministic-Seq_{ptick}*)

lemma (in *After*) *initial-imp-deterministic-After*:
 $\langle ev \ e \in P^0 \implies \text{determ} P \implies \text{determ} (P \ \text{after} \ e) \rangle$
unfolding *deterministic-def* **by** (*simp add: After-projs*)

lemma (in *AfterExt*) *initial-imp-deterministic-After_{tick}*:
 $\langle e \in P^0 \implies (\text{case } e \text{ of } \checkmark(r) \Rightarrow \text{determ} (\Omega \ P \ r)) \implies$
 $\text{determ} P \implies \text{determ} (P \ \text{after}_{\checkmark} \ e) \rangle$
unfolding *deterministic-def* **by** (*cases e*) (*simp-all add: T-After_{tick} F-After_{tick}*)

2.2.6 Operators not (always) preserving Determinism

lemma *deterministic-imp-accepts-initials* : $\langle \text{determ } P \implies \text{determ}^0 P \rangle$
 by (*simp add: Refusals-iff accepts-initials-def deterministic-def initials-def*)

corollary *deterministic-SKIPS-iff* : $\langle \text{determ } (\text{SKIPS } R) \iff R = \{\} \vee (\exists r. R = \{r\}) \rangle$

by (*metis SKIPS-empty SKIPS-singl-is-SKIP accepts-initials-SKIPS-iff deterministic-SKIP deterministic-STOP deterministic-imp-accepts-initials*)

lemma *deterministic-Det*:

$\langle \text{determ } P \implies \text{determ } Q \implies$
 $\text{range tick} \cap P^0 \cap Q^0 \neq \{\} \vee P^0 \cap Q^0 = \{\} \wedge \text{range tick} \cap (P^0 \cup Q^0) = \{\}$
 $\implies \text{determ } (P \square Q) \rangle$

proof (*elim disjE conjE*)

show $\langle \text{determ } P \implies \text{determ } Q \implies \text{range tick} \cap P^0 \cap Q^0 \neq \{\} \implies \text{determ } (P \square Q) \rangle$

by (*auto simp add: deterministic-def Det-projs SKIP-projs*)

next

show $\langle \llbracket \text{determ } P; \text{determ } Q; P^0 \cap Q^0 = \{\}; \text{range tick} \cap (P^0 \cup Q^0) = \{\} \rrbracket \implies \text{determ } (P \square Q) \rangle$

by (*auto simp add: deterministic-def Det-projs disjoint-iff deterministic-div-free initials-def*)

(*metis F-T append-Cons append-Nil is-processT3-TR-append neq-Nil-conv*) +

qed

2.3 Application to Operational Semantics

lemma (*in OpSemFD*) *tickFree-trace-trans-preserves-deterministic*:

$\langle (P :: \langle 'a, 'r \rangle \text{process}_{\text{ptick}}) \text{FD} \rightsquigarrow^* t Q \implies \text{tF } t \implies \text{deterministic } P \implies \text{deterministic } Q \rangle$

proof (*induct rule: trace-trans.induct*)

show $\langle P \text{FD} \rightsquigarrow_{\tau} P' \implies \text{deterministic } P \implies \text{deterministic } P' \rangle$ **for** $P P' :: \langle 'a, 'r \rangle \text{process}_{\text{ptick}} \rangle$

using *deterministic-iff-maximal-for-leFD* by *blast*

next

show $\langle \text{tF } [\checkmark(r)] \implies \text{deterministic } P \rangle$

for $r :: \langle 'r \rangle$ **and** $P :: \langle 'a, 'r \rangle \text{process}_{\text{ptick}} \rangle$ **by** *simp*

next

fix $a P P'$ **and** $t :: \langle 'a, 'r \rangle \text{trace}_{\text{ptick}} \rangle$ **and** $P'' :: \langle 'a, 'r \rangle \text{process}_{\text{ptick}} \rangle$

assume $\langle P \text{FD} \rightsquigarrow_a P' \rangle \langle \text{tF } (ev a \# t) \rangle \langle \text{deterministic } P \rangle$

$\langle \text{tF } t \implies \text{deterministic } P' \implies \text{deterministic } P'' \rangle$

from $\langle P \text{FD} \rightsquigarrow_a P' \rangle \langle \text{deterministic } P \rangle$ **have** $\langle \text{deterministic } P' \rangle$

using *deterministic-iff-maximal-for-leFD ev-trans-is initial-imp-deterministic-After*

by *blast*

with $\langle \text{tF } t \implies \text{deterministic } P' \implies \text{deterministic } P'' \rangle \langle \text{tickFree } (ev a \# t) \rangle$

show $\langle \text{deterministic } P'' \rangle$ **by** *simp*
qed

lemma *deterministic-imp-Refusals-iff*: $\langle \text{deterministic } P \implies X \in \mathcal{R} P \iff X \cap P^0 = \{\} \rangle$

by (*auto simp add: Refusals-iff initials-def deterministic-def disjoint-iff*)
(metis append-Nil empty-subsetI insert-subset process-charn,
metis Nil-elem-T append-Nil is-processT5-S7)

lemma (**in** *OpSemFD*) *deterministic-F-trace-trans-reality-check*:

$\langle \text{deterministic } P \implies tF t \implies$
 $(t, X) \in \mathcal{F} (P :: ('a, 'r) \text{process}_{ptick}) \iff (\exists Q. (P \text{ }_{FD} \rightsquigarrow^* t Q) \wedge X \cap Q^0 = \{\}) \rangle$

by (*simp add: F-trace-trans-reality-check*)
(metis deterministic-imp-Refusals-iff tickFree-trace-trans-preserves-deterministic)

lemma $\langle \neg \text{deterministic } ((a \rightarrow \text{SKIP undefined}) \square \text{SKIP undefined}) \rangle$

by (*metis Det-commute FD-iff-eq-Ndet Sliding-SKIP Sliding-def Un-insert-right initials-write0 insertI1*
singletonD deterministic-iff-maximal-for-leFD event_{ptick}.simps(4) initials-Det initials-SKIP)

Chapter 3

ProcOmata: Functional Automata embedded into CSP Processes

We will often have to perform induction on both the list of automata and the list of states, provided that they have the same length.

lemma *induct-2-lists012* [*consumes 1, case-names Nil single Cons*] :
 $\langle \llbracket \text{length } xs = \text{length } ys; P \ [] \ []; \bigwedge x1 \ y1. P [x1] [y1];$
 $\bigwedge x1 \ x2 \ xs \ y1 \ y2 \ ys. \text{length } xs = \text{length } ys \implies P \ xs \ ys \implies$
 $P (x2 \# \ xs) (y2 \# \ ys) \implies P (x1 \# \ x2 \# \ xs) (y1 \# \ y2 \# \ ys) \rrbracket$
 $\implies P \ xs \ ys \rangle$
 by (*induct xs arbitrary: ys rule: induct-list012*)
 (*auto simp add: Suc-length-conv length-Suc-conv*)

lemma *nat-induct-012* [*case-names 0 1 2 Suc*]:
 $\langle \llbracket P \ 0; P (Suc \ 0); P (Suc (Suc \ 0)); \bigwedge k. Suc (Suc \ 0) \leq k \implies P \ k \implies P (Suc$
 $k) \rrbracket \implies P \ n \rangle$
 by (*metis One-nat-def Suc-1 less-2-cases-iff linorder-not-le nat-induct*)

The following results will be moved to `Restriction_Spaces` in the future.

lemma *restriction-shift-iterated* :
 $\langle \text{restriction-shift } (f \ \hat{\sim} \ k) \ (int \ k \ * \ m) \rangle$
 if $\langle \text{restriction-shift } f \ m \rangle$ **for** $f :: \langle 'a \Rightarrow 'a :: \text{restriction-space} \rangle$
proof (*induct k*)
 show $\langle \text{restriction-shift } (f \ \hat{\sim} \ 0) \ (int \ 0 \ * \ m) \rangle$
 by (*simp add: restriction-shiftI*)
next
 fix k **assume** $*$: $\langle \text{restriction-shift } (f \ \hat{\sim} \ k) \ (int \ k \ * \ m) \rangle$
 have $\langle \text{restriction-shift } (f \ \hat{\sim} \ Suc \ k) \ (int \ (Suc \ k) \ * \ m) \longleftrightarrow$
 $\text{restriction-shift } (\lambda x. f ((f \ \hat{\sim} \ k) \ x)) \ (int \ k \ * \ m + \ m) \rangle$
 by (*simp add: comp-def distrib-left mult.commute add.commute*)
 also have \dots **by** (*fact restriction-shift-comp-restriction-shift[OF that *]*)
 finally show $\langle \text{restriction-shift } (f \ \hat{\sim} \ Suc \ k) \ (int \ (Suc \ k) \ * \ m) \rangle .$

qed

lemma *non-destructive-iterated* :

$\langle \text{non-destructive } f \implies \text{non-destructive } (f \text{ } \overset{\sim}{\sim} k) \rangle$

for $f :: \langle 'a \Rightarrow 'a :: \text{restriction-space} \rangle$

by (*metis mult.commute mult-zero-left non-destructive-def non-destructive-on-def restriction-shift-def restriction-shift-iterated*)

lemma *constructive-iterated* :

$\langle \text{constructive } (f \text{ } \overset{\sim}{\sim} k) \rangle$ **if** $\langle 0 < k \rangle$ $\langle \text{constructive } f \rangle$

for $f :: \langle 'a \Rightarrow 'a :: \text{restriction-space} \rangle$

proof –

from $\langle \text{constructive } f \rangle$ **have** $\langle \text{restriction-shift } f 1 \rangle$

unfolding *constructive-def constructive-on-def restriction-shift-def* **by** *blast*

with *restriction-shift-iterated*

have $\langle \text{restriction-shift } (f \text{ } \overset{\sim}{\sim} k) (\text{int } k * 1) \rangle$.

hence $\langle \text{restriction-shift } (f \text{ } \overset{\sim}{\sim} k) (\text{int } k) \rangle$ **by** *simp*

with $\langle 0 < k \rangle$ **show** $\langle \text{constructive } (f \text{ } \overset{\sim}{\sim} k) \rangle$

by (*metis One-nat-def constructive-def constructive-on-def*

less-eq-Suc-le nat-int-comparison(3) of-nat-1-eq-iff

restriction-shift-def restriction-shift-imp-restriction-shift-le)

qed

lemma *restriction-fix-unique-iterated* :

$\langle [0 < k; \text{constructive } f; (f \text{ } \overset{\sim}{\sim} k) x = x] \implies (v x. f x) = x \rangle$

by (*metis constructive-iterated funpow-swap1 restriction-fix-unique*)

lemma *restriction-fix-iterated* :

$\langle 0 < k \implies \text{constructive } f \implies (v x. (f \text{ } \overset{\sim}{\sim} k) x) = (v x. f x) \rangle$

by (*metis constructive-iterated restriction-fix-eq restriction-fix-unique-iterated*)

corollary *restriction-fix-ind-iterated*

[*consumes 1, case-names constructive adm base step*]:

$\langle P (v x. f x) \rangle$ **if** $\langle 0 < k \rangle$ $\langle \text{constructive } f \rangle$ $\langle \text{adm}_\downarrow P \rangle$ $\langle P x \rangle$ $\langle \bigwedge x. P x \implies P ((f \text{ } \overset{\sim}{\sim} k) x) \rangle$

proof –

from *constructive-iterated that(1, 2)* **have** $\langle \text{constructive } (f \text{ } \overset{\sim}{\sim} k) \rangle$.

from *restriction-fix-ind[OF this that(3–5)]* **have** $\langle P (v x. (f \text{ } \overset{\sim}{\sim} k) x) \rangle$.

also from *restriction-fix-iterated that(1, 2)* **have** $\langle (v x. (f \text{ } \overset{\sim}{\sim} k) x) = (v x. f x) \rangle$

.

finally show $\langle P (v x. f x) \rangle$.

qed

3.1 Definitions

3.1.1 Non-deterministic and deterministic Automata

unbundle *option-type-syntax*

type-synonym $\langle 'a, 'b \rangle \text{ enabl} = \langle 'a \Rightarrow 'b \text{ set} \rangle$

type-synonym $\langle 'a, 'b, 'c \rangle \text{ trans} = \langle 'a \Rightarrow 'b \Rightarrow 'c \rangle$

type-synonym $\langle 'a, 'b \rangle \text{ trans}_d = \langle ('a, 'b \text{ option}) \text{ trans} \rangle$

type-synonym $\langle 'a, 'b \rangle \text{ trans}_{nd} = \langle ('a, 'b \text{ set}) \text{ trans} \rangle$

record $\langle 'a, 'b, 'c, 'd \rangle A =$

$\tau :: \langle ('a, 'b, 'c) \text{ trans} \rangle$

$\omega :: \langle 'a \Rightarrow 'd \rangle$

type-synonym $\langle 'a, 'b, 'c \rangle A_d = \langle ('a, 'b, 'c \text{ option}, 'd \text{ option}) A \rangle$

type-synonym $\langle 'a, 'b, 'c, 'd, 'e \rangle A_d\text{-scheme} = \langle ('a, 'b, 'c \text{ option}, 'd \text{ option}, 'e) A\text{-scheme} \rangle$

type-synonym $\langle 'a, 'b, 'c \rangle A_{nd} = \langle ('a, 'b, 'c \text{ set}, 'd \text{ set}) A \rangle$

type-synonym $\langle 'a, 'b, 'c, 'd, 'e \rangle A_{nd}\text{-scheme} = \langle ('a, 'b, 'c \text{ set}, 'd \text{ set}, 'e) A\text{-scheme} \rangle$

3.1.2 Enableness

consts $\varepsilon :: \langle ('a, 'b, 'c, 'd, 'e) A\text{-scheme} \Rightarrow ('a, 'b) \text{ enabl} \rangle$

overloading

$\varepsilon_d \equiv \langle \varepsilon :: ('a, 'b, 'c \text{ option}, 'd, 'e) A\text{-scheme} \Rightarrow ('a, 'b) \text{ enabl} \rangle$

$\varepsilon_{nd} \equiv \langle \varepsilon :: ('a, 'b, 'c \text{ set}, 'd, 'e) A\text{-scheme} \Rightarrow ('a, 'b) \text{ enabl} \rangle$

begin

fun $\varepsilon_d :: \langle ('a, 'b, 'c \text{ option}, 'd, 'e) A\text{-scheme} \Rightarrow ('a, 'b) \text{ enabl} \rangle$

where $\langle \varepsilon_d A \sigma = \{a. \tau A \sigma a \neq \Diamond\} \rangle$

fun $\varepsilon_{nd} :: \langle ('a, 'b, 'c \text{ set}, 'd, 'e) A\text{-scheme} \Rightarrow ('a, 'b) \text{ enabl} \rangle$

where $\langle \varepsilon_{nd} A \sigma = \{a. \tau A \sigma a \neq \{\}\} \rangle$

end

lemmas $\varepsilon\text{-simps}[simp del] = \varepsilon_d.\text{simps } \varepsilon_{nd}.\text{simps}$

3.1.3 States allowing Termination

consts $\varrho :: \langle ('a, 'b, 'c, 'd, 'e) A\text{-scheme} \Rightarrow 'a \text{ set} \rangle$

overloading

$\varrho_d \equiv \langle \varrho :: ('a, 'b, 'c, 'd \text{ option}, 'e) A\text{-scheme} \Rightarrow 'a \text{ set} \rangle$

$\varrho_{nd} \equiv \langle \varrho :: ('a, 'b, 'c, 'd \text{ set}, 'e) A\text{-scheme} \Rightarrow 'a \text{ set} \rangle$

begin

fun $\varrho_d :: \langle ('a, 'b, 'c, 'd \text{ option}, 'e) A\text{-scheme} \Rightarrow 'a \text{ set} \rangle$

where $\langle \varrho_d A = \{\sigma. \omega A \sigma \neq \Diamond\} \rangle$

fun $\varrho_{nd} :: \langle ('a, 'b, 'c, 'd \text{ set}, 'e) A\text{-scheme} \Rightarrow 'a \text{ set} \rangle$

where $\langle \varrho_{nd} A = \{\sigma. \omega A \sigma \neq \{\}\} \rangle$

end

lemmas $\varrho\text{-simps}[simp del] = \varrho_d.\text{simps } \varrho_{nd}.\text{simps}$

3.1.4 Reachability

inductive-set $\mathcal{R}_d :: \langle ('σ, 'a, 'r, 'α) A_d\text{-scheme} \Rightarrow 'σ \Rightarrow 'σ \text{ set} \rangle$
for $A :: \langle ('σ, 'a, 'r, 'α) A_d\text{-scheme} \rangle$ **and** $σ :: 'σ$
where $init : \langle σ \in \mathcal{R}_d A σ \rangle$
 $| \quad step : \langle σ' \in \mathcal{R}_d A σ \Longrightarrow [\sigma''] = \tau A σ' a \Longrightarrow σ'' \in \mathcal{R}_d A σ \rangle$

inductive-set $\mathcal{R}_{nd} :: \langle ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \Rightarrow 'σ \Rightarrow 'σ \text{ set} \rangle$
for $A :: \langle ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \rangle$ **and** $σ :: 'σ$
where $init : \langle σ \in \mathcal{R}_{nd} A σ \rangle$
 $| \quad step : \langle σ' \in \mathcal{R}_{nd} A σ \Longrightarrow σ'' \in \tau A σ' a \Longrightarrow σ'' \in \mathcal{R}_{nd} A σ \rangle$

lemma $\mathcal{R}_d\text{-trans}$: $\langle σ'' \in \mathcal{R}_d A σ' \Longrightarrow σ' \in \mathcal{R}_d A σ \Longrightarrow σ'' \in \mathcal{R}_d A σ \rangle$
by (*induct rule*: $\mathcal{R}_d.induct$, *simp add*: $\mathcal{R}_d.init$) (*meson* $\mathcal{R}_d.step$)

lemma $\mathcal{R}_{nd}\text{-trans}$: $\langle σ'' \in \mathcal{R}_{nd} A σ' \Longrightarrow σ' \in \mathcal{R}_{nd} A σ \Longrightarrow σ'' \in \mathcal{R}_{nd} A σ \rangle$
by (*induct rule*: $\mathcal{R}_{nd}.induct$, *simp add*: $\mathcal{R}_{nd}.init$) (*meson* $\mathcal{R}_{nd}.step$)

3.1.5 Morphisms

Our morphisms are defined considering that, except from τ , the fields remain unchanged.

definition $from\text{-det}\text{-to}\text{-ndet} ::$
 $\langle ('σ, 'a, 'r, 'α) A_d\text{-scheme} \Rightarrow ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \rangle$
where $\langle from\text{-det}\text{-to}\text{-ndet} A \equiv$
 $\langle \tau = \lambda \sigma a. \text{case } \tau A \sigma a \text{ of } [\sigma'] \Rightarrow \{\sigma'\} \mid \diamond \Rightarrow \{\},$
 $\omega = \lambda \sigma. \text{case } \omega A \sigma \text{ of } [r] \Rightarrow \{r\} \mid \diamond \Rightarrow \{\}, \dots = \text{more } A \rangle \rangle$

definition $from\text{-ndet}\text{-to}\text{-det} ::$
 $\langle ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \Rightarrow ('σ, 'a, 'r, 'α) A_d\text{-scheme} \rangle$
where $\langle from\text{-ndet}\text{-to}\text{-det} A \equiv$
 $\langle \tau = \lambda \sigma a. \text{if } \tau A \sigma a = \{\} \text{ then } \diamond \text{ else } [THE \sigma'. \sigma' \in \tau A \sigma a],$
 $\omega = \lambda \sigma. \text{if } \omega A \sigma = \{\} \text{ then } \diamond \text{ else } [THE r. r \in \omega A \sigma], \dots = \text{more } A \rangle \rangle$

definition $from\text{-}\sigma\text{-to}\text{-}\sigma_s ::$
 $\langle ('σ, 'a, 'r, 'α) A_d\text{-scheme} \Rightarrow ('σ \text{ list}, 'a, 'r, 'α) A_d\text{-scheme} \rangle$
where $\langle from\text{-}\sigma\text{-to}\text{-}\sigma_s A \equiv$
 $\langle \tau = \lambda \sigma s a. \text{case } \tau A (hd \sigma s) a \text{ of } [\sigma'] \Rightarrow [[\sigma']] \mid \diamond \Rightarrow \diamond,$
 $\omega = \lambda \sigma s. \omega A (hd \sigma s), \dots = \text{more } A \rangle \rangle$

definition $from\text{-}\sigma\text{-to}\text{-}\sigma_{nd} ::$
 $\langle ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \Rightarrow ('σ \text{ list}, 'a, 'r, 'α) A_{nd}\text{-scheme} \rangle$
where $\langle from\text{-}\sigma\text{-to}\text{-}\sigma_{nd} A \equiv$
 $\langle \tau = \lambda \sigma s a. \{[\sigma'] \mid \sigma'. \sigma' \in \tau A (hd \sigma s) a\},$
 $\omega = \lambda \sigma s. \omega A (hd \sigma s), \dots = \text{more } A \rangle \rangle$

definition $from\text{-}\sigma s\text{-to}\text{-}\sigma_d ::$
 $\langle ('σ \text{ list}, 'a, 'r, 'α) A_d\text{-scheme} \Rightarrow ('σ, 'a, 'r, 'α) A_d\text{-scheme} \rangle$
where $\langle from\text{-}\sigma s\text{-to}\text{-}\sigma_d A \equiv$
 $\langle \tau = \lambda \sigma a. \text{case } \tau A [\sigma] a \text{ of } [\sigma s'] \Rightarrow [hd \sigma s'] \mid \diamond \Rightarrow \diamond,$

$\omega = \lambda\sigma. \omega A [\sigma], \dots = \text{more } A \rangle$

definition *from- σ s-to- σ_{nd}* ::

$\langle (' \sigma \text{ list}, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \Rightarrow (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \rangle$

where $\langle \text{from-}\sigma\text{s-to-}\sigma_{nd} A \equiv$

$(\tau = \lambda\sigma a. \{hd \sigma s' \mid \sigma s' \in \tau A [\sigma] a\},$

$\omega = \lambda\sigma. \omega A [\sigma], \dots = \text{more } A \rangle$

definition *from-singl-to-list_d* ::

$\langle (' \sigma, 'a, 'r, ' \alpha) A_d\text{-scheme} \Rightarrow (' \sigma \text{ list}, 'a, 'r \text{ list}, ' \alpha) A_d\text{-scheme} \rangle$

where $\langle \text{from-singl-to-list}_d A \equiv$

$(\tau = \lambda\sigma s a. \text{case } \tau A (hd \sigma s) a \text{ of } [\sigma'] \Rightarrow [[\sigma']] \mid \diamond \Rightarrow \diamond,$

$\omega = \lambda\sigma s. \text{case } \omega A (hd \sigma s) \text{ of } [r] \Rightarrow [[r]] \mid \diamond \Rightarrow \diamond, \dots = \text{more } A \rangle$

definition *from-singl-to-list_{nd}* ::

$\langle (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \Rightarrow (' \sigma \text{ list}, 'a, 'r \text{ list}, ' \alpha) A_{nd}\text{-scheme} \rangle$

where $\langle \text{from-singl-to-list}_{nd} A \equiv$

$(\tau = \lambda\sigma s a. \{[\sigma'] \mid \sigma' \in \tau A (hd \sigma s) a\},$

$\omega = \lambda\sigma s. \{[r] \mid r \in \omega A (hd \sigma s)\}, \dots = \text{more } A \rangle$

definition *from-list-to-singl_d* ::

$\langle (' \sigma \text{ list}, 'a, 'r \text{ list}, ' \alpha) A_d\text{-scheme} \Rightarrow (' \sigma, 'a, 'r, ' \alpha) A_d\text{-scheme} \rangle$

where $\langle \text{from-list-to-singl}_d A \equiv$

$(\tau = \lambda\sigma a. \text{case } \tau A [\sigma] a \text{ of } [\sigma s'] \Rightarrow [hd \sigma s'] \mid \diamond \Rightarrow \diamond,$

$\omega = \lambda\sigma. \text{case } \omega A [\sigma] \text{ of } [rs] \Rightarrow [hd rs] \mid \diamond \Rightarrow \diamond, \dots = \text{more } A \rangle$

definition *from-list-to-singl_{nd}* ::

$\langle (' \sigma \text{ list}, 'a, 'r \text{ list}, ' \alpha) A_{nd}\text{-scheme} \Rightarrow (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \rangle$

where $\langle \text{from-list-to-singl}_{nd} A \equiv$

$(\tau = \lambda\sigma a. \{hd \sigma s' \mid \sigma s' \in \tau A [\sigma] a\},$

$\omega = \lambda\sigma. \{hd rs \mid rs \in \omega A [\sigma]\}, \dots = \text{more } A \rangle$

lemmas *det-ndet-conv-defs* = *from-det-to-ndet-def from-ndet-to-det-def*

and *σ - σ s-conv-defs* = *from- σ -to- σs_d -def from- σ -to- σs_{nd} -def*

from- σ s-to- σ_d -def from- σ s-to- σ_{nd} -def

and *singl-list-conv-defs* = *from-singl-to-list_d-def from-singl-to-list_{nd}-def*

from-list-to-singl_d-def from-list-to-singl_{nd}-def

bundle *functional-automata-morphisms-syntax* **begin**

notation *from-det-to-ndet* $\langle \langle \langle - \rangle \rangle_d \hookrightarrow_{nd} \rangle [0]$

notation *from-ndet-to-det* $\langle \langle \langle - \rangle \rangle_{nd} \rightsquigarrow_d \rangle [0]$

notation *from- σ -to- σs_d* $\langle \langle \langle - \rangle \rangle_{\sigma \hookrightarrow \sigma s} \rangle [0]$

notation *from- σ -to- σs_{nd}* $\langle \langle \langle \langle - \rangle \rangle_{\sigma \hookrightarrow \sigma s} \rangle \rangle [0]$

notation *from- σ s-to- σ_d* $\langle \langle \langle \langle - \rangle \rangle_{\sigma s \rightsquigarrow \sigma} \rangle \rangle [0]$

notation *from- σ s-to- σ_{nd}* $\langle \langle \langle \langle \langle - \rangle \rangle_{\sigma s \rightsquigarrow \sigma} \rangle \rangle \rangle [0]$

notation *from-singl-to-list_d* $\langle \langle \langle \langle - \rangle \rangle_{singl \hookrightarrow list} \rangle \rangle [0]$

notation *from-singl-to-list_{nd}* $\langle \langle \langle \langle \langle - \rangle \rangle_{singl \hookrightarrow list} \rangle \rangle \rangle [0]$

notation *from-list-to-singl_d* $\langle \langle \langle \langle - \rangle \rangle_{list \rightsquigarrow singl} \rangle \rangle [0]$

notation *from-list-to-singl_{nd}* $\langle \langle \langle \langle \langle - \rangle \rangle_{list \rightsquigarrow singl} \rangle \rangle \rangle [0]$

end

unbundle *functional-automata-morphisms-syntax*

lemma *morphisms-A-scheme-more-simps* [*simp*] :

$\langle \text{more } \langle A \rangle_{d \mapsto nd} = \text{more } A \rangle \langle \text{more } \langle B \rangle_{nd \rightsquigarrow d} = \text{more } B \rangle$
 $\langle \text{more } d \langle C \rangle_{\sigma \mapsto \sigma s} = \text{more } C \rangle \langle \text{more } nd \langle D \rangle_{\sigma \mapsto \sigma s} = \text{more } D \rangle$
 $\langle \text{more } d \langle E \rangle_{\sigma s \rightsquigarrow \sigma} = \text{more } E \rangle \langle \text{more } nd \langle F \rangle_{\sigma s \rightsquigarrow \sigma} = \text{more } F \rangle$
 $\langle \text{more } d \langle G \rangle_{\text{singl} \mapsto \text{list}} = \text{more } G \rangle \langle \text{more } nd \langle H \rangle_{\text{singl} \mapsto \text{list}} = \text{more } H \rangle$
 $\langle \text{more } d \langle I \rangle_{\text{list} \rightsquigarrow \text{singl}} = \text{more } I \rangle \langle \text{more } nd \langle J \rangle_{\text{list} \rightsquigarrow \text{singl}} = \text{more } J \rangle$
by (*simp-all add: det-ndet-conv-defs σ - σs -conv-defs singl-list-conv-defs*)

3.1.6 Generic update Functions

definition *update-both* **where** $\langle \text{update-both } A_0 A_1 \sigma_0 \sigma_1 e f \equiv f (\tau A_0 \sigma_0 e) (\tau A_1 \sigma_1 e) \rangle$

definition *update-left* **where** $\langle \text{update-left } A_0 \sigma_0 \sigma_1 e f g \equiv f (\tau A_0 \sigma_0 e) (g \sigma_1) \rangle$

definition *update-right* **where** $\langle \text{update-right } A_1 \sigma_0 \sigma_1 e f g \equiv f (g \sigma_0) (\tau A_1 \sigma_1 e) \rangle$

lemmas *update-defs*[*simp*] = *update-both-def update-left-def update-right-def*

abbreviation *f-up-set* **where** $\langle \text{f-up-set } f B C \equiv \{f s t \mid s t. (s, t) \in B \times C\} \rangle$

abbreviation *f-up-opt* **where** $\langle \text{f-up-opt } f s t \equiv \text{case } s \text{ of } \diamond \Rightarrow \diamond \mid [s'] \Rightarrow \text{map-option } (f s') t \rangle$

3.1.7 Assumptions on Automata

definition *finite-trans* :: $\langle (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \Rightarrow \text{bool} \rangle$
where $\langle \text{finite-trans } A \equiv \forall \sigma a. \text{finite } (\tau A \sigma a) \rangle$

lemma *finite-trans-morphisms-simps*[*simp*]:

$\langle \text{finite-trans } \langle A \rangle_{d \mapsto nd} \rangle$
 $\langle \text{finite-trans } B \Longrightarrow \text{finite-trans } nd \langle B \rangle_{\sigma \mapsto \sigma s} \rangle$
 $\langle \text{finite-trans } C \Longrightarrow \text{finite-trans } nd \langle C \rangle_{\sigma s \rightsquigarrow \sigma} \rangle$
 $\langle \text{finite-trans } D \Longrightarrow \text{finite-trans } nd \langle D \rangle_{\text{singl} \mapsto \text{list}} \rangle$
 $\langle \text{finite-trans } E \Longrightarrow \text{finite-trans } nd \langle E \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle$
unfolding *det-ndet-conv-defs σ - σs -conv-defs singl-list-conv-defs finite-trans-def*
by (*simp-all add: option.case-eq-if*)

definition *at-most-1-elem* :: $\langle (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \Rightarrow \text{bool} \rangle$

where $\langle \text{at-most-1-elem } A \equiv$
 $(\forall \sigma a. \tau A \sigma a = \{\}) \vee (\exists \sigma'. \tau A \sigma a = \{\sigma'\}) \wedge$
 $(\forall \sigma. \omega A \sigma = \{\}) \vee (\exists r. \omega A \sigma = \{r\}) \rangle$

lemma *at-most-1-elem-def-bis* :

$\langle \text{at-most-1-elem } A \longleftrightarrow (\forall \sigma a. \exists \sigma'. \tau A \sigma a \subseteq \{\sigma'\}) \wedge (\forall \sigma. \exists r. \omega A \sigma \subseteq \{r\}) \rangle$
by (*auto simp add: at-most-1-elem-def subset-iff*)
 (((*metis empty-iff singleton-iff*)⁺)[2],
 ((*metis equals0D is-singletonI' is-singleton-some-elem*)⁺)[2])

lemma *at-most-1-elemI* :

$\langle \llbracket \bigwedge \sigma a. \tau A \sigma a = \{\} \vee (\exists \sigma'. \tau A \sigma a = \{\sigma'\});$
 $\bigwedge \sigma. \omega A \sigma = \{\} \vee (\exists r. \omega A \sigma = \{r\}) \rrbracket \Longrightarrow \text{at-most-1-elem } A \rangle$
by (*simp add: at-most-1-elem-def*)

lemma *at-most-1-elemE* :

$\langle \llbracket \tau A \sigma a = \{\} \Longrightarrow \text{thesis}; \bigwedge \sigma'. \tau A \sigma a = \{\sigma'\} \Longrightarrow \text{thesis} \rrbracket \Longrightarrow \text{thesis} \rangle$
 $\langle \llbracket \omega A \sigma = \{\} \Longrightarrow \text{thesis}; \bigwedge r. \omega A \sigma = \{r\} \Longrightarrow \text{thesis} \rrbracket \Longrightarrow \text{thesis} \rangle$
if $\langle \text{at-most-1-elem } A \rangle$
by (*meson at-most-1-elem-def* $\langle \text{at-most-1-elem } A \rangle$)⁺

definition *at-most-1-elem-trans* :: $\langle (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \Rightarrow \text{bool} \rangle$

where $\langle \text{at-most-1-elem-trans } A \equiv \forall \sigma a. \tau A \sigma a = \{\} \vee (\exists \sigma'. \tau A \sigma a = \{\sigma'\}) \rangle$

lemma *at-most-1-elem-trans-def-bis* :

$\langle \text{at-most-1-elem-trans } A \longleftrightarrow (\forall \sigma a. \exists \sigma'. \tau A \sigma a \subseteq \{\sigma'\}) \rangle$
by (*auto simp add: at-most-1-elem-trans-def subset-iff*)
 (*metis empty-iff singleton-iff*,
metis equals0D is-singletonI' is-singleton-some-elem)

lemma *at-most-1-elem-transI* :

$\langle \llbracket \bigwedge \sigma a. \tau A \sigma a = \{\} \vee (\exists \sigma'. \tau A \sigma a = \{\sigma'\}) \rrbracket \Longrightarrow \text{at-most-1-elem-trans } A \rangle$
by (*simp add: at-most-1-elem-trans-def*)

lemma *at-most-1-elem-transE* :

$\langle \llbracket \tau A \sigma a = \{\} \Longrightarrow \text{thesis}; \bigwedge \sigma'. \tau A \sigma a = \{\sigma'\} \Longrightarrow \text{thesis} \rrbracket \Longrightarrow \text{thesis} \rangle$
if $\langle \text{at-most-1-elem-trans } A \rangle$
by (*meson at-most-1-elem-trans-def* $\langle \text{at-most-1-elem-trans } A \rangle$)⁺

lemma *at-most-1-elem-imp-at-most-1-elem-trans* :

$\langle \text{at-most-1-elem } A \Longrightarrow \text{at-most-1-elem-trans } A \rangle$
by (*simp add: at-most-1-elem-def at-most-1-elem-trans-def*)

definition *length-1-trans_d* :: $\langle (' \sigma \text{ list}, 'a, 'r, ' \alpha) A_d\text{-scheme} \Rightarrow \text{bool} \rangle$

where $\langle \text{length-1-trans}_d A \equiv$

$\forall \sigma s a. \text{case } \tau A \sigma s a \text{ of } \diamond \Rightarrow \text{True} \mid [\sigma s'] \Rightarrow \text{length } \sigma s' = \text{Suc } 0 \rangle$

lemma *length-1-trans_dI* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \tau A \sigma s a = \lfloor \sigma s' \rfloor \implies \text{length } \sigma s' = \text{Suc } 0 \rrbracket \implies \text{length-1-trans}_d A \rangle$
by (*simp add: length-1-trans_d-def split: option.split*)

lemma *length-1-trans_dE* :

$\langle \llbracket \text{length-1-trans}_d A; \tau A \sigma s a = \lfloor \sigma s' \rfloor; \bigwedge \sigma. \sigma s' = \lfloor \sigma \rfloor \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
by (*simp add: length-1-trans_d-def split: option.split-asm*)
(metis (no-types) length-0-conv length-Suc-conv)

definition *length-1-trans_{nd}* :: $\langle (' \sigma \text{ list}, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \implies \text{bool} \rangle$

where $\langle \text{length-1-trans}_{nd} A \equiv \forall \sigma s a. \forall \sigma s' \in \tau A \sigma s a. \text{length } \sigma s' = \text{Suc } 0 \rangle$

lemma *length-1-trans_{nd}I* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \sigma s' \in \tau A \sigma s a \implies \text{length } \sigma s' = \text{Suc } 0 \rrbracket \implies \text{length-1-trans}_{nd} A \rangle$
by (*simp add: length-1-trans_{nd}-def split: option.split*)

lemma *length-1-trans_{nd}E* :

$\langle \llbracket \text{length-1-trans}_{nd} A; \sigma s' \in \tau A \sigma s a; \bigwedge \sigma. \sigma s' = \lfloor \sigma \rfloor \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
by (*simp add: length-1-trans_{nd}-def split: option.split-asm*)
(metis (no-types) length-0-conv length-Suc-conv)

definition *length-1_d* :: $\langle (' \sigma \text{ list}, 'a, 'r \text{ list}, ' \alpha) A_d\text{-scheme} \implies \text{bool} \rangle$

where $\langle \text{length-1}_d A \equiv$

$(\forall \sigma s a. \text{case } \tau A \sigma s a \text{ of } \diamond \Rightarrow \text{True} \mid \lfloor \sigma s' \rfloor \Rightarrow \text{length } \sigma s' = \text{Suc } 0) \wedge$
 $(\forall \sigma s. \text{case } \omega A \sigma s \text{ of } \diamond \Rightarrow \text{True} \mid \lfloor rs \rfloor \Rightarrow \text{length } rs = \text{Suc } 0) \rangle$

lemma *length-1_dI* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \tau A \sigma s a = \lfloor \sigma s' \rfloor \implies \text{length } \sigma s' = \text{Suc } 0; \bigwedge \sigma s rs. \omega A \sigma s = \lfloor rs \rfloor \implies \text{length } rs = \text{Suc } 0 \rrbracket \implies \text{length-1}_d A \rangle$
by (*simp add: length-1_d-def split: option.split*)

lemma *length-1_dE* :

$\langle \llbracket \text{length-1}_d A; \tau A \sigma s a = \lfloor \sigma s' \rfloor; \bigwedge \sigma. \sigma s' = \lfloor \sigma \rfloor \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
 $\langle \llbracket \text{length-1}_d A; \omega A \sigma s = \lfloor rs \rfloor; \bigwedge r. rs = \lfloor r \rfloor \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
by (*simp add: length-1_d-def split: option.split-asm,*
metis (no-types) length-0-conv length-Suc-conv)+

definition *length-1_{nd}* :: $\langle (' \sigma \text{ list}, 'a, 'r \text{ list}, ' \alpha) A_{nd}\text{-scheme} \implies \text{bool} \rangle$

where $\langle \text{length-1}_{nd} A \equiv (\forall \sigma s a. \forall \sigma s' \in \tau A \sigma s a. \text{length } \sigma s' = \text{Suc } 0) \wedge$
 $(\forall \sigma s. \forall rs \in \omega A \sigma s. \text{length } rs = \text{Suc } 0) \rangle$

lemma *length-1_{nd}I* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \sigma s' \in \tau A \sigma s a \implies \text{length } \sigma s' = \text{Suc } 0; \bigwedge \sigma s rs. rs \in \omega A \sigma s \implies \text{length } rs = \text{Suc } 0 \rrbracket \implies \text{length-1}_{nd} A \rangle$
by (*simp add: length-1_{nd}-def split: option.split*)

lemma *length-1_{nd}E* :

$\langle \llbracket \text{length-1}_{nd} A; \sigma s' \in \tau A \sigma s a; \bigwedge \sigma. \sigma s' = [\sigma] \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
 $\langle \llbracket \text{length-1}_{nd} A; rs \in \omega A \sigma s; \bigwedge r. rs = [r] \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
by (*simp add: length-1_{nd}-def split: option.split-asm,*
metis (no-types) length-0-conv length-Suc-conv)⁺

definition *indep-enabl* :: $\langle (' \sigma_0, 'a, 'r_0, ' \alpha) A_d\text{-scheme} \implies ' \sigma_0 \implies 'a \text{ set} \implies (' \sigma_1, 'a,$
 $'r_1, ' \beta) A_d\text{-scheme} \implies ' \sigma_1 \implies \text{bool} \rangle$
where $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \equiv \forall t_0 \in \mathcal{R}_d A_0 \sigma_0. \forall t_1 \in \mathcal{R}_d A_1 \sigma_1. \varepsilon A_0 t_0 \cap \varepsilon A_1 t_1 \subseteq E \rangle$

lemma *indep-enablI* :
 $\langle (\bigwedge t_0 t_1. t_0 \in \mathcal{R}_d A_0 \sigma_0 \implies t_1 \in \mathcal{R}_d A_1 \sigma_1 \implies \varepsilon A_0 t_0 \cap \varepsilon A_1 t_1 \subseteq E)$
 $\implies \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
and *indep-enablD* :
 $\langle \llbracket \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1; t_0 \in \mathcal{R}_d A_0 \sigma_0; t_1 \in \mathcal{R}_d A_1 \sigma_1 \rrbracket \implies \varepsilon A_0 t_0 \cap \varepsilon$
 $A_1 t_1 \subseteq E \rangle$
by (*simp-all add: indep-enabl-def*)

definition *ρ-disjoint-ε* :: $\langle (' \sigma, 'a, ' \sigma', 'r', ' \alpha) A\text{-scheme} \implies \text{bool} \rangle$
where $\langle \rho\text{-disjoint-}\varepsilon A \equiv \forall \sigma \in \rho A. \varepsilon A \sigma = \{\} \rangle$

lemma *ρ-disjoint-εI* : $\langle (\bigwedge \sigma. \sigma \in \rho A \implies \varepsilon A \sigma = \{\}) \implies \rho\text{-disjoint-}\varepsilon A \rangle$
and *ρ-disjoint-εD* : $\langle \rho\text{-disjoint-}\varepsilon A \implies \sigma \in \rho A \implies \varepsilon A \sigma = \{\} \rangle$
by (*simp-all add: ρ-disjoint-ε-def*)

definition *at-most-1-elem-term* :: $\langle (' \sigma, 'a, 'r, ' \alpha) A_{nd}\text{-scheme} \implies \text{bool} \rangle$
where $\langle \text{at-most-1-elem-term } A \equiv \forall \sigma. \omega A \sigma = \{\} \vee (\exists r. \omega A \sigma = \{r\}) \rangle$

lemma *at-most-1-elem-term-def-bis* :
 $\langle \text{at-most-1-elem-term } A \longleftrightarrow (\forall \sigma. \exists r. \omega A \sigma \subseteq \{r\}) \rangle$
by (*auto simp add: at-most-1-elem-term-def subset-iff*)
(metis empty-iff singleton-iff,
metis equals0D is-singletonI' is-singleton-some-elem)

lemma *at-most-1-elem-termI* :
 $\langle \llbracket \bigwedge \sigma. \omega A \sigma = \{\} \vee (\exists r. \omega A \sigma = \{r\}) \rrbracket \implies \text{at-most-1-elem-term } A \rangle$
by (*simp add: at-most-1-elem-term-def*)

lemma *at-most-1-elem-termE* :
 $\langle \llbracket \omega A \sigma = \{\} \implies \text{thesis}; \bigwedge r. \omega A \sigma = \{r\} \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$
if $\langle \text{at-most-1-elem-term } A \rangle$
by (*meson at-most-1-elem-term-def <at-most-1-elem-term A>*)⁺

lemma *at-most-1-elem-imp-at-most-1-elem-term* :
 $\langle \text{at-most-1-elem } A \implies \text{at-most-1-elem-term } A \rangle$
by (*simp add: at-most-1-elem-def at-most-1-elem-term-def*)

3.2 First Properties

3.2.1 ε , ϱ and ω first equalities

lemma *base-trans- ε [simp]*:
 $\langle \varepsilon (\langle \tau = \lambda\sigma \ a. \diamond, \omega = \lambda\sigma. \diamond, \dots = \text{some} \rangle :: ('\sigma, 'a, 'r, 'a) A_d\text{-scheme}) \sigma = \{\} \rangle$
 $\langle \varepsilon (\langle \tau = \lambda\sigma \ a. \{\}, \omega = \lambda\sigma. \{\}, \dots = \text{some} \rangle :: ('\sigma, 'a, 'r, 'a) A_{nd}\text{-scheme}) \sigma = \{\} \rangle$
by (*simp-all add: ε -simps*)

lemma *base-trans- ϱ [simp]*:
 $\langle \varrho (\langle \tau = \lambda\sigma \ a. \diamond, \omega = \lambda\sigma. \diamond, \dots = \text{some} \rangle :: ('\sigma, 'a, 'r, 'a) A_d\text{-scheme}) = \{\} \rangle$
 $\langle \varrho (\langle \tau = \lambda\sigma \ a. \{\}, \omega = \lambda\sigma. \{\}, \dots = \text{some} \rangle :: ('\sigma, 'a, 'r, 'a) A_{nd}\text{-scheme}) = \{\} \rangle$
by (*simp-all add: ϱ -simps*)

lemma *σ - σ -conv- ε [simp]*:
 $\langle \varepsilon_d \langle A \rangle_{\sigma \mapsto \sigma s} \sigma s = \varepsilon A \ (hd \ \sigma s) \rangle \langle \varepsilon_{nd} \langle B \rangle_{\sigma \mapsto \sigma s} \sigma s = \varepsilon B \ (hd \ \sigma s) \rangle$
 $\langle \varepsilon_d \langle C \rangle_{\sigma s \rightsquigarrow \sigma} \sigma = \varepsilon C \ [\sigma] \rangle \langle \varepsilon_{nd} \langle D \rangle_{\sigma s \rightsquigarrow \sigma} \sigma = \varepsilon D \ [\sigma] \rangle$
by (*simp-all add: σ - σ -conv-defs ε -simps option.case-eq-if*)

lemma *σ - σ -conv- ϱ [simp]*:
 $\langle \varrho_d \langle A \rangle_{\sigma \mapsto \sigma s} = \{\sigma s. hd \ \sigma s \in \varrho A\} \rangle \langle \varrho_{nd} \langle B \rangle_{\sigma \mapsto \sigma s} = \{\sigma s. hd \ \sigma s \in \varrho B\} \rangle$
 $\langle \varrho_d \langle C \rangle_{\sigma s \rightsquigarrow \sigma} = \{\sigma. [\sigma] \in \varrho C\} \rangle \langle \varrho_{nd} \langle D \rangle_{\sigma s \rightsquigarrow \sigma} = \{\sigma. [\sigma] \in \varrho D\} \rangle$
by (*simp-all add: σ - σ -conv-defs ϱ -simps option.case-eq-if*)

lemma *singl-list-conv- ε [simp]*:
 $\langle \varepsilon_d \langle A \rangle_{singl \mapsto list} \sigma s = \varepsilon A \ (hd \ \sigma s) \rangle \langle \varepsilon_{nd} \langle B \rangle_{singl \mapsto list} \sigma s = \varepsilon B \ (hd \ \sigma s) \rangle$
 $\langle \varepsilon_d \langle C \rangle_{list \rightsquigarrow singl} \sigma = \varepsilon C \ [\sigma] \rangle \langle \varepsilon_{nd} \langle D \rangle_{list \rightsquigarrow singl} \sigma = \varepsilon D \ [\sigma] \rangle$
by (*simp-all add: singl-list-conv-defs ε -simps option.case-eq-if*)

lemma *singl-list-conv- ϱ [simp]*:
 $\langle \varrho_d \langle A \rangle_{singl \mapsto list} = \{\sigma s. hd \ \sigma s \in \varrho A\} \rangle \langle \varrho_{nd} \langle B \rangle_{singl \mapsto list} = \{\sigma s. hd \ \sigma s \in \varrho B\} \rangle$
 $\langle \varrho_d \langle C \rangle_{list \rightsquigarrow singl} = \{\sigma. [\sigma] \in \varrho C\} \rangle \langle \varrho_{nd} \langle D \rangle_{list \rightsquigarrow singl} = \{\sigma. [\sigma] \in \varrho D\} \rangle$
by (*simp-all add: singl-list-conv-defs ϱ -simps option.case-eq-if*)

lemma *det-ndet-conv- ε [simp]*: $\langle \varepsilon \langle A \rangle_{d \mapsto nd} = \varepsilon A \rangle \langle \varepsilon \langle B \rangle_{nd \rightsquigarrow d} = \varepsilon B \rangle$
by (*rule ext, simp add: det-ndet-conv-defs ε -simps option.case-eq-if*) $+$

lemma *det-ndet-conv- ϱ [simp]*: $\langle \varrho \langle A \rangle_{d \mapsto nd} = \varrho A \rangle \langle \varrho \langle B \rangle_{nd \rightsquigarrow d} = \varrho B \rangle$
by (*simp-all add: det-ndet-conv-defs ϱ -simps option.case-eq-if*)

lemma ω -from-det-to-ndet :

$\langle \omega \langle A \rangle_{d \mapsto nd} = (\lambda \sigma. \text{case } \omega \ A \ \sigma \ \text{of } [r] \Rightarrow \{r\} \mid \diamond \Rightarrow \{\}) \rangle$
by (auto simp add: det-ndet-conv-defs)

lemma ε - ω -useless [simp] :

$\langle \varepsilon (A(\omega := \text{some-}\omega)) = \varepsilon A \rangle \langle \varepsilon (B(\omega := \text{some-}\omega')) = \varepsilon B \rangle$
for $A :: \langle (' \sigma, 'a, ' \sigma \ \text{option}, 'r \ \text{option}, ' \alpha) \ A\text{-scheme} \rangle$
and $B :: \langle (' \sigma, 'a, ' \sigma \ \text{set}, 'r \ \text{set}, ' \alpha) \ A\text{-scheme} \rangle$
by (rule ext, simp add: ε -simps)+

lemma ϱ -disjoint- ε -updated- ω [simp] :

$\langle \varrho\text{-disjoint-}\varepsilon (A(\omega := \lambda \sigma. \diamond)) \rangle$
 $\langle \varrho\text{-disjoint-}\varepsilon (B(\omega := \lambda \sigma. \{\})) \rangle$
by (simp-all add: ϱ -disjoint- ε -def ϱ -simps)

lemma ϱ -disjoint- ε -det-ndet-conv-iff [simp] :

$\langle \varrho\text{-disjoint-}\varepsilon \langle A \rangle_{d \mapsto nd} \longleftrightarrow \varrho\text{-disjoint-}\varepsilon A \rangle$
 $\langle \varrho\text{-disjoint-}\varepsilon \langle B \rangle_{nd \rightsquigarrow d} \longleftrightarrow \varrho\text{-disjoint-}\varepsilon B \rangle$
by (simp-all add: ϱ -disjoint- ε -def)

lemma at-most-1-elem-term-updated- ω [simp] :

$\langle \text{at-most-1-elem-term } (A(\omega := \lambda \sigma. \{\})) \rangle$
by (simp add: at-most-1-elem-term-def)

lemma at-most-1-elem-term-from-det-to-ndet [simp] :

$\langle \text{at-most-1-elem-term } \langle A \rangle_{d \mapsto nd} \rangle$
by (simp add: det-ndet-conv-defs at-most-1-elem-term-def split: option.split)

lemma at-most-1-elem-term-unit [simp] :

$\langle \text{at-most-1-elem-term } (A :: (' \sigma, 'a, \text{unit}, ' \alpha) \ A_{nd}\text{-scheme}) \rangle$
by (auto simp add: at-most-1-elem-term-def)

3.2.2 Properties of our morphisms

method expand-A-scheme =

match conclusion in $\langle A = B \rangle$ **for** $A \ B :: \langle (' \sigma, 'a, ' \sigma', 'r', ' \alpha) \ A\text{-scheme} \rangle \Rightarrow$
 $\langle \text{cases } A, \text{cases } B \rangle$

lemma base-trans-det-ndet-conv:

$\langle \langle (\tau = \lambda \sigma \ a. \diamond, \omega = \lambda \sigma. \diamond, \dots = \text{some}) \rangle_{d \mapsto nd} =$
 $\langle (\tau = \lambda \sigma \ a. \{\}, \omega = \lambda \sigma. \{\}, \dots = \text{some}) \rangle$
 $\langle \langle (\tau = \lambda \sigma \ a. \{\}, \omega = \lambda \sigma. \{\}, \dots = \text{some}) \rangle_{nd \rightsquigarrow d} =$
 $\langle (\tau = \lambda \sigma \ a. \diamond, \omega = \lambda \sigma. \diamond, \dots = \text{some}) \rangle$

unfolding det-ndet-conv-defs **by** simp-all

lemma from-det-to-ndet- σ - σ s-conv-commute:

$\langle nd \langle \langle A \rangle_{d \mapsto nd} \rangle_{\sigma \mapsto \sigma s} = \langle d \langle \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{d \mapsto nd} \rangle \langle nd \langle \langle B \rangle_{d \mapsto nd} \rangle_{\sigma s \rightsquigarrow \sigma} = \langle d \langle \langle B \rangle_{\sigma s \rightsquigarrow \sigma} \rangle_{d \mapsto nd} \rangle$
by (*simp add: det-ndet-conv-defs σ - σ s-conv-defs, rule ext,*
auto simp add: option.case-eq-if split: if-splits) $\+$

lemma from-det-to-ndet-singl-list-conv-commute:

$\langle nd \langle \langle A \rangle_{d \mapsto nd} \rangle_{singl \mapsto list} = \langle d \langle \langle A \rangle_{singl \mapsto list} \rangle_{d \mapsto nd} \rangle \langle nd \langle \langle B \rangle_{d \mapsto nd} \rangle_{list \rightsquigarrow singl} = \langle d \langle \langle B \rangle_{list \rightsquigarrow singl} \rangle_{d \mapsto nd} \rangle$
by (*simp add: det-ndet-conv-defs singl-list-conv-defs,*
solves $\langle intro conjI ext, auto split: option.split \rangle$) $\+$

lemma from-ndet-to-det- σ - σ s-conv-commute:

$\langle at-most-1-elem-trans A \implies d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{\sigma \mapsto \sigma s} = \langle nd \langle \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{nd \rightsquigarrow d} \rangle$
 $\langle at-most-1-elem-trans B \implies d \langle \langle B \rangle_{nd \rightsquigarrow d} \rangle_{\sigma s \rightsquigarrow \sigma} = \langle nd \langle \langle B \rangle_{\sigma s \rightsquigarrow \sigma} \rangle_{nd \rightsquigarrow d} \rangle$

proof –

assume $*$: $\langle at-most-1-elem-trans A \rangle$

from $*$ **have** $\langle \tau d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{\sigma \mapsto \sigma s} \sigma s a = \tau \langle nd \langle \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{nd \rightsquigarrow d} \sigma s a \rangle$ **for** $\sigma s a$
by (*auto simp add: det-ndet-conv-defs σ - σ s-conv-defs*
elim: at-most-1-elem-transE)

moreover have $\langle \omega d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{\sigma \mapsto \sigma s} \sigma s = \omega \langle nd \langle \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{nd \rightsquigarrow d} \sigma s \rangle$ **for** σs
by (*auto simp add: det-ndet-conv-defs σ - σ s-conv-defs*)

moreover have $\langle more d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{\sigma \mapsto \sigma s} = more \langle nd \langle \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{nd \rightsquigarrow d} \rangle$ **by** *simp*
ultimately show $\langle d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{\sigma \mapsto \sigma s} = \langle nd \langle \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{nd \rightsquigarrow d} \rangle$ **by** *expand-A-scheme*
auto

next

assume $*$: $\langle at-most-1-elem-trans B \rangle$

from $*$ **have** $\langle \tau d \langle \langle B \rangle_{nd \rightsquigarrow d} \rangle_{\sigma s \rightsquigarrow \sigma} \sigma a = \tau \langle nd \langle \langle B \rangle_{\sigma s \rightsquigarrow \sigma} \rangle_{nd \rightsquigarrow d} \sigma a \rangle$ **for** σa
by (*auto simp add: det-ndet-conv-defs σ - σ s-conv-defs*
elim: at-most-1-elem-transE)

moreover have $\langle \omega d \langle \langle B \rangle_{nd \rightsquigarrow d} \rangle_{\sigma s \rightsquigarrow \sigma} \sigma = \omega \langle nd \langle \langle B \rangle_{\sigma s \rightsquigarrow \sigma} \rangle_{nd \rightsquigarrow d} \sigma \rangle$ **for** σ
by (*auto simp add: det-ndet-conv-defs σ - σ s-conv-defs*)

moreover have $\langle more d \langle \langle B \rangle_{nd \rightsquigarrow d} \rangle_{\sigma s \rightsquigarrow \sigma} = more \langle nd \langle \langle B \rangle_{\sigma s \rightsquigarrow \sigma} \rangle_{nd \rightsquigarrow d} \rangle$ **by** *simp*
ultimately show $\langle d \langle \langle B \rangle_{nd \rightsquigarrow d} \rangle_{\sigma s \rightsquigarrow \sigma} = \langle nd \langle \langle B \rangle_{\sigma s \rightsquigarrow \sigma} \rangle_{nd \rightsquigarrow d} \rangle$ **by** *expand-A-scheme*
auto

qed

lemma from-ndet-to-det-singl-list-conv-commute:

$\langle at-most-1-elem A \implies d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{singl \mapsto list} = \langle nd \langle \langle A \rangle_{singl \mapsto list} \rangle_{nd \rightsquigarrow d} \rangle$
 $\langle at-most-1-elem B \implies d \langle \langle B \rangle_{nd \rightsquigarrow d} \rangle_{list \rightsquigarrow singl} = \langle nd \langle \langle B \rangle_{list \rightsquigarrow singl} \rangle_{nd \rightsquigarrow d} \rangle$

proof –

assume $*$: $\langle at-most-1-elem A \rangle$

from $*$ **have** $\langle \tau d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{singl \mapsto list} \sigma s a = \tau \langle nd \langle \langle A \rangle_{singl \mapsto list} \rangle_{nd \rightsquigarrow d} \sigma s a \rangle$
for $\sigma s a$

by (*auto simp add: det-ndet-conv-defs singl-list-conv-defs*
elim: at-most-1-elemE(1))

moreover from $*$ **have** $\langle \omega d \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{singl \mapsto list} \sigma s = \omega \langle nd \langle \langle A \rangle_{singl \mapsto list} \rangle_{nd \rightsquigarrow d} \sigma s \rangle$ **for** σs

by (*auto simp add: det-ndet-conv-defs singl-list-conv-defs*)

$elim: at-most-1-elemE(2)$
moreover have $\langle more\ d\langle\langle A \rangle_{nd\rightsquigarrow d}\rangle_{singl\hookrightarrow list} = more\ \langle\langle nd\langle A \rangle_{singl\hookrightarrow list}\rangle_{nd\rightsquigarrow d} \rangle$
by simp
ultimately show $\langle d\langle\langle A \rangle_{nd\rightsquigarrow d}\rangle_{singl\hookrightarrow list} = \langle\langle nd\langle A \rangle_{singl\hookrightarrow list}\rangle_{nd\rightsquigarrow d} \rangle$ **by ex-**
pand-A-scheme auto
next
assume $* : \langle at-most-1-elem\ B \rangle$
from $*$ **have** $\langle \tau\ d\langle\langle B \rangle_{nd\rightsquigarrow d}\rangle_{list\rightsquigarrow singl}\ \sigma\ a = \tau\ \langle\langle nd\langle B \rangle_{list\rightsquigarrow singl}\rangle_{nd\rightsquigarrow d}\ \sigma\ a \rangle$
for $\sigma\ a$
by $(auto\ simp\ add: det-ndet-conv-defs\ singl-list-conv-defs$
 $elim: at-most-1-elemE(1))$
moreover from $*$ **have** $\langle \omega\ d\langle\langle B \rangle_{nd\rightsquigarrow d}\rangle_{list\rightsquigarrow singl}\ \sigma = \omega\ \langle\langle nd\langle B \rangle_{list\rightsquigarrow singl}\rangle_{nd\rightsquigarrow d}\ \sigma \rangle$
for σ
by $(auto\ simp\ add: det-ndet-conv-defs\ singl-list-conv-defs$
 $elim: at-most-1-elemE(2))$
moreover have $\langle more\ d\langle\langle B \rangle_{nd\rightsquigarrow d}\rangle_{list\rightsquigarrow singl} = more\ \langle\langle nd\langle B \rangle_{list\rightsquigarrow singl}\rangle_{nd\rightsquigarrow d} \rangle$
by simp
ultimately show $\langle d\langle\langle B \rangle_{nd\rightsquigarrow d}\rangle_{list\rightsquigarrow singl} = \langle\langle nd\langle B \rangle_{list\rightsquigarrow singl}\rangle_{nd\rightsquigarrow d} \rangle$ **by ex-**
pand-A-scheme auto
qed

lemma behaviour- σ - σ s-conv:

$\langle \varepsilon\ d\langle A \rangle_{\sigma\hookrightarrow\sigma s}\ [\sigma] = \varepsilon\ A\ \sigma \rangle$
 $\langle \tau\ d\langle A \rangle_{\sigma\hookrightarrow\sigma s}\ [\sigma]\ a = (case\ \tau\ A\ \sigma\ a\ of\ \diamond \Rightarrow \diamond \mid [t] \Rightarrow [[t]]) \rangle$
 $\langle \varrho\ d\langle A \rangle_{\sigma\hookrightarrow\sigma s} = \{\sigma s.\ hd\ \sigma s \in \varrho\ A\} \rangle$
 $\langle \omega\ d\langle A \rangle_{\sigma\hookrightarrow\sigma s}\ [\sigma] = \omega\ A\ \sigma \rangle$
 $\langle \varepsilon\ nd\langle B \rangle_{\sigma\hookrightarrow\sigma s}\ [\sigma] = \varepsilon\ B\ \sigma \rangle$
 $\langle \tau\ nd\langle B \rangle_{\sigma\hookrightarrow\sigma s}\ [\sigma]\ a = \{[\sigma'] \mid \sigma'.\ \sigma' \in \tau\ B\ \sigma\ a\} \rangle$
 $\langle \varrho\ nd\langle B \rangle_{\sigma\hookrightarrow\sigma s} = \{\sigma s.\ hd\ \sigma s \in \varrho\ B\} \rangle$
 $\langle \omega\ nd\langle B \rangle_{\sigma\hookrightarrow\sigma s}\ [\sigma] = \omega\ B\ \sigma \rangle$
 $\langle \varepsilon\ d\langle C \rangle_{\sigma s\rightsquigarrow\sigma}\ \sigma = \varepsilon\ C\ [\sigma] \rangle$
 $\langle \tau\ d\langle C \rangle_{\sigma s\rightsquigarrow\sigma}\ \sigma\ a = (case\ \tau\ C\ [\sigma]\ a\ of\ \diamond \Rightarrow \diamond \mid [\sigma s'] \Rightarrow [hd\ \sigma s']) \rangle$
 $\langle \varrho\ d\langle C \rangle_{\sigma s\rightsquigarrow\sigma} = \{\sigma.\ [\sigma] \in \varrho\ C\} \rangle$
 $\langle \omega\ d\langle C \rangle_{\sigma s\rightsquigarrow\sigma}\ \sigma = \omega\ C\ [\sigma] \rangle$
 $\langle \varepsilon\ nd\langle D \rangle_{\sigma s\rightsquigarrow\sigma}\ \sigma = \varepsilon\ D\ [\sigma] \rangle$
 $\langle \tau\ nd\langle D \rangle_{\sigma s\rightsquigarrow\sigma}\ \sigma\ a = \{hd\ \sigma s' \mid \sigma s'.\ \sigma s' \in \tau\ D\ [\sigma]\ a\} \rangle$
 $\langle \varrho\ nd\langle D \rangle_{\sigma s\rightsquigarrow\sigma} = \{\sigma.\ [\sigma] \in \varrho\ D\} \rangle$ $\langle \omega\ nd\langle D \rangle_{\sigma s\rightsquigarrow\sigma}\ \sigma = \omega\ D\ [\sigma] \rangle$
by simp-all $(simp-all\ add: \sigma\text{-}\sigma s\text{-}conv\text{-}defs)$

lemma behaviour-singl-list-conv:

$\langle \varepsilon\ d\langle A \rangle_{singl\hookrightarrow list}\ [\sigma] = \varepsilon\ A\ \sigma \rangle$
 $\langle \tau\ d\langle A \rangle_{singl\hookrightarrow list}\ [\sigma]\ a = (case\ \tau\ A\ \sigma\ a\ of\ \diamond \Rightarrow \diamond \mid [t] \Rightarrow [[t]]) \rangle$
 $\langle \varrho\ d\langle A \rangle_{singl\hookrightarrow list} = \{\sigma s.\ hd\ \sigma s \in \varrho\ A\} \rangle$
 $\langle \omega\ d\langle A \rangle_{singl\hookrightarrow list}\ [\sigma] = (case\ \omega\ A\ \sigma\ of\ \diamond \Rightarrow \diamond \mid [r] \Rightarrow [[r]]) \rangle$
 $\langle \varepsilon\ nd\langle B \rangle_{singl\hookrightarrow list}\ [\sigma] = \varepsilon\ B\ \sigma \rangle$
 $\langle \tau\ nd\langle B \rangle_{singl\hookrightarrow list}\ [\sigma]\ a = \{[\sigma'] \mid \sigma'.\ \sigma' \in \tau\ B\ \sigma\ a\} \rangle$

$\langle \varrho \text{ nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} = \{ \sigma s. \text{hd } \sigma s \in \varrho B \} \rangle$
 $\langle \omega \text{ nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = \{ [r] \mid r. r \in \omega B \sigma \} \rangle$
 $\langle \varepsilon \text{ d} \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = \varepsilon C [\sigma] \rangle$
 $\langle \tau \text{ d} \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma a = (\text{case } \tau C [\sigma] a \text{ of } \diamond \Rightarrow \diamond \mid [\sigma s'] \Rightarrow [\text{hd } \sigma s']) \rangle$
 $\langle \varrho \text{ d} \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} = \{ \sigma. [\sigma] \in \varrho C \} \rangle$
 $\langle \omega \text{ d} \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = (\text{case } \omega C [\sigma] \text{ of } \diamond \Rightarrow \diamond \mid [rs] \Rightarrow [\text{hd } rs]) \rangle$
 $\langle \varepsilon \text{ nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = \varepsilon D [\sigma] \rangle$
 $\langle \tau \text{ nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma a = \{ \text{hd } \sigma s' \mid \sigma s'. \sigma s' \in \tau D [\sigma] a \} \rangle$
 $\langle \varrho \text{ nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} = \{ \sigma. [\sigma] \in \varrho D \} \rangle$
 $\langle \omega \text{ nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = \{ \text{hd } rs \mid rs. rs \in \omega D [\sigma] \} \rangle$
by *simp-all (simp-all add: singl-list-conv-defs)*

lemma *empty-from-det-to-ndet-is-None-trans [simp]* : $\langle \tau \langle A \rangle_{\text{d} \leftrightarrow \text{nd}} \sigma a = \{ \} \longleftrightarrow \tau A \sigma a = \diamond \rangle$
by (*simp add: ε -simps det-ndet-conv-defs option.case-eq-if*)

lemma *at-most-1-elem-from-det-to-ndet [simp]* : $\langle \text{at-most-1-elem } \langle A \rangle_{\text{d} \leftrightarrow \text{nd}} \rangle$
by (*rule at-most-1-elemI*)
(simp-all add: det-ndet-conv-defs split: option.split)

lemma *from-ndet-to-det-from-det-to-ndet [simp]* : $\langle \langle \langle A \rangle_{\text{d} \leftrightarrow \text{nd}} \rangle_{\text{nd} \rightsquigarrow \text{d}} = A \rangle$
by (*cases A, simp add: det-ndet-conv-defs*)
(intro conjI ext, simp-all split: option.split)

lemma *from-det-to-ndet-from-ndet-to-det [simp]* :
 $\langle \langle \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}} \rangle_{\text{d} \leftrightarrow \text{nd}} = A \rangle$ **if** $\langle \text{at-most-1-elem } A \rangle$
proof –
from that have $\langle \tau \langle \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}} \rangle_{\text{d} \leftrightarrow \text{nd}} \sigma a = \tau A \sigma a \rangle$ **for** σa
by (*auto simp add: det-ndet-conv-defs elim: at-most-1-elemE(1)*)
moreover from that have $\langle \omega \langle \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}} \rangle_{\text{d} \leftrightarrow \text{nd}} \sigma = \omega A \sigma \rangle$ **for** σ
by (*auto simp add: det-ndet-conv-defs elim: at-most-1-elemE(2)*)
moreover have $\langle \text{more } \langle \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}} \rangle_{\text{d} \leftrightarrow \text{nd}} = \text{more } A \rangle$ **by** *simp*
ultimately show $\langle \langle \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}} \rangle_{\text{d} \leftrightarrow \text{nd}} = A \rangle$ **by** *expand-A-scheme fastforce*
qed

theorem *bij-betw-from-det-to-ndet* :
 $\langle \text{bij-betw } (\lambda A. \langle A \rangle_{\text{d} \leftrightarrow \text{nd}}) \text{ UNIV } \{ A. \text{at-most-1-elem } A \} \rangle$
unfolding *bij-betw-iff-bijections*
by (*rule exI[where x = $\langle \lambda A. \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}} \rangle$] simp*)

lemma *bij-betw-from-ndet-to-det* :
 $\langle \text{bij-betw } (\lambda A. \langle A \rangle_{\text{nd} \rightsquigarrow \text{d}}) \{ A. \text{at-most-1-elem } A \} \text{ UNIV} \rangle$
unfolding *bij-betw-iff-bijections*
by (*rule exI[where x = $\langle \lambda A. \langle A \rangle_{\text{d} \leftrightarrow \text{nd}} \rangle$] simp*)

lemma *length-1-trans-from-σ-to-σs* [simp] :
 ⟨length-1-trans_d d⟨A⟩_{σ↔σs}⟩ ⟨length-1-trans_{nd} nd⟨B⟩_{σ↔σs}⟩
by (rule length-1-trans_dI, solves ⟨auto simp add: σ-σs-conv-defs split: option.split-asm⟩)
 (rule length-1-trans_{nd}I, solves ⟨auto simp add: σ-σs-conv-defs split: option.split-asm⟩)

lemma *τ-hd-from-σ-to-σs-eq* [simp] :
 ⟨τ d⟨A⟩_{σ↔σs} [hd σs] a = τ d⟨A⟩_{σ↔σs} σs a⟩
 ⟨τ nd⟨B⟩_{σ↔σs} [hd σs] a = τ nd⟨B⟩_{σ↔σs} σs a⟩
by (simp-all add: σ-σs-conv-defs)

lemma *ω-hd-from-σ-to-σs-eq* [simp] :
 ⟨ω d⟨A⟩_{σ↔σs} [hd σs] = ω d⟨A⟩_{σ↔σs} σs⟩
 ⟨ω nd⟨B⟩_{σ↔σs} [hd σs] = ω nd⟨B⟩_{σ↔σs} σs⟩
by (simp-all add: σ-σs-conv-defs)

lemma *from-σs-to-σ-from-σ-to-σs* [simp] :
 ⟨d⟨d⟨A⟩_{σ↔σs}⟩_{σs↔σ} = A⟩ ⟨nd⟨nd⟨B⟩_{σ↔σs}⟩_{σs↔σ} = B⟩
by (cases A, simp add: σ-σs-conv-defs, intro conjI ext;
 simp add: option.case-eq-if set-eq-iff; metis list.sel(1))
 (cases B, simp add: σ-σs-conv-defs, intro conjI ext;
 simp add: option.case-eq-if set-eq-iff; metis list.sel(1))

lemma *from-σ-to-σs-from-σs-to-σ* [simp] :
 ⟨[length-1-trans_d A; ∧σs a. τ A [hd σs] a = τ A σs a;
 ∧σs. ω A [hd σs] = ω A σs] ⇒ d⟨d⟨A⟩_{σs↔σ}⟩_{σ↔σs} = A⟩
 ⟨[length-1-trans_{nd} B; ∧σs a. τ B [hd σs] a = τ B σs a;
 ∧σs. ω B [hd σs] = ω B σs] ⇒ nd⟨nd⟨B⟩_{σs↔σ}⟩_{σ↔σs} = B⟩

proof –

assume * : ⟨length-1-trans_d A⟩ ⟨∧σs a. τ A [hd σs] a = τ A σs a⟩
 ⟨∧σs. ω A [hd σs] = ω A σs⟩

from *(1) **have** ⟨τ d⟨d⟨A⟩_{σs↔σ}⟩_{σ↔σs} σs a = τ A σs a⟩ **for** σs a
by (auto simp add: σ-σs-conv-defs *(2) split: option.split
 elim: length-1-trans_dE)

moreover have ⟨ω d⟨d⟨A⟩_{σs↔σ}⟩_{σ↔σs} σs = ω A σs⟩ **for** σs
by (simp add: σ-σs-conv-defs *(3))

moreover have ⟨more d⟨d⟨A⟩_{σs↔σ}⟩_{σ↔σs} = more A⟩ **by** simp

ultimately show ⟨d⟨d⟨A⟩_{σs↔σ}⟩_{σ↔σs} = A⟩ **by** expand-A-scheme auto

next

assume * : ⟨length-1-trans_{nd} B⟩ ⟨∧σs a. τ B [hd σs] a = τ B σs a⟩
 ⟨∧σs. ω B [hd σs] = ω B σs⟩

from *(1) **have** ⟨τ nd⟨nd⟨B⟩_{σs↔σ}⟩_{σ↔σs} σs a = τ B σs a⟩ **for** σs a
by (auto simp add: σ-σs-conv-defs *(2) elim: length-1-trans_{nd}E)
 (metis length-1-trans_{nd}E list.sel(1))

moreover have ⟨ω nd⟨nd⟨B⟩_{σs↔σ}⟩_{σ↔σs} σs = ω B σs⟩ **for** σs
by (simp add: σ-σs-conv-defs *(3))

moreover have ⟨more nd⟨nd⟨B⟩_{σs↔σ}⟩_{σ↔σs} = more B⟩ **by** simp

ultimately show ⟨nd⟨nd⟨B⟩_{σs↔σ}⟩_{σ↔σs} = B⟩ **by** expand-A-scheme fastforce

qed

theorem *bij-betw-from-σ-to-σs* :

$\langle \text{bij-betw } (\lambda A. d\langle\langle A \rangle\rangle_{\sigma \hookrightarrow \sigma s}) \text{ UNIV}$
 $\{A. \text{length-1-trans}_d A \wedge (\forall \sigma s a. \tau A [hd \ \sigma s] a = \tau A \ \sigma s a) \wedge (\forall \sigma s. \omega A [hd$
 $\sigma s] = \omega A \ \sigma s)\} \rangle$
(is $\langle \text{bij-betw } (\lambda A. d\langle\langle A \rangle\rangle_{\sigma \hookrightarrow \sigma s}) \text{ UNIV } ?S_d \rangle$
 $\langle \text{bij-betw } (\lambda B. nd\langle\langle B \rangle\rangle_{\sigma \hookrightarrow \sigma s}) \text{ UNIV}$
 $\{B. \text{length-1-trans}_{nd} B \wedge (\forall \sigma s a. \tau B \ \sigma s a = \tau B [hd \ \sigma s] a) \wedge (\forall \sigma s. \omega B [hd$
 $\sigma s] = \omega B \ \sigma s)\} \rangle$
unfolding *bij-betw-iff-bijections*
by (rule *exI*[**where** $x = \langle \lambda A. d\langle\langle A \rangle\rangle_{\sigma s \rightsquigarrow \sigma} \rangle$], *simp*)
(rule *exI*[**where** $x = \langle \lambda A. nd\langle\langle A \rangle\rangle_{\sigma s \rightsquigarrow \sigma} \rangle$], *simp*)

lemma *bij-betw-from-σs-to-σ* :

$\langle \text{bij-betw } (\lambda A. d\langle\langle A \rangle\rangle_{\sigma s \rightsquigarrow \sigma})$
 $\{A. \text{length-1-trans}_d A \wedge (\forall \sigma s a. \tau A [hd \ \sigma s] a = \tau A \ \sigma s a) \wedge (\forall \sigma s. \omega A [hd$
 $\sigma s] = \omega A \ \sigma s)\} \text{ UNIV} \rangle$
 $\langle \text{bij-betw } (\lambda B. nd\langle\langle B \rangle\rangle_{\sigma s \rightsquigarrow \sigma})$
 $\{B. \text{length-1-trans}_{nd} B \wedge (\forall \sigma s a. \tau B \ \sigma s a = \tau B [hd \ \sigma s] a) \wedge (\forall \sigma s. \omega B [hd$
 $\sigma s] = \omega B \ \sigma s)\} \text{ UNIV} \rangle$
unfolding *bij-betw-iff-bijections*
by (rule *exI*[**where** $x = \langle \lambda A. d\langle\langle A \rangle\rangle_{\sigma \hookrightarrow \sigma s} \rangle$], *simp*)
(rule *exI*[**where** $x = \langle \lambda A. nd\langle\langle A \rangle\rangle_{\sigma \hookrightarrow \sigma s} \rangle$], *simp*)

lemma *length-1-from-singl-to-list* [*simp*] :

$\langle \text{length-1}_d d\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \rangle \langle \text{length-1}_{nd} nd\langle\langle B \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \rangle$
by (rule *length-1_dI*; solves $\langle \text{auto simp add: singl-list-conv-defs split: option.split-asm} \rangle$)
(rule *length-1_{nd}I*; solves $\langle \text{auto simp add: singl-list-conv-defs split: option.split-asm} \rangle$)

lemma τ -*hd-from-singl-to-list-eq* [*simp*] :

$\langle \tau d\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} [hd \ \sigma s] a = \tau d\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \ \sigma s a \rangle$
 $\langle \tau nd\langle\langle B \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} [hd \ \sigma s] a = \tau nd\langle\langle B \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \ \sigma s a \rangle$
by (*simp-all add: singl-list-conv-defs*)

lemma ω -*hd-from-singl-to-list-eq* [*simp*] :

$\langle \omega d\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} [hd \ \sigma s] = \omega d\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \ \sigma s \rangle$
 $\langle \omega nd\langle\langle B \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} [hd \ \sigma s] = \omega nd\langle\langle B \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \ \sigma s \rangle$
by (*simp-all add: singl-list-conv-defs*)

lemma *from-list-to-singl-from-singl-to-list* [*simp*] :

$\langle d\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \rangle_{\text{list} \rightsquigarrow \text{singl}} = A \rangle \langle nd\langle\langle nd\langle\langle B \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \rangle_{\text{list} \rightsquigarrow \text{singl}} = B \rangle$
by (*cases A*, *simp add: singl-list-conv-defs*, *intro conjI ext*;
simp add: option.case-eq-if set-eq-iff; *metis list.sel(1)*)
(*cases B*, *simp add: singl-list-conv-defs*, *intro conjI ext*;)

simp add: option.case-eq-iff set-eq-iff; metis list.sel(1)

lemma *from-singl-to-list-from-list-to-singl* [simp] :

⟨ $[\text{length-1}_d A; \bigwedge \sigma s a. \tau A [\text{hd } \sigma s] a = \tau A \sigma s a;$
 $\bigwedge \sigma s. \omega A [\text{hd } \sigma s] = \omega A \sigma s] \implies d \langle d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} = A \rangle$
 $[\text{length-1}_{nd} B; \bigwedge \sigma s a. \tau B [\text{hd } \sigma s] a = \tau B \sigma s a;$
 $\bigwedge \sigma s. \omega B [\text{hd } \sigma s] = \omega B \sigma s] \implies nd \langle nd \langle B \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} = B \rangle$

proof –

assume * : ⟨ $\text{length-1}_d A$ ⟩ ⟨ $\bigwedge \sigma s a. \tau A [\text{hd } \sigma s] a = \tau A \sigma s a$ ⟩

⟨ $\bigwedge \sigma s. \omega A [\text{hd } \sigma s] = \omega A \sigma s$ ⟩

from *(1) **have** ⟨ $\tau d \langle d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} \sigma s a = \tau A \sigma s a$ ⟩ **for** $\sigma s a$

by (*auto simp add: singl-list-conv-defs* *(2))

split: option.split elim: length-1_dE(1)

moreover from *(1) **have** ⟨ $\omega d \langle d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} \sigma s = \omega A \sigma s$ ⟩ **for**

σs

by (*auto simp add: singl-list-conv-defs* *(3))

split: option.split elim: length-1_dE(2)

moreover have ⟨*more* $d \langle d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} = \text{more } A$ ⟩ **by** *simp*

ultimately show ⟨ $d \langle d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} = A$ ⟩ **by** *expand-A-scheme auto*

next

assume * : ⟨ $\text{length-1}_{nd} B$ ⟩ ⟨ $\bigwedge \sigma s a. \tau B [\text{hd } \sigma s] a = \tau B \sigma s a$ ⟩

⟨ $\bigwedge \sigma s. \omega B [\text{hd } \sigma s] = \omega B \sigma s$ ⟩

from *(1) **have** ⟨ $\tau nd \langle nd \langle B \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} \sigma s a = \tau B \sigma s a$ ⟩ **for** $\sigma s a$

by (*auto simp add: singl-list-conv-defs* *(2) *elim: length-1_ndE(1)*)

(*metis length-1_ndE(1) list.sel(1)*)

moreover from *(1) **have** ⟨ $\omega nd \langle nd \langle B \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} \sigma s = \omega B \sigma s$ ⟩ **for**

σs

by (*auto simp add: singl-list-conv-defs* *(3) *elim: length-1_ndE(2)*)

(*metis length-1_ndE(2) list.sel(1)*)

moreover have ⟨*more* $nd \langle nd \langle B \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} = \text{more } B$ ⟩ **by** *simp*

ultimately show ⟨ $nd \langle nd \langle B \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle_{\text{singl} \hookrightarrow \text{list}} = B$ ⟩ **by** *expand-A-scheme*

fastforce

qed

theorem *bij-betw-from-singl-to-list* :

⟨*bij-betw* $(\lambda A. d \langle A \rangle_{\text{singl} \hookrightarrow \text{list}})$ *UNIV*

$\{A. \text{length-1}_d A \wedge (\forall \sigma s a. \tau A [\text{hd } \sigma s] a = \tau A \sigma s a) \wedge (\forall \sigma s. \omega A [\text{hd } \sigma s] = \omega A \sigma s)\}$ ⟩

(**is** ⟨*bij-betw* $(\lambda A. d \langle A \rangle_{\text{singl} \hookrightarrow \text{list}})$ *UNIV* ?*S_d*⟩)

⟨*bij-betw* $(\lambda B. nd \langle B \rangle_{\text{singl} \hookrightarrow \text{list}})$ *UNIV*

$\{B. \text{length-1}_{nd} B \wedge (\forall \sigma s a. \tau B [\text{hd } \sigma s] a = \tau B \sigma s a) \wedge (\forall \sigma s. \omega B [\text{hd } \sigma s] = \omega B \sigma s)\}$ ⟩

unfolding *bij-betw-iff-bijections*

by (*rule exI*[**where** $x = \langle \lambda A. d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle$], *simp*)

(*rule exI*[**where** $x = \langle \lambda A. nd \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle$], *simp*)

lemma *bij-betw-from-list-to-singl* :

⟨*bij-betw* $(\lambda A. d \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}})$

$\{A. \text{length-1}_d A \wedge (\forall \sigma s a. \tau A [\text{hd } \sigma s] a = \tau A \sigma s a) \wedge (\forall \sigma s. \omega A [\text{hd } \sigma s] = \omega A \sigma s)\}$ *UNIV*
 $\langle \text{bij-betw } (\lambda B. \text{nd}\langle\langle B \rangle\rangle_{\text{list} \rightsquigarrow \text{singl}}) \rangle$
 $\{B. \text{length-1}_{nd} B \wedge (\forall \sigma s a. \tau B \sigma s a = \tau B [\text{hd } \sigma s] a) \wedge (\forall \sigma s. \omega B [\text{hd } \sigma s] = \omega B \sigma s)\}$ *UNIV*
unfolding *bij-betw-iff-bijections*
by (*rule exI[where x = $\langle \lambda A. \text{d}\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \rangle$, simp]*, *simp*)
(*rule exI[where x = $\langle \lambda A. \text{nd}\langle\langle A \rangle\rangle_{\text{singl} \hookrightarrow \text{list}} \rangle$, simp]*, *simp*)

3.2.3 Reachability results (for \mathcal{R}_d and \mathcal{R}_{nd})

lemma \mathcal{R} -base-trans[*simp*]: $\langle \mathcal{R}_d \ (\tau = \lambda \sigma a. \diamond, \omega = \lambda \sigma. \diamond, \dots = \text{some}) = (\lambda \sigma. \{\sigma\}) \rangle$
 $\langle \mathcal{R}_{nd} \ (\tau = \lambda \sigma a. \{\}, \omega = \lambda \sigma. \{\}, \dots = \text{some}) = (\lambda \sigma. \{\sigma\}) \rangle$
by (*rule ext, safe, subst (asm) \mathcal{R}_d .simps \mathcal{R}_{nd} .simps, simp-all add: \mathcal{R}_d .init \mathcal{R}_{nd} .init*) $+$

theorem \mathcal{R}_{nd} -from-det-to-ndet : $\langle \mathcal{R}_{nd} \ \langle\langle A \rangle\rangle_{d \hookrightarrow nd} \sigma = \mathcal{R}_d \ A \ \sigma \rangle$

proof *safe*

show $\langle \sigma' \in \mathcal{R}_{nd} \ \langle\langle A \rangle\rangle_{d \hookrightarrow nd} \sigma \implies \sigma' \in \mathcal{R}_d \ A \ \sigma \rangle$ **for** σ'
by (*induct rule: \mathcal{R}_{nd} .induct, fact \mathcal{R}_d .init, erule \mathcal{R}_d .step*)
(*simp add: from-det-to-ndet-def option.case-eq-if split: if-split-asm*)

next

show $\langle \sigma' \in \mathcal{R}_d \ A \ \sigma \implies \sigma' \in \mathcal{R}_{nd} \ \langle\langle A \rangle\rangle_{d \hookrightarrow nd} \sigma \rangle$ **for** σ'
by (*induct rule: \mathcal{R}_d .induct, fact \mathcal{R}_{nd} .init*)
(*metis \mathcal{R}_{nd} .step det-ndet-conv-defs(1) option.case(2)*
option.set-intros option.simps(15) select-convs(1))

qed

lemma *bij-betw- \mathcal{R}_{nd} -if-same- τ* : $\langle \text{bij-betw } f \ (\mathcal{R}_{nd} \ B_0 \ \sigma_0) \ (\mathcal{R}_{nd} \ B_1 \ (f \ \sigma_0)) \rangle$
if $\langle \text{inj-on } f \ (\mathcal{R}_{nd} \ B_0 \ \sigma_0) \rangle$ **and** $\langle \bigwedge \sigma_0' a. \sigma_0' \in \mathcal{R}_{nd} \ B_0 \ \sigma_0 \implies \tau \ B_1 \ (f \ \sigma_0') \ a = f \ ' \ \tau \ B_0 \ \sigma_0' \ a \rangle$

proof (*rule bij-betw-imageI, fact that(1), auto simp add: image-def, goal-cases*)

show $\langle s \in \mathcal{R}_{nd} \ B_0 \ \sigma_0 \implies f \ s \in \mathcal{R}_{nd} \ B_1 \ (f \ \sigma_0) \rangle$ **for** s
by (*induct rule: \mathcal{R}_{nd} .induct, simp add: \mathcal{R}_{nd} .init, metis \mathcal{R}_{nd} .step that(2) image-eqI*)

next

show $\langle s \in \mathcal{R}_{nd} \ B_1 \ (f \ \sigma_0) \implies \exists t \in \mathcal{R}_{nd} \ B_0 \ \sigma_0. s = f \ t \rangle$ **for** s
by (*induct rule: \mathcal{R}_{nd} .induct, metis \mathcal{R}_{nd} .simps, metis (mono-tags, lifting) \mathcal{R}_{nd} .step that(2) image-iff*)

qed

lemma *bij-betw- \mathcal{R}_d -if-same- τ* : $\langle \text{bij-betw } f \ (\mathcal{R}_d \ A_0 \ \sigma_0) \ (\mathcal{R}_d \ A_1 \ (f \ \sigma_0)) \rangle$

if $\langle \text{inj-on } f \ (\mathcal{R}_d \ A_0 \ \sigma_0) \rangle$ **and** $\langle \bigwedge \sigma_0' a. \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \tau \ A_1 \ (f \ \sigma_0') \ a = \text{map-option } f \ (\tau \ A_0 \ \sigma_0' \ a) \rangle$

by (*subst (1 2) \mathcal{R}_{nd} -from-det-to-ndet[symmetric], rule bij-betw- \mathcal{R}_{nd} -if-same- τ*)
(*simp-all add: \mathcal{R}_{nd} -from-det-to-ndet that(1)*,

simp add: det-ndet-conv-defs that(2) option.case-eq-if map-option-case)

lemmas *same- τ -implies-same- $\mathcal{R}_{nd} = \text{bij-betw-}\mathcal{R}_{nd}\text{-if-same-}\tau$* [where $f = \text{id}$, simplified *bij-betw-def*, simplified]

and *same- τ -implies-same- $\mathcal{R}_d = \text{bij-betw-}\mathcal{R}_d\text{-if-same-}\tau$* [where $f = \text{id}$, simplified *bij-betw-def option.map-id*, simplified]

corollary *\mathcal{R}_d - ω -useless* [simp] : $\langle \mathcal{R}_d (A(\omega := \text{some-}\omega)) \sigma = \mathcal{R}_d A \sigma \rangle$

by (*auto intro!: same- τ -implies-same- \mathcal{R}_d*)

corollary *\mathcal{R}_{nd} - ω -useless* [simp] : $\langle \mathcal{R}_{nd} (A(\omega := \text{some-}\omega)) \sigma = \mathcal{R}_{nd} A \sigma \rangle$

by (*auto intro!: same- τ -implies-same- \mathcal{R}_{nd}*)

corollary *indep-enabl- ω -useless* [simp] :

$\langle \text{indep-enabl } (A_0(\omega := \text{some-}\omega)) \sigma_0 E A_1 \sigma_1 \longleftrightarrow \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$

$\langle \text{indep-enabl } A_0 \sigma_0 E (A_1(\omega := \text{some-}\omega)) \sigma_1 \longleftrightarrow \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$

by (*simp-all add: indep-enabl-def*)

method *\mathcal{R} -subset-method uses* *defs opt induct init_simps =*

induct rule: induct, auto simp add: init_defs ε -simps split: if-splits,

(metis (no-types, opaque-lifting)_simps)+

method *\mathcal{R}_d -subset-method uses* *defs opt =*

\mathcal{R} -subset-method defs: defs opt: opt induct: \mathcal{R}_d .induct init: \mathcal{R}_d .init_simps: \mathcal{R}_d .simps

method *\mathcal{R}_{nd} -subset-method uses* *defs opt =*

\mathcal{R} -subset-method defs: defs opt: opt induct: \mathcal{R}_{nd} .induct init: \mathcal{R}_{nd} .init_simps: \mathcal{R}_{nd} .simps

lemma *\mathcal{R}_{nd} -from- σ -to- σ s-description:* $\langle \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\sigma \mapsto \sigma s} [\sigma] = \{[\sigma'] \mid \sigma'. \sigma' \in \mathcal{R}_{nd} B \sigma\} \rangle$

proof *safe*

show $\langle \sigma s \in \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\sigma \mapsto \sigma s} [\sigma] \implies \exists \sigma'. \sigma s = [\sigma'] \wedge \sigma' \in \mathcal{R}_{nd} B \sigma \rangle$ **for** σs

by (*induct rule: \mathcal{R}_{nd} .induct, auto simp add: \mathcal{R}_{nd} .init behaviour- σ - σ s-conv(6), metis \mathcal{R}_{nd} .step*)

next

show $\langle \sigma' \in \mathcal{R}_{nd} B \sigma \implies [\sigma'] \in \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\sigma \mapsto \sigma s} [\sigma] \rangle$ **for** σ'

by (*induct rule: \mathcal{R}_{nd} .induct*) (*simp-all add: \mathcal{R}_{nd} .init \mathcal{R}_{nd} .step behaviour- σ - σ s-conv(6)*)

qed

lemma *\mathcal{R}_d -from- σ -to- σ s-description:* $\langle \mathcal{R}_d \text{ d} \langle \langle A \rangle \rangle_{\sigma \mapsto \sigma s} [\sigma] = \{[\sigma'] \mid \sigma'. \sigma' \in \mathcal{R}_d A \sigma\} \rangle$

by (*simp add: \mathcal{R}_{nd} -from- σ -to- σ s-description*

flip: \mathcal{R}_{nd} -from-det-to-ndet from-det-to-ndet- σ - σ s-conv-commute(1))

lemma \mathcal{R}_{nd} -from-singl-to-list-description: $\langle \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = \{[\sigma'] \mid \sigma' \in \mathcal{R}_{nd} B \sigma\} \rangle$

proof *safe*

show $\langle \sigma s \in \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] \implies \exists \sigma'. \sigma s = [\sigma'] \wedge \sigma' \in \mathcal{R}_{nd} B \sigma \rangle$ **for** σs

by (*induct rule: \mathcal{R}_{nd} .induct, auto simp add: \mathcal{R}_{nd} .init behaviour-singl-list-conv(6), metis \mathcal{R}_{nd} .step*)

next

show $\langle \sigma' \in \mathcal{R}_{nd} B \sigma \implies [\sigma'] \in \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] \rangle$ **for** σ'

by (*induct rule: \mathcal{R}_{nd} .induct*) (*simp-all add: \mathcal{R}_{nd} .init \mathcal{R}_{nd} .step behaviour-singl-list-conv(6)*)

qed

lemma \mathcal{R}_d -from-singl-to-list-description: $\langle \mathcal{R}_d \text{ d} \langle \langle A \rangle \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = \{[\sigma'] \mid \sigma' \in \mathcal{R}_d A \sigma\} \rangle$

by (*simp add: \mathcal{R}_{nd} -from-singl-to-list-description*)

flip: \mathcal{R}_{nd} -from-det-to-ndet from-det-to-ndet-singl-list-conv-commute(1))

lemma *length- \mathcal{R}_d -from- σ -to- σs :*

$\langle \sigma s' \in \mathcal{R}_d \text{ d} \langle \langle A \rangle \rangle_{\sigma \hookrightarrow \sigma s} \sigma s \implies \sigma s' = \sigma s \vee \text{length } \sigma s' = 1 \rangle$

by (*simp add: σ - σs -conv-defs*)

(*induct rule: \mathcal{R}_d .induct, simp-all split: option.split-asm*)

lemma *length- \mathcal{R}_{nd} -from- σ -to- σs :*

$\langle \sigma s' \in \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\sigma \hookrightarrow \sigma s} \sigma s \implies \sigma s' = \sigma s \vee \text{length } \sigma s' = 1 \rangle$

by (*simp add: σ - σs -conv-defs*)

(*induct rule: \mathcal{R}_{nd} .induct, auto*)

lemma *length- \mathcal{R}_d -from-singl-to-list:*

$\langle \sigma s' \in \mathcal{R}_d \text{ d} \langle \langle A \rangle \rangle_{\text{singl} \leftrightarrow \text{list}} \sigma s \implies \sigma s' = \sigma s \vee \text{length } \sigma s' = 1 \rangle$

by (*simp add: singl-list-conv-defs*)

(*induct rule: \mathcal{R}_d .induct, simp-all split: option.split-asm*)

lemma *length- \mathcal{R}_{nd} -from-singl-to-list:*

$\langle \sigma s' \in \mathcal{R}_{nd} \text{ nd} \langle \langle B \rangle \rangle_{\text{singl} \leftrightarrow \text{list}} \sigma s \implies \sigma s' = \sigma s \vee \text{length } \sigma s' = 1 \rangle$

by (*simp add: singl-list-conv-defs*)

(*induct rule: \mathcal{R}_{nd} .induct, auto*)

3.3 Normalization

3.3.1 Non-deterministic Case

First version, without final state notion

abbreviation *P-nd-step* :: $\langle [(\sigma, 'a) \text{ enabl}, (\sigma, 'a) \text{ trans}_{nd}, \sigma \Rightarrow (\sigma, 'r) \text{ process}_{ptick}, \sigma] \Rightarrow (\sigma, 'r) \text{ process}_{ptick} \rangle$

where $\langle \text{P-nd-step } \varepsilon_A \tau_A X \sigma \equiv \square e \in \varepsilon_A \sigma \rightarrow \sqcap \sigma' \in \tau_A \sigma e. X \sigma' \rangle$

definition $P\text{-nd} :: \langle ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \Rightarrow 'σ \Rightarrow ('a, 'r) \text{process}_{ptick} \rangle$
 $(\langle P\langle\!\langle\!-\!\rangle\!\rangle_{nd} \rangle 1000)$
where $\langle P\langle\!\langle\!A\!\rangle\!\rangle_{nd} \equiv v X. P\text{-nd}\text{-step} (\varepsilon A) (\tau A) X \rangle$

lemma $P\text{-nd}\text{-step}\text{-constructive} [simp] : \langle \text{constructive} (P\text{-nd}\text{-step} \varepsilon_A \tau_A) \rangle$ **by** $simp$

lemma $P\text{-nd}\text{-step}\text{-cont} [simp] : \langle \forall \sigma a. \text{finite} (\tau_A \sigma a) \Longrightarrow \text{cont} (P\text{-nd}\text{-step} \varepsilon_A \tau_A) \rangle$
by $(simp \text{ add: cont-fun})$

lemma $P\text{-nd}\text{-step}\text{-constructive}\text{-bis} : \langle \text{constructive} (P\text{-nd}\text{-step} (\varepsilon A) (\tau A)) \rangle$ **by** $simp$

lemma $P\text{-nd}\text{-step}\text{-cont}\text{-bis} [simp] : \langle \text{finite}\text{-trans} A \Longrightarrow \text{cont} (P\text{-nd}\text{-step} (\varepsilon A) (\tau A)) \rangle$
by $(simp \text{ add: finite-trans-def})$

lemma $P\text{-nd}\text{-rec} : \langle P\langle\!\langle\!A\!\rangle\!\rangle_{nd} = (\lambda \sigma. P\text{-nd}\text{-step} (\varepsilon A) (\tau A) P\langle\!\langle\!A\!\rangle\!\rangle_{nd} \sigma) \rangle$
by $(unfold P\text{-nd}\text{-def, rule ext, subst restriction-fix-eq, simp-all)$

lemma $P\text{-nd}\text{-is}\text{-fix} : \langle \text{finite}\text{-trans} A \Longrightarrow P\langle\!\langle\!A\!\rangle\!\rangle_{nd} = (\mu X. P\text{-nd}\text{-step} (\varepsilon A) (\tau A) X) \rangle$
by $(simp \text{ add: P-nd-def restriction-fix-is-fix})$

lemma $\text{non}\text{-destructive}\text{-imp}\text{-restriction}\text{-cont} [simp] : \langle \text{non}\text{-destructive} f \Longrightarrow \text{restriction}\text{-cont} f \rangle$
by $(simp \text{ add: non-destructive-on-def})$

lemma $P\text{-nd}\text{-}\omega\text{-useless} : \langle P\langle\!\langle\!A\!\rangle\!\rangle_{nd} = P\langle\!\langle\!A(\omega := \text{some}\text{-}\omega)\!\rangle\!\rangle_{nd} \rangle$
by $(simp \text{ add: P-nd-def } \varepsilon\text{-simps})$

lemma $P\text{-nd}\text{-}\omega\text{-useless}\text{-bis} : \langle P\langle\!\langle\!A\!\rangle\!\rangle_{nd} = P\langle\!\langle\!A(\omega := \lambda \sigma. \{\})\!\rangle\!\rangle_{nd} \rangle$
by $(fact P\text{-nd}\text{-}\omega\text{-useless})$

lemma $P\text{-nd}\text{-induct} [case\text{-names adm base step] : \langle \text{adm}_{\downarrow} P \Longrightarrow P \sigma \Longrightarrow (\bigwedge X. P X \Longrightarrow P (P\text{-nd}\text{-step} (\varepsilon A) (\tau A) X)) \Longrightarrow P P\langle\!\langle\!A\!\rangle\!\rangle_{nd} \rangle$
unfolding $P\text{-nd}\text{-def}$
by $(rule \text{restriction-fix-ind}[OF P\text{-nd}\text{-step}\text{-constructive}\text{-bis}]) \text{ simp-all}$

lemma $P\text{-nd}\text{-induct}\text{-iterated} [consumes 1, case\text{-names adm base step] : \langle \llbracket 0 < k; \text{adm}_{\downarrow} P; P \sigma; \bigwedge X. P X \Longrightarrow P ((P\text{-nd}\text{-step} (\varepsilon A) (\tau A) \overset{\sim}{\sim} k) X) \rrbracket \Longrightarrow P P\langle\!\langle\!A\!\rangle\!\rangle_{nd} \rangle$
unfolding $P\text{-nd}\text{-def}$
by $(rule \text{restriction-fix-ind-iterated}[\mathbf{where} f = \langle P\text{-nd}\text{-step} (\varepsilon A) (\tau A) \rangle]) \text{ auto}$

New version with final state notion where we just have *SKIPS*.

abbreviation $P_{SKIPS}\text{-nd-step} ::$

$\langle [(\sigma, 'a) \text{ enabl}, (\sigma, 'a) \text{ trans}_{nd}, \sigma \Rightarrow 'r \text{ set}, \sigma \Rightarrow ('a, 'r) \text{ process}_{ptick}, \sigma] \Rightarrow$
 $(('a, 'r) \text{ process}_{ptick})$
where $\langle P_{SKIPS}\text{-nd-step } \varepsilon_A \tau_A \omega_A X \sigma \equiv \text{if } \omega_A \sigma = \{\} \text{ then } P\text{-nd-step } \varepsilon_A \tau_A$
 $X \sigma \text{ else } SKIPS (\omega_A \sigma) \rangle$

definition $P_{SKIPS}\text{-nd} :: \langle ('a, 'a, 'r, 'a) A_{nd}\text{-scheme} \Rightarrow 'a \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$
 $\langle P_{SKIPS}\langle\langle - \rangle\rangle_{nd} 1000 \rangle$

where $\langle P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \equiv v X. P_{SKIPS}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A) X \rangle$

lemma $P_{SKIPS}\text{-nd-step-constructive [simp]} : \langle \text{constructive } (P_{SKIPS}\text{-nd-step } \varepsilon_A$
 $\tau_A \omega_A) \rangle$ **by** *auto*

lemma $P_{SKIPS}\text{-nd-step-cont [simp]} : \langle \forall \sigma a. \text{finite } (\tau_A \sigma a) \Longrightarrow \text{cont } (P_{SKIPS}\text{-nd-step}$
 $\varepsilon_A \tau_A \omega_A) \rangle$
by $(\text{simp add: cont-fun})$

lemma $P_{SKIPS}\text{-nd-step-constructive-bis} : \langle \text{constructive } (P_{SKIPS}\text{-nd-step } (\varepsilon A)$
 $(\tau A) (\omega A)) \rangle$ **by** *simp*

lemma $P_{SKIPS}\text{-nd-step-cont-bis [simp]} : \langle \text{finite-trans } A \Longrightarrow \text{cont } (P_{SKIPS}\text{-nd-step}$
 $(\varepsilon A) (\tau A) (\omega A)) \rangle$
by $(\text{simp add: finite-trans-def})$

lemma $P_{SKIPS}\text{-nd-rec} : \langle P_{SKIPS}\langle\langle A \rangle\rangle_{nd} = (\lambda \sigma. P_{SKIPS}\text{-nd-step } (\varepsilon A) (\tau A) (\omega$
 $A) P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \sigma) \rangle$
by $(\text{unfold } P_{SKIPS}\text{-nd-def, rule ext, subst restriction-fix-eq, simp-all})$

lemma $P_{SKIPS}\text{-nd-is-fix} : \langle \text{finite-trans } A \Longrightarrow P_{SKIPS}\langle\langle A \rangle\rangle_{nd} = (\mu X. P_{SKIPS}\text{-nd-step}$
 $(\varepsilon A) (\tau A) (\omega A) X) \rangle$
by $(\text{simp add: } P_{SKIPS}\text{-nd-def restriction-fix-is-fix})$

lemma $P_{SKIPS}\text{-nd-induct [case-names adm base step]} :$
 $\langle \text{adm}_{\downarrow} P \Longrightarrow P \sigma \Longrightarrow (\bigwedge X. P X \Longrightarrow P (P_{SKIPS}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A)$
 $X)) \Longrightarrow P P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \rangle$
unfolding $P_{SKIPS}\text{-nd-def}$
by $(\text{rule restriction-fix-ind}[OF } P_{SKIPS}\text{-nd-step-constructive-bis]) \text{ simp-all}$

lemma $P_{SKIPS}\text{-nd-induct-iterated [consumes 1, case-names adm base step]} :$
 $\langle [\![0 < k; \text{adm}_{\downarrow} P; P \sigma; \bigwedge X. P X \Longrightarrow P ((P_{SKIPS}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A)$
 $\widetilde{\sim} k) X)]\!] \Longrightarrow P P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \rangle$
unfolding $P_{SKIPS}\text{-nd-def}$
by $(\text{rule restriction-fix-ind-iterated}[\text{where } f = \langle P_{SKIPS}\text{-nd-step } (\varepsilon A) (\tau A) (\omega$
 $A) \rangle]) \text{ auto}$

Correspondence when we always have $\omega A \sigma = \{\}$.

lemma P_{SKIPS} -nd-empty- ϱ : $\langle \varrho A = \{\} \implies P_{SKIPS}\langle A \rangle_{nd} = P\langle A \rangle_{nd} \rangle$
by (*simp add: P_{SKIPS}-nd-def P-nd-def ϱ -simps*)

lemma P_{SKIPS} -nd-updated- ω : $\langle P\langle A \rangle_{nd} = P_{SKIPS}\langle A(\omega := \lambda\sigma. \{\}) \rangle_{nd} \rangle$
by (*metis (mono-tags, lifting) P_{SKIPS}-nd-empty- ϱ P-nd- ω -useless-bis ϱ_{nd} .simps empty-Collect-eq select-convs(2) surjective update-convs(2)*)

lemma P_{SKIPS} -nd-empty- ϱ -inter- \mathcal{R}_{nd} :

$\langle P_{SKIPS}\langle A \rangle_{nd} \sigma = P\langle A \rangle_{nd} \sigma \rangle$ **if** $\langle \varrho A \cap \mathcal{R}_{nd} A \sigma = \{\} \rangle$

proof –

have $\langle \sigma' \in \mathcal{R}_{nd} A \sigma \implies P_{SKIPS}\langle A \rangle_{nd} \sigma' = P\langle A \rangle_{nd} \sigma' \rangle$ **for** σ'

proof (*induct A arbitrary: σ' rule: P_{SKIPS}-nd-induct*)

case adm show ?*case* **by** *simp*

next

case base show $\langle P\langle A \rangle_{nd} \sigma' = P\langle A \rangle_{nd} \sigma' \rangle$..

next

case (*step X*)

from *step.prem*s(1) **that have** $\langle \sigma' \notin \varrho A \rangle$ **by** *blast*

hence $\langle \omega A \sigma' = \{\} \rangle$ **by** (*simp add: ϱ -simps*)

thus ?*case*

by (*subst P-nd-rec, auto intro!: mono-Mprefix-eq mono-GlobalNdet-eq (metis (lifting) \mathcal{R}_{nd} .simps step.hyps step.prem*s))

qed

thus $\langle P_{SKIPS}\langle A \rangle_{nd} \sigma = P\langle A \rangle_{nd} \sigma \rangle$ **by** (*simp add: \mathcal{R}_{nd} .init*)

qed

lemma P_{SKIPS} -nd-rec-notin- ϱ :

$\langle \sigma \notin \varrho A \implies P_{SKIPS}\langle A \rangle_{nd} \sigma = P\text{-nd-step}(\varepsilon A)(\tau A) P_{SKIPS}\langle A \rangle_{nd} \sigma \rangle$

by (*subst P_{SKIPS}-nd-rec (simp add: ϱ -simps)*)

lemma P_{SKIPS} -nd-rec-in- ϱ : $\langle \sigma \in \varrho A \implies P_{SKIPS}\langle A \rangle_{nd} \sigma = SKIPS(\omega A \sigma) \rangle$

by (*subst P_{SKIPS}-nd-rec, simp add: ϱ -simps*)

3.3.2 Deterministic Case

First version, without final state notion.

abbreviation P -d-step :: $\langle [(\sigma, 'a) \text{enabl}, (\sigma, 'a) \text{trans}_d, ' \sigma \Rightarrow ('a, 'r) \text{process}_{ptick}, ' \sigma \Rightarrow ('a, 'r) \text{process}_{ptick}] \rangle$

where $\langle P\text{-d-step } \varepsilon_A \tau_A X s \equiv \square e \in \varepsilon_A s \rightarrow X [\tau_A s e] \rangle$

definition P -d :: $\langle (' \sigma, 'a, 'r, ' \alpha) A_d\text{-scheme} \Rightarrow ' \sigma \Rightarrow ('a, 'r) \text{process}_{ptick} \rangle$ ($\langle P\langle - \rangle_d$ 1000)

where $\langle P\langle A \rangle_d \equiv v X. P\text{-d-step}(\varepsilon A)(\tau A) X \rangle$

lemma P -d-step-constructive[*simp*] : $\langle \text{constructive}(P\text{-d-step } \varepsilon_A \tau_A) \rangle$ **by** *simp*

lemmas $P\text{-}d\text{-step}\text{-constructive}\text{-bis} = P\text{-}d\text{-step}\text{-constructive}[of \langle \varepsilon A \rangle \langle \tau A \rangle]$ **for** A

lemma $P\text{-}d\text{-step}\text{-cont}[simp]: \langle cont (P\text{-}d\text{-step } \varepsilon_A \tau_A) \rangle$
by ($simp$ $add: cont\text{-}fun$)

lemmas $P\text{-}d\text{-step}\text{-cont}\text{-bis} = P\text{-}d\text{-step}\text{-cont}[of \langle \varepsilon A \rangle \langle \tau A \rangle]$ **for** A

lemma $P\text{-}d\text{-rec}: \langle P\langle A \rangle_d = (\lambda s. P\text{-}d\text{-step } (\varepsilon A) (\tau A) P\langle A \rangle_d s) \rangle$
by ($unfold$ $P\text{-}d\text{-def}$, $subst$ $restriction\text{-}fix\text{-}eq$) $simp\text{-}all$

lemma $P\text{-}d\text{-is}\text{-}fix : \langle P\langle A \rangle_d = (\mu X. P\text{-}d\text{-step } (\varepsilon A) (\tau A) X) \rangle$
by ($simp$ $add: P\text{-}d\text{-def}$ $restriction\text{-}fix\text{-}is\text{-}fix$)

lemma $P\text{-}d\text{-}\omega\text{-}useless: \langle P\langle A \rangle_d = P\langle A(\omega := some\text{-}\omega) \rangle_d \rangle$
by ($simp$ $add: P\text{-}d\text{-def}$ $\varepsilon\text{-}simps$)

lemma $P\text{-}d\text{-}\omega\text{-}useless\text{-}bis: \langle P\langle A \rangle_d = P\langle A(\omega := \lambda\sigma. \diamond) \rangle_d \rangle$
by ($fact$ $P\text{-}d\text{-}\omega\text{-}useless$)

lemma $P\text{-}d\text{-}induct$ [$case\text{-}names$ adm $base$ $step$] :
 $\langle \llbracket adm_{\downarrow} P; P \sigma; \bigwedge X. P X \implies P (P\text{-}d\text{-step } (\varepsilon A) (\tau A) X) \rrbracket \implies P P\langle A \rangle_d \rangle$
unfolding $P\text{-}d\text{-def}$
by ($rule$ $restriction\text{-}fix\text{-}ind[OF P\text{-}d\text{-step}\text{-constructive}\text{-}bis]$) $simp\text{-}all$

lemma $P\text{-}d\text{-}induct\text{-}iterated$ [$consumes$ 1 , $case\text{-}names$ adm $base$ $step$] :
 $\langle \llbracket 0 < k; adm_{\downarrow} P; P \sigma; \bigwedge X. P X \implies P ((P\text{-}d\text{-step } (\varepsilon A) (\tau A) \overset{\sim}{\sim} k) X) \rrbracket \implies P P\langle A \rangle_d \rangle$
unfolding $P\text{-}d\text{-def}$
by ($rule$ $restriction\text{-}fix\text{-}ind\text{-}iterated[\mathbf{where} f = \langle P\text{-}d\text{-step } (\varepsilon A) (\tau A) \rangle]$) $auto$

New version with final state notion where we just *SKIP*.

abbreviation $P_{SKIP}\text{-}d\text{-}step ::$
 $\langle [(\sigma, 'a) enabl, (\sigma, 'a) trans_d, ' \sigma \Rightarrow 'r \text{ option}, ' \sigma \Rightarrow ('a, 'r) process_{ptick}, ' \sigma \Rightarrow ('a, 'r) process_{ptick}] \rangle$
where $\langle P_{SKIP}\text{-}d\text{-}step \varepsilon_A \tau_A \omega_A X \sigma \equiv case \omega_A \sigma \text{ of } [r] \Rightarrow SKIP r \mid \diamond \Rightarrow P\text{-}d\text{-}step \varepsilon_A \tau_A X \sigma \rangle$

definition $P_{SKIP}\text{-}d :: \langle (' \sigma, 'a, 'r, ' \alpha) A_d\text{-}scheme \Rightarrow ' \sigma \Rightarrow ('a, 'r) process_{ptick} \rangle$
 $(\langle P_{SKIP}\langle - \rangle_d \rangle 1000)$
where $\langle P_{SKIP}\langle A \rangle_d \equiv v X. P_{SKIP}\text{-}d\text{-}step (\varepsilon A) (\tau A) (\omega A) X \rangle$

lemma $P_{SKIP}\text{-}d\text{-}step\text{-constructive}[simp]: \langle constructive (P_{SKIP}\text{-}d\text{-}step \varepsilon_A \tau_A S_{FA}) \rangle$
by ($auto$ $simp$ $add: option\text{-}case\text{-}eq\text{-}if$)

lemmas $P_{SKIP}\text{-}d\text{-}step\text{-constructive}\text{-}bis = P_{SKIP}\text{-}d\text{-}step\text{-constructive}[of \langle \varepsilon A \rangle \langle \tau A \rangle \langle \omega A \rangle]$ **for** A

lemma $P_{SKIPS}\text{-}d\text{-step}\text{-cont}[simp]$: $\langle cont (P_{SKIPS}\text{-}d\text{-step } \varepsilon_A \tau_A \mathcal{S}_{FA}) \rangle$
by (*simp add: cont-fun option.case-eq-if*)

lemmas $P_{SKIPS}\text{-}d\text{-step}\text{-cont}\text{-bis} = P_{SKIPS}\text{-}d\text{-step}\text{-cont}[of \langle \varepsilon A \rangle \langle \tau A \rangle \langle \omega A \rangle]$
for A

lemma $P_{SKIPS}\text{-}d\text{-rec}$: $\langle P_{SKIPS}\langle A \rangle_d = (\lambda \sigma. P_{SKIPS}\text{-}d\text{-step } (\varepsilon A) (\tau A) (\omega A) P_{SKIPS}\langle A \rangle_d \sigma) \rangle$
by (*unfold P_{SKIPS}-d-def, subst restriction-fix-eq*) *auto*

lemma $P_{SKIPS}\text{-}d\text{-is}\text{-fix}$: $\langle P_{SKIPS}\langle A \rangle_d = (\mu X. P_{SKIPS}\text{-}d\text{-step } (\varepsilon A) (\tau A) (\omega A) X) \rangle$
by (*simp add: P_{SKIPS}-d-def restriction-fix-is-fix*)

lemma $P_{SKIPS}\text{-}d\text{-induct}$ [*case-names adm base step*] :
 $\langle adm_{\downarrow} P \implies P \sigma \implies (\bigwedge X. P X \implies P (P_{SKIPS}\text{-}d\text{-step } (\varepsilon A) (\tau A) (\omega A) X)) \implies P P_{SKIPS}\langle A \rangle_d \rangle$
unfolding $P_{SKIPS}\text{-}d\text{-def}$
by (*rule restriction-fix-ind[OF P_{SKIPS}-d-step-constructive-bis]*) *simp-all*

lemma $P_{SKIPS}\text{-}d\text{-induct}\text{-iterated}$ [*consumes 1, case-names adm base step*] :
 $\langle \llbracket 0 < k; adm_{\downarrow} P; P \sigma; \bigwedge X. P X \implies P ((P_{SKIPS}\text{-}d\text{-step } (\varepsilon A) (\tau A) (\omega A) \overset{\sim}{\sim} k) X) \rrbracket \implies P P_{SKIPS}\langle A \rangle_d \rangle$
unfolding $P_{SKIPS}\text{-}d\text{-def}$
by (*rule restriction-fix-ind-iterated[where f = $\langle P_{SKIPS}\text{-}d\text{-step } (\varepsilon A) (\tau A) (\omega A) \rangle$]*) *auto*

Correspondence when we always have $\omega A \sigma = \{\}$.

lemma $P_{SKIPS}\text{-}d\text{-empty}\text{-}\varrho$: $\langle \varrho A = \{\} \implies P_{SKIPS}\langle A \rangle_d = P\langle A \rangle_d \rangle$
by (*simp add: ϱ -simps P-d-def P_{SKIPS}-d-def*)

lemma $P_{SKIPS}\text{-}d\text{-updated}\text{-}\omega$: $\langle P\langle A \rangle_d = P_{SKIPS}\langle A(\omega := \lambda \sigma. \diamond) \rangle_d \rangle$
by (*simp add: P_{SKIPS}-d-empty- ϱ P-d- ω -useless ϱ -simps*)

lemma $P_{SKIPS}\text{-}d\text{-empty}\text{-}\varrho\text{-inter}\text{-}\mathcal{R}_d$:
 $\langle P_{SKIPS}\langle A \rangle_d \sigma = P\langle A \rangle_d \sigma \rangle$ **if** $\langle \varrho A \cap \mathcal{R}_d A \sigma = \{\} \rangle$

proof –

have $\langle \sigma' \in \mathcal{R}_d A \sigma \implies P_{SKIPS}\langle A \rangle_d \sigma' = P\langle A \rangle_d \sigma' \rangle$ **for** σ'

proof (*induct A arbitrary: σ' rule: P_{SKIPS}-d-induct*)

case adm show ?*case* **by** *simp*

next

case base show $\langle P\langle A \rangle_d \sigma' = P\langle A \rangle_d \sigma' \rangle$..

next

case (*step X*)

from *step.premis(1)* **that have** $\langle \sigma' \notin \varrho A \rangle$ **by** *blast*

hence $\langle \omega A \sigma' = \diamond \rangle$ **by** (*simp add: ϱ -simps*)

thus *?case*
by (*subst P-d-rec, auto intro!: mono-Mprefix-eq mono-GlobalNdet-eq*)
(subst (asm) ε -simps, auto, metis (lifting) \mathcal{R}_d .step step.hyps step.premis)
qed
thus $\langle P_{SKIP} \langle A \rangle_d \sigma = P \langle A \rangle_d \sigma \rangle$ **by** (*simp add: \mathcal{R}_d .init*)
qed

lemma *P_{SKIP}-d-rec-notin- ϱ* :
 $\langle \sigma \notin \varrho A \implies P_{SKIP} \langle A \rangle_d \sigma = P\text{-d-step } (\varepsilon A) (\tau A) P_{SKIP} \langle A \rangle_d \sigma \rangle$
by (*subst P_{SKIP}-d-rec (simp add: ϱ -simps)*)

lemma *P_{SKIP}-d-rec-in- ϱ* : $\langle \sigma \in \varrho A \implies P_{SKIP} \langle A \rangle_d \sigma = SKIP [\omega A \sigma] \rangle$
by (*subst P_{SKIP}-d-rec, simp add: ϱ -simps split: option.split*)

3.3.3 Link between deterministic and non-deterministic ProcOmata

lemma *P_{SKIP}-nd-from-det-to-ndet-is-P_{SKIP}-d*: $\langle P_{SKIP} \langle \langle A \rangle_{d \leftrightarrow nd} \rangle_{nd} = P_{SKIP} \langle A \rangle_d \rangle$
proof (*subst P_{SKIP}-nd-def, rule restriction-fix-unique*)
show $\langle \text{constructive } (P_{SKIP}\text{-nd-step } (\varepsilon \langle A \rangle_{d \leftrightarrow nd}) (\tau \langle A \rangle_{d \leftrightarrow nd}) (\omega \langle A \rangle_{d \leftrightarrow nd})) \rangle$
by *simp*
next
show $\langle P_{SKIP}\text{-nd-step } (\varepsilon \langle A \rangle_{d \leftrightarrow nd}) (\tau \langle A \rangle_{d \leftrightarrow nd}) (\omega \langle A \rangle_{d \leftrightarrow nd}) P_{SKIP} \langle A \rangle_d = P_{SKIP} \langle A \rangle_d \rangle$
by (*subst (3) P_{SKIP}-d-rec*)
(rule ext, auto simp add: from-det-to-ndet-def ε -simps split: option.split intro: mono-Mprefix-eq)
qed

corollary *P-nd-from-det-to-ndet-is-P-d*: $\langle P \langle \langle A \rangle_{d \leftrightarrow nd} \rangle_{nd} = P \langle A \rangle_d \rangle$
proof –
have $\langle P \langle \langle A \rangle_{d \leftrightarrow nd} \rangle_{nd} = P_{SKIP} \langle \langle A \rangle_{d \leftrightarrow nd} (\omega := \lambda \sigma. \{\}) \rangle_{nd} \rangle$
by (*fact P_{SKIP}-nd-updated- ω*)
also have $\langle \langle A \rangle_{d \leftrightarrow nd} (\omega := \lambda \sigma. \{\}) \rangle = \langle A (\omega := \lambda \sigma. \diamond) \rangle_{d \leftrightarrow nd} \rangle$
by (*simp add: from-det-to-ndet-def*)
finally show $\langle P \langle \langle A \rangle_{d \leftrightarrow nd} \rangle_{nd} = P \langle A \rangle_d \rangle$
by (*simp add: P_{SKIP}-d-updated- ω P_{SKIP}-nd-from-det-to-ndet-is-P_{SKIP}-d*)
qed

3.3.4 Prove Equality between ProcOmata

This is the easiest method we can think about.

lemma *P-d-eqI*: $\langle (\bigwedge \sigma a. \tau A \sigma a = \tau B \sigma a) \implies P \langle A \rangle_d = P \langle B \rangle_d \rangle$
by (*simp add: P-d-def ε -simps*)

lemma *P-nd-eqI*: $\langle (\bigwedge \sigma a. \tau A \sigma a = \tau B \sigma a) \implies P \langle A \rangle_{nd} = P \langle B \rangle_{nd} \rangle$

by (simp add: P-nd-def ε -simps)

lemma *P_{SKIP}S-d-eqI* :

$\langle (\bigwedge \sigma a. \sigma \notin \varrho A \implies \tau A \sigma a = \tau B \sigma a) \implies (\bigwedge \sigma. \omega A \sigma = \omega B \sigma) \implies P_{SKIP}S\langle\langle A \rangle\rangle_d = P_{SKIP}S\langle\langle B \rangle\rangle_d \rangle$
 by (subst *P_{SKIP}S-d-def*, rule *restriction-fix-unique*, simp)
 (subst (2) *P_{SKIP}S-d-rec*, auto simp add: ε -simps ϱ -simps split: option.split)

lemma *P_{SKIP}S-nd-eqI* :

$\langle (\bigwedge \sigma a. \sigma \notin \varrho A \implies \tau A \sigma a = \tau B \sigma a) \implies (\bigwedge \sigma. \omega A \sigma = \omega B \sigma) \implies P_{SKIP}S\langle\langle A \rangle\rangle_{nd} = P_{SKIP}S\langle\langle B \rangle\rangle_{nd} \rangle$
 by (subst *P_{SKIP}S-nd-def*, rule *restriction-fix-unique*[OF *P_{SKIP}S-nd-step-constructive*])
 (subst (2) *P_{SKIP}S-nd-rec*, auto simp add: ε -simps ϱ -simps split: option.split)

We establish now a much more powerful theorem.

theorem *P_{SKIP}S-nd-eqI-strong*:

assumes *inj-on-f* : $\langle \text{inj-on } f (\mathcal{R}_{nd} A_0 \sigma_0) \rangle$
 and *eq-trans* : $\langle \bigwedge \sigma_0' a. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \tau A_1 (f \sigma_0') a = f ' (\tau A_0 \sigma_0' a) \rangle$
 and *eq-fin* : $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \omega A_1 (f \sigma_0') = \omega A_0 \sigma_0' \rangle$
 shows $\langle P_{SKIP}S\langle\langle A_0 \rangle\rangle_{nd} \sigma_0 = P_{SKIP}S\langle\langle A_1 \rangle\rangle_{nd} (f \sigma_0) \rangle$
proof –
 have $\langle \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies P_{SKIP}S\langle\langle A_1 \rangle\rangle_{nd} (f \sigma_0') = P_{SKIP}S\langle\langle A_0 \rangle\rangle_{nd} \sigma_0' \rangle$ **for** σ_0'
proof (induct *A₁* arbitrary: σ_0' rule: *P_{SKIP}S-nd-induct*)
 case *adm* show ?case by simp
 next
 show $\langle \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies P_{SKIP}S\langle\langle A_0 \rangle\rangle_{nd} (\text{inv-into } (\mathcal{R}_{nd} A_0 \sigma_0) f (f \sigma_0')) = P_{SKIP}S\langle\langle A_0 \rangle\rangle_{nd} \sigma_0' \rangle$ **for** σ_0'
 by (simp add: *inj-on-f*)
 next
 case (step *X*)
from *step.prem*s *eq-trans* **have** $\langle \varepsilon A_0 \sigma_0' = \varepsilon A_1 (f \sigma_0') \rangle$
 by (auto simp add: ε -simps)
moreover **have** $\langle \omega A_1 (f \sigma_0') = \omega A_0 \sigma_0' \rangle$ by (simp add: *eq-fin* *step.prem*s)
ultimately **show** ?case
 by (subst *P_{SKIP}S-nd-rec*, auto)
 (metis (mono-tags, lifting) *\mathcal{R}_{nd} .step* *eq-trans* *mono-GlobalNdet-eq2* *step.hyps* *step.prem*s)
qed
thus $\langle P_{SKIP}S\langle\langle A_0 \rangle\rangle_{nd} \sigma_0 = P_{SKIP}S\langle\langle A_1 \rangle\rangle_{nd} (f \sigma_0) \rangle$ by (simp add: *\mathcal{R}_{nd} .init*)
qed

theorem *P-nd-eqI-strong*:

$\langle [\text{inj-on } f (\mathcal{R}_{nd} A_0 \sigma_0); \bigwedge \sigma_0' a. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \tau A_1 (f \sigma_0') a = f ' (\tau A_0 \sigma_0' a)] \rangle$

$\implies P\langle A_0 \rangle_{nd} \sigma_0 = P\langle A_1 \rangle_{nd} (f \sigma_0)$
by (*unfold* P_{SKIPS} -nd-updated- ω , rule P_{SKIPS} -nd-eqI-strong) *simp-all*

theorem P_{SKIPS} -d-eqI-strong:

assumes $\langle inj\text{-on } f \ (\mathcal{R}_d \ A_0 \ \sigma_0) \rangle$

and $\langle \bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \tau \ A_1 \ (f \ \sigma_0') \ a = \text{map-option } f \ (\tau \ A_0 \ \sigma_0' \ a) \rangle$

and $\langle \bigwedge \sigma_0' . \ \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \omega \ A_1 \ (f \ \sigma_0') = \omega \ A_0 \ \sigma_0' \rangle$

shows $\langle P_{SKIPS}\langle A_0 \rangle_d \ \sigma_0 = P_{SKIPS}\langle A_1 \rangle_d \ (f \ \sigma_0) \rangle$

by (*fold* P_{SKIPS} -nd-from-det-to-ndet-is- P_{SKIPS} -d, rule P_{SKIPS} -nd-eqI-strong)

(*unfold* \mathcal{R}_{nd} -from-det-to-ndet,

simp-all add: assms from-det-to-ndet-def map-option-case split: option.split)

theorem P -d-eqI-strong:

$\langle [inj\text{-on } f \ (\mathcal{R}_d \ A_0 \ \sigma_0);$

$\bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \tau \ A_1 \ (f \ \sigma_0') \ a = \text{map-option } f \ (\tau \ A_0 \ \sigma_0' \ a)]$

$\implies P\langle A_0 \rangle_d \ \sigma_0 = P\langle A_1 \rangle_d \ (f \ \sigma_0) \rangle$

by (*unfold* P_{SKIPS} -d-updated- ω , rule P_{SKIPS} -d-eqI-strong) *simp-all*

lemmas P_{SKIPS} -nd-eqI-strong-id = P_{SKIPS} -nd-eqI-strong[*of id, simplified*]

and P_{SKIPS} -d-eqI-strong-id = P_{SKIPS} -d-eqI-strong

[*of id, simplified id-def option.map-ident, simplified*]

and P -nd-eqI-strong-id = P -nd-eqI-strong[*of id, simplified*]

and P -d-eqI-strong-id = P -d-eqI-strong

[*of id, simplified id-def option.map-ident, simplified*]

corollary P_{SKIPS} -nd-from- σ -to- σs -is- P_{SKIPS} -nd : $\langle P_{SKIPS}\langle_{nd} \langle A \rangle_{\sigma \hookrightarrow \sigma s} \rangle_{nd} [\sigma]$
 $= P_{SKIPS}\langle A \rangle_{nd} \ \sigma \rangle$

by (*auto simp add: image-iff behaviour- σ - σs -conv(6, 8)*

intro!: inj-onI P_{SKIPS}-nd-eqI-strong[symmetric])

corollary P_{SKIPS} -d-from- σ -to- σs -is- P_{SKIPS} -d : $\langle P_{SKIPS}\langle_d \langle A \rangle_{\sigma \hookrightarrow \sigma s} \rangle_d [\sigma] =$
 $P_{SKIPS}\langle A \rangle_d \ \sigma \rangle$

by (*auto simp add: image-iff behaviour- σ - σs -conv(2, 4)*

intro!: inj-onI P_{SKIPS}-d-eqI-strong[symmetric] split: option.split)

corollary P -nd-from- σ -to- σs -is- P -nd : $\langle P\langle_{nd} \langle A \rangle_{\sigma \hookrightarrow \sigma s} \rangle_{nd} [\sigma] = P\langle A \rangle_{nd} \ \sigma \rangle$

by (*auto simp add: image-iff behaviour- σ - σs -conv(6, 8)*

intro!: inj-onI P-nd-eqI-strong[symmetric])

corollary P -d-from- σ -to- σs -is- P -d : $\langle P\langle_d \langle A \rangle_{\sigma \hookrightarrow \sigma s} \rangle_d [\sigma] = P\langle A \rangle_d \ \sigma \rangle$

by (*auto simp add: image-iff behaviour- σ - σs -conv(2, 4)*

intro!: inj-onI P-d-eqI-strong[symmetric] split: option.split)

Behaviour of normalizations. We will use the following methods in combining theories.

fun *recursive-modifier-fun_d* :: ⟨[($'\sigma \times 'a \Rightarrow ' \sigma$ option, ($'\sigma \times 'a \times ' \sigma$ option) list)
 $\Rightarrow (' \sigma \times 'a) \Rightarrow ' \sigma$ option⟩
where ⟨*recursive-modifier-fun_d* f [] = f⟩
| ⟨*recursive-modifier-fun_d* f ((s, e), t) # \mathcal{G}_A) = *recursive-modifier-fun_d* (f((s,
e) := t)) \mathcal{G}_A ⟩

abbreviation *recursive-constructor-A_d* :: ⟨[($'\sigma \times 'a \times ' \sigma$ option) list, $'\sigma \Rightarrow 'r$
option] $\Rightarrow (' \sigma, 'a, 'r) A_d$ ⟩
where ⟨*recursive-constructor-A_d* $\mathcal{G}_A \omega_A \equiv (\tau = \text{curry } (\text{recursive-modifier-fun}_d$
 $(\lambda(s, e). \diamond) \mathcal{G}_A), \omega = \omega_A)$ ⟩

lemma ε -det-breaker:

⟨ ε ($(\tau = \text{curry } (g((\sigma'::'\sigma, a) \mapsto \sigma''::'\sigma)), \omega = \text{some-}\omega, \dots = \text{some-more})$) $\sigma =$
(*if* $\sigma = \sigma'$ then {a} $\cup \varepsilon$ ($\tau = \text{curry } g, \omega = \text{some-}\omega$) σ' else ε ($\tau = \text{curry } g, \omega$
= *some-}\omega*) σ)⟩
by (*auto simp add: ε -simps split: if-splits*)

method ε -det-calc = (*unfold recursive-modifier-fun_d.simps ε -det-breaker, simp cong:*
if-cong)[1]

method τ -det-calc = (*unfold recursive-modifier-fun_d.simps, simp cong: if-cong*)[1]

lemma *bij-Renaming-P_{SKIPS}-nd* :

fixes A :: ⟨($'\sigma, 'a, 'r, ' \alpha$) *A_{nd}-scheme*⟩ **and** f :: ⟨ $'a \Rightarrow 'b$ ⟩ **and** g :: ⟨ $'r \Rightarrow 's$ ⟩
assumes ⟨*bij f*⟩
defines B-def : ⟨B $\equiv (\tau = \lambda \sigma b. \tau A \sigma (\text{inv } f b), \omega = \lambda \sigma. g ' (\omega A \sigma))$ ⟩
shows ⟨*Renaming* (*P_{SKIPS}*⟨A⟩_{nd} σ) f g = *P_{SKIPS}*⟨B⟩_{nd} σ ⟩ (**is** ⟨*?lhs $\sigma = -$* ⟩)
proof (*rule fun-cong*[of *?lhs* ⟨*P_{SKIPS}*⟨B⟩_{nd}⟩ σ])
show ⟨*?lhs = P_{SKIPS}*⟨B⟩_{nd}⟩
proof (*rule restriction-fix-unique*[OF *P_{SKIPS}-nd-step-constructive-bis*[of B],
symmetric, folded P_{SKIPS}-nd-def])
show ⟨*P_{SKIPS}-nd-step* (εB) (τB) (ωB) *?lhs = ?rhs*⟩
proof (*rule ext*)
have * : ⟨ $\varepsilon B \sigma = f ' \varepsilon A \sigma$ ⟩ **for** σ
by (*simp add: B-def ε -simps image-def*) (*metis* ⟨*bij f*⟩ *bij-inv-eq-iff*)
have ** : ⟨*inv f* (f a) = a⟩ **for** a
by (*metis* ⟨*bij f*⟩ *bij-inv-eq-iff*)
have *** : ⟨(THE a'. f a' = f a) = a⟩ **for** a
by (*rule the1-equality', metis* (*mono-tags, lifting*) *Uniq-I assms(1) bij-betw-def*
injD, simp)
show ⟨*P_{SKIPS}-nd-step* (εB) (τB) (ωB) *?lhs $\sigma = ?lhs \sigma$* ⟩ **for** σ

by (*subst* (2) *P_{SKIP}S-nd-rec*, *simp add: **)
 (*auto simp add: ** *** B-def Renaming-distrib-GlobalNdet*
Renaming-Mprefix-image-inj[OF ‹bij f›[THEN bij-is-inj]]
intro: mono-Mprefix-eq)

qed

qed

qed

lemma *bij-Renaming-P_{SKIP}S-d* :

‹*bij f* \implies *Renaming* (*P_{SKIP}S*⟨⟨*A*⟩⟩_{*d*} σ) *f g* =
P_{SKIP}S⟨⟨ $\tau = \lambda \sigma b. \tau A \sigma$ (*inv f b*), $\omega = \lambda \sigma. \text{map-option } g (\omega A \sigma)$ ⟩⟩_{*d*}
 σ ›

by (*subst* (1 2) *P_{SKIP}S-nd-from-det-to-ndet-is-P_{SKIP}S-d[symmetric]*,
subst bij-Renaming-P_{SKIP}S-nd, *assumption*)
 (*rule fun-cong[of - - σ]*, *rule P_{SKIP}S-nd-eqI*,
simp-all add: from-det-to-ndet-def split: option.split)

lemma *RenamingTick-P_{SKIP}S-nd* :

‹*RenamingTick* (*P_{SKIP}S*⟨⟨*A*⟩⟩_{*nd*} σ) *g* = *P_{SKIP}S*⟨⟨ $\tau = \tau A$, $\omega = \lambda \sigma. g \text{ ' } \omega A$
 σ ⟩⟩_{*nd*} σ ›

by (*simp add: bij-Renaming-P_{SKIP}S-nd*)

lemma *RenamingTick-P_{SKIP}S-d* :

‹*RenamingTick* (*P_{SKIP}S*⟨⟨*A*⟩⟩_{*d*} σ) *g* = *P_{SKIP}S*⟨⟨ $\tau = \tau A$, $\omega = \lambda \sigma. \text{map-option}$
 $g (\omega A \sigma)$ ⟩⟩_{*d*} σ ›

by (*simp add: bij-Renaming-P_{SKIP}S-d*)

Chapter 4

Advanced Properties of ProcOmata

Chapter 5

Combining Automata for Synchronization Product

5.1 Definitions

5.1.1 General Patterns

abbreviation *combine-sets-Sync* :: $\langle 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set} \rangle$

where $\langle \text{combine-sets-Sync } S_0 \ E \ S_1 \equiv (S_0 - E - S_1) \cup (S_1 - E - S_0) \cup (S_0 \cap S_1 - E) \cup S_0 \cap S_1 \cap E \rangle$

definition *combine-Sync- ε* ::

$\langle [(\sigma_0, 'e, 'r_0, 'r_0, 'r_0, 'r_0) \ A\text{-scheme}, 'e \ \text{set},$
 $(\sigma_1, 'e, 'r_1, 'r_1, 'r_1, 'r_1) \ A\text{-scheme}, 'e \Rightarrow 'e, 'e \Rightarrow 'e, 'e \Rightarrow 'e] \Rightarrow 'e \ \text{set} \rangle$

where $\langle \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ i_0 \ i_1 \ \sigma s \equiv \text{combine-sets-Sync } (\varepsilon \ A_0 \ (i_0 \ \sigma s)) \ E$
 $(\varepsilon \ A_1 \ (i_1 \ \sigma s)) \rangle$

lemma *combine-Sync- ε -def-bis* :

$\langle \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ i_0 \ i_1 \ \sigma s =$
 $\varepsilon \ A_0 \ (i_0 \ \sigma s) \cup \varepsilon \ A_1 \ (i_1 \ \sigma s) - E \cup \varepsilon \ A_0 \ (i_0 \ \sigma s) \cap \varepsilon \ A_1 \ (i_1 \ \sigma s) \rangle$

by (*auto simp add: combine-Sync- ε -def*)

fun *combine_d-Sync- τ* ::

$\langle [(\sigma_0, 'e, 'r_0, 'r_0, 'r_0, 'r_0) \ A_d\text{-scheme}, 'e \ \text{set}, (\sigma_1, 'e, 'r_1, 'r_1, 'r_1, 'r_1) \ A_d\text{-scheme},$
 $'e \Rightarrow 'e, 'e \Rightarrow 'e, 'e \Rightarrow 'e] \Rightarrow ('e, 'e) \ \text{trans}_d \rangle$

and *combine_{nd}-Sync- τ* ::

$\langle [(\sigma_0, 'e, 'r_0, 'r_0, 'r_0, 'r_0) \ A_{nd}\text{-scheme}, 'e \ \text{set}, (\sigma_1, 'e, 'r_1, 'r_1, 'r_1, 'r_1) \ A_{nd}\text{-scheme},$
 $'e \Rightarrow 'e, 'e \Rightarrow 'e, 'e \Rightarrow 'e] \Rightarrow ('e, 'e) \ \text{trans}_{nd} \rangle$

where $\langle \text{combine}_d\text{-Sync-}\tau \ A_0 \ E \ A_1 \ i_0 \ i_1 \ f \ \sigma s \ e =$

(
 if $e \in \varepsilon \ A_0 \ (i_0 \ \sigma s) \cap \varepsilon \ A_1 \ (i_1 \ \sigma s)$
 then *update-both* $A_0 \ A_1 \ (i_0 \ \sigma s) \ (i_1 \ \sigma s) \ e \ (f\text{-up-opt } f)$
 else if $e \in \varepsilon \ A_0 \ (i_0 \ \sigma s) - E - \varepsilon \ A_1 \ (i_1 \ \sigma s)$
 then *update-left* $A_0 \ (i_0 \ \sigma s) \ (i_1 \ \sigma s) \ e \ (f\text{-up-opt } f) \ (\lambda s. \lfloor s \rfloor)$
 else if $e \in \varepsilon \ A_1 \ (i_1 \ \sigma s) - E - \varepsilon \ A_0 \ (i_0 \ \sigma s)$

$$\begin{aligned} & \text{then update-right } A_1 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-opt } f) (\lambda s. [s]) \\ & \text{else } \diamond \rangle \\ | & \langle \text{combine}_{nd}\text{-Sync-}\tau A_0 E A_1 i_0 i_1 f \sigma s e = \\ & \quad (\text{if } e \in \varepsilon A_0 (i_0 \sigma s) \cap \varepsilon A_1 (i_1 \sigma s) \cap E \\ & \quad \text{then update-both } A_0 A_1 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) \\ & \quad \text{else if } e \in \varepsilon A_0 (i_0 \sigma s) \cap \varepsilon A_1 (i_1 \sigma s) - E \\ & \quad \text{then update-left } A_0 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) (\lambda s. \{s\}) \\ & \quad \quad \cup \text{update-right } A_1 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) (\lambda s. \{s\}) \\ & \quad \text{else if } e \in \varepsilon A_0 (i_0 \sigma s) - E - \varepsilon A_1 (i_1 \sigma s) \\ & \quad \text{then update-left } A_0 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) (\lambda s. \{s\}) \\ & \quad \text{else if } e \in \varepsilon A_1 (i_1 \sigma s) - E - \varepsilon A_0 (i_0 \sigma s) \\ & \quad \text{then update-right } A_1 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) (\lambda s. \{s\}) \\ & \quad \text{else } \{\} \rangle \end{aligned}$$

fun $\text{combine}_d\text{-Sync-}\omega ::$
 $\langle [(\sigma_0, 'e, 'r_0, '\alpha_0) A_d\text{-scheme}, 'e \text{ set}, ('\sigma_1, 'e, 'r_1, '\alpha_1) A_d\text{-scheme},$
 $'\sigma \Rightarrow '\sigma_0, '\sigma \Rightarrow '\sigma_1, 'r_0 \Rightarrow 'r_1 \Rightarrow 'r \text{ option}] \Rightarrow (' \sigma \Rightarrow 'r \text{ option}) \rangle$
and $\text{combine}_{nd}\text{-Sync-}\omega ::$
 $\langle [(\sigma_0, 'e, 'r_0, '\alpha_0) A_{nd}\text{-scheme}, 'e \text{ set}, ('\sigma_1, 'e, 'r_1, '\alpha_1) A_{nd}\text{-scheme},$
 $'\sigma \Rightarrow '\sigma_0, '\sigma \Rightarrow '\sigma_1, 'r_0 \Rightarrow 'r_1 \Rightarrow 'r \text{ option}] \Rightarrow (' \sigma \Rightarrow 'r \text{ set}) \rangle$
where $\langle \text{combine}_d\text{-Sync-}\omega A_0 E A_1 i_0 i_1 g \sigma s =$
 $(\text{case } \omega A_0 (i_0 \sigma s)$
 $\text{of } \diamond \Rightarrow \diamond \mid [r_0] \Rightarrow (\text{case } \omega A_1 (i_1 \sigma s) \text{ of } \diamond \Rightarrow \diamond \mid [r_1] \Rightarrow g r_0 r_1)) \rangle$
 $| \langle \text{combine}_{nd}\text{-Sync-}\omega A_0 E A_1 i_0 i_1 g \sigma s =$
 $\{r \mid r r_0 r_1. g r_0 r_1 = [r] \wedge r_0 \in \omega A_0 (i_0 \sigma s) \wedge r_1 \in \omega A_1 (i_1 \sigma s)\} \rangle$

fun $\text{combine}_d\text{-Sync} ::$
 $\langle [(\sigma_0, 'e, 'r_0, '\alpha_0) A_d\text{-scheme}, 'e \text{ set}, ('\sigma_1, 'e, 'r_1, '\alpha_1) A_d\text{-scheme},$
 $'\sigma \Rightarrow '\sigma_0, '\sigma \Rightarrow '\sigma_1, '\sigma_0 \Rightarrow '\sigma_1 \Rightarrow '\sigma, 'r_0 \Rightarrow 'r_1 \Rightarrow 'r \text{ option}] \Rightarrow (' \sigma, 'e, 'r) A_d \rangle$
and $\text{combine}_{nd}\text{-Sync} ::$
 $\langle [(\sigma_0, 'e, 'r_0, '\alpha_0) A_{nd}\text{-scheme}, 'e \text{ set}, ('\sigma_1, 'e, 'r_1, '\alpha_1) A_{nd}\text{-scheme},$
 $'\sigma \Rightarrow '\sigma_0, '\sigma \Rightarrow '\sigma_1, '\sigma_0 \Rightarrow '\sigma_1 \Rightarrow '\sigma, 'r_0 \Rightarrow 'r_1 \Rightarrow 'r \text{ option}] \Rightarrow (' \sigma, 'e, 'r)$
 $A_{nd} \rangle$
where $\langle \text{combine}_d\text{-Sync } A_0 E A_1 i_0 i_1 f g =$
 $(\tau = \text{combine}_d\text{-Sync-}\tau A_0 E A_1 i_0 i_1 f, \omega = \text{combine}_d\text{-Sync-}\omega A_0 E A_1 i_0$
 $i_1 g) \rangle$
 $| \langle \text{combine}_{nd}\text{-Sync } A_0 E A_1 i_0 i_1 f g =$
 $(\tau = \text{combine}_{nd}\text{-Sync-}\tau A_0 E A_1 i_0 i_1 f, \omega = \text{combine}_{nd}\text{-Sync-}\omega A_0 E A_1$
 $i_0 i_1 g) \rangle$

5.1.2 Specializations

definition $\text{combine}_d\text{Pairlist-Sync} ::$
 $\langle [(\sigma, 'e, 'r, '\alpha) A_d\text{-scheme}, 'e \text{ set}, (' \sigma, 'e, 'r, '\beta) A_d\text{-scheme}] \Rightarrow (' \sigma \text{ list}, 'e, 'r)$
 $A_d \rangle$
where $\langle \text{combine}_d\text{Pairlist-Sync } A_0 E A_1 \equiv$
 $\text{combine}_d\text{-Sync } A_0 E A_1 \text{hd } (\lambda \sigma s. \text{hd } (tl \sigma s)) (\lambda s t. [s, t]) (\lambda s t. \text{if } s = t$
 $\text{then } [s] \text{ else } \diamond) \rangle$
definition $\text{combine}_{nd}\text{Pairlist-Sync} ::$

$\langle [(\sigma, 'e, 'r, ' \alpha) A_{nd}\text{-scheme}, 'e \text{ set}, (\sigma, 'e, 'r, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_{nd} \rangle$
where $\langle \text{combine}_{ndPairlist}\text{-Sync } A_0 E A_1 \equiv$
 $\text{combine}_{nd}\text{-Sync } A_0 E A_1 \text{ hd } (\lambda \sigma s. \text{hd } (\text{tl } \sigma s)) (\lambda s t. [s, t]) (\lambda s t. \text{if } s = t$
 $\text{then } [s] \text{ else } \diamond) \rangle$

definition $\text{combine}_{dPair}\text{-Sync} ::$
 $\langle [(\sigma_0, 'e, 'r, ' \alpha) A_d\text{-scheme}, 'e \text{ set}, (\sigma_1, 'e, 'r, ' \beta) A_d\text{-scheme}] \Rightarrow (' \sigma_0 \times ' \sigma_1, 'e, 'r) A_d \rangle$
where $\langle \text{combine}_{dPair}\text{-Sync } A_0 E A_1 \equiv$
 $\text{combine}_d\text{-Sync } A_0 E A_1 \text{ fst snd Pair } (\lambda s t. \text{if } s = t \text{ then } [s] \text{ else } \diamond) \rangle$

definition $\text{combine}_{ndPair}\text{-Sync} ::$
 $\langle [(\sigma_0, 'e, 'r, ' \alpha) A_{nd}\text{-scheme}, 'e \text{ set}, (\sigma_1, 'e, 'r, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma_0 \times ' \sigma_1, 'e, 'r) A_{nd} \rangle$
where $\langle \text{combine}_{ndPair}\text{-Sync } A_0 E A_1 \equiv$
 $\text{combine}_{nd}\text{-Sync } A_0 E A_1 \text{ fst snd Pair } (\lambda s t. \text{if } s = t \text{ then } [s] \text{ else } \diamond) \rangle$

definition $\text{combine}_{dListstlenL}\text{-Sync} ::$
 $\langle [(\sigma \text{ list}, 'e, 'r, ' \alpha) A_d\text{-scheme}, \text{nat}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r, ' \beta) A_d\text{-scheme}] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_d \rangle$
where $\langle \text{combine}_{dListstlenL}\text{-Sync } A_0 \text{ len}_0 E A_1 \equiv$
 $\text{combine}_d\text{-Sync } A_0 E A_1 (\text{take } \text{len}_0) (\text{drop } \text{len}_0) (@) (\lambda s t. \text{if } s = t \text{ then } [s]$
 $\text{else } \diamond) \rangle$

definition $\text{combine}_{ndListstlenL}\text{-Sync} ::$
 $\langle [(\sigma \text{ list}, 'e, 'r, ' \alpha) A_{nd}\text{-scheme}, \text{nat}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_{nd} \rangle$
where $\langle \text{combine}_{ndListstlenL}\text{-Sync } A_0 \text{ len}_0 E A_1 \equiv$
 $\text{combine}_{nd}\text{-Sync } A_0 E A_1 (\text{take } \text{len}_0) (\text{drop } \text{len}_0) (@) (\lambda s t. \text{if } s = t \text{ then } [s]$
 $\text{else } \diamond) \rangle$

definition $\text{combine}_{dRlist}\text{-Sync} ::$
 $\langle [(\sigma, 'e, 'r, ' \alpha) A_d\text{-scheme}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r, ' \beta) A_d\text{-scheme}] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_d \rangle$
where $\langle \text{combine}_{dRlist}\text{-Sync } A_0 E A_1 \equiv$
 $\text{combine}_d\text{-Sync } A_0 E A_1 \text{ hd tl } (\#) (\lambda s t. \text{if } s = t \text{ then } [s] \text{ else } \diamond) \rangle$

definition $\text{combine}_{ndRlist}\text{-Sync} ::$
 $\langle [(\sigma, 'e, 'r, ' \alpha) A_{nd}\text{-scheme}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_{nd} \rangle$
where $\langle \text{combine}_{ndRlist}\text{-Sync } A_0 E A_1 \equiv$
 $\text{combine}_{nd}\text{-Sync } A_0 E A_1 \text{ hd tl } (\#) (\lambda s t. \text{if } s = t \text{ then } [s] \text{ else } \diamond) \rangle$

lemmas $\text{combine}_{Pairlist}\text{-Sync-defs} = \text{combine}_{dPairlist}\text{-Sync-def } \text{combine}_{ndPairlist}\text{-Sync-def}$
and $\text{combine}_{Pair}\text{-Sync-defs} = \text{combine}_{dPair}\text{-Sync-def } \text{combine}_{ndPair}\text{-Sync-def}$
and $\text{combine}_{ListstlenL}\text{-Sync-defs} = \text{combine}_{dListstlenL}\text{-Sync-def } \text{combine}_{ndListstlenL}\text{-Sync-def}$
and $\text{combine}_{Rlist}\text{-Sync-defs} = \text{combine}_{dRlist}\text{-Sync-def } \text{combine}_{ndRlist}\text{-Sync-def}$

lemmas $\text{combine}\text{-Sync-defs} =$
 $\text{combine}_{Pairlist}\text{-Sync-defs } \text{combine}_{Pair}\text{-Sync-defs } \text{combine}_{ListstlenL}\text{-Sync-defs } \text{combine}_{Rlist}\text{-Sync-defs}$

bundle *combine_{nd}-Sync-syntax* **begin**

notation *combine_{dPairlist}-Sync* ($\langle \langle - \ d \otimes [-]_{Pairlist} \ - \rangle \rangle [0, 0, 0]$)
notation *combine_{ndPairlist}-Sync* ($\langle \langle - \ nd \otimes [-]_{Pairlist} \ - \rangle \rangle [0, 0, 0]$)
notation *combine_{dPair}-Sync* ($\langle \langle - \ d \otimes [-]_{Pair} \ - \rangle \rangle [0, 0, 0]$)
notation *combine_{ndPair}-Sync* ($\langle \langle - \ nd \otimes [-]_{Pair} \ - \rangle \rangle [0, 0, 0]$)
notation *combine_{dListslenL}-Sync* ($\langle \langle - \ d \otimes [-, -]_{ListslenL} \ - \rangle \rangle [0, 0, 0, 0]$)
notation *combine_{ndListslenL}-Sync* ($\langle \langle - \ nd \otimes [-, -]_{ListslenL} \ - \rangle \rangle [0, 0, 0, 0]$)
notation *combine_{dRlist}-Sync* ($\langle \langle - \ d \otimes [-]_{Rlist} \ - \rangle \rangle [0, 0, 0]$)
notation *combine_{ndRlist}-Sync* ($\langle \langle - \ nd \otimes [-]_{Rlist} \ - \rangle \rangle [0, 0, 0]$)

end

unbundle *combine_{nd}-Sync-syntax*

5.2 First Properties

lemma *finite-trans-combine_{nd}-Sync-simps* [*simp*] :

$\langle \text{finite-trans } A_0 \implies \text{finite-trans } A_1 \implies \text{finite-trans } \langle \langle A_0 \ nd \otimes [E]_{Pairlist} A_1 \rangle \rangle \rangle$
 $\langle \text{finite-trans } B_0 \implies \text{finite-trans } B_1 \implies \text{finite-trans } \langle \langle B_0 \ nd \otimes [E]_{Pair} B_1 \rangle \rangle \rangle$
 $\langle \text{finite-trans } C_0 \implies \text{finite-trans } C_1 \implies \text{finite-trans } \langle \langle C_0 \ nd \otimes [len_0, E]_{ListslenL} C_1 \rangle \rangle \rangle$
 $\langle \text{finite-trans } D_0 \implies \text{finite-trans } D_1 \implies \text{finite-trans } \langle \langle D_0 \ nd \otimes [E]_{Rlist} D_1 \rangle \rangle \rangle$

unfolding *combine_{ndPairlist}-Sync-def* *combine_{ndPair}-Sync-def* *combine_{ndListslenL}-Sync-def* *combine_{ndRlist}-Sync-def*

by (*simp-all* *add: finite-trans-def finite-image-set2*)

lemma ε -*combine_{Pairlist}-Sync*:

$\langle \varepsilon \langle \langle A_0 \ d \otimes [E]_{Pairlist} A_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ \text{hd} \ (\text{hd} \circ \text{tl}) \ \sigma s \rangle$
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes [E]_{Pairlist} B_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ B_0 \ E \ B_1 \ \text{hd} \ (\text{hd} \circ \text{tl}) \ \sigma s \rangle$
by (*auto simp* *add: combine-Sync-}\varepsilon-def-bis* *combine_{Pairlist}-Sync-defs* ε -*simps*)

lemma ε -*combine_{Pair}-Sync*:

$\langle \varepsilon \langle \langle A_0 \ d \otimes [E]_{Pair} A_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ \text{fst} \ \text{snd} \ \sigma s \rangle$
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes [E]_{Pair} B_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ B_0 \ E \ B_1 \ \text{fst} \ \text{snd} \ \sigma s \rangle$
by (*auto simp* *add: combine-Sync-}\varepsilon-def-bis* *combine_{Pair}-Sync-defs* ε -*simps*)

lemma ε -*combine_{ListslenL}-Sync*:

$\langle \varepsilon \langle \langle A_0 \ d \otimes [len_0, E]_{ListslenL} A_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ (\text{take } len_0) \ (\text{drop } len_0) \ \sigma s \rangle$
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes [len_0, E]_{ListslenL} B_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ B_0 \ E \ B_1 \ (\text{take } len_0) \ (\text{drop } len_0) \ \sigma s \rangle$
by (*auto simp* *add: combine-Sync-}\varepsilon-def-bis* *combine_{ListslenL}-Sync-defs* ε -*simps*)

lemma ε -*combine_{Rlist}-Sync*:

$\langle \varepsilon \langle \langle A_0 \ d \otimes [E]_{Rlist} A_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ \text{hd} \ \text{tl} \ \sigma s \rangle$
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes [E]_{Rlist} B_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ B_0 \ E \ B_1 \ \text{hd} \ \text{tl} \ \sigma s \rangle$

by (auto simp add: combine-Sync- ε -def-bis combine_{RList}-Sync-defs ε -simps)

lemma ϱ -combine_{Pairlist}-Sync:

$\langle \varrho \langle \langle A_0 \text{ } d \otimes [E]_{\text{Pairlist}} A_1 \rangle \rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho A_0 \wedge \text{hd } (\text{tl } \sigma s) \in \varrho A_1 \wedge \omega A_0 (\text{hd } \sigma s) = \omega A_1 (\text{hd } (\text{tl } \sigma s)) \} \rangle$

$\langle \varrho \langle \langle B_0 \text{ } nd \otimes [E]_{\text{Pairlist}} B_1 \rangle \rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho B_0 \wedge \text{hd } (\text{tl } \sigma s) \in \varrho B_1 \wedge \omega B_0 (\text{hd } \sigma s) \cap \omega B_1 (\text{hd } (\text{tl } \sigma s)) \neq \{ \} \} \rangle$

by (auto simp add: combine_{Pairlist}-Sync-defs ϱ -simps split: option.split)

lemma ϱ -combine_{Pair}-Sync:

$\langle \varrho \langle \langle A_0 \text{ } d \otimes [E]_{\text{Pair}} A_1 \rangle \rangle = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho A_0 \wedge \sigma_1 \in \varrho A_1 \wedge \omega A_0 \sigma_0 = \omega A_1 \sigma_1 \} \rangle$

$\langle \varrho \langle \langle B_0 \text{ } nd \otimes [E]_{\text{Pair}} B_1 \rangle \rangle = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho B_0 \wedge \sigma_1 \in \varrho B_1 \wedge \omega B_0 \sigma_0 \cap \omega B_1 \sigma_1 \neq \{ \} \} \rangle$

by (auto simp add: combine_{Pair}-Sync-defs ϱ -simps split: option.split)

lemma ϱ -combine_{ListstlenL}-Sync:

$\langle \varrho \langle \langle A_0 \text{ } d \otimes [len_0, E]_{\text{ListstlenL}} A_1 \rangle \rangle = \{ \sigma s. \text{take } len_0 \sigma s \in \varrho A_0 \wedge \text{drop } len_0 \sigma s \in \varrho A_1 \wedge \omega A_0 (\text{take } len_0 \sigma s) = \omega A_1 (\text{drop } len_0 \sigma s) \} \rangle$

$\langle \varrho \langle \langle B_0 \text{ } nd \otimes [len_0, E]_{\text{ListstlenL}} B_1 \rangle \rangle = \{ \sigma s. \text{take } len_0 \sigma s \in \varrho B_0 \wedge \text{drop } len_0 \sigma s \in \varrho B_1 \wedge \omega B_0 (\text{take } len_0 \sigma s) \cap \omega B_1 (\text{drop } len_0 \sigma s) \neq \{ \} \} \rangle$

by (auto simp add: combine_{ListstlenL}-Sync-defs ϱ -simps split: option.split)

lemma ϱ -combine_{RList}-Sync:

$\langle \varrho \langle \langle A_0 \text{ } d \otimes [E]_{\text{RList}} A_1 \rangle \rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho A_0 \wedge \text{tl } \sigma s \in \varrho A_1 \wedge \omega A_0 (\text{hd } \sigma s) = \omega A_1 (\text{tl } \sigma s) \} \rangle$

$\langle \varrho \langle \langle B_0 \text{ } nd \otimes [E]_{\text{RList}} B_1 \rangle \rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho B_0 \wedge \text{tl } \sigma s \in \varrho B_1 \wedge \omega B_0 (\text{hd } \sigma s) \cap \omega B_1 (\text{tl } \sigma s) \neq \{ \} \} \rangle$

by (auto simp add: combine_{RList}-Sync-defs ϱ -simps split: option.split)

lemma combine-Sync- τ [simp] :

$\langle \text{combine}_d\text{-Sync-}\tau (A_0(\omega := \text{some-}\omega_0)) E (A_1(\omega := \text{some-}\omega_1)) = \text{combine}_d\text{-Sync-}\tau A_0 E A_1 \rangle$

$\langle \text{combine}_{nd}\text{-Sync-}\tau (B_0(\omega := \text{some-}\omega_0')) E (B_1(\omega := \text{some-}\omega_1')) = \text{combine}_{nd}\text{-Sync-}\tau B_0 E B_1 \rangle$

for $A :: \langle (' \sigma, 'a, ' \sigma \text{ option}, 'r \text{ option}, ' \alpha) A\text{-scheme} \rangle$

and $B :: \langle (' \sigma, 'a, ' \sigma \text{ set}, 'r \text{ set}, ' \alpha) A\text{-scheme} \rangle$

by (intro ext, simp)+

5.3 Reachability

lemma \mathcal{R}_d -combine_{dListstlenL}-Sync-subset:

$\langle \mathcal{R}_d \langle \langle A_0 \text{ } d \otimes [len_0, E]_{\text{ListstlenL}} A_1 \rangle \rangle (s_0 @ s_1) \subseteq \{ t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1 \} \rangle$ (is $\langle ?S_A \subseteq \rightarrow \rangle$)

if same-length- \mathcal{R}_d : $\langle \bigwedge t_0. t_0 \in \mathcal{R}_d A_0 s_0 \implies \text{length } t_0 = \text{len}_0 \rangle$
proof safe
show $\langle t \in ?S_A \implies \exists t_0 t_1. t = t_0 @ t_1 \wedge t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1 \rangle$ **for** t
apply (*induct rule: \mathcal{R}_d .induct, metis \mathcal{R}_d .init*)
by (*simp-all add: combine_{ListslenL-Sync-defs} same-length- \mathcal{R}_d ε -simps split: if-splits*)
(metis (no-types, opaque-lifting) \mathcal{R}_d .simps)+
qed

lemma \mathcal{R}_{nd} -combine_{ndListslenL-Sync-subset}:
 $\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [\text{len}_0, E]_{\text{ListslenL}} B_1 \rangle \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$ (**is** $\langle ?S_B \subseteq - \rangle$)
if same-length- \mathcal{R}_{nd} : $\langle \bigwedge t_0. t_0 \in \mathcal{R}_{nd} B_0 s_0 \implies \text{length } t_0 = \text{len}_0 \rangle$
proof safe
show $\langle t \in ?S_B \implies \exists t_0 t_1. t = t_0 @ t_1 \wedge t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1 \rangle$
for t
apply (*induct rule: \mathcal{R}_{nd} .induct, metis \mathcal{R}_{nd} .init*)
by (*simp-all add: combine_{ListslenL-Sync-defs} same-length- \mathcal{R}_{nd} ε -simps split: if-splits*)
(metis (no-types, opaque-lifting) \mathcal{R}_{nd} .simps)+
qed

lemma \mathcal{R}_d -combine_{dPairlist-Sync-subset}:
 $\langle \mathcal{R}_d \langle \langle A_0 \text{ d} \otimes [E]_{\text{Pairlist}} A_1 \rangle \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1\} \rangle$ (**is** $\langle ?S_A \subseteq - \rangle$)
and \mathcal{R}_{nd} -combine_{ndPairlist-Sync-subset}:
 $\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [E]_{\text{Pairlist}} B_1 \rangle \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$ (**is** $\langle ?S_B \subseteq - \rangle$)
proof safe
show $\langle t \in ?S_A \implies \exists t_0 t_1. t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1 \rangle$ **for** t
by (*\mathcal{R}_d -subset-method defs: combine_{Pairlist-Sync-defs}*)
show $\langle t \in ?S_B \implies \exists t_0 t_1. t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1 \rangle$
for t
by (*\mathcal{R}_{nd} -subset-method defs: combine_{Pairlist-Sync-defs}*)
qed

lemma \mathcal{R}_d -combine_{dPair-Sync-subset}:
 $\langle \mathcal{R}_d \langle \langle A_0 \text{ d} \otimes [E]_{\text{Pair}} A_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_d A_0 s_0 \times \mathcal{R}_d A_1 s_1 \rangle$ (**is** $\langle ?S_A \subseteq - \rangle$)
and \mathcal{R}_{nd} -combine_{ndPair-Sync-subset}:
 $\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [E]_{\text{Pair}} B_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_{nd} B_0 s_0 \times \mathcal{R}_{nd} B_1 s_1 \rangle$ (**is** $\langle ?S_B \subseteq - \rangle$)
proof –
have $\langle t \in ?S_A \implies \text{fst } t \in \mathcal{R}_d A_0 s_0 \wedge \text{snd } t \in \mathcal{R}_d A_1 s_1 \rangle$ **for** t
by (*\mathcal{R}_d -subset-method defs: combine_{Pair-Sync-defs}*)
thus $\langle ?S_A \subseteq \mathcal{R}_d A_0 s_0 \times \mathcal{R}_d A_1 s_1 \rangle$ **by force**
next
have $\langle t \in ?S_B \implies \text{fst } t \in \mathcal{R}_{nd} B_0 s_0 \wedge \text{snd } t \in \mathcal{R}_{nd} B_1 s_1 \rangle$ **for** t
by (*\mathcal{R}_{nd} -subset-method defs: combine_{Pair-Sync-defs}*)
thus $\langle ?S_B \subseteq \mathcal{R}_{nd} B_0 s_0 \times \mathcal{R}_{nd} B_1 s_1 \rangle$ **by force**

qed

lemma \mathcal{R}_d -combine $_d$ Rlist-Sync-subset:

$\langle \mathcal{R}_d \langle \langle A_0 \text{ } d \otimes [E]_{Rlist} A_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \sigma t. t_0 \in \mathcal{R}_d A_0 s_0 \wedge \sigma t \in \mathcal{R}_d A_1 \sigma s\} \rangle$ (is $\langle ?S_A \subseteq - \rangle$)

and \mathcal{R}_{nd} -combine $_{nd}$ Rlist-Sync-subset:

$\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ } nd \otimes [E]_{Rlist} B_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \sigma t. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge \sigma t \in \mathcal{R}_{nd} B_1 \sigma s\} \rangle$ (is $\langle ?S_B \subseteq - \rangle$)

proof safe

show $\langle t \in ?S_A \implies \exists t_0 \sigma t. t = t_0 \# \sigma t \wedge t_0 \in \mathcal{R}_d A_0 s_0 \wedge \sigma t \in \mathcal{R}_d A_1 \sigma s \rangle$ for t

by (\mathcal{R}_d -subset-method defs: combine $_{Rlist}$ -Sync-defs)

next

show $\langle t \in ?S_B \implies \exists t_0 \sigma t. t = t_0 \# \sigma t \wedge t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge \sigma t \in \mathcal{R}_{nd} B_1 \sigma s \rangle$ for t

by (\mathcal{R}_{nd} -subset-method defs: combine $_{Rlist}$ -Sync-defs)

qed

5.4 Normalization

lemma ω -combine $_{Pairlist}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ } d \otimes [E]_{Pairlist} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} [s_0, s_1] = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} nd \otimes [E]_{Pairlist} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle [s_0, s_1] \rangle$

by (auto simp add: combine $_{Pairlist}$ -Sync-defs det-ndet-conv-defs option.case-eq-if)

lemma ω -combine $_{Pair}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ } d \otimes [E]_{Pair} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (s_0, s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} nd \otimes [E]_{Pair} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle (s_0, s_1) \rangle$

by (auto simp add: combine $_{Pair}$ -Sync-defs det-ndet-conv-defs option.case-eq-if)

lemma ω -combine $_{ListslenL}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ } d \otimes [len_0, E]_{ListslenL} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 @ \sigma s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} nd \otimes [len_0, E]_{ListslenL} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle (\sigma s_0 @ \sigma s_1) \rangle$

by (auto simp add: combine $_{ListslenL}$ -Sync-defs det-ndet-conv-defs option.case-eq-if)

lemma ω -combine $_{Rlist}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ } d \otimes [E]_{Rlist} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (s_0 \# \sigma s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} nd \otimes [E]_{Rlist} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle (s_0 \# \sigma s_1) \rangle$

by (auto simp add: combine $_{Rlist}$ -Sync-defs det-ndet-conv-defs option.case-eq-if)

lemma τ -combine $_{Pairlist}$ -Sync-behaviour-when-indep:

$\langle \varepsilon A_0 s_0 \cap \varepsilon A_1 s_1 \subseteq E \implies$

$\tau \langle \langle \langle A_0 \text{ } d \otimes [E]_{Pairlist} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} [s_0, s_1] e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} nd \otimes [E]_{Pairlist} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle [s_0, s_1] e \rangle$

by (auto simp add: combine $_{Pairlist}$ -Sync-defs det-ndet-conv-defs option.case-eq-if ε -simps)

lemma τ -combine_{Pair}-Sync-behaviour-when-indep:

$\langle \varepsilon A_0 s_0 \cap \varepsilon A_1 s_1 \subseteq E \implies$
 $\tau \langle \langle A_0 \ d \otimes [E]_{Pair} A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0, s_1) e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Pair} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle$
 $(s_0, s_1) e \rangle$
by (auto simp add: combine_{Pair}-Sync-defs det-ndet-conv-defs option.case-eq-if
 ε -simps)

lemma τ -combine_{ListLenL}-Sync-behaviour-when-indep:

$\langle \varepsilon A_0 \sigma s_0 \cap \varepsilon A_1 \sigma s_1 \subseteq E \implies \text{length } \sigma s_0 = \text{len}_0 \implies$
 $\tau \langle \langle A_0 \ d \otimes [len_0, E]_{ListLenL} A_1 \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 @ \sigma s_1) e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [len_0,$
 $E]_{ListLenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (\sigma s_0 @ \sigma s_1) e \rangle$
by (auto simp add: combine_{ListLenL}-Sync-defs det-ndet-conv-defs option.case-eq-if
 ε -simps)

lemma τ -combine_{Rlist}-Sync-behaviour-when-indep:

$\langle \varepsilon A_0 s_0 \cap \varepsilon A_1 \sigma s_1 \subseteq E \implies$
 $\tau \langle \langle A_0 \ d \otimes [E]_{Rlist} A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0 \# \sigma s_1) e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Rlist}$
 $\langle A_1 \rangle_{d \hookrightarrow nd} \rangle (s_0 \# \sigma s_1) e \rangle$
by (auto simp add: combine_{Rlist}-Sync-defs det-ndet-conv-defs option.case-eq-if
 ε -simps)

method *P*_{SKIPs}-when-indep-method **uses** *R*-*d*-subset =

fold *P*_{SKIPs}-nd-from-det-to-ndet-is-*P*_{SKIPs}-*d*,
rule *P*_{SKIPs}-nd-eqI-strong-id,
unfold \mathcal{R}_{nd} -from-det-to-ndet,
all $\langle \text{drule set-mp}[OF \ i\text{-}d\text{-subset, rotated}] \rangle$

method *P*-when-indep-method **uses** *R*-*d*-subset =

fold *P*-nd-from-det-to-ndet-is-*P*-*d*,
rule *P*-nd-eqI-strong-id,
unfold \mathcal{R}_{nd} -from-det-to-ndet,
all $\langle \text{drule set-mp}[OF \ i\text{-}d\text{-subset, rotated}] \rangle$

lemma *P*_{SKIPs}-combine_{PairList}-Sync-behaviour-when-indep:

$\langle P_{SKIPs} \langle \langle A_0 \ d \otimes [E]_{PairList} A_1 \rangle \rangle_d [s_0, s_1] = P_{SKIPs} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{PairList}$
 $\langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} [s_0, s_1] \rangle$
if $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$
by (*P*_{SKIPs}-when-indep-method *R*-*d*-subset: \mathcal{R}_d -combine_{dPairList}-Sync-subset,
simp-all)
(metis τ -combine_{PairList}-Sync-behaviour-when-indep *indep-enablD* *that*,
metis ω -combine_{PairList}-Sync-behaviour)

lemma *P*-combine_{PairList}-Sync-behaviour-when-indep:

$\langle P \langle \langle A_0 \ d \otimes [E]_{PairList} A_1 \rangle \rangle_d [s_0, s_1] = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{PairList} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd}$
 $[s_0, s_1] \rangle$

if $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$
by (*P-when-indep-method R-d-subset: $\mathcal{R}_d\text{-combine}_{d\text{Pairlist-Sync-subset}}$, simp-all*)
(metis $\tau\text{-combine}_{\text{Pairlist-Sync-behaviour-when-indep indep-enablD}}$ that)

lemma *P_{SKIPs}-combine_{Pair-Sync-behaviour-when-indep}*:

$\langle P_{SKIPs} \langle \langle A_0 \ d \otimes [E]_{\text{Pair}} \ A_1 \rangle \rangle_d (s_0, s_1) = P_{SKIPs} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{\text{Pair}} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0, s_1) \rangle$

if $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$

by (*P_{SKIPs}-when-indep-method R-d-subset: $\mathcal{R}_d\text{-combine}_{d\text{Pair-Sync-subset}}$, all*
(elim SigmaE))

*(metis $\tau\text{-combine}_{\text{Pair-Sync-behaviour-when-indep indep-enablD}}$ that,
auto simp add: $\omega\text{-combine}_{\text{Pair-Sync-behaviour option.case-eq-if}}$)*

lemma *P-combine_{Pair-Sync-behaviour-when-indep}*:

$\langle P \langle \langle A_0 \ d \otimes [E]_{\text{Pair}} \ A_1 \rangle \rangle_d (s_0, s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{\text{Pair}} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0, s_1) \rangle$

if $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$

by (*P-when-indep-method R-d-subset: $\mathcal{R}_d\text{-combine}_{d\text{Pair-Sync-subset}}$, elim SigmaE*)

(metis $\tau\text{-combine}_{\text{Pair-Sync-behaviour-when-indep indep-enablD}}$ that)

lemma *P_{SKIPs}-combine_{ListLenL-Sync-behaviour-when-indep}*:

$\langle P_{SKIPs} \langle \langle A_0 \ d \otimes [len_0, E]_{\text{ListLenL}} \ A_1 \rangle \rangle_d (\sigma s_0 \ @ \ \sigma s_1) = P_{SKIPs} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [len_0, E]_{\text{ListLenL}} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 \ @ \ \sigma s_1) \rangle$

if $\langle \text{indep-enabl } A_0 \ \sigma s_0 \ E \ A_1 \ \sigma s_1 \rangle$ **and** $\langle \bigwedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d \ A_0 \ \sigma s_0 \implies \text{length } \sigma t_0 = \text{len}_0 \rangle$

by (*P_{SKIPs}-when-indep-method R-d-subset: $\mathcal{R}_d\text{-combine}_{d\text{ListLenL-Sync-subset}}$, simp-all add: that(2))*

*(metis $\tau\text{-combine}_{\text{ListLenL-Sync-behaviour-when-indep indep-enablD}}$ that,
metis $\omega\text{-combine}_{\text{ListLenL-Sync-behaviour}}$)*

lemma *P-combine_{ListLenL-Sync-behaviour-when-indep}*:

$\langle P \langle \langle A_0 \ d \otimes [len_0, E]_{\text{ListLenL}} \ A_1 \rangle \rangle_d (\sigma s_0 \ @ \ \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [len_0, E]_{\text{ListLenL}} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 \ @ \ \sigma s_1) \rangle$

if $\langle \text{indep-enabl } A_0 \ \sigma s_0 \ E \ A_1 \ \sigma s_1 \rangle$ **and** $\langle \bigwedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d \ A_0 \ \sigma s_0 \implies \text{length } \sigma t_0 = \text{len}_0 \rangle$

by (*P-when-indep-method R-d-subset: $\mathcal{R}_d\text{-combine}_{d\text{ListLenL-Sync-subset}}$, simp-all add: that(2))*

(metis $\tau\text{-combine}_{\text{ListLenL-Sync-behaviour-when-indep indep-enablD}}$ that)

lemma *P_{SKIPs}-combine_{Rlist-Sync-behaviour-when-indep}*:

$\langle P_{SKIPs} \langle \langle A_0 \ d \otimes [E]_{\text{Rlist}} \ A_1 \rangle \rangle_d (s_0 \ \# \ \sigma s_1) = P_{SKIPs} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{\text{Rlist}} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0 \ \# \ \sigma s_1) \rangle$

if $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ \sigma s_1 \rangle$

by (*P_{SKIPs}-when-indep-method R-d-subset: $\mathcal{R}_d\text{-combine}_{d\text{Rlist-Sync-subset}}$, simp-all*)

(metis $\tau\text{-combine}_{\text{Rlist-Sync-behaviour-when-indep indep-enablD}}$ that,

metis ω -combine_{Relist}-Sync-behaviour)

lemma *P-combine_{Relist}-Sync-behaviour-when-indep*:

$\langle P \langle \langle A_0 \rangle_d \otimes [E]_{Relist} A_1 \rangle_d (s_0 \# \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \mapsto nd} \rangle_{nd} \otimes [E]_{Relist} \langle A_1 \rangle_{d \mapsto nd} \rangle_{nd} (s_0 \# \sigma s_1) \rangle$

if $\langle indep-enabl A_0 s_0 E A_1 \sigma s_1 \rangle$

by (*P-when-indep-method R-d-subset: \mathcal{R}_d -combine_{dRelist}-Sync-subset, simp*)
(*metis τ -combine_{Relist}-Sync-behaviour-when-indep indep-enablD that*)

Chapter 6

Compactification of Synchronization Product

6.1 Iterated Combine

6.1.1 Definitions

fun *iterated-combine_d-Sync* :: $\langle 'e \text{ set} \Rightarrow (' \sigma, 'e, 'r) A_d \text{ list} \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_d \rangle$
 $\langle \langle \langle d \otimes [-] - \rangle \rangle [\theta, \theta] \rangle$
where $\langle \langle \langle d \otimes [E] [] \rangle \rangle = \langle \tau = \lambda \sigma s a. \diamond, \omega = \lambda \sigma s. \diamond \rangle \rangle$
 $| \quad \langle \langle \langle d \otimes [E] [A_0] \rangle \rangle = \langle d \langle A_0 \rangle_{\sigma \leftrightarrow \sigma s} \rangle \rangle$
 $| \quad \langle \langle \langle d \otimes [E] A_0 \# A_1 \# As \rangle \rangle = \langle A_0 d \otimes [E]_{Rlist} \langle \langle d \otimes [E] A_1 \# As \rangle \rangle \rangle \rangle$

fun *iterated-combine_{nd}-Sync* :: $\langle 'e \text{ set} \Rightarrow (' \sigma, 'e, 'r) A_{nd} \text{ list} \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_{nd} \rangle$
 $\langle \langle \langle nd \otimes [-] - \rangle \rangle [\theta, \theta] \rangle$
where $\langle \langle \langle nd \otimes [E] [] \rangle \rangle = \langle \tau = \lambda \sigma s a. \{ \}, \omega = \lambda \sigma s. \{ \} \rangle \rangle$
 $| \quad \langle \langle \langle nd \otimes [E] [A_0] \rangle \rangle = \langle nd \langle A_0 \rangle_{\sigma \leftrightarrow \sigma s} \rangle \rangle$
 $| \quad \langle \langle \langle nd \otimes [E] A_0 \# A_1 \# As \rangle \rangle = \langle A_0 nd \otimes [E]_{Rlist} \langle \langle nd \otimes [E] A_1 \# As \rangle \rangle \rangle \rangle$

lemma *iterated-combine_d-Sync-simps-bis*: $\langle As \neq [] \implies \langle \langle d \otimes [E] A_0 \# As \rangle \rangle = \langle A_0 d \otimes [E]_{Rlist} \langle \langle d \otimes [E] As \rangle \rangle \rangle \rangle$
and *iterated-combine_{nd}-Sync-simps-bis*: $\langle Bs \neq [] \implies \langle \langle nd \otimes [E] B_0 \# Bs \rangle \rangle = \langle B_0 nd \otimes [E]_{Rlist} \langle \langle nd \otimes [E] Bs \rangle \rangle \rangle \rangle$
by (*induct As, simp-all*) (*induct Bs, simp-all*)

fun *iterated-combine_d-Sync-ε* :: $\langle (' \sigma, 'e, 'r, ' \alpha) A_d \text{-scheme list} \Rightarrow 'e \text{ set} \Rightarrow ' \sigma \text{ list} \Rightarrow 'e \text{ set} \rangle$
where $\langle \langle \langle \langle \langle \langle d \otimes [E] \sigma s = \{ \} \rangle \rangle \rangle \rangle \rangle$
 $| \quad \langle \langle \langle \langle \langle \langle d \otimes [E] \sigma s = \varepsilon A_0 (hd \sigma s) \rangle \rangle \rangle \rangle \rangle$
 $| \quad \langle \langle \langle \langle \langle \langle \langle d \otimes [E] \sigma s = \text{combine-sets-Sync} (\varepsilon A_0 (hd \sigma s)) E (\text{iterated-combine}_d\text{-Sync-}\varepsilon (A_1 \# As) E (tl \sigma s)) \rangle \rangle \rangle \rangle \rangle \rangle$

fun *iterated-combine_{nd}-Sync-ε* :: ⟨('σ, 'e, 'r, 'α) *A_{nd}-scheme list* ⇒ 'e set ⇒ 'σ list ⇒ 'e set⟩
where ⟨*iterated-combine_{nd}-Sync-ε* [] *E* σs = {}⟩
| ⟨*iterated-combine_{nd}-Sync-ε* [A₀] *E* σs = ε A₀ (hd σs)⟩
| ⟨*iterated-combine_{nd}-Sync-ε* (A₀ # A₁ # As) *E* σs =
combine-sets-Sync (ε A₀ (hd σs)) *E* (*iterated-combine_{nd}-Sync-ε* (A₁ # As)
E (tl σs))⟩

lemma *iterated-combine_d-Sync-ε-simps-bis*:
⟨As ≠ [] ⇒ *iterated-combine_d-Sync-ε* (A₀ # As) *E* σs =
combine-sets-Sync (ε A₀ (hd σs)) *E* (*iterated-combine_d-Sync-ε* As *E*
(tl σs))⟩
and *iterated-combine_{nd}-Sync-ε-simps-bis*:
⟨Bs ≠ [] ⇒ *iterated-combine_{nd}-Sync-ε* (B₀ # Bs) *E* σs =
combine-sets-Sync (ε B₀ (hd σs)) *E* (*iterated-combine_{nd}-Sync-ε* Bs
E (tl σs))⟩
by (*induct As, simp-all*) (*induct Bs, simp-all*)

6.1.2 First Results

lemma *ε-iterated-combine_d-Sync*:
⟨length σs = length As ⇒ ε ⟨_d⊗ [E] As⟩ σs = *iterated-combine_d-Sync-ε* As *E*
σs⟩
by (*induct σs As rule: induct-2-lists012*)
(*simp-all add: ε-combine_{Rlist}-Sync combine-Sync-ε-def*)

lemma *ε-iterated-combine_{nd}-Sync*:
⟨length σs = length Bs ⇒ ε ⟨_{nd}⊗ [E] Bs⟩ σs = *iterated-combine_{nd}-Sync-ε* Bs
E σs⟩
by (*induct σs Bs rule: induct-2-lists012*)
(*simp-all add: ε-combine_{Rlist}-Sync combine-Sync-ε-def*)

lemma *combine_{ListslenL}-Sync-combine_{Rlist}-Sync-eq*:
⟨ε ⟨_d⟨A₀⟩_{σ↔σs} _d⊗ [1, E] _{ListslenL} A₁⟩ σs = ε ⟨A₀ _d⊗ [E] _{Rlist} A₁⟩ σs⟩
⟨τ ⟨_d⟨A₀⟩_{σ↔σs} _d⊗ [1, E] _{ListslenL} A₁⟩ (s₀ # σs) e = τ ⟨A₀ _d⊗ [E] _{Rlist} A₁⟩ (s₀
σs) e⟩
⟨ε ⟨_{nd}⟨B₀⟩_{σ↔σs} _{nd}⊗ [1, E] _{ListslenL} B₁⟩ σs = ε ⟨B₀ _{nd}⊗ [E] _{Rlist} B₁⟩ σs⟩
⟨τ ⟨_{nd}⟨B₀⟩_{σ↔σs} _{nd}⊗ [1, E] _{ListslenL} B₁⟩ (s₀ # σs) e = τ ⟨B₀ _{nd}⊗ [E] _{Rlist} B₁⟩
(s₀ # σs) e⟩
by (*simp-all add: ε-combine_{ListslenL}-Sync ε-combine_{Rlist}-Sync drop-Suc com-
bine-Sync-ε-def*,
auto simp add: combine_{ListslenL}-Sync-defs combine_{Rlist}-Sync-defs σ-σs-conv-defs
ε-simps)
(*metis append-Cons append-Nil*)

lemma *combinePairlist-Sync-and-iterated-combine_{nd}-Sync-eq*:
 $\langle \varepsilon \langle A_0 \ d \otimes [E]_{Pairlist} A_1 \rangle [s_0, s_1] = \varepsilon \langle d \otimes [E] [A_0, A_1] \rangle [s_0, s_1] \rangle$
 $\langle \tau \langle A_0 \ d \otimes [E]_{Pairlist} A_1 \rangle [s_0, s_1] \ e = \tau \langle d \otimes [E] [A_0, A_1] \rangle [s_0, s_1] \ e \rangle$
 $\langle \varepsilon \langle B_0 \ nd \otimes [E]_{Pairlist} B_1 \rangle [s_0, s_1] = \varepsilon \langle nd \otimes [E] [B_0, B_1] \rangle [s_0, s_1] \rangle$
 $\langle \tau \langle B_0 \ nd \otimes [E]_{Pairlist} B_1 \rangle [s_0, s_1] \ e = \tau \langle nd \otimes [E] [B_0, B_1] \rangle [s_0, s_1] \ e \rangle$
by (*simp-all add: ε -combine_{Pairlist}-Sync ε -combine_{Rlist}-Sync*)
(auto simp add: combine_{Pairlist}-Sync-defs combine_{Rlist}-Sync-defs σ - σ s-conv-defs)
option.case-eq-if ε -simps combine-Sync- ε -def)

lemmas *combine_{Pairlist}-Sync-and-combine_{Rlist}-Sync-eq = combine_{Pairlist}-Sync-and-iterated-combine_{nd}-Sync-eq[simplified]*

6.1.3 Reachability

lemma *same-length- \mathcal{R}_d -iterated-combine_d-Sync-description*:
 $\langle \text{length } \sigma s = \text{length } As \implies \sigma s' \in \mathcal{R}_d \langle d \otimes [E] As \rangle \sigma s \implies$
 $\text{length } \sigma s' = \text{length } As \wedge (\forall i < \text{length } As. \sigma s' ! i \in \mathcal{R}_d (As ! i) (\sigma s ! i)) \rangle$
proof (*induct σs As arbitrary: $\sigma s'$ rule: induct-2-lists012*)
case Nil thus *?case by simp*
next
case (*single $\sigma 1 A1$*)
thus *?case by (auto simp add: \mathcal{R}_d -from- σ -to- σ s-description)*
next
case (*Cons $\sigma 1 \sigma 2 \sigma s A1 A2 As$*)
from *set-mp[OF \mathcal{R}_d -combine_{dRlist}-Sync-subset, OF Cons.premis[simplified]]*
obtain $\sigma 1' \sigma s''$ **where** $\langle \sigma s' = \sigma 1' \# \sigma s'' \rangle \langle \sigma 1' \in \mathcal{R}_d A1 \sigma 1 \rangle$
 $\langle \sigma s'' \in \mathcal{R}_d \langle d \otimes [E] A2 \# As \rangle (\sigma 2 \# \sigma s) \rangle$ **by** *blast*
from *Cons.hyps(3)[OF this(3)] this(1, 2)*
show *?case using less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc by simp auto*
qed

lemma *same-length- \mathcal{R}_{nd} -iterated-combine_{nd}-Sync-description*:
 $\langle \text{length } \sigma s = \text{length } Bs \implies \sigma s' \in \mathcal{R}_{nd} \langle nd \otimes [E] Bs \rangle \sigma s \implies$
 $\text{length } \sigma s' = \text{length } Bs \wedge (\forall i < \text{length } Bs. \sigma s' ! i \in \mathcal{R}_{nd} (Bs ! i) (\sigma s ! i)) \rangle$
proof (*induct σs Bs arbitrary: $\sigma s'$ rule: induct-2-lists012*)
case Nil thus *?case by simp*
next
case (*single $\sigma 1 B1$*)
thus *?case by (auto simp add: \mathcal{R}_{nd} -from- σ -to- σ s-description)*
next
case (*Cons $\sigma 1 \sigma 2 \sigma s A1 A2 As$*)
from *set-mp[OF \mathcal{R}_{nd} -combine_{ndRlist}-Sync-subset, OF Cons.premis[simplified]]*
obtain $\sigma 1' \sigma s''$ **where** $\langle \sigma s' = \sigma 1' \# \sigma s'' \rangle \langle \sigma 1' \in \mathcal{R}_{nd} A1 \sigma 1 \rangle$
 $\langle \sigma s'' \in \mathcal{R}_{nd} \langle nd \otimes [E] A2 \# As \rangle (\sigma 2 \# \sigma s) \rangle$ **by** *blast*
from *Cons.hyps(3)[OF this(3)] this(1, 2)*

show *?case* **using** *less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc* **by** *simp auto*
qed

6.1.4 Transmission of Properties

lemma *finite-trans-transmission-to-iterated-combine_{nd}-Sync*:

$\langle (\bigwedge A. A \in \text{set } As \implies \text{finite-trans } A) \implies \text{finite-trans } \llbracket_{nd} \otimes [E] As \rrbracket \rangle$

by (*induct As rule: induct-list012*)

(*auto simp add: σ - σ s-conv-defs combine_{Rlist}-Sync-defs finite-trans-def finite-image-set2*)

lemma *ρ -disjoint- ε -transmission-to-iterated-combine_d-Sync*:

$\langle (\bigwedge A. A \in \text{set } As \implies \rho\text{-disjoint-}\varepsilon A) \implies \rho\text{-disjoint-}\varepsilon \llbracket_d \otimes [E] As \rrbracket \rangle$

by (*induct As rule: induct-list012*)

(*simp-all add: ρ -combine_{Rlist}-Sync ε -combine_{Rlist}-Sync ρ -disjoint- ε -def combine-Sync- ε -def*)

lemma *ρ -disjoint- ε -transmission-to-iterated-combine_{nd}-Sync*:

$\langle (\bigwedge A. A \in \text{set } As \implies \rho\text{-disjoint-}\varepsilon A) \implies \rho\text{-disjoint-}\varepsilon \llbracket_{nd} \otimes [E] As \rrbracket \rangle$

by (*induct As rule: induct-list012*)

(*simp-all add: ρ -combine_{Rlist}-Sync ε -combine_{Rlist}-Sync ρ -disjoint- ε -def combine-Sync- ε -def*)

lemma *at-most-1-elem-term-transmission-to-iterated-combine_{nd}-Sync*:

$\langle (\bigwedge A. A \in \text{set } As \implies \text{at-most-1-elem-term } A) \implies \text{at-most-1-elem-term } \llbracket_{nd} \otimes [E] As \rrbracket \rangle$

by (*induct As rule: induct-list012*,

simp-all add: at-most-1-elem-term-def σ - σ s-conv-defs combine_{Rlist}-Sync-defs)
fastforce

lemma *same-length-indep-transmission-to-iterated-combine_d-Sync*:

$\langle \text{length } \sigma s = \text{length } As \implies$

$(\bigwedge i j. i \leq \text{length } As \implies j \leq \text{length } As \implies i \neq j \implies$

$\text{indep-enabl } ((A_0 \# As) ! i) ((s_0 \# \sigma s) ! i) E ((A_0 \# As) ! j) ((s_0 \# \sigma s) ! j))$

\implies

$\text{indep-enabl } A_0 s_0 E \llbracket_d \otimes [E] As \rrbracket \sigma s \rangle$

proof (*induct $\sigma s As$ rule: induct-2-lists012*)

case *Nil*

then show *?case* **by** (*simp add: indep-enablI*)

next

case (*single $\sigma_1 A_1$*)

from *single.premis[of 0 1]* **show** *?case*

by (*auto simp add: \mathcal{R}_d -from- σ -to- σ s-description*

intro!: indep-enablI dest!: indep-enablD)

next

case (*Cons $\sigma_1 \sigma_2 \sigma s A_1 A_2 As$*)

show *?case*

proof (*rule indep-enablI*)

fix *t₀ ts* **assume** *assms* : $\langle t_0 \in \mathcal{R}_d A_0 s_0 \rangle \langle ts \in \mathcal{R}_d \llbracket_d \otimes [E] A_1 \# A_2 \# As \rrbracket \rangle$

```

( $\sigma_1 \# \sigma_2 \# \sigma_s$ )
from assms(2) obtain  $t_1 \ ts'$  where  $\langle ts = t_1 \# ts' \rangle$ 
  by (metis Cons.hyps(1) Zero-not-Suc length-Cons list.exhaust-sel list.size(3)
    same-length- $\mathcal{R}_d$ -iterated-combine $_d$ -Sync-description)
with assms(2)[simplified, THEN set-mp[OF  $\mathcal{R}_d$ -combine $_d$ RList-Sync-subset]]
have  $\langle ts' \in \mathcal{R}_d \llbracket E \rrbracket A_2 \# As \rangle$  ( $\sigma_2 \# \sigma_s$ ) by simp

have  $\langle indep-enabl \ A_0 \ s_0 \ E \llbracket E \rrbracket A_2 \# As \rangle$  ( $\sigma_2 \# \sigma_s$ )
proof (rule Cons.hyps(3))
  show  $\langle i \leq length \ (A_2 \# As) \implies j \leq length \ (A_2 \# As) \implies i \neq j \implies$ 
    indep-enabl  $((A_0 \# A_2 \# As) ! i) ((s_0 \# \sigma_2 \# \sigma_s) ! i) \ E$ 
     $((A_0 \# A_2 \# As) ! j) ((s_0 \# \sigma_2 \# \sigma_s) ! j) \rangle$  for  $i \ j$ 
  using Cons.prems[of  $\langle if \ i = 0 \ then \ 0 \ else \ Suc \ i \rangle$   $\langle if \ j = 0 \ then \ 0 \ else \ Suc \ j \rangle$ ]
  by (cases  $i$ ; cases  $j$ ) simp-all
qed
from this[THEN indep-enablD, OF assms(1)  $\langle ts' \in \mathcal{R}_d \llbracket E \rrbracket A_2 \# As \rangle$  ( $\sigma_2$ 
 $\# \sigma_s$ )]
have  $\langle \varepsilon \ A_0 \ t_0 \cap \varepsilon \llbracket E \rrbracket A_2 \# As \rangle \ ts' \subseteq E$  .
moreover from Cons.prems[THEN indep-enablD, of  $0 \ \langle Suc \ 0 \rangle \ t_0 \ t_1$ ]
assms(2)[simplified, THEN set-mp[OF  $\mathcal{R}_d$ -combine $_d$ RList-Sync-subset]] assms(1)
have  $\langle \varepsilon \ A_0 \ t_0 \cap \varepsilon \ A_1 \ t_1 \subseteq E \rangle$  by (simp add:  $\langle ts = t_1 \# ts' \rangle$ )
ultimately show  $\langle \varepsilon \ A_0 \ t_0 \cap \varepsilon \llbracket E \rrbracket A_1 \# A_2 \# As \rangle \ ts \subseteq E$ 
by (auto simp add:  $\varepsilon$ -combine $_d$ RList-Sync  $\langle ts = t_1 \# ts' \rangle$  combine-Sync- $\varepsilon$ -def)
qed
qed

```

lemma ω -iterated-combine $_d$ -Sync :

```

 $\langle length \ \sigma_s = length \ As \implies$ 
   $\omega \llbracket E \rrbracket As \rangle \ \sigma_s = (case \ those \ (map2 \ \omega \ As \ \sigma_s) \ of \ \diamond \ \Rightarrow \ \diamond$ 
   $| \ [terms] \ \Rightarrow \ if \ card \ (set \ terms) = Suc \ 0 \ then \ [THE \ r. \ set \ terms = \{r\}] \ else$ 
   $\diamond) \rangle$ 
by (induct  $\sigma_s \ As$  rule: induct-2-lists012)
  (auto simp add:  $\sigma$ - $\sigma_s$ -conv-defs combine $_d$ RList-Sync-defs card-1-singleton-iff the-equality
split: option.split)

```

lemma ω -iterated-combine $_n$ $_d$ -Sync :

```

 $\langle length \ \sigma_s = length \ As \implies$ 
   $\omega \llbracket E \rrbracket As \rangle \ \sigma_s = (if \ As = [] \ then \ \{\} \ else \ \{r. \ \forall i < length \ As. \ r \in \omega \ (As \ !$ 
   $i) \ (\sigma_s \ ! \ i)\}) \rangle$ 
proof (induct  $\sigma_s \ As$  rule: induct-2-lists012)
  case Nil show ?case by simp
next
  case (single  $\sigma_1 \ A_1$ )
  from length-Suc-conv show ?case
    by (auto simp add:  $\sigma$ - $\sigma_s$ -conv-defs)
next
  case (Cons  $\sigma_1 \ \sigma_2 \ \sigma_s \ A_1 \ A_2 \ As$ )
  show ?case (is  $\langle - = ?rhs \ \sigma_1 \ \sigma_2 \ \sigma_s \ A_1 \ A_2 \ As \rangle$ )

```

proof (*intro subset-antisym subsetI*)
fix r **assume** $\langle r \in \omega \llbracket_{nd} \otimes [E] A1 \# A2 \# As \rrbracket (\sigma1 \# \sigma2 \# \sigma s) \rangle$
hence $\langle r \in \omega A1 \sigma1 \rangle \langle r \in \omega \llbracket_{nd} \otimes [E] A2 \# As \rrbracket (\sigma2 \# \sigma s) \rangle$
by (*simp-all add: combine_{RList}-Sync-defs split: if-split-asm*)
from *this(2)* **have** $\langle \forall i < Suc (length As). r \in \omega ((A2 \# As) ! i) ((\sigma2 \# \sigma s) ! i) \rangle$
by (*simp add: Cons.hyps(3)*)
with $\langle r \in \omega A1 \sigma1 \rangle$ **show** $\langle r \in ?rhs \sigma1 \sigma2 \sigma s A1 A2 As \rangle$
by (*auto simp add: less-Suc-eq-0-disj*)
next
from *Cons.hyps(3)*
show $\langle r \in ?rhs \sigma1 \sigma2 \sigma s A1 A2 As \implies$
 $r \in \omega \llbracket_{nd} \otimes [E] A1 \# A2 \# As \rrbracket (\sigma1 \# \sigma2 \# \sigma s) \rangle$ **for** r
by (*auto simp add: combine_{RList}-Sync-defs*)
qed
qed

6.1.5 Normalization

lemma ω -iterated-combine_{nd}-Sync-det-ndet-conv:
 $\langle length \sigma s = length As \implies$
 $\omega \llbracket_{nd} \otimes [E] map (\lambda A. \llbracket A \rrbracket_{d \mapsto nd}) As \rrbracket \sigma s = \omega \llbracket \llbracket_d \otimes [E] As \rrbracket \rrbracket_{d \mapsto nd} \sigma s \rangle$
proof (*induct σs As rule: induct-2-lists012*)
case *Nil*
show *?case* **by** (*simp add: base-trans-det-ndet-conv(1)*)
next
case (*single $\sigma1 A1$*)
show *?case* **by** (*simp add: from-det-to-ndet- σ - σs -conv-commute*)
next
case (*Cons $\sigma1 \sigma2 \sigma s A1 A2 As$*)
thus *?case*
by (*auto simp add: det-ndet-conv-defs combine_{RList}-Sync-defs split: option.split*)
qed

lemma τ -iterated-combine_{nd}-Sync-behaviour-when-indep:
 $\langle length \sigma s = length As \implies$
 $(\bigwedge i j. \llbracket i < length As; j < length As; i \neq j \rrbracket$
 $\implies indep-enabl (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j)) \implies$
 $\tau \llbracket \llbracket_d \otimes [E] As \rrbracket \rrbracket_{d \mapsto nd} \sigma s e = \tau \llbracket_{nd} \otimes [E] map (\lambda A. \llbracket A \rrbracket_{d \mapsto nd}) As \rrbracket \sigma s e \rangle$
proof (*induct σs As rule: induct-2-lists012*)
case *Nil*
show *?case* **by** *simp*
next
case (*single $\sigma1 A1$*)
show *?case* **by** (*simp add: from-det-to-ndet- σ - σs -conv-commute(1)*)
next
case (*Cons $\sigma1 \sigma2 \sigma s A1 A2 As$*)
have $*$: $\langle \tau \llbracket \llbracket_d \otimes [E] A2 \# As \rrbracket \rrbracket_{d \mapsto nd} (\sigma2 \# \sigma s) e =$
 $\tau \llbracket_{nd} \otimes [E] map (\lambda A. \llbracket A \rrbracket_{d \mapsto nd}) (A2 \# As) \rrbracket (\sigma2 \# \sigma s) e \rangle$

proof (*rule Cons.hyyps(3)*)
show $\langle [i < \text{length } (A2 \# As); j < \text{length } (A2 \# As); i \neq j]$
 $\implies \text{indep-enabl } ((A2 \# As) ! i) ((\sigma2 \# \sigma s) ! i) E$
 $((A2 \# As) ! j) ((\sigma2 \# \sigma s) ! j)\rangle$ **for** $i j$
using *Cons.premis[of ‹Suc i› ‹Suc j›]* **by** *simp*
qed
have $\langle \tau \lll \langle \langle d \otimes [E] A1 \# A2 \# As \rangle \rangle_{d \mapsto nd} (\sigma1 \# \sigma2 \# \sigma s) e =$
 $\tau \lll \langle \langle A1 \rangle \rangle_{d \mapsto nd} nd \otimes [E] Rlist \lll \langle \langle d \otimes [E] A2 \# As \rangle \rangle_{d \mapsto nd} (\sigma1 \# \sigma2 \# \sigma s)$
 $e \rangle$
proof (*subst iterated-combine_d-Sync.simps(3)*, *rule τ -combine_{Rlist}-Sync-behaviour-when-indep*)
show $\langle \varepsilon A1 \sigma1 \cap \varepsilon \lll \langle \langle d \otimes [E] A2 \# As \rangle \rangle_{d \mapsto nd} (\sigma2 \# \sigma s) \subseteq E \rangle$
proof (*rule indep-enablD[OF - \mathcal{R}_d .init \mathcal{R}_d .init]*)
show $\langle \text{indep-enabl } A1 \sigma1 E \lll \langle \langle d \otimes [E] A2 \# As \rangle \rangle_{d \mapsto nd} (\sigma2 \# \sigma s) \rangle$
by (*simp add: Cons.hyyps(1) Cons.premis order-le-less-trans*
same-length-indep-transmission-to-iterated-combine_d-Sync)
qed
qed
also have $\langle \dots = \tau \lll \langle \langle nd \otimes [E] \text{map } (\lambda A. \langle \langle A \rangle \rangle_{d \mapsto nd}) (A1 \# A2 \# As) \rangle \rangle_{d \mapsto nd} (\sigma1 \#$
 $\sigma2 \# \sigma s) e \rangle$
by (*use * in ‹simp add: combine_{Rlist}-Sync-defs ε -simps›*)
(metis empty-from-det-to-ndet-is-None-trans option.exhaust)
finally show *?case* .
qed

lemma *P_{SKIPS}-iterated-combine_{nd}-Sync-behaviour-when-indep*:
assumes *same-length: ‹length $\sigma s = \text{length } As \rangle$*
and *indep: ‹ $\bigwedge i j. [i < \text{length } As; j < \text{length } As; i \neq j] \implies$*
indep-enabl (As ! i) (σs ! i) E (As ! j) (σs ! j)›
shows $\langle P_{SKIPS} \lll \langle \langle d \otimes [E] As \rangle \rangle_d \sigma s = P_{SKIPS} \lll \langle \langle nd \otimes [E] \text{map } (\lambda A. \langle \langle A \rangle \rangle_{d \mapsto nd})$
 $As \rangle \rangle_{nd} \sigma s \rangle$
proof (*fold P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*, *rule P_{SKIPS}-nd-eqI-strong-id*)
show $\langle \sigma s' \in \mathcal{R}_{nd} \lll \langle \langle d \otimes [E] As \rangle \rangle_{d \mapsto nd} \sigma s \implies$
 $\tau \lll \langle \langle nd \otimes [E] \text{map } (\lambda A. \langle \langle A \rangle \rangle_{d \mapsto nd}) As \rangle \rangle \sigma s' e = \tau \lll \langle \langle d \otimes [E] As \rangle \rangle_{d \mapsto nd} \sigma s'$
 $e \rangle$ **for** $\sigma s' e$
proof (*rule τ -iterated-combine_{nd}-Sync-behaviour-when-indep[symmetric]*)
show $\langle \sigma s' \in \mathcal{R}_{nd} \lll \langle \langle d \otimes [E] As \rangle \rangle_{d \mapsto nd} \sigma s \implies \text{length } \sigma s' = \text{length } As \rangle$
by (*metis \mathcal{R}_{nd} -from-det-to-ndet same-length same-length- \mathcal{R}_d -iterated-combine_d-Sync-description*)
next
show $\langle [\sigma s' \in \mathcal{R}_{nd} \lll \langle \langle d \otimes [E] As \rangle \rangle_{d \mapsto nd} \sigma s; i < \text{length } As; j < \text{length } As; i \neq$
 $j]$
 $\implies \text{indep-enabl } (As ! i) (\sigma s' ! i) E (As ! j) (\sigma s' ! j)\rangle$ **for** $i j$
by (*subst (asm) \mathcal{R}_{nd} -from-det-to-ndet*,
drule same-length- \mathcal{R}_d -iterated-combine_d-Sync-description[OF same-length])
(meson \mathcal{R}_d -trans indep-enabl-def indep)
qed
next
show $\langle \sigma s' \in \mathcal{R}_{nd} \lll \langle \langle d \otimes [E] As \rangle \rangle_{d \mapsto nd} \sigma s \implies$

$\omega \llbracket_{nd} \otimes [E] \text{ map } (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) \text{ As} \rrbracket \sigma s' = \omega \llbracket_{d} \otimes [E] \text{ As} \rrbracket_{d \hookrightarrow nd} \sigma s'$
for $\sigma s'$
by (*metis* \mathcal{R}_{nd} -from-det-to-ndet ω -iterated-combine $_{nd}$ -Sync-det-ndet-conv same-length
same-length- \mathcal{R}_d -iterated-combine $_d$ -Sync-description)
qed

lemma *P-d-iterated-combine $_{nd}$ -Sync-behaviour-when-indep*:

assumes same-length: $\langle \text{length } \sigma s = \text{length } \text{As} \rangle$
and indep: $\langle \bigwedge i j. \llbracket i < \text{length } \text{As}; j < \text{length } \text{As}; i \neq j \rrbracket \implies$
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$
shows $\langle P \llbracket_{d} \otimes [E] \text{ As} \rrbracket_d \sigma s = P \llbracket_{nd} \otimes [E] \text{ map } (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) \text{ As} \rrbracket_{nd} \sigma s \rangle$
proof (*fold P-nd-from-det-to-ndet-is-P-d*, *rule P-nd-eqI-strong-id*)
show $\langle \sigma s' \in \mathcal{R}_{nd} \llbracket_{d} \otimes [E] \text{ As} \rrbracket_{d \hookrightarrow nd} \sigma s \implies$
 $\tau \llbracket_{nd} \otimes [E] \text{ map } (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) \text{ As} \rrbracket \sigma s' e = \tau \llbracket_{d} \otimes [E] \text{ As} \rrbracket_{d \hookrightarrow nd} \sigma s' e \rangle$
for $\sigma s' e$
proof (*rule* τ -iterated-combine $_{nd}$ -Sync-behaviour-when-indep[*symmetric*])
show $\langle \sigma s' \in \mathcal{R}_{nd} \llbracket_{d} \otimes [E] \text{ As} \rrbracket_{d \hookrightarrow nd} \sigma s \implies \text{length } \sigma s' = \text{length } \text{As} \rangle$
by (*metis* \mathcal{R}_{nd} -from-det-to-ndet same-length same-length- \mathcal{R}_d -iterated-combine $_d$ -Sync-description)
next
show $\langle \llbracket \sigma s' \in \mathcal{R}_{nd} \llbracket_{d} \otimes [E] \text{ As} \rrbracket_{d \hookrightarrow nd} \sigma s; i < \text{length } \text{As}; j < \text{length } \text{As}; i \neq j \rrbracket$
 $\implies \text{indep-enabl } (As ! i) (\sigma s' ! i) E (As ! j) (\sigma s' ! j) \rangle$ **for** $i j$
by (*subst* *asm*) \mathcal{R}_{nd} -from-det-to-ndet,
drule same-length- \mathcal{R}_d -iterated-combine $_d$ -Sync-description[*OF* same-length]
meson \mathcal{R}_d -trans indep-enabl-def indep)
qed
qed

6.2 Compactification Theorems

6.2.1 Binary

Pair

theorem *P_{SKIP_S}-nd-combine_{Pair}-Sync* :
fixes $E :: \langle 'a \text{ set} \rangle$ **and** $A_0 :: \langle ('\sigma, 'a, 'r, 'a) A_{nd}\text{-scheme} \rangle$
assumes ϱ -disjoint- ε : $\langle \varrho$ -disjoint- $\varepsilon A_0 \rangle \langle \varrho$ -disjoint- $\varepsilon A_1 \rangle$
and at-most-1-elem-term : $\langle \text{at-most-1-elem-term } A_0 \rangle \langle \text{at-most-1-elem-term } A_1 \rangle$
defines *A-def*: $\langle A \equiv \llbracket_{nd} \otimes [E] \text{ Pair } A_1 \rrbracket \rangle$
defines *P-def*: $\langle P \equiv P_{SKIP_S} \llbracket_{nd} \otimes [E] \text{ Pair } A_1 \rrbracket \rangle$ **and** *Q-def*: $\langle Q \equiv P_{SKIP_S} \llbracket_{nd} \otimes [E] \text{ Pair } A_1 \rrbracket \rangle$ **and**
S-def: $\langle S \equiv P_{SKIP_S} \llbracket_{nd} \otimes [E] \text{ Pair } A_1 \rrbracket \rangle$
shows $\langle P \sigma_0 [E] Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$
proof –
let $?f = \langle P_{SKIP_S}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A) (\lambda \sigma'. \text{case } \sigma' \text{ of } (\sigma_0, \sigma_1) \Rightarrow P \sigma_0$
 $[E] Q \sigma_1) \rangle$
note *cartprod-rwrt* = *GlobalNdet-cartprod*[*of* - - $\langle \lambda x y. - (x, y) \rangle$, *simplified*]
note *Ndet-and-Sync* = *Sync-distrib-GlobalNdet-left*
Sync-distrib-GlobalNdet-right

note *Mprefix-Sync-constant* =
SKIP-Sync-Mprefix Mprefix-Sync-SKIP
STOP-Sync-Mprefix Mprefix-Sync-STOP

note *P-rec* = *restriction-fix-eq[OF P_{SKIP}S-nd-step-constructive-bis[of A₀], folded P_{SKIP}S-nd-def P-def, THEN fun-cong]*

note *Q-rec* = *restriction-fix-eq[OF P_{SKIP}S-nd-step-constructive-bis[of A₁], folded P_{SKIP}S-nd-def Q-def, THEN fun-cong]*

have $\omega\text{-}A : \langle \omega A (\sigma_0', \sigma_1') = \omega A_0 \sigma_0' \cap \omega A_1 \sigma_1' \rangle$ **for** $\sigma_0' \sigma_1'$

by (*auto simp add: A-def combine_{P_{air}}-Sync-defs*)

have $\varepsilon\text{-}A : \langle \varepsilon A (\sigma_0', \sigma_1') = \text{combine-sets-Sync} (\varepsilon A_0 \sigma_0') E (\varepsilon A_1 \sigma_1') \rangle$ **for** $\sigma_0' \sigma_1'$

by (*simp add: A-def ε -combine_{P_{air}}-Sync combine-Sync- ε -def*)

have *Mprefix-Sync-Mprefix-for-procomata* :

$\langle \Box a \in A \rightarrow P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b =$
 $(\Box a \in (A - E - B) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b)) \quad \Box$
 $(\Box b \in (B - E - A) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b)) \quad \Box$
 $(\Box x \in (A \cap B - E) \rightarrow (P x \llbracket E \rrbracket \Box b \in B \rightarrow Q b) \cap (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q x)) \Box$
 $(\Box x \in (A \cap B \cap E) \rightarrow (P x \llbracket E \rrbracket Q x)) \rangle$ (**is ?eq**) **for** *A B and P Q* :: $\langle 'a \Rightarrow ('a,$
 $'r) \text{ process}_{ptick} \rangle$

proof –

have * : $\langle \Box a \in (A - E) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b) =$
 $(\Box a \in (A - E - B) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b)) \Box$
 $(\Box a \in (A \cap B - E) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b)) \rangle$

by (*metis Diff-Int2 Diff-Int-distrib2 Mprefix-Un-distrib Un-Diff-Int*)

have ** : $\langle \Box b \in (B - E) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b) =$
 $(\Box b \in (B - E - A) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b)) \Box$
 $(\Box b \in (A \cap B - E) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b)) \rangle$

by (*metis (no-types) Int-Diff Int-commute Mprefix-Un-distrib Un-Diff-Int*)

have $\langle \Box a \in A \rightarrow P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b =$
 $(\Box a \in (A - E - B) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b)) \quad \Box$
 $(\Box b \in (B - E - A) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b)) \quad \Box$
 $((\Box a \in (A \cap B - E) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b)) \quad \Box$
 $(\Box b \in (A \cap B - E) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b))) \quad \Box$
 $(\Box x \in (A \cap B \cap E) \rightarrow (P x \llbracket E \rrbracket Q x)) \rangle$

unfolding *Mprefix-Sync-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 $\langle (\Box a \in (A \cap B - E) \rightarrow (P a \llbracket E \rrbracket \Box b \in B \rightarrow Q b)) \Box$
 $(\Box b \in (A \cap B - E) \rightarrow (\Box a \in A \rightarrow P a \llbracket E \rrbracket Q b)) =$
 $\Box x \in (A \cap B - E) \rightarrow ((P x \llbracket E \rrbracket \Box b \in B \rightarrow Q b) \cap (\Box a \in A \rightarrow P a \llbracket E \rrbracket$
 $Q x)) \rangle$

by (*simp add: Mprefix-Det-Mprefix, rule mono-Mprefix-eq, simp*)

finally show ?thesis .

qed

show $\langle P \sigma_0 \llbracket E \rrbracket Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$

proof (*rule fun-cong[of $\langle \lambda (\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket Q \sigma_1 \rangle - \langle (\sigma_0, \sigma_1) \rangle$, simplified]*)

show $\langle \lambda (\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket Q \sigma_1 = S \rangle$

proof (*rule restriction-fix-unique[OF P_{SKIP}S-nd-step-constructive-bis[of A],*

symmetric, folded P_{SKIPS}-nd-def S-def)

show $\langle ?f = (\lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket Q \sigma_1) \rangle$

proof (*rule ext, clarify*)

have ϱ -disjoint- ε -bis : $\langle \omega A_0 \sigma_0 \neq \{\} \implies \varepsilon A_0 \sigma_0 = \{\} \rangle$
 $\langle \omega A_1 \sigma_1 \neq \{\} \implies \varepsilon A_1 \sigma_1 = \{\} \rangle$ **for** $\sigma_0 \sigma_1$

by (*simp-all add: ϱ -simps ϱ -disjoint- εD ϱ -disjoint- ε*)

show $\langle ?f (\sigma_0, \sigma_1) = P \sigma_0 \llbracket E \rrbracket Q \sigma_1 \rangle$ **for** $\sigma_0 \sigma_1$

proof (*cases $\langle \omega A_0 \sigma_0 = \{\} \rangle$; cases $\langle \omega A_1 \sigma_1 = \{\} \rangle$*)

assume $\langle \omega A_0 \sigma_0 = \{\} \rangle \langle \omega A_1 \sigma_1 = \{\} \rangle$

hence P -rec' : $\langle P \sigma_0 = P$ -nd-step (εA_0) (τA_0) $P \sigma_0 \rangle$

and Q -rec' : $\langle Q \sigma_1 = P$ -nd-step (εA_1) (τA_1) $Q \sigma_1 \rangle$

and S -rec' : $\langle P_{SKIPS}$ -nd-step (εA) (τA) (ωA) ($\lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket Q \sigma_1$) (σ_0, σ_1) =

P -nd-step (εA) (τA) ($\lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket Q \sigma_1$) (σ_0, σ_1) \rangle

by (*simp-all add: P -rec[of σ_0] Q -rec[of σ_1] ω -A*)

show $\langle ?f (\sigma_0, \sigma_1) = P \sigma_0 \llbracket E \rrbracket Q \sigma_1 \rangle$

unfolding P -rec' Q -rec' S -rec' *Mprefix-Sync-Mprefix-for-procomata*

unfolding ε -A *Mprefix-Un-distrib*

by (*intro arg-cong2[where $f = \langle (\square) \rangle$] mono-Mprefix-eq, fold P -rec' Q -rec', auto simp add: A-def Ndet-and-Sync cartprod-rwrt combine_{P_{air}}-Sync-defs ε -simps GlobalNdet-sets-commute[of $\langle \tau A_1 - \rightarrow \rangle$]*)

simp flip: GlobalNdet-factorization-union

intro!: mono-GlobalNdet-eq arg-cong2[where $f = \langle (\square) \rangle$])

next

assume $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \langle \omega A_1 \sigma_1 = \{\} \rangle$

from ϱ -disjoint- $\varepsilon(1)$ $\langle \omega A_0 \sigma_0 \neq \{\} \rangle$ **have** $\langle \varepsilon A_0 \sigma_0 = \{\} \rangle$

by (*simp add: ϱ -disjoint- ε -def ϱ -simps*)

have $\langle ?f (\sigma_0, \sigma_1) = \square b \in (\varepsilon A_1 \sigma_1 - E) \rightarrow (\square \sigma_1' \in \tau A_1 \sigma_1 b. (SKIPS (\omega A_0 \sigma_0) \llbracket E \rrbracket Q \sigma_1')) \rangle$

by (*auto simp add: ω -A ε -A $\langle \omega A_1 \sigma_1 = \{\} \rangle \langle \varepsilon A_0 \sigma_0 = \{\} \rangle$ intro!: mono-Mprefix-eq,*

auto simp add: A-def combine_{P_{air}}-Sync-defs $\langle \omega A_0 \sigma_0 \neq \{\} \rangle$

$\langle \varepsilon A_0 \sigma_0 = \{\} \rangle$ *cartprod-rwrt P -rec[of σ_0] intro!: mono-GlobalNdet-eq*)

also have $\langle \dots = P \sigma_0 \llbracket E \rrbracket Q \sigma_1 \rangle$

by (*unfold P -rec[of σ_0] Q -rec[of σ_1]*)

(auto simp add: SKIPS-def Ndet-and-Sync Mprefix-Sync-constant $\langle \omega A_0 \sigma_0 \neq \{\} \rangle$

GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[of $\langle \tau A_1 - \rightarrow \rangle$] $\langle \omega A_1 \sigma_1 = \{\} \rangle$

intro!: mono-Mprefix-eq mono-GlobalNdet-eq)

finally show $\langle ?f (\sigma_0, \sigma_1) = \dots \rangle$.

next

assume $\langle \omega A_0 \sigma_0 = \{\} \rangle \langle \omega A_1 \sigma_1 \neq \{\} \rangle$

from ϱ -disjoint- $\varepsilon(2)$ $\langle \omega A_1 \sigma_1 \neq \{\} \rangle$ **have** $\langle \varepsilon A_1 \sigma_1 = \{\} \rangle$

by (*simp add: ϱ -disjoint- ε -def ϱ -simps*)

have $\langle ?f (\sigma_0, \sigma_1) = \square a \in (\varepsilon A_0 \sigma_0 - E) \rightarrow (\square \sigma_0' \in \tau A_0 \sigma_0 a. (P \sigma_0' \llbracket E \rrbracket SKIPS (\omega A_1 \sigma_1))) \rangle$

by (*auto simp add: ω -A ε -A $\langle \omega A_0 \sigma_0 = \{\} \rangle \langle \varepsilon A_1 \sigma_1 = \{\} \rangle$ intro!:*

mono-Mprefix-eq,
auto simp add: A-def combine_{Pair}-Sync-defs $\langle \omega A_1 \sigma_1 \neq \{\} \rangle$
 $\langle \varepsilon A_1 \sigma_1 = \{\} \rangle$ *cartprod-rwrt Q-rec[of σ_1] intro!: mono-GlobalNdet-eq*
also have $\langle \dots = P \sigma_0 \llbracket E \rrbracket Q \sigma_1 \rangle$
by (*unfold P-rec[of σ_0] Q-rec[of σ_1]*)
(auto simp add: SKIPS-def Ndet-and-Sync Mprefix-Sync-constant $\langle \omega$
 $A_1 \sigma_1 \neq \{\} \rangle$
*GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[of $\langle \tau A_0 - \rangle$] $\langle \omega$
 $A_0 \sigma_0 = \{\} \rangle$
intro!: mono-Mprefix-eq mono-GlobalNdet-eq)
finally show $\langle ?f (\sigma_0, \sigma_1) = \dots \rangle$.
next
assume $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \langle \omega A_1 \sigma_1 \neq \{\} \rangle$
with ϱ -*disjoint- ε* **have** $\langle \varepsilon A_0 \sigma_0 = \{\} \rangle \langle \varepsilon A_1 \sigma_1 = \{\} \rangle$
by (*simp-all add: ϱ -disjoint- ε -def ϱ -simps*)
from *at-most-1-elem-term*
show $\langle \omega A_0 \sigma_0 \neq \{\} \implies \omega A_1 \sigma_1 \neq \{\} \implies ?f (\sigma_0, \sigma_1) = P \sigma_0 \llbracket E \rrbracket Q \sigma_1 \rangle$
by (*auto simp add: $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \langle \omega A_1 \sigma_1 \neq \{\} \rangle \omega$ -A P-rec[of σ_0]
Q-rec[of σ_1]
SKIPS-def Ndet-and-Sync cartprod-rwrt GlobalNdet-sets-commute[of
 $\langle \omega A_0 - \rangle$
 ε -A $\langle \varepsilon A_0 \sigma_0 = \{\} \rangle \langle \varepsilon A_1 \sigma_1 = \{\} \rangle$ elim!: *at-most-1-elem-termE*)*
qed
qed
qed
qed
qed*

corollary *P-nd-combine_{Pair}-Sync* :

$\langle P \langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P \langle A_1 \rangle_{nd} \sigma_1 = P \langle \langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1) \rangle$
proof –
have $\langle P \langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P \langle A_1 \rangle_{nd} \sigma_1 =$
 $P_{SKIPS} \langle A_0 (\omega := \lambda \sigma. \{\}) \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS} \langle A_1 (\omega := \lambda \sigma. \{\}) \rangle_{nd} \sigma_1 \rangle$
by (*simp add: P_{SKIPS}-nd-updated- ω*)
also have $\langle \dots = P_{SKIPS} \langle \langle A_0 (\omega := \lambda \sigma. \{\}) \text{ nd} \otimes \llbracket E \rrbracket_{Pair} A_1 (\omega := \lambda \sigma. \{\}) \rangle \rangle_{nd}$
 $(\sigma_0, \sigma_1) \rangle$
by (*rule P_{SKIPS}-nd-combine_{Pair}-Sync*) *simp-all*
also have $\langle \langle A_0 (\omega := \lambda \sigma. \{\}) \text{ nd} \otimes \llbracket E \rrbracket_{Pair} A_1 (\omega := \lambda \sigma. \{\}) \rangle = \langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{Pair}$
 $A_1 \rangle (\omega := \lambda \sigma. \{\}) \rangle$
by (*simp add: combine_{Pair}-Sync-defs, intro ext, simp*)
also have $\langle P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle (\omega := \lambda \sigma. \{\}) \rangle_{nd} = P \langle \langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{Pair}$
 $A_1 \rangle \rangle_{nd} \rangle$
by (*simp add: P_{SKIPS}-nd-updated- ω*)
finally show *?thesis* .
qed

corollary *P_{SKIPS}-d-combine_{Pair}-Sync*:

$\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{ d} \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle \rangle_d (\sigma_0,$
 $\sigma_1) \rangle$

if $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$ **and** $\langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$ **and** $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \rangle_{d \rightarrow nd}\rangle_{nd} \sigma_0$
 $\llbracket E \rrbracket P_{SKIPS}\langle\langle A_1 \rangle_{d \rightarrow nd}\rangle_{nd} \sigma_1 \rangle$
by (*simp flip: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*)
also from that(1, 2) **have** $\langle \dots = P_{SKIPS}\langle\langle\langle A_0 \rangle_{d \rightarrow nd} nd \otimes \llbracket E \rrbracket_{Pair} \langle A_1 \rangle_{d \rightarrow nd}\rangle\rangle_{nd}$
 $(\sigma_0, \sigma_1) \rangle$
by (*auto intro: P_{SKIPS}-nd-combine_{Pair}-Sync*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0 d \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle\rangle_d (\sigma_0, \sigma_1) \rangle$
by (*simp add: P_{SKIPS}-combine_{Pair}-Sync-behaviour-when-indep* $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$)
finally show *?thesis* .
qed

corollary *P-d-combine_{Pair}-Sync*:

$\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 d \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle\rangle_d (\sigma_0, \sigma_1) \rangle$
if $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_d \sigma_1 =$
 $P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_1 \rangle$
by (*simp add: P_{SKIPS}-d-updated- ω*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \diamond) d \otimes \llbracket E \rrbracket_{Pair} A_1(\omega := \lambda\sigma. \diamond) \rangle\rangle_d (\sigma_0,$
 $\sigma_1) \rangle$
by (*subst P_{SKIPS}-d-combine_{Pair}-Sync, simp-all add: ϱ -simps ϱ -disjoint- ε -def*)
(rule indep-enablI,
use $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$ *[THEN indep-enablD] in* $\langle \text{simp add: } \varepsilon\text{-simps} \rangle$)
also have $\langle \langle A_0(\omega := \lambda\sigma. \diamond) d \otimes \llbracket E \rrbracket_{Pair} A_1(\omega := \lambda\sigma. \diamond) \rangle = \langle A_0 d \otimes \llbracket E \rrbracket_{Pair}$
 $A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle$
by (*simp add: combine_{Pair}-Sync-defs, intro ext, simp*)
also have $\langle P_{SKIPS}\langle\langle A_0 d \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle_d = P\langle\langle A_0 d \otimes \llbracket E \rrbracket_{Pair}$
 $A_1 \rangle\rangle_d \rangle$
by (*simp add: P_{SKIPS}-d-updated- ω*)
finally show *?thesis* .
qed

Pairlist

theorem *P_{SKIPS}-nd-combine_{Pairlist}-Sync* :

$\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS}\langle\langle A_0 nd \otimes \llbracket E \rrbracket_{Pairlist}$
 $A_1 \rangle\rangle_{nd} [\sigma_0, \sigma_1] \rangle$
if $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$ $\langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$ $\langle \text{at-most-1-elem-term } A_0 \rangle$ $\langle \text{at-most-1-elem-term}$
 $A_1 \rangle$
proof –
from *P_{SKIPS}-nd-combine_{Pair}-Sync that*
have $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS}\langle\langle A_0 nd \otimes \llbracket E \rrbracket_{Pair}$
 $A_1 \rangle\rangle_{nd} (\sigma_0, \sigma_1) \rangle$.
also have $\langle \dots = P_{SKIPS}\langle\langle A_0 nd \otimes \llbracket E \rrbracket_{Pairlist} A_1 \rangle\rangle_{nd} [\sigma_0, \sigma_1] \rangle$
by (*auto intro!: P_{SKIPS}-nd-eqI-strong[of* $\langle \lambda(r, s). [r, s] \rangle$ *-* $\langle (\sigma_0, \sigma_1) \rangle$, *simplified*)
inj-onI)

(*auto simp add: combine-Sync-defs split: if-split-asm*)
finally show ?thesis .
qed

corollary *P-nd-combine_{Pairlist}-Sync* :
 $\langle P\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_{nd} \sigma_1 = P\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle_{nd} [\sigma_0, \sigma_1] \rangle$
proof –
have $\langle P\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_{nd} \sigma_1 =$
 $P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \{\}) \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \{\}) \rangle_{nd} \sigma_1 \rangle$
by (*simp only: P_{SKIPS}-nd-updated- ω*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } A_1(\omega := \lambda\sigma. \{\}) \rangle\rangle_{nd} [\sigma_0, \sigma_1] \rangle$
by (*rule P_{SKIPS}-nd-combine_{Pairlist}-Sync*) *simp-all*
also have $\langle\langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } A_1(\omega := \lambda\sigma. \{\}) \rangle\rangle = \langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle(\omega := \lambda\sigma. \{\}) \rangle$
by (*simp add: combine_{Pairlist}-Sync-defs, intro ext, simp*)
also have $\langle P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle(\omega := \lambda\sigma. \{\}) \rangle_{nd} = P\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle_{nd} \rangle$
by (*simp only: P_{SKIPS}-nd-updated- ω*)
finally show ?thesis .
qed

corollary *P_{SKIPS}-d-combine_{Pairlist}-Sync* :
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$
if $\langle \rho\text{-disjoint-}\varepsilon A_0 \rangle$ **and** $\langle \rho\text{-disjoint-}\varepsilon A_1 \rangle$ **and** $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \rightarrow \text{nd} \rangle\rangle_{nd} \sigma_0$
 $\llbracket E \rrbracket P_{SKIPS}\langle\langle A_1 \text{ d} \rightarrow \text{nd} \rangle\rangle_{nd} \sigma_1 \rangle$
by (*simp flip: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*)
also from that (1, 2) **have** $\langle \dots = P_{SKIPS}\langle\langle A_0 \text{ d} \rightarrow \text{nd} \text{ nd} \otimes \llbracket E \rrbracket \text{Pairlist } \langle A_1 \text{ d} \rightarrow \text{nd} \rangle\rangle_{nd} [\sigma_0, \sigma_1] \rangle$
by (*auto intro: P_{SKIPS}-nd-combine_{Pairlist}-Sync*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$
by (*simp add: P_{SKIPS}-combine_{Pairlist}-Sync-behaviour-when-indep* $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$)
finally show ?thesis .
qed

corollary *P-d-combine_{Pairlist}-Sync* :
 $\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \text{ d} \otimes \llbracket E \rrbracket \text{Pairlist } A_1 \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$
if $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_d \sigma_1 =$
 $P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_1 \rangle$
by (*simp only: P_{SKIPS}-d-updated- ω*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \diamond) \text{ d} \otimes \llbracket E \rrbracket \text{Pairlist } A_1(\omega := \lambda\sigma. \diamond) \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$
by (*subst P_{SKIPS}-d-combine_{Pairlist}-Sync, simp-all*)

(rule indep-enablI,
 use $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle [\text{THEN indep-enablD}]$ in simp)
 also have $\langle \langle A_0(\omega := \lambda\sigma. \diamond) \text{ d} \otimes [E]_{\text{Pairlist}} A_1(\omega := \lambda\sigma. \diamond) \rangle = \langle A_0 \text{ d} \otimes [E]_{\text{Pairlist}} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle$
 by (simp add: combine_{Pairlist}-Sync-defs, intro ext, simp)
 also have $\langle P_{\text{SKIPS}} \langle \langle A_0 \text{ d} \otimes [E]_{\text{Pairlist}} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle_d = P \langle \langle A_0 \text{ d} \otimes [E]_{\text{Pairlist}} A_1 \rangle \rangle_d \rangle$
 by (simp only: P_{SKIPS}-d-updated- ω)
 finally show ?thesis .
 qed

6.2.2 Rlist

theorem $P_{\text{SKIPS-nd-combineRlist-Sync}}$:
 $\langle P_{\text{SKIPS}} \langle A_0 \rangle_{\text{nd}} \sigma_0 [E] P_{\text{SKIPS}} \langle A_1 \rangle_{\text{nd}} \sigma_1 = P_{\text{SKIPS}} \langle \langle A_0 \text{ nd} \otimes [E]_{\text{Rlist}} A_1 \rangle \rangle_{\text{nd}} (\sigma_0 \# \sigma_1) \rangle$
 if $\langle \rho\text{-disjoint-}\varepsilon A_0 \rangle \langle \rho\text{-disjoint-}\varepsilon A_1 \rangle \langle \text{at-most-1-elem-term } A_0 \rangle \langle \text{at-most-1-elem-term } A_1 \rangle$
proof –
 from $P_{\text{SKIPS-nd-combinePair-Sync}}$ that
 have $\langle P_{\text{SKIPS}} \langle A_0 \rangle_{\text{nd}} \sigma_0 [E] P_{\text{SKIPS}} \langle A_1 \rangle_{\text{nd}} \sigma_1 = P_{\text{SKIPS}} \langle \langle A_0 \text{ nd} \otimes [E]_{\text{Pair}} A_1 \rangle \rangle_{\text{nd}} (\sigma_0, \sigma_1) \rangle$.
 also have $\langle \dots = P_{\text{SKIPS}} \langle \langle A_0 \text{ nd} \otimes [E]_{\text{Rlist}} A_1 \rangle \rangle_{\text{nd}} (\sigma_0 \# \sigma_1) \rangle$
 by (auto intro!: P_{SKIPS}-nd-eqI-strong[*of* $\langle \lambda(r, s). r \# s \rangle$ - $\langle (\sigma_0, \sigma_1) \rangle$, simplified] inj-onI)
 (auto simp add: combine-Sync-defs split: if-split-asm)
 finally show ?thesis .
 qed

corollary $P\text{-nd-combineRlist-Sync}$:
 $\langle P \langle A_0 \rangle_{\text{nd}} \sigma_0 [E] P \langle A_1 \rangle_{\text{nd}} \sigma_1 = P \langle \langle A_0 \text{ nd} \otimes [E]_{\text{Rlist}} A_1 \rangle \rangle_{\text{nd}} (\sigma_0 \# \sigma_1) \rangle$
proof –
 have $\langle P \langle A_0 \rangle_{\text{nd}} \sigma_0 [E] P \langle A_1 \rangle_{\text{nd}} \sigma_1 = P_{\text{SKIPS}} \langle A_0(\omega := \lambda\sigma. \{\}) \rangle_{\text{nd}} \sigma_0 [E] P_{\text{SKIPS}} \langle A_1(\omega := \lambda\sigma. \{\}) \rangle_{\text{nd}} \sigma_1 \rangle$
 by (simp only: P_{SKIPS}-nd-updated- ω)
 also have $\langle \dots = P_{\text{SKIPS}} \langle \langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes [E]_{\text{Rlist}} A_1(\omega := \lambda\sigma. \{\}) \rangle \rangle_{\text{nd}} (\sigma_0 \# \sigma_1) \rangle$
 by (rule P_{SKIPS-nd-combineRlist-Sync}) simp-all
 also have $\langle \langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes [E]_{\text{Rlist}} A_1(\omega := \lambda\sigma. \{\}) \rangle = \langle A_0 \text{ nd} \otimes [E]_{\text{Rlist}} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle$
 by (simp add: combine_{Rlist}-Sync-defs, intro ext, simp)
 also have $\langle P_{\text{SKIPS}} \langle \langle A_0 \text{ nd} \otimes [E]_{\text{Rlist}} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle_{\text{nd}} = P \langle \langle A_0 \text{ nd} \otimes [E]_{\text{Rlist}} A_1 \rangle \rangle_{\text{nd}} \rangle$
 by (simp only: P_{SKIPS}-nd-updated- ω)
 finally show ?thesis .
 qed

corollary $P_{\text{SKIPS-d-combineRlist-Sync}}$:
 $\langle P_{\text{SKIPS}} \langle A_0 \rangle_d \sigma_0 [E] P_{\text{SKIPS}} \langle A_1 \rangle_d \sigma_1 = P_{\text{SKIPS}} \langle \langle A_0 \text{ d} \otimes [E]_{\text{Rlist}} A_1 \rangle \rangle_d (\sigma_0$

$\# \sigma_1 \rangle$
if $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$ **and** $\langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$ **and** $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle A_0 \rangle_{d \rightarrow nd} \rangle_{nd} \sigma_0$
 $\llbracket E \rrbracket P_{SKIPS} \langle \langle A_1 \rangle_{d \rightarrow nd} \rangle_{nd} \sigma_1 \rangle$
by (*simp flip: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*)
also from that (1, 2) **have** $\langle \dots = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \rightarrow nd} \text{nd} \otimes \llbracket E \rrbracket_{Rlist} \langle A_1 \rangle_{d \rightarrow nd} \rangle \rangle_{nd}$
 $(\sigma_0 \# \sigma_1) \rangle$
by (*auto intro: P_{SKIPS}-nd-combine_{Rlist}-Sync*)
also have $\langle \dots = P_{SKIPS} \langle \langle A_0 \text{d} \otimes \llbracket E \rrbracket_{Rlist} A_1 \rangle \rangle_d (\sigma_0 \# \sigma_1) \rangle$
by (*simp add: P_{SKIPS}-combine_{Rlist}-Sync-behaviour-when-indep* $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$)
finally show *?thesis* .
qed

corollary *P-d-combine_{Rlist}-Sync* :
 $\langle P \langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P \langle A_1 \rangle_d \sigma_1 = P \langle \langle A_0 \text{d} \otimes \llbracket E \rrbracket_{Rlist} A_1 \rangle \rangle_d (\sigma_0 \# \sigma_1) \rangle$
if $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P \langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P \langle A_1 \rangle_d \sigma_1 =$
 $P_{SKIPS} \langle A_0 (\omega := \lambda \sigma. \diamond) \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS} \langle A_1 (\omega := \lambda \sigma. \diamond) \rangle_d \sigma_1 \rangle$
by (*simp only: P_{SKIPS}-d-updated- ω*)
also have $\langle \dots = P_{SKIPS} \langle \langle A_0 (\omega := \lambda \sigma. \diamond) \text{d} \otimes \llbracket E \rrbracket_{Rlist} A_1 (\omega := \lambda \sigma. \diamond) \rangle \rangle_d (\sigma_0$
 $\# \sigma_1) \rangle$
by (*subst P_{SKIPS}-d-combine_{Rlist}-Sync, simp-all*)
(rule indep-enabl,
use $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$ [THEN indep-enablD] in simp)
also have $\langle \langle A_0 (\omega := \lambda \sigma. \diamond) \text{d} \otimes \llbracket E \rrbracket_{Rlist} A_1 (\omega := \lambda \sigma. \diamond) \rangle = \langle A_0 \text{d} \otimes \llbracket E \rrbracket_{Rlist}$
 $A_1 \rangle (\omega := \lambda \sigma. \diamond) \rangle$
by (*simp add: combine_{Rlist}-Sync-defs, intro ext, simp*)
also have $\langle P_{SKIPS} \langle \langle A_0 \text{d} \otimes \llbracket E \rrbracket_{Rlist} A_1 \rangle (\omega := \lambda \sigma. \diamond) \rangle_d = P \langle \langle A_0 \text{d} \otimes \llbracket E \rrbracket_{Rlist}$
 $A_1 \rangle \rangle_d \rangle$
by (*simp only: P_{SKIPS}-d-updated- ω*)
finally show *?thesis* .
qed

6.2.3 ListslenL

theorem *P_{SKIPS}-nd-combine_{ListslenL}-Sync* :
assumes *same-length-reach0* : $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$
and *$\varrho\text{-disjoint-}\varepsilon$* : $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle \langle \text{at-most-1-elem-term } A_0 \rangle$
 $\langle \text{at-most-1-elem-term } A_1 \rangle$
shows $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS} \langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{nd} \otimes \llbracket \text{len}_0, E \rrbracket_{ListslenL} A_1 \rangle \rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
proof –
from *set-mp[OF $\mathcal{R}_{nd}\text{-combine}_{ndPair}\text{-Sync-subset}$]*
have $*$: $\langle \sigma' \in \mathcal{R}_{nd} \langle A_0 \text{nd} \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle (\sigma_0, \sigma_1) \implies$
 $\exists \sigma_0' \sigma_1'. \sigma' = (\sigma_0', \sigma_1') \wedge \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \wedge \sigma_1' \in \mathcal{R}_{nd} A_1 \sigma_1 \rangle$ **for** σ'
by *fast*

from $P_{SKIPS}\text{-nd-combine}_{Pair}\text{-Sync assms}(2-5)$
have $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{Pair} A_1 \rangle\rangle_{nd} (\sigma_0, \sigma_1) \rangle$.
also have $\langle \dots = P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle\rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
by (*auto intro!*: $P_{SKIPS}\text{-nd-eqI-strong}$ [*of* $\langle \lambda(r, s). r @ s \rangle$ - $\langle (\sigma_0, \sigma_1) \rangle$, *simplified inj-onI*
dest!: * *same-length-reach0 simp add: same-length-reach0 image-iff*)
(auto simp add: combine-Sync-defs ε -simps split: if-split-asm,
(metis ΣI $UnCI$ case-prod-conv insertI1)+)
finally show *?thesis* .
qed

corollary $P\text{-nd-combine}_{ListslenL}\text{-Sync}$:
 $\langle P\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_{nd} \sigma_1 = P\langle\langle A_0 \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle\rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
if $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \text{length } \sigma_0' = len_0 \rangle$
proof –
have $\langle P\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \{\}) \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \{\}) \rangle_{nd} \sigma_1 \rangle$
by (*simp only: $P_{SKIPS}\text{-nd-updated-}\omega$*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1(\omega := \lambda\sigma. \{\}) \rangle\rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
by (*rule $P_{SKIPS}\text{-nd-combine}_{ListslenL}\text{-Sync}$ (simp-all add: that)*)
also have $\langle \langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1(\omega := \lambda\sigma. \{\}) \rangle = \langle A_0 \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle$
by (*simp add: combine $_{ListslenL}\text{-Sync-defs}$, intro ext, simp*)
also have $\langle P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle_{nd} = P\langle\langle A_0 \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle\rangle_{nd} \rangle$
by (*simp only: $P_{SKIPS}\text{-nd-updated-}\omega$*)
finally show *?thesis* .
qed

corollary $P_{SKIPS}\text{-d-combine}_{ListslenL}\text{-Sync}$:
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle\rangle_d (\sigma_0 @ \sigma_1) \rangle$
if $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies \text{length } \sigma_0' = len_0 \rangle$
 $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle \langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
have $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle\langle A_1 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_1 \rangle$
by (*simp flip: $P_{SKIPS}\text{-nd-from-det-to-ndet-is-}\mathcal{P}_{SKIPS}\text{-d}$*)
also from *that(1-3)* **have** $\langle \dots = P_{SKIPS}\langle\langle\langle A_0 \rangle_{d \hookrightarrow nd} \text{ nd} \otimes \llbracket len_0, E \rrbracket_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle\rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
by (*auto simp add: $\mathcal{R}_{nd}\text{-from-det-to-ndet intro!:$ $P_{SKIPS}\text{-nd-combine}_{ListslenL}\text{-Sync}$*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket len_0, E \rrbracket_{ListslenL} A_1 \rangle\rangle_d (\sigma_0 @ \sigma_1) \rangle$
by (*simp add: $P_{SKIPS}\text{-combine}_{ListslenL}\text{-Sync-behaviour-when-indep that(1, 4)$*)
finally show *?thesis* .
qed

corollary P - d -combine $_{ListslenL}$ -Sync :

$\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} A_1 \rangle_d (\sigma_0 \textcircled{+} \sigma_1) \rangle$
if $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies length \sigma_0' = len_0 \rangle \langle indep-enabl A_0 \sigma_0 E A_1 \sigma_1 \rangle$

proof –

have $\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket P\langle A_1 \rangle_d \sigma_1 =$
 $P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_1 \rangle$
by (*simp only: P_{SKIPS}-d-updated- ω*)
also have $\langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \diamond) \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} A_1(\omega := \lambda\sigma. \diamond) \rangle_d (\sigma_0 \textcircled{+} \sigma_1) \rangle$
by (*subst P_{SKIPS}-d-combine_{ListslenL}-Sync*) (*simp-all add: that*)
also have $\langle\langle A_0(\omega := \lambda\sigma. \diamond) \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} A_1(\omega := \lambda\sigma. \diamond) \rangle = \langle A_0 \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} \langle A_1 \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} (\omega := \lambda\sigma. \diamond) \rangle$
by (*simp add: combine_{ListslenL}-Sync-defs, intro ext, simp*)
also have $\langle P_{SKIPS}\langle\langle A_0 \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle_d = P\langle\langle A_0 \rangle_{d \otimes \llbracket len_0, E \rrbracket ListslenL} A_1 \rangle_d \rangle$
by (*simp only: P_{SKIPS}-d-updated- ω*)
finally show *?thesis* .
qed

6.2.4 Multiple

theorem P_{SKIPS} - nd -compactification-Sync:

$\langle \llbracket length \sigma s = length As; \bigwedge A. A \in set As \implies \varrho$ -disjoint- ε A ;
 $\bigwedge A. A \in set As \implies at-most-1$ -elem-term $A \rrbracket$
 $\implies \llbracket E \rrbracket (\sigma, A) \in \# mset (zip \sigma s As). P_{SKIPS}\langle A \rangle_{nd} \sigma = P_{SKIPS}\langle\langle_{nd} \otimes \llbracket E \rrbracket As \rangle \rangle_{nd} \sigma s \rangle$

proof (*induct $\sigma s As$ rule: induct-2-lists012*)

case Nil show *?case by (simp, subst P_{SKIPS}-nd-rec, simp)*

next

case (single $\sigma_0 A_0$) show *?case*

by (*auto simp add: σ - σs -conv-defs intro!: inj-onI P_{SKIPS}-nd-eqI-strong*)

next

case (Cons $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

have $\langle \llbracket E \rrbracket (\sigma, A) \in \# mset (zip (\sigma_0 \# \sigma_1 \# \sigma s) (A_0 \# A_1 \# As)). P_{SKIPS}\langle A \rangle_{nd} \sigma =$

$P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket \llbracket E \rrbracket (\sigma, A) \in \# mset (zip (\sigma_1 \# \sigma s) (A_1 \# As)). P_{SKIPS}\langle A \rangle_{nd} \sigma \rangle$ **by** *simp*

also have $\langle \llbracket E \rrbracket (\sigma, A) \in \# mset (zip (\sigma_1 \# \sigma s) (A_1 \# As)). P_{SKIPS}\langle A \rangle_{nd} \sigma =$
 $P_{SKIPS}\langle\langle_{nd} \otimes \llbracket E \rrbracket A_1 \# As \rangle \rangle_{nd} (\sigma_1 \# \sigma s) \rangle$

by (*rule Cons.hyps(3)*) (*simp-all add: Cons.prem*s)

also have $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket P_{SKIPS}\langle\langle_{nd} \otimes \llbracket E \rrbracket A_1 \# As \rangle \rangle_{nd} (\sigma_1 \# \sigma s)$

$=$

$P_{SKIPS}\langle\langle A_0 \rangle_{nd \otimes \llbracket E \rrbracket Rlist} \langle\langle_{nd} \otimes \llbracket E \rrbracket A_1 \# As \rangle \rangle \rangle_{nd} (\sigma_0 \# \sigma_1 \# \sigma s) \rangle$

by (*rule P_{SKIPS}-nd-combine_{Rlist}-Sync*

[OF - ϱ -disjoint- ε -transmission-to-iterated-combine_{nd}-Sync

- at-most-1-elem-term-transmission-to-iterated-combine_{nd}-Sync])

(*simp-all add: Cons.prem*s)

also have $\langle\langle A_0 \rangle_{nd \otimes \llbracket E \rrbracket Rlist} \langle\langle_{nd} \otimes \llbracket E \rrbracket A_1 \# As \rangle \rangle = \langle_{nd} \otimes \llbracket E \rrbracket A_0 \# A_1 \#$

$As\gg\rangle$ by *simp*
finally show *?case* .
qed

lemma *P-nd-compactification-Sync*:
 $\langle \text{length } \sigma s = \text{length } As \implies$
 $\llbracket E \rrbracket (\sigma, A) \in \# \text{ mset } (\text{zip } \sigma s As). P \llbracket A \rrbracket_{nd} \sigma = P \llbracket \llbracket_{nd} \otimes \llbracket E \rrbracket As \rrbracket \rrbracket_{nd} \sigma s \rangle$
proof (*induct* $\sigma s As$ *rule: induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst P-nd-rec*, *simp*)
next
case (*single* $\sigma_0 A_0$) **show** *?case*
by (*simp add: P-nd-from- σ -to- σs -is-P-nd*)
next
case (*Cons* $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$) **thus** *?case*
by (*simp add: P-nd-combine_{RList}-Sync*)
qed

lemma *P_{SKIPS}-d-compactification-Sync*:
 $\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A;$
 $\bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies$
 $\llbracket E \rrbracket (\sigma, A) \in \# \text{ mset } (\text{zip } \sigma s As). P_{SKIPS} \llbracket A \rrbracket_d \sigma = P_{SKIPS} \llbracket \llbracket_d \otimes \llbracket E \rrbracket As \rrbracket \rrbracket_d$
 $\sigma s \rangle$
proof (*induct* $\sigma s As$ *rule: induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst P_{SKIPS}-d-rec*, *simp*)
next
case (*single* $\sigma_0 A_0$) **show** *?case*
by (*auto simp add: RenamingTick-P_{SKIPS}-d σ - σs -conv-defs*
intro!: inj-onI P_{SKIPS}-d-eqI-strong split: option.split)
next
case (*Cons* $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)
have $\varrho\text{-disjoint-}\varepsilon : \langle A \in \text{set } (A_1 \# As) \implies \varrho\text{-disjoint-}\varepsilon A \rangle$ **for** A
by (*simp add: Cons.prem(1)*)
have *indep-enabl* :
 $\langle \llbracket i < \text{length } (A_1 \# As); j < \text{length } (A_1 \# As); i \neq j \rrbracket \implies$
 $\text{indep-enabl } ((A_1 \# As) ! i) ((\sigma_1 \# \sigma s) ! i) E ((A_1 \# As) ! j) ((\sigma_1 \# \sigma s) ! j) \rrbracket$
for $i j$
by (*metis Cons.prem(2) Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)
have $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$ **by** (*simp add: Cons.prem(1)*)
moreover **have** $\langle \varrho\text{-disjoint-}\varepsilon \llbracket_d \otimes \llbracket E \rrbracket A_1 \# As \rrbracket \rangle$
by (*meson $\varrho\text{-disjoint-}\varepsilon \varrho\text{-disjoint-}\varepsilon\text{-transmission-to-iterated-combine}_d\text{-Sync}$*)
moreover **have** $\langle \text{indep-enabl } A_0 \sigma_0 E \llbracket_d \otimes \llbracket E \rrbracket A_1 \# As \rrbracket (\sigma_1 \# \sigma s) \rangle$
by (*metis Cons.hyps(1) Cons.prem(2) length-Cons less-Suc-eq-le*
same-length-indep-transmission-to-iterated-combine}_d\text{-Sync})
ultimately show *?case*
by (*simp add: P_{SKIPS}-d-combine_{RList}-Sync*
*Cons.hyps(3)[OF $\varrho\text{-disjoint-}\varepsilon \text{indep-enabl}$, *simplified*]*)
qed

lemma *P-d-compactification-Sync*:

$\langle \llbracket \text{length } \sigma s = \text{length } As; \wedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies \llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P \langle \langle A \rangle_d \sigma = P \langle \langle_d \otimes \llbracket E \rrbracket As \rangle_d \sigma s \rangle \rangle$
proof (*induct* σs *As* *rule: induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst P-d-rec*, *simp*)
next
case (*single* σ_0 *A*₀) **show** *?case*
by (*simp*, *subst (1 2) P_SKIPS-d-updated- ω*)
(auto simp add: RenamingTick-P_SKIPS-d σ - σs -conv-defs intro!: inj-onI P_SKIPS-d-eqI-strong split: option.split)
next
case (*Cons* σ_0 σ_1 σs *A*₀ *A*₁ *As*)
have *indep-enabl* :
 $\langle \llbracket i < \text{length } (A_1 \# As); j < \text{length } (A_1 \# As); i \neq j \rrbracket \implies \text{indep-enabl } ((A_1 \# As) ! i) ((\sigma_1 \# \sigma s) ! i) E ((A_1 \# As) ! j) ((\sigma_1 \# \sigma s) ! j) \rangle$
for *i j*
by (*metis Cons.premS Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)
have $\langle \text{indep-enabl } A_0 \sigma_0 E \langle \langle_d \otimes \llbracket E \rrbracket A_1 \# As \rangle (\sigma_1 \# \sigma s) \rangle \rangle$
by (*metis Cons.hyps(1) Cons.premS length-Cons less-Suc-eq-le same-length-indep-transmission-to-iterated-combine_d-Sync*)
thus *?case*
by (*simp add: P-d-combine_{RList}-Sync Cons.hyps(3)[OF indep-enabl, simplified]*)
qed

corollary *P_SKIPS-nd-compactification-Sync-order-is-arbitrary*:

$\langle P_{SKIPS} \langle \langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle \rangle_{nd} \sigma s = P_{SKIPS} \langle \langle \langle_{nd} \otimes \llbracket E \rrbracket As' \rangle \rangle_{nd} \sigma s' \rangle$
if $\langle \text{length } \sigma s = \text{length } As \rangle \langle \text{length } \sigma s' = \text{length } As' \rangle$
 $\langle \text{mset } (\text{zip } \sigma s As) = \text{mset } (\text{zip } \sigma s' As') \rangle$
 $\langle \wedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A \rangle$
 $\langle \wedge A. A \in \text{set } As \implies \text{at-most-1-elem-term } A \rangle$
proof –
have $\langle P_{SKIPS} \langle \langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle \rangle_{nd} \sigma s = \llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P_{SKIPS} \langle \langle A \rangle_{nd} \sigma \rangle \rangle$
by (*rule P_SKIPS-nd-compactification-Sync[OF that(1, 4, 5), symmetric]*)
also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s' As'). P_{SKIPS} \langle \langle A \rangle_{nd} \sigma \rangle \rangle$
by (*simp only: that(3)*)
also have $\langle \dots = P_{SKIPS} \langle \langle \langle_{nd} \otimes \llbracket E \rrbracket As' \rangle \rangle_{nd} \sigma s' \rangle$
proof (*rule P_SKIPS-nd-compactification-Sync[OF that(2)]*)
show $\langle A \in \text{set } As' \implies \varrho\text{-disjoint-}\varepsilon A \rangle$ **for** *A*
by (*metis in-set-impl-in-set-zip2 set-mset-mset set-zip-rightD that(2-4)*)
next
show $\langle A \in \text{set } As' \implies \text{at-most-1-elem-term } A \rangle$ **for** *A*
by (*metis in-set-impl-in-set-zip2 set-mset-mset set-zip-rightD that(2, 3, 5)*)

qed
 finally show ?thesis .
 qed

corollary *P-nd-compactification-Sync-order-is-arbitrary:*

$\langle P \langle \langle \text{nd} \otimes [E] \text{ As} \rangle \rangle_{\text{nd}} \sigma s = P \langle \langle \text{nd} \otimes [E] \text{ As}' \rangle \rangle_{\text{nd}} \sigma s' \rangle$
 if $\langle \text{length } \sigma s = \text{length } \text{As} \rangle \langle \text{length } \sigma s' = \text{length } \text{As}' \rangle$
 $\langle \text{mset } (\text{zip } \sigma s \text{ As}) = \text{mset } (\text{zip } \sigma s' \text{ As}') \rangle$

proof –

have $\langle P \langle \langle \text{nd} \otimes [E] \text{ As} \rangle \rangle_{\text{nd}} \sigma s = [E] (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s \text{ As}). P \langle \langle A \rangle \rangle_{\text{nd}} \sigma \rangle$
 by (fact *P-nd-compactification-Sync[OF that(1), symmetric]*)
also have $\langle \dots = [E] (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s' \text{ As}'). P \langle \langle A \rangle \rangle_{\text{nd}} \sigma \rangle$
 by (simp only: *that(3)*)
also have $\langle \dots = P \langle \langle \text{nd} \otimes [E] \text{ As}' \rangle \rangle_{\text{nd}} \sigma s' \rangle$
 by (fact *P-nd-compactification-Sync[OF that(2)]*)
 finally show ?thesis .

qed

corollary *P_SKIPS-d-compactification-Sync-order-is-arbitrary:*

$\langle P_{SKIPS} \langle \langle d \otimes [E] \text{ As} \rangle \rangle_d \sigma s = P_{SKIPS} \langle \langle d \otimes [E] \text{ As}' \rangle \rangle_d \sigma s' \rangle$
 if $\langle \text{length } \sigma s = \text{length } \text{As} \rangle \langle \text{length } \sigma s' = \text{length } \text{As}' \rangle$
 $\langle \text{mset } (\text{zip } \sigma s \text{ As}) = \text{mset } (\text{zip } \sigma s' \text{ As}') \rangle$
 $\langle \bigwedge A. A \in \text{set } \text{As} \implies \varrho\text{-disjoint-}\varepsilon \text{ A} \rangle$
 $\langle \bigwedge i j. [i < \text{length } \text{As}; j < \text{length } \text{As}; i \neq j] \implies$
 $\text{indep-enabl } (\text{As} ! i) (\sigma s ! i) E (\text{As} ! j) (\sigma s ! j) \rangle$

proof –

have $\langle P_{SKIPS} \langle \langle d \otimes [E] \text{ As} \rangle \rangle_d \sigma s = [E] (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s \text{ As}). P_{SKIPS} \langle \langle A \rangle \rangle_d \sigma \rangle$

by (rule *P_SKIPS-d-compactification-Sync[OF that(1, 4, 5), symmetric]*)

also have $\langle \dots = [E] (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s' \text{ As}'). P_{SKIPS} \langle \langle A \rangle \rangle_d \sigma \rangle$
 by (simp only: *that(3)*)

also have $\langle \dots = P_{SKIPS} \langle \langle d \otimes [E] \text{ As}' \rangle \rangle_d \sigma s' \rangle$

proof (rule *P_SKIPS-d-compactification-Sync[OF that(2)]*)

show $\langle A \in \text{set } \text{As}' \implies \varrho\text{-disjoint-}\varepsilon \text{ A} \rangle$ **for** A

by (metis *in-set-impl-in-set-zip2 set-mset-mset set-zip-rightD that(2-4)*)

next

from $\langle \text{length } \sigma s = \text{length } \text{As} \rangle \langle \text{length } \sigma s' = \text{length } \text{As}' \rangle \langle \text{mset } (\text{zip } \sigma s \text{ As}) = \text{mset } (\text{zip } \sigma s' \text{ As}') \rangle$

obtain n **where** $\ast : \langle \text{length } \sigma s = n \rangle \langle \text{length } \sigma s' = n \rangle \langle \text{length } \text{As} = n \rangle \langle \text{length } \text{As}' = n \rangle$

by (metis *length-zip min-less-iff-conj nat-neq-iff size-mset*)

from *that(3)[symmetric, THEN permutation-Ex-bij]* **obtain** f

where $\ast\ast : \langle \text{bij-betw } f \{..<n\} \{..<n\} \rangle$

$\langle i < n \implies \text{zip } \sigma s' \text{ As}' ! i = \text{zip } \sigma s \text{ As} ! f i \rangle$ **for** i **by** (auto simp add: \ast)

{ **fix** i **assume** $\langle i < n \rangle$

hence $\langle f i < n \rangle$ **using** $\ast\ast(1)$ *bij-betwE* **by** blast

from $\langle i < n \rangle$ **have** $\langle \text{zip } \sigma s' \text{ As}' ! i = (\sigma s' ! i, \text{As}' ! i) \rangle$ **by** (simp add: $\ast(2, 4)$)

moreover from $\langle f i < n \rangle$ **have** $\langle \text{zip } \sigma s \text{ As} ! f i = (\sigma s ! f i, \text{As} ! f i) \rangle$

by (simp add: $\ast(1, 3)$)

ultimately have $\langle \sigma s' ! i = \sigma s ! f i \rangle \langle As' ! i = As ! f i \rangle$
 using ******(2)[OF $\langle i < n \rangle$] by *simp-all*
 } note ****** = *this*
 fix $i j$ assume $\langle i < \text{length } As' \rangle \langle j < \text{length } As' \rangle \langle i \neq j \rangle$
 hence $\langle i < n \rangle \langle j < n \rangle$ by (*simp-all add: *(2, 4)*)
 with *bij-betw-imp-surj-on*[OF ******(1)] *bij-betw-imp-inj-on*[OF ******(1)] $\langle i \neq j \rangle$
 have $\langle f i < \text{length } As \rangle \langle f j < \text{length } As \rangle \langle f i \neq f j \rangle$
 by (*auto simp add: * dest: inj-onD*)
 from *that*(5)[OF *this*]
 show $\langle \text{indep-enabl } (As' ! i) (\sigma s' ! i) E (As' ! j) (\sigma s' ! j) \rangle$
 by (*simp add: ******(1, 2) $\langle i < n \rangle \langle j < n \rangle$*)
qed
 finally show *?thesis* .
qed

corollary *P-d-compactification-Sync-order-is-arbitrary:*

$\langle P \langle \langle d \otimes [E] As \rangle \rangle_d \sigma s = P \langle \langle d \otimes [E] As' \rangle \rangle_d \sigma s' \rangle$
 if $\langle \text{length } \sigma s = \text{length } As \rangle \langle \text{length } \sigma s' = \text{length } As' \rangle$
 $\langle \text{mset } (\text{zip } \sigma s As) = \text{mset } (\text{zip } \sigma s' As') \rangle$
 $\langle \bigwedge i j. [i < \text{length } As; j < \text{length } As; i \neq j] \implies$
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$

proof –

have $\langle P \langle \langle d \otimes [E] As \rangle \rangle_d \sigma s = [E] (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P \langle A \rangle_d \sigma \rangle$
 by (*rule P-d-compactification-Sync*[OF *that*(1, 4), *symmetric*])
 also have $\langle \dots = [E] (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s' As'). P \langle A \rangle_d \sigma \rangle$
 by (*simp only: that*(3))
 also have $\langle \dots = P \langle \langle d \otimes [E] As' \rangle \rangle_d \sigma s' \rangle$
proof (*rule P-d-compactification-Sync*[OF *that*(2)])
 from $\langle \text{length } \sigma s = \text{length } As \rangle \langle \text{length } \sigma s' = \text{length } As' \rangle \langle \text{mset } (\text{zip } \sigma s As) =$
 $\text{mset } (\text{zip } \sigma s' As') \rangle$
 obtain n where $* : \langle \text{length } \sigma s = n \rangle \langle \text{length } \sigma s' = n \rangle \langle \text{length } As = n \rangle \langle \text{length } As' = n \rangle$
 by (*metis length-zip min-less-iff-conj nat-neq-iff size-mset*)
 from *that*(3)[*symmetric*, *THEN permutation-Ex-bij*] obtain f
 where ****** : $\langle \text{bij-betw } f \{..<n\} \{..<n\} \rangle$
 $\langle i < n \implies \text{zip } \sigma s' As' ! i = \text{zip } \sigma s As ! f i \rangle$ for i by (*auto simp add: **)
 { fix i assume $\langle i < n \rangle$
 hence $\langle f i < n \rangle$ using ******(1) *bij-betwE* by *blast*
 from $\langle i < n \rangle$ have $\langle \text{zip } \sigma s' As' ! i = (\sigma s' ! i, As' ! i) \rangle$ by (*simp add: *(2, 4)*)
 moreover from $\langle f i < n \rangle$ have $\langle \text{zip } \sigma s As ! f i = (\sigma s ! f i, As ! f i) \rangle$
 by (*simp add: *(1, 3)*)
 ultimately have $\langle \sigma s' ! i = \sigma s ! f i \rangle \langle As' ! i = As ! f i \rangle$
 using ******(2)[OF $\langle i < n \rangle$] by *simp-all*
 } note ****** = *this*
 fix $i j$ assume $\langle i < \text{length } As' \rangle \langle j < \text{length } As' \rangle \langle i \neq j \rangle$
 hence $\langle i < n \rangle \langle j < n \rangle$ by (*simp-all add: *(2, 4)*)
 with *bij-betw-imp-surj-on*[OF ******(1)] *bij-betw-imp-inj-on*[OF ******(1)] $\langle i \neq j \rangle$
 have $\langle f i < \text{length } As \rangle \langle f j < \text{length } As \rangle \langle f i \neq f j \rangle$
 by (*auto simp add: * dest: inj-onD*)

from *that(4)[OF this]*
show $\langle \text{indep-enabl } (As' ! i) (\sigma s' ! i) E (As' ! j) (\sigma s' ! j) \rangle$
by (*simp add: ***(1, 2) $\langle i < n \rangle \langle j < n \rangle$*)
qed
finally show *?thesis .*
qed

6.3 Derived Versions

lemma *P_{SKIPS}-nd-compactification-Sync-upt-version:*

$\langle \llbracket E \rrbracket P \in \# \text{ mset } (\text{map } Q [0..<n]). P = P_{SKIPS} \llbracket \llbracket_{nd} \otimes \llbracket E \rrbracket \text{ map } A [0..<n] \rrbracket \rrbracket_{nd} (\text{replicate } n 0) \rangle$

if $\langle \bigwedge i. i < n \implies \rho\text{-disjoint-}\varepsilon (A i) \rangle$
 $\langle \bigwedge i. i < n \implies \text{at-most-1-elem-term } (A i) \rangle$
 $\langle \bigwedge i. i < n \implies P_{SKIPS} \llbracket A i \rrbracket_{nd} 0 = Q i \rangle$

proof –

have $\langle \llbracket E \rrbracket P \in \# \text{ mset } (\text{map } Q [0..<n]). P = \llbracket E \rrbracket i \in \# (\text{mset-set } \{0..<n\}). Q i \rangle$

by (*auto intro: mono-MultiSync-eq2*)

also have $\langle \dots = \llbracket E \rrbracket i \in \# (\text{mset-set } \{0..<n\}). P_{SKIPS} \llbracket A i \rrbracket_{nd} 0 \rangle$

by (*auto simp add: that(3) intro: mono-MultiSync-eq*)

also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \# \text{ mset } (\text{zip } (\text{replicate } n 0) (\text{map } A [0..<n])). P_{SKIPS} \llbracket A \rrbracket_{nd} \sigma \rangle$

proof (*induct n rule: nat-induct-012*)

case (*Suc k*)

have $\langle \llbracket E \rrbracket i \in \# \text{ mset-set } \{0..<\text{Suc } k\}. P_{SKIPS} \llbracket A i \rrbracket_{nd} 0 = P_{SKIPS} \llbracket A k \rrbracket_{nd} 0 \llbracket E \rrbracket \llbracket E \rrbracket i \in \# \text{ mset-set } \{0..<k\}. P_{SKIPS} \llbracket A i \rrbracket_{nd} 0 \rangle$

by (*subst atLeastLessThanSuc, simp, rule MultiSync-add*)
(metis Suc.hyps(1) atLeastLessThan-empty-iff2 finite-lessThan gr0-conv-Suc lessThan-atLeast0 mset-set-empty-iff order-less-le-trans)

also have $\langle \dots = P_{SKIPS} \llbracket A k \rrbracket_{nd} 0 \llbracket E \rrbracket \llbracket E \rrbracket (\sigma, A) \in \# \text{ mset } (\text{zip } (\text{replicate } k 0) (\text{map } A [0..<k])) \rangle$

$P_{SKIPS} \llbracket A \rrbracket_{nd} \sigma \rangle$

by (*simp only: Suc.hyps(2)*)

also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \# \text{ mset } (\text{zip } (\text{replicate } (\text{Suc } k) 0) (\text{map } A [0..<\text{Suc } k])). P_{SKIPS} \llbracket A \rrbracket_{nd} \sigma \rangle$

by (*simp flip: replicate-append-same, subst MultiSync-add*)

(use Suc.hyps(1) in auto)

finally show *?case .*

qed (*simp-all add: atLeastLessThanSuc Sync-commute*)

also have $\langle \dots = P_{SKIPS} \llbracket \llbracket_{nd} \otimes \llbracket E \rrbracket \text{ map } A [0..<n] \rrbracket \rrbracket_{nd} (\text{replicate } n 0) \rangle$

by (*rule P_{SKIPS}-nd-compactification-Sync*)

(auto simp add: that(1, 2))

finally show *?thesis .*

qed

lemma *P-nd-compactification-Sync-upt-version:*

$\langle \llbracket E \rrbracket P \in \# \text{ mset } (\text{map } Q [0..<n]). P = P \llbracket \llbracket_{nd} \otimes \llbracket E \rrbracket \text{ map } A [0..<n] \rrbracket \rrbracket_{nd} (\text{replicate } n 0) \rangle$

if $\langle \bigwedge i. i < n \implies P \langle A \ i \rangle_{nd} \ 0 = Q \ i \rangle$
proof –
have $\langle \llbracket E \rrbracket P \in \# \text{mset} (\text{map } Q \ [0..<n]). P = \llbracket E \rrbracket i \in \# (\text{mset-set } \{0..<n\}). Q \ i \rangle$
by (*auto intro: mono-MultiSync-eq2*)
also have $\langle \dots = \llbracket E \rrbracket i \in \# (\text{mset-set } \{0..<n\}). P \langle A \ i \rangle_{nd} \ 0 \rangle$
by (*auto simp add: that(1) intro: mono-MultiSync-eq*)
also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip} (\text{replicate } n \ 0) (\text{map } A \ [0..<n])) \rangle$
 $P \langle A \rangle_{nd} \ \sigma \rangle$
proof (*induct n rule: nat-induct-012*)
case (*Suc k*)
have $\langle \llbracket E \rrbracket i \in \# \text{mset-set } \{0..<\text{Suc } k\}. P \langle A \ i \rangle_{nd} \ 0 =$
 $P \langle A \ k \rangle_{nd} \ 0 \llbracket E \rrbracket \llbracket E \rrbracket i \in \# \text{mset-set } \{0..<k\}. P \langle A \ i \rangle_{nd} \ 0 \rangle$
by (*subst atLeastLessThanSuc, simp, rule MultiSync-add*)
(metis Suc.hyps(1) atLeastLessThan-empty-iff2 finite-lessThan
gr0-conv-Suc lessThan-atLeast0 mset-set-empty-iff order-less-le-trans)
also have $\langle \dots = P \langle A \ k \rangle_{nd} \ 0 \llbracket E \rrbracket \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip} (\text{replicate } k \ 0) (\text{map}$
 $A \ [0..<k])) \rangle$. $P \langle A \rangle_{nd} \ \sigma \rangle$
by (*simp only: Suc.hyps(2)*)
also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip} (\text{replicate} (\text{Suc } k) \ 0) (\text{map } A \ [0..<\text{Suc}$
 $k])) \rangle$. $P \langle A \rangle_{nd} \ \sigma \rangle$
by (*simp flip: replicate-append-same, subst MultiSync-add*)
(use Suc.hyps(1) in auto)
finally show *?case .*
qed (*simp-all add: atLeastLessThanSuc Sync-commute*)
also have $\langle \dots = P \langle \langle \langle A \rangle_{nd} \otimes \llbracket E \rrbracket \text{map } A \ [0..<n] \rangle \rangle_{nd} (\text{replicate } n \ 0) \rangle$
by (*auto intro: P-nd-compactification-Sync*)
finally show *?thesis .*
qed

lemma *P_{SKIPS}-d-compactification-Sync-upt-version:*

$\langle \llbracket E \rrbracket P \in \# \text{mset} (\text{map } Q \ [0..<n]). P = P_{SKIPS} \langle \langle \langle A \rangle_{nd} \otimes \llbracket E \rrbracket \text{map } A \ [0..<n] \rangle \rangle_d$
(replicate n 0)

if $\langle \bigwedge i. i < n \implies \varrho\text{-disjoint-}\varepsilon (A \ i) \rangle$

$\langle \bigwedge i \ j. i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) \ 0 \ E \ (A \ j) \ 0 \rangle$

$\langle \bigwedge i. i < n \implies P_{SKIPS} \langle A \ i \rangle_d \ 0 = Q \ i \rangle$

proof –

have $\langle \llbracket E \rrbracket P \in \# \text{mset} (\text{map } Q \ [0..<n]). P = \llbracket E \rrbracket i \in \# (\text{mset-set } \{0..<n\}). Q \ i \rangle$

by (*auto intro: mono-MultiSync-eq2*)

also have $\langle \dots = \llbracket E \rrbracket i \in \# (\text{mset-set } \{0..<n\}). P_{SKIPS} \langle A \ i \rangle_d \ 0 \rangle$

by (*auto simp add: that(3) intro: mono-MultiSync-eq*)

also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip} (\text{replicate } n \ 0) (\text{map } A \ [0..<n])) \rangle$
 $P_{SKIPS} \langle A \rangle_d \ \sigma \rangle$

proof (*induct n rule: nat-induct-012*)

case (*Suc k*)

have $\langle \llbracket E \rrbracket i \in \# \text{mset-set } \{0..<\text{Suc } k\}. P_{SKIPS} \langle A \ i \rangle_d \ 0 =$

$P_{SKIPS} \langle A \ k \rangle_d \ 0 \llbracket E \rrbracket \llbracket E \rrbracket i \in \# \text{mset-set } \{0..<k\}. P_{SKIPS} \langle A \ i \rangle_d \ 0 \rangle$

by (*subst atLeastLessThanSuc, simp, rule MultiSync-add*)

(metis Suc.hyps(1) atLeastLessThan-empty-iff2 finite-lessThan
 gr0-conv-Suc lessThan-atLeast0 mset-set-empty-iff order-less-le-trans)
also have $\langle \dots = P_{SKIPS} \langle A \ k \rangle_d \ 0 \ \llbracket E \rrbracket$
 $\llbracket E \rrbracket (\sigma, A) \in \#mset (\text{zip} (\text{replicate } k \ 0) (\text{map } A \ [0..<k])). P_{SKIPS} \langle A \rangle_d$
 $\sigma \rangle$
by (simp only: Suc.hyps(2))
also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \#mset (\text{zip} (\text{replicate} (Suc \ k) \ 0) (\text{map } A \ [0..<Suc$
 $k])). P_{SKIPS} \langle A \rangle_d \ \sigma \rangle$
by (simp flip: replicate-append-same, subst MultiSync-add)
 (use Suc.hyps(1) in auto)
finally show ?case .
qed (simp-all add: atLeastLessThanSuc Sync-commute)
also have $\langle \dots = P_{SKIPS} \langle \langle_d \otimes \llbracket E \rrbracket \text{map } A \ [0..<n] \rangle \rangle_d (\text{replicate } n \ 0) \rangle$
by (rule P_SKIPS-d-compactification-Sync)
 (auto simp add: that(1, 2))
finally show ?thesis .
qed

lemma *P-d-compactification-Sync-upt-version:*

$\langle \llbracket E \rrbracket P \in \#mset (\text{map } Q \ [0..<n]). P = P \langle \langle_d \otimes \llbracket E \rrbracket \text{map } A \ [0..<n] \rangle \rangle_d (\text{replicate}$
 $n \ 0) \rangle$

if $\langle \bigwedge i. i < n \implies P \langle A \ i \rangle_d \ 0 = Q \ i \rangle$

$\langle \bigwedge i \ j. i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) \ 0 \ E \ (A \ j) \ 0 \rangle$

proof –

have $\langle \llbracket E \rrbracket P \in \#mset (\text{map } Q \ [0..<n]). P = \llbracket E \rrbracket i \in \#(mset\text{-set } \{0..<n\}). Q$
 $i \rangle$

by (auto intro: mono-MultiSync-eq2)

also have $\langle \dots = \llbracket E \rrbracket i \in \#(mset\text{-set } \{0..<n\}). P \langle A \ i \rangle_d \ 0 \rangle$

by (auto simp add: that(1) intro: mono-MultiSync-eq)

also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \#mset (\text{zip} (\text{replicate } n \ 0) (\text{map } A \ [0..<n])). P \langle A \rangle_d$
 $\sigma \rangle$

proof (induct n rule: nat-induct-012)

case (Suc k)

have $\langle \llbracket E \rrbracket i \in \#mset\text{-set } \{0..<Suc \ k\}. P \langle A \ i \rangle_d \ 0 =$

$P \langle A \ k \rangle_d \ 0 \ \llbracket E \rrbracket \llbracket E \rrbracket i \in \#mset\text{-set } \{0..<k\}. P \langle A \ i \rangle_d \ 0 \rangle$

by (subst atLeastLessThanSuc, simp, rule MultiSync-add)

(metis Suc.hyps(1) atLeastLessThan-empty-iff2 finite-lessThan

gr0-conv-Suc lessThan-atLeast0 mset-set-empty-iff order-less-le-trans)

also have $\langle \dots = P \langle A \ k \rangle_d \ 0 \ \llbracket E \rrbracket \llbracket E \rrbracket (\sigma, A) \in \#mset (\text{zip} (\text{replicate } k \ 0) (\text{map}$
 $A \ [0..<k])). P \langle A \rangle_d \ \sigma \rangle$

by (simp only: Suc.hyps(2))

also have $\langle \dots = \llbracket E \rrbracket (\sigma, A) \in \#mset (\text{zip} (\text{replicate} (Suc \ k) \ 0) (\text{map } A \ [0..<Suc$
 $k])). P \langle A \rangle_d \ \sigma \rangle$

by (simp flip: replicate-append-same, subst MultiSync-add)

(use Suc.hyps(1) in auto)

finally show ?case .

qed (simp-all add: atLeastLessThanSuc Sync-commute)

also have $\langle \dots = P \langle \langle_d \otimes \llbracket E \rrbracket \text{map } A \ [0..<n] \rangle \rangle_d (\text{replicate } n \ 0) \rangle$

by (rule P-d-compactification-Sync) (simp-all add: that(2))

finally show *?thesis* .
qed

6.4 More on Iterated Combine

lemma ε -iterated-combine_{*n*d}-Sync-general-form:

$\langle \text{length } \sigma s = \text{length } As \implies \varepsilon \llbracket_{nd} \otimes [E] As \rrbracket \sigma s =$
 (if $As = []$ then $\{\}$)
 else $(\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \cup (\bigcap i < \text{length } As. \varepsilon (As ! i)$
 $(\sigma s ! i)) \rangle$

for $As :: \langle (' \sigma, 'e, 'r) A_{nd} \text{ list} \rangle$

proof (subst ε -iterated-combine_{*n*d}-Sync, assumption, induct σs As rule: induct-2-lists012)

case Nil show *?case* by simp

next

case (single $\sigma_0 A_0$) show *?case* by auto

next

case (Cons $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

define U and I

where $U\text{-def} : \langle U As \sigma s \equiv \bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i) \rangle$

and $I\text{-def} : \langle I As \sigma s \equiv \bigcap i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i) \rangle$

for $As :: \langle (' \sigma, 'e, 'r) A_{nd} \text{ list} \rangle$ and σs

have $*$: $\langle U (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma s) = \varepsilon A_0 \sigma_0 \cup U (A_1 \# As) (\sigma_1 \# \sigma s) \rangle$

by (auto simp add: U-def nth-Cons split: nat.split-asm)

have $**$: $\langle I (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma s) = \varepsilon A_0 \sigma_0 \cap I (A_1 \# As) (\sigma_1 \# \sigma s) \rangle$

by (auto simp add: I-def nth-Cons split: nat.splits)

have $\langle \text{iterated-combine}_{nd}\text{-Sync-}\varepsilon (A_0 \# A_1 \# As) E (\sigma_0 \# \sigma_1 \# \sigma s) =$
 $\text{combine-sets-Sync } (\varepsilon A_0 \sigma_0) E (U (A_1 \# As) (\sigma_1 \# \sigma s) - E \cup (I (A_1 \#$
 $As) (\sigma_1 \# \sigma s))) \rangle$

by (simp add: U-def I-def Cons.hyps(3))

also have $\langle \dots = U (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma s) - E \cup I (A_0 \# A_1 \# As)$
 $(\sigma_0 \# \sigma_1 \# \sigma s) \rangle$

unfolding $*$ $**$ by (auto simp add: U-def I-def)

finally show *?case* by (simp add: U-def I-def)

qed

lemma ε -iterated-combine_{*d*}-Sync-general-form:

$\langle \text{length } \sigma s = \text{length } As \implies \varepsilon \llbracket_d \otimes [E] As \rrbracket \sigma s =$
 (if $As = []$ then $\{\}$)
 else $(\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \cup (\bigcap i < \text{length } As. \varepsilon (As ! i)$
 $(\sigma s ! i)) \rangle$

for $As :: \langle (' \sigma, 'e, 'r) A_d \text{ list} \rangle$

proof (subst ε -iterated-combine_{*d*}-Sync, assumption, induct σs As rule: induct-2-lists012)

case Nil show *?case* by simp

next

case (single $\sigma_0 A_0$) show *?case* by auto

next

case (Cons $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

define U and I
where $U\text{-def} : \langle U\ As\ \sigma s \equiv \bigcup_{i < \text{length}\ As} \varepsilon (As\ !\ i)\ (\sigma s\ !\ i) \rangle$
and $I\text{-def} : \langle I\ As\ \sigma s \equiv \bigcap_{i < \text{length}\ As} \varepsilon (As\ !\ i)\ (\sigma s\ !\ i) \rangle$
for $As :: \langle (' \sigma, 'e, 'r)\ A_d\ list \rangle$ **and** σs
have $*$: $\langle U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \varepsilon\ A_0\ \sigma_0 \cup U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) \rangle$
by (*auto simp add: U-def nth-Cons split: nat.split-asm*)
have $**$: $\langle I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \varepsilon\ A_0\ \sigma_0 \cap I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) \rangle$
by (*auto simp add: I-def nth-Cons split: nat.splits*)
have $\langle \text{iterated-combine}_d\text{-Sync-}\varepsilon\ (A_0\ \#\ A_1\ \#\ As)\ E\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \text{combine-sets-Sync}\ (\varepsilon\ A_0\ \sigma_0)\ E\ (U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) - E \cup I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)) \rangle$
by (*simp add: U-def I-def Cons.hyps(3)*)
also have $\langle \dots = U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) - E \cup I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) \rangle$
unfolding $**$ **by** (*auto simp add: U-def I-def*)
finally show $?case$ **by** (*simp add: U-def I-def*)
qed

lemma τ -iterated-combine $_d$ -Sync-general-form:
 $\langle [\text{length}\ \sigma s = \text{length}\ As; \sigma s' \in \tau\ \langle \langle_{nd} \otimes [E] \rangle As \rangle \sigma s\ a; i < \text{length}\ As] \implies \sigma s' !\ i \in \text{insert}\ (\sigma s !\ i)\ (\tau\ (As !\ i)\ (\sigma s !\ i)\ a) \rangle$
proof (*induct* $\sigma s\ As$ *arbitrary: $\sigma s'$ i rule: induct-2-lists012*)
case Nil **thus** $?case$ **by** *simp*
next
case (*single* $\sigma_0\ A_0$)
from *single.prem*s **show** $?case$ **by** (*auto simp add: behaviour- σ - σs -conv*)
next
case (*Cons* $\sigma_0\ \sigma_1\ \sigma s\ A_0\ A_1\ As$)
from $\langle \text{length}\ \sigma s = \text{length}\ As \rangle$ **have** $\langle \text{length}\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \text{length}\ (A_0\ \#\ A_1\ \#\ As) \rangle$ **by** *simp*
from *same-length- \mathcal{R}_{nd} -iterated-combine $_d$ -Sync-description*[*OF this*]
have $\langle \text{length}\ \sigma s' = \text{length}\ (A_0\ \#\ A_1\ \#\ As) \rangle$
by (*metis Cons.prem*s(1) $\mathcal{R}_{nd}.\text{init}\ \mathcal{R}_{nd}.\text{step}$)
then obtain $\sigma_0' \sigma_1' \sigma s''$ **where** $\langle \sigma s' = \sigma_0' \#\ \sigma_1' \#\ \sigma s'' \rangle$ **by** (*metis length-Suc-conv*)
with *Cons.prem*s *Cons.hyps*(3)[*of* $\langle \sigma_1' \#\ \sigma s'' \rangle$ $\langle \text{nat.pred}\ i \rangle$] **show** $?case$
by (*auto simp add: combine-Sync-defs nth-Cons split: if-split-asm nat.splits*)
qed

lemma τ -iterated-combine $_d$ -Sync-general-form:
 $\langle \text{length}\ \sigma s = \text{length}\ As \implies \tau\ \langle \langle_d \otimes [E] \rangle As \rangle \sigma s\ a = \text{if}\ As = []\ \text{then}\ \diamond\ \text{else}\ \text{if}\ a \in (\bigcup_{i < \text{length}\ As} \varepsilon (As\ !\ i)\ (\sigma s\ !\ i)) - E \cup (\bigcap_{i < \text{length}\ As} \varepsilon (As\ !\ i)\ (\sigma s\ !\ i))\ \text{then}\ [\text{map2}\ (\lambda\ \sigma\ A.\ \text{if}\ a \in \varepsilon\ A\ \sigma\ \text{then}\ [\tau\ A\ \sigma\ a]\ \text{else}\ \sigma)]\ \sigma s\ As\ \text{else}\ \diamond \rangle$

for $As :: \langle ('σ, 'e, 'r) A_d \text{ list} \rangle$
proof (*induct* $σs$ As *rule: induct-2-lists012*)
case *Nil* **show** *?case* **by** *simp*
next
case (*single* $σ_0$ A_0) **show** *?case* **by** (*auto simp add: behaviour-σ-σs-conv ε-simps*)
next
case (*Cons* $σ_0$ $σ_1$ $σs$ A_0 A_1 As)
let $?U = \langle \lambda As \ σs. \bigcup_{i < \text{length } As} \varepsilon (As ! i) (σs ! i) \rangle$
let $?I = \langle \lambda As \ σs. \bigcap_{i < \text{length } As} \varepsilon (As ! i) (σs ! i) \rangle$
show *?case*
proof (*split if-split, split if-split, intro conjI impI*)
show $\langle A_0 \# A_1 \# As = [] \implies \tau \llbracket_d \otimes [E] A_0 \# A_1 \# As \rrbracket (\sigma_0 \# \sigma_1 \# \sigma_s) a = \diamond \rangle$
and $\langle A_0 \# A_1 \# As = [] \implies \tau \llbracket_d \otimes [E] A_0 \# A_1 \# As \rrbracket (\sigma_0 \# \sigma_1 \# \sigma_s) a = \diamond \rangle$ **by** *simp-all*
next
assume $\langle a \notin ?U (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) - E \cup ?I (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) \rangle$
hence $\langle a \notin \varepsilon \llbracket_d \otimes [E] A_0 \# A_1 \# As \rrbracket (\sigma_0 \# \sigma_1 \# \sigma_s) \rangle$
by (*subst ε-iterated-combine_d-Sync-general-form*)
(simp-all add: Cons.hyps(1))
thus $\langle \tau \llbracket_d \otimes [E] A_0 \# A_1 \# As \rrbracket (\sigma_0 \# \sigma_1 \# \sigma_s) a = \diamond \rangle$
by (*simp add: ε-simps*)
next
assume $*$: $\langle a \in ?U (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) - E \cup ?I (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) \rangle$
have $**$: $\langle ?U (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) = \varepsilon A_0 \sigma_0 \cup ?U (A_1 \# As) (\sigma_1 \# \sigma_s) \rangle$
by (*auto simp add: nth-Cons split: nat.split-asm*)
have $***$: $\langle ?I (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) = \varepsilon A_0 \sigma_0 \cap ?I (A_1 \# As) (\sigma_1 \# \sigma_s) \rangle$
by (*auto simp add: nth-Cons split: nat.splits*)
have $****$: $\langle ?U (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) - E \cup ?I (A_0 \# A_1 \# As) (\sigma_0 \# \sigma_1 \# \sigma_s) =$
 $\text{combine-sets-Sync } (\varepsilon A_0 \sigma_0) E (?U (A_1 \# As) (\sigma_1 \# \sigma_s) - E \cup ?I (A_1 \# As) (\sigma_1 \# \sigma_s)) \rangle$
unfolding $**$ $***$ **by** *auto*
have $\$$: $\langle ?U (A_1 \# As) (\sigma_1 \# \sigma_s) - E \cup ?I (A_1 \# As) (\sigma_1 \# \sigma_s) = \varepsilon \llbracket_d \otimes [E] A_1 \# As \rrbracket (\sigma_1 \# \sigma_s) \rangle$
by (*subst ε-iterated-combine_d-Sync-general-form*)
(simp-all add: Cons.hyps(1))
from *Cons.hyps(1)* **have** $\langle a \notin ?U As \ σs \implies$
 $\text{map2 } (\lambda \sigma A. \text{if } a \in \varepsilon A \ \sigma \text{ then } \lceil \tau A \ \sigma \ a \rceil \text{ else } \sigma) \ \sigma_s \ As = \sigma_s \rangle$
by (*induct* $σs$ As *rule: induct-2-lists012*)
(auto simp add: ε-simps lessThan-def, fastforce)
moreover **have** $\langle ?U As \ σs \subseteq ?U (A_1 \# As) (\sigma_1 \# \sigma_s) \rangle$ **by** *force*
ultimately **show** $\langle \tau \llbracket_d \otimes [E] A_0 \# A_1 \# As \rrbracket (\sigma_0 \# \sigma_1 \# \sigma_s) a =$
 $\text{map2 } (\lambda x y. \text{if } a \in \varepsilon y \ x \text{ then } \lceil \tau y \ x \ a \rceil \text{ else } x) (\sigma_0 \# \sigma_1 \# \sigma_s) \rangle$

```

(A0 # A1 # As)]›
  using * unfolding **** $
  by (simp add: ε-combineRlist-Sync combine-Sync-ε-def, safe,
      auto simp add: combineRlist-Sync-defs ε-simps Cons.hyps(3) lessThan-def
      subset-iff
      split: option.splits if-splits)
  (metis (no-types, lifting) not-less-eq nth-Cons-Suc)
qed
qed

```

lemma *indep-implies-only-one-enabled'*:

```

⟨∃!i. i < length As ∧ a ∈ ε (As ! i) (σs ! i)›
  if ⟨length σs = length As›
  and ⟨∧i j. [[i < length As; j < length As; i ≠ j]] ⇒
        ε (As ! i) (σs ! i) ∩ ε (As ! j) (σs ! j) ⊆ E›
  and ⟨a ∈ (∪i < length As. ε (As ! i) (σs ! i)) - E›
proof (rule ex-ex1I)
  from that(3) show ⟨∃i < length As. a ∈ ε (As ! i) (σs ! i)› by auto
next
  fix i j assume ⟨i < length As ∧ a ∈ ε (As ! i) (σs ! i)›
    ⟨j < length As ∧ a ∈ ε (As ! j) (σs ! j)›
  moreover from that(3) have ⟨a ∉ E› by blast
  ultimately show ⟨i = j› using that(2)[of i j] by auto
qed

```

lemma *indep-implies-only-one-enabled*:

```

⟨[[length σs = length As;
  ∧i j. [[i < length As; j < length As; i ≠ j]] ⇒
        indep-enabl (As ! i) (σs ! i) E (As ! j) (σs ! j);
  a ∈ (∪i < length As. ε (As ! i) (σs ! i)) - E]] ⇒
  ∃!i. i < length As ∧ a ∈ ε (As ! i) (σs ! i)›
by (erule indep-implies-only-one-enabled'[where E = E])
  (simp-all add: indep-enabl-def subset-iff, meson IntI  $\mathcal{R}_d$ .init)

```

lemma *τ-iterated-combine_d-Sync-general-form-when-indep*:

```

⟨τ ⟨ $\mathbb{A}_d \otimes [E]$  As⟩ σs a =
  (if As = [] then ◇
   else if a ∈ (∩i < length As. ε (As ! i) (σs ! i))
     then [map2 (λσ A. [τ A σ a]) σs As]
     else if a ∈ (∪i < length As. ε (As ! i) (σs ! i)) - E
       then let i = THE i. i < length As ∧ a ∈ ε (As ! i) (σs ! i)
         in [σs[i := [τ (As ! i) (σs ! i) a]]]
         else ◇)›
(is ⟨- = (if As = [] then ◇ else
        if a ∈ ?I As σs then [map2 (λσ A. [τ A σ a]) σs As] else
        if a ∈ ?U As σs - E then ?upd As σs else ◇)›)
if ⟨length σs = length As›

```

$\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$
proof (*subst τ -iterated-combine_d-Sync-general-form*[*OF that(1)*], *rule if-cong*)
show $\langle (\text{if } a \in ?U As \sigma s - E \cup ?I As \sigma s$
 $\text{then } \llbracket \text{map2 } (\lambda x y. \text{if } a \in \varepsilon y x \text{ then } \lceil \tau y x a \rceil \text{ else } x) \sigma s As \rrbracket \text{ else } \diamond \rangle =$
 $(\text{if } a \in ?I As \sigma s \text{ then } \llbracket \text{map2 } (\lambda x y. \lceil \tau y x a \rceil) \sigma s As \rrbracket \text{ else}$
 $\text{if } a \in ?U As \sigma s - E \text{ then } ?\text{upd } As \sigma s \text{ else } \diamond) \rangle$
proof (*split if-split, intro conjI impI*)
assume $*$: $\langle a \in ?I As \sigma s \rangle$
with *that(1)* **have** $\langle (\sigma, A) \in \text{set } (\text{zip } \sigma s As) \implies a \in \varepsilon A \sigma \rangle$ **for** σA
by (*induct $\sigma s As$ rule: list-induct2*) (*use lessThan-Suc-eq-insert-0 in auto*)
with $*$ **show** $\langle (\text{if } a \in ?U As \sigma s - E \cup ?I As \sigma s$
 $\text{then } \llbracket \text{map2 } (\lambda x y. \text{if } a \in \varepsilon y x \text{ then } \lceil \tau y x a \rceil \text{ else } x) \sigma s As \rrbracket \text{ else}$
 $\diamond) =$
 $\llbracket \text{map2 } (\lambda x y. \lceil \tau y x a \rceil) \sigma s As \rrbracket$ **by** *auto*

next
assume $*$: $\langle a \notin ?I As \sigma s \rangle$
show $\langle (\text{if } a \in ?U As \sigma s - E \cup ?I As \sigma s$
 $\text{then } \llbracket \text{map2 } (\lambda x y. \text{if } a \in \varepsilon y x \text{ then } \lceil \tau y x a \rceil \text{ else } x) \sigma s As \rrbracket \text{ else } \diamond) =$
 $(\text{if } a \in ?U As \sigma s - E \text{ then } ?\text{upd } As \sigma s \text{ else } \diamond) \rangle$
proof (*rule if-cong*)
assume $\langle a \in ?U As \sigma s - E \rangle$
from *indep-implies-only-one-enabled*[*OF that this*]
obtain i **where** $\$$: $\langle i < \text{length } As \rangle$ $\langle a \in \varepsilon (As ! i) (\sigma s ! i) \rangle$
 $\langle j < \text{length } As \implies j \neq i \implies a \notin \varepsilon (As ! j) (\sigma s ! j) \rangle$ **for** j **by** *blast*
have $\$\$$: $\langle (\text{THE } i. i < \text{length } As \wedge a \in \varepsilon (As ! i) (\sigma s ! i)) = i \rangle$
using $\$$ **by** *blast*
have $\langle \llbracket \text{map2 } (\lambda x y. \text{if } a \in \varepsilon y x \text{ then } \lceil \tau y x a \rceil \text{ else } x) \sigma s As \rrbracket =$
 $\llbracket \sigma s[i := \lceil \tau (As ! i) (\sigma s ! i) a \rceil] \rrbracket \rangle$
by (*auto intro!: nth-equalityI simp add: that(1)*)
(use that(1) $\$(3)$ in fastforce, metis $\$(2)$ nth-list-update-neg)
also have $\langle \dots = ?\text{upd } As \sigma s \rangle$
using $\$\$$ **by** *auto*
finally show $\langle \llbracket \text{map2 } (\lambda x y. \text{if } a \in \varepsilon y x \text{ then } \lceil \tau y x a \rceil \text{ else } x) \sigma s As \rrbracket = ?\text{upd}$
 $As \sigma s \rangle$.
qed (*use * in auto*)
qed
qed *simp-all*

6.5 More on Events

lemma *events-of-MultiSync-P_SKIP_S-nd* :
 $\langle \alpha(\llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P_SKIP_S \langle A \rangle_{nd} \sigma) =$
 $(\text{if } As = [] \text{ then } \{\} \text{ else}$
 $\bigcup \sigma s' \in \mathcal{R}_{nd} \langle \llbracket E \rrbracket As \rangle \sigma s. (\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s' ! i)) - E \cup$
 $(\bigcap i < \text{length } As. \varepsilon (As ! i) (\sigma s' ! i))) \rangle$
(is $\leftarrow = ?rhs$) **if** $\langle \text{length } \sigma s = \text{length } As \rangle$
 $\langle \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A \rangle$ $\langle \bigwedge A. A \in \text{set } As \implies \text{at-most-1-elem-term}$
 $A \rangle$

proof –
have $\langle \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P_{SKIPs} \langle A \rangle_{nd} \sigma = P_{SKIPs} \langle \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \rangle_{nd} \sigma s \rangle$
by (*simp only: P_{SKIPs}-nd-compactification-Sync[OF that]*)
also have $\langle \alpha(\dots) = \bigcup (\varepsilon \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \text{ ‘ } \mathcal{R}_{nd} \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s) \rangle$
proof (*rule events-of-P_{SKIPs}-nd*)
show $\langle \varrho\text{-disjoint-}\varepsilon \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \rangle$
by (*simp only: \varrho-disjoint-}\varepsilon-transmission-to-iterated-combine_{nd}-Sync that(2)*)
qed
also from *same-length-}\mathcal{R}_{nd}-iterated-combine_{nd}-Sync-description[OF that(1)]*
have $\langle \dots = ?rhs \rangle$ **by** (*auto simp add: \varepsilon-iterated-combine_{nd}-Sync-general-form*)
finally show *?thesis* .
qed

lemma *events-of-MultiSync-P-nd* :
 $\langle \alpha(\llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P \langle A \rangle_{nd} \sigma) =$
(if As = [] then {} else
 $\bigcup \sigma s' \in \mathcal{R}_{nd} \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s. (\bigcup i < \text{length As}. \varepsilon (As ! i) (\sigma s' ! i)) - E \cup$
 $(\bigcap i < \text{length As}. \varepsilon (As ! i) (\sigma s' ! i))) \rangle$
(is <- = ?rhs) if <length \sigma s = length As>
proof –
have $\langle \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P \langle A \rangle_{nd} \sigma = P \langle \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \rangle_{nd} \sigma s \rangle$
by (*fact P-nd-compactification-Sync[OF that]*)
also have $\langle \alpha(\dots) = \bigcup (\varepsilon \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \text{ ‘ } \mathcal{R}_{nd} \langle_{nd} \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s) \rangle$ **by** (*fact events-of-P-nd*)
also from *same-length-}\mathcal{R}_{nd}-iterated-combine_{nd}-Sync-description[OF that(1)]*
have $\langle \dots = ?rhs \rangle$ **by** (*auto simp add: \varepsilon-iterated-combine_{nd}-Sync-general-form*)
finally show *?thesis* .
qed

lemma *events-of-MultiSync-P_{SKIPs}-d* :
 $\langle \alpha(\llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P_{SKIPs} \langle A \rangle_d \sigma) =$
(if As = [] then {} else
 $\bigcup \sigma s' \in \mathcal{R}_d \langle_d \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s. (\bigcup i < \text{length As}. \varepsilon (As ! i) (\sigma s' ! i) - E \cup$
 $(\bigcap i < \text{length As}. \varepsilon (As ! i) (\sigma s' ! i))) \rangle$
(is <- = ?rhs) if <length \sigma s = length As> <\bigwedge A. A \in set As \implies \varrho-disjoint-}\varepsilon A>
 $\langle \bigwedge i j. [i < \text{length As}; j < \text{length As}; i \neq j] \implies$
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$
proof –

have $\langle \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P_{SKIPs} \langle A \rangle_d \sigma = P_{SKIPs} \langle \langle_d \otimes \llbracket E \rrbracket \text{As} \rangle \rangle_d \sigma s \rangle$
by (*simp add: P_{SKIPs}-d-compactification-Sync[OF that]*)
also have $\langle \alpha(\dots) = \bigcup (\varepsilon \langle_d \otimes \llbracket E \rrbracket \text{As} \rangle \text{ ‘ } \mathcal{R}_d \langle_d \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s) \rangle$
proof (*rule events-of-P_{SKIPs}-d*)
show $\langle \varrho\text{-disjoint-}\varepsilon \langle_d \otimes \llbracket E \rrbracket \text{As} \rangle \rangle$
by (*simp only: \varrho-disjoint-}\varepsilon-transmission-to-iterated-combine_d-Sync that(2)*)

qed
also from *same-length- \mathcal{R}_d -iterated-combine $_d$ -Sync-description*[*OF that(1)*]
have $\langle \dots = ?rhs \rangle$ **by** (*auto simp add: ε -iterated-combine $_d$ -Sync-general-form*)
finally show *?thesis* .
qed

lemma *events-of-MultiSync-P-d* :

$\langle \alpha(\llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P\langle A \rangle_d \sigma) =$
(if $\text{As} = []$ *then* $\{\}$ *else*
 $\bigcup \sigma s' \in \mathcal{R}_d \langle \langle d \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s. (\bigcup_{i < \text{length As}} \varepsilon (\text{As} ! i) (\sigma s' ! i) - E \cup$
 $(\bigcap_{i < \text{length As}} \varepsilon (\text{As} ! i) (\sigma s' ! i))) \rangle$
(is $\langle - = ?rhs \rangle$ *if* $\langle \text{length } \sigma s = \text{length As} \rangle$
and $\langle \bigwedge i j. \llbracket i < \text{length As}; j < \text{length As}; i \neq j \rrbracket \implies$
 $\text{indep-enabl } (\text{As} ! i) (\sigma s ! i) E (\text{As} ! j) (\sigma s ! j) \rangle$

proof –

have $\langle \llbracket E \rrbracket (\sigma, A) \in \# \text{mset} (\text{zip } \sigma s \text{ As}). P\langle A \rangle_d \sigma = P\langle \langle d \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s \rangle$
by (*simp add: P-d-compactification-Sync*[*OF that*])
also have $\langle \alpha(\dots) = \bigcup (\varepsilon \langle \langle d \otimes \llbracket E \rrbracket \text{As} \rangle ' \mathcal{R}_d \langle \langle d \otimes \llbracket E \rrbracket \text{As} \rangle \sigma s \rangle) \rangle$ **by** (*fact*
events-of-P-d)
also from *same-length- \mathcal{R}_d -iterated-combine $_d$ -Sync-description*[*OF that(1)*]
have $\langle \dots = ?rhs \rangle$ **by** (*auto simp add: ε -iterated-combine $_d$ -Sync-general-form*)
finally show *?thesis* .
qed

Chapter 7

Combining Automata for Generalized Synchronization Product

7.1 Definitions

7.1.1 Specializations

definition $combine_{dPairlist-Sync_{ptick}} ::$
 $\langle [(\sigma, 'e, 'r, 'α) A_d\text{-scheme}, 'e \text{ set}, (\sigma, 'e, 'r, 'β) A_d\text{-scheme}] \Rightarrow (\sigma \text{ list}, 'e, 'r \text{ list}) A_d \rangle$

where $\langle combine_{dPairlist-Sync_{ptick}} A_0 E A_1 \equiv$
 $combine_{d-Sync} A_0 E A_1 hd (\lambda \sigma s. hd (tl \sigma s)) (\lambda s t. [s, t]) (\lambda s t. [[s, t]]) \rangle$

definition $combine_{ndPairlist-Sync_{ptick}} ::$
 $\langle [(\sigma, 'e, 'r, 'α) A_{nd}\text{-scheme}, 'e \text{ set}, (\sigma, 'e, 'r, 'β) A_{nd}\text{-scheme}] \Rightarrow (\sigma \text{ list}, 'e, 'r \text{ list}) A_{nd} \rangle$

where $\langle combine_{ndPairlist-Sync_{ptick}} A_0 E A_1 \equiv combine_{nd-Sync} A_0 E A_1 hd$
 $(\lambda \sigma s. hd (tl \sigma s)) (\lambda s t. [s, t]) (\lambda s t. [[s, t]]) \rangle$

definition $combine_{dPair-Sync_{ptick}} ::$
 $\langle [(\sigma_0, 'e, 'r_0, 'α) A_d\text{-scheme}, 'e \text{ set}, (\sigma_1, 'e, 'r_1, 'β) A_d\text{-scheme}] \Rightarrow (\sigma_0 \times \sigma_1,$
 $'e, 'r_0 \times 'r_1) A_d \rangle$

where $\langle combine_{dPair-Sync_{ptick}} A_0 E A_1 \equiv combine_{d-Sync} A_0 E A_1 fst snd Pair$
 $(\lambda s r. [(s, r)]) \rangle$

definition $combine_{ndPair-Sync_{ptick}} ::$
 $\langle [(\sigma_0, 'e, 'r_0, 'α) A_{nd}\text{-scheme}, 'e \text{ set}, (\sigma_1, 'e, 'r_1, 'β) A_{nd}\text{-scheme}] \Rightarrow (\sigma_0 \times$
 $\sigma_1, 'e, 'r_0 \times 'r_1) A_{nd} \rangle$

where $\langle combine_{ndPair-Sync_{ptick}} A_0 E A_1 \equiv combine_{nd-Sync} A_0 E A_1 fst snd$
 $Pair (\lambda s r. [(s, r)]) \rangle$

definition $combine_{dListslenL-Sync_{ptick}} ::$
 $\langle [(\sigma \text{ list}, 'e, 'r \text{ list}, 'α) A_d\text{-scheme}, nat, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, 'β) A_d\text{-scheme}]$
 $\Rightarrow (\sigma \text{ list}, 'e, 'r \text{ list}) A_d \rangle$

where $\langle \text{combinedListslenL-Syncptick } A_0 \text{ len}_0 E A_1 \equiv \text{combined-Sync } A_0 E A_1$
 $(\text{take len}_0) (\text{drop len}_0) (\text{@}) (\lambda s r. [s \text{@} r]) \rangle$

definition $\text{combine}_{\text{ndListslenL-Syncptick}} ::$

$\langle [(\sigma \text{ list}, 'e, 'r \text{ list}, ' \alpha) A_{\text{nd-scheme}}, \text{nat}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, ' \beta) A_{\text{nd-scheme}}]$
 $\Rightarrow (\sigma \text{ list}, 'e, 'r \text{ list}) A_{\text{nd}} \rangle$

where $\langle \text{combine}_{\text{ndListslenL-Syncptick}} A_0 \text{ len}_0 E A_1 \equiv \text{combine}_{\text{nd-Sync}} A_0 E A_1$
 $(\text{take len}_0) (\text{drop len}_0) (\text{@}) (\lambda s r. [s \text{@} r]) \rangle$

definition $\text{combine}_{\text{dRlist-Syncptick}} ::$

$\langle [(\sigma, 'e, 'r, ' \alpha) A_{\text{d-scheme}}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, ' \beta) A_{\text{d-scheme}}] \Rightarrow (\sigma \text{ list},$
 $'e, 'r \text{ list}) A_{\text{d}} \rangle$

where $\langle \text{combine}_{\text{dRlist-Syncptick}} A_0 E A_1 \equiv \text{combine}_{\text{d-Sync}} A_0 E A_1 \text{hd tl} (\#)$
 $(\lambda s r. [s \# r]) \rangle$

definition $\text{combine}_{\text{ndRlist-Syncptick}} ::$

$\langle [(\sigma, 'e, 'r, ' \alpha) A_{\text{nd-scheme}}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, ' \beta) A_{\text{nd-scheme}}] \Rightarrow (\sigma$
 $\text{list}, 'e, 'r \text{ list}) A_{\text{nd}} \rangle$

where $\langle \text{combine}_{\text{ndRlist-Syncptick}} A_0 E A_1 \equiv \text{combine}_{\text{nd-Sync}} A_0 E A_1 \text{hd tl} (\#)$
 $(\lambda s r. [s \# r]) \rangle$

lemmas $\text{combine}_{\text{Pairlist-Syncptick-defs}} = \text{combine}_{\text{dPairlist-Syncptick-def}} \text{combine}_{\text{ndPairlist-Syncptick-def}}$

and $\text{combine}_{\text{Pair-Syncptick-defs}} = \text{combine}_{\text{dPair-Syncptick-def}} \text{combine}_{\text{ndPair-Syncptick-def}}$

and $\text{combine}_{\text{ListslenL-Syncptick-defs}} = \text{combine}_{\text{dListslenL-Syncptick-def}} \text{combine}_{\text{ndListslenL-Syncptick-def}}$

and $\text{combine}_{\text{Rlist-Syncptick-defs}} = \text{combine}_{\text{dRlist-Syncptick-def}} \text{combine}_{\text{ndRlist-Syncptick-def}}$

lemmas $\text{combine-Syncptick-defs} =$

$\text{combine}_{\text{Pairlist-Syncptick-defs}} \text{combine}_{\text{Pair-Syncptick-defs}} \text{combine}_{\text{ListslenL-Syncptick-defs}}$
 $\text{combine}_{\text{Rlist-Syncptick-defs}}$

bundle $\text{combine}_{\text{nd-Syncptick-syntax}} \text{begin}$

notation $\text{combine}_{\text{dPairlist-Syncptick}} (\langle \langle - \text{d} \otimes [-] \checkmark_{\text{Pairlist}} - \rangle \rangle [0, 0, 0])$

notation $\text{combine}_{\text{ndPairlist-Syncptick}} (\langle \langle - \text{nd} \otimes [-] \checkmark_{\text{Pairlist}} - \rangle \rangle [0, 0, 0])$

notation $\text{combine}_{\text{dPair-Syncptick}} (\langle \langle - \text{d} \otimes [-] \checkmark_{\text{Pair}} - \rangle \rangle [0, 0, 0])$

notation $\text{combine}_{\text{ndPair-Syncptick}} (\langle \langle - \text{nd} \otimes [-] \checkmark_{\text{Pair}} - \rangle \rangle [0, 0, 0])$

notation $\text{combine}_{\text{dListslenL-Syncptick}} (\langle \langle - \text{d} \otimes [-, -] \checkmark_{\text{ListslenL}} - \rangle \rangle [0, 0, 0, 0])$

notation $\text{combine}_{\text{ndListslenL-Syncptick}} (\langle \langle - \text{nd} \otimes [-, -] \checkmark_{\text{ListslenL}} - \rangle \rangle [0, 0, 0, 0])$

notation $\text{combine}_{\text{dRlist-Syncptick}} (\langle \langle - \text{d} \otimes [-] \checkmark_{\text{Rlist}} - \rangle \rangle [0, 0, 0])$

notation $\text{combine}_{\text{ndRlist-Syncptick}} (\langle \langle - \text{nd} \otimes [-] \checkmark_{\text{Rlist}} - \rangle \rangle [0, 0, 0])$

end

unbundle $\text{combine}_{\text{nd-Syncptick-syntax}}$

7.2 First Properties

lemma $\text{finite-trans-combine}_{\text{nd-Syncptick-simps}} [\text{simp}] :$

$\langle \text{finite-trans } A_0 \Rightarrow \text{finite-trans } A_1 \Rightarrow \text{finite-trans } \langle \langle A_0 \text{nd} \otimes [E] \checkmark_{\text{Pairlist}} A_1 \rangle \rangle \rangle$

$\langle \text{finite-trans } B_0 \implies \text{finite-trans } B_1 \implies \text{finite-trans } \langle\langle B_0 \text{ nd} \otimes [E] \checkmark_{Pair} B_1 \rangle\rangle \rangle$
 $\langle \text{finite-trans } C_0 \implies \text{finite-trans } C_1 \implies \text{finite-trans } \langle\langle C_0 \text{ nd} \otimes [len_0, E] \checkmark_{ListslenL} C_1 \rangle\rangle \rangle$
 $\langle \text{finite-trans } D_0 \implies \text{finite-trans } D_1 \implies \text{finite-trans } \langle\langle D_0 \text{ nd} \otimes [E] \checkmark_{Rlist} D_1 \rangle\rangle \rangle$
unfolding $\text{combine}_{\text{ndPairlist-Sync}_{\text{ptick-def}}} \text{combine}_{\text{ndPair-Sync}_{\text{ptick-def}}} \text{combine}_{\text{ndListslenL-Sync}_{\text{ptick-def}}} \text{combine}_{\text{ndRlist-Sync}_{\text{ptick-def}}}$
by (*simp-all add: finite-trans-def finite-image-set2*)

lemma ε - $\text{combine}_{\text{Pairlist-Sync}_{\text{ptick}}}$:

$\langle \varepsilon \langle\langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 \text{ hd } (hd \circ tl) \sigma s \rangle$
 $\langle \varepsilon \langle\langle B_0 \text{ nd} \otimes [E] \checkmark_{Pairlist} B_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 \text{ hd } (hd \circ tl) \sigma s \rangle$
by (*auto simp add: combine-Sync-}\varepsilon-def-bis combine_{\text{Pairlist-Sync}_{\text{ptick-defs}}} \varepsilon-simps*)

lemma ε - $\text{combine}_{\text{Pair-Sync}_{\text{ptick}}}$:

$\langle \varepsilon \langle\langle A_0 \text{ d} \otimes [E] \checkmark_{Pair} A_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 \text{ fst snd } \sigma s \rangle$
 $\langle \varepsilon \langle\langle B_0 \text{ nd} \otimes [E] \checkmark_{Pair} B_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 \text{ fst snd } \sigma s \rangle$
by (*auto simp add: combine-Sync-}\varepsilon-def-bis combine_{\text{Pair-Sync}_{\text{ptick-defs}}} \varepsilon-simps*)

lemma ε - $\text{combine}_{\text{ListslenL-Sync}_{\text{ptick}}}$:

$\langle \varepsilon \langle\langle A_0 \text{ d} \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 \text{ (take len}_0) \sigma s \rangle$
 $\langle \text{drop len}_0 \sigma s \rangle$
 $\langle \varepsilon \langle\langle B_0 \text{ nd} \otimes [len_0, E] \checkmark_{ListslenL} B_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 \text{ (take len}_0) \sigma s \rangle$
 $\langle \text{drop len}_0 \sigma s \rangle$
by (*auto simp add: combine-Sync-}\varepsilon-def-bis combine_{\text{ListslenL-Sync}_{\text{ptick-defs}}} \varepsilon-simps*)

lemma ε - $\text{combine}_{\text{Rlist-Sync}_{\text{ptick}}}$:

$\langle \varepsilon \langle\langle A_0 \text{ d} \otimes [E] \checkmark_{Rlist} A_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 \text{ hd tl } \sigma s \rangle$
 $\langle \varepsilon \langle\langle B_0 \text{ nd} \otimes [E] \checkmark_{Rlist} B_1 \rangle\rangle \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 \text{ hd tl } \sigma s \rangle$
by (*auto simp add: combine-Sync-}\varepsilon-def-bis combine_{\text{Rlist-Sync}_{\text{ptick-defs}}} \varepsilon-simps*)

lemma ϱ - $\text{combine}_{\text{Pairlist-Sync}_{\text{ptick}}}$:

$\langle \varrho \langle\langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle\rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho A_0 \wedge \text{hd } (tl \sigma s) \in \varrho A_1 \} \rangle$
 $\langle \varrho \langle\langle B_0 \text{ nd} \otimes [E] \checkmark_{Pairlist} B_1 \rangle\rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho B_0 \wedge \text{hd } (tl \sigma s) \in \varrho B_1 \} \rangle$
by (*auto simp add: combine_{\text{Pairlist-Sync}_{\text{ptick-defs}}} \varrho-simps split: option.split*)

lemma ϱ - $\text{combine}_{\text{Pair-Sync}_{\text{ptick}}}$:

$\langle \varrho \langle\langle A_0 \text{ d} \otimes [E] \checkmark_{Pair} A_1 \rangle\rangle = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho A_0 \wedge \sigma_1 \in \varrho A_1 \} \rangle$
 $\langle \varrho \langle\langle B_0 \text{ nd} \otimes [E] \checkmark_{Pair} B_1 \rangle\rangle = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho B_0 \wedge \sigma_1 \in \varrho B_1 \} \rangle$
by (*auto simp add: combine_{\text{Pair-Sync}_{\text{ptick-defs}}} \varrho-simps split: option.split*)

lemma ϱ - $\text{combine}_{\text{ListslenL-Sync}_{\text{ptick}}}$:

$\langle \varrho \langle\langle A_0 \text{ d} \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle\rangle = \{ \sigma s. \text{take len}_0 \sigma s \in \varrho A_0 \wedge \text{drop len}_0 \sigma s \in \varrho A_1 \} \rangle$
 $\langle \varrho \langle\langle B_0 \text{ nd} \otimes [len_0, E] \checkmark_{ListslenL} B_1 \rangle\rangle = \{ \sigma s. \text{take len}_0 \sigma s \in \varrho B_0 \wedge \text{drop len}_0 \sigma s \in \varrho B_1 \} \rangle$
by (*auto simp add: combine_{\text{ListslenL-Sync}_{\text{ptick-defs}}} \varrho-simps split: option.split*)

lemma ϱ - $\text{combine}_{\text{Rlist-Sync}_{\text{ptick}}}$:

$\langle \varrho \langle A_0 \text{ } d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho A_0 \wedge \text{tl } \sigma s \in \varrho A_1 \}$
 $\langle \varrho \langle B_0 \text{ } nd \otimes [E] \checkmark_{Rlist} B_1 \rangle \rangle = \{ \sigma s. \text{hd } \sigma s \in \varrho B_0 \wedge \text{tl } \sigma s \in \varrho B_1 \}$
by (*auto simp add: combine_{Rlist-Sync_{ptick}}-defs ϱ -simps split: option.split*)

7.3 Transitions are unchanged in the Generalization

In the generalization, only the ω function is modified.

lemma τ -combine_{Pairlist-Sync_{ptick}} :

$\langle \tau \langle A_0 \text{ } d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle = \tau \langle A_0 \text{ } d \otimes [E]_{Pairlist} A_1 \rangle$
 $\langle \tau \langle B_0 \text{ } nd \otimes [E] \checkmark_{Pairlist} B_1 \rangle \rangle = \tau \langle B_0 \text{ } nd \otimes [E]_{Pairlist} B_1 \rangle$
by (*simp-all add: combine-Sync-defs combine-Sync_{ptick}-defs*)

lemma τ -combine_{Pair-Sync_{ptick}} :

$\langle \tau \langle A_0 \text{ } d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle = \tau \langle A_0 \text{ } d \otimes [E]_{Pair} A_1 \rangle$
 $\langle \tau \langle B_0 \text{ } nd \otimes [E] \checkmark_{Pair} B_1 \rangle \rangle = \tau \langle B_0 \text{ } nd \otimes [E]_{Pair} B_1 \rangle$
by (*simp-all add: combine-Sync-defs combine-Sync_{ptick}-defs*)

lemma τ -combine_{ListLenL-Sync_{ptick}} :

$\langle \tau \langle A_0 \text{ } d \otimes [len_0, E] \checkmark_{ListLenL} A_1 \rangle \rangle = \tau \langle A_0 \text{ } d \otimes [len_0, E]_{ListLenL} A_1 \rangle$
 $\langle \tau \langle B_0 \text{ } nd \otimes [len_0, E] \checkmark_{ListLenL} B_1 \rangle \rangle = \tau \langle B_0 \text{ } nd \otimes [len_0, E]_{ListLenL} B_1 \rangle$
by (*simp-all add: combine-Sync-defs combine-Sync_{ptick}-defs*)

$\tau \langle A_0 \text{ } d \otimes [E] \checkmark_{Rlist} A_1 \rangle$ and $\tau \langle B_0 \text{ } nd \otimes [E] \checkmark_{Rlist} B_1 \rangle$ cannot be obtained that easily because of the types of terminations.

7.4 Reachability

lemma \mathcal{R}_d -combine_{dListLenL-Sync_{ptick}-subset}:

$\langle \mathcal{R}_d \langle A_0 \text{ } d \otimes [len_0, E] \checkmark_{ListLenL} A_1 \rangle (s_0 @ s_1) \subseteq \{ t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1 \} \rangle$ (**is** $\langle ?S_A \subseteq \cdot \rangle$)
if $\langle \bigwedge t_0. t_0 \in \mathcal{R}_d A_0 s_0 \implies \text{length } t_0 = len_0 \rangle$
by (*subst same- τ -implies-same- \mathcal{R}_d [of - - $\langle A_0 \text{ } d \otimes [len_0, E]_{ListLenL} A_1 \rangle$]*)
(simp-all add: τ -combine_{ListLenL-Sync_{ptick}} \mathcal{R}_d -combine_{dListLenL-Sync-subset} that)

lemma \mathcal{R}_{nd} -combine_{ndListLenL-Sync_{ptick}-subset}:

$\langle \mathcal{R}_{nd} \langle B_0 \text{ } nd \otimes [len_0, E] \checkmark_{ListLenL} B_1 \rangle (s_0 @ s_1) \subseteq \{ t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1 \} \rangle$ (**is** $\langle ?S_B \subseteq \cdot \rangle$)
if $\langle \bigwedge t_0. t_0 \in \mathcal{R}_{nd} B_0 s_0 \implies \text{length } t_0 = len_0 \rangle$
by (*subst same- τ -implies-same- \mathcal{R}_{nd} [of - - $\langle B_0 \text{ } nd \otimes [len_0, E]_{ListLenL} B_1 \rangle$]*)
(simp-all add: τ -combine_{ListLenL-Sync_{ptick}} \mathcal{R}_{nd} -combine_{ndListLenL-Sync-subset} that)

lemma \mathcal{R}_d -combine_{dPairlist-Sync_{ptick}-subset}:

$\langle \mathcal{R}_d \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 \text{ } t_1. t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1\} \rangle$ (is $\langle ?S_A \subseteq - \rangle$)
and \mathcal{R}_{nd} -combine $_{ndPairlist}$ -Sync $_{ptick}$ -subset:
 $\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ } nd \otimes [E] \checkmark_{Pairlist} B_1 \rangle \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 \text{ } t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$ (is $\langle ?S_B \subseteq - \rangle$)
proof safe
show $\langle t \in ?S_A \implies \exists t_0 \text{ } t_1. t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1 \rangle$ **for** t
by (\mathcal{R}_d -subset-method defs: combine $_{Pairlist}$ -Sync $_{ptick}$ -defs)
show $\langle t \in ?S_B \implies \exists t_0 \text{ } t_1. t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1 \rangle$
for t
by (\mathcal{R}_{nd} -subset-method defs: combine $_{Pairlist}$ -Sync $_{ptick}$ -defs)
qed

lemma \mathcal{R}_d -combine $_{dPair}$ -Sync $_{ptick}$ -subset:
 $\langle \mathcal{R}_d \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_d A_0 s_0 \times \mathcal{R}_d A_1 s_1 \rangle$ (is $\langle ?S_A \subseteq - \rangle$)
and \mathcal{R}_{nd} -combine $_{ndPair}$ -Sync $_{ptick}$ -subset:
 $\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ } nd \otimes [E] \checkmark_{Pair} B_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_{nd} B_0 s_0 \times \mathcal{R}_{nd} B_1 s_1 \rangle$ (is $\langle ?S_B \subseteq - \rangle$)
proof –
have $\langle t \in ?S_A \implies fst \text{ } t \in \mathcal{R}_d A_0 s_0 \wedge snd \text{ } t \in \mathcal{R}_d A_1 s_1 \rangle$ **for** t
by (\mathcal{R}_d -subset-method defs: combine $_{Pair}$ -Sync $_{ptick}$ -defs)
thus $\langle ?S_A \subseteq \mathcal{R}_d A_0 s_0 \times \mathcal{R}_d A_1 s_1 \rangle$ **by force**
next
have $\langle t \in ?S_B \implies fst \text{ } t \in \mathcal{R}_{nd} B_0 s_0 \wedge snd \text{ } t \in \mathcal{R}_{nd} B_1 s_1 \rangle$ **for** t
by (\mathcal{R}_{nd} -subset-method defs: combine $_{Pair}$ -Sync $_{ptick}$ -defs)
thus $\langle ?S_B \subseteq \mathcal{R}_{nd} B_0 s_0 \times \mathcal{R}_{nd} B_1 s_1 \rangle$ **by force**
qed

lemma \mathcal{R}_d -combine $_{dRlist}$ -Sync $_{ptick}$ -subset:
 $\langle \mathcal{R}_d \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \text{ } \sigma t. t_0 \in \mathcal{R}_d A_0 s_0 \wedge \sigma t \in \mathcal{R}_d A_1 \sigma s\} \rangle$ (is $\langle ?S_A \subseteq - \rangle$)
and \mathcal{R}_{nd} -combine $_{ndRlist}$ -Sync $_{ptick}$ -subset:
 $\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ } nd \otimes [E] \checkmark_{Rlist} B_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \text{ } \sigma t. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge \sigma t \in \mathcal{R}_{nd} B_1 \sigma s\} \rangle$ (is $\langle ?S_B \subseteq - \rangle$)
proof safe
show $\langle t \in ?S_A \implies \exists t_0 \text{ } \sigma t. t = t_0 \# \sigma t \wedge t_0 \in \mathcal{R}_d A_0 s_0 \wedge \sigma t \in \mathcal{R}_d A_1 \sigma s \rangle$ **for** t
by (\mathcal{R}_d -subset-method defs: combine $_{Rlist}$ -Sync $_{ptick}$ -defs)
next
show $\langle t \in ?S_B \implies \exists t_0 \text{ } \sigma t. t = t_0 \# \sigma t \wedge t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge \sigma t \in \mathcal{R}_{nd} B_1 \sigma s \rangle$
for t
by (\mathcal{R}_{nd} -subset-method defs: combine $_{Rlist}$ -Sync $_{ptick}$ -defs)
qed

7.5 Normalization

lemma ω -combine $_{Pairlist}$ -Sync $_{ptick}$ -behaviour:
 $\langle \omega \langle \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle \rangle_{d \rightarrow nd} [s_0, s_1] = \omega \langle \langle \langle A_0 \rangle \rangle_{d \rightarrow nd} \langle \langle B_0 \text{ } nd \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_{d \rightarrow nd} \rangle [s_0, s_1] \rangle$

by (*simp add: combinePairlist-Syncptick-defs det-ndet-conv-defs option.case-eq-if*)

lemma ω -combinePair-Syncptick-behaviour:

$\langle \omega \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_{d \mapsto nd} (s_0, s_1) = \omega \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pair} \langle A_1 \rangle_{d \mapsto nd} \rangle (s_0, s_1) \rangle$

by (*simp add: combinePair-Syncptick-defs det-ndet-conv-defs option.case-eq-if*)

lemma ω -combineListLenL-Syncptick-behaviour:

$\langle \omega \langle \langle A_0 \ d \otimes [len_0, E] \checkmark_{ListLenL} A_1 \rangle \rangle_{d \mapsto nd} (\sigma s_0 \ @ \ \sigma s_1) = \omega \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [len_0, E] \checkmark_{ListLenL} \langle A_1 \rangle_{d \mapsto nd} \rangle (\sigma s_0 \ @ \ \sigma s_1) \rangle$

by (*simp add: combineListLenL-Syncptick-defs det-ndet-conv-defs option.case-eq-if*)

lemma ω -combineRlist-Syncptick-behaviour:

$\langle \omega \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_{d \mapsto nd} (s_0 \ # \ \sigma s_1) = \omega \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Rlist} \langle A_1 \rangle_{d \mapsto nd} \rangle (s_0 \ # \ \sigma s_1) \rangle$

by (*simp add: combineRlist-Syncptick-defs det-ndet-conv-defs option.case-eq-if*)

lemma τ -combinePairlist-Syncptick-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ s_0 \ \cap \ \varepsilon \ A_1 \ s_1 \ \subseteq \ E \implies$

$\tau \langle \langle A_0 \ d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_{d \mapsto nd} [s_0, s_1] \ e = \tau \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pairlist} \langle A_1 \rangle_{d \mapsto nd} \rangle [s_0, s_1] \ e \rangle$

by (*auto simp add: combinePairlist-Syncptick-defs det-ndet-conv-defs option.case-eq-if \varepsilon-simps*)

lemma τ -combinePair-Syncptick-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ s_0 \ \cap \ \varepsilon \ A_1 \ s_1 \ \subseteq \ E \implies$

$\tau \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_{d \mapsto nd} (s_0, s_1) \ e = \tau \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pair} \langle A_1 \rangle_{d \mapsto nd} \rangle (s_0, s_1) \ e \rangle$

by (*auto simp add: combinePair-Syncptick-defs det-ndet-conv-defs option.case-eq-if \varepsilon-simps*)

lemma τ -combineListLenL-Syncptick-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ \sigma s_0 \ \cap \ \varepsilon \ A_1 \ \sigma s_1 \ \subseteq \ E \implies \text{length } \sigma s_0 = \text{len}_0 \implies$

$\tau \langle \langle A_0 \ d \otimes [len_0, E] \checkmark_{ListLenL} A_1 \rangle \rangle_{d \mapsto nd} (\sigma s_0 \ @ \ \sigma s_1) \ e = \tau \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [len_0, E] \checkmark_{ListLenL} \langle A_1 \rangle_{d \mapsto nd} \rangle (\sigma s_0 \ @ \ \sigma s_1) \ e \rangle$

by (*auto simp add: combineListLenL-Syncptick-defs det-ndet-conv-defs option.case-eq-if \varepsilon-simps*)

lemma τ -combineRlist-Syncptick-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ s_0 \ \cap \ \varepsilon \ A_1 \ \sigma s_1 \ \subseteq \ E \implies$

$\tau \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_{d \mapsto nd} (s_0 \ # \ \sigma s_1) \ e = \tau \langle \langle A_0 \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Rlist} \langle A_1 \rangle_{d \mapsto nd} \rangle (s_0 \ # \ \sigma s_1) \ e \rangle$

by (*auto simp add: combineRlist-Syncptick-defs det-ndet-conv-defs option.case-eq-if \varepsilon-simps*)

lemma P_{SKIPS} -combinePairlist-Syncptick-behaviour-when-indep:

$\langle P_{SKIPS} \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [s_0, s_1] = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes [E] \checkmark_{Pairlist} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} [s_0, s_1] \rangle$
if $\langle indep\text{-enabl } A_0 \text{ } s_0 \text{ } E \text{ } A_1 \text{ } s_1 \rangle$
by ($P_{SKIPS}\text{-when-indep-method } R\text{-}d\text{-subset: } \mathcal{R}_d\text{-combine}_{dPairlist}\text{-Sync}_{ptick}\text{-subset, simp-all}$)
 (*metis* $\tau\text{-combine}_{Pairlist}\text{-Sync}_{ptick}\text{-behaviour-when-indep indep-enablD}$ that,
metis $\omega\text{-combine}_{Pairlist}\text{-Sync}_{ptick}\text{-behaviour}$)

lemma $P\text{-combine}_{Pairlist}\text{-Sync}_{ptick}\text{-behaviour-when-indep}$:
 $\langle P \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [s_0, s_1] = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes [E] \checkmark_{Pairlist} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} [s_0, s_1] \rangle$
if $\langle indep\text{-enabl } A_0 \text{ } s_0 \text{ } E \text{ } A_1 \text{ } s_1 \rangle$
by ($P\text{-when-indep-method } R\text{-}d\text{-subset: } \mathcal{R}_d\text{-combine}_{dPairlist}\text{-Sync}_{ptick}\text{-subset, simp-all}$)
 (*metis* $\tau\text{-combine}_{Pairlist}\text{-Sync}_{ptick}\text{-behaviour-when-indep indep-enablD}$ that)

lemma $P_{SKIPS}\text{-combine}_{Pair}\text{-Sync}_{ptick}\text{-behaviour-when-indep}$:
 $\langle P_{SKIPS} \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_d (s_0, s_1) = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes [E] \checkmark_{Pair} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0, s_1) \rangle$
if $\langle indep\text{-enabl } A_0 \text{ } s_0 \text{ } E \text{ } A_1 \text{ } s_1 \rangle$
by ($P_{SKIPS}\text{-when-indep-method } R\text{-}d\text{-subset: } \mathcal{R}_d\text{-combine}_{dPair}\text{-Sync}_{ptick}\text{-subset, all } \langle elim \text{ } SigmaE \rangle$)
 (*metis* $\tau\text{-combine}_{Pair}\text{-Sync}_{ptick}\text{-behaviour-when-indep indep-enablD}$ that,
auto simp add: $\omega\text{-combine}_{Pair}\text{-Sync}_{ptick}\text{-behaviour option.case-eq-if}$)

lemma $P\text{-combine}_{Pair}\text{-Sync}_{ptick}\text{-behaviour-when-indep}$:
 $\langle P \langle \langle A_0 \text{ } d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_d (s_0, s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes [E] \checkmark_{Pair} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0, s_1) \rangle$
if $\langle indep\text{-enabl } A_0 \text{ } s_0 \text{ } E \text{ } A_1 \text{ } s_1 \rangle$
by ($P\text{-when-indep-method } R\text{-}d\text{-subset: } \mathcal{R}_d\text{-combine}_{dPair}\text{-Sync}_{ptick}\text{-subset, elim } SigmaE$)
 (*metis* $\tau\text{-combine}_{Pair}\text{-Sync}_{ptick}\text{-behaviour-when-indep indep-enablD}$ that)

lemma $P_{SKIPS}\text{-combine}_{ListslenL}\text{-Sync}_{ptick}\text{-behaviour-when-indep}$:
 $\langle P_{SKIPS} \langle \langle A_0 \text{ } d \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_d (\sigma s_0 \text{ } @ \text{ } \sigma s_1) = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes [len_0, E] \checkmark_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 \text{ } @ \text{ } \sigma s_1) \rangle$
if $\langle indep\text{-enabl } A_0 \text{ } \sigma s_0 \text{ } E \text{ } A_1 \text{ } \sigma s_1 \rangle$ **and** $\langle \bigwedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma t_0 = len_0 \rangle$
by ($P_{SKIPS}\text{-when-indep-method } R\text{-}d\text{-subset: } \mathcal{R}_d\text{-combine}_{dListslenL}\text{-Sync}_{ptick}\text{-subset, simp-all add: that(2)}$)
 (*metis* $\tau\text{-combine}_{ListslenL}\text{-Sync}_{ptick}\text{-behaviour-when-indep indep-enablD}$ that,
metis $\omega\text{-combine}_{ListslenL}\text{-Sync}_{ptick}\text{-behaviour}$)

lemma $P\text{-combine}_{ListslenL}\text{-Sync}_{ptick}\text{-behaviour-when-indep}$:
 $\langle P \langle \langle A_0 \text{ } d \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_d (\sigma s_0 \text{ } @ \text{ } \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes [len_0, E] \checkmark_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 \text{ } @ \text{ } \sigma s_1) \rangle$
if $\langle indep\text{-enabl } A_0 \text{ } \sigma s_0 \text{ } E \text{ } A_1 \text{ } \sigma s_1 \rangle$ **and** $\langle \bigwedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma t_0 = len_0 \rangle$

by (*P-when-indep-method R-d-subset: \mathcal{R}_d -combine $_{dListslenL}$ -Sync $_{ptick}$ -subset*,
simp-all add: that(2))

(*metis τ -combine $_{ListslenL}$ -Sync $_{ptick}$ -behaviour-when-indep indep-enablD that*)

lemma *P_{SKIP}S-combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour-when-indep*:

$\langle P_{SKIP}S \langle \langle A_0 \text{ }_d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_d (s_0 \# \sigma s_1) = P_{SKIP}S \langle \langle \langle A_0 \rangle_{d \rightarrow nd} \text{ }_{nd} \otimes [E] \checkmark_{Rlist} \langle A_1 \rangle_{d \rightarrow nd} \rangle \rangle_{nd} (s_0 \# \sigma s_1) \rangle$

if $\langle \textit{indep-enabl } A_0 \text{ }_{s_0} E A_1 \text{ }_{\sigma s_1} \rangle$

by (*P_{SKIP}S-when-indep-method R-d-subset: \mathcal{R}_d -combine $_{dRlist}$ -Sync $_{ptick}$ -subset*,
simp-all)

(*metis τ -combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour-when-indep indep-enablD that*,

metis ω -combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour)

lemma *P-combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour-when-indep*:

$\langle P \langle \langle A_0 \text{ }_d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_d (s_0 \# \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \rightarrow nd} \text{ }_{nd} \otimes [E] \checkmark_{Rlist} \langle A_1 \rangle_{d \rightarrow nd} \rangle \rangle_{nd} (s_0 \# \sigma s_1) \rangle$

if $\langle \textit{indep-enabl } A_0 \text{ }_{s_0} E A_1 \text{ }_{\sigma s_1} \rangle$

by (*P-when-indep-method R-d-subset: \mathcal{R}_d -combine $_{dRlist}$ -Sync $_{ptick}$ -subset*, *simp*)

(*metis τ -combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour-when-indep indep-enablD that*)

Chapter 8

Compactification of Synchronization Product Generalized

8.1 Iterated Combine

8.1.1 Definitions

fun *iterated-combine_d-Sync_{ptick}* :: ⟨'e set ⇒ ('σ, 'e, 'r) A_d list ⇒ ('σ list, 'e, 'r list) A_d⟩ (⟨⟨_d⊗ [-]✓ -⟩⟩ [0, 0])
where ⟨⟨_d⊗ [E]✓ []⟩⟩ = (τ = λσ s a. ◇, ω = λσ s. ◇)⟨
| ⟨⟨_d⊗ [E]✓ [A₀]⟩⟩ = _d⟨A₀⟩_{single→list}⟨
| ⟨⟨_d⊗ [E]✓ A₀ # A₁ # A_s⟩⟩ = ⟨A₀ _d⊗ [E]✓ Rlist ⟨_d⊗ [E]✓ A₁ # A_s⟩⟩⟩

fun *iterated-combine_{nd}-Sync_{ptick}* :: ⟨'e set ⇒ ('σ, 'e, 'r) A_{nd} list ⇒ ('σ list, 'e, 'r list) A_{nd}⟩ (⟨⟨_{nd}⊗ [-]✓ -⟩⟩ [0, 0])
where ⟨⟨_{nd}⊗ [E]✓ []⟩⟩ = (τ = λσ s a. {}, ω = λσ s. {})⟨
| ⟨⟨_{nd}⊗ [E]✓ [A₀]⟩⟩ = _{nd}⟨A₀⟩_{single→list}⟨
| ⟨⟨_{nd}⊗ [E]✓ A₀ # A₁ # A_s⟩⟩ = ⟨A₀ _{nd}⊗ [E]✓ Rlist ⟨_{nd}⊗ [E]✓ A₁ # A_s⟩⟩⟩

lemma *iterated-combine_d-Sync_{ptick}-simps-bis*: ⟨As ≠ [] ⇒ ⟨_d⊗ [E]✓ A₀ # A_s⟩ = ⟨A₀ _d⊗ [E]✓ Rlist ⟨_d⊗ [E]✓ A_s⟩⟩⟩
and *iterated-combine_{nd}-Sync_{ptick}-simps-bis*: ⟨Bs ≠ [] ⇒ ⟨_{nd}⊗ [E]✓ B₀ # B_s⟩ = ⟨B₀ _{nd}⊗ [E]✓ Rlist ⟨_{nd}⊗ [E]✓ B_s⟩⟩⟩
by (induct As, simp-all) (induct Bs, simp-all)

8.1.2 First Results

lemma *τ-iterated-combine-Sync_{ptick}*:
⟨τ ⟨_d⊗ [E]✓ As⟩ = τ ⟨_d⊗ [E] As⟩⟩ ⟨τ ⟨_{nd}⊗ [E]✓ Bs⟩ = τ ⟨_{nd}⊗ [E] Bs⟩⟩
by (intro ext, induct rule: induct-list012;
simp add: σ-σs-conv-defs single-list-conv-defs
combine_{Rlist}-Sync-defs combine_{Rlist}-Sync_{ptick}-defs ε-simps)+

corollary ε -iterated-combine-Sync_{ptick}:

$$\begin{aligned} &\langle \varepsilon \langle d \otimes [E] \checkmark As \rangle \sigma s = \varepsilon \langle d \otimes [E] As \rangle \sigma s \rangle \\ &\langle \varepsilon \langle nd \otimes [E] \checkmark Bs \rangle \sigma s = \varepsilon \langle nd \otimes [E] Bs \rangle \sigma s \rangle \\ &\text{by (simp-all add: } \varepsilon\text{-simps } \tau\text{-iterated-combine-Sync}_{ptick}\text{)} \end{aligned}$$

corollary \mathcal{R} -iterated-combine-Sync_{ptick}:

$$\begin{aligned} &\langle \mathcal{R}_d \langle d \otimes [E] \checkmark As \rangle = \mathcal{R}_d \langle d \otimes [E] As \rangle \rangle \langle \mathcal{R}_{nd} \langle nd \otimes [E] \checkmark Bs \rangle = \mathcal{R}_{nd} \langle nd \otimes [E] Bs \rangle \rangle \\ &\text{by (intro ext same-}\tau\text{-implies-same-}\mathcal{R}_d\text{ same-}\tau\text{-implies-same-}\mathcal{R}_{nd}\text{,} \\ &\quad \text{simp add: } \tau\text{-iterated-combine-Sync}_{ptick}\text{)}+ \end{aligned}$$

lemma combine_{ListLenL}-Sync_{ptick}-combine_{Rlist}-Sync_{ptick}-eq:

$$\begin{aligned} &\langle \varepsilon \langle d \langle A_0 \rangle_{\text{singl} \rightarrow \text{list}} d \otimes [1, E] \checkmark_{ListLenL} A_1 \rangle \sigma s = \varepsilon \langle A_0 d \otimes [E] \checkmark_{Rlist} A_1 \rangle \sigma s \rangle \\ &\langle \tau \langle d \langle A_0 \rangle_{\text{singl} \rightarrow \text{list}} d \otimes [1, E] \checkmark_{ListLenL} A_1 \rangle (s_0 \# \sigma s) e = \tau \langle A_0 d \otimes [E] \checkmark_{Rlist} A_1 \rangle (s_0 \# \sigma s) e \rangle \\ &\langle \varepsilon \langle nd \langle B_0 \rangle_{\text{singl} \rightarrow \text{list}} nd \otimes [1, E] \checkmark_{ListLenL} B_1 \rangle \sigma s = \varepsilon \langle B_0 nd \otimes [E] \checkmark_{Rlist} B_1 \rangle \sigma s \rangle \\ &\langle \tau \langle nd \langle B_0 \rangle_{\text{singl} \rightarrow \text{list}} nd \otimes [1, E] \checkmark_{ListLenL} B_1 \rangle (s_0 \# \sigma s) e = \tau \langle B_0 nd \otimes [E] \checkmark_{Rlist} B_1 \rangle (s_0 \# \sigma s) e \rangle \\ &\text{by (simp-all add: } \varepsilon\text{-combine}_{ListLenL}\text{-Sync}_{ptick}\text{ } \varepsilon\text{-combine}_{Rlist}\text{-Sync}_{ptick}\text{ drop-Suc} \\ &\quad \text{combine-Sync-}\varepsilon\text{-def,} \\ &\quad \text{auto simp add: combine}_{ListLenL}\text{-Sync}_{ptick}\text{-defs combine}_{Rlist}\text{-Sync}_{ptick}\text{-defs} \\ &\quad \text{singl-list-conv-defs } \varepsilon\text{-simps)} \\ &\quad \text{(metis append-Cons append-Nil)} \end{aligned}$$

lemma combine_{Pairlist}-Sync_{ptick}-and-iterated-combine_{nd}-Sync_{ptick}-eq:

$$\begin{aligned} &\langle \varepsilon \langle A_0 d \otimes [E] \checkmark_{Pairlist} A_1 \rangle [s_0, s_1] = \varepsilon \langle d \otimes [E] \checkmark [A_0, A_1] \rangle [s_0, s_1] \rangle \\ &\langle \tau \langle A_0 d \otimes [E] \checkmark_{Pairlist} A_1 \rangle [s_0, s_1] e = \tau \langle d \otimes [E] \checkmark [A_0, A_1] \rangle [s_0, s_1] e \rangle \\ &\langle \varepsilon \langle B_0 nd \otimes [E] \checkmark_{Pairlist} B_1 \rangle [s_0, s_1] = \varepsilon \langle nd \otimes [E] \checkmark [B_0, B_1] \rangle [s_0, s_1] \rangle \\ &\langle \tau \langle B_0 nd \otimes [E] \checkmark_{Pairlist} B_1 \rangle [s_0, s_1] e = \tau \langle nd \otimes [E] \checkmark [B_0, B_1] \rangle [s_0, s_1] e \rangle \\ &\text{by (simp-all add: } \varepsilon\text{-combine}_{Pairlist}\text{-Sync}_{ptick}\text{ } \varepsilon\text{-combine}_{Rlist}\text{-Sync}_{ptick}\text{)} \\ &\quad \text{(auto simp add: combine}_{Pairlist}\text{-Sync}_{ptick}\text{-defs combine}_{Rlist}\text{-Sync}_{ptick}\text{-defs} \\ &\quad \text{singl-list-conv-defs} \\ &\quad \text{option.case-eq-if } \varepsilon\text{-simps combine-Sync-}\varepsilon\text{-def)} \end{aligned}$$

lemmas combine_{Pairlist}-Sync_{ptick}-and-combine_{Rlist}-Sync_{ptick}-eq =
combine_{Pairlist}-Sync_{ptick}-and-iterated-combine_{nd}-Sync_{ptick}-eq[simplified]

8.1.3 Transmission of Properties

lemma finite-trans-transmission-to-iterated-combine_{nd}-Sync_{ptick}:

$$\begin{aligned} &\langle (\bigwedge A. A \in \text{set } As \implies \text{finite-trans } A) \implies \text{finite-trans } \langle nd \otimes [E] \checkmark As \rangle \rangle \\ &\text{by (induct } As \text{ rule: induct-list012)} \end{aligned}$$

(*auto simp add: singl-list-conv-defs combine_{Rlist-Sync_{ptick}}-defs finite-trans-def finite-image-set2*)

lemma ϱ -disjoint- ε -transmission-to-iterated-combine _{d -Sync_{ptick}}:
 $\langle (\bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A) \implies \varrho\text{-disjoint-}\varepsilon \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} As \rrbracket \rangle$
by (*induct As rule: induct-list012*)
(*simp-all add: ϱ -combine_{Rlist-Sync_{ptick}} ε -combine_{Rlist-Sync_{ptick}} ϱ -disjoint- ε -def combine-Sync- ε -def*)

lemma ϱ -disjoint- ε -transmission-to-iterated-combine _{nd -Sync_{ptick}}:
 $\langle \forall B \in \text{set } Bs. \varrho\text{-disjoint-}\varepsilon B \implies \varrho\text{-disjoint-}\varepsilon \llbracket_{nd} \otimes \llbracket E \rrbracket_{\checkmark} Bs \rrbracket \rangle$
by (*induct Bs rule: induct-list012*)
(*simp-all add: ϱ -combine_{Rlist-Sync_{ptick}} ε -combine_{Rlist-Sync_{ptick}} ϱ -disjoint- ε -def combine-Sync- ε -def*)

lemma same-length-indep-transmission-to-iterated-combine _{d -Sync_{ptick}}:
 $\langle \text{indep-enabl } A_0 \ s_0 \ E \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} As \rrbracket \ \sigma s \rangle$
if $\langle \text{length } \sigma s = \text{length } As \rangle$
 $\langle \bigwedge i \ j. \llbracket i \leq \text{length } As; j \leq \text{length } As; i \neq j \rrbracket \implies$
 $\text{indep-enabl } ((A_0 \# As) ! i) ((s_0 \# \sigma s) ! i) E ((A_0 \# As) ! j) ((s_0 \# \sigma s) ! j) \rangle$
using same-length-indep-transmission-to-iterated-combine _{d -Sync}[*OF that*]
by (*simp add: indep-enabl-def ε -iterated-combine-Sync_{ptick} τ -iterated-combine-Sync_{ptick} \mathcal{R} -iterated-combine-Sync_{ptick} that(1)*)

lemma ω -iterated-combine _{d -Sync_{ptick}} :
 $\langle \text{length } \sigma s = \text{length } As \implies$
 $\omega \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} As \rrbracket \ \sigma s = (\text{if } As = [] \text{ then } \diamond \text{ else those } (\text{map2 } \omega \ As \ \sigma s)) \rangle$
by (*induct σs As rule: induct-2-lists012*)
(*simp-all add: singl-list-conv-defs combine_{Rlist-Sync_{ptick}}-defs split: option.split*)

lemma ω -iterated-combine _{nd -Sync_{ptick}} :
 $\langle \text{length } \sigma s = \text{length } As \implies$
 $\omega \llbracket_{nd} \otimes \llbracket E \rrbracket_{\checkmark} As \rrbracket \ \sigma s =$
 $(\text{if } As = [] \text{ then } \{\} \text{ else } \{rs. \text{length } rs = \text{length } As \wedge (\forall i < \text{length } As. rs ! i \in \omega$
 $(As ! i) (\sigma s ! i))\}) \rangle$
proof (*induct σs As rule: induct-2-lists012*)
case Nil show ?case by simp
next
case (single $\sigma 1 \ A1$)
from length-Suc-conv show ?case
by (*auto simp add: singl-list-conv-defs*)
next
case (Cons $\sigma 1 \ \sigma 2 \ \sigma s \ A1 \ A2 \ As$)
show ?case (is $\langle - = ?rhs \ \sigma 1 \ \sigma 2 \ \sigma s \ A1 \ A2 \ As \rangle$)
proof (*intro subset-antisym subsetI*)

fix rs **assume** $\langle rs \in \omega \llbracket_{nd} \otimes [E] \checkmark A1 \# A2 \# As \rrbracket (\sigma1 \# \sigma2 \# \sigma s) \rangle$
then obtain $r1 \ rs'$ **where** $\langle rs = r1 \# rs' \rangle \langle r1 \in \omega A1 \ \sigma1 \rangle$
 $\langle rs' \in \omega \llbracket_{nd} \otimes [E] \checkmark A2 \# As \rrbracket (\sigma2 \# \sigma s) \rangle$
by (*auto simp add: combine_{RList-Sync_{ptick}-defs}*)
from *this*(3) *Cons.hyps*(3) **obtain** $r2 \ rs''$
where $\langle rs' = r2 \# rs'' \rangle \langle length \ rs'' = length \ As \rangle$
 $\langle \forall i < Suc \ (length \ As). \ (r2 \# rs'') ! i \in \omega \ ((A2 \# As) ! i) \ ((\sigma2 \# \sigma s) ! i) \rangle$
by *simp* (*metis* (*no-types*, *lifting*) *length-Suc-conv*)
with $\langle r1 \in \omega A1 \ \sigma1 \rangle$ **show** $\langle rs \in ?rhs \ \sigma1 \ \sigma2 \ \sigma s \ A1 \ A2 \ As \rangle$
by (*auto simp add: \langle rs = r1 \# rs' \rangle less-Suc-eq-0-disj*)
next
from *Cons.hyps*(3)
show $\langle rs \in ?rhs \ \sigma1 \ \sigma2 \ \sigma s \ A1 \ A2 \ As \implies$
 $rs \in \omega \llbracket_{nd} \otimes [E] \checkmark A1 \# A2 \# As \rrbracket (\sigma1 \# \sigma2 \# \sigma s) \rangle$ **for** rs
by (*cases* rs ; *cases* $\langle tl \ rs \rangle$, *simp-all add: combine_{RList-Sync_{ptick}-defs}*) *auto*
qed
qed

8.1.4 Normalization

lemma ω -*iterated-combine_{nd-Sync_{ptick}-det-ndet-conv}*:
 $\langle length \ \sigma s = length \ As \implies$
 $\omega \llbracket_{nd} \otimes [E] \checkmark map \ (\lambda A. \ \langle A \rangle_{d \mapsto nd}) \ As \rrbracket \ \sigma s = \omega \llbracket_{d} \otimes [E] \checkmark As \rrbracket_{d \mapsto nd} \ \sigma s \rangle$
proof (*induct* $\sigma s \ As$ *rule: induct-2-lists012*)
case *Nil*
show $?case$ **by** (*simp add: base-trans-det-ndet-conv*(1))
next
case (*single* $\sigma1 \ A1$)
show $?case$ **by** (*simp add: from-det-to-ndet-singl-list-conv-commute*)
next
case (*Cons* $\sigma1 \ \sigma2 \ \sigma s \ A1 \ A2 \ As$)
thus $?case$
by (*auto simp add: det-ndet-conv-defs combine_{RList-Sync_{ptick}-defs}* *split: option.split*)
qed

lemma τ -*iterated-combine_{nd-Sync_{ptick}-behaviour-when-indep}*:
 $\langle length \ \sigma s = length \ As \implies$
 $(\bigwedge i \ j. \ [i < length \ As; j < length \ As; i \neq j] \implies$
 $\implies indep-enabl \ (As ! i) \ (\sigma s ! i) \ E \ (As ! j) \ (\sigma s ! j)) \implies$
 $\tau \llbracket_{d} \otimes [E] \checkmark As \rrbracket_{d \mapsto nd} \ \sigma s \ e = \tau \llbracket_{nd} \otimes [E] \checkmark map \ (\lambda A. \ \langle A \rangle_{d \mapsto nd}) \ As \rrbracket \ \sigma s \ e \rangle$
proof (*induct* $\sigma s \ As$ *rule: induct-2-lists012*)
case *Nil*
show $?case$ **by** *simp*
next
case (*single* $\sigma1 \ A1$)
show $?case$ **by** (*simp add: from-det-to-ndet-singl-list-conv-commute*(1))
next
case (*Cons* $\sigma1 \ \sigma2 \ \sigma s \ A1 \ A2 \ As$)

have * : $\langle \tau \ll \langle \langle d \otimes [E] \checkmark A2 \# As \rangle \rangle_{d \hookrightarrow nd} (\sigma 2 \# \sigma s) e = \tau \ll \langle \langle nd \otimes [E] \checkmark map (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) (A2 \# As) \rangle \rangle (\sigma 2 \# \sigma s) e \rangle$
proof (rule *Cons.hyps*(3))
show $\langle [i < \text{length } (A2 \# As); j < \text{length } (A2 \# As); i \neq j] \implies \text{indep-enabl } ((A2 \# As) ! i) ((\sigma 2 \# \sigma s) ! i) E ((A2 \# As) ! j) ((\sigma 2 \# \sigma s) ! j) \rangle$ **for** $i j$
using *Cons.prem*s[*of* $\langle \text{Suc } i \rangle \langle \text{Suc } j \rangle$] **by** *simp*
qed
have $\langle \tau \ll \langle \langle d \otimes [E] \checkmark A1 \# A2 \# As \rangle \rangle_{d \hookrightarrow nd} (\sigma 1 \# \sigma 2 \# \sigma s) e = \tau \ll \langle \langle A1 \rangle_{d \hookrightarrow nd} nd \otimes [E] \checkmark Rlist \ll \langle \langle d \otimes [E] \checkmark A2 \# As \rangle \rangle_{d \hookrightarrow nd} \rangle (\sigma 1 \# \sigma 2 \# \sigma s) e \rangle$
proof (*subst iterated-combine_d-Sync_{ptick}.sims*(3), rule τ -*combine_{Rlist}-Sync_{ptick}-behaviour-when-indep*)
show $\langle \varepsilon A1 \sigma 1 \cap \varepsilon \ll \langle \langle d \otimes [E] \checkmark A2 \# As \rangle \rangle (\sigma 2 \# \sigma s) \subseteq E \rangle$
proof (rule *indep-enablD*[*OF* - \mathcal{R}_d .*init* \mathcal{R}_d .*init*])
show $\langle \text{indep-enabl } A1 \sigma 1 E \ll \langle \langle d \otimes [E] \checkmark A2 \# As \rangle \rangle (\sigma 2 \# \sigma s) \rangle$
by (*simp add: Cons.hyps*(1) *Cons.prem*s *order-le-less-trans* *same-length-indep-transmission-to-iterated-combine_d-Sync_{ptick}*)
qed
qed
also have $\langle \dots = \tau \ll \langle \langle nd \otimes [E] \checkmark map (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) (A1 \# A2 \# As) \rangle \rangle (\sigma 1 \# \sigma 2 \# \sigma s) e \rangle$
by (*use* * **in** $\langle \text{simp add: combine_{Rlist}-Sync_{ptick}-defs } \varepsilon\text{-sims} \rangle$)
(*metis empty-from-det-to-ndet-is-None-trans option.exhaust*)
finally show ?*case* .
qed

lemma *P_{SKIPS}-iterated-combine_{nd}-Sync_{ptick}-behaviour-when-indep*:

assumes *same-length*: $\langle \text{length } \sigma s = \text{length } As \rangle$
and *indep*: $\langle \bigwedge i j. [i < \text{length } As; j < \text{length } As; i \neq j] \implies \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$
shows $\langle P_{SKIPS} \ll \langle \langle \langle d \otimes [E] \checkmark As \rangle \rangle_{d \hookrightarrow nd} \sigma s = P_{SKIPS} \ll \langle \langle nd \otimes [E] \checkmark map (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) As \rangle \rangle_{nd} \sigma s \rangle$
proof (*fold P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*, rule *P_{SKIPS}-nd-eqI-strong-id*)
show $\langle \sigma s' \in \mathcal{R}_{nd} \ll \langle \langle d \otimes [E] \checkmark As \rangle \rangle_{d \hookrightarrow nd} \sigma s \implies \tau \ll \langle \langle nd \otimes [E] \checkmark map (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) As \rangle \rangle \sigma s' e = \tau \ll \langle \langle d \otimes [E] \checkmark As \rangle \rangle_{d \hookrightarrow nd} \sigma s' e \rangle$
for $\sigma s' e$
proof (rule τ -*iterated-combine_{nd}-Sync_{ptick}-behaviour-when-indep*[*symmetric*])
show $\langle \sigma s' \in \mathcal{R}_{nd} \ll \langle \langle d \otimes [E] \checkmark As \rangle \rangle_{d \hookrightarrow nd} \sigma s \implies \text{length } \sigma s' = \text{length } As \rangle$
by (*metis* \mathcal{R}_{nd} -*from-det-to-ndet* \mathcal{R} -*iterated-combine-Sync_{ptick}*(1) *same-length same-length- \mathcal{R}_d -iterated-combine_d-Sync-description*)
next
show $\langle [\sigma s' \in \mathcal{R}_{nd} \ll \langle \langle d \otimes [E] \checkmark As \rangle \rangle_{d \hookrightarrow nd} \sigma s; i < \text{length } As; j < \text{length } As; i \neq j] \implies \text{indep-enabl } (As ! i) (\sigma s' ! i) E (As ! j) (\sigma s' ! j) \rangle$ **for** $i j$
by (*unfold* \mathcal{R}_{nd} -*from-det-to-ndet* \mathcal{R} -*iterated-combine-Sync_{ptick}*, *drule same-length- \mathcal{R}_d -iterated-combine_d-Sync-description*[*OF* *same-length*])
(*meson* \mathcal{R}_d -*trans indep-enabl-def indep*)

qed
next
show $\langle \sigma s' \in \mathcal{R}_{nd} \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_{d \hookrightarrow nd} \sigma s \rangle \implies$
 $\omega \langle \langle \langle nd \otimes [E]_{\checkmark} \text{map} (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) As \rangle \rangle \sigma s' = \omega \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_{d \hookrightarrow nd} \sigma s' \rangle$
for $\sigma s'$
by (*metis* \mathcal{R}_{nd} -*from-det-to-ndet* \mathcal{R} -*iterated-combine-Sync*_{ptick}(1)
*ω-iterated-combine*_{nd}-*Sync*_{ptick}-*det-ndet-conv* *same-length*
*same-length- \mathcal{R}_d -iterated-combine*_d-*Sync-description*)
qed

lemma *P-d-iterated-combine*_{nd}-*Sync*_{ptick}-*behaviour-when-indep*:
assumes *same-length*: $\langle \text{length } \sigma s = \text{length } As \rangle$
and *indep*: $\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$
shows $\langle P \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_d \sigma s = P \langle \langle \langle nd \otimes [E]_{\checkmark} \text{map} (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) As \rangle \rangle_{nd} \sigma s \rangle$
proof (*fold* *P-nd-from-det-to-ndet-is-P-d*, *rule* *P-nd-eqI-strong-id*)
show $\langle \sigma s' \in \mathcal{R}_{nd} \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_{d \hookrightarrow nd} \sigma s \rangle \implies$
 $\tau \langle \langle \langle nd \otimes [E]_{\checkmark} \text{map} (\lambda A. \langle A \rangle_{d \hookrightarrow nd}) As \rangle \rangle \sigma s' e = \tau \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_{d \hookrightarrow nd} \sigma s' e \rangle$
for $\sigma s' e$
proof (*rule* τ -*iterated-combine*_{nd}-*Sync*_{ptick}-*behaviour-when-indep*[*symmetric*])
show $\langle \sigma s' \in \mathcal{R}_{nd} \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_{d \hookrightarrow nd} \sigma s \rangle \implies \text{length } \sigma s' = \text{length } As$
by (*metis* \mathcal{R}_{nd} -*from-det-to-ndet* \mathcal{R} -*iterated-combine-Sync*_{ptick}(1) *same-length*
*same-length- \mathcal{R}_d -iterated-combine*_d-*Sync-description*)
next
show $\langle \llbracket \sigma s' \in \mathcal{R}_{nd} \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_{d \hookrightarrow nd} \sigma s; i < \text{length } As; j < \text{length } As; i \neq j \rrbracket$
 $\implies \text{indep-enabl } (As ! i) (\sigma s' ! i) E (As ! j) (\sigma s' ! j) \rangle$ **for** $i j$
by (*unfold* \mathcal{R}_{nd} -*from-det-to-ndet* \mathcal{R} -*iterated-combine-Sync*_{ptick},
drule *same-length- \mathcal{R}_d -iterated-combine*_d-*Sync-description*[*OF same-length*])
(meson \mathcal{R}_d -*trans indep-enabl-def indep*)
qed
qed

8.2 Compactification Theorems

8.2.1 Binary

Pair

theorem *P*_{SKIPS}-*nd-combine*_{Pair}-*Sync*_{ptick} :
fixes $E :: \langle 'a \text{ set} \rangle$
assumes ϱ -*disjoint-ε* : $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$
defines *A-def*: $\langle A \equiv \langle A_0 \text{ } nd \otimes [E]_{\checkmark} \text{Pair } A_1 \rangle \rangle$
defines *P-def*: $\langle P \equiv P_{SKIPS} \langle A_0 \rangle_{nd} \rangle$ **and** *Q-def*: $\langle Q \equiv P_{SKIPS} \langle A_1 \rangle_{nd} \rangle$ **and**
S-def: $\langle S \equiv P_{SKIPS} \langle A \rangle_{nd} \rangle$
shows $\langle P \sigma_0 [E]_{\checkmark} \text{Pair } Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$
proof –

let $?f = \langle P_{SKIPs}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A) (\lambda\sigma'. \text{ case } \sigma' \text{ of } (\sigma_0, \sigma_1) \Rightarrow P \sigma_0$
 $\llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1) \rangle$
note $\text{cartprod-rwrt} = \text{GlobalNdet-cartprod}[of - - \langle \lambda x y. - (x, y) \rangle, \text{simplified}]$
note $\text{Ndet-and-Sync}_{Pair} = \text{Sync}_{Pair}.\text{Sync}_{ptick}\text{-distrib-GlobalNdet-left}$
 $\text{Sync}_{Pair}.\text{Sync}_{ptick}\text{-distrib-GlobalNdet-right}$
note $\text{Mprefix-Sync}_{Pair}\text{-constant} =$
 $\text{Sync}_{Pair}.\text{SKIP-Sync}_{ptick}\text{-Mprefix } \text{Sync}_{Pair}.\text{Mprefix-Sync}_{ptick}\text{-SKIP}$
 $\text{Sync}_{Pair}.\text{STOP-Sync}_{ptick}\text{-Mprefix } \text{Sync}_{Pair}.\text{Mprefix-Sync}_{ptick}\text{-STOP}$
note $P\text{-rec} = \text{restriction-fix-eq}[OF P_{SKIPs}\text{-nd-step-constructive-bis}[of A_0], \text{folded}$
 $P_{SKIPs}\text{-nd-def } P\text{-def}, \text{ THEN fun-cong}]$
note $Q\text{-rec} = \text{restriction-fix-eq}[OF P_{SKIPs}\text{-nd-step-constructive-bis}[of A_1], \text{folded}$
 $P_{SKIPs}\text{-nd-def } Q\text{-def}, \text{ THEN fun-cong}]$
have $\omega\text{-}A : \langle \omega A (\sigma_0', \sigma_1') = \omega A_0 \sigma_0' \times \omega A_1 \sigma_1' \rangle$ **for** $\sigma_0' \sigma_1'$
by (*auto simp add: A-def combine_{Pair}-Sync_{ptick}-defs*)
have $\varepsilon\text{-}A : \langle \varepsilon A (\sigma_0', \sigma_1') = \text{combine-sets-Sync } (\varepsilon A_0 \sigma_0') E (\varepsilon A_1 \sigma_1') \rangle$ **for** σ_0'
 σ_1'
by (*simp add: A-def $\varepsilon\text{-combine}_{Pair}\text{-Sync}_{ptick}$ combine-Sync- $\varepsilon\text{-def}$*)
show $\langle P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$
proof (*rule fun-cong[$of \langle \lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 \rangle - \langle (\sigma_0, \sigma_1) \rangle, \text{simplified}$]*)
show $\langle \lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 = S \rangle$
proof (*rule restriction-fix-unique[$OF P_{SKIPs}\text{-nd-step-constructive-bis}[of A],$*
symmetric, folded $P_{SKIPs}\text{-nd-def } S\text{-def}$])
show $\langle ?f = (\lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1) \rangle$
proof (*rule ext, clarify*)
have $\varrho\text{-disjoint-}\varepsilon\text{-bis} : \langle \omega A_0 \sigma_0 \neq \{\} \implies \varepsilon A_0 \sigma_0 = \{\} \rangle$
 $\langle \omega A_1 \sigma_1 \neq \{\} \implies \varepsilon A_1 \sigma_1 = \{\} \rangle$ **for** $\sigma_0 \sigma_1$
by (*simp-all add: $\varrho\text{-simps } \varrho\text{-disjoint-}\varepsilon D \varrho\text{-disjoint-}\varepsilon$*)
show $\langle ?f (\sigma_0, \sigma_1) = P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 \rangle$ **for** $\sigma_0 \sigma_1$
proof (*cases $\langle \omega A_0 \sigma_0 = \{\} \rangle$; cases $\langle \omega A_1 \sigma_1 = \{\} \rangle$*)
assume $\langle \omega A_0 \sigma_0 = \{\} \rangle \langle \omega A_1 \sigma_1 = \{\} \rangle$
hence $P\text{-rec}' : \langle P \sigma_0 = P\text{-nd-step } (\varepsilon A_0) (\tau A_0) P \sigma_0 \rangle$
and $Q\text{-rec}' : \langle Q \sigma_1 = P\text{-nd-step } (\varepsilon A_1) (\tau A_1) Q \sigma_1 \rangle$
and $S\text{-rec}' : \langle P_{SKIPs}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A) (\lambda(\sigma_0, \sigma_1). P \sigma_0$
 $\llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1) (\sigma_0, \sigma_1) =$
 $P\text{-nd-step } (\varepsilon A) (\tau A) (\lambda(\sigma_0, \sigma_1). P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1) (\sigma_0,$
 $\sigma_1) \rangle$
by (*simp-all add: $P\text{-rec}[of } \sigma_0] Q\text{-rec}[of } \sigma_1] \omega\text{-}A$*)
show $\langle ?f (\sigma_0, \sigma_1) = P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 \rangle$
unfolding $P\text{-rec}' Q\text{-rec}' S\text{-rec}' \text{Sync}_{Pair}.\text{Mprefix-Sync}_{ptick}\text{-Mprefix-for-procomata}$
unfolding $\varepsilon\text{-}A \text{Mprefix-Un-distrib}$
by (*intro arg-cong2[$\text{where } f = \langle (\square) \rangle$] mono-Mprefix-eq, fold $P\text{-rec}' Q\text{-rec}'$,*
auto simp add: $A\text{-def Ndet-and-Sync}_{Pair}$ cartprod-rwrt
combine_{Pair}-Sync_{ptick}-defs $\varepsilon\text{-simps GlobalNdet-sets-commute}[of } \tau$
 $A_1 - -]$
simp flip: $\text{GlobalNdet-factorization-union}$
intro!: mono-GlobalNdet-eq arg-cong2[$\text{where } f = \langle (\square) \rangle$])
next
assume $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \langle \omega A_1 \sigma_1 = \{\} \rangle$
from $\varrho\text{-disjoint-}\varepsilon(1) \langle \omega A_0 \sigma_0 \neq \{\} \rangle$ **have** $\langle \varepsilon A_0 \sigma_0 = \{\} \rangle$

by (*simp add: ϱ -disjoint- ε -def ϱ -simps*)
have $\langle ?f (\sigma_0, \sigma_1) = \square b \in (\varepsilon A_1 \sigma_1 - E) \rightarrow (\prod \sigma_1' \in \tau A_1 \sigma_1 b. (SKIPS (\omega A_0 \sigma_0) \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1')) \rangle$
by (*auto simp add: ω -A ε -A $\langle \omega A_1 \sigma_1 = \{\} \rangle \langle \varepsilon A_0 \sigma_0 = \{\} \rangle$ intro!: mono-Mprefix-eq,*
auto simp add: A-def combine_{Pair}-Sync_{ptick}-defs $\langle \omega A_0 \sigma_0 \neq \{\} \rangle$
 $\langle \varepsilon A_0 \sigma_0 = \{\} \rangle$ cartprod-rwrt P-rec[of σ_0] intro!: mono-GlobalNdet-eq)
also have $\langle \dots = P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 \rangle$
by (*unfold P-rec[of σ_0] Q-rec[of σ_1]*)
(auto simp add: SKIPS-def Ndet-and-Sync_{Pair} Mprefix-Sync_{Pair}-constant
 $\langle \omega A_0 \sigma_0 \neq \{\} \rangle$
GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[of $\langle \tau A_1 - \rangle$] $\langle \omega$
 $A_1 \sigma_1 = \{\} \rangle$
intro!: mono-Mprefix-eq mono-GlobalNdet-eq)
finally show $\langle ?f (\sigma_0, \sigma_1) = \dots \rangle$.
next
assume $\langle \omega A_0 \sigma_0 = \{\} \rangle \langle \omega A_1 \sigma_1 \neq \{\} \rangle$
from ϱ -disjoint- ε (2) $\langle \omega A_1 \sigma_1 \neq \{\} \rangle$ **have** $\langle \varepsilon A_1 \sigma_1 = \{\} \rangle$
by (*simp add: ϱ -disjoint- ε -def ϱ -simps*)
have $\langle ?f (\sigma_0, \sigma_1) = \square a \in (\varepsilon A_0 \sigma_0 - E) \rightarrow (\prod \sigma_0' \in \tau A_0 \sigma_0 a. (P \sigma_0' \llbracket E \rrbracket_{\checkmark Pair} SKIPS (\omega A_1 \sigma_1))) \rangle$
by (*auto simp add: ω -A ε -A $\langle \omega A_0 \sigma_0 = \{\} \rangle \langle \varepsilon A_1 \sigma_1 = \{\} \rangle$ intro!: mono-Mprefix-eq,*
auto simp add: A-def combine_{Pair}-Sync_{ptick}-defs $\langle \omega A_1 \sigma_1 \neq \{\} \rangle$
 $\langle \varepsilon A_1 \sigma_1 = \{\} \rangle$ cartprod-rwrt Q-rec[of σ_1] intro!: mono-GlobalNdet-eq)
also have $\langle \dots = P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 \rangle$
by (*unfold P-rec[of σ_0] Q-rec[of σ_1]*)
(auto simp add: SKIPS-def Ndet-and-Sync_{Pair} Mprefix-Sync_{Pair}-constant
 $\langle \omega A_1 \sigma_1 \neq \{\} \rangle$
GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[of $\langle \tau A_0 - \rangle$] $\langle \omega$
 $A_0 \sigma_0 = \{\} \rangle$
intro!: mono-Mprefix-eq mono-GlobalNdet-eq)
finally show $\langle ?f (\sigma_0, \sigma_1) = \dots \rangle$.
next
assume $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \langle \omega A_1 \sigma_1 \neq \{\} \rangle$
with ϱ -disjoint- ε **have** $\langle \varepsilon A_0 \sigma_0 = \{\} \rangle \langle \varepsilon A_1 \sigma_1 = \{\} \rangle$
by (*simp-all add: ϱ -disjoint- ε -def ϱ -simps*)
show $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \implies \langle \omega A_1 \sigma_1 \neq \{\} \rangle \implies ?f (\sigma_0, \sigma_1) = P \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} Q \sigma_1 \rangle$
by (*simp add: $\langle \omega A_0 \sigma_0 \neq \{\} \rangle \langle \omega A_1 \sigma_1 \neq \{\} \rangle$ ω -A P-rec[of σ_0] Q-rec[of σ_1] SKIPS-def*
Ndet-and-Sync_{Pair} cartprod-rwrt GlobalNdet-sets-commute[of $\langle \omega A_0$
 \rightarrow])
qed
qed
qed
qed
qed

corollary P - nd - $combine_{Pair}$ - $Sync_{ptick}$:

$$\langle P\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P\langle A_1 \rangle_{nd} \sigma_1 = P\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle\rangle_{nd} (\sigma_0, \sigma_1) \rangle$$

proof –

$$\text{have } \langle P\langle A_0 \rangle_{nd} \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P\langle A_1 \rangle_{nd} \sigma_1 =$$

$$P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \{\}) \rangle_{nd} \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \{\}) \rangle_{nd} \sigma_1 \rangle$$

by (*simp add: P_{SKIPS}-nd-updated- ω*)

$$\text{also have } \langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1(\omega := \lambda\sigma. \{\}) \rangle\rangle_{nd} (\sigma_0, \sigma_1) \rangle$$

by (*rule P_{SKIPS}-nd-combine_{Pair}-Sync_{ptick}*) *simp-all*

$$\text{also have } \langle \langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1(\omega := \lambda\sigma. \{\}) \rangle = \langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle$$

by (*auto simp add: combine_{Pair}-Sync_{ptick}-defs*)

$$\text{also have } \langle P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle\rangle(\omega := \lambda\sigma. \{\})_{nd} = P\langle\langle A_0 \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle\rangle_{nd} \rangle$$

by (*simp add: P_{SKIPS}-nd-updated- ω*)

finally show *?thesis* .

qed

corollary P_{SKIPS} - d - $combine_{Pair}$ - $Sync_{ptick}$:

$$\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle\rangle_d (\sigma_0, \sigma_1) \rangle$$

if $\langle \varrho$ -*disjoint- ε A₀* and $\langle \varrho$ -*disjoint- ε A₁* and \langle *indep-enabl A₀ σ_0 E A₁ σ_1* \rangle

proof –

$$\text{have } \langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P_{SKIPS}\langle\langle A_1 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_1 \rangle$$

by (*simp flip: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*)

$$\text{also have } \langle \dots = P_{SKIPS}\langle\langle\langle A_0 \rangle_{d \hookrightarrow nd} \text{ nd} \otimes \llbracket E \rrbracket_{\checkmark Pair} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle\rangle_{nd} (\sigma_0, \sigma_1) \rangle$$

by (*rule P_{SKIPS}-nd-combine_{Pair}-Sync_{ptick}*)

(*metis ϱ -disjoint- ε -def det-ndet-conv- ε (1) det-ndet-conv- ϱ (1) $\langle \varrho$ -disjoint- ε A₀*,
metis ϱ -disjoint- ε -def det-ndet-conv- ε (1) det-ndet-conv- ϱ (1) $\langle \varrho$ -disjoint- ε

A₁ \rangle)

$$\text{also have } \langle \dots = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle\rangle_d (\sigma_0, \sigma_1) \rangle$$

$$\text{by } (\textit{simp add: P}_{SKIPS}\textit{-combine}_{Pair}\textit{-Sync}_{ptick}\textit{-behaviour-when-indep } \langle \textit{indep-enabl } A_0 \textit{ } \sigma_0 \textit{ } E \textit{ } A_1 \textit{ } \sigma_1 \rangle)$$

finally show *?thesis* .

qed

corollary P - d - $combine_{Pair}$ - $Sync_{ptick}$:

$$\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \text{ d} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1 \rangle\rangle_d (\sigma_0, \sigma_1) \rangle$$

if \langle *indep-enabl A₀ σ_0 E A₁ σ_1* \rangle

proof –

$$\text{have } \langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P\langle A_1 \rangle_d \sigma_1 =$$

$$P_{SKIPS}\langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pair} P_{SKIPS}\langle A_1(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_1 \rangle$$

by (*simp add: P_{SKIPS}-d-updated- ω*)

$$\text{also have } \langle \dots = P_{SKIPS}\langle\langle A_0(\omega := \lambda\sigma. \diamond) \text{ d} \otimes \llbracket E \rrbracket_{\checkmark Pair} A_1(\omega := \lambda\sigma. \diamond) \rangle\rangle_d (\sigma_0, \sigma_1) \rangle$$

by (*subst P_{SKIPS}-d-combine_{Pair}-Sync_{ptick}, simp-all add: ϱ -simps ϱ -disjoint- ε -def*,
rule indep-enabl,

use $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle [\text{THEN indep-enabl}D]$ in $\langle \text{simp add: } \varepsilon\text{-simps} \rangle$
 also have $\langle \langle A_0(\omega := \lambda\sigma. \diamond) \text{ d}\otimes[E]_{\checkmark\text{Pair}} A_1(\omega := \lambda\sigma. \diamond) \rangle = \langle A_0 \text{ d}\otimes[E]_{\checkmark\text{Pair}} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle$
 by $(\text{simp add: combine}_{\text{Pair-Sync}_{\text{ptick}}}\text{-defs, intro ext, simp})$
 also have $\langle P_{\text{SKIPS}}\langle\langle A_0 \text{ d}\otimes[E]_{\checkmark\text{Pair}} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle_d = P\langle\langle A_0 \text{ d}\otimes[E]_{\checkmark\text{Pair}} A_1 \rangle\rangle_d \rangle$
 by $(\text{simp add: } P_{\text{SKIPS}}\text{-d-updated-}\omega)$
 finally show $?thesis$.
 qed

Pairlist

theorem $P_{\text{SKIPS}}\text{-nd-combine}_{\text{Pairlist-Sync}_{\text{ptick}}}$:

fixes $E :: \langle 'a \text{ set} \rangle$

assumes $\varrho\text{-disjoint-}\varepsilon : \langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$

shows $\langle P_{\text{SKIPS}}\langle A_0 \rangle_{\text{nd}} \sigma_0 \llbracket E \rrbracket_{\checkmark\text{Pairlist}} P_{\text{SKIPS}}\langle A_1 \rangle_{\text{nd}} \sigma_1 = P_{\text{SKIPS}}\langle\langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pairlist}} A_1 \rangle\rangle_{\text{nd}} [\sigma_0, \sigma_1] \rangle$

proof –

let $?A = \langle (\tau = \tau \langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pair}} A_1 \rangle, \omega = \lambda\sigma. (\lambda(r, s). [r, s]) \text{ ‘ } \omega \langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pair}} A_1 \rangle \sigma) \rangle$

have $\langle P_{\text{SKIPS}}\langle A_0 \rangle_{\text{nd}} \sigma_0 \llbracket E \rrbracket_{\checkmark\text{Pairlist}} P_{\text{SKIPS}}\langle A_1 \rangle_{\text{nd}} \sigma_1 =$

$\text{RenamingTick} (P_{\text{SKIPS}}\langle A_0 \rangle_{\text{nd}} \sigma_0 \llbracket E \rrbracket_{\checkmark\text{Pair}} P_{\text{SKIPS}}\langle A_1 \rangle_{\text{nd}} \sigma_1) (\lambda(r, s). [r, s]) \rangle$

by $(\text{simp only: Sync}_{\text{Pair-to-Pairlist}})$

also have $\langle \dots = \text{RenamingTick} (P_{\text{SKIPS}}\langle\langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pair}} A_1 \rangle\rangle_{\text{nd}} (\sigma_0, \sigma_1)) (\lambda(r, s). [r, s]) \rangle$

by $(\text{simp add: } P_{\text{SKIPS}}\text{-nd-combine}_{\text{Pair-Sync}_{\text{ptick}}}\text{-} \varrho\text{-disjoint-}\varepsilon)$

also have $\langle \dots = P_{\text{SKIPS}}\langle ?A \rangle_{\text{nd}} (\sigma_0, \sigma_1) \rangle$

by $(\text{simp only: RenamingTick-}P_{\text{SKIPS}}\text{-nd})$

also have $\langle \dots = P_{\text{SKIPS}}\langle\langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pairlist}} A_1 \rangle\rangle_{\text{nd}} [\sigma_0, \sigma_1] \rangle$

by $(\text{auto intro!: } P_{\text{SKIPS}}\text{-nd-eqI-strong}[of \langle \lambda(r, s). [r, s] \rangle - \langle (\sigma_0, \sigma_1) \rangle, \text{simplified}])$

$\text{inj-onI} (\text{auto simp add: combine-Sync}_{\text{ptick}}\text{-defs split: if-split-asm})$

finally show $?thesis$.

qed

corollary $P\text{-nd-combine}_{\text{Pairlist-Sync}_{\text{ptick}}}$:

$\langle P\langle A_0 \rangle_{\text{nd}} \sigma_0 \llbracket E \rrbracket_{\checkmark\text{Pairlist}} P\langle A_1 \rangle_{\text{nd}} \sigma_1 = P\langle\langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pairlist}} A_1 \rangle\rangle_{\text{nd}} [\sigma_0, \sigma_1] \rangle$

proof –

have $\langle P\langle A_0 \rangle_{\text{nd}} \sigma_0 \llbracket E \rrbracket_{\checkmark\text{Pairlist}} P\langle A_1 \rangle_{\text{nd}} \sigma_1 =$

$P_{\text{SKIPS}}\langle A_0(\omega := \lambda\sigma. \{\}) \rangle_{\text{nd}} \sigma_0 \llbracket E \rrbracket_{\checkmark\text{Pairlist}} P_{\text{SKIPS}}\langle A_1(\omega := \lambda\sigma. \{\}) \rangle_{\text{nd}} \sigma_1 \rangle$

by $(\text{simp add: } P_{\text{SKIPS}}\text{-nd-updated-}\omega)$

also have $\langle \dots = P_{\text{SKIPS}}\langle\langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd}\otimes[E]_{\checkmark\text{Pairlist}} A_1(\omega := \lambda\sigma. \{\}) \rangle\rangle_{\text{nd}} [\sigma_0, \sigma_1] \rangle$

by $(\text{rule } P_{\text{SKIPS}}\text{-nd-combine}_{\text{Pairlist-Sync}_{\text{ptick}}}\text{-simp-all})$

also have $\langle \langle A_0(\omega := \lambda\sigma. \{\}) \text{ nd}\otimes[E]_{\checkmark\text{Pairlist}} A_1(\omega := \lambda\sigma. \{\}) \rangle = \langle A_0 \text{ nd}\otimes[E]_{\checkmark\text{Pairlist}} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle$

by (*simp add: combinePairlist-Syncptick-defs, intro ext, simp*)
 also have $\langle P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pairlist} A_1 \rangle (\omega := \lambda \sigma. \{\}) \rangle_{nd} = P \langle \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pairlist} A_1 \rangle_{nd} \rangle$
 by (*simp add: P_{SKIPS}-nd-updated- ω*)
 finally show *?thesis* .
 qed

corollary *P_{SKIPS}-d-combinePairlist-Syncptick* :
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E] \checkmark_{Pairlist} P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1]$
 if $\langle \rho\text{-disjoint-}\varepsilon A_0 \rangle$ and $\langle \rho\text{-disjoint-}\varepsilon A_1 \rangle$ and $\langle indep\text{-enabl} A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
 have $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E] \checkmark_{Pairlist} P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle A_0 \rangle_{d \rightarrow nd} \rangle_{nd} \sigma_0 [E] \checkmark_{Pairlist} P_{SKIPS} \langle \langle A_1 \rangle_{d \rightarrow nd} \rangle_{nd} \sigma_1$
 by (*simp flip: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*)
 also have $\langle \dots = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \rightarrow nd} \text{ nd} \otimes [E] \checkmark_{Pairlist} \langle A_1 \rangle_{d \rightarrow nd} \rangle \rangle_{nd} [\sigma_0, \sigma_1]$
 by (*rule P_{SKIPS}-nd-combinePairlist-Syncptick*)
 (*metis $\rho\text{-disjoint-}\varepsilon\text{-def det-ndet-conv-}\varepsilon(1)$ det-ndet-conv- $\rho(1)$ $\langle \rho\text{-disjoint-}\varepsilon A_0 \rangle$,
metis $\rho\text{-disjoint-}\varepsilon\text{-def det-ndet-conv-}\varepsilon(1)$ det-ndet-conv- $\rho(1)$ $\langle \rho\text{-disjoint-}\varepsilon A_1 \rangle$)
 also have $\langle \dots = P_{SKIPS} \langle \langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1]$
 by (*simp add: P_{SKIPS}-combinePairlist-Syncptick-behaviour-when-indep $\langle indep\text{-enabl} A_0 \sigma_0 E A_1 \sigma_1 \rangle$*)
 finally show *?thesis* .
 qed*

corollary *P-d-combinePairlist-Syncptick* :
 $\langle P \langle A_0 \rangle_d \sigma_0 [E] \checkmark_{Pairlist} P \langle A_1 \rangle_d \sigma_1 = P \langle \langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1]$
 if $\langle indep\text{-enabl} A_0 \sigma_0 E A_1 \sigma_1 \rangle$
proof –
 have $\langle P \langle A_0 \rangle_d \sigma_0 [E] \checkmark_{Pairlist} P \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle A_0 (\omega := \lambda \sigma. \diamond) \rangle_d \sigma_0 [E] \checkmark_{Pairlist} P_{SKIPS} \langle A_1 (\omega := \lambda \sigma. \diamond) \rangle_d \sigma_1$
 by (*simp add: P_{SKIPS}-d-updated- ω*)
 also have $\langle \dots = P_{SKIPS} \langle \langle A_0 (\omega := \lambda \sigma. \diamond) \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 (\omega := \lambda \sigma. \diamond) \rangle \rangle_d [\sigma_0, \sigma_1]$
 by (*subst P_{SKIPS}-d-combinePairlist-Syncptick, simp-all*)
 (*rule indep-enablI*,
*use $\langle indep\text{-enabl} A_0 \sigma_0 E A_1 \sigma_1 \rangle [THEN indep\text{-enablD}]$ in *simp*)
 also have $\langle \langle A_0 (\omega := \lambda \sigma. \diamond) \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 (\omega := \lambda \sigma. \diamond) \rangle = \langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle (\omega := \lambda \sigma. \diamond)$
 by (*simp add: combinePairlist-Syncptick-defs, intro ext, simp*)
 also have $\langle P_{SKIPS} \langle \langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle (\omega := \lambda \sigma. \diamond) \rangle_d = P \langle \langle A_0 \text{ d} \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d$
 by (*simp add: P_{SKIPS}-d-updated- ω*)
 finally show *?thesis* .
 qed*

8.2.2 Rlist

theorem $P_{SKIPS}\text{-nd-combine}_{Rlist}\text{-Sync}_{ptick}$:

fixes $E :: \langle 'a \text{ set} \rangle$

assumes $\rho\text{-disjoint-}\varepsilon : \langle \rho\text{-disjoint-}\varepsilon A_0 \rangle \langle \rho\text{-disjoint-}\varepsilon A_1 \rangle$

defines $A\text{-def}$: $\langle A \equiv \langle A_0 \text{ nd} \otimes [E]_{\checkmark Rlist} A_1 \rangle \rangle$

defines $P\text{-def}$: $\langle P \equiv P_{SKIPS} \langle A_0 \rangle_{nd} \rangle$ **and** $Q\text{-def}$: $\langle Q \equiv P_{SKIPS} \langle A_1 \rangle_{nd} \rangle$ **and** $S\text{-def}$: $\langle S \equiv P_{SKIPS} \langle A \rangle_{nd} \rangle$

shows $\langle P \sigma_0 [E]_{\checkmark Rlist} Q \sigma s = S (\sigma_0 \# \sigma s) \rangle$

proof –

let $?A' = \langle (\tau = \tau \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle, \omega = \lambda \sigma. (\lambda(x, y). x \# y) \text{ ' } \omega \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \sigma) \rangle$

from $P_{SKIPS}\text{-nd-combine}_{Pair}\text{-Sync}_{ptick}[OF \ \rho\text{-disjoint-}\varepsilon]$

have $\langle P \sigma_0 [E]_{\checkmark Pair} Q \sigma s = P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma s) \rangle$ **by** (*simp add: P-def Q-def*)

hence $\langle \text{RenamingTick} (P \sigma_0 [E]_{\checkmark Pair} Q \sigma s) (\lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s) =$

$\text{RenamingTick} (P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma s)) (\lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s) \rangle$ **by** *simp*

also have $\langle \text{RenamingTick} (P \sigma_0 [E]_{\checkmark Pair} Q \sigma s) (\lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s) = P \sigma_0 [E]_{\checkmark Rlist} Q \sigma s \rangle$

by (*auto intro: inj-onI Sync_{Pair}.inj-on-RenamingTick-Sync_{ptick}*
[*of* $\langle \lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s \rangle$, *simplified*])

also have $\langle \text{RenamingTick} (P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma s)) (\lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s) = S (\sigma_0 \# \sigma s) \rangle$

proof (*unfold RenamingTick-P_{SKIPS}\text{-nd S-def}*,

*rule P_{SKIPS}\text{-nd-eqI-strong}[*of* $\langle \lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s \rangle ?A' (\sigma_0, \sigma s) \rangle$, *simplified*])*

show $\langle \text{inj-on} (\lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s) (\mathcal{R}_{nd} ?A' (\sigma_0, \sigma s)) \rangle$ **by** (*auto intro: inj-onI*)

next

show $\langle \tau A \text{ (case } \sigma' \text{ of } (\sigma_0, \sigma s) \Rightarrow \sigma_0 \# \sigma s) e =$

$(\lambda \sigma''. \text{ case } \sigma'' \text{ of } (\sigma_0, \sigma s) \Rightarrow \sigma_0 \# \sigma s) \text{ ' } \tau \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \sigma' e \rangle$

for $\sigma' e$

by (*cases* σ') (*auto simp add: A-def combine_{Rlist}\text{-Sync}_{ptick}\text{-defs combine}_{Pair}\text{-Sync}_{ptick}\text{-defs}*)

next

show $\langle \omega A \text{ (case } \sigma' \text{ of } (\sigma_0, \sigma s) \Rightarrow \sigma_0 \# \sigma s) =$

$(\lambda \sigma''. \text{ case } \sigma'' \text{ of } (\sigma_0, \sigma s) \Rightarrow \sigma_0 \# \sigma s) \text{ ' } \omega \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \sigma' \rangle$ **for**

σ'

by (*cases* σ') (*auto simp add: A-def combine_{Rlist}\text{-Sync}_{ptick}\text{-defs combine}_{Pair}\text{-Sync}_{ptick}\text{-defs}*)

qed

finally show $\langle P \sigma_0 [E]_{\checkmark Rlist} Q \sigma s = S (\sigma_0 \# \sigma s) \rangle$.

qed

corollary $P\text{-nd-combine}_{Rlist}\text{-Sync}_{ptick}$:

$\langle P \langle A_0 \rangle_{nd} \sigma_0 [E]_{\checkmark Rlist} P \langle A_1 \rangle_{nd} \sigma s = P \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Rlist} A_1 \rangle \rangle_{nd} (\sigma_0 \# \sigma s) \rangle$

proof –

have $\langle P \langle A_0 \rangle_{nd} \sigma_0 [E]_{\checkmark Rlist} P \langle A_1 \rangle_{nd} \sigma s =$

$P_{SKIPS} \langle A_0 (\omega := \lambda \sigma. \{\}) \rangle_{nd} \sigma_0 [E]_{\checkmark Rlist} P_{SKIPS} \langle A_1 (\omega := \lambda \sigma. \{\}) \rangle_{nd} \sigma s \rangle$

by (*simp add: P_{SKIPS}\text{-nd-updated-}\omega*)

also have $\langle \dots = P_{SKIPS} \langle \langle A_0 (\omega := \lambda \sigma. \{\}) \text{ nd} \otimes [E]_{\checkmark Rlist} A_1 (\omega := \lambda \sigma. \{\}) \rangle \rangle_{nd} (\sigma_0 \# \sigma s) \rangle$

by (rule P_{SKIPS} -nd-combine $_{Rlist}$ -Sync $_{ptick}$)
 (simp-all add: ϱ -simps ϱ -disjoint- ε -def)
 also have $\langle \langle A_0(\omega := \lambda\sigma. \{\}) \rangle_{nd} \otimes [E]_{\checkmark Rlist} A_1(\omega := \lambda\sigma. \{\}) \rangle = \langle A_0 \rangle_{nd} \otimes [E]_{\checkmark Rlist} A_1 \rangle(\omega := \lambda\sigma. \{\})$
 by (simp add: combine $_{Rlist}$ -Sync $_{ptick}$ -defs, intro ext, simp add: ε -simps)
 also have $\langle P_{SKIPS} \langle \langle A_0 \rangle_{nd} \otimes [E]_{\checkmark Rlist} A_1 \rangle(\omega := \lambda\sigma. \{\}) \rangle_{nd} = P \langle \langle A_0 \rangle_{nd} \otimes [E]_{\checkmark Rlist} A_1 \rangle \rangle_{nd}$
 by (simp add: P_{SKIPS} -nd-updated- ω)
 finally show ?thesis .
 qed

corollary P_{SKIPS} -d-combine $_{Rlist}$ -Sync $_{ptick}$:
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E]_{\checkmark Rlist} P_{SKIPS} \langle A_1 \rangle_d \sigma s = P_{SKIPS} \langle \langle A_0 \rangle_d \otimes [E]_{\checkmark Rlist} A_1 \rangle_d (\sigma_0 \# \sigma s)$
 if $\langle \varrho$ -disjoint- ε $A_0 \rangle$ and $\langle \varrho$ -disjoint- ε $A_1 \rangle$ and $\langle indep$ -enabl $A_0 \sigma_0 E A_1 \sigma s \rangle$
proof –
 have $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E]_{\checkmark Rlist} P_{SKIPS} \langle A_1 \rangle_d \sigma s = P_{SKIPS} \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_0 [E]_{\checkmark Rlist} P_{SKIPS} \langle \langle A_1 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma s$
 by (simp flip: P_{SKIPS} -nd-from-det-to-ndet-is- P_{SKIPS} -d)
 also have $\langle \dots = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd} \otimes [E]_{\checkmark Rlist} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma_0 \# \sigma s)$
 by (rule P_{SKIPS} -nd-combine $_{Rlist}$ -Sync $_{ptick}$)
 (metis ϱ -disjoint- ε -def det-ndet-conv- $\varepsilon(1)$ det-ndet-conv- $\varrho(1)$ $\langle \varrho$ -disjoint- ε $A_0 \rangle$,
 metis ϱ -disjoint- ε -def det-ndet-conv- $\varepsilon(1)$ det-ndet-conv- $\varrho(1)$ $\langle \varrho$ -disjoint- ε $A_1 \rangle$)
 also have $\langle \dots = P_{SKIPS} \langle \langle A_0 \rangle_d \otimes [E]_{\checkmark Rlist} A_1 \rangle_d (\sigma_0 \# \sigma s)$
 by (simp add: P_{SKIPS} -combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour-when-indep $\langle indep$ -enabl $A_0 \sigma_0 E A_1 \sigma s \rangle$)
 finally show ?thesis .
 qed

corollary P -d-combine $_{Rlist}$ -Sync $_{ptick}$:
 $\langle P \langle A_0 \rangle_d \sigma_0 [E]_{\checkmark Rlist} P \langle A_1 \rangle_d \sigma s = P \langle \langle A_0 \rangle_d \otimes [E]_{\checkmark Rlist} A_1 \rangle_d (\sigma_0 \# \sigma s)$
 if $\langle indep$ -enabl $A_0 \sigma_0 E A_1 \sigma s \rangle$
proof –
 have $\langle P \langle A_0 \rangle_d \sigma_0 [E]_{\checkmark Rlist} P \langle A_1 \rangle_d \sigma s = P_{SKIPS} \langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \sigma_0 [E]_{\checkmark Rlist} P_{SKIPS} \langle A_1(\omega := \lambda\sigma. \diamond) \rangle_d \sigma s$
 by (simp add: P_{SKIPS} -d-updated- ω)
 also have $\langle \dots = P_{SKIPS} \langle \langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \otimes [E]_{\checkmark Rlist} A_1(\omega := \lambda\sigma. \diamond) \rangle \rangle_d (\sigma_0 \# \sigma s)$
 by (rule P_{SKIPS} -d-combine $_{Rlist}$ -Sync $_{ptick}$, simp-all add: ϱ -simps ϱ -disjoint- ε -def)
 (rule indep-enablI,
 use $\langle indep$ -enabl $A_0 \sigma_0 E A_1 \sigma s \rangle$ [THEN indep-enablD] in $\langle simp$ add: ε -simps)
 also have $\langle \langle A_0(\omega := \lambda\sigma. \diamond) \rangle_d \otimes [E]_{\checkmark Rlist} A_1(\omega := \lambda\sigma. \diamond) \rangle = \langle A_0 \rangle_d \otimes [E]_{\checkmark Rlist} A_1 \rangle(\omega := \lambda\sigma. \diamond)$
 by (simp add: combine $_{Rlist}$ -Sync $_{ptick}$ -defs, intro ext, simp add: ε -simps)
 also have $\langle P_{SKIPS} \langle \langle A_0 \rangle_d \otimes [E]_{\checkmark Rlist} A_1 \rangle(\omega := \lambda\sigma. \diamond) \rangle_d = P \langle \langle A_0 \rangle_d \otimes [E]_{\checkmark Rlist} A_1 \rangle \rangle_d$
 by (simp add: P_{SKIPS} -d-updated- ω)
 finally show ?thesis .

qed

8.2.3 ListslenL

theorem $P_{SKIPS}\text{-nd-combine}_{ListslenL}\text{-Sync}_{ptick}$:

fixes $E :: \langle 'a \text{ set} \rangle$

assumes $\text{same-length-reach0} : \langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$

and $\text{same-length-term0} : \langle \bigwedge \sigma_0' \text{ rs. } \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \text{rs} \in \omega A_0 \sigma_0' \implies \text{length rs} = \text{len}_0 \rangle$

and $\varrho\text{-disjoint-}\varepsilon : \langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$

defines $A\text{-def} : \langle A \equiv \langle A_0 \text{ nd} \otimes [\text{len}_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle$

defines $P\text{-def} : \langle P \equiv P_{SKIPS} \langle A_0 \rangle_{nd} \rangle$ **and** $Q\text{-def} : \langle Q \equiv P_{SKIPS} \langle A_1 \rangle_{nd} \rangle$ **and** $S\text{-def} : \langle S \equiv P_{SKIPS} \langle A \rangle_{nd} \rangle$

shows $\langle P \sigma_0 \text{ len}_0 [E] \checkmark_{ListslenL} Q \sigma_1 = S (\sigma_0 @ \sigma_1) \rangle$

proof –

let $?A' = \langle (\tau = \tau \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pair} A_1 \rangle, \omega = \lambda \sigma. (\lambda(x, y). x @ y) ' \omega \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pair} A_1 \rangle \sigma) \rangle$

let $?RT = \text{RenamingTick}$

have $* : \langle \sigma' \in \mathcal{R}_{nd} ?A' (\sigma_0, \sigma_1) \implies \text{length } (\text{fst } \sigma') = \text{len}_0 \rangle$ **for** σ'

by ($\text{metis (no-types, lifting) } \mathcal{R}_{nd}\text{-combine}_{ndPair}\text{-Sync}_{ptick}\text{-subset mem-Sigma-iff prod.collapse}$)

$\text{same-}\tau\text{-implies-same-}\mathcal{R}_{nd} \text{ same-length-reach0 select-conv}(1) \text{ subset-iff}$)

from $P_{SKIPS}\text{-nd-combine}_{Pair}\text{-Sync}_{ptick}[OF \varrho\text{-disjoint-}\varepsilon]$

have $\langle P \sigma_0 [E] \checkmark_{Pair} Q \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1) \rangle$

by ($\text{simp add: } P\text{-def } Q\text{-def}$)

hence $\langle ?RT (P \sigma_0 [E] \checkmark_{Pair} Q \sigma_1) (\lambda(\sigma_0, \sigma s). \sigma_0 @ \sigma s) = ?RT (P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1)) (\lambda(\sigma_0, \sigma s). \sigma_0 @ \sigma s) \rangle$

by simp

also have $\langle ?RT (P \sigma_0 [E] \checkmark_{Pair} Q \sigma_1) (\lambda(\sigma_0, \sigma s). \sigma_0 @ \sigma s) = P \sigma_0 \text{ len}_0 [E] \checkmark_{ListslenL} Q \sigma_1 \rangle$

by ($\text{rule Sync}_{Pair}\text{-to-Sync}_{ListslenL}[OF \text{is-ticks-length-}P_{SKIPS}\text{-nd}[of } A_0, \text{ folded } P\text{-def}]]$)

($\text{fact same-length-term0}$)

also have $\langle ?RT (P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1)) (\lambda(\sigma_0, \sigma s). \sigma_0 @ \sigma s) = S (\sigma_0 @ \sigma_1) \rangle$

by ($\text{auto simp add: RenamingTick-}P_{SKIPS}\text{-nd } S\text{-def dest!: } * \text{ intro!: } P_{SKIPS}\text{-nd-eqI-strong[of } \langle \lambda(\sigma_0, \sigma s). \sigma_0 @ \sigma s \rangle ?A' \langle (\sigma_0, \sigma_1) \rangle, \text{ simplified } \text{ inj-onI}$)

($\text{force simp add: image-iff } A\text{-def combine}_{ListslenL}\text{-Sync}_{ptick}\text{-defs combine}_{Pair}\text{-Sync}_{ptick}\text{-defs split: if-split-asm}$)+

finally show $?thesis$.

qed

corollary $P\text{-nd-combine}_{ListslenL}\text{-Sync}_{ptick}$:

$\langle P \langle A_0 \rangle_{nd} \sigma_0 \text{ len}_0 [E] \checkmark_{ListslenL} P \langle A_1 \rangle_{nd} \sigma_1 = P \langle \langle A_0 \text{ nd} \otimes [\text{len}_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$

if $\text{same-length-reach0} : \langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$

proof –

have $\ast : \langle \sigma s \in \mathcal{R}_{nd} \langle A_0(\omega := \lambda\sigma_0. \{\}) \rangle_{nd \otimes} [len_0, E] \checkmark_{ListslenL} A_1(\omega := \lambda\sigma_1. \{\}) \rangle (\sigma_0 @ \sigma_1) \implies len_0 \leq length \sigma s \rangle$ **for** σs
by (*auto dest: set-rev-mp[OF - \mathcal{R}_{nd} -combine $_{ndListslenL}$ -Sync $_{ptick}$ -subset]*
simp add: same-length-reach0)
have $\langle P \langle A_0 \rangle_{nd} \sigma_0 len_0 [E] \checkmark_{ListslenL} P \langle A_1 \rangle_{nd} \sigma_1 =$
 $PSKIPS \langle A_0(\omega := \lambda\sigma_0. \{\}) \rangle_{nd} \sigma_0 len_0 [E] \checkmark_{ListslenL} PSKIPS \langle A_1(\omega :=$
 $\lambda\sigma_1. \{\}) \rangle_{nd} \sigma_1 \rangle$
by (*simp only: PSKIPS-nd-updated- ω*)
also have $\langle \dots = PSKIPS \langle \langle A_0(\omega := \lambda\sigma_0. \{\}) \rangle_{nd \otimes} [len_0, E] \checkmark_{ListslenL} A_1(\omega :=$
 $\lambda\sigma_1. \{\}) \rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
by (*auto intro: PSKIPS-nd-combine $_{ListslenL}$ -Sync $_{ptick}$ simp add: same-length-reach0*)
also have $\langle \dots = PSKIPS \langle \langle A_0 \rangle_{nd \otimes} [len_0, E] \checkmark_{ListslenL} A_1 \rangle (\omega := \lambda\sigma_1. \{\}) \rangle_{nd}$
 $(\sigma_0 @ \sigma_1) \rangle$
by (*auto intro!: PSKIPS-nd-eqI-strong-id dest!: \ast*)
(auto simp add: combine $_{ListslenL}$ -Sync $_{ptick}$ -defs split: if-split-asm)
also have $\langle \dots = P \langle \langle A_0 \rangle_{nd \otimes} [len_0, E] \checkmark_{ListslenL} A_1 \rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
by (*simp only: PSKIPS-nd-updated- ω*)
finally show *?thesis .*

qed

corollary $PSKIPS$ - d -combine $_{ListslenL}$ -Sync $_{ptick}$:

assumes *same-length-reach0* : $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies length \sigma_0' = len_0 \rangle$
and *same-length-term0* : $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies \omega A_0 \sigma_0' \neq \diamond \implies length$
 $[\omega A_0 \sigma_0'] = len_0 \rangle$
and ρ -disjoint- ε : $\langle \rho$ -disjoint- $\varepsilon A_0 \rangle \langle \rho$ -disjoint- $\varepsilon A_1 \rangle$
and *indep-enabl* : $\langle indep-enabl A_0 \sigma_0 E A_1 \sigma_1 \rangle$
shows $\langle PSKIPS \langle A_0 \rangle_d \sigma_0 len_0 [E] \checkmark_{ListslenL} PSKIPS \langle A_1 \rangle_d \sigma_1 =$
 $PSKIPS \langle \langle A_0 \rangle_{d \otimes} [len_0, E] \checkmark_{ListslenL} A_1 \rangle_d (\sigma_0 @ \sigma_1) \rangle$

proof –

have $\ast : \langle \sigma s \in \mathcal{R}_{nd} \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd \otimes} [len_0, E] \checkmark_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (\sigma_0 @ \sigma_1)$
 \implies
 $\exists \sigma_0' \sigma_1'. \sigma s = \sigma_0' @ \sigma_1' \wedge \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \wedge \sigma_1' \in \mathcal{R}_d A_1 \sigma_1 \rangle$ **for** σs
by (*auto dest!: set-rev-mp[OF - \mathcal{R}_{nd} -combine $_{ndListslenL}$ -Sync $_{ptick}$ -subset]*
simp add: \mathcal{R}_{nd} -from-det-to-ndet same-length-reach0)
have $\langle PSKIPS \langle A_0 \rangle_d \sigma_0 len_0 [E] \checkmark_{ListslenL} PSKIPS \langle A_1 \rangle_d \sigma_1 =$
 $PSKIPS \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_0 len_0 [E] \checkmark_{ListslenL} PSKIPS \langle \langle A_1 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_1 \rangle$
by (*simp only: PSKIPS-nd-from-det-to-ndet-is-PSKIPS-d*)
also from *same-length-term0* **have** $\langle \dots = PSKIPS \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd \otimes} [len_0, E] \checkmark_{ListslenL}$
 $\langle A_1 \rangle_{d \hookrightarrow nd} \rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$
by (*auto intro!: PSKIPS-nd-combine $_{ListslenL}$ -Sync $_{ptick}$ split: option.split-asm*
simp add: ρ -disjoint- ε \mathcal{R}_{nd} -from-det-to-ndet same-length-reach0 ω -from-det-to-ndet)
(metis option.sel))
also from *indep-enabl* **have** $\langle \dots = PSKIPS \langle \langle \langle A_0 \rangle_{d \otimes} [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle_{nd}$
 $(\sigma_0 @ \sigma_1) \rangle$
by (*auto intro!: PSKIPS-nd-eqI-strong-id*
 τ -combine $_{ListslenL}$ -Sync $_{ptick}$ -behaviour-when-indep ω -combine $_{ListslenL}$ -Sync $_{ptick}$ -behaviour
simp add: same-length-reach0 dest!: \ast indep-enablD)
also have $\langle \dots = PSKIPS \langle \langle A_0 \rangle_{d \otimes} [len_0, E] \checkmark_{ListslenL} A_1 \rangle_d (\sigma_0 @ \sigma_1) \rangle$

by (*simp only: P_SKIPS-nd-from-det-to-ndet-is-P_SKIPS-d*)
finally show *?thesis* .
qed

corollary *P-d-combine_{ListslenL-Sync_{ptick}}* :

assumes *same-length-reach0* : $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$
and *indep-enabl* : $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$
shows $\langle P \langle A_0 \rangle_d \sigma_0 \text{len}_0 [E] \checkmark_{\text{ListslenL}} P \langle A_1 \rangle_d \sigma_1 =$
 $P \langle \langle A_0 \rangle_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \rangle_d (\sigma_0 @ \sigma_1) \rangle$

proof –

have $*$: $\langle \sigma s \in \mathcal{R}_d \langle A_0 (\omega := \lambda \sigma_0. \diamond) \rangle_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 (\omega := \lambda \sigma_1. \diamond) \rangle$
 $(\sigma_0 @ \sigma_1) \implies \text{len}_0 \leq \text{length } \sigma s \rangle$ **for** σs

by (*auto dest: set-rev-mp[OF - \mathcal{R}_d -combine_{dListslenL-Sync_{ptick}-subset]}*
simp add: same-length-reach0)

have $\langle P \langle A_0 \rangle_d \sigma_0 \text{len}_0 [E] \checkmark_{\text{ListslenL}} P \langle A_1 \rangle_d \sigma_1 =$
 $P_{SKIPS} \langle A_0 (\omega := \lambda \sigma_0. \diamond) \rangle_d \sigma_0 \text{len}_0 [E] \checkmark_{\text{ListslenL}} P_{SKIPS} \langle A_1 (\omega := \lambda \sigma_1.$
 $\diamond) \rangle_d \sigma_1 \rangle$

by (*simp only: P_SKIPS-d-updated- ω*)

also from *indep-enabl*

have $\langle \dots = P_{SKIPS} \langle \langle A_0 (\omega := \lambda \sigma_0. \diamond) \rangle_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 (\omega := \lambda \sigma_1.$
 $\diamond) \rangle_d (\sigma_0 @ \sigma_1) \rangle$

by (*auto intro: P_SKIPS-d-combine_{ListslenL-Sync_{ptick}}* *simp add: same-length-reach0*)

also have $\langle \dots = P_{SKIPS} \langle \langle A_0 \rangle_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \rangle (\omega := \lambda \sigma_0. \diamond) \rangle_d (\sigma_0$
 $@ \sigma_1) \rangle$

by (*auto intro!: P_SKIPS-d-eqI-strong-id dest!: **)

(*auto simp add: combine_{ListslenL-Sync_{ptick}-defs split: if-split-asm}*)

also have $\langle \dots = P \langle \langle A_0 \rangle_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \rangle_d (\sigma_0 @ \sigma_1) \rangle$

by (*simp only: P_SKIPS-d-updated- ω*)

finally show *?thesis* .

qed

8.2.4 Multiple

theorem *P_SKIPS-nd-compactification-Sync_{ptick}*:

$\langle \text{length } \sigma s = \text{length } As \implies (\bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A) \implies$

$[E] \checkmark (\sigma, A) \in @ \text{zip } \sigma s As. P_{SKIPS} \langle A \rangle_{nd} \sigma = P_{SKIPS} \langle \langle nd \otimes [E] \checkmark As \rangle \rangle_{nd}$
 $\sigma s \rangle$

proof (*induct σs As rule: induct-2-lists012*)

case Nil show *?case by (simp, subst P_SKIPS-nd-rec, simp)*

next

case (single $\sigma_0 A_0$) show *?case*

by (*auto simp add: RenamingTick-P_SKIPS-nd singl-list-conv-defs*
intro!: inj-onI P_SKIPS-nd-eqI-strong)

next

case (Cons $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

have $\varrho\text{-disjoint-}\varepsilon$: $\langle A \in \text{set } (A_1 \# As) \implies \varrho\text{-disjoint-}\varepsilon A \rangle$ **for** A

by (*simp add: Cons.prem*)

show *?case*

by (*simp add: Cons.hyps(3)[OF $\varrho\text{-disjoint-}\varepsilon$, simplified] Cons.prem*)

$P_{SKIPs-nd-combine_{Rlist-Sync_{ptick}}}$ ϱ -disjoint- ε -transmission-to-iterated-combine $_{nd-Sync_{ptick}}$)
qed

corollary P -nd-compactification- $Sync_{ptick}$:

$\langle \text{length } \sigma s = \text{length } As \implies \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{ zip } \sigma s As. P \langle A \rangle_{nd} \sigma = P \langle \langle_{nd} \otimes \llbracket E \rrbracket_{\checkmark} As \rangle \rangle_{nd} \sigma s \rangle$

proof (induct σs As rule: induct-2-lists012)

case *Nil* **show** ?case **by** (simp, subst P -nd-rec, simp)

next

case (single $\sigma_0 A_0$) **show** ?case

by (simp, subst (1 2) $P_{SKIPs-nd-updated-\omega}$)

(auto simp add: RenamingTick- $P_{SKIPs-nd-singl-list-conv-defs}$)

intro!: inj-onI $P_{SKIPs-nd-eqI-strong}$)

next

case (Cons $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

show ?case

by (simp add: Cons.hyps(3)[simplified] Cons.prem)

P -nd-combine $_{Rlist-Sync_{ptick}}$ ϱ -disjoint- ε -transmission-to-iterated-combine $_{nd-Sync_{ptick}}$)

qed

corollary $P_{SKIPs-d-compactification-Sync_{ptick}}$:

$\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A; \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$

$\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies$

$\llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{ zip } \sigma s As. P_{SKIPs} \langle A \rangle_d \sigma = P_{SKIPs} \langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} As \rangle \rangle_d \sigma s \rangle$

proof (induct σs As rule: induct-2-lists012)

case *Nil* **show** ?case **by** (simp, subst $P_{SKIPs-d-rec}$, simp)

next

case (single $\sigma_0 A_0$) **show** ?case

by (auto simp add: RenamingTick- $P_{SKIPs-d-singl-list-conv-defs}$)

intro!: inj-onI $P_{SKIPs-d-eqI-strong}$ split: option.split)

next

case (Cons $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

have $\varrho\text{-disjoint-}\varepsilon : \langle A \in \text{set } (A_1 \# As) \implies \varrho\text{-disjoint-}\varepsilon A \rangle$ **for** A

by (simp add: Cons.prem(1))

have indep-enabl :

$\langle \llbracket i < \text{length } (A_1 \# As); j < \text{length } (A_1 \# As); i \neq j \rrbracket \implies$

$\text{indep-enabl } ((A_1 \# As) ! i) ((\sigma_1 \# \sigma s) ! i) E ((A_1 \# As) ! j) ((\sigma_1 \# \sigma s) ! j) \rangle$

for $i j$

by (metis Cons.prem(2) Suc-less-eq length-Cons nat.inject nth-Cons-Suc)

have $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$ **by** (simp add: Cons.prem(1))

moreover **have** $\langle \varrho\text{-disjoint-}\varepsilon \langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} A_1 \# As \rangle \rangle$

by (meson $\varrho\text{-disjoint-}\varepsilon$ $\varrho\text{-disjoint-}\varepsilon\text{-transmission-to-iterated-combine}_d\text{-Sync}_{ptick}$)

moreover **have** $\langle \text{indep-enabl } A_0 \sigma_0 E \langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} A_1 \# As \rangle (\sigma_1 \# \sigma s) \rangle$

by (metis Cons.hyps(1) Cons.prem(2) length-Cons less-Suc-eq-le same-length-indep-transmission-to-iterated-combine $_d\text{-Sync}_{ptick}$)

ultimately **show** ?case

by (simp add: $P_{SKIPs-d-combine_{Rlist-Sync_{ptick}}$

Cons.hyps(3)[OF $\varrho\text{-disjoint-}\varepsilon$ indep-enabl, simplified])

qed

corollary *P-d-compactification-Sync_{ptick}*:

$\langle \llbracket \text{length } \sigma s = \text{length } As; \wedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip } \sigma s As. P \langle A \rangle_d \sigma = P \langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} As \rangle \rangle_d \sigma s \rangle$

proof (*induct* $\sigma s As$ *rule: induct-2-lists012*)

case *Nil* **show** *?case* **by** (*simp*, *subst P-d-rec*, *simp*)

next

case (*single* $\sigma_0 A_0$) **show** *?case*

by (*simp*, *subst (1 2) P_SKIPS-d-updated- ω*)

(*auto simp add: RenamingTick-P_SKIPS-d singl-list-conv-defs*)

intro!: *inj-onI P_SKIPS-d-eqI-strong split: option.split*)

next

case (*Cons* $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

have *indep-enabl* :

$\langle \llbracket i < \text{length } (A_1 \# As); j < \text{length } (A_1 \# As); i \neq j \rrbracket \implies$

$\text{indep-enabl } ((A_1 \# As) ! i) ((\sigma_1 \# \sigma s) ! i) E ((A_1 \# As) ! j) ((\sigma_1 \# \sigma s) ! j) \rrbracket$

for *i j*

by (*metis Cons.premS Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)

have $\langle \text{indep-enabl } A_0 \sigma_0 E \langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} A_1 \# As \rangle \rangle (\sigma_1 \# \sigma s) \rangle$

by (*metis Cons.hyps(1) Cons.premS length-Cons less-Suc-eq-le same-length-indep-transmission-to-iterated-combine_d-Sync_{ptick}*)

thus *?case*

by (*simp add: P-d-combine_{Rlist}-Sync_{ptick}*)

Cons.hyps(3)[OF indep-enabl, simplified])

qed

8.3 Derived Versions

lemma *P_SKIPS-nd-compactification-Sync_{ptick}-upt-version*:

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n]. P = P_{SKIPS} \langle \langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rangle \rangle \rangle_{nd}$
(*replicate n 0*)

if $\langle \wedge i. i < n \implies \varrho\text{-disjoint-}\varepsilon (A i) \rangle$

$\langle \wedge i. i < n \implies P_{SKIPS} \langle A i \rangle_{nd} 0 = Q i \rangle$

proof –

have $\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n]. P = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n]. Q i \rangle$

by (*auto intro: mono-MultiSync_{ptick}-eq2*)

also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n]. P_{SKIPS} \langle A i \rangle_{nd} 0 \rangle$

by (*auto simp add: that(2) intro: mono-MultiSync_{ptick}-eq*)

also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip } (\text{replicate } n 0) (\text{map } A [0..<n]). P_{SKIPS} \langle A \rangle_{nd} \sigma \rangle$

proof (*induct n rule: nat-induct-012*)

case (*Suc k*)

have $\langle \llbracket E \rrbracket_{\checkmark} i \in @ [0..<\text{Suc } k]. P_{SKIPS} \langle A i \rangle_{nd} 0 =$

$\llbracket E \rrbracket_{\checkmark} i \in @ [0..<k]. P_{SKIPS} \langle A i \rangle_{nd} 0 \llbracket E \rrbracket_{\checkmark} \text{Llist } P_{SKIPS} \langle A k \rangle_{nd} 0 \rangle$

using *Suc.hyps(1)* **by** (*simp add: MultiSync_{ptick}-snoc*)

also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip } (\text{replicate } k 0) (\text{map } A [0..<k]).$

$PSKIPS\langle A \rangle_{nd} \sigma \llbracket E \rrbracket_{\checkmark}^{Llist} PSKIPS\langle A \ k \rangle_{nd} 0 \rangle$
 by (*simp only: Suc.hyps(2)*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip} (\text{replicate} (Suc\ k) 0) (\text{map } A [0..<Suc\ k]) . PSKIPS\langle A \rangle_{nd} \sigma \rangle$
 using *Suc.hyps(1)* by (*simp flip: replicate-append-same add: MultiSync_{ptick}-snoc*)
 finally show *?case* .
 qed *simp-all*
 also have $\langle \dots = PSKIPS\langle \langle_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rangle \rangle_{nd} (\text{replicate } n\ 0) \rangle$
 by (*rule PSKIPS-nd-compactification-Sync_{ptick}*) (*auto simp add: that(1)*)
 finally show *?thesis* .
 qed

lemma *P-nd-compactification-Sync_{ptick}-upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n] . P = P\langle \langle_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rangle \rangle_{nd} (\text{replicate } n\ 0) \rangle$

if $\langle \bigwedge i . i < n \implies P\langle A \ i \rangle_{nd} 0 = Q \ i \rangle$

proof –

have $\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n] . P = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n] . Q \ i \rangle$
 by (*auto intro: mono-MultiSync_{ptick}-eq2*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n] . P\langle A \ i \rangle_{nd} 0 \rangle$
 by (*auto simp add: that intro: mono-MultiSync_{ptick}-eq*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip} (\text{replicate } n\ 0) (\text{map } A [0..<n]) . P\langle A \rangle_{nd} \sigma \rangle$

proof (*induct n rule: nat-induct-012*)

case (*Suc k*)

have $\langle \llbracket E \rrbracket_{\checkmark} i \in @ [0..<Suc\ k] . P\langle A \ i \rangle_{nd} 0 = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<k] . P\langle A \ i \rangle_{nd} 0 \llbracket E \rrbracket_{\checkmark}^{Llist} P\langle A \ k \rangle_{nd} 0 \rangle$
 using *Suc.hyps(1)* by (*simp add: MultiSync_{ptick}-snoc*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip} (\text{replicate } k\ 0) (\text{map } A [0..<k]) . P\langle A \rangle_{nd} \sigma \llbracket E \rrbracket_{\checkmark}^{Llist} P\langle A \ k \rangle_{nd} 0 \rangle$
 by (*simp only: Suc.hyps(2)*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ \text{zip} (\text{replicate} (Suc\ k) 0) (\text{map } A [0..<Suc\ k]) . P\langle A \rangle_{nd} \sigma \rangle$
 using *Suc.hyps(1)* by (*simp flip: replicate-append-same add: MultiSync_{ptick}-snoc*)
 finally show *?case* .
 qed *simp-all*
 also have $\langle \dots = P\langle \langle_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rangle \rangle_{nd} (\text{replicate } n\ 0) \rangle$
 by (*rule P-nd-compactification-Sync_{ptick}*) *simp*
 finally show *?thesis* .
 qed

lemma *PSKIPS-d-compactification-Sync_{ptick}-upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n] . P = PSKIPS\langle \langle_d \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rangle \rangle_d (\text{replicate } n\ 0) \rangle$

if $\langle \bigwedge i . i < n \implies \varrho\text{-disjoint-}\varepsilon (A \ i) \rangle$

$\langle \bigwedge i \ j . i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) 0 \ E (A \ j) 0 \rangle$

$\langle \bigwedge i . i < n \implies PSKIPS\langle A \ i \rangle_d 0 = Q \ i \rangle$

proof –

have $\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n] . P = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n] . Q \ i \rangle$

by (*auto intro: mono-MultiSync_{ptick}-eq2*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n]. P_{SKIPS} \langle A \ i \rangle_d 0 \rangle$
 by (*auto simp add: that(3) intro: mono-MultiSync_{ptick}-eq*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ zip (replicate\ n\ 0) (map\ A\ [0..<n]). P_{SKIPS} \langle A \rangle_d \sigma \rangle$
proof (*induct n rule: nat-induct-012*)
 case (*Suc k*)
 have $\langle \llbracket E \rrbracket_{\checkmark} i \in @ [0..<Suc\ k]. P_{SKIPS} \langle A \ i \rangle_d 0 = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<k]. P_{SKIPS} \langle A \ i \rangle_d 0 \llbracket E \rrbracket_{\checkmark} Llist\ P_{SKIPS} \langle A \ k \rangle_d 0 \rangle$
 using *Suc.hyps(1)* by (*simp add: MultiSync_{ptick}-snoc*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ zip (replicate\ k\ 0) (map\ A\ [0..<k]). P_{SKIPS} \langle A \rangle_d \sigma \llbracket E \rrbracket_{\checkmark} Llist\ P_{SKIPS} \langle A \ k \rangle_d 0 \rangle$
 by (*simp only: Suc.hyps(2)*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ zip (replicate\ (Suc\ k)\ 0) (map\ A\ [0..<Suc\ k]). P_{SKIPS} \langle A \rangle_d \sigma \rangle$
 using *Suc.hyps(1)* by (*simp flip: replicate-append-same add: MultiSync_{ptick}-snoc*)
 finally show *?case* .
qed simp-all
 also have $\langle \dots = P_{SKIPS} \langle \langle d \otimes \llbracket E \rrbracket_{\checkmark} map\ A\ [0..<n] \rangle \rangle_d (replicate\ n\ 0) \rangle$
 by (*rule P_{SKIPS}-d-compactification-Sync_{ptick}*) (*auto simp add: that(1, 2)*)
 finally show *?thesis* .
qed

lemma *P-d-compactification-Sync_{ptick}-upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ map\ Q\ [0..<n]. P = P \langle \langle d \otimes \llbracket E \rrbracket_{\checkmark} map\ A\ [0..<n] \rangle \rangle_d (replicate\ n\ 0) \rangle$
 if $\langle \bigwedge i\ j. i < n \implies j < n \implies i \neq j \implies indep-enabl\ (A\ i)\ 0\ E\ (A\ j)\ 0 \rangle$
 $\langle \bigwedge i. i < n \implies P \langle A \ i \rangle_d 0 = Q \ i \rangle$

proof –

have $\langle \llbracket E \rrbracket_{\checkmark} P \in @ map\ Q\ [0..<n]. P = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n]. Q \ i \rangle$
 by (*auto intro: mono-MultiSync_{ptick}-eq2*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<n]. P \langle A \ i \rangle_d 0 \rangle$
 by (*auto simp add: that intro: mono-MultiSync_{ptick}-eq*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ (zip (replicate\ n\ 0) (map\ A\ [0..<n])). P \langle A \rangle_d \sigma \rangle$
proof (*induct n rule: nat-induct-012*)
 case (*Suc k*)
 have $\langle \llbracket E \rrbracket_{\checkmark} i \in @ [0..<Suc\ k]. P \langle A \ i \rangle_d 0 = \llbracket E \rrbracket_{\checkmark} i \in @ [0..<k]. P \langle A \ i \rangle_d 0 \llbracket E \rrbracket_{\checkmark} Llist\ P \langle A \ k \rangle_d 0 \rangle$
 using *Suc.hyps(1)* by (*simp add: MultiSync_{ptick}-snoc*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ zip (replicate\ k\ 0) (map\ A\ [0..<k]). P \langle A \rangle_d \sigma \llbracket E \rrbracket_{\checkmark} Llist\ P \langle A \ k \rangle_d 0 \rangle$
 by (*simp only: Suc.hyps(2)*)
 also have $\langle \dots = \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in @ zip (replicate\ (Suc\ k)\ 0) (map\ A\ [0..<Suc\ k]). P \langle A \rangle_d \sigma \rangle$
 using *Suc.hyps(1)* by (*simp flip: replicate-append-same add: MultiSync_{ptick}-snoc*)
 finally show *?case* .
qed simp-all
 also have $\langle \dots = P \langle \langle d \otimes \llbracket E \rrbracket_{\checkmark} map\ A\ [0..<n] \rangle \rangle_d (replicate\ n\ 0) \rangle$

by (rule *P-d-compactification-Sync_{ptick}*) (auto simp add: that(1))
finally show ?thesis .
qed

8.4 More on Iterated Combine and Events

Through τ -iterated-combine-Sync_{ptick} ε -iterated-combine-Sync_{ptick} \mathcal{R} -iterated-combine-Sync_{ptick},
we immediately recover the results proven in *HOL-CSP-Proc-Omata.Compactification-Synchronization-F*

Chapter 9

Combining Automata for Sequential Composition Generalized

9.1 Definitions

9.2 General Patterns

definition *combine_d-Seq-ε* ::

$\langle [(\sigma_0, 'a, 'r, '\alpha_0) A_d\text{-scheme}, 'r \Rightarrow (\sigma_1, 'a, 's, '\alpha_1) A_d\text{-scheme}, '\sigma \Rightarrow \sigma_0, '\sigma \Rightarrow \sigma_1, '\sigma] \Rightarrow 'a \text{ set} \rangle$

where $\langle \text{combine}_d\text{-Seq-}\varepsilon A_0 A_1 i_0 i_1 \sigma s \equiv$
 if $i_0 \sigma s \in \varrho A_0$
 then if $i_1 \sigma s \in \varrho (A_1 [\omega A_0 (i_0 \sigma s)])$
 then {}
 else $\varepsilon (A_1 [\omega A_0 (i_0 \sigma s)]) (i_1 \sigma s)$
 else $\varepsilon A_0 (i_0 \sigma s) \rangle$

definition *combine_{nd}-Seq-ε* ::

$\langle [(\sigma_0, 'a, 'r, '\alpha_0) A_{nd}\text{-scheme}, 'r \Rightarrow (\sigma_1, 'a, 's, '\alpha_1) A_{nd}\text{-scheme}, '\sigma \Rightarrow \sigma_0, '\sigma \Rightarrow \sigma_1, '\sigma] \Rightarrow 'a \text{ set} \rangle$

where $\langle \text{combine}_{nd}\text{-Seq-}\varepsilon A_0 A_1 i_0 i_1 \sigma s \equiv$
 if $i_0 \sigma s \in \varrho A_0$
 then if $i_1 \sigma s \in (\bigcup r \in \omega A_0 (i_0 \sigma s). \varrho (A_1 r))$
 then {}
 else $(\bigcup r \in \omega A_0 (i_0 \sigma s). \varepsilon (A_1 r) (i_1 \sigma s))$
 else $\varepsilon A_0 (i_0 \sigma s) \rangle$

lemmas *combine-Seq-ε-defs* = *combine_d-Seq-ε-def combine_{nd}-Seq-ε-def*

fun *combine_d-Seq* ::

$\langle [(\sigma_0, 'e, 'r, '\alpha_0) A_d\text{-scheme}, 'r \Rightarrow (\sigma_1, 'e, 's, '\alpha_1) A_d\text{-scheme},$

$\langle ' \sigma \Rightarrow ' \sigma_0, ' \sigma \Rightarrow ' \sigma_1, ' \sigma_0 \Rightarrow ' \sigma_1 \Rightarrow ' \sigma \rangle \Rightarrow (' \sigma, ' e, ' s) A_d \rangle$
and *combine_{nd}-Seq* ::
 $\langle [(' \sigma_0, ' e, ' r, ' \alpha_0) A_{nd}\text{-scheme}, ' r \Rightarrow (' \sigma_1, ' e, ' s, ' \alpha_1) A_{nd}\text{-scheme}, ' \sigma \Rightarrow ' \sigma_0, ' \sigma \Rightarrow ' \sigma_1, ' \sigma_0 \Rightarrow ' \sigma_1 \Rightarrow ' \sigma] \Rightarrow (' \sigma, ' e, ' s) A_{nd} \rangle$
where $\langle \text{combine}_d\text{-Seq } A_0 A_1 i_0 i_1 f =$
 $(\tau = \lambda \sigma s e. \text{ if } i_0 \sigma s \in \varrho A_0$
 $\text{ then if } i_1 \sigma s \in \varrho (A_1 [\omega A_0 (i_0 \sigma s)])$
 $\text{ then } \diamond$
 $\text{ else update-right } (A_1 [\omega A_0 (i_0 \sigma s)]) (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-opt}$
 $f) (\lambda \sigma. [\sigma])$
 $\text{ else update-left } A_0 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-opt } f) (\lambda \sigma. [\sigma]),$
 $\omega = \lambda \sigma s. \text{ case } \omega A_0 (i_0 \sigma s) \text{ of } \diamond \Rightarrow \diamond \mid [r] \Rightarrow \omega (A_1 r) (i_1 \sigma s) \rangle$
 $\mid \langle \text{combine}_{nd}\text{-Seq } A_0 A_1 i_0 i_1 f =$
 $(\tau = \lambda \sigma s e. \text{ if } i_0 \sigma s \in \varrho A_0$
 $\text{ then if } i_1 \sigma s \in (\bigcup r \in \omega A_0 (i_0 \sigma s). \varrho (A_1 r))$
 $\text{ then } \{ \}$
 $\text{ else } (\bigcup r \in \omega A_0 (i_0 \sigma s). \text{ update-right } (A_1 r) (i_0 \sigma s) (i_1 \sigma s) e$
 $(f\text{-up-set } f) (\lambda \sigma. \{ \sigma \}))$
 $\text{ else update-left } A_0 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) (\lambda \sigma. \{ \sigma \}),$
 $\omega = \lambda \sigma s. \bigcup r \in \omega A_0 (i_0 \sigma s). \omega (A_1 r) (i_1 \sigma s) \rangle$

9.3 Specializations

definition *combine_dPairlist-Seq_{ptick}* ::
 $\langle [(' \sigma, ' e, ' r, ' \alpha) A_d\text{-scheme}, ' r \Rightarrow (' \sigma, ' e, ' s, ' \beta) A_d\text{-scheme}] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_d \rangle$
where $\langle \text{combine}_d\text{Pairlist-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_d\text{-Seq } A_0 A_1 \text{ hd } (\lambda \sigma s. \text{hd } (tl \sigma s)) (\lambda s t. [s, t]) \rangle$

definition *combine_{nd}Pairlist-Seq_{ptick}* ::
 $\langle [(' \sigma, ' e, ' r, ' \alpha) A_{nd}\text{-scheme}, ' r \Rightarrow (' \sigma, ' e, ' s, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_{nd} \rangle$
where $\langle \text{combine}_{nd}\text{Pairlist-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_{nd}\text{-Seq } A_0 A_1 \text{ hd } (\lambda \sigma s. \text{hd } (tl \sigma s)) (\lambda s t. [s, t]) \rangle$

definition *combine_dPair-Seq_{ptick}* ::
 $\langle [(' \sigma_0, ' e, ' r, ' \alpha) A_d\text{-scheme}, ' r \Rightarrow (' \sigma_1, ' e, ' s, ' \beta) A_d\text{-scheme}] \Rightarrow (' \sigma_0 \times ' \sigma_1, ' e, ' s) A_d \rangle$
where $\langle \text{combine}_d\text{Pair-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_d\text{-Seq } A_0 A_1 \text{ fst snd Pair} \rangle$

definition *combine_{nd}Pair-Seq_{ptick}* ::
 $\langle [(' \sigma_0, ' e, ' r, ' \alpha) A_{nd}\text{-scheme}, ' r \Rightarrow (' \sigma_1, ' e, ' s, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma_0 \times ' \sigma_1, ' e, ' s) A_{nd} \rangle$
where $\langle \text{combine}_{nd}\text{Pair-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_{nd}\text{-Seq } A_0 A_1 \text{ fst snd Pair} \rangle$

definition *combine_dListslenL-Seq_{ptick}* ::
 $\langle [(' \sigma \text{ list}, ' e, ' r, ' \alpha) A_d\text{-scheme}, \text{ nat}, ' r \Rightarrow (' \sigma \text{ list}, ' e, ' s, ' \beta) A_d\text{-scheme}] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_d \rangle$
where $\langle \text{combine}_d\text{ListslenL-Seq}_{ptick} A_0 \text{ len}_0 A_1 \equiv \text{combine}_d\text{-Seq } A_0 A_1 (\text{take len}_0) (\text{drop len}_0) (@) \rangle$

definition *combine_{nd}ListslenL-Seq_{ptick}* ::
 $\langle [(' \sigma \text{ list}, ' e, ' r, ' \alpha) A_{nd}\text{-scheme}, \text{ nat}, ' r \Rightarrow (' \sigma \text{ list}, ' e, ' s, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_{nd} \rangle$

list, 'e, 's) A_{nd}
where *⟨combine_{ndListslenL-Seqptick} A₀ len₀ A₁ ≡ combine_{nd-Seq} A₀ A₁ (take len₀) (drop len₀) (@)⟩*

definition *combine_{dRlist-Seqptick} ::*
⟨[(('σ, 'e, 'r, 'α) A_d-scheme, 'r ⇒ ('σ list, 'e, 's, 'β) A_d-scheme) ⇒ ('σ list, 'e, 's) A_d⟩

where *⟨combine_{dRlist-Seqptick} A₀ A₁ ≡ combine_{d-Seq} A₀ A₁ hd tl (#)⟩*

definition *combine_{ndRlist-Seqptick} ::*
⟨[(('σ, 'e, 'r, 'α) A_{nd}-scheme, 'r ⇒ ('σ list, 'e, 's, 'β) A_{nd}-scheme) ⇒ ('σ list, 'e, 's) A_{nd}⟩

where *⟨combine_{ndRlist-Seqptick} A₀ A₁ ≡ combine_{nd-Seq} A₀ A₁ hd tl (#)⟩*

lemmas *combine_{Pairlist-Seqptick-defs} = combine_{dPairlist-Seqptick-def} combine_{ndPairlist-Seqptick-def}*
and *combine_{Pair-Seqptick-defs} = combine_{dPair-Seqptick-def} combine_{ndPair-Seqptick-def}*
and *combine_{ListslenL-Seqptick} = combine_{dListslenL-Seqptick-def} combine_{ndListslenL-Seqptick-def}*
and *combine_{Rlist-Seqptick-defs} = combine_{dRlist-Seqptick-def} combine_{ndRlist-Seqptick-def}*

lemmas *combine-Seq-defs =*
combine_{Pairlist-Seqptick-defs} combine_{Pair-Seqptick-defs} combine_{ListslenL-Seqptick}
combine_{Rlist-Seqptick-defs}

bundle *combine-Seq-syntax begin*

notation *combine_{dPairlist-Seqptick} (⟨⟨- d⊗;✓_{Pairlist} -⟩⟩ [0, 0])*
notation *combine_{ndPairlist-Seqptick} (⟨⟨- nd⊗;✓_{Pairlist} -⟩⟩ [0, 0])*
notation *combine_{dPair-Seqptick} (⟨⟨- d⊗;✓_{Pair} -⟩⟩ [0, 0])*
notation *combine_{ndPair-Seqptick} (⟨⟨- nd⊗;✓_{Pair} -⟩⟩ [0, 0])*
notation *combine_{dListslenL-Seqptick} (⟨⟨- d⊗;ListslenL -⟩⟩ [0, 0, 0])*
notation *combine_{ndListslenL-Seqptick} (⟨⟨- nd⊗;ListslenL -⟩⟩ [0, 0, 0])*
notation *combine_{dRlist-Seqptick} (⟨⟨- d⊗;✓_{Rlist} -⟩⟩ [0, 0])*
notation *combine_{ndRlist-Seqptick} (⟨⟨- nd⊗;✓_{Rlist} -⟩⟩ [0, 0])*

end

unbundle *combine-Seq-syntax*

9.4 First Properties

lemma *ε-combine_{Pairlist-Seqptick} :*
⟨ε ⟨A₀ d⊗;✓_{Pairlist} A₁⟩ σs = combine_{d-Seq-ε} A₀ A₁ hd (hd ∘ tl) σs⟩
⟨ε ⟨B₀ nd⊗;✓_{Pairlist} B₁⟩ σs = combine_{nd-Seq-ε} B₀ B₁ hd (hd ∘ tl) σs⟩
by *(auto simp add: combine-Seq-ε-defs combine_{Pairlist-Seqptick-defs}*
ε-simps ρ-simps option.case-eq-if)

lemma *ε-combine_{Pair-Seqptick} :*

$\langle \varepsilon \langle A_0 \text{ d} \otimes; \checkmark_{Pair} A_1 \rangle \sigma s = \text{combine}_d\text{-Seq-}\varepsilon A_0 A_1 \text{ fst snd } \sigma s \rangle$
 $\langle \varepsilon \langle B_0 \text{ nd} \otimes; \checkmark_{Pair} B_1 \rangle \sigma s = \text{combine}_{nd}\text{-Seq-}\varepsilon B_0 B_1 \text{ fst snd } \sigma s \rangle$
by (*auto simp add: combine-Seq- ε -defs combine_{Pair}-Seq_{ptick}-defs*
 ε -simps ϱ -simps *option.case-eq-if*)

lemma ε -combine_{ListLenL}-Seq_{ptick} :

$\langle \varepsilon \langle A_0 \text{ d} \otimes \text{ len}_0; \text{ListLenL } A_1 \rangle \sigma s = \text{combine}_d\text{-Seq-}\varepsilon A_0 A_1 \text{ (take len}_0) \text{ (drop len}_0) \sigma s \rangle$

$\langle \varepsilon \langle B_0 \text{ nd} \otimes \text{ len}_0; \text{ListLenL } B_1 \rangle \sigma s = \text{combine}_{nd}\text{-Seq-}\varepsilon B_0 B_1 \text{ (take len}_0) \text{ (drop len}_0) \sigma s \rangle$

by (*auto simp add: combine-Seq- ε -defs combine_{ListLenL}-Seq_{ptick}*
 ε -simps ϱ -simps *option.case-eq-if*)

lemma ε -combine_{RList}-Seq_{ptick} :

$\langle \varepsilon \langle A_0 \text{ d} \otimes; \checkmark_{RList} A_1 \rangle \sigma s = \text{combine}_d\text{-Seq-}\varepsilon A_0 A_1 \text{ hd tl } \sigma s \rangle$

$\langle \varepsilon \langle B_0 \text{ nd} \otimes; \checkmark_{RList} B_1 \rangle \sigma s = \text{combine}_{nd}\text{-Seq-}\varepsilon B_0 B_1 \text{ hd tl } \sigma s \rangle$

by (*auto simp add: combine-Seq- ε -defs combine_{RList}-Seq_{ptick}-defs*
 ε -simps ϱ -simps *option.case-eq-if*)

9.4.1 Reachability

lemma \mathcal{R}_d -combine_{ListLenL}-Seq_{ptick}-subset:

$\langle \mathcal{R}_d \langle A_0 \text{ d} \otimes \text{ len}_0; \text{ListLenL } A_1 \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$ (**is**
 $\langle ?S_A \subseteq \cdot \rangle$)

if *same-length- \mathcal{R}_d* : $\langle \bigwedge t_0. t_0 \in \mathcal{R}_d A_0 s_0 \implies \text{length } t_0 = \text{len}_0 \rangle$

proof safe

show $\langle t \in ?S_A \implies \exists t_0 t_1. t = t_0 @ t_1 \wedge t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ **for** t

proof (*induct rule: \mathcal{R}_d .induct*)

case *init* **show** $?case$ **by** (*metis \mathcal{R}_d .init*)

next

case (*step* $\sigma' \sigma'' a$)

from *step.hyps*(2) *same-length- \mathcal{R}_d* **obtain** $t_0 t_1$

where $\langle \sigma' = t_0 @ t_1 \rangle \langle t_0 \in \mathcal{R}_d A_0 s_0 \rangle \langle \text{length } t_0 = \text{len}_0 \rangle$ **by** *blast*

with *step.hyps*(3) **show** $?case$

by (*auto simp add: combine_{ListLenL}-Seq_{ptick} map-option-case*
split: if-split-asm option.splits) (*metis \mathcal{R}_d .step*)

qed

qed

lemma \mathcal{R}_{nd} -combine_{ndListLenL}-Seq_{ptick}-subset:

$\langle \mathcal{R}_{nd} \langle A_0 \text{ nd} \otimes \text{ len}_0; \text{ListLenL } A_1 \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} A_0 s_0\} \rangle$
(**is** $\langle ?S_A \subseteq \cdot \rangle$)

if *same-length- \mathcal{R}_{nd}* : $\langle \bigwedge t_0. t_0 \in \mathcal{R}_{nd} A_0 s_0 \implies \text{length } t_0 = \text{len}_0 \rangle$

proof safe

show $\langle t \in ?S_A \implies \exists t_0 t_1. t = t_0 @ t_1 \wedge t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ **for** t

proof (*induct rule: \mathcal{R}_{nd} .induct*)

case *init* **show** $?case$ **by** (*metis \mathcal{R}_{nd} .init*)

next

case (*step* $\sigma' \sigma'' a$)

from *step.hyps(2)* *same-length- \mathcal{R}_{nd}* **obtain** $t_0 t_1$
where $\langle \sigma' = t_0 @ t_1 \rangle \langle t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle \langle \text{length } t_0 = \text{len}_0 \rangle$ **by** *blast*
with *step.hyps(3)* **show** *?case*
by (*auto simp add: combine_{ListsLenL-Seqptick} split: if-split-asm*) (*metis*
 $\mathcal{R}_{nd}.\text{step}$)
qed
qed

lemma *\mathcal{R}_d -combine_{dPairlist-Seqptick}-subset*:
 $\langle \mathcal{R}_d \langle A_0 d \otimes; \checkmark_{Pairlist} A_1 \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$ (**is** $\langle ?S_A \subseteq - \rangle$)
proof *safe*
show $\langle t \in ?S_A \implies \exists t_0 t_1. t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ **for** t
proof (*induct rule: $\mathcal{R}_d.\text{induct}$*)
case *init* **show** *?case* **by** (*metis $\mathcal{R}_d.\text{init}$*)
next
case (*step $\sigma' \sigma'' a$*)
from *step.hyps(2)* **obtain** $t_0 t_1$ **where** $\langle \sigma' = [t_0, t_1] \rangle \langle t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ **by** *blast*
with *step.hyps(3)* **show** *?case*
by (*auto simp add: combine_{Pairlist-Seqptick-defs} map-option-case split: if-split-asm option.split-asm*) (*metis $\mathcal{R}_d.\text{step}$*)
qed
qed

lemma *\mathcal{R}_{nd} -combine_{ndPairlist-Seqptick}-subset*:
 $\langle \mathcal{R}_{nd} \langle A_0 nd \otimes; \checkmark_{Pairlist} A_1 \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} A_0 s_0\} \rangle$ (**is** $\langle ?S_A \subseteq - \rangle$)
proof *safe*
show $\langle t \in ?S_A \implies \exists t_0 t_1. t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ **for** t
proof (*induct rule: $\mathcal{R}_{nd}.\text{induct}$*)
case *init* **show** *?case* **by** (*metis $\mathcal{R}_{nd}.\text{init}$*)
next
case (*step $\sigma' \sigma'' a$*)
from *step.hyps(2)* **obtain** $t_0 t_1$ **where** $\langle \sigma' = [t_0, t_1] \rangle \langle t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ **by**
blast
with *step.hyps(3)* **show** *?case*
by (*auto simp add: combine_{Pairlist-Seqptick-defs} split: if-split-asm*) (*metis*
 $\mathcal{R}_{nd}.\text{step}$)
qed
qed

lemma *\mathcal{R}_d -combine_{dPair-Seqptick}-subset*:
 $\langle \mathcal{R}_d \langle A_0 d \otimes; \checkmark_{Pair} A_1 \rangle (s_0, s_1) \subseteq \{(t_0, t_1) \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$ (**is** $\langle ?S_A \subseteq - \rangle$)
proof *safe*
show $\langle t \in ?S_A \implies \exists t_0 t_1. t = (t_0, t_1) \wedge t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ **for** t
proof (*induct rule: $\mathcal{R}_d.\text{induct}$*)

case *init* show ?case by (metis $\mathcal{R}_d.init$)
 next
 case (step $\sigma' \sigma'' a$)
 from step.hyps(2) obtain $t_0 t_1$ where $\langle \sigma' = (t_0, t_1) \rangle \langle t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ by blast
 with step.hyps(3) show ?case
 by (auto simp add: combine_{Pair}-Seq_{ptick}-defs map-option-case
 split: if-split-asm option.split-asm) (metis $\mathcal{R}_d.step$)
 qed
 qed

lemma \mathcal{R}_{nd} -combine_{ndPair}-Seq_{ptick}-subset:
 $\langle \mathcal{R}_{nd} \langle A_0 \text{ nd} \otimes; \checkmark_{Pair} A_1 \rangle (s_0, s_1) \subseteq \{(t_0, t_1) \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} A_0 s_0\} \rangle$ (is $\langle ?S_A \subseteq - \rangle$)
proof safe
 show $\langle t \in ?S_A \implies \exists t_0 t_1. t = (t_0, t_1) \wedge t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ for t
proof (induct rule: $\mathcal{R}_{nd}.induct$)
 case *init* show ?case by (metis $\mathcal{R}_{nd}.init$)
 next
 case (step $\sigma' \sigma'' a$)
 from step.hyps(2) obtain $t_0 t_1$ where $\langle \sigma' = (t_0, t_1) \rangle \langle t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ by blast
 with step.hyps(3) show ?case
 by (auto simp add: combine_{Pair}-Seq_{ptick}-defs split: if-split-asm) (metis $\mathcal{R}_{nd}.step$)
 qed
 qed

lemma \mathcal{R}_d -combine_{dRlist}-Seq_{ptick}-subset:
 $\langle \mathcal{R}_d \langle A_0 \text{ d} \otimes; \checkmark_{Rlist} A_1 \rangle (s_0 \# s_1) \subseteq \{t_0 \# t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$ (is $\langle ?S_A \subseteq - \rangle$)
proof safe
 show $\langle t \in ?S_A \implies \exists t_0 t_1. t = t_0 \# t_1 \wedge t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ for t
proof (induct rule: $\mathcal{R}_d.induct$)
 case *init* show ?case by (metis $\mathcal{R}_d.init$)
 next
 case (step $\sigma' \sigma'' a$)
 from step.hyps(2) obtain $t_0 t_1$ where $\langle \sigma' = t_0 \# t_1 \rangle \langle t_0 \in \mathcal{R}_d A_0 s_0 \rangle$ by blast
 with step.hyps(3) show ?case
 by (auto simp add: combine_{Rlist}-Seq_{ptick}-defs map-option-case
 split: if-split-asm option.split-asm) (metis $\mathcal{R}_d.step$)
 qed
 qed

lemma \mathcal{R}_{nd} -combine_{ndRlist}-Seq_{ptick}-subset:
 $\langle \mathcal{R}_{nd} \langle A_0 \text{ nd} \otimes; \checkmark_{Rlist} A_1 \rangle (s_0 \# s_1) \subseteq \{t_0 \# t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} A_0 s_0\} \rangle$ (is $\langle ?S_A \subseteq - \rangle$)
proof safe
 show $\langle t \in ?S_A \implies \exists t_0 t_1. t = t_0 \# t_1 \wedge t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ for t

proof (*induct rule: $\mathcal{R}_{nd}.induct$*)
case *init* **show** *?case* **by** (*metis $\mathcal{R}_{nd}.init$*)
next
case (*step $\sigma' \sigma'' a$*)
from *step.hyps(2)* **obtain** $t_0 t_1$ **where** $\langle \sigma' = t_0 \# t_1 \rangle \langle t_0 \in \mathcal{R}_{nd} A_0 s_0 \rangle$ **by**
blast
with *step.hyps(3)* **show** *?case*
by (*auto simp add: combine $_{Rlist-Seqptick-defs}$ split: if-split-asm*) (*metis $\mathcal{R}_{nd}.step$*)
qed
qed

9.5 Normalization

lemma τ -combine $_{Pairlist-Seqptick}$ -behaviour:

$\langle \tau \langle \langle A_0 \ d \otimes ; \checkmark_{Pairlist} A_1 \rangle \rangle_{d \hookrightarrow nd} [s_0, s_1] e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes ; \checkmark_{Pairlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle [s_0, s_1] e \rangle$
by (*auto simp add: combine $_{Pairlist-Seqptick-defs}$ det-ndet-conv-defs option.case-eq-if ε -simps ϱ -simps*)

lemma τ -combine $_{Pair-Seqptick}$ -behaviour:

$\langle \tau \langle \langle A_0 \ d \otimes ; \checkmark_{Pair} A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0, s_1) e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes ; \checkmark_{Pair} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle (s_0, s_1) e \rangle$
by (*auto simp add: combine $_{Pair-Seqptick-defs}$ det-ndet-conv-defs option.case-eq-if ε -simps ϱ -simps*)

lemma τ -combine $_{ListslenL-Seqptick}$ -behaviour:

$\langle \tau \langle \langle A_0 \ d \otimes len_0 ; ListslenL A_1 \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 @ \sigma s_1) e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes len_0 ; ListslenL (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle (\sigma s_0 @ \sigma s_1) e \rangle$
by (*auto simp add: combine $_{ListslenL-Seqptick}$ det-ndet-conv-defs option.case-eq-if ε -simps ϱ -simps*)

lemma τ -combine $_{Rlist-Seqptick}$ -behaviour:

$\langle \tau \langle \langle A_0 \ d \otimes ; \checkmark_{Rlist} A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0 \# \sigma s_1) e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes ; \checkmark_{Rlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle (s_0 \# \sigma s_1) e \rangle$
by (*auto simp add: combine $_{Rlist-Seqptick-defs}$ det-ndet-conv-defs option.case-eq-if ε -simps ϱ -simps*)

Behaviour of normalisations

lemma P_{SKIPS} -combine $_{Pairlist-Seqptick}$ -behaviour:

$\langle P_{SKIPS} \langle \langle A_0 \ d \otimes ; \checkmark_{Pairlist} A_1 \rangle \rangle_d [s_0, s_1] = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes ; \checkmark_{Pairlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle_{nd} [s_0, s_1] \rangle$
by (*fold P_{SKIPS} -nd-from-det-to-ndet-is- P_{SKIPS} -d, rule P_{SKIPS} -nd-eqI-strong-id, unfold \mathcal{R}_{nd} -from-det-to-ndet*)
(all $\langle drule \ set\ mp[OF \ \mathcal{R}_d$ -combine $_d$ Pairlist-Seqptick-subset] \rangle,
auto simp add: combine $_{Pairlist-Seqptick-defs}$ from-det-to-ndet-def ϱ -simps split: option.split)

lemma P -combine $_{Pairlist-Seqptick}$ -behaviour:

$\langle P\langle\langle A_0 \text{ } d\otimes; \checkmark_{Pairlist} A_1 \rangle\rangle_d [s_0, s_1] = P\langle\langle\langle A_0 \rangle_{d\hookrightarrow nd} \text{ } nd\otimes; \checkmark_{Pairlist} (\lambda r. \langle A_1 r \rangle_{d\hookrightarrow nd}) \rangle\rangle_{nd} [s_0, s_1] \rangle$
by (fold *P-nd-from-det-to-ndet-is-P-d*,
rule P-nd-eqI-strong-id, unfold \mathcal{R}_{nd} -from-det-to-ndet)
(drule set-mp[OF \mathcal{R}_d -combine $_d$ Pairlist-Seqptick-subset],
auto simp add: combine $_{Pairlist}$ -Seqptick-defs from-det-to-ndet-def ϱ -simps
split: option.split)

lemma *P $_{SKIPS}$ -combine $_{Pair}$ -Seqptick-behaviour:*

$\langle P_{SKIPS}\langle\langle A_0 \text{ } d\otimes; \checkmark_{Pair} A_1 \rangle\rangle_d (s_0, s_1) = P_{SKIPS}\langle\langle\langle A_0 \rangle_{d\hookrightarrow nd} \text{ } nd\otimes; \checkmark_{Pair} (\lambda r. \langle A_1 r \rangle_{d\hookrightarrow nd}) \rangle\rangle_{nd} (s_0, s_1) \rangle$
by (fold *P $_{SKIPS}$ -nd-from-det-to-ndet-is-P $_{SKIPS}$ -d*,
rule P $_{SKIPS}$ -nd-eqI-strong-id, unfold \mathcal{R}_{nd} -from-det-to-ndet)
(all \langle drule set-mp[OF \mathcal{R}_d -combine $_d$ Pair-Seqptick-subset],
auto simp add: combine $_{Pair}$ -Seqptick-defs from-det-to-ndet-def ϱ -simps split:
option.split)

lemma *P-combine $_{Pair}$ -Seqptick-behaviour:*

$\langle P\langle\langle A_0 \text{ } d\otimes; \checkmark_{Pair} A_1 \rangle\rangle_d (s_0, s_1) = P\langle\langle\langle A_0 \rangle_{d\hookrightarrow nd} \text{ } nd\otimes; \checkmark_{Pair} (\lambda r. \langle A_1 r \rangle_{d\hookrightarrow nd}) \rangle\rangle_{nd} (s_0, s_1) \rangle$
by (fold *P-nd-from-det-to-ndet-is-P-d*,
rule P-nd-eqI-strong-id, unfold \mathcal{R}_{nd} -from-det-to-ndet)
(drule set-mp[OF \mathcal{R}_d -combine $_d$ Pair-Seqptick-subset],
auto simp add: combine $_{Pair}$ -Seqptick-defs from-det-to-ndet-def ϱ -simps split:
option.split)

lemma *P $_{SKIPS}$ -combine $_{Rlist}$ -Seqptick-behaviour:*

$\langle P_{SKIPS}\langle\langle A_0 \text{ } d\otimes; \checkmark_{Rlist} A_1 \rangle\rangle_d (s_0 \# s_1) = P_{SKIPS}\langle\langle\langle A_0 \rangle_{d\hookrightarrow nd} \text{ } nd\otimes; \checkmark_{Rlist} (\lambda r. \langle A_1 r \rangle_{d\hookrightarrow nd}) \rangle\rangle_{nd} (s_0 \# s_1) \rangle$
by (fold *P $_{SKIPS}$ -nd-from-det-to-ndet-is-P $_{SKIPS}$ -d*,
rule P $_{SKIPS}$ -nd-eqI-strong-id, unfold \mathcal{R}_{nd} -from-det-to-ndet)
(all \langle drule set-mp[OF \mathcal{R}_d -combine $_d$ Rlist-Seqptick-subset],
auto simp add: combine $_{Rlist}$ -Seqptick-defs from-det-to-ndet-def ϱ -simps split:
option.split)

lemma *P-combine $_{Rlist}$ -Seqptick-behaviour:*

$\langle P\langle\langle A_0 \text{ } d\otimes; \checkmark_{Rlist} A_1 \rangle\rangle_d (s_0 \# s_1) = P\langle\langle\langle A_0 \rangle_{d\hookrightarrow nd} \text{ } nd\otimes; \checkmark_{Rlist} (\lambda r. \langle A_1 r \rangle_{d\hookrightarrow nd}) \rangle\rangle_{nd} (s_0 \# s_1) \rangle$
by (fold *P-nd-from-det-to-ndet-is-P-d*,
rule P-nd-eqI-strong-id, unfold \mathcal{R}_{nd} -from-det-to-ndet)
(drule set-mp[OF \mathcal{R}_d -combine $_d$ Rlist-Seqptick-subset],
auto simp add: combine $_{Rlist}$ -Seqptick-defs from-det-to-ndet-def ϱ -simps split:
option.split)

lemma *P $_{SKIPS}$ -combine $_{ListslenL}$ -Seqptick-behaviour:*

$\langle P_{SKIPS}\langle\langle A_0 \text{ } d\otimes len_0; ListslenL A_1 \rangle\rangle_d (\sigma s_0 @ \sigma s_1) = P_{SKIPS}\langle\langle\langle A_0 \rangle_{d\hookrightarrow nd}$

$nd \otimes len_0; ListslenL (\lambda r. \langle \langle A_1 r \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 @ \sigma s_1))$
if $\langle \bigwedge \sigma s_0'. \sigma s_0' \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma s_0' = len_0 \rangle$
by (fold $P_{SKIPS-nd-from-det-to-ndet-is-P_{SKIPS-d}$,
rule $P_{SKIPS-nd-eqI-strong-id}$, unfold $\mathcal{R}_{nd-from-det-to-ndet}$)
(all $\langle drule set-mp[OF \mathcal{R}_d-combine_{ListslenL-Seq_{ptick-subset}}[OF that], rotated] \rangle$,
auto simp add: $combine_{ListslenL-Seq_{ptick}}$ from-det-to-ndet-def ρ -simps split:
option.split)

lemma $P-combine_{ListslenL-Seq_{ptick}}$ -behaviour:

$\langle P \langle \langle A_0 d \otimes len_0; ListslenL A_1 \rangle \rangle_d (\sigma s_0 @ \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd \otimes len_0; ListslenL} (\lambda r. \langle \langle A_1 r \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 @ \sigma s_1)) \rangle$
if $\langle \bigwedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma t_0 = len_0 \rangle$
by (fold $P-nd-from-det-to-ndet-is-P-d$,
rule $P-nd-eqI-strong-id$, unfold $\mathcal{R}_{nd-from-det-to-ndet}$)
(drule set-mp[OF $\mathcal{R}_d-combine_{ListslenL-Seq_{ptick-subset}}[OF that], rotated]$,
auto simp add: $combine_{ListslenL-Seq_{ptick}}$ from-det-to-ndet-def ρ -simps split:
option.split)

Chapter 10

Compactification of Sequential Composition Generalized

10.1 Iterated Combine

10.1.1 Definitions

fun *iterated-combine_d-Seq_{ptick}* :: $\langle [('r \Rightarrow (' \sigma, 'e, 'r) A_d) \textit{list}, 'r] \Rightarrow (' \sigma \textit{list}, 'e, 'r) A_d \rangle$ ($\langle \langle d \otimes ; \checkmark - \rangle \rangle [0]$)
where $\langle \langle d \otimes ; \checkmark [] \rangle \rangle r = \langle \tau = \lambda \sigma s a. \diamond, \omega = \lambda \sigma s. [r] \rangle$
 $| \quad \langle \langle d \otimes ; \checkmark [A_0] \rangle \rangle r = d \langle \langle A_0 r \rangle \rangle_{\sigma \mapsto \sigma s}$
 $| \quad \langle \langle d \otimes ; \checkmark A_0 \# A_1 \# A_s \rangle \rangle r = \langle \langle A_0 r \ d \otimes ; \checkmark Rlist \ \langle \langle d \otimes ; \checkmark A_1 \# A_s \rangle \rangle \rangle$

fun *iterated-combine_{nd}-Seq_{ptick}* :: $\langle [('r \Rightarrow (' \sigma, 'e, 'r) A_{nd}) \textit{list}, 'r] \Rightarrow (' \sigma \textit{list}, 'e, 'r) A_{nd} \rangle$ ($\langle \langle nd \otimes ; \checkmark - \rangle \rangle [0]$)
where $\langle \langle nd \otimes ; \checkmark [] \rangle \rangle r = \langle \tau = \lambda \sigma s a. \{\}, \omega = \lambda \sigma s. \{r\} \rangle$
 $| \quad \langle \langle nd \otimes ; \checkmark [A_0] \rangle \rangle r = nd \langle \langle A_0 r \rangle \rangle_{\sigma \mapsto \sigma s}$
 $| \quad \langle \langle nd \otimes ; \checkmark A_0 \# A_1 \# A_s \rangle \rangle r = \langle \langle A_0 r \ nd \otimes ; \checkmark Rlist \ \langle \langle nd \otimes ; \checkmark A_1 \# A_s \rangle \rangle \rangle$

lemma *iterated-combine_d-Seq_{ptick}-simps-bis*: $\langle As \neq [] \implies \langle \langle d \otimes ; \checkmark A_0 \# A_s \rangle \rangle r = \langle \langle A_0 r \ d \otimes ; \checkmark Rlist \ \langle \langle d \otimes ; \checkmark As \rangle \rangle \rangle$
and *iterated-combine_{nd}-Seq_{ptick}-simps-bis*: $\langle Bs \neq [] \implies \langle \langle nd \otimes ; \checkmark B_0 \# Bs \rangle \rangle r = \langle \langle B_0 r \ nd \otimes ; \checkmark Rlist \ \langle \langle nd \otimes ; \checkmark Bs \rangle \rangle \rangle$
by (*induct As, simp-all*) (*induct Bs, simp-all*)

10.1.2 First Results

lemma *combine_{ListslenL}-Seq_{ptick}-combine_{Rlist}-Seq_{ptick}-eq*:
 $\langle \varepsilon \langle \langle d \langle \langle A_0 \rangle \rangle_{\sigma \mapsto \sigma s} \ d \otimes ; \checkmark 1; ListslenL \ A_1 \rangle \rangle (s_0 \# \sigma s) = \varepsilon \langle \langle A_0 \ d \otimes ; \checkmark Rlist \ A_1 \rangle \rangle (s_0 \# \sigma s) \rangle$
 $\langle \tau \langle \langle d \langle \langle A_0 \rangle \rangle_{\sigma \mapsto \sigma s} \ d \otimes ; \checkmark 1; ListslenL \ A_1 \rangle \rangle (s_0 \# \sigma s) \ e = \tau \langle \langle A_0 \ d \otimes ; \checkmark Rlist \ A_1 \rangle \rangle (s_0 \# \sigma s) \ e \rangle$
 $\langle \varepsilon \langle \langle nd \langle \langle B_0 \rangle \rangle_{\sigma \mapsto \sigma s} \ nd \otimes ; \checkmark 1; ListslenL \ B_1 \rangle \rangle (s_0 \# \sigma s) = \varepsilon \langle \langle B_0 \ nd \otimes ; \checkmark Rlist \ B_1 \rangle \rangle (s_0 \# \sigma s) \rangle$

$\sigma s \rangle$
 $\langle \tau \langle \langle nd \langle B_0 \rangle \sigma \hookrightarrow \sigma s \ nd \otimes 1; List\ len L \ B_1 \rangle (s_0 \# \sigma s) \ e = \tau \langle B_0 \ nd \otimes; \checkmark Rlist \ B_1 \rangle (s_0 \# \sigma s) \ e \rangle$
by (*simp-all add: ε -simps ϱ -simps combine_{List\ len L}-Seq_{ptick} combine_{Rlist}-Seq_{ptick}-defs σ - σs -conv-defs*)
(safe, auto simp add: map-option-case split: option.splits)

lemma *combine_{Pairlist}-Seq_{ptick}-and-iterated-combine-Seq_{ptick}-eq*:
 $\langle \varepsilon \langle A_0 \ r \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] = \varepsilon \langle \langle d \otimes; \checkmark [A_0, A_1] \rangle r \rangle [s_0, s_1] \rangle$
 $\langle \tau \langle A_0 \ r \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] \ e = \tau \langle \langle d \otimes; \checkmark [A_0, A_1] \rangle r \rangle [s_0, s_1] \ e \rangle$
 $\langle \varepsilon \langle B_0 \ r \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] = \varepsilon \langle \langle nd \otimes; \checkmark [B_0, B_1] \rangle r \rangle [s_0, s_1] \rangle$
 $\langle \tau \langle B_0 \ r \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] \ e = \tau \langle \langle nd \otimes; \checkmark [B_0, B_1] \rangle r \rangle [s_0, s_1] \ e \rangle$
by (*simp-all add: ε -simps ϱ -simps combine_{Pairlist}-Seq_{ptick}-defs combine_{Rlist}-Seq_{ptick}-defs σ - σs -conv-defs*)
(safe, auto simp add: map-option-case split: option.splits)

lemma *combine_{Pairlist}-Seq_{ptick}-and-combine_{Rlist}-Seq_{ptick}-eq* :
 $\langle \varepsilon \langle A_0 \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] = \varepsilon \langle A_0 \ d \otimes; \checkmark Rlist \ \langle d \otimes; \checkmark [A_1] \rangle \rangle [s_0, s_1] \rangle$
 $\langle \tau \langle A_0 \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] \ e = \tau \langle A_0 \ d \otimes; \checkmark Rlist \ \langle d \otimes; \checkmark [A_1] \rangle \rangle [s_0, s_1] \ e \rangle$
 $\langle \varepsilon \langle B_0 \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] = \varepsilon \langle B_0 \ nd \otimes; \checkmark Rlist \ \langle nd \otimes; \checkmark [B_1] \rangle \rangle [s_0, s_1] \rangle$
 $\langle \tau \langle B_0 \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] \ e = \tau \langle B_0 \ nd \otimes; \checkmark Rlist \ \langle nd \otimes; \checkmark [B_1] \rangle \rangle [s_0, s_1] \ e \rangle$
by (*simp-all add: ε -simps ϱ -simps combine_{Pairlist}-Seq_{ptick}-defs combine_{Rlist}-Seq_{ptick}-defs σ - σs -conv-defs*)
(safe, auto simp add: map-option-case split: option.splits)

10.1.3 Reachability

lemma *same-length- τ -iterated-combine_d-Seq_{ptick}* :
 $\langle length \ \sigma s = length \ As \implies [\sigma t] = \tau \langle \langle d \otimes; \checkmark As \rangle r \rangle \ \sigma s \ a \implies length \ \sigma t = length \ As \rangle$

proof (*induct arbitrary: $\sigma t \ r$ rule: induct-2-lists012*)

case *Nil* **thus** *?case by simp*

next

case (*single* $\sigma_0 \ A_0$) **thus** *?case*

by *simp (metis length-1-trans_dE length-1-trans-from- σ -to- σs (1) length-Cons list.size(3))*

next

case (*Cons* $\sigma_0 \ \sigma_1 \ \sigma s \ A_0 \ A_1 \ As$)

from *Cons.premis Cons.hyps(1, 3)* **show** *?case*

by (*simp add: combine_{Rlist}-Seq_{ptick}-defs map-option-case ϱ -simps split: if-split-asm option.split-asm*) *metis*

qed

lemma *same-length- τ -iterated-combine_{nd}-Seq_{ptick}* :
 $\langle length \ \sigma s = length \ As \implies \sigma t \in \tau \langle \langle nd \otimes; \checkmark As \rangle r \rangle \ \sigma s \ a \implies length \ \sigma t = length \ As \rangle$

proof (*induct arbitrary: σt r rule: induct-2-lists012*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*single* σ_0 A_0) **thus** *?case*
by *simp* (*meson length-1-trans_{nd}-def length-1-trans-from- σ -to- σs (2)*)
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 As)
from *Cons.prem*s *Cons.hyps*(1, 3) **show** *?case*
by (*auto simp add: combine_{Rlist}-Seq_{ptick}-defs map-option-case ϱ -simps*
split: if-split-asm)
qed

lemma *same-length- \mathcal{R}_d -iterated-combine_d-Seq_{ptick}* :
 $\langle \sigma t \in \mathcal{R}_d (\langle \langle d \otimes ; \checkmark As \rangle r) \sigma s \implies \text{length } \sigma s = \text{length } As \implies \text{length } \sigma t = \text{length } As \rangle$
by (*induct rule: \mathcal{R}_d .induct, simp*) (*meson same-length- τ -iterated-combine_d-Seq_{ptick}*)

lemma *same-length- \mathcal{R}_{nd} -iterated-combine_{nd}-Seq_{ptick}* :
 $\langle \sigma t \in \mathcal{R}_{nd} (\langle \langle nd \otimes ; \checkmark As \rangle r) \sigma s \implies \text{length } \sigma s = \text{length } As \implies \text{length } \sigma t = \text{length } As \rangle$
by (*induct rule: \mathcal{R}_{nd} .induct, simp*) (*meson same-length- τ -iterated-combine_{nd}-Seq_{ptick}*)

10.1.4 Transmission of Properties

lemma *ϱ -disjoint- ε -transmission-to-iterated-combine-Seq_{ptick}* :
 $\langle As \neq [] \implies \varrho\text{-disjoint-}\varepsilon ((\text{last } As) r) \implies \varrho\text{-disjoint-}\varepsilon (\langle \langle d \otimes ; \checkmark As \rangle r) \rangle$
 $\langle Bs \neq [] \implies \varrho\text{-disjoint-}\varepsilon ((\text{last } Bs) r) \implies \varrho\text{-disjoint-}\varepsilon (\langle \langle nd \otimes ; \checkmark Bs \rangle r) \rangle$
by (*induct rule: induct-list012;*
auto simp add: ϱ -disjoint- ε -def combine_{Rlist}-Seq_{ptick}-defs ε -simps ϱ -simps
 σ - σs -conv-defs)+

10.1.5 Normalization

lemma *ω -iterated-combine-Seq_{ptick}-det-ndet-conv*:
 $\langle \omega (\langle \langle nd \otimes ; \checkmark \text{map } (\lambda A r. \langle A r \rangle_{d \mapsto nd}) As \rangle r) \sigma s = \omega \langle \langle d \otimes ; \checkmark As \rangle r \rangle_{d \mapsto nd} \sigma s \rangle$
by (*induct As arbitrary: σs r rule: induct-list012[case-names Nil singl Cons]*)
(simp-all add: from-det-to-ndet- σ - σs -conv-commute combine_{Rlist}-Seq_{ptick}-defs
 ω -from-det-to-ndet split: option.split)

lemma *ϱ -iterated-combine-Seq_{ptick}-det-ndet-conv* :
 $\langle \varrho (\langle \langle nd \otimes ; \checkmark \text{map } (\lambda A r. \langle A r \rangle_{d \mapsto nd}) As \rangle r) = \varrho \langle \langle d \otimes ; \checkmark As \rangle r \rangle_{d \mapsto nd} \rangle$
by (*simp add: ϱ -simps ω -iterated-combine-Seq_{ptick}-det-ndet-conv*)

lemma *τ -iterated-combine-Seq_{ptick}-behaviour*:
 $\langle \text{length } \sigma s = \text{length } As \implies$
 $\tau \langle \langle d \otimes ; \checkmark As \rangle r \rangle_{d \mapsto nd} \sigma s e = \tau (\langle \langle nd \otimes ; \checkmark \text{map } (\lambda A r. \langle A r \rangle_{d \mapsto nd}) As \rangle r) \sigma s$
 $e \rangle$

proof (*induct* σs *As arbitrary*: *r rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** *simp*
next
case (*single* σ_0 A_0)
show *?case* **by** (*simp add*: *from-det-to-ndet- σ - σ s-conv-commute(1)*)
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 *As*)
show *?case*
by (*simp add*: *τ -combine $_{Rlist-Seq_{ptick}}$ -behaviour split: option.split,*
simp add: combine $_{Rlist-Seq_{ptick}}$ -defs ϱ -iterated-combine- Seq_{ptick} -det-ndet-conv
 ω -from-det-to-ndet split: option.split)
(*metis Cons.hyps(3) ϱ -iterated-combine- Seq_{ptick} -det-ndet-conv det-ndet-conv- $\varrho(1)$*
list.simps(9))
qed

lemma *P $_{SKIPs}$ -iterated-combine- Seq_{ptick} -behaviour*:
assumes *same-length*: $\langle \text{length } \sigma s = \text{length } As \rangle$
shows $\langle P_{SKIPs} \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_d \sigma s = P_{SKIPs} \langle \langle \langle nd \otimes ; \checkmark \rangle \text{map } (\lambda A r. \langle A r \rangle_{d \hookrightarrow nd}) As \rangle r \rangle_{nd} \sigma s \rangle$
proof (*fold* *P $_{SKIPs}$ -nd-from-det-to-ndet-is-P $_{SKIPs}$ -d*, *rule* *P $_{SKIPs}$ -nd-eqI-strong-id*)
show $\langle \sigma t \in \mathcal{R}_{nd} \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_{d \hookrightarrow nd} \sigma s \implies$
 $\tau (\langle \langle nd \otimes ; \checkmark \rangle \text{map } (\lambda A r. \langle A r \rangle_{d \hookrightarrow nd}) As \rangle r) \sigma t a = \tau \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_{d \hookrightarrow nd}$
 $\sigma t a \rangle$ **for** $\sigma t a$
by (*simp add*: *\mathcal{R}_{nd} -from-det-to-ndet τ -iterated-combine- Seq_{ptick} -behaviour*
same-length same-length- \mathcal{R}_d -iterated-combine $_d$ - Seq_{ptick})
next
show $\langle \omega (\langle \langle nd \otimes ; \checkmark \rangle \text{map } (\lambda A r. \langle A r \rangle_{d \hookrightarrow nd}) As \rangle r) \sigma t = \omega \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_{d \hookrightarrow nd}$
 $\sigma t \rangle$ **for** σt
by (*simp add*: *ω -iterated-combine- Seq_{ptick} -det-ndet-conv*)
qed

lemma *P-iterated-combine- Seq_{ptick} -behaviour*:
assumes *same-length*: $\langle \text{length } \sigma s = \text{length } As \rangle$
shows $\langle P \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_d \sigma s = P \langle \langle \langle nd \otimes ; \checkmark \rangle \text{map } (\lambda A r. \langle A r \rangle_{d \hookrightarrow nd}) As \rangle r \rangle_{nd} \sigma s \rangle$
proof (*fold* *P-nd-from-det-to-ndet-is-P-d*, *rule* *P-nd-eqI-strong-id*)
show $\langle \sigma t \in \mathcal{R}_{nd} \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_{d \hookrightarrow nd} \sigma s \implies$
 $\tau (\langle \langle nd \otimes ; \checkmark \rangle \text{map } (\lambda A r. \langle A r \rangle_{d \hookrightarrow nd}) As \rangle r) \sigma t a = \tau \langle \langle \langle d \otimes ; \checkmark \rangle As \rangle r \rangle_{d \hookrightarrow nd}$
 $\sigma t a \rangle$ **for** $\sigma t a$
by (*simp add*: *\mathcal{R}_{nd} -from-det-to-ndet τ -iterated-combine- Seq_{ptick} -behaviour*
same-length same-length- \mathcal{R}_d -iterated-combine $_d$ - Seq_{ptick})
qed

10.2 Compactification Theorems

10.2.1 Binary

Pair

lemma $P_{SKIPS}\text{-nd-combine}_{Pair}\text{-Seq}_{ptick}$:

fixes $A_0 A_1$

assumes $at\text{-most-1-elem-term}$: $\langle at\text{-most-1-elem-term } A_0 \rangle$

— This assumption is necessary in the new setup, otherwise the result is not always a Procomaton (for example if $\omega A_0 \sigma_0 = UNIV$, we have $P \sigma_0 ;\checkmark Q \sigma_1 = GlobalNdet UNIV (Q \sigma_1)$).

defines $A\text{-def}$: $\langle A \equiv \langle A_0 \text{ nd}\otimes;\checkmark_{Pair} A_1 \rangle \rangle$

defines $P\text{-def}$: $\langle P \equiv P_{SKIPS}\langle A_0 \rangle_{nd} \rangle$

and $Q\text{-def}$: $\langle Q \equiv \lambda \sigma_1 r. P_{SKIPS}\langle A_1 r \rangle_{nd} \sigma_1 \rangle$

and $S\text{-def}$: $\langle S \equiv P_{SKIPS}\langle A \rangle_{nd} \rangle$

shows $\langle P \sigma_0 ;\checkmark Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$

proof –

let $?f = \langle P_{SKIPS}\text{-nd-step } (\varepsilon A) (\tau A) (\omega A) (\lambda \sigma'. \text{case } \sigma' \text{ of } (\sigma_0, \sigma_1) \Rightarrow P \sigma_0 ;\checkmark Q \sigma_1) \rangle$

note $cartprod\text{-rwrt} = GlobalNdet\text{-cartprod}[of - - \langle \lambda x y. - (x, y) \rangle, \text{simplified}]$

note $Ndet\text{-and-Seq} = Seq_{ptick}\text{-distrib-GlobalNdet-left Seq}_{ptick}\text{-distrib-GlobalNdet-right}$

have $P\text{-rec}$: $\langle P \sigma_0 = P_{SKIPS}\text{-nd-step } (\varepsilon A_0) (\tau A_0) (\omega A_0) P \sigma_0 \rangle$ **for** σ_0

by ($fact\text{ restriction-fix-eq}[OF P_{SKIPS}\text{-nd-step-constructive-bis}[of A_0],$
 $folded P_{SKIPS}\text{-nd-def } P\text{-def}, THEN fun\text{-cong}]$)

have $Q\text{-rec}$: $\langle Q \sigma_1 = (\lambda r. P_{SKIPS}\text{-nd-step } (\varepsilon (A_1 r)) (\tau (A_1 r)) (\omega (A_1 r))) (\lambda \sigma_1. Q \sigma_1 r) \sigma_1 \rangle$ **for** σ_1

by ($rule\ ext, simp\ add: Q\text{-def } P_{SKIPS}\text{-nd-rec}$)

show $\langle P \sigma_0 ;\checkmark Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$

proof ($rule\ fun\text{-cong}[of \langle \lambda(\sigma_0, \sigma_1). P \sigma_0 ;\checkmark Q \sigma_1 \rangle - \langle (\sigma_0, \sigma_1) \rangle, \text{simplified}]$)

show $\langle \lambda(\sigma_0, \sigma_1). P \sigma_0 ;\checkmark Q \sigma_1 = S \rangle$

proof ($rule\ restriction\text{-fix-unique}[OF P_{SKIPS}\text{-nd-step-constructive-bis}[of A],$
 $symmetric, folded P_{SKIPS}\text{-nd-def } S\text{-def}]$)

show $\langle ?f = (\lambda(\sigma_0, \sigma_1). P \sigma_0 ;\checkmark Q \sigma_1) \rangle$

proof ($rule\ ext, clarify$)

show $\langle ?f (\sigma_0, \sigma_1) = P \sigma_0 ;\checkmark Q \sigma_1 \rangle$ **for** $\sigma_0 \sigma_1$

proof ($cases \langle \sigma_0 \in \varrho A_0 \rangle$)

show $\langle \sigma_0 \notin \varrho A_0 \implies ?f (\sigma_0, \sigma_1) = P \sigma_0 ;\checkmark Q \sigma_1 \rangle$

by ($subst (2) P\text{-rec}$)

($auto\ simp\ add: combine_{Pair}\text{-Seq}_{ptick}\text{-defs } \varepsilon\text{-simps } A\text{-def } \varrho\text{-simps}$)

($Mprefix\text{-Seq}_{ptick}\ cartprod\text{-rwrt } Ndet\text{-and-Seq intro!: mono-Mprefix-eq}$)

next

assume $\langle \sigma_0 \in \varrho A_0 \rangle$

hence $\langle \omega A_0 \sigma_0 \neq \{\} \rangle$ **by** ($simp\ add: \varrho\text{-simps}$)

then obtain r **where** $\langle \omega A_0 \sigma_0 = \{r\} \rangle$

by ($meson\ at\text{-most-1-elem-term } at\text{-most-1-elem-termE}$)

from $\langle \sigma_0 \in \varrho A_0 \rangle$ **have** $P\text{-rec}'$: $\langle P \sigma_0 = SKIPS (\omega A_0 \sigma_0) \rangle$ **by** ($simp\ add: P\text{-rec } \varrho\text{-simps}$)

have $\varepsilon\text{-A}$: $\langle \varepsilon A (\sigma_0, \sigma_1) = (\text{if } \sigma_1 \in \varrho (A_1 r) \text{ then } \{\} \text{ else } \varepsilon (A_1 r) \sigma_1) \rangle$

by (*simp add: A-def ε -combine_{Pair-Seqptick} combine-Seq- ε -defs* $\langle \sigma_0 \in \varrho$
 $A_0 \rangle \langle \omega A_0 \sigma_0 = \{r\} \rangle$)
 from $\langle \sigma_0 \in \varrho A_0 \rangle$ have $\omega\text{-}A : \langle \omega A (\sigma_0, \sigma_1) = \omega (A_1 r) \sigma_1 \rangle$
 by (*simp add: A-def combine_{Pair-Seqptick}-defs* $\langle \omega A_0 \sigma_0 = \{r\} \rangle$)
 show $\langle ?f (\sigma_0, \sigma_1) = P \sigma_0 ; \checkmark Q \sigma_1 \rangle$
 proof (*cases* $\langle \sigma_1 \in \varrho (A_1 r) \rangle$)
 show $\langle \sigma_1 \in \varrho (A_1 r) \implies ?f (\sigma_0, \sigma_1) = P \sigma_0 ; \checkmark Q \sigma_1 \rangle$
 by (*subst* (2) *Q-rec*)
 (*simp add: P-rec' ε -A ω -A SKIPS-def Ndet-and-Seq* $\langle \omega A_0 \sigma_0 = \{r\} \rangle$)
 ϱ -simps)
 next
 show $\langle \sigma_1 \notin \varrho (A_1 r) \implies ?f (\sigma_0, \sigma_1) = P \sigma_0 ; \checkmark Q \sigma_1 \rangle$
 by (*subst* (2) *Q-rec, unfold ε -A ω -A SKIPS-def*)
 (*auto simp add: A-def P-rec' Ndet-and-Seq* $\langle \omega A_0 \sigma_0 = \{r\} \rangle$)
 ϱ -simps *cartprod-rwrt combine_{Pair-Seqptick}-defs intro!: mono-Mprefix-eq*)
 qed
 qed
 qed
 qed
 qed
 qed

corollary *P_{SKIPS-d-combine_{Pair-Seqptick}}* :
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_d \sigma_1) = P_{SKIPS} \langle \langle A_0 d \otimes ; \checkmark_{Pair} A_1 \rangle \rangle_d$
 $(\sigma_0, \sigma_1) \rangle$
proof –
 have $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_d \sigma_1) =$
 $P_{SKIPS} \langle \langle A_0 \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle \langle A_1 r \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_1) \rangle$
 by (*simp add: P_{SKIPS-nd-from-det-to-ndet-is-P_{SKIPS-d}}*)
 also have $\langle \dots = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} nd \otimes ; \checkmark_{Pair} (\lambda r. \langle A_1 r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (\sigma_0,$
 $\sigma_1) \rangle$
 by (*metis P_{SKIPS-nd-combine_{Pair-Seqptick} at-most-1-elem-from-det-to-ndet}*
at-most-1-elem-imp-at-most-1-elem-term)
 also have $\langle \dots = P_{SKIPS} \langle \langle A_0 d \otimes ; \checkmark_{Pair} A_1 \rangle \rangle_d (\sigma_0, \sigma_1) \rangle$
 by (*simp add: P_{SKIPS-combine_{Pair-Seqptick}-behaviour}*)
 finally show *?thesis* .
 qed

Pairlist

lemma *P_{SKIPS-nd-combine_{Pairlist-Seqptick}}* :
 $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_{nd} \sigma_1) = P_{SKIPS} \langle \langle A_0 nd \otimes ; \checkmark_{Pairlist} A_1 \rangle \rangle_{nd} [\sigma_0, \sigma_1] \rangle$
 if *at-most-1-elem-term* A_0
proof –
 from *P_{SKIPS-nd-combine_{Pair-Seqptick}[OF at-most-1-elem-term}* A_0 *]*
 have $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_{nd} \sigma_1) =$
 $P_{SKIPS} \langle \langle A_0 nd \otimes ; \checkmark_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1) \rangle$.
 also have $\langle \dots = P_{SKIPS} \langle \langle A_0 nd \otimes ; \checkmark_{Pairlist} A_1 \rangle \rangle_{nd} [\sigma_0, \sigma_1] \rangle$

proof (rule $P_{SKIPS}\text{-nd-eqI-strong}$ [of $\langle \lambda(\sigma_0, \sigma_1). [\sigma_0, \sigma_1] \rangle - \langle (\sigma_0, \sigma_1) \rangle$, *simplified*])
show $\langle inj\text{-on } (\lambda(\sigma_0, \sigma_1). [\sigma_0, \sigma_1]) (\mathcal{R}_{nd} \langle A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rangle (\sigma_0, \sigma_1)) \rangle$
by (auto intro: *inj-onI*)
next
show $\langle \tau \langle A_0 \text{ nd}\otimes; \checkmark_{Pairlist} A_1 \rangle (\text{case } \sigma' \text{ of } (\sigma_0, \sigma_1) \Rightarrow [\sigma_0, \sigma_1]) a =$
 $(\lambda\sigma'. \text{case } \sigma' \text{ of } (\sigma_0, \sigma_1) \Rightarrow [\sigma_0, \sigma_1]) ' \tau \langle A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rangle \sigma' a \rangle$ **for** $\sigma' a$
by (cases σ') (auto simp add: *combinePair-Seqptick-defs combinePairlist-Seqptick-defs*)
next
show $\langle \omega \langle A_0 \text{ nd}\otimes; \checkmark_{Pairlist} A_1 \rangle (\text{case } \sigma' \text{ of } (\sigma_0, \sigma_1) \Rightarrow [\sigma_0, \sigma_1]) = \omega \langle A_0$
 $\text{nd}\otimes; \checkmark_{Pair} A_1 \rangle \sigma' \rangle$ **for** σ'
by (cases σ') (auto simp add: *combinePair-Seqptick-defs combinePairlist-Seqptick-defs*)
qed
finally show *?thesis* .
qed

corollary $P_{SKIPS}\text{-d-combinePairlist-Seqptick}$:
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_d \sigma_1) = P_{SKIPS} \langle \langle A_0 \text{ d}\otimes; \checkmark_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1] \rangle$
proof –
have $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_d \sigma_1) =$
 $P_{SKIPS} \langle \langle A_0 \text{ d}\hookrightarrow \text{nd} \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle \langle A_1 r \rangle_{d\hookrightarrow \text{nd}} \rangle_{nd} \sigma_1) \rangle$
by (*simp add: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d*)
also have $\langle \dots = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d\hookrightarrow \text{nd}} \text{ nd}\otimes; \checkmark_{Pairlist} (\lambda r. \langle A_1 r \rangle_{d\hookrightarrow \text{nd}}) \rangle \rangle_{nd}$
 $[\sigma_0, \sigma_1] \rangle$
by (*metis P_{SKIPS}-nd-combinePairlist-Seqptick at-most-1-elem-from-det-to-ndet*
at-most-1-elem-imp-at-most-1-elem-term)
also have $\langle \dots = P_{SKIPS} \langle \langle A_0 \text{ d}\otimes; \checkmark_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1] \rangle$
by (*simp add: P_{SKIPS}-combinePairlist-Seqptick-behaviour*)
finally show *?thesis* .
qed

Rlist

lemma $P_{SKIPS}\text{-nd-combineRlist-Seqptick}$:
 $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_{nd} \sigma s) = P_{SKIPS} \langle \langle A_0 \text{ nd}\otimes; \checkmark_{Rlist} A_1 \rangle \rangle_{nd} (\sigma_0 \# \sigma s) \rangle$
if $\langle at\text{-most-1-elem-term } A_0 \rangle$
proof –
from $P_{SKIPS}\text{-nd-combinePair-Seqptick}$ [*OF* $\langle at\text{-most-1-elem-term } A_0 \rangle$]
have $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle A_1 r \rangle_{nd} \sigma s) =$
 $P_{SKIPS} \langle \langle A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma s) \rangle$.
also have $\langle \dots = P_{SKIPS} \langle \langle A_0 \text{ nd}\otimes; \checkmark_{Rlist} A_1 \rangle \rangle_{nd} (\sigma_0 \# \sigma s) \rangle$
proof (rule $P_{SKIPS}\text{-nd-eqI-strong}$ [of $\langle \lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s \rangle - \langle (\sigma_0, \sigma s) \rangle$, *simplified*])
show $\langle inj\text{-on } (\lambda(\sigma_0, \sigma s). \sigma_0 \# \sigma s) (\mathcal{R}_{nd} \langle A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rangle (\sigma_0, \sigma s)) \rangle$
by (auto intro: *inj-onI*)
next
show $\langle \tau \langle A_0 \text{ nd}\otimes; \checkmark_{Rlist} A_1 \rangle (\text{case } \sigma' \text{ of } (\sigma_0, \sigma s) \Rightarrow \sigma_0 \# \sigma s) a =$

$(\lambda\sigma'. \text{case } \sigma' \text{ of } (\sigma_0, \sigma_s) \Rightarrow \sigma_0 \# \sigma_s) \text{ ' } \tau \llbracket A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rrbracket \sigma' a \text{ ' for } \sigma' a$
by (cases σ') (auto simp add: combine_{Pair}-Seq_{ptick}-defs combine_{Rlist}-Seq_{ptick}-defs)
next
show $\langle \omega \llbracket A_0 \text{ nd}\otimes; \checkmark_{Rlist} A_1 \rrbracket (\text{case } \sigma' \text{ of } (\sigma_0, \sigma_s) \Rightarrow \sigma_0 \# \sigma_s) = \omega \llbracket A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rrbracket \sigma' \rangle$ **for** σ'
by (cases σ') (auto simp add: combine_{Pair}-Seq_{ptick}-defs combine_{Rlist}-Seq_{ptick}-defs)
qed
finally show ?thesis .
qed

corollary P_{SKIPS} -d-combine_{Rlist}-Seq_{ptick} :

$\langle P_{SKIPS} \llbracket A_0 \rrbracket_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \llbracket A_1 r \rrbracket_d \sigma_s) = P_{SKIPS} \llbracket \llbracket A_0 d\otimes; \checkmark_{Rlist} A_1 \rrbracket_d (\sigma_0 \# \sigma_s) \rangle$

proof –

have $\langle P_{SKIPS} \llbracket A_0 \rrbracket_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \llbracket A_1 r \rrbracket_d \sigma_s) = P_{SKIPS} \llbracket \llbracket A_0 \rrbracket_{d \hookrightarrow nd} \rrbracket_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \llbracket \llbracket A_1 r \rrbracket_{d \hookrightarrow nd} \rrbracket_{nd} \sigma_s) \rangle$
by (simp add: P_{SKIPS}-nd-from-det-to-ndet-is-P_{SKIPS}-d)
also have $\langle \dots = P_{SKIPS} \llbracket \llbracket \llbracket A_0 \rrbracket_{d \hookrightarrow nd} \text{ nd}\otimes; \checkmark_{Rlist} (\lambda r. \llbracket A_1 r \rrbracket_{d \hookrightarrow nd}) \rrbracket \rrbracket_{nd} (\sigma_0 \# \sigma_s) \rangle$
by (metis P_{SKIPS}-nd-combine_{Rlist}-Seq_{ptick} at-most-1-elem-from-det-to-ndet at-most-1-elem-imp-at-most-1-elem-term)
also have $\langle \dots = P_{SKIPS} \llbracket \llbracket A_0 d\otimes; \checkmark_{Rlist} A_1 \rrbracket \rrbracket_d (\sigma_0 \# \sigma_s) \rangle$
by (simp add: P_{SKIPS}-combine_{Rlist}-Seq_{ptick}-behaviour)
finally show ?thesis .
qed

10.2.2 ListslenL

lemma P_{SKIPS} -nd-combine_{ListslenL}-Seq_{ptick} :

$\langle P_{SKIPS} \llbracket A_0 \rrbracket_{nd} \sigma_{s_0} ; \checkmark (\lambda r. P_{SKIPS} \llbracket A_1 r \rrbracket_{nd} \sigma_{s_1}) = P_{SKIPS} \llbracket \llbracket A_0 \text{ nd}\otimes \text{ len}_0; \text{ListslenL} A_1 \rrbracket \rrbracket_{nd} (\sigma_{s_0} @ \sigma_{s_1}) \rangle$

if same-length-reach : $\langle \bigwedge \sigma_{s_0}'. \sigma_{s_0}' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \implies \text{length } \sigma_{s_0}' = \text{len}_0 \rangle$
and (at-most-1-elem-term A_0)

proof –

from P_{SKIPS}-nd-combine_{Pair}-Seq_{ptick}[OF (at-most-1-elem-term A_0)]
have $\langle P_{SKIPS} \llbracket A_0 \rrbracket_{nd} \sigma_{s_0} ; \checkmark (\lambda r. P_{SKIPS} \llbracket A_1 r \rrbracket_{nd} \sigma_{s_1}) = P_{SKIPS} \llbracket \llbracket A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rrbracket \rrbracket_{nd} (\sigma_{s_0}, \sigma_{s_1}) \rangle$.
also have $\langle \dots = P_{SKIPS} \llbracket \llbracket A_0 \text{ nd}\otimes \text{ len}_0; \text{ListslenL} A_1 \rrbracket \rrbracket_{nd} (\sigma_{s_0} @ \sigma_{s_1}) \rangle$
proof (rule P_{SKIPS}-nd-eqI-strong[of $\langle \lambda(\sigma_{s_0}, \sigma_{s_1}). \sigma_{s_0} @ \sigma_{s_1} \rangle$ - $\langle (\sigma_{s_0}, \sigma_{s_1}) \rangle$, simplified])
show $\langle \text{inj-on } (\lambda(\sigma_{s_0}, \sigma_{s_1}). \sigma_{s_0} @ \sigma_{s_1}) (\mathcal{R}_{nd} \llbracket A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rrbracket (\sigma_{s_0}, \sigma_{s_1})) \rangle$
proof (rule inj-onI, clarify)
fix $\sigma_{s_0}' \sigma_{s_1}' \sigma_{s_0}'' \sigma_{s_1}''$
assume $\text{assms} : \langle (\sigma_{s_0}', \sigma_{s_1}') \in \mathcal{R}_{nd} \llbracket A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rrbracket (\sigma_{s_0}, \sigma_{s_1}) \rangle$
 $\langle (\sigma_{s_0}'', \sigma_{s_1}'') \in \mathcal{R}_{nd} \llbracket A_0 \text{ nd}\otimes; \checkmark_{Pair} A_1 \rrbracket (\sigma_{s_0}, \sigma_{s_1}) \rangle$
 $\langle \sigma_{s_0}' @ \sigma_{s_1}' = \sigma_{s_0}'' @ \sigma_{s_1}'' \rangle$
from \mathcal{R}_{nd} -combine_{ndPair}-Seq_{ptick}-subset $\text{assms}(1, 2)$
have $\langle \sigma_{s_0}' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \rangle \langle \sigma_{s_0}'' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \rangle$ **by** fast+
with same-length-reach **have** $\langle \text{length } \sigma_{s_0}' = \text{length } \sigma_{s_0}'' \rangle$ **by** blast

with $assms(\beta)$ **show** $\langle \sigma_{s_0}' = \sigma_{s_0}'' \wedge \sigma_{s_1}' = \sigma_{s_1}'' \rangle$ **by** *simp*
qed
next
fix $\sigma s' a$
assume $\langle \sigma s' \in \mathcal{R}_{nd} \langle \langle A_0 \text{ nd} \otimes; \checkmark_{Pair} A_1 \rangle \rangle (\sigma_{s_0}, \sigma_{s_1}) \rangle$
with $\mathcal{R}_{nd-combine_{ndPair-Seq_{ptick-subset}}$
obtain $\langle \sigma_{s_0}' \sigma_{s_1}' \text{ where } \langle \sigma s' = (\sigma_{s_0}', \sigma_{s_1}') \rangle \langle \sigma_{s_0}' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \rangle$ **by** *fast*
from $\langle \sigma_{s_0}' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \rangle$ *same-length-reach* **have** $\langle \text{length } \sigma_{s_0}' = \text{len}_0 \rangle$ **by**
blast
show $\langle \tau \langle \langle A_0 \text{ nd} \otimes \text{len}_0; ListslenL A_1 \rangle \rangle (\text{case } \sigma s' \text{ of } (\sigma_{s_0}, \sigma_{s_1}) \Rightarrow \sigma_{s_0} @ \sigma_{s_1}) a =$
 $(\lambda \sigma s'. \text{case } \sigma s' \text{ of } (\sigma_{s_0}, \sigma_{s_1}) \Rightarrow \sigma_{s_0} @ \sigma_{s_1}) ' \tau \langle \langle A_0 \text{ nd} \otimes; \checkmark_{Pair} A_1 \rangle \rangle \sigma s' a \rangle$
by (*auto simp add:* $\langle \sigma s' = (\sigma_{s_0}', \sigma_{s_1}') \rangle \langle \text{length } \sigma_{s_0}' = \text{len}_0 \rangle$
combine_{Pair-Seq_{ptick-defs} combine_{ListslenL-Seq_{ptick}})
next
fix $\sigma s' a$
assume $\langle \sigma s' \in \mathcal{R}_{nd} \langle \langle A_0 \text{ nd} \otimes; \checkmark_{Pair} A_1 \rangle \rangle (\sigma_{s_0}, \sigma_{s_1}) \rangle$
with $\mathcal{R}_{nd-combine_{ndPair-Seq_{ptick-subset}}$
obtain $\langle \sigma_{s_0}' \sigma_{s_1}' \text{ where } \langle \sigma s' = (\sigma_{s_0}', \sigma_{s_1}') \rangle \langle \sigma_{s_0}' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \rangle$ **by** *fast*
from $\langle \sigma_{s_0}' \in \mathcal{R}_{nd} A_0 \sigma_{s_0} \rangle$ *same-length-reach* **have** $\langle \text{length } \sigma_{s_0}' = \text{len}_0 \rangle$ **by**
blast
show $\langle \omega \langle \langle A_0 \text{ nd} \otimes \text{len}_0; ListslenL A_1 \rangle \rangle (\text{case } \sigma s' \text{ of } (\sigma_{s_0}, \sigma_{s_1}) \Rightarrow \sigma_{s_0} @ \sigma_{s_1}) =$
 $\omega \langle \langle A_0 \text{ nd} \otimes; \checkmark_{Pair} A_1 \rangle \rangle \sigma s' \rangle$
by (*auto simp add:* $\langle \sigma s' = (\sigma_{s_0}', \sigma_{s_1}') \rangle \langle \text{length } \sigma_{s_0}' = \text{len}_0 \rangle$
combine_{Pair-Seq_{ptick-defs} combine_{ListslenL-Seq_{ptick}})
qed
finally show *?thesis* .
qed

corollary $P_{SKIPS-d-combine_{ListslenL-Seq_{ptick}}$:
 $\langle P_{SKIPS} \langle \langle A_0 \rangle \rangle_d \sigma_{s_0}; \checkmark (\lambda r. P_{SKIPS} \langle \langle A_1 r \rangle \rangle_d \sigma_{s_1}) = P_{SKIPS} \langle \langle \langle A_0 d \otimes \text{len}_0; ListslenL A_1 \rangle \rangle \rangle_d (\sigma_{s_0} @ \sigma_{s_1}) \rangle$
if *same-length-reach* : $\langle \bigwedge \sigma_{s_0}'. \sigma_{s_0}' \in \mathcal{R}_d A_0 \sigma_{s_0} \implies \text{length } \sigma_{s_0}' = \text{len}_0 \rangle$
proof –
have $\langle P_{SKIPS} \langle \langle A_0 \rangle \rangle_d \sigma_{s_0}; \checkmark (\lambda r. P_{SKIPS} \langle \langle A_1 r \rangle \rangle_d \sigma_{s_1}) =$
 $P_{SKIPS} \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_{s_0}; \checkmark (\lambda r. P_{SKIPS} \langle \langle \langle A_1 r \rangle \rangle_{d \hookrightarrow nd} \rangle_{nd} \sigma_{s_1}) \rangle$
by (*simp add:* $P_{SKIPS-nd-from-det-to-ndet-is-P_{SKIPS-d}$)
also have $\langle \dots = P_{SKIPS} \langle \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \rangle_{nd} \otimes \text{len}_0; ListslenL (\lambda r. \langle \langle A_1 r \rangle \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd}$
 $(\sigma_{s_0} @ \sigma_{s_1}) \rangle$
by (*rule* $P_{SKIPS-nd-combine_{ListslenL-Seq_{ptick}}$ *[OF same-length-reach]*)
(simp-all add: $\mathcal{R}_{nd-from-det-to-ndet-at-most-1-elem-imp-at-most-1-elem-term}$)
also have $\langle \dots = P_{SKIPS} \langle \langle \langle A_0 d \otimes \text{len}_0; ListslenL A_1 \rangle \rangle \rangle_d (\sigma_{s_0} @ \sigma_{s_1}) \rangle$
by (*simp add:* $P_{SKIPS-combine_{ListslenL-Seq_{ptick-behaviour}}$ *same-length-reach*)
finally show *?thesis* .
qed

10.2.3 Multiple

theorem $P_{SKIPS-nd-compactification-Seq_{ptick}}$:

$\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A r. A \in \text{set } (\text{butlast } As) \implies \text{at-most-1-elem-term } (A r) \rrbracket \implies$
 $\langle \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } \sigma s As. (\lambda r. P_{SKIPS} \langle A r \rangle_{nd} \sigma) = (\lambda r. P_{SKIPS} \langle \langle nd \otimes ; \checkmark$
 $As \rangle r \rangle_{nd} \sigma s) \rangle$
proof (*induct* $\sigma s As$ *rule: induct-2-lists012*)
case Nil **show** *?case* **by** (*simp add: P_{SKIPS}-nd-rec*)
next
case (*single* $\sigma_0 A_0$) **show** *?case* **by** (*simp add: P_{SKIPS}-nd-from- σ -to- σs -is- P_{SKIPS} -nd*)
next
case (*Cons* $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)
have $\langle \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } (\sigma_0 \# \sigma_1 \# \sigma s) (A_0 \# A_1 \# As). (\lambda r. P_{SKIPS} \langle A$
 $r \rangle_{nd} \sigma) =$
 $(\lambda r. P_{SKIPS} \langle A_0 r \rangle_{nd} \sigma_0 ; \checkmark \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } (\sigma_1 \# \sigma s) (A_1 \# As).$
 $(\lambda r. P_{SKIPS} \langle A r \rangle_{nd} \sigma)) \rangle$ **by** *simp*
also have $\langle \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } (\sigma_1 \# \sigma s) (A_1 \# As). (\lambda r. P_{SKIPS} \langle A r \rangle_{nd}$
 $\sigma) =$
 $(\lambda r. P_{SKIPS} \langle \langle nd \otimes ; \checkmark A_1 \# As \rangle r \rangle_{nd} (\sigma_1 \# \sigma s)) \rangle$
by (*rule Cons.hyps(3)*) (*simp add: Cons.prem*s)
also have $\langle (\lambda r. P_{SKIPS} \langle A_0 r \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle \langle nd \otimes ; \checkmark A_1 \# As \rangle r \rangle_{nd}$
 $(\sigma_1 \# \sigma s))) =$
 $(\lambda r. P_{SKIPS} \langle \langle A_0 r \text{ nd} \otimes ; \checkmark Rlist \langle nd \otimes ; \checkmark A_1 \# As \rangle \rangle \rangle_{nd} (\sigma_0 \# \sigma_1 \#$
 $\sigma s)) \rangle$
by (*intro ext P_{SKIPS}-nd-combine_{Rlist}-Seq_{ptick}*) (*simp add: Cons.prem*s)
also have $\langle \dots = (\lambda r. P_{SKIPS} \langle \langle nd \otimes ; \checkmark A_0 \# A_1 \# As \rangle r \rangle_{nd} (\sigma_0 \# \sigma_1 \# \sigma s)) \rangle$
by *simp*
finally show *?case* .
qed

corollary *P_{SKIPS}-d-compactification-Seq_{ptick}*:

$\langle \text{length } \sigma s = \text{length } As \implies$
 $\langle \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } \sigma s As. (\lambda r. P_{SKIPS} \langle A r \rangle_d \sigma) = (\lambda r. P_{SKIPS} \langle \langle d \otimes ; \checkmark$
 $As \rangle r \rangle_d \sigma s) \rangle$

proof (*induct* $\sigma s As$ *rule: induct-2-lists012*)

case Nil **show** *?case* **by** (*simp add: P_{SKIPS}-d-rec*)

next

case (*single* $\sigma_0 A_0$) **show** *?case* **by** (*simp add: P_{SKIPS}-d-from- σ -to- σs -is- P_{SKIPS} -d*)

next

case (*Cons* $\sigma_0 \sigma_1 \sigma s A_0 A_1 As$)

have $\langle \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } (\sigma_0 \# \sigma_1 \# \sigma s) (A_0 \# A_1 \# As). (\lambda r. P_{SKIPS} \langle A$
 $r \rangle_d \sigma) =$

$(\lambda r. P_{SKIPS} \langle A_0 r \rangle_d \sigma_0 ; \checkmark \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } (\sigma_1 \# \sigma s) (A_1 \# As).$
 $(\lambda r. P_{SKIPS} \langle A r \rangle_d \sigma)) \rangle$ **by** *simp*

also have $\langle \text{SEQ}_{\checkmark}(\sigma, A) \in @ \text{zip } (\sigma_1 \# \sigma s) (A_1 \# As). (\lambda r. P_{SKIPS} \langle A r \rangle_d \sigma)$
 $=$

$(\lambda r. P_{SKIPS} \langle \langle d \otimes ; \checkmark A_1 \# As \rangle r \rangle_d (\sigma_1 \# \sigma s)) \rangle$

by (*fact Cons.hyps(3)*)

also have $\langle (\lambda r. P_{SKIPS} \langle A_0 r \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS} \langle \langle d \otimes ; \checkmark A_1 \# As \rangle r \rangle_d$
 $(\sigma_1 \# \sigma s))) =$

$(\lambda r. P_{SKIPS} \langle \langle A_0 r \text{ d} \otimes ; \checkmark Rlist \langle d \otimes ; \checkmark A_1 \# As \rangle \rangle \rangle_d (\sigma_0 \# \sigma_1 \# \sigma s)) \rangle$

by (*intro ext P_{SKIPS}-d-combine_{list-Seq}_{tick}*)
 also have $\langle \dots = (\lambda r. P_{SKIPS} \langle \langle d \otimes ; \checkmark A_0 \# A_1 \# As \rangle \rangle r \rangle_d (\sigma_0 \# \sigma_1 \# \sigma s) \rangle$
 by *simp*
 finally show *?case* .
 qed

Chapter 11

Application : May Philosophers dine ?

11.1 Preliminaries

11.1.1 Preliminary lemmas for proof automation

lemma *Suc-mod*: $\langle n > 1 \implies i \neq \text{Suc } i \text{ mod } n \rangle$
by (*metis One-nat-def mod-Suc mod-if mod-mod-trivial n-not-Suc-n*)

lemmas *suc-mods* = *Suc-mod Suc-mod*[*symmetric*]

lemma *l-suc*: $\langle n > 1 \implies \neg n \leq \text{Suc } 0 \rangle$
by *simp*

lemma *minus-suc*: $\langle n > 0 \implies n - \text{Suc } 0 \neq n \rangle$
by *linarith*

declare *Un-insert-right*[*simp del*] *Un-insert-left*[*simp del*]

11.2 The dining processes definition

context *DiningPhilosophers* **begin**

lemma *RPHIL-restriction-fix-def*:
 $\langle \text{RPHIL } i = (\nu X. \text{picks } i \ i \rightarrow \text{picks } i \ ((i - 1) \text{ mod } N) \rightarrow$
 $\text{putsdwn } i \ ((i - 1) \text{ mod } N) \rightarrow \text{putsdwn } i \ i \rightarrow X) \rangle$
by (*simp add: RPHIL-def restriction-fix-is-fix*)

lemma *LPHIL0-restriction-fix-def*:
 $\langle \text{LPHIL0} = (\nu X. \text{picks } 0 \ (N - 1) \rightarrow \text{picks } 0 \ 0 \rightarrow$
 $\text{putsdwn } 0 \ 0 \rightarrow \text{putsdwn } 0 \ (N - 1) \rightarrow X) \rangle$
by (*simp add: LPHIL0-def restriction-fix-is-fix*)

lemma *FORK-restriction-fix-def*:

$\langle \text{FORK } i = (v \ X. (\text{picks } i \ i \rightarrow \text{putsdown } i \ i \rightarrow X) \square$
 $\quad (\text{picks } ((i + 1) \bmod N) \ i \rightarrow \text{putsdown } ((i + 1) \bmod N) \ i \rightarrow X)) \rangle$
 by (*simp add: FORK-def restriction-fix-is-fix*)

11.2.1 Unfolding rules

lemmas *RPHIL-rec* = *cont-process-rec*[*OF RPHIL-def*[*THEN meta-eq-to-obj-eq*],
simplified]
and *LPHIL0-rec* = *cont-process-rec*[*OF LPHIL0-def*[*THEN meta-eq-to-obj-eq*],
simplified]
and *FORK-rec* = *cont-process-rec*[*OF FORK-def*[*THEN meta-eq-to-obj-eq*],
simplified]

11.3 Translation into normal form

lemma *N-pos*[*simp*]: $\langle N > 0 \rangle$
 using *N-g1 neq0-conv* by *blast*

lemmas *N-pos-simps*[*simp*] = *suc-mods*[*OF N-g1*] *l-suc*[*OF N-g1*] *minus-suc*[*OF N-pos*]

11.3.1 FORK, LPHIL0 and RPHIL are normalizable

Definition of one *fork* and one *philosopher* automata

type-synonym *id_fork* = *nat*

type-synonym σ_{fork} = *nat*

type-synonym *id_phil* = *nat*

type-synonym σ_{phil} = *nat*

definition *fork-A* :: $\langle id_{fork} \Rightarrow (\sigma_{fork}, \text{dining-event}, \text{unit}) \ A_d \rangle \langle A_f \rangle$

where $\langle A_f \ i \equiv \text{recursive-constructor-}A_d$

$\quad [((0, \text{picks } i \ i), \ [1]), ((0, \text{picks } ((i + 1) \bmod N) \ i), \ [2]),$
 $\quad ((1, \text{putsdown } i \ i), \ [0]), ((2, \text{putsdown } ((i + 1) \bmod N) \ i), \ [0])]]$

$(\lambda\sigma. \diamond) \rangle$

definition *rphil-A* :: $\langle id_{phil} \Rightarrow (\sigma_{phil}, \text{dining-event}, \text{unit}) \ A_d \rangle \langle A_{rp} \rangle$

where $\langle A_{rp} \ i \equiv \text{recursive-constructor-}A_d$

$\quad [((0, \text{picks } i \ i), \ [1]), ((1, \text{picks } i \ ((i - 1) \bmod N)), \ [2]),$
 $\quad ((2, \text{putsdown } i \ ((i - 1) \bmod N)), \ [3]), ((3, \text{putsdown } i \ i), \ [0])]]$

$(\lambda\sigma. \diamond) \rangle$

definition *lphil0-A* :: $\langle \sigma_{phil}, \text{dining-event}, \text{unit} \rangle \ A_d \rangle \langle A_{lp} \rangle$

where $\langle A_{lp} \equiv \text{recursive-constructor-}A_d$

$$[(0, \text{picks } 0 (N - 1)), [1]], ((1, \text{picks } 0 0), [2]), \\ ((2, \text{putsdwn } 0 0), [3]), ((3, \text{putsdwn } 0 (N - 1)), [0])] (\lambda\sigma. \diamond)$$

Definition and first properties of associated normal processes

definition *fork-P-d* :: $\langle id_{fork} \Rightarrow \sigma_{fork} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle \text{fork-P-d } i \equiv P\langle A_f i \rangle_d \rangle$

definition *rphil-P-d* :: $\langle id_{rphil} \Rightarrow \sigma_{rphil} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle \text{rphil-P-d } i \equiv P\langle A_{rp} i \rangle_d \rangle$

definition *lphil0-P-d* :: $\langle \sigma_{rphil} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle \text{lphil0-P-d} \equiv P\langle A_{lp} \rangle_d \rangle$

lemmas *fork-P-d-rec* = *P-d-rec*[of $\langle A_f - \rangle$, *folded fork-P-d-def*]
and *rphil-P-d-rec* = *P-d-rec*[of $\langle A_{rp} - \rangle$, *folded rphil-P-d-def*]
and *lphil0-P-d-rec* = *P-d-rec*[of $\langle A_{lp} \rangle$, *folded lphil0-P-d-def*]

schematic-goal *fork-ε*: $\langle \varepsilon (A_f i) \sigma = ?S \rangle$
and *rphil-ε*: $\langle \varepsilon (A_{rp} i) \sigma = ?T \rangle$
and *lphil0-ε*: $\langle \varepsilon A_{lp} \sigma = ?U \rangle$
unfolding *fork-A-def* *rphil-A-def* *lphil0-A-def* **by** (all ε -*det-calc*)

schematic-goal *fork-τ*: $\langle \tau (A_f i) \sigma = ?S \rangle$
and *rphil-τ*: $\langle \tau (A_{rp} i) \sigma = ?T \rangle$
and *lphil0-τ*: $\langle \tau A_{lp} \sigma = ?U \rangle$
unfolding *fork-A-def* *rphil-A-def* *lphil0-A-def* **by** (all τ -*det-calc*)

corollary *ev-id_{fork}x*: $\langle e \in \varepsilon (A_f i) \sigma \implies \text{fork } e = i \rangle$
and *rphil-phil*: $\langle e \in \varepsilon (A_{rp} i) \sigma \implies \text{phil } e = i \rangle$
and *lphil0-phil*: $\langle e \in \varepsilon A_{lp} \sigma \implies \text{phil } e = 0 \rangle$
by (*auto simp add: fork-ε rphil-ε lphil0-ε split: if-split-asm*)

corollary *ev-id_{rphil}x*: $\langle i < n \implies \sigma \in \varepsilon ((A_{lp} \# \text{map } A_{rp} [1..< n]) ! i) s \implies \text{phil } \sigma = i \rangle$
by (*cases* $\langle i = 0 \rangle$) (*simp-all add: lphil0-phil rphil-phil*)

lemma *indep-forks*: $\langle i \neq j \implies \varepsilon (A_f i) \sigma \cap \varepsilon (A_f j) \sigma' = \{\} \rangle$
and *indep-phils*: $\langle i \neq 0 \implies \varepsilon A_{lp} \sigma \cap \varepsilon (A_{rp} i) \sigma' = \{\} \rangle$
 $\langle i \neq j \implies \varepsilon (A_{rp} i) \sigma \cap \varepsilon (A_{rp} j) \sigma' = \{\} \rangle$
using *ev-id_{fork}x* *lphil0-phil* *rphil-phil* **by** (*blast, simp, fastforce, blast*)

Equalities between *FORK*, *RPHIL*, *LPHILO* and respectively *fork-P-d*, *rphil-P-d*, *lphil0-P-d*

lemma *FORK-is-fork-P-d*: $\langle \text{FORK } i = \text{fork-P-d } i 0 \rangle$
proof (*unfold fork-P-d-def*)

have $\langle 0 :: \text{nat} \rangle < 2 \rangle$ **by** *simp*
thus $\langle \text{FORK } i = P\langle A_f \ i \rangle_d \ 0 \rangle$
proof (*induct rule: P-d-induct-iterated*)
show $\langle \text{FORK } i = X \ 0 \implies \text{FORK } i = (P\text{-d-step } (\varepsilon (A_f \ i)) (\tau (A_f \ i)) \ \sim\sim 2) \ X \ 0 \rangle$ **for** X
by (*subst FORK-rec*)
(auto simp add: Mprefix-Det-Mprefix write0-def numeral-eq-Suc fork-ε fork-τ Un-commute intro!: mono-Mprefix-eq)
qed *simp-all*
qed

lemma *RPHIL-is-rphil-P-d*: $\langle \text{RPHIL } i = \text{rphil-P-d } i \ 0 \rangle$
proof (*unfold rphil-P-d-def*)
have $\langle 0 :: \text{nat} \rangle < 4 \rangle$ **by** *simp*
thus $\langle \text{RPHIL } i = P\langle A_{rp} \ i \rangle_d \ 0 \rangle$
proof (*induct rule: P-d-induct-iterated*)
show $\langle \text{RPHIL } i = X \ 0 \implies \text{RPHIL } i = (P\text{-d-step } (\varepsilon (A_{rp} \ i)) (\tau (A_{rp} \ i)) \ \sim\sim 4) \ X \ 0 \rangle$ **for** X
by (*subst RPHIL-rec*)
(auto simp add: write0-def numeral-eq-Suc rphil-ε rphil-τ intro!: mono-Mprefix-eq)
qed *simp-all*
qed

lemma *LPHILO-is-lphil0-P-d*: $\langle \text{LPHILO} = \text{lphil0-P-d } 0 \rangle$
proof (*unfold lphil0-P-d-def*)
have $\langle 0 :: \text{nat} \rangle < 4 \rangle$ **by** *simp*
thus $\langle \text{LPHILO} = P\langle A_{lp} \rangle_d \ 0 \rangle$
proof (*induct rule: P-d-induct-iterated*)
show $\langle \text{LPHILO} = X \ 0 \implies \text{LPHILO} = (P\text{-d-step } (\varepsilon A_{lp}) (\tau A_{lp}) \ \sim\sim 4) \ X \ 0 \rangle$
for X
by (*subst LPHILO-rec*)
(auto simp add: write0-def numeral-eq-Suc lphil0-ε lphil0-τ intro!: mono-Mprefix-eq)
qed *simp-all*
qed

11.3.2 FORKS is normalizable

Definition of the all-forks automaton

type-synonym $\sigma_{\text{forks}} = \langle \text{nat list} \rangle$

definition *forks-A* :: $\langle (\sigma_{\text{forks}}, \text{dining-event}, \text{unit}) \ A_d \rangle (\langle A_F \rangle)$ **where** $\langle A_F \equiv \langle \langle_d \otimes \{\} \rangle \ \text{map } A_f \ [0..<N] \rangle \rangle$

Definition and first properties of the associated normal process

definition *forks-P-d*:: $\langle \sigma_{\text{forks}} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle \text{forks-P-d} \equiv P\langle A_F \rangle_d \rangle$

lemma forks- ε : $\langle \text{length } fs = N \implies \varepsilon A_F fs = (\bigcup i < N. \varepsilon (A_f i) (fs ! i)) \rangle$
unfolding forks-A-def using $N\text{-pos}$ **by** $(\text{subst } \varepsilon\text{-iterated-combine}_d\text{-Sync-general-form})$
force+

Equality between *FORKS* and *forks-P-d*

lemma NFORKS-is-forks-P-d: $\langle \text{FORKS} = \text{forks-P-d } (\text{replicate } N \ 0) \rangle$
unfolding *forks-P-d-def forks-A-def FORKS-def*
apply $(\text{subst } P\text{-d-compactification-Sync-upt-version})$
by $(\text{simp-all add: FORK-is-fork-P-d}[\text{unfolded fork-P-d-def}] \text{ indep-forks indep-enabl-def})$

11.3.3 PHILS is normalizable

Definition of the all-philosophers automaton

type-synonym $\sigma_{phils} = \langle \text{nat list} \rangle$

definition phils-A :: $\langle (\sigma_{phils}, \text{dining-event}, \text{unit}) A_d \rangle (\langle A_P \rangle)$ **where** $\langle A_P \equiv \langle \langle_d \otimes [\{\}] \rangle \rangle$
 $A_{lp} \# \text{map } A_{rp} [1..<N] \rangle \rangle$

lemma phils-A-def-bis: $\langle A_P = \langle \langle_d \otimes [\{\}] \rangle \text{map } (\lambda i. \text{if } i = 0 \text{ then } A_{lp} \text{ else } A_{rp} \ i) [0..<N] \rangle \rangle$
unfolding *phils-A-def apply* $(\text{subst } (2) \text{upt-rec, simp})$
apply $(\text{subgoal-tac } \langle \text{map } (\lambda i. \text{if } i = 0 \text{ then } A_{lp} \text{ else } A_{rp} \ i) [Suc \ 0..<N] = \text{map } A_{rp} [Suc \ 0..<N] \rangle)$
by $(\text{presburger, subst list-eq-iff-nth-eq, simp})$

Definition and first properties of the associated normal process

definition phils-P-d:: $\langle \sigma_{phils} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle \text{phils-P-d} \equiv P \langle \langle A_P \rangle \rangle_d \rangle$

lemma phils- ε : $\langle \text{length } ps = N \implies \varepsilon A_P ps = \varepsilon A_{lp} (ps ! 0) \cup (\bigcup i \in \{1..<N\}. \varepsilon (A_{rp} \ i) (ps ! i)) \rangle$
unfolding *phils-A-def-bis using* $N\text{-pos}$
by $(\text{subst } \varepsilon\text{-iterated-combine}_d\text{-Sync-general-form, (auto split: if-splits)})$

Equality between *PHILS* and *phils-P-d*

lemma NPHILS-is-phils-P-d: $\langle \text{PHILS} = \text{phils-P-d } (\text{replicate } N \ 0) \rangle$
unfolding *phils-P-d-def phils-A-def-bis PHILS-def*
apply $(\text{subst } P\text{-d-compactification-Sync-upt-version}[\text{symmetric}])$
apply $(\text{simp-all add: indep-enabl-def indep-phils}(1,2) \text{ inf-sup-aci}(1))$
apply $(\text{subgoal-tac } \langle \{0..<N\} = \text{insert } 0 \ \{1..<N\} \rangle)$
apply $(\text{simp add: LPHIL0-is-lphil0-P-d lphil0-P-d-def})$
by $(\text{rule arg-cong}[OF \ \text{image-mset-cong}] \text{ (auto simp add: RPHIL-is-rphil-P-d rphil-P-d-def)})$

11.3.4 The complete process *DINING* is normalizable

Definition of the dining automaton

definition *dining-A* :: $\langle (\sigma_{phils} \times \sigma_{forks}, \text{dining-event}, \text{unit}) A_d \rangle (\langle A_D \rangle)$ **where** $\langle A_D \equiv \langle A_P \text{ } d \otimes \llbracket UNIV \rrbracket_{Pair} A_F \rangle \rangle$

Definition and first properties of the associated normal process

definition *dining-P-d*:: $\langle \sigma_{phils} \times \sigma_{forks} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle \text{dining-P-d} \equiv P \langle A_D \rangle_d \rangle$

lemma *dining-ε*:

$\langle \text{length } ps = N \implies \text{length } fs = N \implies$
 $\varepsilon A_D (ps, fs) = (\varepsilon A_{lp} (ps ! 0) \cup (\bigcup_{i \in \{1..<N\}} \varepsilon (A_{rp} i) (ps ! i))) \cap (\bigcup_{i < N.}$
 $\varepsilon (A_f i) (fs ! i)) \rangle$
by (*simp add: dining-A-def ε-combine_{Pair}-Sync combine-Sync-ε-def forks-ε phils-ε*)

Equality between *DINING* and *dining-P-d*

lemma *DINING-is-dining-P-d*: $\langle \text{DINING} = \text{dining-P-d} (\text{replicate } N \ 0, \text{replicate } N \ 0) \rangle$

unfolding *dining-P-d-def dining-A-def*
apply (*subst P-d-combine_{Pair}-Sync[symmetric]*)
apply (*simp add: indep-enabl-def*)
by (*simp add: DINING-def NFORKS-is-forks-P-d NPHILS-is-phils-P-d Sync-commute forks-P-d-def phils-P-d-def*)

11.4 And finally: Philosophers may dine ! Always !

method *ε-sets-simp uses opt = (simp-all split: if-split-asm)?,*
simp-all add: fork-ε lphil0-ε rphil-ε opt split: if-splits

method *A-defs-simp uses opt = (simp-all split: if-split-asm)?,*
simp-all add: fork-A-def lphil0-A-def rphil-A-def opt split: if-splits

11.4.1 Construction of an invariant for the dining automaton

definition $\langle \text{inv-dining } ps \ fs \equiv$
 $\text{length } fs = N \wedge \text{length } ps = N$
 $\wedge (\forall i < N. fs ! i = 0 \vee fs ! i = 1 \vee fs ! i = 2)$
 $\wedge (\forall i < N. ps ! i = 0 \vee ps ! i = 1 \vee ps ! i = 2 \vee ps ! i = 3)$
 $\wedge (\forall i. \text{Suc } i < N \longrightarrow ((fs ! \text{Suc } i = 1) \longleftrightarrow ps ! \text{Suc } i \neq 0)) \wedge (fs ! (N$
 $- 1) = 2 \longleftrightarrow ps ! 0 \neq 0)$
 $\wedge (\forall i < N - 1. fs ! i = 2 \longleftrightarrow ps ! \text{Suc } i = 2) \wedge (fs ! 0 =$
 $1 \longleftrightarrow ps ! 0 = 2) \rangle$

lemma *show-inv-dining*:

$\langle \text{length } fs = N \wedge \text{length } ps = N \implies$
 $(\forall i < N. fs ! i = 0 \vee fs ! i = 1 \vee fs ! i = 2) \implies$
 $(\forall i < N. ps ! i = 0 \vee ps ! i = 1 \vee ps ! i = 2 \vee ps ! i = 3) \implies$
 $(\forall i. \text{Suc } i < N \longrightarrow (fs ! \text{Suc } i = 1 \longleftrightarrow ps ! \text{Suc } i \neq 0)) \implies (fs ! (N - 1) =$
 $2 \longleftrightarrow ps ! 0 \neq 0) \implies$

$(\forall i < N - 1. fs ! i = 2 \longleftrightarrow ps ! Suc i = 2) \implies (fs ! 0 = 1 \longleftrightarrow ps ! 0 = 2)$
 \implies
 $\langle inv-dining ps fs \rangle$
by (*simp add: inv-dining-def*)

lemma *inv-DINING*: $\langle s \in \mathcal{R}_d A_D (replicate N 0, replicate N 0) \implies inv-dining (fst s) (snd s) \rangle$

proof(*induct rule: $\mathcal{R}_d.induct$*)

case *init* **show** *?case* **by** (*simp add: inv-dining-def*)

next

case (*step t u e*)

obtain *t-ps t-fs u-ps u-fs* **where** *t-pair*: $\langle t = (t-ps, t-fs) \rangle$ **and** *u-pair*: $\langle u = (u-ps, u-fs) \rangle$ **by** *fastforce*

hence *inv-hyp*: $\langle length t-fs = N \rangle \langle length t-ps = N \rangle$

$\langle i < N \implies t-fs ! i = 0 \vee t-fs ! i = 1 \vee t-fs ! i = 2 \rangle$

$\langle i < N \implies t-ps ! i = 0 \vee t-ps ! i = 1 \vee t-ps ! i = 2 \vee t-ps ! i = 3 \rangle$

$\langle Suc i < N \implies (t-fs ! Suc i = 1) = (t-ps ! Suc i \neq 0) \rangle$

$\langle (t-fs ! (N - 1) = 2) = (t-ps ! 0 \neq 0) \rangle$

$\langle i < N - 1 \implies (t-fs ! i = 2) = (t-ps ! Suc i = 2) \rangle$

$\langle (t-fs ! 0 = 1) = (t-ps ! 0 = 2) \rangle$ **for** *i*

using *step.hyps(2)[simplified inv-dining-def]* **by** *simp-all*

have *u-in- \mathcal{R}_d -prem*: $\langle (u-ps, u-fs) \in \mathcal{R}_d A_P (replicate N 0) \times \mathcal{R}_d A_F (replicate N 0) \rangle$

using *$\mathcal{R}_d.step[OF step.hyps(1, 3), simplified dining-A-def]$*

by (*simp add: u-pair[symmetric], rule set-mp[OF $\mathcal{R}_d.combine_{dPair-Sync-subset}$]*)

note *u-in- $\mathcal{R}_d = u-in-\mathcal{R}_d$ -prem[simplified mem-Times-iff, THEN conjunct1, simplified]*

u-in- \mathcal{R}_d -prem[simplified mem-Times-iff, THEN conjunct2, simplified]

have *same-length-u*: $\langle length u-ps = N \rangle \langle length u-fs = N \rangle$

using *same-length- \mathcal{R}_d -iterated-combine_d-Sync-description[rotated, OF u-in- $\mathcal{R}_d(1)$][unfolded phils-A-def]*

same-length- \mathcal{R}_d -iterated-combine_d-Sync-description[rotated, OF u-in- $\mathcal{R}_d(2)$][unfolded forks-A-def]

by *simp+*

have *u-is*: $\langle [u-ps] = \tau A_P t-ps e \rangle \langle [u-fs] = \tau A_F t-fs e \rangle$

using *step(3)[simplified dining-A-def, simplified combine-Sync-defs]*

by (*simp-all add: t-pair u-pair option.case-eq-if map-option-case split: if-splits*)

have *e-in*: $\langle e \in \varepsilon A_{lp} (t-ps ! 0) \cup (\bigcup i \in \{1..<N\}. \varepsilon (A_{rp} i) (t-ps ! i)) \rangle$

$\langle e \in (\bigcup i < N. \varepsilon (A_f i) (t-fs ! i)) \rangle$

by (*subst phils- ε [symmetric], fact inv-hyp(2), simp add: ε -simps,metis u-is(1)*)

(*subst forks- ε [symmetric], fact inv-hyp(1), simp add: ε -simps,metis u-is(2)*)

have *u-nth*:

$\langle i < N \implies u\text{-ps} ! i =$
 (if $i = 0$ then (if $e \in \varepsilon A_{lp} (t\text{-ps} ! 0)$ then $\lceil \tau A_{lp} (t\text{-ps} ! 0) e \rceil$ else $t\text{-ps} ! 0$)
 else if $e \in \varepsilon (A_{rp} i) (t\text{-ps} ! i)$ then $\lceil \tau (A_{rp} i) (t\text{-ps} ! i) e \rceil$ else $t\text{-ps} ! i$)
 $\langle i < N \implies u\text{-fs} ! i =$
 (if $e \in \varepsilon (A_f i) (t\text{-fs} ! i)$ then $\lceil \tau (A_f i) (t\text{-fs} ! i) e \rceil$ else $t\text{-fs} ! i$) **for** i
by (insert $u\text{-is}(1)$, unfold $\text{phils-}A\text{-def}$, subst (asm) τ -iterated-combine_d-Sync-general-form,
 simp-all add: $\text{inv-hyp}(2)$ split: if-splits)
 (insert $u\text{-is}(2)$, unfold $\text{forks-}A\text{-def}$, subst (asm) τ -iterated-combine_d-Sync-general-form,
 simp-all add: $\text{inv-hyp}(1)$ split: if-splits)
have $\langle e \in \varepsilon A_P t\text{-ps} \rangle \langle e \in \varepsilon A_F t\text{-fs} \rangle$ **using** $u\text{-is} \varepsilon\text{-simps}$ **by** fastforce+
hence $e\text{-equiv}$: $\langle e \in \varepsilon A_{lp} (t\text{-ps} ! 0) \longleftrightarrow \text{phil } e = 0 \rangle$
 $\langle \text{Suc } i < N \implies e \in \varepsilon (A_{rp} (\text{Suc } i)) (t\text{-ps} ! \text{Suc } i) \longleftrightarrow \text{phil } e = \text{Suc } i \rangle$
 $\langle i < N \implies e \in \varepsilon (A_f i) (t\text{-fs} ! i) \longleftrightarrow \text{fork } e = i \rangle$ **for** i
apply (simp-all add: $\text{phils-}\varepsilon[\text{OF } \text{inv-hyp}(2)]$ $\text{forks-}\varepsilon[\text{OF } \text{inv-hyp}(1)]$)
using $\text{rphil-phil lphil0-phil ev-id}_{\text{fork}x}$ **by** auto ($\text{metis Suc-le-eq less-irrefl-nat}$,
 blast , metis)

show $?case$
proof (simp add: $u\text{-pair}$, rule $\text{show-inv-dining}[\text{rule-format}]$, simp add: same-length-u ,
 goal-cases)
case (1 i) **thus** $?case$ **using** $u\text{-nth}(2)[\text{of } i]$ $\text{inv-hyp}(3)$ **by** $\varepsilon\text{-sets-simp } A\text{-defs-simp}$
next
case (2 i) **thus** $?case$ **using** $u\text{-nth}(1)[\text{of } i]$ $\text{inv-hyp}(4)$ **by** $\varepsilon\text{-sets-simp } A\text{-defs-simp}$
next
case (3 i)
with $u\text{-nth}(1)[\text{of } \langle \text{Suc } i \rangle]$ $u\text{-nth}(2)[\text{of } \langle \text{Suc } i \rangle]$ **show** $?case$
using $\text{inv-hyp}(5)[\text{of } i]$ **apply** $\varepsilon\text{-sets-simp}$ **apply** $A\text{-defs-simp}$
using $e\text{-equiv}(3)$ $\text{fork-}\varepsilon$ $e\text{-equiv}(2)$ $\text{rphil-}\varepsilon$ **by** fastforce+
next
case 4
with $u\text{-nth}(1)[\text{of } 0]$ $u\text{-nth}(2)$ **show** $?case$ **using** $\text{inv-hyp}(6)$ $N\text{-g1}$ **apply**
 $\varepsilon\text{-sets-simp}$ **apply** $A\text{-defs-simp}$
apply ($\text{metis } N\text{-pos One-nat-def Suc-pred fork-}\varepsilon \text{ dining-event.sel}(3)$
 $\text{dining-event.simps}(3)$ $\text{inv-hyp}(3)$ $\text{lessI singletonD } e\text{-equiv}(3)$)
using $\text{lphil0-}\varepsilon$ $e\text{-equiv}(1)$ **by** force+
next
case (5 i)
hence $\langle \text{Suc } i < N \rangle$ **by** linarith
with $u\text{-nth}(1)[\text{of } \langle \text{Suc } i \rangle]$ $u\text{-nth}(2)[\text{of } i]$ 5 **show** $?case$
using $\text{inv-hyp}(7)[\text{of } i]$ **apply** $\varepsilon\text{-sets-simp}$ **apply** $A\text{-defs-simp}$
apply ($\text{metis Suc-lessD fork-}\varepsilon \text{ dining-event.sel}(3)$ $\text{dining-event.simps}(3)$
 $\text{singletonD } e\text{-equiv}(3)$)
apply ($\text{metis One-nat-def Suc-lessD bot-nat-0.not-eq-extremum inv-hyp}(3)$)
using $\text{rphil-}\varepsilon$ $e\text{-equiv}(2)$ **by** force+
next
case 6
with $u\text{-nth}(1)[\text{of } 0]$ $u\text{-nth}(2)[\text{of } 0]$ **show** $?case$
using $N\text{-g1 inv-hyp}(8)$ **apply** (simp split: if-split-asm) **apply** $\varepsilon\text{-sets-simp}$
apply $A\text{-defs-simp}$

```

    using lphil0-ε e-equiv(1) fork-ε e-equiv(3) by force+
  qed
qed

```

11.4.2 The invariant *inv-dining* implies that *DINING* is dead-lock-free

method *nonempty-Int-by-common-element* for $x = \text{rule-tac } ex\text{-in-conv}[THEN \text{iff}D1, OF \text{ex}I, OF \text{Int}I, \text{of } x]$

lemma *inv-implies-DF*: $\langle \varepsilon A_D (ps, fs) \neq \{\} \rangle$ if *hyp-inv*: $\langle \text{inv-dining } ps \text{ fs} \rangle$

```

  apply (subst dining-ε)
  apply (insert hyp-inv, unfold inv-dining-def, simp-all add: lphil0-ε)
proof(elim conjE, intro conjI impI, goal-cases)
  case 1
  with 1(3)[rule-format, of 0, OF N-pos] show ?case
proof(elim disjE, goal-cases)
  case 11:1
  thus ?case
  using 1(3)[rule-format, of 1, OF N-g1] apply(elim disjE)

    apply (subgoal-tac ⟨ps ! 1 = 0⟩, nonempty-Int-by-common-element ⟨picks 1
1⟩)
  using N-g1 apply ε-sets-simp[3]
    apply (metis atLeastLessThan-iff le-refl less-irrefl-nat, blast,
metis less-zeroE linorder-neqE-nat)

    apply (cases ⟨ps ! 1 = 1⟩, nonempty-Int-by-common-element ⟨picks 1 0⟩)
  using N-g1 apply ε-sets-simp[2]
    apply (metis One-nat-def atLeastLessThan-iff diff-self-eq-0 le-refl less-numeral-extra(1)
mod-mod-trivial mod-self,
metis N-pos lessThan-iff mod-less)
    apply (nonempty-Int-by-common-element ⟨putsdown 1 1⟩)
  using N-g1 apply ε-sets-simp[2]
    apply (metis atLeastLessThan-iff le-refl less-numeral-extra(3) zero-less-diff,
metis gr0-conv-Suc lessThan-iff)

    apply (cases ⟨N = 2⟩, simp)
  apply (subgoal-tac ⟨ps ! 2 = 2⟩, nonempty-Int-by-common-element ⟨putsdown
2 1⟩)
  using N-g1 apply ε-sets-simp
  by (metis One-nat-def Suc-lessI atLeastLessThan-iff diff-Suc-1 le-SucI le-numeral-extra(4)
mod-less n-not-Suc-n numeral-2-eq-2 zero-less-Suc,
metis One-nat-def Suc-1 Suc-lessI gr0-conv-Suc lessThan-iff
less-diff-conv mod-less plus-1-eq-Suc,
metis One-nat-def Suc-1)
next
  case 12:2
  thus ?case by linarith

```

```

next
  case 13:3
  thus ?case
  apply (subgoal-tac ⟨ps ! 1 = 2⟩, nonempty-Int-by-common-element ⟨putsdown
1 0⟩)
  using N-g1 apply ε-sets-simp
  apply (metis atLeastLessThan-iff diff-self-eq-0 dvd-1-left le-Suc-eq less-2-cases-iff
mod-0 odd-one)
  by (metis 13(10) N-pos lessThan-iff mod-less n-not-Suc-n numeral-2-eq-2
zero-less-Suc)
  qed
next
case 2
from 2(3, 7, 8, 10) N-g1 have f1: ⟨fs ! 0 ≠ 0 ⇒ ps ! 1 = 2 ∧ fs ! 0 = 2⟩
by auto
from 2 show ?case
  apply (cases ⟨fs ! 0 = 0⟩)
  apply (nonempty-Int-by-common-element ⟨picks 0 0⟩)
  using N-g1 apply ε-sets-simp[2]
  using N-pos apply blast
  apply (nonempty-Int-by-common-element ⟨putsdown 1 0⟩)
  apply ε-sets-simp
  apply (metis N-g1 One-nat-def atLeastLessThan-iff bot-nat-0.not-eq-extremum
cancel-comm-monoid-add-class.diff-cancel dual-order.refl local.f1 mod-0)
  by (metis N-g1 N-pos One-nat-def lessThan-iff less-irrefl-nat mod-less)
next
case 3 thus ?case by (nonempty-Int-by-common-element ⟨putsdown 0 0⟩)
(ε-sets-simp, metis N-pos lessThan-iff zero-less-Suc)
next
case 4 thus ?case by (nonempty-Int-by-common-element ⟨putsdown 0 (N -
1)⟩)
(ε-sets-simp, metis N-pos One-nat-def Suc-1 Suc-diff-1 diff-less
gr0-conv-Suc lessThan-iff mod-self n-not-Suc-n)
next
case 5 thus ?case using 5(4)[rule-format, of 0] by simp
qed

```

11.4.3 Conclusion

```

corollary deadlock-free-DINING: ⟨deadlock-free DINING⟩
  unfolding DINING-is-dining-P-d dining-P-d-def
  apply (subst deadlock-free-P-d-iff)
  using inv-DINING inv-implies-DF by force

```

11.5 Alternative version with only right-handed philosophers (in order to show that it's not deadlock-free)

11.5.1 Setup

definition $\langle RPHILS \equiv \parallel P \in \# \text{ mset } (\text{map } RPHIL [0..< N]). P \rangle$

corollary $\langle N = 3 \implies RPHILS = (RPHIL\ 0 \parallel RPHIL\ 1 \parallel RPHIL\ 2) \rangle$
unfolding $RPHILS\text{-def}$ by (*simp add: eval-nat-numeral upt-rec Sync-assoc*)

definition $RDINING :: \langle \text{dining-event process} \rangle$
where $\langle RDINING = (FORKS \parallel RPHILS) \rangle$

11.5.2 Normalization

definition $rphils\text{-}A :: \langle (\sigma_{rphils}, \text{dining-event}, \text{unit}) A_d \rangle (\langle A_{RP} \rangle)$ **where** $\langle A_{RP} \equiv \langle \langle A_d \otimes \{\{\}\} \text{ map } A_{rp} [0..< N] \rangle \rangle \rangle$

definition $rphils\text{-}P\text{-}d :: \langle \sigma_{rphils} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle rphils\text{-}P\text{-}d \equiv P \langle \langle A_{RP} \rangle \rangle_d \rangle$

definition $rdining\text{-}A :: \langle (\sigma_{rphils} \times \sigma_{forks}, \text{dining-event}, \text{unit}) A_d \rangle (\langle A_{RD} \rangle)$ **where**
 $\langle A_{RD} \equiv \langle \langle A_{RP} \text{ }_d \otimes \langle UNIV \rangle_{Pair} A_F \rangle \rangle \rangle$

definition $rdining\text{-}P\text{-}d :: \langle \sigma_{rphils} \times \sigma_{forks} \Rightarrow \text{dining-event process} \rangle$ **where** $\langle rdining\text{-}P\text{-}d \equiv P \langle \langle A_{RD} \rangle \rangle_d \rangle$

11.5.3 Correspondance between our normalized processes and the previous definitions

lemma $rphils\text{-}\varepsilon :: \langle \text{length } ps = N \implies \varepsilon A_{RP} ps = (\bigcup i \in \{0..< N\}. \varepsilon (A_{rp} i) (ps ! i)) \rangle$

unfolding $rphils\text{-}A\text{-}def$ using $N\text{-}pos$
by (*subst $\varepsilon\text{-iterated-combine}_d\text{-Sync-general-form}$, (*auto split: if-splits*)*)

lemma $NRPHILS\text{-}is\text{-}rphils\text{-}P\text{-}d :: \langle RPHILS = rphils\text{-}P\text{-}d (\text{replicate } N\ 0) \rangle$

unfolding $rphils\text{-}P\text{-}d\text{-}def$ $rphils\text{-}A\text{-}def$ $RPHILS\text{-}def$
apply (*subst $P\text{-}d\text{-compactification-Sync-upt-version}$*)
by (*simp-all add: RPHIL-is-rphil-P-d[unfolded rphil-P-d-def] indep-phils indep-enabl-def*)

lemma $rdining\text{-}\varepsilon ::$

$\langle \text{length } ps = N \implies \text{length } fs = N \implies \varepsilon A_{RD} (ps, fs) = (\bigcup i \in \{0..< N\}. \varepsilon (A_{rp} i) (ps ! i)) \cap (\bigcup i < N. \varepsilon (A_f i) (fs ! i)) \rangle$
by (*simp add: rdining-A-def $\varepsilon\text{-combine}_{Pair}\text{-Sync combine-Sync-}\varepsilon\text{-def forks-}\varepsilon rphils\text{-}\varepsilon$*)

lemma $RDINING\text{-}is\text{-}rdining\text{-}P\text{-}d :: \langle RDINING = rdining\text{-}P\text{-}d (\text{replicate } N\ 0, \text{replicate } N\ 0) \rangle$

apply (*unfold rdining-P-d-def rdining-A-def*)
apply (*subst P-d-combine_{P_{air}}-Sync[symmetric]*)
apply (*simp add: indep-enabl-def*)
by (*simp add: NFORKS-is-forks-P-d NRPHILS-is-rphils-P-d RDINING-def Sync-commute forks-P-d-def rphils-P-d-def*)

11.5.4 Proof that we have a deadlock in the state (*replicate N 1, replicate N 1*)

lemma *empty-enabl-replicate1*: $\langle \varepsilon A_{RD} (\text{replicate } N \ 1, \text{replicate } N \ 1) = \{\} \rangle$
by (*subst rdining- ε , auto simp add: rphil- ε fork- ε*)

corollary *non-deadlock-free-rdining*: $\langle \neg \text{deadlock-free} (\text{rdining-P-d} (\text{replicate } N \ 1, \text{replicate } N \ 1)) \rangle$

unfolding *rdining-P-d-def*
by (*subst P-d-rec, subst empty-enabl-replicate1, simp add: non-deadlock-free-STOP*)

11.5.5 Proof that this state is reachable from our initial state, i.e. (*replicate N 1, replicate N 1*) $\in \mathcal{R}_d A_{RD} (\text{replicate } N \ 0, \text{replicate } N \ 0)$

lemma *rdining- τ* : $\langle \text{length } ps = N \implies \text{length } fs = N \implies e \in \varepsilon A_{RD} (ps, fs) \implies \tau A_{RD} (ps, fs) e = [(\tau A_{RP} ps e), [\tau A_F fs e]] \rangle$
by (*auto simp add: rdining-A-def combine_{P_{air}}-Sync-defs ε -simps split: if-split-asm*)

lemma *replicate1-reachable-from-replicate0-prelim*:

$\langle n \leq N \implies (\text{replicate } n \ 1 \ @ \ \text{replicate } (N - n) \ 0, \text{replicate } n \ 1 \ @ \ \text{replicate } (N - n) \ 0) \in \mathcal{R}_d A_{RD} (\text{replicate } N \ 0, \text{replicate } N \ 0) \rangle$

proof (*induct n, simp add: \mathcal{R}_d .init*)

case (*Suc n*)

have $\langle n \leq N \rangle$ **by** (*simp add: Suc.premS Suc-leD*)

define $\sigma s \ \sigma t$ **where** $a1$: $\langle \sigma s \equiv \text{replicate } n \ (1::\text{nat}) \ @ \ \text{replicate } (N - n) \ 0 \rangle$

and $a2$: $\langle \sigma t \equiv \text{replicate } (\text{Suc } n) \ (1::\text{nat}) \ @ \ \text{replicate } (N - \text{Suc } n) \ 0 \rangle$

have $\langle \text{length } \sigma s = N \rangle \langle \text{length } \sigma t = N \rangle \langle \text{length } \sigma s = \text{length } \sigma t \rangle$

by (*simp-all add: $\langle n \leq N \rangle \ a1 \ a2 \ \text{Suc.premS}$*)

have $f1$: $\langle (\sigma s, \sigma s) \in \mathcal{R}_d A_{RD} (\text{replicate } N \ 0, \text{replicate } N \ 0) \rangle$

using *Suc.hyps(1) a1 Suc.premS Suc-leD* **by** *presburger*

have $f2$: $\langle \text{picks } n \ n \in \varepsilon A_{RP} \ \sigma s \rangle \langle \text{picks } n \ n \in \varepsilon A_F \ \sigma s \rangle \langle \text{picks } n \ n \in \varepsilon A_{RD} (\sigma s, \sigma s) \rangle$

by (*subst rphils- ε forks- ε , insert Suc.premS Suc-le-eq Suc-leD a1, auto simp add: rdining- ε rphil- ε fork- ε nth-append*)**+**

have $\langle a \in \varepsilon (A_{rp} \ i) (\sigma s \ ! \ i) \implies i < N \implies a \in \varepsilon (A_{rp} \ j) (\sigma s \ ! \ j) \implies j < N \implies j = i \rangle$ **for** $i \ j \ a$

by (*metis rphil-phil*)

hence $*$: $\langle a \in \varepsilon (A_{rp} \ i) (\sigma s \ ! \ i) \implies i < N \implies$

$(\text{THE } j. j < N \wedge a \in \varepsilon (\text{map } A_{rp} [0..<N] \ ! \ j) (\sigma s \ ! \ j)) = i \rangle$ **for** $i \ a$ **by**

auto

have $\langle a \in \varepsilon (A_f i) (\sigma s ! i) \implies i < N \implies a \in \varepsilon (A_f j) (\sigma s ! j) \implies j < N \implies j = i \rangle$ **for** $i j a$

by (*metis ev-id_{fork}x*)

hence $** : \langle a \in \varepsilon (A_f i) (\sigma s ! i) \implies i < N \implies$

$(THE j. j < N \wedge a \in \varepsilon (map A_f [0..<N] ! j) (\sigma s ! j)) = i \rangle$ **for** $i a$ **by**

auto

have $f3: \langle \sigma t = [\tau A_{RP} \sigma s (picks n n)] \rangle$

apply (*unfold rphils-A-def, subst τ -iterated-combine_d-Sync-general-form-when-indep*)

subgoal by (*simp add: $\langle length \sigma s = N \rangle$*)

subgoal by (*simp add: indep-enabl indep-phils(2)*)

using N -pos **apply** (*auto simp del: N -pos*)

subgoal by (*metis N -g1 N -pos lessThan-iff rphil-phil zero-neq-one*)

subgoal

apply (*subst *, simp-all*)

apply (*auto simp add: $\langle length \sigma s = length \sigma t \rangle$ intro!: nth-equalityI*)

apply (*auto simp add: a1 a2 nth-Cons nth-append nth-list-update dest: rphil-phil split: if-split-asm nat.split*)

by (*simp-all add: rphil-A-def*)

using $rphils$ - ε $\langle length \sigma s = N \rangle$ *atLeast0LessThan* $f2(1)$ **by** *force*

have $f4: \langle \sigma t = [\tau A_F \sigma s (picks n n)] \rangle$

apply (*unfold forks-A-def, subst τ -iterated-combine_d-Sync-general-form-when-indep*)

subgoal by (*simp add: $\langle length \sigma s = N \rangle$*)

subgoal by (*simp add: indep-enabl-def indep-forks*)

using N -pos **apply** (*auto simp del: N -pos*)

subgoal by (*metis N -g1 N -pos ev-id_{fork}x lessThan-iff zero-neq-one*)

subgoal

apply (*subst **, simp-all*)

apply (*auto simp add: $\langle length \sigma s = length \sigma t \rangle$ intro!: nth-equalityI*)

apply (*auto simp add: a1 a2 nth-Cons nth-append nth-list-update dest: ev-id_{fork}x split: if-split-asm nat.split*)

using N -pos-simps(1) *Suc-lessI* **by** (*auto simp add: fork-A-def intro: Suc-lessI*)

using $forks$ - ε $\langle length \sigma s = N \rangle$ *atLeast0LessThan* $f2(2)$ **by** *force*

show $\langle (\sigma t, \sigma t) \in \mathcal{R}_d A_{RD} (replicate N 0, replicate N 0) \rangle$

apply (*rule \mathcal{R}_d .step[OF $f1$], subst rdining- τ*)

using *Suc.premis a1* **apply** *fastforce*

apply (*rule $f2$*)

using $f3 f4$ **by** *blast*

qed

corollary *replicate1-reachable-from-replicate0*: $\langle (replicate N 1, replicate N 1) \in \mathcal{R}_d A_{RD} (replicate N 0, replicate N 0) \rangle$

by (*simp add: replicate1-reachable-from-replicate0-prelim[of N , simplified]*)

theorem *not-deadlock-free-RDINING*: $\langle \neg \text{deadlock-free } RDINING \rangle$
apply (*subst RDINING-is-rdining-P-d[unfolded rdining-P-d-def]*)
apply (*subst deadlock-free-P-d-iff*)
using *empty-enabl-replicate1 replicate1-reachable-from-replicate0* **by** *blast*

end

Chapter 12

Other Results similar to Compactification

Unlike *Sync* and $(;)$, some operators like *Det* do not enjoy a compactification result. Nevertheless, we still can prove some useful lemmas.

12.1 Some preliminary Results

lemma *Mprefix-Det-Mprefix-bis* :

$\langle (\Box a \in A \rightarrow P a) \Box (\Box b \in B \rightarrow Q b) =$
 $(\Box x \in (A \cap B) \rightarrow P x \sqcap Q x) \Box (\Box a \in (A - B) \rightarrow P a) \Box (\Box b \in (B - A) \rightarrow$
 $Q b) \rangle$
(is $\langle ?lhs = ?rhs \rangle$)

proof (*subst Process-eq-spec, safe*)

show $\langle (t, X) \in \mathcal{F} ?lhs \implies (t, X) \in \mathcal{F} ?rhs \rangle$ **for** $t X$

by (*cases t*) (*auto simp add: F-Det F-Mprefix F-Ndet*)

next

show $\langle (t, X) \in \mathcal{F} ?rhs \implies (t, X) \in \mathcal{F} ?lhs \rangle$ **for** $t X$

by (*cases t, simp-all add: F-Mprefix F-Ndet F-Det D-Det T-Det disjoint-iff*)

blast+

next

show $\langle t \in \mathcal{D} ?lhs \implies t \in \mathcal{D} ?rhs \rangle$ **for** t

by (*auto simp add: D-Det D-Mprefix image-iff D-Ndet*)

next

show $\langle t \in \mathcal{D} ?rhs \implies t \in \mathcal{D} ?lhs \rangle$ **for** t

by (*auto simp add: D-Mprefix D-Ndet D-Det split: if-split-asm*)

qed

lemma *GlobalNdet-Ndet-GlobalNdet*:

$\langle A \neq \{\} \implies B \neq \{\} \implies (\Box a \in A. P a) \sqcap (\Box b \in B. Q b) =$
 $\Box x \in (A \cup B). (if x \in A \cap B then P x \sqcap Q x else if x \in A then P x else Q x) \rangle$
by (*simp add: Process-eq-spec F-Ndet D-Ndet F-GlobalNdet D-GlobalNdet, safe*)
(*auto simp add: F-Ndet D-Ndet split: if-splits*)

lemma *GlobalNdet-Ndet-GlobalNdet-bis:*

$\langle A \cap B \neq \{\} \implies A - B \neq \{\} \implies B - A \neq \{\} \implies$
 $(\sqcap a \in A. P a) \sqcap (\sqcap b \in B. Q b) =$
 $(\sqcap x \in (A \cap B). P x \sqcap Q x) \sqcap (\sqcap a \in (A - B). P a) \sqcap (\sqcap b \in (B - A). Q b) \rangle$
by (*auto simp add: Process-eq-spec F-Ndet D-Ndet F-GlobalNdet D-GlobalNdet*)

lemma *GlobalNdet-GlobalNdet:*

$\langle (\sqcap a \in A. \sqcap b \in B a. P b) =$
 $(\text{if } \forall a \in A. B a \neq \{\} \text{ then } \sqcap b \in (\bigcup a \in A. B a). P b \text{ else } (\sqcap b \in (\bigcup a \in A. B a).$
 $a). P b) \sqcap \text{STOP} \rangle$
by (*auto simp add: Process-eq-spec F-GlobalNdet D-GlobalNdet F-Ndet D-Ndet F-STOP D-STOP*)

12.2 Results for *Det*

lemma *P-nd-set-almost-compactification-Det :*

$\langle (\sqcap (s, A) \in s\text{-}A\text{-set}. P\langle\langle A \rangle\rangle_{nd} s) =$
 $\sqcap e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$
 $\sqcap (s, A) \in \{(s, A) \in s\text{-}A\text{-set}. e \in \varepsilon A s\}.$
 $\sqcap s' \in \tau A s e. P\langle\langle A \rangle\rangle_{nd} s' \rangle$ (**is** $\langle ?lhs = ?rhs \rangle$)

proof –

have $\langle ?lhs = (\sqcap (s, A) \in s\text{-}A\text{-set}. P\text{-nd-step } (\varepsilon A) (\tau A) P\langle\langle A \rangle\rangle_{nd} s) \rangle$

by (*auto intro: mono-GlobalDet-eq arg-cong[OF P-nd-rec]*)

also have $\langle \dots = \sqcap s\text{-}A \in s\text{-}A\text{-set}. P\text{-nd-step } (\varepsilon (snd\ s\text{-}A)) (\tau (snd\ s\text{-}A)) P\langle\langle snd$
 $s\text{-}A \rangle\rangle_{nd} (fst\ s\text{-}A) \rangle$

by (*simp only: case-prod-beta'*)

also have $\langle \dots = \sqcap e \in (\bigcup s\text{-}A \in s\text{-}A\text{-set}. \varepsilon (snd\ s\text{-}A) (fst\ s\text{-}A)) \rightarrow$

$\sqcap s\text{-}A \in \{s\text{-}A \in s\text{-}A\text{-set}. e \in \varepsilon (snd\ s\text{-}A) (fst\ s\text{-}A)\}.$

$GlobalNdet (\tau (snd\ s\text{-}A) (fst\ s\text{-}A) e) P\langle\langle snd\ s\text{-}A \rangle\rangle_{nd} \rangle$

by (*simp add: GlobalDet-Mprefix*)

also have $\langle \dots = ?rhs \rangle$ **by** (*simp add: case-prod-beta'*)

finally show $\langle ?lhs = ?rhs \rangle .$

qed

lemma *P-nd-set-almost-compactification-Det-bis :*

$\langle (\sqcap (s, A) \in s\text{-}A\text{-set}. P\langle\langle A \rangle\rangle_{nd} s) =$
 $\sqcap e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$
 $\sqcap (s', A) \in \{(s', A) \mid s' s A. (s, A) \in s\text{-}A\text{-set} \wedge e \in \varepsilon A s \wedge s' \in \tau A s e\}. P\langle\langle A \rangle\rangle_{nd}$
 $s' \rangle$

(**is** $\langle - = ?rhs \rangle$)

by (*subst P-nd-set-almost-compactification-Det, intro mono-Mprefix-eq*)

(*auto simp add: Process-eq-spec GlobalNdet-projs ε -simps split: if-split-asm, blast+*)

lemma *P-d-set-almost-compactification-Det:*

shows $\langle \square (s, A) \in s\text{-}A\text{-set}. P\langle A \rangle_d s =$
 $\square e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$
 $\square (s, A) \in \{(s, A) \in s\text{-}A\text{-set}. e \in \varepsilon A s\}. P\langle A \rangle_d \lceil \tau A s e \rceil \rangle$ (**is** $\langle ?lhs =$
 $?rhs \rangle$)
proof –
have $\langle ?lhs = (\square (s, A) \in s\text{-}A\text{-set}. P\text{-}d\text{-step} (\varepsilon A) (\tau A) P\langle A \rangle_d s) \rangle$
by (*auto intro: mono-GlobalDet-eq arg-cong[OF P-d-rec]*)
also have $\langle \dots = \square s\text{-}A \in s\text{-}A\text{-set}. P\text{-}d\text{-step} (\varepsilon (snd\ s\text{-}A)) (\tau (snd\ s\text{-}A)) P\langle snd$
 $s\text{-}A \rangle_d (fst\ s\text{-}A) \rangle$
by (*simp only: case-prod-beta'*)
also have $\langle \dots = \square e \in (\bigcup s\text{-}A \in s\text{-}A\text{-set}. \varepsilon (snd\ s\text{-}A) (fst\ s\text{-}A)) \rightarrow$
 $\square s\text{-}A \in \{s\text{-}A \in s\text{-}A\text{-set}. e \in \varepsilon (snd\ s\text{-}A) (fst\ s\text{-}A)\}.$
 $P\langle snd\ s\text{-}A \rangle_d \lceil \tau (snd\ s\text{-}A) (fst\ s\text{-}A) e \rceil \rangle$
by (*simp add: GlobalDet-Mprefix*)
also have $\langle \dots = ?rhs \rangle$ **by** (*simp add: case-prod-beta'*)
finally show $\langle ?lhs = ?rhs \rangle$.
qed

lemma *P-d-set-almost-compactification-Det-bis:*

shows $\langle \square (s, A) \in s\text{-}A\text{-set}. P\langle A \rangle_d s =$
 $\square e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$
 $\square (s', A) \in \{(\lceil \tau A s e \rceil, A) \mid s\text{-}A. (s, A) \in s\text{-}A\text{-set} \wedge e \in \varepsilon A s\}. P\langle A \rangle_d s' \rangle$
by (*subst P-d-set-almost-compactification-Det, intro mono-Mprefix-eq*)
(auto simp add: Process-eq-spec GlobalNdet-projs ε -simps, (metis option.sel)+)

12.3 Results for *Ndet*

12.4 Other Operators

12.4.1 *initials*

lemma *initials-P_{SKIP}S-nd :*

$\langle (P_{SKIP}S\langle A \rangle_{nd} \sigma)^0 = (if\ \sigma \in \varrho\ A\ then\ tick\ ' \omega\ A\ \sigma\ else\ ev\ ' \varepsilon\ A\ \sigma) \rangle$
by (*subst P_{SKIP}S-nd-rec*) (*simp add: initials-Mprefix ϱ -simps*)

lemma *initials-P_{SKIP}S-d :*

$\langle (P_{SKIP}S\langle A \rangle_d \sigma)^0 = (if\ \sigma \in \varrho\ A\ then\ \{\checkmark(\lceil \omega\ A\ \sigma \rceil)\} else\ ev\ ' \varepsilon\ A\ \sigma) \rangle$
by (*subst P_{SKIP}S-d-rec*) (*auto simp add: initials-Mprefix ϱ -simps*)

lemma *initials-P-nd :* $\langle (P\langle A \rangle_{nd} s)^0 = ev\ ' \varepsilon\ A\ s \rangle$

by (*subst P-nd-rec*) (*simp-all add: initials-Mprefix*)

lemma *initials-P-d :* $\langle (P\langle A \rangle_d s)^0 = ev\ ' \varepsilon\ A\ s \rangle$

by (subst P-d-rec) (simp add: initials-Mprefix)

12.4.2 Throw

lemma *Throw-PSKIPS-nd* :

$\langle P_{SKIPS}\langle A \rangle_{nd} \sigma \Theta b \in B. Q b =$
 (if $\sigma \in \varrho A$ then $SKIPS (\omega A \sigma)$ else
 $\square a \in \varepsilon A \sigma \rightarrow$ (if $a \in B$ then $Q a$ else $\sqcap \sigma' \in \tau A \sigma a. (P_{SKIPS}\langle A \rangle_{nd} \sigma' \Theta b)$)
 $\in B. Q b) \rangle$

by (auto simp add: P_{SKIPS}-nd-rec-in- ϱ Throw-disjoint-events-of
 events-of-SKIPS P_{SKIPS}-nd-rec-notin- ϱ Throw-Mprefix
 Throw-distrib-GlobalNdet-right
 intro: mono-Mprefix-eq)

lemma *Throw-PSKIPS-d* :

$\langle P_{SKIPS}\langle A \rangle_d \sigma \Theta b \in B. Q b =$
 (if $\sigma \in \varrho A$ then $SKIP [\omega A \sigma]$ else
 $\square a \in \varepsilon A \sigma \rightarrow$ (if $a \in B$ then $Q a$ else $P_{SKIPS}\langle A \rangle_d [\tau A \sigma a] \Theta b \in B. Q b) \rangle$

by (simp add: P_{SKIPS}-d-rec-in- ϱ P_{SKIPS}-d-rec-notin- ϱ Throw-Mprefix)

lemma *Throw-P-nd* :

$\langle P\langle A \rangle_{nd} \sigma \Theta b \in B. Q b =$
 $\square a \in \varepsilon A \sigma \rightarrow$ (if $a \in B$ then $Q a$ else $\sqcap \sigma' \in \tau A \sigma a. (P\langle A \rangle_{nd} \sigma' \Theta b \in B. Q b) \rangle$

by (subst P-nd-rec)

(auto simp add: Throw-Mprefix Throw-distrib-GlobalNdet-right intro: mono-Mprefix-eq)

lemma *Throw-P-d* :

$\langle P\langle A \rangle_d \sigma \Theta b \in B. Q b =$
 $\square a \in \varepsilon A \sigma \rightarrow$ (if $a \in B$ then $Q a$ else $P\langle A \rangle_d [\tau A \sigma a] \Theta b \in B. Q b) \rangle$

by (subst P-d-rec) (simp add: Throw-Mprefix)

12.4.3 (Δ)

lemma *SKIPS-Interrupt-is-SKIPS-Det* :

$\langle SKIPS R \Delta P = SKIPS R \square P \rangle$

by (auto simp add: SKIPS-def Interrupt-distrib-GlobalNdet-right

Det-distrib-GlobalNdet-right SKIP-Interrupt-is-SKIP-Det intro: mono-GlobalNdet-eq)

lemma *Interrupt-PSKIPS-nd* :

$\langle P_{SKIPS}\langle A \rangle_{nd} \sigma \Delta Q =$
 $Q \square$ (if $\sigma \in \varrho A$ then $SKIPS (\omega A \sigma)$ else $\square a \in \varepsilon A \sigma \rightarrow \sqcap \sigma' \in \tau A \sigma a.$
 $P_{SKIPS}\langle A \rangle_{nd} \sigma' \Delta Q) \rangle$

by (subst P_{SKIPS}-nd-rec)

(auto simp add: Interrupt-Mprefix SKIPS-Interrupt-is-SKIPS-Det Det-commute
 ϱ -simps ε -simps Interrupt-distrib-GlobalNdet-right
 intro!: arg-cong2[**where** $f = \langle \square \rangle$] mono-Mprefix-eq split: if-split-asm)

lemma *Interrupt-PSKIPS-d* :

$\langle P_{SKIPS}\langle A \rangle_d \sigma \Delta Q =$

$Q \sqcap (\text{if } \sigma \in \varrho A \text{ then SKIP } [\omega A \sigma] \text{ else } \sqcap a \in \varepsilon A \sigma \rightarrow P_{SKIPS}\langle\langle A \rangle\rangle_d [\tau A \sigma a] \Delta Q)$

by (*simp add: Det-commute Interrupt-Mprefix P_{SKIPS}-d-rec-in- ϱ P_{SKIPS}-d-rec-notin- ϱ SKIP-Interrupt-is-SKIP-Det*)

lemma *Interrupt-P-nd* :

$\langle P\langle\langle A \rangle\rangle_{nd} \sigma \Delta Q = Q \sqcap (\sqcap a \in \varepsilon A \sigma \rightarrow \sqcap \sigma' \in \tau A \sigma a. P\langle\langle A \rangle\rangle_{nd} \sigma' \Delta Q) \rangle$

by (*subst P-nd-rec*)

(*auto simp add: Interrupt-Mprefix Interrupt-distrib-GlobalNdet-right ε -simps intro!: arg-cong2[**where** $f = \langle(\sqcap)\rangle$] mono-Mprefix-eq split: if-split-asm*)

lemma *Interrupt-P-d* :

$\langle P\langle\langle A \rangle\rangle_d \sigma \Delta Q = Q \sqcap (\sqcap a \in \varepsilon A \sigma \rightarrow P\langle\langle A \rangle\rangle_d [\tau A \sigma a] \Delta Q) \rangle$

by (*metis Interrupt-Mprefix P-d-rec*)

12.4.4 After

context *After*

begin

lemma *After-SKIPS* : $\langle \text{SKIPS } R \text{ after } a = \Psi (\text{SKIPS } R) a \rangle$

by (*simp add: Process-eq-spec After-projs image-iff*)

lemma *After-P_{SKIPS}-nd* :

$\langle P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \sigma \text{ after } a =$

(*if* $\sigma \in \varrho A$ *then* $\Psi (\text{SKIPS } (\omega A \sigma)) a$ *else*

if $a \in \varepsilon A \sigma$ *then* $\sqcap \sigma' \in \tau A \sigma a. P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \sigma'$ *else* $\Psi (P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \sigma)$

$a) \rangle$

by (*subst (1 4) P_{SKIPS}-nd-rec*)

(*simp add: ε -simps ϱ -simps After-SKIPS After-Mprefix*)

lemma *After-P_{SKIPS}-d* :

$\langle P_{SKIPS}\langle\langle A \rangle\rangle_d \sigma \text{ after } a =$

(*if* $\sigma \in \varrho A$ *then* $\Psi (\text{SKIP } [\omega A \sigma]) a$ *else*

if $a \in \varepsilon A \sigma$ *then* $P_{SKIPS}\langle\langle A \rangle\rangle_d [\tau A \sigma a]$ *else* $\Psi (P_{SKIPS}\langle\langle A \rangle\rangle_d \sigma) a) \rangle$

by (*subst (1 2) P_{SKIPS}-d-rec*)

(*auto simp add: ε -simps ϱ -simps After-SKIP After-Mprefix*)

lemma *After-P-nd* :

$\langle P\langle\langle A \rangle\rangle_{nd} \sigma \text{ after } a = (\text{if } a \in \varepsilon A \sigma \text{ then } \sqcap \sigma' \in \tau A \sigma a. P\langle\langle A \rangle\rangle_{nd} \sigma' \text{ else } \Psi (P\langle\langle A \rangle\rangle_{nd} \sigma) a) \rangle$

by (*subst (1 4) P-nd-rec (simp add: ε -simps After-Mprefix)*)

lemma *After-P-d* :

$\langle P\langle\langle A \rangle\rangle_d \sigma \text{ after } a = (\text{if } a \in \varepsilon A \sigma \text{ then } P\langle\langle A \rangle\rangle_d [\tau A \sigma a] \text{ else } \Psi (P\langle\langle A \rangle\rangle_d \sigma) a) \rangle$

by (*subst (1 2) P-d-rec*)

(*simp add: ε -simps After-SKIP After-Mprefix*)

end

context *AfterExt*

begin

lemma *After_{tick}-SKIPS* :

$\langle \text{SKIPS } R \text{ after } \checkmark e = (\text{case } e \text{ of } ev \ a \Rightarrow \Psi (\text{SKIPS } R) \ a \mid \checkmark(r) \Rightarrow \Omega (\text{SKIPS } R) \ r) \rangle$

by (*cases e*) (*simp-all add: After_{tick}-def After-SKIPS*)

lemma *After_{tick}-P_{SKIPS}-nd* :

$\langle P_{SKIPS} \langle A \rangle_{nd} \ \sigma \text{ after } \checkmark e =$
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } \sigma \in \varrho \ A \text{ then } \Psi (\text{SKIPS } (\omega \ A \ \sigma)) \ a \text{ else}$
 $\text{if } a \in \varepsilon \ A \ \sigma \text{ then } \sqcap \sigma' \in \tau \ A \ \sigma \ a. \ P_{SKIPS} \langle A \rangle_{nd} \ \sigma' \text{ else } \Psi$
 $(P_{SKIPS} \langle A \rangle_{nd} \ \sigma) \ a$
 $\mid \checkmark(r) \Rightarrow \text{if } \sigma \in \varrho \ A \text{ then } \Omega (\text{SKIPS } (\omega \ A \ \sigma)) \ r \text{ else } \Omega (P_{SKIPS} \langle A \rangle_{nd}$
 $\sigma) \ r) \rangle$

proof (*cases e*)

show $\langle e = ev \ a \Rightarrow ?thesis \rangle$ **for** *a*

by (*simp add: After-P_{SKIPS}-nd After_{tick}-def*)

next

show $\langle e = \checkmark(r) \Rightarrow ?thesis \rangle$ **for** *r*

by (*subst (1 5) P_{SKIPS}-nd-rec*) (*simp add: After_{tick}-def ϱ -simps*)

qed

lemma *After_{tick}-P_{SKIPS}-d* :

$\langle P_{SKIPS} \langle A \rangle_d \ \sigma \text{ after } \checkmark e =$
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } \sigma \in \varrho \ A \text{ then } \Psi (\text{SKIP } [\omega \ A \ \sigma]) \ a \text{ else}$
 $\text{if } a \in \varepsilon \ A \ \sigma \text{ then } P_{SKIPS} \langle A \rangle_d \ [\tau \ A \ \sigma \ a] \text{ else } \Psi (P_{SKIPS} \langle A \rangle_d$
 $\sigma) \ a$
 $\mid \checkmark(r) \Rightarrow \text{if } \sigma \in \varrho \ A \text{ then } \Omega (\text{SKIP } [\omega \ A \ \sigma]) \ r \text{ else } \Omega (P_{SKIPS} \langle A \rangle_d \ \sigma)$
 $r) \rangle$

proof (*cases e*)

show $\langle e = ev \ a \Rightarrow ?thesis \rangle$ **for** *a*

by (*simp add: After-P_{SKIPS}-d After_{tick}-def*)

next

show $\langle e = \checkmark(r) \Rightarrow ?thesis \rangle$ **for** *r*

by (*subst (1 2) P_{SKIPS}-d-rec*) (*auto simp add: After_{tick}-def ϱ -simps*)

qed

lemma *After_{tick}-P-nd* :

$\langle P \langle A \rangle_{nd} \ \sigma \text{ after } \checkmark e =$
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } a \in \varepsilon \ A \ \sigma \text{ then } \sqcap \sigma' \in \tau \ A \ \sigma \ a. \ P \langle A \rangle_{nd} \ \sigma' \text{ else } \Psi (P \langle A \rangle_{nd}$
 $\sigma) \ a$
 $\mid \checkmark(r) \Rightarrow \Omega (P \langle A \rangle_{nd} \ \sigma) \ r) \rangle$

proof (*cases e*)

show $\langle e = ev \ a \Rightarrow ?thesis \rangle$ **for** *a*

by (*simp add: After-P-nd After_{tick}-def*)

next

show $\langle e = \mathcal{V}(r) \implies ?thesis \rangle$ **for** r
by (*subst* (1 2) *P-nd-rec*) (*auto simp add: After_{tick}-def*)
qed

lemma *After_{tick}-P-d* :
 $\langle P\langle A \rangle_d \sigma \text{ after } \mathcal{V} e =$
(case e of ev a \implies if $a \in \varepsilon A \sigma$ then $P\langle A \rangle_d [\tau A \sigma a]$ else $\Psi (P\langle A \rangle_d \sigma) a$
 $| \mathcal{V}(r) \implies \Omega (P\langle A \rangle_d \sigma) r \rangle$
proof (*cases e*)
show $\langle e = ev a \implies ?thesis \rangle$ **for** a
by (*simp add: After-P-d After_{tick}-def*)
next
show $\langle e = \mathcal{V}(r) \implies ?thesis \rangle$ **for** r
by (*subst* (1 2) *P-d-rec*) (*auto simp add: After_{tick}-def*)
qed

end

12.5 OpSem

context *OpSemTransitions*
begin

lemma *SKIPS- τ -trans-SKIP* : $\langle r \in R \implies SKIPS R \rightsquigarrow_{\tau} SKIP r \rangle$
by (*simp add: SKIPS-def τ -trans-GlobalNdet*)

In the ProcOmata, we will absorb the τ transitions that appear when we unfold the fixed-point operator.

lemma *τ -trans- P_{SKIPS} -nd* :
 $\langle r \in \omega A \sigma \implies P_{SKIPS}\langle A \rangle_{nd} \sigma \rightsquigarrow_{\tau} SKIP r \rangle$
by (*subst P_{SKIPS} -nd-rec*) (*auto simp add: SKIPS- τ -trans-SKIP*)

lemma *τ -trans- P_{SKIPS} -d* :
 $\langle \sigma \in \varrho A \implies P_{SKIPS}\langle A \rangle_d \sigma \rightsquigarrow_{\mathcal{V}[\omega A \sigma]} \Omega (SKIP [\omega A \sigma]) [\omega A \sigma] \rangle$
by (*auto simp add: P_{SKIPS} -d-rec SKIP-trans-tick- Ω -SKIP ϱ -simps*)

lemma *ev-trans- P_{SKIPS} -nd* :
 $\langle \sigma \notin \varrho A \implies \sigma' \in \tau A \sigma a \implies P_{SKIPS}\langle A \rangle_{nd} \sigma \rightsquigarrow_a P_{SKIPS}\langle A \rangle_{nd} \sigma' \rangle$
by (*subst P_{SKIPS} -nd-rec*)
(auto simp add: ϱ -simps ε -simps
intro: ev-trans- τ -trans[OF ev-trans-Mprefix τ -trans-GlobalNdet])

lemma *ev-trans- P_{SKIPS} -d* :
 $\langle \sigma \notin \varrho A \implies a \in \varepsilon A \sigma \implies P_{SKIPS}\langle A \rangle_d \sigma \rightsquigarrow_a P_{SKIPS}\langle A \rangle_d [\tau A \sigma a] \rangle$
by (*subst P_{SKIPS} -d-rec*)

(*auto simp add: ϱ -simps intro: ev-trans-Mprefix*)

lemma *ev-trans-P-nd* :

$\langle \sigma' \in \tau A \sigma a \implies P\langle\langle A \rangle\rangle_{nd} \sigma \rightsquigarrow_a P\langle\langle A \rangle\rangle_{nd} \sigma' \rangle$

by (*subst P-nd-rec*)

(*auto simp add: ε -simps*

intro: ev-trans- τ -trans[OF ev-trans-Mprefix τ -trans-GlobalNdet])

lemma *ev-trans-P-d* :

$\langle a \in \varepsilon A \sigma \implies P\langle\langle A \rangle\rangle_d \sigma \rightsquigarrow_a P\langle\langle A \rangle\rangle_d [\tau A \sigma a] \rangle$

by (*subst P-d-rec*) (*auto intro: ev-trans-Mprefix*)

end

Chapter 13

Conclusion

13.1 Entry Point

This is where `HOL-CSP_Proc-Omata` should be imported from.

13.2 Conclusion

In this entry we have developed the Proc-Omata framework on top of `HOL-CSP` and its extensions. Starting from functional automata, we introduced Proc-Omata in four variants: deterministic, terminating deterministic, non-deterministic, and terminating non-deterministic. They enjoy strong structural properties, for example deadlocks can be characterized directly and established by invariant reasoning:

$$\frac{\text{deadlock-free } (P \llbracket A \rrbracket_{nd} \sigma) = (\forall \sigma' \in \mathcal{R}_{nd} A \sigma. \varepsilon A \sigma' \neq \emptyset)}{\text{\textit{\rho-disjoint-}\varepsilon A}}{\text{deadlock-free}_{SKIPS} (P_{SKIPS} \llbracket A \rrbracket_{nd} \sigma) = (\forall \sigma' \in \mathcal{R}_{nd} A \sigma. \sigma' \in \rho A \vee \varepsilon A \sigma' \neq \emptyset)}$$

We then lifted sequential composition and synchronization product to the automata level, by defining suitable combination functions and proving their correctness. A major generalization of our development is the treatment of parameterized termination. For sequential composition we worked directly with the generalized operator $(; \checkmark)$, since the standard one $(;)$ is easily recovered (indeed $P ; \checkmark (\lambda r. Q) = P ; Q$). In contrast, for synchronization product we had to provide two distinct versions, as the handling of ticks prevents any straightforward reduction from $P \llbracket A \rrbracket Q$ to $P \llbracket A \rrbracket \checkmark Q$.

Another central ingredient is the library `Restriction_Spaces` [1]. Proc-Omata are indeed defined as fixed points of endofunctions which, in the non-deterministic case, are not always continuous due to global non-deterministic

choice. While deterministic prefix choice does not suffice to restore continuity under composition, it does guarantee constructiveness, allowing us to rely on the fixed-point operator $\nu x. f x$ in all cases.

The resulting framework yields compactification theorems that support invariant-based reasoning over large process architectures:

$$\frac{|\sigma s| = |As|}{\llbracket E \rrbracket (\sigma, A) \in \#mset (zip \sigma s As). P \langle\langle A \rangle\rangle_{nd} \sigma = P \langle\langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle\rangle_{nd} \sigma s}$$

Finally, we demonstrated the applicability of our approach with the Dining Philosophers case study, where Proc-Omata compactification enables proofs that scale to an arbitrary number of participants in this parameterized process architecture.

Bibliography

- [1] B. Ballenghien, B. Puyobro, and B. Wolff. Restriction spaces: a fixed-point theory. *Archive of Formal Proofs*, May 2025. https://isa-afp.org/entries/Restriction_Spaces.html, Formal proof development.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] B. Ballenghien and B. Wolff. A theory of proc-omataand proof methods for process architectures. In *Theoretical Aspects of Computing IC-TAC 2024: 21st International Colloquium, Bangkok, Thailand, November 2529, 2024, Proceedings*, page 272289, Berlin, Heidelberg, 2024. Springer-Verlag.
- [6] 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.
- [7] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1997.
- [8] S. Taha, B. Wolff, and L. Ye. Philosophers may dine - definitively! In *Integrated Formal Methods: 16th International Conference, IFM 2020, Lugano, Switzerland, November 1620, 2020, Proceedings*, page 419439, Berlin, Heidelberg, 2020. Springer-Verlag.