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

Benoît Ballenghien Burkhart Wolff

February 4, 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 *FORK*, *LPHIL0* and *RPHIL* are normalizable . . . 164
11.3.2 *FORKS* is normalizable . . . 166
11.3.3 *PHILS* is normalizable . . . 167
11.3.4 The complete process *DINING* 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 *inv-dining* implies that *DINING* is *deadlock-free* . . . 171
11.4.3 Conclusion . . . 172
11.5 Alternative version with only right-handed philosophers (in order to show that it's not *deadlock-free*) . . . 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 (*replicate N 1*, *replicate N 1*) . . . 174
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} (*replicate N 0*, *replicate N 0*) . . . 174

12 Other Results similar to Compactification **177**
12.1 Some preliminary Results . . . 177
12.2 Results for *Det* . . . 178
12.3 Results for *Ndet* . . . 179
12.4 Other Operators . . . 179
12.4.1 *initials* . . . 179
12.4.2 *Throw* . . . 180
12.4.3 ($\triangle$) . . . 180
12.4.4 After . . . 181
12.5 OpSem . . . 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* :: ‹$('a, 'r)$ $process_{ptick}$ $\Rightarrow$ *bool*› (‹$determ^0$›)
  where ‹$determ^0\ P \equiv \forall\, e \in P^0.\ \{e\} \notin \mathcal{R}\ P$›

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

lemma *accepts-initials-def-bis*:
‹$determ^0\ P \longleftrightarrow (\forall\, e \in P^0.\ \forall\, X \in \mathcal{R}\ P.\ e \notin X)$›
by (*auto simp add*: *accepts-initials-def*)
(*metis Refusals-iff Un-upper1 insert-Diff insert-is-Un is-processT4*)

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

2.1.2 First properties

lemma *accepts-initials-STOP* [*simp*] : ‹$determ^0\ STOP$›
by (*simp add*: *accepts-initials-def*)

lemma *accepts-initials-SKIP* [*simp*] : ‹$determ^0\ (SKIP\ r)$›
by (*simp add*: *accepts-initials-def Refusals-iff F-SKIP*)

lemma *not-accepts-initials-BOT* [*simp*] : ‹$\neg\ determ^0\ \bot$›
by (*simp add*: *accepts-initials-def Refusals-iff F-BOT*)

lemma *accepts-initials-imp-initial-tick-iff-is-SKIP*:
‹$determ^0\ P \Longrightarrow \checkmark(r) \in P^0 \longleftrightarrow P = SKIP\ r$›
proof (*rule iffI*)
show ‹$P = SKIP\ r \Longrightarrow \checkmark(r) \in P^0$› **by** *simp*
next
assume *assms* : ‹$determ^0\ P$› ‹$\checkmark(r) \in P^0$›
hence *initials-is* : ‹$P^0 = \{\checkmark(r)\}$›
by (*auto simp add*: *accepts-initials-def initials-def Refusals-iff subset-iff*)
(*metis append-self-conv2 is-processT6-TR-notin singletonD*)
show ‹$P = SKIP\ r$›
proof (*subst SKIP-F-iff*[*symmetric*], *unfold failure-refine-def*, *safe*)
from *assms* **show** ‹$(s,\ X) \in \mathcal{F}\ P \Longrightarrow (s,\ X) \in \mathcal{F}\ (SKIP\ r)$› **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*:
‹$determ^0\ P \Longrightarrow (range\ tick \cap P^0 = \{\}) \longleftrightarrow P = STOP \vee (\exists\, e.\ ev\ e \in P^0)$›
by (*simp add*: *disjoint-iff image-iff*)
(*metis accepts-initials-imp-initial-tick-iff-is-SKIP all-not-in-conv* $event_{ptick}$*.exhaust*
$event_{ptick}$*.simps*(*3*) *initials-SKIP initials-empty-iff-STOP singletonD*)

2.1.3 Monotonicity

lemma *mono-accepts-initials-F*: ‹$P \sqsubseteq_F Q \Longrightarrow determ^0\ P \Longrightarrow determ^0\ Q$›
  by (*frule anti-mono-initials-F*)
    (*auto simp add*: *failure-refine-def accepts-initials-def Refusals-iff subset-iff*)

lemma *mono-accepts-initials-FD*: ‹$P \sqsubseteq_{FD} Q \Longrightarrow determ^0\ P \Longrightarrow determ^0\ Q$›
  using *leFD-imp-leF mono-accepts-initials-F* **by** *blast*

lemma *mono-accepts-initials*: ‹$P \sqsubseteq Q \Longrightarrow determ^0\ P \Longrightarrow determ^0\ Q$›
  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*]
:
  ‹$adm_\downarrow\ (\lambda x.\ determ^0\ (f\ x))$› **if** ‹$cont_\downarrow\ f$›
for f :: ‹$'b$ :: *restriction* $\Rightarrow$ $('a,\ 'r)\ process_{ptick}$›
proof (*rule restriction-adm-subst*)
  from ‹$cont_\downarrow\ f$› **show** ‹$cont_\downarrow\ f$› **.**
next
  show ‹$adm_\downarrow\ (determ^0 :: ('a,\ 'r)\ process_{ptick} \Rightarrow bool)$›
  proof (*rule restriction-admI*)
    fix σ **and** Σ :: ‹$('a,\ 'r)\ process_{ptick}$›
    assume ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› ‹$determ^0\ (\sigma\ n)$› **for** n
    show ‹$determ^0\ \Sigma$›
    proof (*rule accepts-initialsI*)
      fix e **assume** ‹$e \in \Sigma^0$›
      from ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› **obtain** *n0*
        where $*$: ‹$\Sigma \downarrow Suc\ (Suc\ 0) = \sigma\ n0 \downarrow Suc\ (Suc\ 0)$›
        by (*blast dest*: *restriction-tendstoD*)
      with ‹$e \in \Sigma^0$› **have** ‹$e \in (\sigma\ n0)^0$›
        by (*metis initials-restriction-process$_{ptick}$ nat.distinct*(*1*))
      with ‹$determ^0\ (\sigma\ n0)$› **have** ‹$\{e\} \notin \mathcal{R}\ (\sigma\ n0)$›
        by (*fact accepts-initialsD*)
      with $*$ **show** ‹$\{e\} \notin \mathcal{R}\ \Sigma$›
        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*] : ‹$determ^0\ (\Box a \in A \rightarrow P\ a)$›
  by (*simp add*: *accepts-initials-def initials-Mprefix Refusals-iff F-Mprefix*)

lemma *accepts-initials-write0* [*simp*] : ‹$determ^0\ (a \rightarrow P)$›
  by (*simp add*: *write0-def*)

lemma *accepts-initials-write* [*simp*] : ‹$determ^0\ (c!a \rightarrow P)$›

by (*simp add*: *write-def*)

lemma *accepts-initials-read* [*simp*] : ‹$determ^0$ (c**?**$a \in A \rightarrow P\ a$)›
by (*simp add*: *read-def*)

lemma *accepts-initials-Ndet-iff*:
‹$determ^0\ (P \sqcap Q) \longleftrightarrow determ^0\ P \wedge determ^0\ Q \wedge P^0 = Q^0$›
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*:
‹$determ^0\ (\sqcap a \in A.\ P\ a) \longleftrightarrow$
$(\forall\, a \in A.\ determ^0\ (P\ a) \wedge (\forall\, b \in A.\ (P\ a)^0 = (P\ b)^0))$›
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*:
‹$determ^0\ (\sqcap a \in A \rightarrow P\ a) \longleftrightarrow (\exists\, a.\ A \subseteq \{a\})$›
by (*simp add*: *Mndetprefix-GlobalNdet accepts-initials-GlobalNdet-iff initials-write0*) *blast*

lemma *accepts-initials-ndet-write-iff*:
‹$determ^0\ (c$**!!**$a \in A \rightarrow P\ a) \longleftrightarrow (\exists\, b.\ c\ `\ A \subseteq \{b\})$›
by (*simp add*: *ndet-write-def accepts-initials-Mndetprefix-iff*)

lemma *accepts-initials-SKIPS-iff* :
‹$determ^0\ (SKIPS\ R) \longleftrightarrow R = \{\} \vee (\exists\, r.\ R = \{r\})$›
by (*simp add*: *SKIPS-def accepts-initials-GlobalNdet-iff*) *blast*

lemma *accepts-initials-Det* :
‹$determ^0\ (P \Box Q) \longleftrightarrow P = STOP \vee Q = STOP \vee range\ tick \cap P^0 \cap Q^0 \neq \{\}$
$\vee$
$range\ tick \cap (P^0 \cup Q^0) = \{\}$›
(**is** ‹- $\longleftrightarrow$ *?rhs*›) **if** *accepts-initials* : ‹$determ^0\ P$› ‹$determ^0\ Q$›
proof (*cases* ‹$P = \bot$›; *cases* ‹$Q = \bot$›)
show ‹*accepts-initials* $(P \Box Q) \longleftrightarrow$ *?rhs*› **if** *non-BOT* : ‹$P \neq \bot$› ‹$Q \neq \bot$›
proof (*rule iffI*)
show ‹*?rhs* $\Longrightarrow determ^0\ (P \Box Q)$›
proof (*elim disjE*)
show ‹$P = STOP \Longrightarrow determ^0\ (P \Box Q)$› ‹$Q = STOP \Longrightarrow determ^0\ (P \Box Q)$›
by (*simp-all add*: *accepts-initials*)
next

from *non-BOT accepts-initials*
show ‹*range tick* $\cap$ P^0 $\cap$ Q^0 $\neq$ {} $\Longrightarrow$ *determ*^{0} (P $\Box$ Q)›
‹*range tick* $\cap$ (P^0 $\cup$ Q^0) = {} $\Longrightarrow$ *determ*^{0} (P $\Box$ Q)›
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 $*$: ‹⟦Q $\neq$ *STOP*; ✓(r) $\in$ P^0; ✓(r) $\notin$ Q^0; *determ*^{0} P; *determ*^{0} (P $\Box$ Q)⟧ $\Longrightarrow$ *False*›
for P Q :: ‹($'a$, $'r$) *process*$_{ptick}$› **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 ‹*determ*^{0} (P $\Box$ Q) $\Longrightarrow$ *?rhs*›
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* :
‹*determ*^{0} ($\Box a \in A.\ P\ a$)› **if** ‹$\bigwedge a.\ a \in A \Longrightarrow$ *determ*^{0} (P a)›
‹*range tick* $\cap$ ($\bigcap a \in A.\ (P\ a)^0$) $\neq$ {} $\vee$ *range tick* $\cap$ ($\bigcup a \in A.\ (P\ a)^0$) = {}›
proof (*use that*(*2*) **in** ‹*elim disjE*›)
from *that*(*1*) **show** ‹*range tick* $\cap$ ($\bigcap a{\in}A.\ (P\ a)^0$) $\neq$ {} $\Longrightarrow$ *determ*^{0} ($\Box a \in A.\ P\ a$)›
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** ‹*range tick* $\cap$ ($\bigcup a{\in}A.\ (P\ a)^0$) = {} $\Longrightarrow$ *determ*^{0} ($\Box a \in A.\ P\ a$)›
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*$_{ptick}$:
‹*determ*^{0} (P ;$_{✓}$ Q) $\longleftrightarrow$ ($\forall r.$ ✓(r) $\in$ P^0 $\longrightarrow$ *determ*^{0} (Q r))› **if** ‹*determ*^{0} P›
proof (*intro iffI allI impI*)
show ‹*determ*^{0} (P ;$_{✓}$ Q) $\Longrightarrow$ ✓(r) $\in$ P^0 $\Longrightarrow$ *determ*^{0} (Q r)› **for** r
by (*simp add*: *accepts-initials-imp-initial-tick-iff-is-SKIP* ‹*determ*^{0} P›)
next
have ‹P $\neq$ $\bot$› **using** *not-accepts-initials-BOT that* **by** *blast*
show ‹*determ*^{0} (P ;$_{✓}$ Q)› **if** $*$: ‹$\forall r.$ ✓(r) $\in$ P^0 $\longrightarrow$ *determ*^{0} (Q r)›
proof (*rule accepts-initialsI*)
fix e **assume** ‹e $\in$ (P ;$_{✓}$ Q)^0›
then consider a **where** ‹e = *ev* a› ‹*ev* a $\in$ P^0› | r **where** ‹✓(r) $\in$ P^0› ‹e $\in$

$(Q\ r)^0$›
by (*simp add*: *initials-Seq*$_{ptick}$ ‹$P \neq \bot$›) *blast*
thus ‹$\{e\} \notin \mathcal{R}\ (P\ ;_{\checkmark}\ Q)$›
proof *cases*
from ‹*determ*^{0} P› ‹$P \neq \bot$› **show** ‹$e = ev\ a \Longrightarrow ev\ a \in P^0 \Longrightarrow \{e\} \notin \mathcal{R}\ (P\ ;_{\checkmark}\ Q)$› **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* ‹*determ*^{0} P› *accepts-initials-imp-initial-tick-iff-is-SKIP event*$_{ptick}$*.distinct*(*1*)
initials-SKIP initials-memI insertI1 singletonD)
next
show ‹$\checkmark(r) \in P^0 \Longrightarrow e \in (Q\ r)^0 \Longrightarrow \{e\} \notin \mathcal{R}\ (P\ ;_{\checkmark}\ Q)$› **for** r
by (*simp add*: ‹*determ*^{0} P› *accepts-initialsD accepts-initials-imp-initial-tick-iff-is-SKIP* $*$)
qed
qed
qed

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

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

False
by (*fold Refusals-iff*) (*metis accepts-initialsD-bis* ‹$determ^0$ P› ‹$determ^0$ Q›)
qed
next
show ‹$\{e\} \notin \mathcal{R}$ (P $[\![S]\!]_{\checkmark}$ Q)› **if** ‹$a \notin S$› ‹$ev\ a \in P^0$›
proof (*rule notI*)
assume ‹$\{e\} \in \mathcal{R}$ (P $[\![S]\!]_{\checkmark}$ Q)›
with ‹$P \neq \bot$› ‹$Q \neq \bot$› **obtain** *X-P X-Q*
where ‹([], *X-P*) $\in \mathcal{F}$ P› ‹([], *X-Q*) $\in \mathcal{F}$ Q› ‹$e \in$ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark$) *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** ‹$ev\ a \in$ *X-P*›
by (*simp add*: ‹$e = ev\ a$› ‹$a \notin S$› *super-ref-Sync*$_{ptick}$*-def*)
with ‹$ev\ a \in P^0$› ‹([], *X-P*) $\in \mathcal{F}$ P› **show** *False*
by (*fold Refusals-iff*) (*metis accepts-initialsD-bis* ‹$determ^0$ P›)
qed
next
show ‹$\{e\} \notin \mathcal{R}$ (P $[\![S]\!]_{\checkmark}$ Q)› **if** ‹$a \notin S$› ‹$ev\ a \in Q^0$›
proof (*rule notI*)
assume ‹$\{e\} \in \mathcal{R}$ (P $[\![S]\!]_{\checkmark}$ Q)›
with ‹$P \neq \bot$› ‹$Q \neq \bot$› **obtain** *X-P X-Q*
where ‹([], *X-P*) $\in \mathcal{F}$ P› ‹([], *X-Q*) $\in \mathcal{F}$ Q› ‹$e \in$ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark$) *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** ‹$ev\ a \in$ *X-Q*›
by (*simp add*: ‹$e = ev\ a$› ‹$a \notin S$› *super-ref-Sync*$_{ptick}$*-def*)
with ‹$ev\ a \in Q^0$› ‹([], *X-Q*) $\in \mathcal{F}$ Q› **show** *False*
by (*fold Refusals-iff*) (*metis accepts-initialsD-bis* ‹$determ^0$ Q›)
qed
qed
next
fix *r-s r s* **assume** ‹$e = \checkmark$(*r-s*)› ‹$r \otimes\checkmark$ $s = \lfloor$*r-s*$\rfloor$› ‹$\checkmark(r) \in P^0$› ‹$\checkmark(s) \in Q^0$›
show ‹$\{e\} \notin \mathcal{R}$ (P $[\![S]\!]_{\checkmark}$ Q)›
proof (*rule notI*)
assume ‹$\{e\} \in \mathcal{R}$ (P $[\![S]\!]_{\checkmark}$ Q)›
with ‹$P \neq \bot$› ‹$Q \neq \bot$› **obtain** *X-P X-Q*
where ‹([], *X-P*) $\in \mathcal{F}$ P› ‹([], *X-Q*) $\in \mathcal{F}$ Q› ‹$e \in$ *super-ref-Sync*$_{ptick}$ ($\otimes\checkmark$) *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** ‹$\checkmark(r) \in$ *X-P*›
by (*simp flip*: ‹$r \otimes\checkmark$ $s = \lfloor$*r-s*$\rfloor$› *add*: ‹$e = \checkmark$(*r-s*)› *super-ref-Sync*$_{ptick}$*-def*)
(*metis Refusals-iff* ‹([], *X-Q*) $\in \mathcal{F}$ Q› ‹$\checkmark(s) \in Q^0$› ‹$r \otimes\checkmark$ $s = \lfloor$*r-s*$\rfloor$› *accepts-initials-def-bis inj-tick-join that*(*2*))
with ‹$\checkmark(r) \in P^0$› ‹([], *X-P*) $\in \mathcal{F}$ P› **show** *False*
by (*fold Refusals-iff*) (*metis accepts-initialsD-bis* ‹$determ^0$ P›)
qed
qed
qed

corollary *accepts-initials-Sync*:
‹$determ^0$ $P \Longrightarrow determ^0$ $Q \Longrightarrow determ^0$ $(P$ $[\![S]\!]$ $Q)$›
by (*metis* $Sync_{Classic}$.*accepts-initials-*$Sync_{ptick}$ $Sync_{Classic}$*-is-Sync*)

lemma *accepts-initials-Renaming* : ‹$determ^0$ (*Renaming P f g*)› **if** ‹$determ^0$ P›
proof −
from ‹$determ^0$ P› **have** ‹$P \neq \bot$› **by** *auto*
with ‹$determ^0$ P› **show** ‹$determ^0$ (*Renaming P f g*)›
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* : ‹$determ^0$ $(P$ Θ $a \in A.$ Q $a)$ $\longleftrightarrow$ $determ^0$ P›
using *D-F* **by** (*auto simp add*: *accepts-initials-def initials-Throw Refusals-iff F-Throw*)

lemma *accepts-initials-Sliding*:
‹$determ^0$ $P \Longrightarrow determ^0$ $Q \Longrightarrow determ^0$ $(P \rhd Q) \longleftrightarrow$
$P = STOP \lor P^0 \subseteq Q^0 \land (range\ tick \cap P^0 \neq \{\} \lor range\ tick \cap Q^0 = \{\})$›
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*:
‹$P \sqsubseteq_{FD}$ $SKIPS$ $\{r.\ \checkmark(r) \in P^0\}$ $\Box$ $(\Box a \in \{a.\ ev\ a \in P^0\} \rightarrow P\ after\ a)$› (**is** ‹$P \sqsubseteq_{FD}$ *?rhs*›)
proof (*unfold failure-divergence-refine-def failure-refine-def divergence-refine-def*, *safe*)
show ‹$s \in \mathcal{D}$ *?rhs* $\Longrightarrow s \in \mathcal{D}$ P› **for** s
by (*auto simp add*: *D-Det D-SKIPS D-Mprefix D-After*)
next
show ‹$(s, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow (s, X) \in \mathcal{F}$ P› **for** s X
proof (*cases s*)
show ‹$s = [] \Longrightarrow (s, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow (s, X) \in \mathcal{F}$ P›
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-neutral initials-def is-processT6-TR-notin)
next
show ‹$s = e \# s' \Longrightarrow (s, X) \in \mathcal{F}\ ?rhs \Longrightarrow (s, X) \in \mathcal{F}\ P$› **for** $e\ s'$
by (*auto simp add*: *F-Det SKIPS-projs STOP-projs F-Mprefix F-After disjoint-iff image-iff split*: *if-split-asm*)
(*metis CollectD append-butlast-last-id initials-def last-ConsL list.distinct*(*1*) *tick-T-F*)
qed
qed

lemma *accepts-initials-imp-eq-Mprefix-After*:
‹$P = ($ *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* $a)$› (**is** ‹$P = ?rhs$›)
if ‹$determ^0\ P$›
proof −
from *not-accepts-initials-BOT* ‹$determ^0\ P$› **have** *non-BOT*: ‹$P \neq \bot$› **by** *blast*
note *initial-tick-iff-is-SKIP* = *accepts-initials-imp-initial-tick-iff-is-SKIP*[*OF* ‹$determ^0\ P$›]

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

show ‹$P = ?rhs$›
proof (*rule FD-antisym*)
show ‹$P \sqsubseteq_{FD}\ ?rhs$› **by** (*unfold* $*$) (*fact leFD-SKIPS-Det-Mprefix-After*)
next
show ‹$?rhs \sqsubseteq_{FD}\ P$›
proof (*unfold failure-divergence-refine-def failure-refine-def divergence-refine-def*, *safe*)
from *non-BOT* **show** ‹$s \in \mathcal{D}\ P \Longrightarrow s \in \mathcal{D}\ ?rhs$› **for** s
by (*cases s*; *simp add*: *D-Det D-SKIP D-STOP D-Mprefix BOT-iff-Nil-D image-iff D-After initial-tick-iff-is-SKIP*)
(*metis BOT-iff-tick-D initials-memI D-T D-imp-front-tickFree* $event_{ptick}$*.disc*(*2*) $event_{ptick}$*.exhaust front-tickFree-Cons-iff initials-SKIP non-BOT singletonD*)
next
have $*$: ‹$\exists r.\ \checkmark(r) \in P^0 \Longrightarrow \exists! r.\ \checkmark(r) \in P^0$›
by (*metis* $event_{ptick}$*.inject*(*2*) *initial-tick-iff-is-SKIP initials-SKIP singletonD*)
show ‹$(t, X) \in \mathcal{F}\ P \Longrightarrow (t, X) \in \mathcal{F}\ ?rhs$› **for** $t\ X$
proof (*cases t*)
from $*$ **show** ‹$t = [] \Longrightarrow (t, X) \in \mathcal{F}\ P \Longrightarrow (t, X) \in \mathcal{F}\ ?rhs$›
by (*simp add*: *add*: *F-Mprefix disjoint-iff image-iff initial-tick-iff-is-SKIP*)
(*metis* (*lifting*) ‹$determ^0\ P$› *Refusals-iff*[*of X P*]
accepts-initialsD-bis[*of P - X*] *the-equality*[*of* ‹$\lambda r.\ P = SKIP\ r$›])
next
from $*$ **show** ‹$t = e \# t' \Longrightarrow (t, X) \in \mathcal{F}\ P \Longrightarrow (t, X) \in \mathcal{F}\ ?rhs$› **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*:
‹$(\exists A\ Q.\ P = \Box a \in A \rightarrow Q\ a) \longleftrightarrow$ *range tick* $\cap\ P^0 = \{\} \wedge$ *accepts-initials P*›
for *P* :: ‹$('a,\ 'r)$ $process_{ptick}$›
proof (*intro iffI exI*)
show ‹$\exists A\ Q.\ P = Mprefix\ A\ Q \Longrightarrow$ *range tick* $\cap\ P^0 = \{\} \wedge$ *accepts-initials P*›
by (*auto simp add*: *initials-Mprefix image-iff disjoint-iff*)
next
from *accepts-initials-imp-eq-Mprefix-After*
show ‹*range tick* $\cap\ P^0 = \{\} \wedge$ *accepts-initials P* $\Longrightarrow$
$P = \Box a \in \{e.\ ev\ e \in P^0\} \rightarrow P\ after\ a$›
by (*meson disjoint-iff rangeI*)
qed

lemma *tick-not-initial-imp-STOP-Ndet-Mndetprefix-After-FD*:
‹*range tick* $\cap\ P^0 = \{\} \Longrightarrow STOP \sqcap (\Box a \in \{e.\ ev\ e \in P^0\} \rightarrow P\ after\ a) \sqsubseteq_{FD} P$›
by (*cases* ‹$P = \bot$›, *solves* ‹*simp*›,
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 ‹*lifelock-free* $P \longleftrightarrow \mathcal{D}\ P = \{\} \wedge (\forall t \in \mathcal{T}\ P.\ tF\ t)$›
using *lifelock-free-is-non-terminating non-terminating-is-right nonterminating-implies-div-free*
by *blast*

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

| $a\ t'$ **where** ‹$t = ev\ a\ \#\ t'$› ‹$a \in \{a.\ ev\ a \in P^0\}$›
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 ‹$(t,\ X) \in \mathcal{F}\ (STOP \sqcap ?lhs1 \sqcap ?lhs2)$›
proof *cases*
show ‹$t = [] \Longrightarrow (t,\ X) \in \mathcal{F}\ (STOP \sqcap ?lhs1 \sqcap ?lhs2)$› **by** (*simp add*: *F-Ndet F-STOP*)
next
fix r **assume** ‹$t = [\checkmark(r)]$›
with ‹$(t,\ X) \in \mathcal{F}\ P$› **have** ‹$(t,\ X) \in \mathcal{F}\ (?lhs1)$›
by (*auto simp add*: *F-SKIPS intro*!: *initials-memI′ F-T*)
thus ‹$(t,\ X) \in \mathcal{F}\ (STOP \sqcap ?lhs1 \sqcap ?lhs2)$› **by** (*simp add*: *F-Ndet*)
next
fix $a\ t'$ **assume** ‹$t = ev\ a\ \#\ t'$›
with ‹$(t,\ X) \in \mathcal{F}\ P$› **have** ‹$(t,\ X) \in \mathcal{F}\ ?lhs2$›
by (*auto simp add*: *F-Mprefix F-After intro*!: *initials-memI F-T*)
thus ‹$(t,\ X) \in \mathcal{F}\ (STOP \sqcap ?lhs1 \sqcap ?lhs2)$› **by** (*simp add*: *F-Ndet*)
qed
qed

lemma *non-BOT-imp-Mprefix-After-leD* :
‹$\Box a \in \{e.\ ev\ e \in P^0\} \rightarrow P\ after\ a \sqsubseteq_D P$› (**is** ‹-$?lhs \sqsubseteq_D P$›) **if** ‹$P \neq \bot$›
proof (*unfold divergence-refine-def*, *rule subsetI*)
fix t **assume** ‹$t \in \mathcal{D}\ P$›
with ‹$P \neq \bot$› **obtain** $a\ t'$ **where** ‹$t = ev\ a\ \#\ t'$›
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 ‹$t \in \mathcal{D}\ P$› **show** ‹$t \in \mathcal{D}\ ?lhs$›
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* :
‹$P \neq \bot \Longrightarrow STOP \sqcap SKIPS\ \{r.\ \checkmark(r) \in P^0\} \sqcap (\Box a \in \{e.\ ev\ e \in P^0\} \rightarrow P\ after\ a) \sqsubseteq_{FD} P$›
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*:
‹$P = (if\ UNIV \notin \mathcal{R}\ P\ then\ a \rightarrow P\ after\ a\ else\ STOP \sqcap (a \rightarrow P\ after\ a))$›
if *initials-is* : ‹$initials\ P = \{ev\ a\}$›
proof (*split if-split*, *intro conjI impI*)
assume *not-all-refusals* : ‹$UNIV \notin \mathcal{R}\ P$›
have \$: ‹$e \neq ev\ a \Longrightarrow [e] \notin \mathcal{T}\ P$› **for** e **using** *initials-is* **unfolding** *initials-def* **by** *auto*
{ assume ‹$\{ev\ a\} \in \mathcal{R}\ P$›
from *is-processT5*[*rule-format*, *of* ‹[]› ‹$\{ev\ a\}$› P ‹$UNIV - \{ev\ a\}$›, *folded*

Refusals-iff,
simplified this T-F-spec, *simplified*, *rule-format*, *OF* \$, *simplified*]
have ‹$UNIV \in \mathcal{R}\ P$› **.**
with *not-all-refusals* **have** *False* **by** *simp*
} **hence** *not-in-refusal*: ‹$\{ev\ a\} \notin \mathcal{R}\ P$› **by** *blast*
show ‹$P = a \rightarrow P$ *after* a›
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* : ‹$\neg\ UNIV \notin \mathcal{R}\ P$›
from *tick-not-initial-imp-STOP-Ndet-Mndetprefix-After-FD*[*of P*]
have $*$: ‹$STOP \sqcap (a \rightarrow P$ *after* $a) \sqsubseteq_{FD} P$›
by (*simp add*: *that*(*1*) *Mprefix-singl image-iff*)
from *leFD-SKIPS-Det-Mprefix-After*[*of P*] *all-refusals*
have $**$: ‹$P \sqsubseteq_{FD} STOP \sqcap (a \rightarrow P$ *after* $a)$›
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** ‹$P = STOP \sqcap (a \rightarrow P$ *after* $a)$› **by** (*fact FD-antisym*)
qed

lemma ‹$\{ev\ e\} \notin \mathcal{R}\ P \Longrightarrow ev\ e \in P^0$›
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* :: ‹$('a, 'r)\ process_{ptick} \Rightarrow bool$› (‹*determ*›)
where ‹$determ\ P \equiv \forall\ s\ e.\ s\ @\ [e] \in \mathcal{T}\ P \longrightarrow (s, \{e\}) \notin \mathcal{F}\ (P)$›

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

lemma *deterministic-STOP* [*simp*] : ‹*determ STOP*›
and *deterministic-SKIP* [*simp*] : ‹*determ* (*SKIP r*)›
by (*simp-all add*: *deterministic-def T-STOP SKIP-projs*)

lemma *deterministic-div-free* : ‹$determ\ P \Longrightarrow \mathcal{D}\ P = \{\}$›
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*] : ‹$\neg$ *determ* $\bot$›
  using *BOT-iff-Nil-D deterministic-div-free* **by** *blast*

2.2.2 Monotonicity

lemma *mono-deterministic-F*: ‹$P \sqsubseteq_F Q \Longrightarrow$ *determ* $P \Longrightarrow$ *determ* Q›
  by (*meson T-F-spec deterministic-def failure-refine-def subset-iff*)

lemma *mono-deterministic-FD*: ‹$P \sqsubseteq_{FD} Q \Longrightarrow$ *determ* $P \Longrightarrow$ *determ* Q›
  using *leFD-imp-leF mono-deterministic-F* **by** *blast*

lemma *mono-deterministic*: ‹$P \sqsubseteq Q \Longrightarrow$ *determ* $P \Longrightarrow$ *determ* Q›
  using *le-approx-imp-le-ref mono-deterministic-FD* **by** *auto*

lemma *restriction-adm-deterministic* [*restriction-adm-process*$_{ptick}$*-simpset*, *simp*]
:
  ‹$adm_{\downarrow}$ (λx. *determ* (f x))› **if** ‹$cont_{\downarrow}$ f›
for f :: ‹$'b$ :: *restriction* $\Rightarrow$ ($'a$, $'r$) *process*$_{ptick}$›
proof (*rule restriction-adm-subst*)
  from ‹$cont_{\downarrow}$ f› **show** ‹$cont_{\downarrow}$ f› **.**
next
  show ‹$adm_{\downarrow}$ (*determ* :: ($'a$, $'r$) *process*$_{ptick}$ $\Rightarrow$ *bool*)›
  proof (*rule restriction-admI*)
    fix σ **and** Σ :: ‹($'a$, $'r$) *process*$_{ptick}$›
    assume ‹σ $-\downarrow\rightarrow$ Σ› ‹*determ* (σ n)› **for** n
    show ‹*determ* Σ›
    proof (*rule deterministicI*)
      fix t e **assume** ‹t @ [e] $\in$ $\mathcal{T}$ Σ›
      from ‹σ $-\downarrow\rightarrow$ Σ› **obtain** *n0*
        where $*$: ‹$\Sigma \downarrow$ *Suc* (*length* (t @ [e])) = σ *n0* $\downarrow$ *Suc* (*length* (t @ [e]))›
        by (*blast dest*: *restriction-tendstoD*)
      with ‹t @ [e] $\in$ $\mathcal{T}$ Σ› **have** ‹t @ [e] $\in$ $\mathcal{T}$ (σ *n0*)›
        by (*metis T-restriction-process*$_{ptick}$*-Suc-length-iff-T*)
      with ‹*deterministic* (σ *n0*)› **have** ‹(t, {e}) $\notin$ $\mathcal{F}$ (σ *n0*)›
        by (*fact deterministicD*)
      with $*$ **show** ‹(t, {e}) $\notin$ $\mathcal{F}$ Σ›
        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*$_T$ :: ‹($'a$, $'r$) *trace*$_{ptick}$ *set* $\Rightarrow$ *bool*›
  where ‹*is-process*$_T$ T $\equiv$

$[] \in T \wedge (\forall t \in T.\ ftF\ t) \wedge (\forall t\ u.\ t\ @\ u \in T \longrightarrow t \in T) \wedge$
$(\forall t\ r\ e.\ t\ @\ [\checkmark(r)] \in T \longrightarrow e \neq \checkmark(r) \longrightarrow t\ @\ [e] \notin T)$›

typedef $('a,\ 'r)\ process_T$ = ‹$\{T :: ('a,\ 'r)\ trace_{ptick}\ set\ .\ is\text{-}process_T\ T\}$›
proof (*rule exI*)
  show ‹$\{[]\} \in \{T.\ is\text{-}process_T\ T\}$› **unfolding** $is\text{-}process_T\text{-}def$ **by** *simp*
qed

setup-lifting $type\text{-}definition\text{-}process_T$

lift-definition $Traces_T$ ::
  ‹$('a,\ 'r)\ process_T \Rightarrow ('a,\ 'r)\ trace_{ptick}\ set$› (‹$\mathcal{T}_T$›)
  is ‹$\lambda P.\ Rep\text{-}process_T\ P$› **.**

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

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

lemmas $is\text{-}process_T\text{-}def\text{-}bis = is\text{-}process_T\text{-}def$[*of* ‹$Rep\text{-}process_T$ -›, *folded* $Traces_T.rep\text{-}eq$]

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

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

lemmas $process_{ptick}\text{-}of\text{-}process_T\text{-}projs = F\text{-}process_{ptick}\text{-}of\text{-}process_T$

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

Now the big results

lemma *bij-betw-det* :
‹*bij-betw process$_{ptick}$-of-process$_T$ UNIV* {*P* :: ('*a*, '*r*) *process$_{ptick}$*. *determ P*}›
(**is** ‹*bij-betw process$_{ptick}$-of-process$_T$?S1 ?S2*›)
proof (*intro bij-betw-imageI*)
show ‹*inj-on process$_{ptick}$-of-process$_T$?S1*›
by (*rule inj-onI*) (*auto simp add*: *Process-eq-spec process$_{ptick}$-of-process$_T$-projs Process$_T$-eq-spec*)
next
show ‹*process$_{ptick}$-of-process$_T$* ‘ *?S1* = *?S2*›
proof (*intro subset-antisym subsetI*; *clarify*)
show ‹*determ* (*process$_{ptick}$-of-process$_T$ P*)› **for** *P* :: ‹('*a*, '*r*) *process$_T$*›
by (*rule deterministicI*) (*simp add*: *process$_{ptick}$-of-process$_T$-projs*)
next
fix *P* :: ‹('*a*, '*r*) *process$_{ptick}$*›
assume *det* : ‹*deterministic P*›
hence ∗ : ‹*Rep-process$_T$* (*Abs-process$_T$* (𝒯 *P*)) = 𝒯 *P*›
by (*intro Abs-process$_T$-inverse*)
(*simp add*: *deterministic-def is-process$_T$-def is-processT2-TR*,
metis T-F-spec is-processT3 is-processT6-notin singletonD)
show ‹*P* ∈ *process$_{ptick}$-of-process$_T$* ‘ *?S1*›
proof (*subst image-iff*, *rule bexI*)
show ‹*P* = *process$_{ptick}$-of-process$_T$* (*Abs-process$_T$* (𝒯 *P*))›
by (*auto intro*!: *Process-eq-optimizedI simp add*: *process$_{ptick}$-of-process$_T$-projs*
Traces$_T$-def ∗ *det deterministic-div-free subset-iff F-T*)
(*meson det deterministic-def empty-subsetI insert-subset is-processT*,
use is-processT5-S7 **in** *blast*)
next
show ‹*Abs-process$_T$* (𝒯 *P*) ∈ *?S1*› **by** (*simp add*: *Traces$_T$-def* ∗)
qed
qed
qed

lemma *SKIPS-is-GlobalDet-SKIP* : ‹*SKIPS R* = □*r* ∈ *R*. *SKIP r*›
by (*auto simp add*: *Process-eq-spec SKIPS-projs GlobalDet-projs SKIP-projs*)

lemma *SKIP-Ndet-SKIP-is-SKIP-Det-SKIP* : ‹*SKIP r* ⊓ *SKIP s* = *SKIP r* □ *SKIP s*›
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* :: ‹$('a, 'r)$ $trace_{ptick} \Rightarrow 'r$›
assumes ‹$\bigwedge t.\ \exists\, r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P \Longrightarrow$ *termination-choice* $t \in \{r.\ t\ @\ [\checkmark(r)] \in \mathcal{T}\ P\}$›
defines ‹$T \equiv \{t \in \mathcal{T}\ P.\ \forall\, t' < t.\ (\exists\, r.\ t'\ @\ [\checkmark(r)] \in \mathcal{T}\ P) \longrightarrow t = t'\ @\ [\checkmark(\textit{termination-choice}\ t')]\}$›
shows ‹$P \sqsubseteq_{FD}$ $process_{ptick}$-*of-process*$_T$ (*Abs-process*$_T$ T)›
proof (*unfold failure-divergence-refine-def failure-refine-def divergence-refine-def*, *intro conjI*)
show ‹$\mathcal{D}$ ($process_{ptick}$-*of-process*$_T$ (*Abs-process*$_T$ T)) $\subseteq \mathcal{D}\ P$› **by** (*simp add*: *D-process*$_{ptick}$-*of-process*$_T$)
next
have $*$: ‹$T \in \{T.$ *is-process*$_T$ $T\}$›
by (*auto simp add*: *T-def is-process*$_T$*-def T-imp-front-tickFree intro*: *is-processT3-TR-append*)
(*metis prefix-prefix append-eq-first-pref-spec less-list-def nless-le self-append-conv*, *metis less-self*)

show ‹$\mathcal{F}$ ($process_{ptick}$-*of-process*$_T$ (*Abs-process*$_T$ T)) $\subseteq \mathcal{F}\ P$›
proof *safe*
fix $s\ X$ **assume** ‹$(s, X) \in \mathcal{F}$ ($process_{ptick}$-*of-process*$_T$ (*Abs-process*$_T$ T))›
hence ‹$s \in \mathcal{T}\ P$›
by (*simp add*: *F-process*$_{ptick}$*-of-process*$_T$ *Traces*$_T$*-def Abs-process*$_T$*-inverse*[*OF* $*$]) (*simp add*: *T-def*)
show ‹$(s, X) \in \mathcal{F}\ P$›
proof (*cases* ‹$\exists\, r.\ s\ @\ [\checkmark(r)] \in \mathcal{T}\ P$›)
assume ‹$\exists\, r.\ s\ @\ [\checkmark(r)] \in \mathcal{T}\ P$›
hence ‹$s\ @\ [\checkmark(\textit{termination-choice}\ s)] \in \mathcal{T}\ P$› **by** (*metis assms*(*1*) *mem-Collect-eq*)
with ‹$(s, X) \in \mathcal{F}$ ($process_{ptick}$-*of-process*$_T$ (*Abs-process*$_T$ T))› **have** ‹$\checkmark(\textit{termination-choice}\ s) \notin X$›
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 ‹$s\ @\ [\checkmark(\textit{termination-choice}\ s)] \in \mathcal{T}\ P$› **show** ‹$(s, X) \in \mathcal{F}\ P$›
by (*metis is-processT6-TR-notin*)
next
assume ‹$\nexists\, r.\ s\ @\ [\checkmark(r)] \in \mathcal{T}\ P$›
with ‹$(s, X) \in \mathcal{F}$ ($process_{ptick}$-*of-process*$_T$ (*Abs-process*$_T$ T))› **have** ‹$X \subseteq -\{e.\ s\ @\ [e] \in \mathcal{T}\ P\}$›
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-processT5-S7*[*OF* ‹$s \in \mathcal{T}\ P$›] **show** ‹$(s, X) \in \mathcal{F}\ P$› **by** *blast*
qed
qed
qed

theorem *deterministic-iff-maximal-for-leFD*:
‹*determ* $P \longleftrightarrow (\forall Q.\ P \sqsubseteq_{FD} Q \longrightarrow P = Q)$› **for** P :: ‹$('a,\ 'r)$ $process_{ptick}$›
— see TPC, chapter 9)
proof (*intro iffI allI impI*)
fix Q **assume** ‹*determ* P› **and** ‹$P \sqsubseteq_{FD} Q$›
from ‹$P \sqsubseteq_{FD} Q$› **have** *no-div* : ‹$\mathcal{D}\ P = \{\}$› ‹$\mathcal{D}\ Q = \{\}$› **and** *F-subset* : ‹$\mathcal{F}\ Q \subseteq \mathcal{F}\ P$›
by (*simp-all add*: ‹*determ* P› *deterministic-div-free refine-defs*)

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

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

have ‹$e \in X \Longrightarrow s$ @ $[e] \notin \mathcal{T}\ P$› **for** e
by (*metis* $*(1)$ *DiffD1* ‹*determ* P› *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 ‹$\mathcal{F}\ Q \subseteq \mathcal{F}\ P$› **by** (*fact F-subset*)
qed
show ‹$P = Q$› **by** (*simp add*: *Process-eq-spec failure-refine-def divergence-refine-def*

no-div same-F)

next
define *termination-choice* **where** ‹*termination-choice s* ≡ (*SOME r. s* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*)› **for** *s*
have \$: ‹∃ *r. s* @ [✓(*r*)] ∈ $\mathcal{T}$ *P* ⟹ *termination-choice s* ∈ {*r. s* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*}› **for** *s*
by (*simp add*: *termination-choice-def*) (*meson someI*)

define *T* **where** ‹*T* ≡ {*s* ∈ $\mathcal{T}$ *P*. ∀ *s'* < *s*. (∃ *r. s'* @ [✓(*r*)] ∈ $\mathcal{T}$ *P*) ⟶ *s* = *s'* @ [✓(*termination-choice s'*)]}›
have ∗ : ‹*T* ∈ {*T. is-process*$_T$ *T*}›
by (*auto simp add*: *T-def is-process*$_T$*-def T-imp-front-tickFree intro*: *is-processT3-TR-append*)
(*metis prefix-prefix append-eq-first-pref-spec less-list-def nless-le self-append-conv*, *metis less-self*)
assume *maximal* : ‹∀ *Q. P* $\sqsubseteq_{FD}$ *Q* ⟶ *P* = *Q*›
with \$ *P-FD-some-det T-def* **have** ‹*P* = *process*$_{ptick}$*-of-process*$_T$ (*Abs-process*$_T$ *T*)› **by** *blast*
moreover have ‹*Abs-process*$_T$ *T* ∈ {*P*. ∀ *s r e. s* @ [✓(*r*)] ∈ $\mathcal{T}_T$ *P* ⟶ *e* ≠ ✓(*r*) ⟶ *s* @ [*e*] ∉ $\mathcal{T}_T$ *P*}›
by (*simp add*: *Traces*$_T$*-def Abs-process*$_T$*-inverse*[*OF* ∗])
(*metis* ∗ *is-process*$_T$*-def mem-Collect-eq*)
ultimately show ‹*determ P*› **using** *bij-betwE*[*OF bij-betw-det*] **by** *blast*
qed

lemma ‹*determ P* ⟹ *X* ∈ $\mathcal{R}$ *P* ⟹ *X* ⊆ − P^0›
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* :
‹*determ P* ⟹ *P* = *SKIPS* {*r.* ✓(*r*) ∈ P^0} □ (□*a* ∈ {*a. ev a* ∈ P^0} → *P after a*)›
by (*simp add*: *deterministic-iff-maximal-for-leFD leFD-SKIPS-Det-Mprefix-After*)

lemma *deterministic-imp-initial-tick-iff-eq-SKIP* [*simp*] :
‹*determ P* ⟹ ✓(*r*) ∈ P^0 ⟷ *P* = *SKIP r*›
by (*meson deterministic-iff-maximal-for-leFD dual-order.refl initial-tick-iff-FD-SKIP*)

lemma *deterministic-imp-constraints-on-initials* :
‹*determ P* ⟹ P^0 = {} ∨ {*a. ev a* ∈ P^0} = {} ∧ (∃ *r.* P^0 = {✓(*r*)}) ∨
{*a. ev a* ∈ P^0} ≠ {} ∧ {*r.* ✓(*r*) ∈ P^0} = {}›
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* :
‹*determ P* $\Longrightarrow$ *P* = (*if* $\exists$ *r*. ✓(*r*) $\in$ P^0 *then SKIP* (*THE r*. P^0 = {✓(*r*)})
else $\Box a \in \{a.\ ev\ a \in P^0\} \rightarrow P\ after\ a$)›
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*$_{trace}$:
‹*determ P* $\longleftrightarrow$ ($\forall$ *t* $\in$ $\mathcal{T}$ *P*. *tF t* $\longrightarrow$ *determ*^{0} (*P after*$_{\mathcal{T}}$ *t*))›
proof (*intro iffI ballI impI*)
show ‹*determ P* $\Longrightarrow$ *t* $\in$ $\mathcal{T}$ *P* $\Longrightarrow$ *tF t* $\Longrightarrow$ *determ*^{0} (*P after*$_{\mathcal{T}}$ *t*)› **for** *t*
by (*rule accepts-initialsI*)
(*simp add*: *initials-def Refusals-iff T-After*$_{trace}$*-eq F-After*$_{trace}$*-eq deterministic-def*)
next
show ‹*determ P*› **if** ‹$\forall$ *t*$\in$$\mathcal{T}$ *P*. *tF t* $\longrightarrow$ *determ*^{0} (*P after*$_{\mathcal{T}}$ *t*)›
proof (*rule deterministicI*)
fix *t e* **assume** ‹*t* @ [*e*] $\in$ $\mathcal{T}$ *P*›
have ‹*t* $\in$ $\mathcal{T}$ *P*› **and** ‹*tF t*›
by (*meson prefixI* ‹*t* @ [*e*] $\in$ $\mathcal{T}$ *P*› *is-processT3-TR*)
(*use* ‹*t* @ [*e*] $\in$ $\mathcal{T}$ *P*› *append-T-imp-tickFree* **in** *blast*)
with ‹*t* $\in$ $\mathcal{T}$ *P*› *that*[*rule-format*, *OF this*] **show** ‹*t* @ [*e*] $\in$ $\mathcal{T}$ *P* $\Longrightarrow$ (*t*, {*e*}) $\notin$ $\mathcal{F}$ *P*›
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* :
‹*determ* ($\Box a \in A \rightarrow P\ a$) $\longleftrightarrow$ ($\forall$ *a* $\in$ *A*. *determ* (*P a*))›
by (*auto simp add*: *deterministic-def Mprefix-projs*) (*metis append-Cons*)

corollary *deterministic-write0-iff* : ‹*determ* (*a* $\rightarrow$ *P*) $\longleftrightarrow$ *determ P*›
unfolding *write0-def* **by** (*simp add*: *deterministic-Mprefix-iff*)

corollary *deterministic-write-iff* : ‹*determ* (*c*!*a* $\rightarrow$ *P*) $\longleftrightarrow$ *determ P*›
unfolding *write-def* **by** (*simp add*: *deterministic-Mprefix-iff*)

corollary *deterministic-inj-on-read-iff* :
‹*inj-on c A* $\Longrightarrow$ *determ* (*c*?*a* $\in$ *A* $\rightarrow$ *P a*) $\longleftrightarrow$ ($\forall$ *a* $\in$ *A*. *determ* (*P a*))›
unfolding *read-def* **by** (*simp add*: *deterministic-Mprefix-iff*)

lemma *deterministic-inj-Renaming* :
‹*determ* (*Renaming P f g*)› **if** ‹*inj f*› ‹*inj g*› ‹*determ P*›
proof (*rule deterministicI*)

have \$: ‹*inj* (*map-event*$_{ptick}$ *f g*)› **by** (*simp add*: *event*$_{ptick}$*.inj-map that*(*1*,*2*))
fix *t e*
assume ‹*t* @ [*e*] ∈ $\mathcal{T}$ (*Renaming P f g*)›
then obtain *t1* **where** ∗ : ‹*t1* ∈ $\mathcal{T}$ *P*› ‹*t* @ [*e*] = *map* (*map-event*$_{ptick}$ *f g*) *t1*›
by (*simp add*: *T-Renaming deterministic-div-free*[*OF* ‹*determ P*›]) *blast*
have ‹(*s1*, *map-event*$_{ptick}$ *f g* −‘ {*e*}) ∈ $\mathcal{F}$ *P* ⟹ *t* ≠ *map* (*map-event*$_{ptick}$ *f g*) *s1*› **for** *s1*
proof (*rule ccontr*, *clarify*)
assume *assms* : ‹(*s1*, *map-event*$_{ptick}$ *f g* −‘ {*e*}) ∈ $\mathcal{F}$ *P*› ‹*t* = *map* (*map-event*$_{ptick}$ *f g*) *s1*›
from *assms*(*2*) ∗(*2*) **have** ‹*t1* = *s1* @ [*inv* (*map-event*$_{ptick}$ *f g*) *e*]›
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 ‹(*s1*, {*inv* (*map-event*$_{ptick}$ *f g*) *e*}) ∉ $\mathcal{F}$ *P*› **.**
also have ‹{*inv* (*map-event*$_{ptick}$ *f g*) *e*} = *map-event*$_{ptick}$ *f g* −‘ {*e*}›
using *inj-vimage-singleton*[*OF* \$, *of e*] ∗(*2*)
‹*t1* = *s1* @ [*inv* (*map-event*$_{ptick}$ *f g*) *e*]› **by** (*auto simp add*: \$)
finally have ‹(*s1*, *map-event*$_{ptick}$ *f g* −‘ {*e*}) ∉ $\mathcal{F}$ *P*› **.**
with *assms*(*1*) **show** *False* **by** *simp*
qed
thus ‹(*t*, {*e*}) ∉ $\mathcal{F}$ (*Renaming P f g*)›
by (*auto simp add*: *F-Renaming deterministic-div-free*[*OF* ‹*determ P*›])
qed

lemma *deterministic-bij-Renaming-iff* :
‹*determ* (*Renaming P f g*) ⟷ *determ P*› **if** ‹*bij f*› **and** ‹*bij g*›
proof (*rule iffI*)
show ‹*determ* (*Renaming P f g*) ⟹ *determ P*›
by (*metis Renaming-inv bij-betw-def deterministic-iff-maximal-for-leFD*
mono-Renaming-FD ‹*bij f*› ‹*bij g*›)
next
show ‹*determ P* ⟹ *determ* (*Renaming P f g*)›
by (*simp add*: *bij-is-inj deterministic-inj-Renaming* ‹*bij f*› ‹*bij g*›)
qed

lemma *deterministic-Throw* : ‹*determ* (*P* Θ *a* ∈ *A*. *Q a*)›
if ‹*determ P*› ‹⋀*a*. *a* ∈ *A* ⟹ *a* ∈ α(*P*) ⟹ *determ* (*Q a*)›
proof (*subst Throw-is-restrictable-on-events-of*)
show ‹*determ* (*Throw P* (*A* ∩ α(*P*)) *Q*)›
proof (*rule deterministicI*)
fix *t e* **assume** ‹*t* @ [*e*] ∈ $\mathcal{T}$ (*P* Θ *a* ∈ (*A* ∩ α(*P*)). *Q a*)›
moreover from ‹*determ P*› **have** ‹$\mathcal{D}$ *P* = {}›
by (*simp add*: *deterministic-div-free*)
ultimately consider
(*traceL*) ‹*t* @ [*e*] ∈ $\mathcal{T}$ *P*› ‹*e* ∉ *ev* ‘ (*A* ∩ α(*P*))› ‹*set t* ∩ *ev* ‘ (*A* ∩ α(*P*)) = {}›

```
      | (traceR) t1 a t2 where ‹t @ [e] = t1 @ ev a # t2› ‹t1 @ [ev a] ∈ 𝒯 P›
        ‹set t1 ∩ ev ‘ (A ∩ α(P)) = {}› ‹a ∈ A› ‹a ∈ α(P)› ‹t2 ∈ 𝒯 (Q a)›
      unfolding T-Throw by auto
    thus ‹(t, {e}) ∉ ℱ (P Θ a ∈ (A ∩ α(P)). Q a)›
    proof cases
      case traceL
      from traceL(1) ‹determ P› have ‹(t, {e}) ∉ ℱ P›
        by (simp add: deterministicD)
      with traceL(3) show ‹(t, {e}) ∉ ℱ (P Θ a ∈ (A ∩ α(P)). Q a)›
        by (auto simp add: F-Throw ‹𝒟 P = {}›)
    next
      case traceR
      from traceR(1) consider ‹t2 = []› ‹t = t1› ‹e = ev a›
        | t2' where ‹t2 = t2' @ [e]› ‹t = t1 @ ev a # t2'›
        by (cases t2 rule: rev-cases) simp-all
      thus ‹(t, {e}) ∉ ℱ (P Θ a ∈ (A ∩ α(P)). Q a)›
      proof cases
        assume ‹t2 = []› ‹t = t1› ‹e = ev a›
        from ‹determ P› traceR(2) have ‹(t1, {ev a}) ∉ ℱ P›
          by (simp add: deterministicD)
        with traceR(3) show ‹(t, {e}) ∉ ℱ (P Θ a ∈ (A ∩ α(P)). Q a)›
          by (auto simp add: ‹t = t1› ‹e = ev a› ‹𝒟 P = {}› F-Throw)
      next
        fix t2' assume ‹t2 = t2' @ [e]› ‹t = t1 @ ev a # t2'›
        from traceR(4, 5) have ‹determ (Q a)›
          by (rule ‹⋀a. a ∈ A ⟹ a ∈ α(P) ⟹ determ (Q a)›)
        with ‹t2 = t2' @ [e]› have ‹(t2', {e}) ∉ ℱ (Q a)›
          using traceR(6) by (simp add: deterministicD)
        with traceR(3−5) show ‹(t, {e}) ∉ ℱ (P Θ a ∈ (A ∩ α(P)). Q a)›
       by (simp add: ‹t = t1 @ ev a # t2'› ‹𝒟 P = {}› F-Throw Throw-T-third-clause-breaker)
      qed
    qed
  qed
qed


lemma T-snoc-tick-imp-no-continuation-if-deterministic :
  ‹u = [] ∧ e = ✓(r)› if ‹determ P› ‹t @ u @ [e] ∈ 𝒯 P› ‹t @ [✓(r)] ∈ 𝒯 P›
proof −
  have ∗ : ‹t @ [e] ∈ 𝒯 P ⟹ e = ✓(r)› for e
    by (metis deterministicD is-processT6-TR-notin singletonD that(1, 3))
  show ‹u = [] ∧ e = ✓(r)›
  proof (cases u)
    from ∗ that(2) show ‹u = [] ⟹ u = [] ∧ e = ✓(r)› by auto
  next
    fix e' u'
    assume ‹u = e' # u'›
    with that(2) have ‹t @ [e'] ∈ 𝒯 P›
```

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

lemma *T-snoc-ev-imp-no-tick-continuation-if-deterministic* :
‹$u \neq [] \wedge$ *is-ev* (*hd u*) $\vee$ *is-ev e*› **if** ‹*determ P*› ‹t @ u @ $[e] \in \mathcal{T}\ P$› ‹t @ $[ev\ a] \in \mathcal{T}\ P$›
proof −
have ‹t @ $[e] \in \mathcal{T}\ P \Longrightarrow$ *is-ev e*› **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 ‹$u \neq [] \wedge$ *is-ev* (*hd u*) $\vee$ *is-ev e*›
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} : ‹*determ* (P ;$_\checkmark$ Q)›
if ‹*determ P*› ‹$\bigwedge r.\ r \in \checkmark\mathbf{s}(P) \Longrightarrow$ *determ* ($Q\ r$)›
proof (*rule deterministicI*)
fix *t e* **assume** ‹t @ $[e] \in \mathcal{T}$ (P ;$_\checkmark$ Q)›
moreover from ‹*determ P*› **have** ‹$\mathcal{D}\ P = \{\}$›
by (*simp add*: *deterministic-div-free*)
ultimately consider *a u* **where** ‹$e = ev\ a$› ‹$t = map\ (ev \circ of\text{-}ev)\ u$› ‹$u$ @ $[ev\ a] \in \mathcal{T}\ P$› ‹*tF u*›
| *u v r* **where** ‹t @ $[e] = map\ (ev \circ of\text{-}ev)\ u$ @ v› ‹u @ $[\checkmark(r)] \in \mathcal{T}\ P$› ‹*tF u*› ‹$v \in \mathcal{T}\ (Q\ r)$› ‹$v \neq []$›
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 ‹$(t, \{e\}) \notin \mathcal{F}\ (P$;$_\checkmark$ $Q)$›
proof *cases*
show ‹$e = ev\ a \Longrightarrow t = map\ (ev \circ of\text{-}ev)\ u \Longrightarrow u$ @ $[ev\ a] \in \mathcal{T}\ P \Longrightarrow tF\ u \Longrightarrow (t, \{e\}) \notin \mathcal{F}\ (P$;$_\checkmark$ $Q)$› **for** *a u*
by (*auto simp add*: *tickFree-map-ev-of-ev-inj* ‹$\mathcal{D}\ P = \{\}$› Seq_{ptick}*-projs ref-*Seq_{ptick}*-def map-eq-append-conv*)
(*meson deterministicD empty-subsetI insertI1 insert-subset is-processT4* ‹*determ P*›,
meson T-snoc-tick-imp-no-continuation-if-deterministic $event_{ptick}$.*distinct*(*1*) ‹*determ P*›)
next
fix *u v r* **assume** ‹t @ $[e] = map\ (ev \circ of\text{-}ev)\ u$ @ v› ‹u @ $[\checkmark(r)] \in \mathcal{T}\ P$› ‹*tF*

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

corollary *deterministic-Seq* : ‹*determ* $P \Longrightarrow$ *determ* $Q \Longrightarrow$ *determ* $(P\ ;\ Q)$›
by (*metis Seq*$_{ptick}$*-const deterministic-Seq*$_{ptick}$)

lemma (**in** *After*) *initial-imp-deterministic-After*:
‹$ev\ e \in P^0 \Longrightarrow$ *determ* $P \Longrightarrow$ *determ* $(P$ *after* $e)$›
unfolding *deterministic-def* **by** (*simp add*: *After-projs*)

lemma (**in** *AfterExt*) *initial-imp-deterministic-After*$_{tick}$:
‹$e \in P^0 \Longrightarrow$ (*case* e *of* $\checkmark(r) \Rightarrow$ *determ* $(\Omega\ P\ r)) \Longrightarrow$
determ $P \Longrightarrow$ *determ* $(P$ *after*$_{\checkmark}$ $e)$›
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* : ‹*determ P* $\Longrightarrow$ *determ*^{0} *P*›
by (*simp add*: *Refusals-iff accepts-initials-def deterministic-def initials-def*)

corollary *deterministic-SKIPS-iff* : ‹*determ* (*SKIPS R*) $\longleftrightarrow$ $R = \{\} \vee (\exists\, r.\ R = \{r\})$›
by (*metis SKIPS-empty SKIPS-singl-is-SKIP accepts-initials-SKIPS-iff deterministic-SKIP deterministic-STOP deterministic-imp-accepts-initials*)

lemma *deterministic-Det*:
‹*determ P* $\Longrightarrow$ *determ Q* $\Longrightarrow$
range tick $\cap$ P^0 $\cap$ Q^0 $\neq$ $\{\}$ $\vee$ P^0 $\cap$ Q^0 $=$ $\{\}$ $\wedge$ *range tick* $\cap$ (P^0 $\cup$ Q^0) $=$ $\{\}$ $\Longrightarrow$ *determ* (P $\Box$ Q)›
proof (*elim disjE conjE*)
show ‹*determ P* $\Longrightarrow$ *determ Q* $\Longrightarrow$ *range tick* $\cap$ P^0 $\cap$ Q^0 $\neq$ $\{\}$ $\Longrightarrow$ *determ* (P $\Box$ Q)›
by (*auto simp add*: *deterministic-def Det-projs SKIP-projs*)
next
show ‹$[\![$*determ P*; *determ Q*; P^0 $\cap$ Q^0 $=$ $\{\}$; *range tick* $\cap$ (P^0 $\cup$ Q^0) $=$ $\{\}]\!]$ $\Longrightarrow$ *determ* (P $\Box$ Q)›
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*:
‹(P :: ($'a$, $'r$) *process*$_{ptick}$) $_{FD}\rightsquigarrow^*$ t Q $\Longrightarrow$ *tF t* $\Longrightarrow$ *deterministic P* $\Longrightarrow$ *deterministic Q*›
proof (*induct rule*: *trace-trans.induct*)
show ‹P $_{FD}\rightsquigarrow_\tau$ P' $\Longrightarrow$ *deterministic P* $\Longrightarrow$ *deterministic P'*› **for** P P' :: ‹($'a$, $'r$) *process*$_{ptick}$›
using *deterministic-iff-maximal-for-leFD* **by** *blast*
next
show ‹*tF* [✓(r)] $\Longrightarrow$ *deterministic P*›
for r :: ‹$'r$› **and** P :: ‹($'a$, $'r$) *process*$_{ptick}$› **by** *simp*
next
fix a P P' **and** t :: ‹($'a$, $'r$) *trace*$_{ptick}$› **and** P'' :: ‹($'a$, $'r$) *process*$_{ptick}$›
assume ‹P $_{FD}\rightsquigarrow_a$ P'› ‹*tF* (*ev a* # *t*)› ‹*deterministic P*›
‹*tF t* $\Longrightarrow$ *deterministic P'* $\Longrightarrow$ *deterministic P''*›
from ‹P $_{FD}\rightsquigarrow_a$ P'› ‹*deterministic P*› **have** ‹*deterministic P'*›
using *deterministic-iff-maximal-for-leFD ev-trans-is initial-imp-deterministic-After* **by** *blast*
with ‹*tF t* $\Longrightarrow$ *deterministic P'* $\Longrightarrow$ *deterministic P''*› ‹*tickFree* (*ev a* # *t*)›

show ‹*deterministic* P''› **by** *simp*
qed

lemma *deterministic-imp-Refusals-iff*: ‹*deterministic* $P \Longrightarrow X \in \mathcal{R}\ P \longleftrightarrow X \cap P^0 = \{\}$›
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*:
‹*deterministic* $P \Longrightarrow tF\ t \Longrightarrow$
$(t,\ X) \in \mathcal{F}\ (P :: ('a,\ 'r)\ process_{ptick}) \longleftrightarrow (\exists\, Q.\ (P\ {}_{FD}\!\rightsquigarrow^{*} t\ Q) \wedge X \cap Q^0 = \{\})$›
by (*simp add*: *F-trace-trans-reality-check*)
(*metis deterministic-imp-Refusals-iff tickFree-trace-trans-preserves-deterministic*)

lemma ‹$\neg$ *deterministic* (($a \rightarrow$ *SKIP undefined*) $\Box$ *SKIP undefined*)›
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*] :
‹$[\![$*length xs* = *length ys*; *P* [] []; $\bigwedge$*x1 y1*. *P* [*x1*] [*y1*];
$\bigwedge$*x1 x2 xs y1 y2 ys*. *length xs* = *length ys* $\Longrightarrow$ *P xs ys* $\Longrightarrow$
P (*x2* # *xs*) (*y2* # *ys*) $\Longrightarrow$ *P* (*x1* # *x2* # *xs*) (*y1* # *y2* # *ys*)$]\!]$
$\Longrightarrow$ *P xs ys*›
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*]:
‹$[\![$*P 0*; *P* (*Suc 0*); *P* (*Suc* (*Suc 0*)); $\bigwedge$*k*. *Suc* (*Suc 0*) $\leq$ *k* $\Longrightarrow$ *P k* $\Longrightarrow$ *P* (*Suc k*)$]\!]$ $\Longrightarrow$ *P n*›
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* :
‹*restriction-shift* (*f* ^^ *k*) (*int k* ∗ *m*)›
if ‹*restriction-shift f m*› **for** *f* :: ‹*'a* ⇒ *'a* :: *restriction-space*›
proof (*induct k*)
show ‹*restriction-shift* (*f* ^^ *0*) (*int 0* ∗ *m*)›
by (*simp add*: *restriction-shiftI*)
next
fix *k* **assume** ∗ : ‹*restriction-shift* (*f* ^^ *k*) (*int k* ∗ *m*)›
have ‹*restriction-shift* (*f* ^^ *Suc k*) (*int* (*Suc k*) ∗ *m*) ⟷
restriction-shift (λ*x*. *f* ((*f* ^^ *k*) *x*)) (*int k* ∗ *m* + *m*)›
by (*simp add*: *comp-def distrib-left mult.commute add.commute*)
also have . . . **by** (*fact restriction-shift-comp-restriction-shift*[*OF that* ∗])
finally show ‹*restriction-shift* (*f* ^^ *Suc k*) (*int* (*Suc k*) ∗ *m*)› **.**

qed

lemma *non-destructive-iterated* :
‹*non-destructive f* $\Longrightarrow$ *non-destructive* (*f* ^^ *k*)›
for *f* :: ‹*'a* $\Rightarrow$ *'a* :: *restriction-space*›
by (*metis mult.commute mult-zero-left non-destructive-def non-destructive-on-def restriction-shift-def restriction-shift-iterated*)

lemma *constructive-iterated* :
‹*constructive* (*f* ^^ *k*)› **if** ‹*0* < *k*› ‹*constructive f*›
for *f* :: ‹*'a* $\Rightarrow$ *'a* :: *restriction-space*›
proof −
from ‹*constructive f*› **have** ‹*restriction-shift f 1*›
unfolding *constructive-def constructive-on-def restriction-shift-def* **by** *blast*
with *restriction-shift-iterated*
have ‹*restriction-shift* (*f* ^^ *k*) (*int k* ∗ *1*)› **.**
hence ‹*restriction-shift* (*f* ^^ *k*) (*int k*)› **by** *simp*
with ‹*0* < *k*› **show** ‹*constructive* (*f* ^^ *k*)›
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* :
‹⟦*0* < *k*; *constructive f*; (*f* ^^ *k*) *x* = *x*⟧ $\Longrightarrow$ (υ *x*. *f x*) = *x*›
by (*metis constructive-iterated funpow-swap1 restriction-fix-unique*)

lemma *restriction-fix-iterated* :
‹*0* < *k* $\Longrightarrow$ *constructive f* $\Longrightarrow$ (υ *x*. (*f* ^^ *k*) *x*) = (υ *x*. *f x*)›
by (*metis constructive-iterated restriction-fix-eq restriction-fix-unique-iterated*)

corollary *restriction-fix-ind-iterated*
[*consumes 1*, *case-names constructive adm base step*]:
‹*P* (υ *x*. *f x*)› **if** ‹*0* < *k*› ‹*constructive f*› ‹$adm_{\downarrow}$ *P*› ‹*P x*› ‹⋀*x*. *P x* $\Longrightarrow$ *P* ((*f* ^^ *k*) *x*)›
proof −
from *constructive-iterated that*(*1*, *2*) **have** ‹*constructive* (*f* ^^ *k*)› **.**
from *restriction-fix-ind*[*OF this that*(*3*−*5*)] **have** ‹*P* (υ *x*. (*f* ^^ *k*) *x*)› **.**
also from *restriction-fix-iterated that*(*1*, *2*) **have** ‹(υ *x*. (*f* ^^ *k*) *x*) = (υ *x*. *f x*)› **.**
finally show ‹*P* (υ *x*. *f x*)› **.**
qed

3.1 Definitions

3.1.1 Non-deterministic and deterministic Automata

unbundle *option-type-syntax*

type-synonym $('\sigma, 'a)$ *enabl* $= \langle '\sigma \Rightarrow 'a\ set\rangle$
type-synonym $('\sigma, 'a, '\sigma')$ *trans* $= \langle '\sigma \Rightarrow 'a \Rightarrow '\sigma'\rangle$
type-synonym $('\sigma, 'a)$ $trans_d$ $= \langle ('\sigma, 'a, '\sigma\ option)\ trans\rangle$
type-synonym $('\sigma, 'a)$ $trans_{nd}$ $= \langle ('\sigma, 'a, '\sigma\ set)\ trans\rangle$

record $('\sigma, 'a, '\sigma', 'r)$ A =
  τ :: $\langle ('\sigma, 'a, '\sigma')\ trans\rangle$
  ω :: $\langle '\sigma \Rightarrow 'r\rangle$

type-synonym $('\sigma, 'a, 'r)$ A_d $= \langle ('\sigma, 'a, '\sigma\ option, 'r\ option)\ A\rangle$
type-synonym $('\sigma, 'a, 'r, '\alpha)$ A_d*-scheme* $= \langle ('\sigma, 'a, '\sigma\ option, 'r\ option, '\alpha)$ *A-scheme*$\rangle$
type-synonym $('\sigma, 'a, 'r)$ A_{nd} $= \langle ('\sigma, 'a, '\sigma\ set, 'r\ set)\ A\rangle$
type-synonym $('\sigma, 'a, 'r, '\alpha)$ A_{nd}*-scheme* $= \langle ('\sigma, 'a, '\sigma\ set, 'r\ set, '\alpha)\ A\text{-}scheme\rangle$

3.1.2 Enableness

consts ε :: $\langle ('\sigma, 'a, '\sigma', 'r', '\alpha)\ A\text{-}scheme \Rightarrow ('\sigma, 'a)\ enabl\rangle$
overloading
  $\varepsilon_d \equiv \langle \varepsilon :: ('\sigma, 'a, '\sigma\ option, 'r', '\alpha)\ A\text{-}scheme \Rightarrow ('\sigma, 'a)\ enabl\rangle$
  $\varepsilon_{nd} \equiv \langle \varepsilon :: ('\sigma, 'a, '\sigma\ set, 'r', '\alpha)\ A\text{-}scheme \Rightarrow ('\sigma, 'a)\ enabl\rangle$
begin
fun ε_d :: $\langle ('\sigma, 'a, '\sigma\ option, 'r', '\alpha)\ A\text{-}scheme \Rightarrow ('\sigma, 'a)\ enabl\rangle$
  where $\langle \varepsilon_d\ A\ \sigma = \{a.\ \tau\ A\ \sigma\ a \neq \Diamond\}\rangle$
fun ε_{nd} :: $\langle ('\sigma, 'a, '\sigma\ set, 'r', '\alpha)\ A\text{-}scheme \Rightarrow ('\sigma, 'a)\ enabl\rangle$
  where $\langle \varepsilon_{nd}\ A\ \sigma = \{a.\ \tau\ A\ \sigma\ a \neq \{\}\}\rangle$
end

lemmas ε*-simps*[*simp del*] = ε_d*.simps* ε_{nd}*.simps*

3.1.3 States allowing Termination

consts ϱ :: $\langle ('\sigma, 'a, '\sigma', 'r', '\alpha)\ A\text{-}scheme \Rightarrow '\sigma\ set\rangle$
overloading
  $\varrho_d \equiv \langle \varrho :: ('\sigma, 'a, '\sigma', 'r\ option, '\alpha)\ A\text{-}scheme \Rightarrow '\sigma\ set\rangle$
  $\varrho_{nd} \equiv \langle \varrho :: ('\sigma, 'a, '\sigma', 'r\ set, '\alpha)\ A\text{-}scheme \Rightarrow '\sigma\ set\rangle$
begin
fun ϱ_d :: $\langle ('\sigma, 'a, '\sigma', 'r\ option, '\alpha)\ A\text{-}scheme \Rightarrow '\sigma\ set\rangle$
  where $\langle \varrho_d\ A = \{\sigma.\ \omega\ A\ \sigma \neq \Diamond\}\rangle$
fun ϱ_{nd} :: $\langle ('\sigma, 'a, '\sigma', 'r\ set, '\alpha)\ A\text{-}scheme \Rightarrow '\sigma\ set\rangle$
  where $\langle \varrho_{nd}\ A = \{\sigma.\ \omega\ A\ \sigma \neq \{\}\}\rangle$
end

lemmas ϱ*-simps*[*simp del*] = ϱ_d*.simps* ϱ_{nd}*.simps*

3.1.4 Reachability

inductive-set $\mathcal{R}_d$ :: ‹$('\sigma, 'a, 'r, '\alpha)$ A_d-*scheme* $\Rightarrow$ $'\sigma \Rightarrow '\sigma$ *set*›
for A :: ‹$('\sigma, 'a, 'r, '\alpha)$ A_d-*scheme*› **and** σ :: $'\sigma$
where *init* : ‹$\sigma \in \mathcal{R}_d\ A\ \sigma$›
| *step* : ‹$\sigma' \in \mathcal{R}_d\ A\ \sigma \Longrightarrow \lfloor\sigma''\rfloor = \tau\ A\ \sigma'\ a \Longrightarrow \sigma'' \in \mathcal{R}_d\ A\ \sigma$›

inductive-set $\mathcal{R}_{nd}$ :: ‹$('\sigma, 'a, 'r, '\alpha)$ A_{nd}-*scheme* $\Rightarrow$ $'\sigma \Rightarrow '\sigma$ *set*›
for A :: ‹$('\sigma, 'a, 'r, '\alpha)$ A_{nd}-*scheme*› **and** σ :: $'\sigma$
where *init* : ‹$\sigma \in \mathcal{R}_{nd}\ A\ \sigma$›
| *step* : ‹$\sigma' \in \mathcal{R}_{nd}\ A\ \sigma \Longrightarrow \sigma'' \in \tau\ A\ \sigma'\ a \Longrightarrow \sigma'' \in \mathcal{R}_{nd}\ A\ \sigma$›

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

lemma $\mathcal{R}_{nd}$-*trans*: ‹$\sigma'' \in \mathcal{R}_{nd}\ A\ \sigma' \Longrightarrow \sigma' \in \mathcal{R}_{nd}\ A\ \sigma \Longrightarrow \sigma'' \in \mathcal{R}_{nd}\ A\ \sigma$›
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-det-to-ndet* ::
‹$('\sigma, 'a, 'r, '\alpha)$ A_d-*scheme* $\Rightarrow$ $('\sigma, 'a, 'r, '\alpha)$ A_{nd}-*scheme*›
where ‹*from-det-to-ndet* $A \equiv$
$(\!|\tau = \lambda\sigma\ a.\ case\ \tau\ A\ \sigma\ a\ of\ \lfloor\sigma'\rfloor \Rightarrow \{\sigma'\} \mid \Diamond \Rightarrow \{\},$
$\omega = \lambda\sigma.\ case\ \omega\ A\ \sigma\ of\ \lfloor r\rfloor \Rightarrow \{r\} \mid \Diamond \Rightarrow \{\}, \ldots = more\ A|\!)$›
definition *from-ndet-to-det* ::
‹$('\sigma, 'a, 'r, '\alpha)$ A_{nd}-*scheme* $\Rightarrow$ $('\sigma, 'a, 'r, '\alpha)$ A_d-*scheme*›
where ‹*from-ndet-to-det* $A \equiv$
$(\!|\tau = \lambda\sigma\ a.\ if\ \tau\ A\ \sigma\ a = \{\}\ then\ \Diamond\ else\ \lfloor THE\ \sigma'.\ \sigma' \in \tau\ A\ \sigma\ a\rfloor,$
$\omega = \lambda\sigma.\ if\ \omega\ A\ \sigma = \{\}\ then\ \Diamond\ else\ \lfloor THE\ r.\ r \in \omega\ A\ \sigma\rfloor, \ldots = more\ A|\!)$›

definition *from-*σ*-to-*σs_d ::
‹$('\sigma, 'a, 'r, '\alpha)$ A_d-*scheme* $\Rightarrow$ $('\sigma\ list, 'a, 'r, '\alpha)$ A_d-*scheme*›
where ‹*from-*σ*-to-*σs_d $A \equiv$
$(\!|\tau = \lambda\sigma s\ a.\ case\ \tau\ A\ (hd\ \sigma s)\ a\ of\ \lfloor\sigma'\rfloor \Rightarrow \lfloor\lfloor\sigma'\rfloor\rfloor \mid \Diamond \Rightarrow \Diamond,$
$\omega = \lambda\sigma s.\ \omega\ A\ (hd\ \sigma s), \ldots = more\ A|\!)$›
definition *from-*σ*-to-*σs_{nd} ::
‹$('\sigma, 'a, 'r, '\alpha)$ A_{nd}-*scheme* $\Rightarrow$ $('\sigma\ list, 'a, 'r, '\alpha)$ A_{nd}-*scheme*›
where ‹*from-*σ*-to-*σs_{nd} $A \equiv$
$(\!|\tau = \lambda\sigma s\ a.\ \{\lfloor\sigma'\rfloor\ |\sigma'.\ \sigma' \in \tau\ A\ (hd\ \sigma s)\ a\},$
$\omega = \lambda\sigma s.\ \omega\ A\ (hd\ \sigma s), \ldots = more\ A|\!)$›

definition *from-*σs*-to-*σ_d ::
‹$('\sigma\ list, 'a, 'r, '\alpha)$ A_d-*scheme* $\Rightarrow$ $('\sigma, 'a, 'r, '\alpha)$ A_d-*scheme*›
where ‹*from-*σs*-to-*σ_d $A \equiv$
$(\!|\tau = \lambda\sigma\ a.\ case\ \tau\ A\ [\sigma]\ a\ of\ \lfloor\sigma s'\rfloor \Rightarrow \lfloor hd\ \sigma s'\rfloor \mid \Diamond \Rightarrow \Diamond,$

$\omega = \lambda\sigma.\ \omega\ A\ [\sigma],\ \ldots = more\ A\|\rangle$
definition *from-σs-to-σ$_{nd}$* ::
‹($'\sigma$ *list*, $'a$, $'r$, $'\alpha$) A_{nd}*-scheme* ⇒ ($'\sigma$, $'a$, $'r$, $'\alpha$) A_{nd}*-scheme*›
where ‹*from-σs-to-σ$_{nd}$* A ≡
$\|\tau = \lambda\sigma\ a.\ \{hd\ \sigma s'\ |\sigma s'.\ \sigma s' \in \tau\ A\ [\sigma]\ a\},$
$\omega = \lambda\sigma.\ \omega\ A\ [\sigma],\ \ldots = more\ A\|\rangle$

definition *from-singl-to-list$_d$* ::
‹($'\sigma$, $'a$, $'r$, $'\alpha$) A_d*-scheme* ⇒ ($'\sigma$ *list*, $'a$, $'r$ *list*, $'\alpha$) A_d*-scheme*›
where ‹*from-singl-to-list$_d$* A ≡
$\|\tau = \lambda\sigma s\ a.\ case\ \tau\ A\ (hd\ \sigma s)\ a\ of\ \lfloor\sigma'\rfloor \Rightarrow \lfloor[\sigma']\rfloor \mid \Diamond \Rightarrow \Diamond,$
$\omega = \lambda\sigma s.\ case\ \omega\ A\ (hd\ \sigma s)\ of\ \lfloor r\rfloor \Rightarrow \lfloor[r]\rfloor \mid \Diamond \Rightarrow \Diamond,\ \ldots = more\ A\|\rangle$
definition *from-singl-to-list$_{nd}$* ::
‹($'\sigma$, $'a$, $'r$, $'\alpha$) A_{nd}*-scheme* ⇒ ($'\sigma$ *list*, $'a$, $'r$ *list*, $'\alpha$) A_{nd}*-scheme*›
where ‹*from-singl-to-list$_{nd}$* A ≡
$\|\tau = \lambda\sigma s\ a.\ \{[\sigma']\ |\sigma'.\ \sigma' \in \tau\ A\ (hd\ \sigma s)\ a\},$
$\omega = \lambda\sigma s.\ \{[r]\ |r.\ r \in \omega\ A\ (hd\ \sigma s)\},\ \ldots = more\ A\|\rangle$

definition *from-list-to-singl$_d$* ::
‹($'\sigma$ *list*, $'a$, $'r$ *list*, $'\alpha$) A_d*-scheme* ⇒ ($'\sigma$, $'a$, $'r$, $'\alpha$) A_d*-scheme*›
where ‹*from-list-to-singl$_d$* A ≡
$\|\tau = \lambda\sigma\ a.\ case\ \tau\ A\ [\sigma]\ a\ of\ \lfloor\sigma s'\rfloor \Rightarrow \lfloor hd\ \sigma s'\rfloor \mid \Diamond \Rightarrow \Diamond,$
$\omega = \lambda\sigma.\ case\ \omega\ A\ [\sigma]\ of\ \lfloor rs\rfloor \Rightarrow \lfloor hd\ rs\rfloor \mid \Diamond \Rightarrow \Diamond,\ \ldots = more\ A\|\rangle$
definition *from-list-to-singl$_{nd}$* ::
‹($'\sigma$ *list*, $'a$, $'r$ *list*, $'\alpha$) A_{nd}*-scheme* ⇒ ($'\sigma$, $'a$, $'r$, $'\alpha$) A_{nd}*-scheme*›
where ‹*from-list-to-singl$_{nd}$* A ≡
$\|\tau = \lambda\sigma\ a.\ \{hd\ \sigma s'\ |\sigma s'.\ \sigma s' \in \tau\ A\ [\sigma]\ a\},$
$\omega = \lambda\sigma.\ \{hd\ rs\ |rs.\ rs \in \omega\ A\ [\sigma]\},\ \ldots = more\ A\|\rangle$

lemmas *det-ndet-conv-defs* = *from-det-to-ndet-def from-ndet-to-det-def*
and *σ-σs-conv-defs* = *from-σ-to-σs$_d$-def from-σ-to-σs$_{nd}$-def*
from-σs-to-σ$_d$-def from-σs-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* (‹$\langle\!\langle$-$\rangle\!\rangle_{d\hookrightarrow nd}$› [*0*])
notation *from-ndet-to-det* (‹$\langle\!\langle$-$\rangle\!\rangle_{nd\rightsquigarrow d}$› [*0*])
notation *from-σ-to-σs$_d$* (‹$_d\langle\!\langle$-$\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$› [*0*])
notation *from-σ-to-σs$_{nd}$* (‹$_{nd}\langle\!\langle$-$\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$› [*0*])
notation *from-σs-to-σ$_d$* (‹$_d\langle\!\langle$-$\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$› [*0*])
notation *from-σs-to-σ$_{nd}$* (‹$_{nd}\langle\!\langle$-$\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$› [*0*])
notation *from-singl-to-list$_d$* (‹$_d\langle\!\langle$-$\rangle\!\rangle_{singl\hookrightarrow list}$› [*0*])
notation *from-singl-to-list$_{nd}$* (‹$_{nd}\langle\!\langle$-$\rangle\!\rangle_{singl\hookrightarrow list}$› [*0*])
notation *from-list-to-singl$_d$* (‹$_d\langle\!\langle$-$\rangle\!\rangle_{list\rightsquigarrow singl}$› [*0*])
notation *from-list-to-singl$_{nd}$* (‹$_{nd}\langle\!\langle$-$\rangle\!\rangle_{list\rightsquigarrow singl}$› [*0*])

end

unbundle *functional-automata-morphisms-syntax*

lemma *morphisms-A-scheme-more-simps* [*simp*] :
‹*more* $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$ = *more A*› ‹*more* $\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}$ = *more B*›
‹*more* $_{d}\langle\!\langle C\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *more C*› ‹*more* $_{nd}\langle\!\langle D\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *more D*›
‹*more* $_{d}\langle\!\langle E\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = *more E*› ‹*more* $_{nd}\langle\!\langle F\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = *more F*›
‹*more* $_{d}\langle\!\langle G\rangle\!\rangle_{singl\hookrightarrow list}$ = *more G*› ‹*more* $_{nd}\langle\!\langle H\rangle\!\rangle_{singl\hookrightarrow list}$ = *more H*›
‹*more* $_{d}\langle\!\langle I\rangle\!\rangle_{list\rightsquigarrow singl}$ = *more I*› ‹*more* $_{nd}\langle\!\langle J\rangle\!\rangle_{list\rightsquigarrow singl}$ = *more J*›
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** ‹*update-both* A_0 A_1 σ_0 σ_1 *e f* ≡ *f* (τ A_0 σ_0 *e*) (τ A_1 σ_1 *e*)›

definition *update-left* **where** ‹*update-left* A_0 σ_0 σ_1 *e f g* ≡ *f* (τ A_0 σ_0 *e*) (*g* σ_1)›

definition *update-right* **where** ‹*update-right* A_1 σ_0 σ_1 *e f g* ≡ *f* (*g* σ_0) (τ A_1 σ_1 *e*)›

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

abbreviation *f-up-set* **where** ‹*f-up-set f B C* ≡ {*f s t*| *s t*. (*s*, *t*) ∈ *B* × *C*}›

abbreviation *f-up-opt* **where** ‹*f-up-opt f s t* ≡ *case s of* ◊ ⇒ ◊ | ⌊*s′*⌋ ⇒ *map-option* (*f s′*) *t*›

3.1.7 Assumptions on Automata

definition *finite-trans* :: ‹(′σ, ′*a*, ′*r*, ′α) A_{nd}-*scheme* ⇒ *bool*›
where ‹*finite-trans A* ≡ ∀σ *a*. *finite* (τ *A* σ *a*)›

lemma *finite-trans-morphisms-simps*[*simp*]:
‹*finite-trans* $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$›
‹*finite-trans B* ⟹ *finite-trans* $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›
‹*finite-trans C* ⟹ *finite-trans* $_{nd}\langle\!\langle C\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$›
‹*finite-trans D* ⟹ *finite-trans* $_{nd}\langle\!\langle D\rangle\!\rangle_{singl\hookrightarrow list}$›
‹*finite-trans E* ⟹ *finite-trans* $_{nd}\langle\!\langle E\rangle\!\rangle_{list\rightsquigarrow singl}$›
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* :: ‹(′σ, ′*a*, ′*r*, ′α) A_{nd}-*scheme* ⇒ *bool*›
where ‹*at-most-1-elem A* ≡
(∀σ *a*. τ *A* σ *a* = {} ∨ (∃σ′. τ *A* σ *a* = {σ′})) ∧
(∀σ. ω *A* σ = {} ∨ (∃ *r*. ω *A* σ = {*r*}))›

lemma *at-most-1-elem-def-bis* :
‹*at-most-1-elem A* $\longleftrightarrow$ $(\forall \sigma\ a.\ \exists \sigma'.\ \tau\ A\ \sigma\ a \subseteq \{\sigma'\}) \wedge (\forall \sigma.\ \exists r.\ \omega\ A\ \sigma \subseteq \{r\})$›
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* :
‹$[\![\bigwedge \sigma\ a.\ \tau\ A\ \sigma\ a = \{\} \vee (\exists \sigma'.\ \tau\ A\ \sigma\ a = \{\sigma'\});$
$\bigwedge \sigma.\ \omega\ A\ \sigma = \{\} \vee (\exists r.\ \omega\ A\ \sigma = \{r\})]\!] \Longrightarrow$ *at-most-1-elem A*›
by (*simp add*: *at-most-1-elem-def*)

lemma *at-most-1-elemE* :
‹$[\![\tau\ A\ \sigma\ a = \{\} \Longrightarrow thesis;\ \bigwedge \sigma'.\ \tau\ A\ \sigma\ a = \{\sigma'\} \Longrightarrow thesis]\!] \Longrightarrow thesis$›
‹$[\![\omega\ A\ \sigma = \{\} \Longrightarrow thesis;\ \bigwedge r.\ \omega\ A\ \sigma = \{r\} \Longrightarrow thesis]\!] \Longrightarrow thesis$›
if ‹*at-most-1-elem A*›
by (*meson at-most-1-elem-def* ‹*at-most-1-elem A*›)+

definition *at-most-1-elem-trans* :: ‹$('\sigma,\ 'a,\ 'r,\ '\alpha)\ A_{nd}$*-scheme* $\Rightarrow$ *bool*›
where ‹*at-most-1-elem-trans A* $\equiv \forall \sigma\ a.\ \tau\ A\ \sigma\ a = \{\} \vee (\exists \sigma'.\ \tau\ A\ \sigma\ a = \{\sigma'\})$›

lemma *at-most-1-elem-trans-def-bis* :
‹*at-most-1-elem-trans A* $\longleftrightarrow$ $(\forall \sigma\ a.\ \exists \sigma'.\ \tau\ A\ \sigma\ a \subseteq \{\sigma'\})$›
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* :
‹$[\![\bigwedge \sigma\ a.\ \tau\ A\ \sigma\ a = \{\} \vee (\exists \sigma'.\ \tau\ A\ \sigma\ a = \{\sigma'\})]\!] \Longrightarrow$ *at-most-1-elem-trans A*›
by (*simp add*: *at-most-1-elem-trans-def*)

lemma *at-most-1-elem-transE* :
‹$[\![\tau\ A\ \sigma\ a = \{\} \Longrightarrow thesis;\ \bigwedge \sigma'.\ \tau\ A\ \sigma\ a = \{\sigma'\} \Longrightarrow thesis]\!] \Longrightarrow thesis$›
if ‹*at-most-1-elem-trans A*›
by (*meson at-most-1-elem-trans-def* ‹*at-most-1-elem-trans A*›)+

lemma *at-most-1-elem-imp-at-most-1-elem-trans* :
‹*at-most-1-elem A* $\Longrightarrow$ *at-most-1-elem-trans A*›
by (*simp add*: *at-most-1-elem-def at-most-1-elem-trans-def*)

definition *length-1-trans*$_d$ :: ‹$('\sigma\ list,\ 'a,\ 'r,\ '\alpha)\ A_d$*-scheme* $\Rightarrow$ *bool*›
where ‹*length-1-trans*$_d$ *A* $\equiv$
$\forall \sigma s\ a.\ case\ \tau\ A\ \sigma s\ a\ of\ \Diamond \Rightarrow True \mid \lfloor \sigma s' \rfloor \Rightarrow length\ \sigma s' = Suc\ 0$›

lemma *length-1-trans*$_d$*I* :

‹$[\![\bigwedge \sigma s\ a\ \sigma s'.\ \tau\ A\ \sigma s\ a = \lfloor \sigma s' \rfloor \Longrightarrow \mathit{length}\ \sigma s' = \mathit{Suc}\ 0]\!] \Longrightarrow \mathit{length\text{-}1\text{-}trans}_d\ A$›
by (*simp add*: *length-1-trans*$_d$*-def split*: *option.split*)

lemma *length-1-trans*$_d$*E* :
‹$[\![\mathit{length\text{-}1\text{-}trans}_d\ A;\ \tau\ A\ \sigma s\ a = \lfloor \sigma s' \rfloor;\ \bigwedge \sigma.\ \sigma s' = [\sigma] \Longrightarrow \mathit{thesis}]\!] \Longrightarrow \mathit{thesis}$›
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*$_{nd}$ :: ‹$('\sigma\ \mathit{list},\ 'a,\ 'r,\ '\alpha)\ A_{nd}\text{-}\mathit{scheme} \Rightarrow \mathit{bool}$›
where ‹$\mathit{length\text{-}1\text{-}trans}_{nd}\ A \equiv \forall \sigma s\ a.\ \forall \sigma s' \in \tau\ A\ \sigma s\ a.\ \mathit{length}\ \sigma s' = \mathit{Suc}\ 0$›

lemma *length-1-trans*$_{nd}$*I* :
‹$[\![\bigwedge \sigma s\ a\ \sigma s'.\ \sigma s' \in \tau\ A\ \sigma s\ a \Longrightarrow \mathit{length}\ \sigma s' = \mathit{Suc}\ 0]\!] \Longrightarrow \mathit{length\text{-}1\text{-}trans}_{nd}\ A$›
by (*simp add*: *length-1-trans*$_{nd}$*-def split*: *option.split*)

lemma *length-1-trans*$_{nd}$*E* :
‹$[\![\mathit{length\text{-}1\text{-}trans}_{nd}\ A;\ \sigma s' \in \tau\ A\ \sigma s\ a;\ \bigwedge \sigma.\ \sigma s' = [\sigma] \Longrightarrow \mathit{thesis}]\!] \Longrightarrow \mathit{thesis}$›
by (*simp add*: *length-1-trans*$_{nd}$*-def split*: *option.split-asm*)
(*metis* (*no-types*) *length-0-conv length-Suc-conv*)

definition *length-1*$_d$ :: ‹$('\sigma\ \mathit{list},\ 'a,\ 'r\ \mathit{list},\ '\alpha)\ A_d\text{-}\mathit{scheme} \Rightarrow \mathit{bool}$›
where ‹$\mathit{length\text{-}1}_d\ A \equiv$
$(\forall \sigma s\ a.\ \mathit{case}\ \tau\ A\ \sigma s\ a\ \mathit{of}\ \Diamond \Rightarrow \mathit{True} \mid \lfloor \sigma s' \rfloor \Rightarrow \mathit{length}\ \sigma s' = \mathit{Suc}\ 0) \wedge$
$(\forall \sigma s.\ \mathit{case}\ \omega\ A\ \sigma s\ \mathit{of}\ \Diamond \Rightarrow \mathit{True} \mid \lfloor rs \rfloor \Rightarrow \mathit{length}\ rs = \mathit{Suc}\ 0)$›

lemma *length-1*$_d$*I* :
‹$[\![\bigwedge \sigma s\ a\ \sigma s'.\ \tau\ A\ \sigma s\ a = \lfloor \sigma s' \rfloor \Longrightarrow \mathit{length}\ \sigma s' = \mathit{Suc}\ 0;$
$\bigwedge \sigma s\ rs.\ \omega\ A\ \sigma s = \lfloor rs \rfloor \Longrightarrow \mathit{length}\ rs = \mathit{Suc}\ 0]\!] \Longrightarrow \mathit{length\text{-}1}_d\ A$›
by (*simp add*: *length-1*$_d$*-def split*: *option.split*)

lemma *length-1*$_d$*E* :
‹$[\![\mathit{length\text{-}1}_d\ A;\ \tau\ A\ \sigma s\ a = \lfloor \sigma s' \rfloor;\ \bigwedge \sigma.\ \sigma s' = [\sigma] \Longrightarrow \mathit{thesis}]\!] \Longrightarrow \mathit{thesis}$›
‹$[\![\mathit{length\text{-}1}_d\ A;\ \omega\ A\ \sigma s = \lfloor rs \rfloor;\ \bigwedge r.\ rs = [r] \Longrightarrow \mathit{thesis}]\!] \Longrightarrow \mathit{thesis}$›
by (*simp add*: *length-1*$_d$*-def split*: *option.split-asm*,
metis (*no-types*) *length-0-conv length-Suc-conv*)+

definition *length-1*$_{nd}$ :: ‹$('\sigma\ \mathit{list},\ 'a,\ 'r\ \mathit{list},\ '\alpha)\ A_{nd}\text{-}\mathit{scheme} \Rightarrow \mathit{bool}$›
where ‹$\mathit{length\text{-}1}_{nd}\ A \equiv (\forall \sigma s\ a.\ \forall \sigma s' \in \tau\ A\ \sigma s\ a.\ \mathit{length}\ \sigma s' = \mathit{Suc}\ 0) \wedge$
$(\forall \sigma s.\ \forall rs \in \omega\ A\ \sigma s.\ \mathit{length}\ rs = \mathit{Suc}\ 0)$›

lemma *length-1*$_{nd}$*I* :
‹$[\![\bigwedge \sigma s\ a\ \sigma s'.\ \sigma s' \in \tau\ A\ \sigma s\ a \Longrightarrow \mathit{length}\ \sigma s' = \mathit{Suc}\ 0;$
$\bigwedge \sigma s\ rs.\ rs \in \omega\ A\ \sigma s \Longrightarrow \mathit{length}\ rs = \mathit{Suc}\ 0]\!] \Longrightarrow \mathit{length\text{-}1}_{nd}\ A$›
by (*simp add*: *length-1*$_{nd}$*-def split*: *option.split*)

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

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

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

lemma *indep-enablI* :
‹($\bigwedge t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow t_1 \in \mathcal{R}_d\ A_1\ \sigma_1 \Longrightarrow \varepsilon\ A_0\ t_0 \cap \varepsilon\ A_1\ t_1 \subseteq E$)
$\Longrightarrow$ *indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
and *indep-enablD* :
‹$[\![$*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$; $t_0 \in \mathcal{R}_d\ A_0\ \sigma_0$; $t_1 \in \mathcal{R}_d\ A_1\ \sigma_1]\!] \Longrightarrow \varepsilon\ A_0\ t_0 \cap \varepsilon\ A_1\ t_1 \subseteq E$›
by (*simp-all add*: *indep-enabl-def*)

definition *ϱ-disjoint-ε* :: ‹($'\sigma$, $'a$, $'\sigma'$, $'r'$, $'\alpha$) *A-scheme* $\Rightarrow$ *bool*›
where ‹*ϱ-disjoint-ε* $A \equiv \forall\, \sigma \in \varrho\ A.\ \varepsilon\ A\ \sigma = \{\}$›

lemma *ϱ-disjoint-εI* : ‹($\bigwedge\sigma.\ \sigma \in \varrho\ A \Longrightarrow \varepsilon\ A\ \sigma = \{\}$) $\Longrightarrow$ *ϱ-disjoint-ε* A›
and *ϱ-disjoint-εD* : ‹*ϱ-disjoint-ε* $A \Longrightarrow \sigma \in \varrho\ A \Longrightarrow \varepsilon\ A\ \sigma = \{\}$›
by (*simp-all add*: *ϱ-disjoint-ε-def*)

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

lemma *at-most-1-elem-term-def-bis* :
‹*at-most-1-elem-term* $A \longleftrightarrow (\forall\, \sigma.\ \exists\, r.\ \omega\ A\ \sigma \subseteq \{r\})$›
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* :
‹$[\![\bigwedge\sigma.\ \omega\ A\ \sigma = \{\} \vee (\exists\, r.\ \omega\ A\ \sigma = \{r\})]\!] \Longrightarrow$ *at-most-1-elem-term* A›
by (*simp add*: *at-most-1-elem-term-def*)

lemma *at-most-1-elem-termE* :
‹$[\![\omega\ A\ \sigma = \{\} \Longrightarrow$ *thesis*; $\bigwedge r.\ \omega\ A\ \sigma = \{r\} \Longrightarrow$ *thesis*$]\!] \Longrightarrow$ *thesis*›
if ‹*at-most-1-elem-term* A›
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* :
‹*at-most-1-elem A* $\Longrightarrow$ *at-most-1-elem-term A*›
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*]:
‹ε ($\lfloor\!\lfloor\tau = \lambda\sigma\ a.\ \Diamond, \omega = \lambda\sigma.\ \Diamond, \ldots = some\rfloor\!\rfloor$ :: ($'\sigma$, $'a$, $'r$, $'\alpha$) A_d-*scheme*) $\sigma = \{\}$›
‹ε ($\lfloor\!\lfloor\tau = \lambda\sigma\ a.\ \{\}, \omega = \lambda\sigma.\ \{\}, \ldots = some\rfloor\!\rfloor$ :: ($'\sigma$, $'a$, $'r$, $'\alpha$) A_{nd}-*scheme*) $\sigma = \{\}$›
by (*simp-all add*: *ε-simps*)

lemma *base-trans-ϱ*[*simp*]:
‹ϱ ($\lfloor\!\lfloor\tau = \lambda\sigma\ a.\ \Diamond, \omega = \lambda\sigma.\ \Diamond, \ldots = some\rfloor\!\rfloor$ :: ($'\sigma$, $'a$, $'r$, $'\alpha$) A_d-*scheme*) = $\{\}$›
‹ϱ ($\lfloor\!\lfloor\tau = \lambda\sigma\ a.\ \{\}, \omega = \lambda\sigma.\ \{\}, \ldots = some\rfloor\!\rfloor$ :: ($'\sigma$, $'a$, $'r$, $'\alpha$) A_{nd}-*scheme*) = $\{\}$›
by (*simp-all add*: *ϱ-simps*)

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

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

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

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

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

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

lemma *ω-from-det-to-ndet* :
‹ω $\langle\!\langle A \rangle\!\rangle_{d \hookrightarrow nd} = (\lambda\sigma.\ case\ \omega\ A\ \sigma\ of\ \lfloor r \rfloor \Rightarrow \{r\} \mid \Diamond \Rightarrow \{\})$›
by (*auto simp add*: *det-ndet-conv-defs*)

lemma *ε-ω-useless* [*simp*] :
‹$\varepsilon\ (A(\!|\omega := some\text{-}\omega|\!)) = \varepsilon\ A$› ‹$\varepsilon\ (B(\!|\omega := some\text{-}\omega'|\!)) = \varepsilon\ B$›
for A :: ‹$('\sigma,\ 'a,\ '\sigma\ option,\ 'r\ option,\ '\alpha)\ A\text{-}scheme$›
and B :: ‹$('\sigma,\ 'a,\ '\sigma\ set,\ 'r\ set,\ '\alpha)\ A\text{-}scheme$›
by (*rule ext*, *simp add*: *ε-simps*)+

lemma *ϱ-disjoint-ε-updated-ω* [*simp*] :
‹$\varrho\text{-}disjoint\text{-}\varepsilon\ (A(\!|\omega := \lambda\sigma.\ \Diamond|\!))$›
‹$\varrho\text{-}disjoint\text{-}\varepsilon\ (B(\!|\omega := \lambda\sigma.\ \{\}|\!))$›
by (*simp-all add*: *ϱ-disjoint-ε-def ϱ-simps*)

lemma *ϱ-disjoint-ε-det-ndet-conv-iff* [*simp*] :
‹$\varrho\text{-}disjoint\text{-}\varepsilon\ \langle\!\langle A \rangle\!\rangle_{d \hookrightarrow nd} \longleftrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A$›
‹$\varrho\text{-}disjoint\text{-}\varepsilon\ \langle\!\langle B \rangle\!\rangle_{nd \rightsquigarrow d} \longleftrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ B$›
by (*simp-all add*: *ϱ-disjoint-ε-def*)

lemma *at-most-1-elem-term-updated-ω* [*simp*] :
‹$at\text{-}most\text{-}1\text{-}elem\text{-}term\ (A(\!|\omega := \lambda\sigma.\ \{\}|\!))$›
by (*simp add*: *at-most-1-elem-term-def*)

lemma *at-most-1-elem-term-from-det-to-ndet* [*simp*] :
‹$at\text{-}most\text{-}1\text{-}elem\text{-}term\ \langle\!\langle A \rangle\!\rangle_{d \hookrightarrow nd}$›
by (*simp add*: *det-ndet-conv-defs at-most-1-elem-term-def split*: *option.split*)

lemma *at-most-1-elem-term-unit* [*simp*] :
‹$at\text{-}most\text{-}1\text{-}elem\text{-}term\ (A :: ('\sigma,\ 'a,\ unit,\ '\alpha)\ A_{nd}\text{-}scheme)$›
by (*auto simp add*: *at-most-1-elem-term-def*)

3.2.2 Properties of our morphisms

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

lemma *base-trans-det-ndet-conv*:
‹$\langle\!\langle(\!|\tau = \lambda\sigma\ a.\ \Diamond,\ \omega = \lambda\sigma.\ \Diamond,\ \ldots = some|\!)\rangle\!\rangle_{d \hookrightarrow nd} =$
$(\!|\tau = \lambda\sigma\ a.\ \{\},\ \omega = \lambda\sigma.\ \{\},\ \ldots = some|\!)$›
‹$\langle\!\langle(\!|\tau = \lambda\sigma\ a.\ \{\},\ \omega = \lambda\sigma.\ \{\},\ \ldots = some|\!)\rangle\!\rangle_{nd \rightsquigarrow d} =$
$(\!|\tau = \lambda\sigma\ a.\ \Diamond,\ \omega = \lambda\sigma.\ \Diamond,\ \ldots = some|\!)$›
unfolding *det-ndet-conv-defs* **by** *simp-all*

lemma *from-det-to-ndet-σ-σs-conv-commute*:
‹$_{nd}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s} = \langle\!\langle{}_{d}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{d\hookrightarrow nd}$› ‹$_{nd}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma} = \langle\!\langle{}_{d}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{d\hookrightarrow nd}$›
by (*simp add*: *det-ndet-conv-defs σ-σs-conv-defs*, *rule ext*,
auto simp add: *option.case-eq-if split*: *if-splits*)+

lemma *from-det-to-ndet-singl-list-conv-commute*:
‹$_{nd}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{singl\hookrightarrow list} = \langle\!\langle{}_{d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{d\hookrightarrow nd}$› ‹$_{nd}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{list\rightsquigarrow singl} = \langle\!\langle{}_{d}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{d\hookrightarrow nd}$›
by (*simp add*: *det-ndet-conv-defs singl-list-conv-defs*,
solves ‹*intro conjI ext*, *auto split*: *option.split*›)+

lemma *from-ndet-to-det-σ-σs-conv-commute*:
‹*at-most-1-elem-trans A* $\Longrightarrow$ $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s} = \langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd\rightsquigarrow d}$›
‹*at-most-1-elem-trans B* $\Longrightarrow$ $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma} = \langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{nd\rightsquigarrow d}$›
proof −
assume ∗ : ‹*at-most-1-elem-trans A*›
from ∗ **have** ‹τ $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ σs $a = \tau$ $\langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd\rightsquigarrow d}$ σs a› **for** σs a
by (*auto simp add*: *det-ndet-conv-defs σ-σs-conv-defs*
elim: *at-most-1-elem-transE*)
moreover have ‹ω $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $\sigma s = \omega$ $\langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd\rightsquigarrow d}$ σs› **for** σs
by (*auto simp add*: *det-ndet-conv-defs σ-σs-conv-defs*)
moreover have ‹*more* $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *more* $\langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *simp*
ultimately show ‹$_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s} = \langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *expand-A-scheme auto*
next
assume ∗ : ‹*at-most-1-elem-trans B*›
from ∗ **have** ‹τ $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ $a = \tau$ $\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{nd\rightsquigarrow d}$ σ a› **for** σ a
by (*auto simp add*: *det-ndet-conv-defs σ-σs-conv-defs*
elim: *at-most-1-elem-transE*)
moreover have ‹ω $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ $\sigma = \omega$ $\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{nd\rightsquigarrow d}$ σ› **for** σ
by (*auto simp add*: *det-ndet-conv-defs σ-σs-conv-defs*)
moreover have ‹*more* $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = *more* $\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *simp*
ultimately show ‹$_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma} = \langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *expand-A-scheme auto*
qed

lemma *from-ndet-to-det-singl-list-conv-commute*:
‹*at-most-1-elem A* $\Longrightarrow$ $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{singl\hookrightarrow list} = \langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{nd\rightsquigarrow d}$›
‹*at-most-1-elem B* $\Longrightarrow$ $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{list\rightsquigarrow singl} = \langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{nd\rightsquigarrow d}$›
proof −
assume ∗ : ‹*at-most-1-elem A*›
from ∗ **have** ‹τ $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{singl\hookrightarrow list}$ σs $a = \tau$ $\langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{nd\rightsquigarrow d}$ σs a› **for** σs a
by (*auto simp add*: *det-ndet-conv-defs singl-list-conv-defs*
elim: *at-most-1-elemE*(*1*))
moreover from ∗ **have** ‹ω $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{singl\hookrightarrow list}$ $\sigma s = \omega$ $\langle\!\langle{}_{nd}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{nd\rightsquigarrow d}$ σs› **for** σs
by (*auto simp add*: *det-ndet-conv-defs singl-list-conv-defs*

elim: *at-most-1-elemE*(*2*))
moreover have ‹*more* $_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{singl\hookrightarrow list}$ = *more* $\langle\!\langle_{nd}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *simp*
ultimately show ‹$_{d}\langle\!\langle\langle\!\langle A\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{singl\hookrightarrow list}$ = $\langle\!\langle_{nd}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *expand-A-scheme auto*
next
assume ∗ : ‹*at-most-1-elem B*›
from ∗ **have** ‹τ $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{list\rightsquigarrow singl}$ σ a = τ $\langle\!\langle_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{nd\rightsquigarrow d}$ σ a› **for** σ a
by (*auto simp add*: *det-ndet-conv-defs singl-list-conv-defs*
elim: *at-most-1-elemE*(*1*))
moreover from ∗ **have** ‹ω $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{list\rightsquigarrow singl}$ σ = ω $\langle\!\langle_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{nd\rightsquigarrow d}$ σ› **for** σ
by (*auto simp add*: *det-ndet-conv-defs singl-list-conv-defs*
elim: *at-most-1-elemE*(*2*))
moreover have ‹*more* $_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{list\rightsquigarrow singl}$ = *more* $\langle\!\langle_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *simp*
ultimately show ‹$_{d}\langle\!\langle\langle\!\langle B\rangle\!\rangle_{nd\rightsquigarrow d}\rangle\!\rangle_{list\rightsquigarrow singl}$ = $\langle\!\langle_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{nd\rightsquigarrow d}$› **by** *expand-A-scheme auto*
qed

lemma *behaviour-σ-σs-conv*:
‹ε $_{d}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $[\sigma]$ = ε A σ›
‹τ $_{d}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $[\sigma]$ a = (*case* τ A σ a *of* $\Diamond \Rightarrow \Diamond$ | $\lfloor t\rfloor \Rightarrow \lfloor[t]\rfloor$)›
‹ϱ $_{d}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = $\{\sigma s.\ hd\ \sigma s \in \varrho\ A\}$›
‹ω $_{d}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $[\sigma]$ = ω A σ›
‹ε $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $[\sigma]$ = ε B σ›
‹τ $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $[\sigma]$ a = $\{[\sigma'] \mid \sigma'.\ \sigma' \in \tau\ B\ \sigma\ a\}$›
‹ϱ $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = $\{\sigma s.\ hd\ \sigma s \in \varrho\ B\}$›
‹ω $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ $[\sigma]$ = ω B σ›
‹ε $_{d}\langle\!\langle C\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ = ε C $[\sigma]$›
‹τ $_{d}\langle\!\langle C\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ a = (*case* τ C $[\sigma]$ a *of* $\Diamond \Rightarrow \Diamond$ | $\lfloor\sigma s'\rfloor \Rightarrow \lfloor hd\ \sigma s'\rfloor$)›
‹ϱ $_{d}\langle\!\langle C\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = $\{\sigma.\ [\sigma] \in \varrho\ C\}$›
‹ω $_{d}\langle\!\langle C\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ = ω C $[\sigma]$›
‹ε $_{nd}\langle\!\langle D\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ = ε D $[\sigma]$›
‹τ $_{nd}\langle\!\langle D\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ a = $\{hd\ \sigma s' \mid \sigma s'.\ \sigma s' \in \tau\ D\ [\sigma]\ a\}$›
‹ϱ $_{nd}\langle\!\langle D\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = $\{\sigma.\ [\sigma] \in \varrho\ D\}$› ‹$\omega$ $_{nd}\langle\!\langle D\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ σ = ω D $[\sigma]$›
by *simp-all* (*simp-all add*: *σ-σs-conv-defs*)

lemma *behaviour-singl-list-conv*:
‹ε $_{d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ $[\sigma]$ = ε A σ›
‹τ $_{d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ $[\sigma]$ a = (*case* τ A σ a *of* $\Diamond \Rightarrow \Diamond$ | $\lfloor t\rfloor \Rightarrow \lfloor[t]\rfloor$)›
‹ϱ $_{d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ = $\{\sigma s.\ hd\ \sigma s \in \varrho\ A\}$›
‹ω $_{d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ $[\sigma]$ = (*case* ω A σ *of* $\Diamond \Rightarrow \Diamond$ | $\lfloor r\rfloor \Rightarrow \lfloor[r]\rfloor$)›
‹ε $_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$ $[\sigma]$ = ε B σ›
‹τ $_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$ $[\sigma]$ a = $\{[\sigma'] \mid \sigma'.\ \sigma' \in \tau\ B\ \sigma\ a\}$›

‹$\varrho\ _{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list} = \{\sigma s.\ hd\ \sigma s \in \varrho\ B\}$›
‹$\omega\ _{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}\ [\sigma] = \{[r]\ |r.\ r \in \omega\ B\ \sigma\}$›
‹$\varepsilon\ _{d}\langle\!\langle C\rangle\!\rangle_{list\rightsquigarrow singl}\ \sigma = \varepsilon\ C\ [\sigma]$›
‹$\tau\ _{d}\langle\!\langle C\rangle\!\rangle_{list\rightsquigarrow singl}\ \sigma\ a = (case\ \tau\ C\ [\sigma]\ a\ of\ \Diamond \Rightarrow \Diamond\ |\ \lfloor\sigma s'\rfloor \Rightarrow \lfloor hd\ \sigma s'\rfloor)$›
‹$\varrho\ _{d}\langle\!\langle C\rangle\!\rangle_{list\rightsquigarrow singl} = \{\sigma.\ [\sigma] \in \varrho\ C\}$›
‹$\omega\ _{d}\langle\!\langle C\rangle\!\rangle_{list\rightsquigarrow singl}\ \sigma = (case\ \omega\ C\ [\sigma]\ of\ \Diamond \Rightarrow \Diamond\ |\ \lfloor rs\rfloor \Rightarrow \lfloor hd\ rs\rfloor)$›
‹$\varepsilon\ _{nd}\langle\!\langle D\rangle\!\rangle_{list\rightsquigarrow singl}\ \sigma = \varepsilon\ D\ [\sigma]$›
‹$\tau\ _{nd}\langle\!\langle D\rangle\!\rangle_{list\rightsquigarrow singl}\ \sigma\ a = \{hd\ \sigma s'|\ \sigma s'.\ \sigma s' \in \tau\ D\ [\sigma]\ a\}$›
‹$\varrho\ _{nd}\langle\!\langle D\rangle\!\rangle_{list\rightsquigarrow singl} = \{\sigma.\ [\sigma] \in \varrho\ D\}$›
‹$\omega\ _{nd}\langle\!\langle D\rangle\!\rangle_{list\rightsquigarrow singl}\ \sigma = \{hd\ rs\ |rs.\ rs \in \omega\ D\ [\sigma]\}$›
by *simp-all* (*simp-all add*: *singl-list-conv-defs*)

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

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

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

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

lemma *length-1-trans-from-σ-to-σs* [*simp*] :
‹*length-1-trans*$_d$ $_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$› ‹*length-1-trans*$_{nd}$ $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›
by (*rule length-1-trans*$_d$*I*, *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 $_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ [*hd σs*] *a* = τ $_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *σs a*›
‹τ $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ [*hd σs*] *a* = τ $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *σs a*›
by (*simp-all add*: *σ-σs-conv-defs*)

lemma *ω-hd-from-σ-to-σs-eq* [*simp*] :
‹ω $_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ [*hd σs*] = ω $_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *σs*›
‹ω $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ [*hd σs*] = ω $_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *σs*›
by (*simp-all add*: *σ-σs-conv-defs*)

lemma *from-σs-to-σ-from-σ-to-σs* [*simp*] :
‹$_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = *A*› ‹$_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$ = *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*; $\bigwedge$*σs a*. τ *A* [*hd σs*] *a* = τ *A σs a*;
$\bigwedge$*σs*. ω *A* [*hd σs*] = ω *A σs*⟧ $\Longrightarrow$ $_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *A*›
‹⟦*length-1-trans*$_{nd}$ *B*; $\bigwedge$*σs a*. τ *B* [*hd σs*] *a* = τ *B σs a*;
$\bigwedge$*σs*. ω *B* [*hd σs*] = ω *B σs*⟧ $\Longrightarrow$ $_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *B*›
proof −
assume ∗ : ‹*length-1-trans*$_d$ *A*› ‹$\bigwedge$*σs a*. τ *A* [*hd σs*] *a* = τ *A σs a*›
‹$\bigwedge$*σs*. ω *A* [*hd σs*] = ω *A σs*›
from ∗(*1*) **have** ‹τ $_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *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*$_d$*E*)
moreover have ‹ω $_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *σs* = ω *A σs*› **for** *σs*
by (*simp add*: *σ-σs-conv-defs* ∗(*3*))
moreover have ‹*more* $_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *more A*› **by** *simp*
ultimately show ‹$_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *A*› **by** *expand-A-scheme auto*
next
assume ∗ : ‹*length-1-trans*$_{nd}$ *B*› ‹$\bigwedge$*σs a*. τ *B* [*hd σs*] *a* = τ *B σs a*›
‹$\bigwedge$*σs*. ω *B* [*hd σs*] = ω *B σs*›
from ∗(*1*) **have** ‹τ $_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *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 $_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ *σs* = ω *B σs*› **for** *σs*
by (*simp add*: *σ-σs-conv-defs* ∗(*3*))
moreover have ‹*more* $_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *more B*› **by** *simp*
ultimately show ‹$_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$ = *B*› **by** *expand-A-scheme fastforce*

qed

theorem *bij-betw-from-σ-to-σs* :
‹*bij-betw* ($\lambda A.\ {}_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$) *UNIV*
{*A*. *length-1-trans*$_d$ *A* ∧ (∀ *σs a*. *τ A* [*hd σs*] *a* = *τ A σs a*) ∧ (∀ *σs*. *ω A* [*hd σs*] = *ω A σs*)}›
(**is** ‹*bij-betw* ($\lambda A.\ {}_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$) *UNIV* ?$S_d$›)
‹*bij-betw* ($\lambda B.\ {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$) *UNIV*
{*B*. *length-1-trans*$_{nd}$ *B* ∧ (∀ *σs a*. *τ B σs a* = *τ B* [*hd σs*] *a*) ∧ (∀ *σs*. *ω B* [*hd σs*] = *ω B σs*)}›
unfolding *bij-betw-iff-bijections*
by (*rule exI*[**where** *x* = ‹$\lambda A.\ {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$›], *simp*)
(*rule exI*[**where** *x* = ‹$\lambda A.\ {}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$›], *simp*)

lemma *bij-betw-from-σs-to-σ* :
‹*bij-betw* ($\lambda A.\ {}_d\langle\!\langle A\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$)
{*A*. *length-1-trans*$_d$ *A* ∧ (∀ *σs a*. *τ A* [*hd σs*] *a* = *τ A σs a*) ∧ (∀ *σs*. *ω A* [*hd σs*] = *ω A σs*)} *UNIV*›
‹*bij-betw* ($\lambda B.\ {}_{nd}\langle\!\langle B\rangle\!\rangle_{\sigma s\rightsquigarrow\sigma}$)
{*B*. *length-1-trans*$_{nd}$ *B* ∧ (∀ *σs a*. *τ B σs a* = *τ B* [*hd σs*] *a*) ∧ (∀ *σs*. *ω B* [*hd σs*] = *ω B σs*)} *UNIV*›
unfolding *bij-betw-iff-bijections*
by (*rule exI*[**where** *x* = ‹$\lambda A.\ {}_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›], *simp*)
(*rule exI*[**where** *x* = ‹$\lambda A.\ {}_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›], *simp*)

lemma *length-1-from-singl-to-list* [*simp*] :
‹*length-1*$_d$ ${}_d\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$› ‹*length-1*$_{nd}$ ${}_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$›
by (*rule length-1*$_d$*I*; *solves* ‹*auto simp add*: *singl-list-conv-defs split*: *option.split-asm*›)
(*rule length-1*$_{nd}$*I*; *solves* ‹*auto simp add*: *singl-list-conv-defs split*: *option.split-asm*›)

lemma *τ-hd-from-singl-to-list-eq* [*simp*] :
‹*τ* ${}_d\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ [*hd σs*] *a* = *τ* ${}_d\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ *σs a*›
‹*τ* ${}_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$ [*hd σs*] *a* = *τ* ${}_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$ *σs a*›
by (*simp-all add*: *singl-list-conv-defs*)

lemma *ω-hd-from-singl-to-list-eq* [*simp*] :
‹*ω* ${}_d\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ [*hd σs*] = *ω* ${}_d\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}$ *σs*›
‹*ω* ${}_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$ [*hd σs*] = *ω* ${}_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}$ *σs*›
by (*simp-all add*: *singl-list-conv-defs*)

lemma *from-list-to-singl-from-singl-to-list* [*simp*] :
‹${}_d\langle\!\langle {}_d\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{list\rightsquigarrow singl}$ = *A*› ‹${}_{nd}\langle\!\langle {}_{nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}\rangle\!\rangle_{list\rightsquigarrow singl}$ = *B*›
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-if set-eq-iff*; *metis list.sel*(*1*))

lemma *from-singl-to-list-from-list-to-singl* [*simp*] :
‹$[\![$*length-1*$_d$ A; $\bigwedge\sigma s\ a.\ \tau\ A\ [hd\ \sigma s]\ a = \tau\ A\ \sigma s\ a$;
$\bigwedge\sigma s.\ \omega\ A\ [hd\ \sigma s] = \omega\ A\ \sigma s]\!] \Longrightarrow {}_d\langle\!\langle{}_d\langle\!\langle A\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list} = A$›
‹$[\![$*length-1*$_{nd}$ B; $\bigwedge\sigma s\ a.\ \tau\ B\ [hd\ \sigma s]\ a = \tau\ B\ \sigma s\ a$;
$\bigwedge\sigma s.\ \omega\ B\ [hd\ \sigma s] = \omega\ B\ \sigma s]\!] \Longrightarrow {}_{nd}\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list} = B$›
proof −
assume $*$: ‹*length-1*$_d$ A› ‹$\bigwedge\sigma s\ a.\ \tau\ A\ [hd\ \sigma s]\ a = \tau\ A\ \sigma s\ a$›
‹$\bigwedge\sigma s.\ \omega\ A\ [hd\ \sigma s] = \omega\ A\ \sigma s$›
from $*(1)$ **have** ‹$\tau\ {}_d\langle\!\langle{}_d\langle\!\langle A\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list}\ \sigma s\ a = \tau\ A\ \sigma s\ a$› **for** $\sigma s\ a$
by (*auto simp add*: *singl-list-conv-defs* $*(2)$
split: *option.split elim*: *length-1*$_d E(1)$)
moreover from $*(1)$ **have** ‹$\omega\ {}_d\langle\!\langle{}_d\langle\!\langle A\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list}\ \sigma s = \omega\ A\ \sigma s$› **for** σs
by (*auto simp add*: *singl-list-conv-defs* $*(3)$
split: *option.split elim*: *length-1*$_d E(2)$)
moreover have ‹*more* ${}_d\langle\!\langle{}_d\langle\!\langle A\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list}$ = *more* A› **by** *simp*
ultimately show ‹${}_d\langle\!\langle{}_d\langle\!\langle A\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list} = A$› **by** *expand-A-scheme auto*
next
assume $*$: ‹*length-1*$_{nd}$ B› ‹$\bigwedge\sigma s\ a.\ \tau\ B\ [hd\ \sigma s]\ a = \tau\ B\ \sigma s\ a$›
‹$\bigwedge\sigma s.\ \omega\ B\ [hd\ \sigma s] = \omega\ B\ \sigma s$›
from $*(1)$ **have** ‹$\tau\ {}_{nd}\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list}\ \sigma s\ a = \tau\ B\ \sigma s\ a$› **for** $\sigma s\ a$
by (*auto simp add*: *singl-list-conv-defs* $*(2)$ *elim*: *length-1*$_{nd} E(1)$)
(*metis length-1*$_{nd} E(1)$ *list.sel*(*1*))
moreover from $*(1)$ **have** ‹$\omega\ {}_{nd}\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list}\ \sigma s = \omega\ B\ \sigma s$› **for** σs
by (*auto simp add*: *singl-list-conv-defs* $*(3)$ *elim*: *length-1*$_{nd} E(2)$)
(*metis length-1*$_{nd} E(2)$ *list.sel*(*1*))
moreover have ‹*more* ${}_{nd}\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list}$ = *more* B› **by** *simp*
ultimately show ‹${}_{nd}\langle\!\langle{}_{nd}\langle\!\langle B\rangle\!\rangle_{list\rightsquigarrow singl}\rangle\!\rangle_{singl\hookrightarrow list} = B$› **by** *expand-A-scheme fastforce*
qed

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

lemma *bij-betw-from-list-to-singl* :
‹*bij-betw* ($\lambda A.\ {}_d\langle\!\langle A\rangle\!\rangle_{list\rightsquigarrow singl}$)

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

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

lemma $\mathcal{R}$*-base-trans*[*simp*]: ‹$\mathcal{R}_d$ $(\!|\tau = \lambda\sigma\ a.\ \Diamond, \omega = \lambda\sigma.\ \Diamond, \ldots = some|\!) = (\lambda\sigma.\ \{\sigma\})$›
‹$\mathcal{R}_{nd}$ $(\!|\tau = \lambda\sigma\ a.\ \{\}, \omega = \lambda\sigma.\ \{\}, \ldots = some|\!) = (\lambda\sigma.\ \{\sigma\})$›
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* : ‹$\mathcal{R}_{nd}\ \langle\!\langle A\rangle\!\rangle_{d \hookrightarrow nd}\ \sigma = \mathcal{R}_d\ A\ \sigma$›
proof *safe*
show ‹$\sigma' \in \mathcal{R}_{nd}\ \langle\!\langle A\rangle\!\rangle_{d \hookrightarrow nd}\ \sigma \Longrightarrow \sigma' \in \mathcal{R}_d\ A\ \sigma$› **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 ‹$\sigma' \in \mathcal{R}_d\ A\ \sigma \Longrightarrow \sigma' \in \mathcal{R}_{nd}\ \langle\!\langle A\rangle\!\rangle_{d \hookrightarrow nd}\ \sigma$› **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 : ‹*bij-betw* f ($\mathcal{R}_{nd}\ B_0\ \sigma_0$) ($\mathcal{R}_{nd}\ B_1\ (f\ \sigma_0)$)›
if ‹*inj-on* f ($\mathcal{R}_{nd}\ B_0\ \sigma_0$)› **and** ‹$\bigwedge \sigma_0'\ a.\ \sigma_0' \in \mathcal{R}_{nd}\ B_0\ \sigma_0 \Longrightarrow \tau\ B_1\ (f\ \sigma_0')\ a = f\ `\ \tau\ B_0\ \sigma_0'\ a$›
proof (*rule bij-betw-imageI*, *fact that*(*1*), *auto simp add*: *image-def*, *goal-cases*)
show ‹$s \in \mathcal{R}_{nd}\ B_0\ \sigma_0 \Longrightarrow f\ s \in \mathcal{R}_{nd}\ B_1\ (f\ \sigma_0)$› **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 ‹$s \in \mathcal{R}_{nd}\ B_1\ (f\ \sigma_0) \Longrightarrow \exists\, t \in \mathcal{R}_{nd}\ B_0\ \sigma_0.\ s = f\ t$› **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: ‹*bij-betw* f ($\mathcal{R}_d\ A_0\ \sigma_0$) ($\mathcal{R}_d\ A_1\ (f\ \sigma_0)$)›
if ‹*inj-on* f ($\mathcal{R}_d\ A_0\ \sigma_0$)› **and** ‹$\bigwedge \sigma_0'\ a.\ \sigma_0' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow \tau\ A_1\ (f\ \sigma_0')\ a = map\text{-}option\ f\ (\tau\ A_0\ \sigma_0'\ a)$›
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-$\mathcal{R}_{nd}$* = *bij-betw-$\mathcal{R}_{nd}$-if-same-τ*[**where** *f* = *id*, *simplified bij-betw-def*, *simplified*]
and *same-τ-implies-same-$\mathcal{R}_d$* = *bij-betw-$\mathcal{R}_d$-if-same-τ*[**where** *f* = *id*, *simplified bij-betw-def option.map-id*, *simplified*]

corollary *$\mathcal{R}_d$-ω-useless* [*simp*] : ‹$\mathcal{R}_d$ (A(|ω := *some-ω*|)) σ = $\mathcal{R}_d$ A σ›
by (*auto intro*!: *same-τ-implies-same-$\mathcal{R}_d$*)

corollary *$\mathcal{R}_{nd}$-ω-useless* [*simp*] : ‹$\mathcal{R}_{nd}$ (A(|ω := *some-ω*|)) σ = $\mathcal{R}_{nd}$ A σ›
by (*auto intro*!: *same-τ-implies-same-$\mathcal{R}_{nd}$*)

corollary *indep-enabl-ω-useless* [*simp*] :
‹*indep-enabl* (A_0(|ω := *some-ω*|)) σ_0 E A_1 σ_1 $\longleftrightarrow$ *indep-enabl* A_0 σ_0 E A_1 σ_1›
‹*indep-enabl* A_0 σ_0 E (A_1(|ω := *some-ω*|)) σ_1 $\longleftrightarrow$ *indep-enabl* A_0 σ_0 E A_1 σ_1›
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-σs-description*: ‹$\mathcal{R}_{nd}$ $_{nd}$⟪B⟫$_{\sigma\hookrightarrow\sigma s}$ [σ] = {[σ']| σ'. $\sigma' \in \mathcal{R}_{nd}$ B σ}›
proof *safe*
show ‹$\sigma s \in \mathcal{R}_{nd}$ $_{nd}$⟪B⟫$_{\sigma\hookrightarrow\sigma s}$ [σ] $\Longrightarrow$ $\exists\,\sigma'$. σs = [σ'] $\wedge$ $\sigma' \in \mathcal{R}_{nd}$ B σ› **for** σs
by (*induct rule*: *$\mathcal{R}_{nd}$.induct*, *auto simp add*: *$\mathcal{R}_{nd}$.init behaviour-σ-σs-conv*(*6*), *metis $\mathcal{R}_{nd}$.step*)
next
show ‹$\sigma' \in \mathcal{R}_{nd}$ B σ $\Longrightarrow$ [σ'] $\in \mathcal{R}_{nd}$ $_{nd}$⟪B⟫$_{\sigma\hookrightarrow\sigma s}$ [σ]› **for** σ'
by (*induct rule*: *$\mathcal{R}_{nd}$.induct*) (*simp-all add*: *$\mathcal{R}_{nd}$.init $\mathcal{R}_{nd}$.step behaviour-σ-σs-conv*(*6*))
qed

lemma *$\mathcal{R}_d$-from-σ-to-σs-description*: ‹$\mathcal{R}_d$ $_d$⟪A⟫$_{\sigma\hookrightarrow\sigma s}$ [σ] = {[σ']| σ'. $\sigma' \in \mathcal{R}_d$ A σ}›
by (*simp add*: *$\mathcal{R}_{nd}$-from-σ-to-σs-description*
flip: *$\mathcal{R}_{nd}$-from-det-to-ndet from-det-to-ndet-σ-σs-conv-commute*(*1*))

lemma $\mathcal{R}_{nd}$*-from-singl-to-list-description*: ‹$\mathcal{R}_{nd\ nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}\ [\sigma] = \{[\sigma']|\ \sigma'.\ \sigma' \in \mathcal{R}_{nd}\ B\ \sigma\}$›
proof *safe*
show ‹$\sigma s \in \mathcal{R}_{nd\ nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}\ [\sigma] \Longrightarrow \exists\,\sigma'.\ \sigma s = [\sigma'] \wedge \sigma' \in \mathcal{R}_{nd}\ B\ \sigma$› **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 ‹$\sigma' \in \mathcal{R}_{nd}\ B\ \sigma \Longrightarrow [\sigma'] \in \mathcal{R}_{nd\ nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}\ [\sigma]$› **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*: ‹$\mathcal{R}_{d\ d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\ [\sigma] = \{[\sigma']|\ \sigma'.\ \sigma' \in \mathcal{R}_{d}\ A\ \sigma\}$›
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:
‹$\sigma s' \in \mathcal{R}_{d\ d}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\ \sigma s \Longrightarrow \sigma s' = \sigma s \vee length\ \sigma s' = 1$›
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:
‹$\sigma s' \in \mathcal{R}_{nd\ nd}\langle\!\langle B\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\ \sigma s \Longrightarrow \sigma s' = \sigma s \vee length\ \sigma s' = 1$›
by (*simp add*: σ*-*σs*-conv-defs*)
(*induct rule*: $\mathcal{R}_{nd}$*.induct*, *auto*)

lemma *length-*$\mathcal{R}_{d}$*-from-singl-to-list*:
‹$\sigma s' \in \mathcal{R}_{d\ d}\langle\!\langle A\rangle\!\rangle_{singl\hookrightarrow list}\ \sigma s \Longrightarrow \sigma s' = \sigma s \vee length\ \sigma s' = 1$›
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*:
‹$\sigma s' \in \mathcal{R}_{nd\ nd}\langle\!\langle B\rangle\!\rangle_{singl\hookrightarrow list}\ \sigma s \Longrightarrow \sigma s' = \sigma s \vee length\ \sigma s' = 1$›
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* :: ‹$[('\sigma,\ 'a)\ enabl,\ ('\sigma,\ 'a)\ trans_{nd},\ '\sigma \Rightarrow ('a,\ 'r)\ process_{ptick},\ '\sigma] \Rightarrow ('a,\ 'r)\ process_{ptick}$›
where ‹*P-nd-step* $\varepsilon_A\ \tau_A\ X\ \sigma \equiv \Box\ e \in \varepsilon_A\ \sigma \rightarrow \sqcap\ \sigma' \in \tau_A\ \sigma\ e.\ X\ \sigma'$›

definition *P-nd* :: ‹$('\sigma,\ 'a,\ 'r,\ '\alpha)\ A_{nd}$-*scheme* $\Rightarrow$ $'\sigma$ $\Rightarrow$ $('a,\ 'r)$ *process*$_{ptick}$› (‹$P\langle\!\langle$-$\rangle\!\rangle_{nd}$› *1000*)
where ‹$P\langle\!\langle A\rangle\!\rangle_{nd} \equiv \upsilon\ X.$ *P-nd-step* $(\varepsilon\ A)\ (\tau\ A)\ X$›

lemma *P-nd-step-constructive* [*simp*] : ‹*constructive* (*P-nd-step* $\varepsilon_A\ \tau_A$)› **by** *simp*

lemma *P-nd-step-cont* [*simp*] : ‹$\forall\sigma\ a.$ *finite* $(\tau_A\ \sigma\ a) \Longrightarrow$ *cont* (*P-nd-step* $\varepsilon_A\ \tau_A$)›
by (*simp add*: *cont-fun*)

lemma *P-nd-step-constructive-bis* : ‹*constructive* (*P-nd-step* $(\varepsilon\ A)\ (\tau\ A)$)› **by** *simp*

lemma *P-nd-step-cont-bis* [*simp*] : ‹*finite-trans* $A \Longrightarrow$ *cont* (*P-nd-step* $(\varepsilon\ A)\ (\tau\ A)$)›
by (*simp add*: *finite-trans-def*)

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

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

lemma *non-destructive-imp-restriction-cont* [*simp*] :
‹*non-destructive* $f \Longrightarrow$ *restriction-cont* f›
by (*simp add*: *non-destructive-on-def*)

lemma *P-nd-ω-useless*: ‹$P\langle\!\langle A\rangle\!\rangle_{nd} = P\langle\!\langle A(\!|\omega :=$ *some-ω*$|\!)\rangle\!\rangle_{nd}$›
by (*simp add*: *P-nd-def ε-simps*)

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

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

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

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

abbreviation P_{SKIPS}*-nd-step* ::
‹[('σ, 'a) *enabl*, ('σ, 'a) $trans_{nd}$, 'σ ⇒ 'r *set*, 'σ ⇒ ('a, 'r) $process_{ptick}$, 'σ] ⇒ ('a, 'r) $process_{ptick}$›
where ‹P_{SKIPS}*-nd-step* ε_A τ_A ω_A *X* σ ≡ *if* ω_A σ = {} *then P-nd-step* ε_A τ_A *X* σ *else SKIPS* (ω_A σ)›

definition P_{SKIPS}*-nd* :: ‹('σ, 'a, 'r, 'α) A_{nd}*-scheme* ⇒ 'σ ⇒ ('a, 'r) $process_{ptick}$› (‹P_{SKIPS}⟪-⟫$_{nd}$› *1000*)
where ‹P_{SKIPS}⟪*A*⟫$_{nd}$ ≡ υ *X*. P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) *X*›

lemma P_{SKIPS}*-nd-step-constructive* [*simp*] : ‹*constructive* (P_{SKIPS}*-nd-step* ε_A τ_A ω_A)› **by** *auto*

lemma P_{SKIPS}*-nd-step-cont* [*simp*] : ‹∀ σ *a*. *finite* (τ_A σ *a*) ⟹ *cont* (P_{SKIPS}*-nd-step* ε_A τ_A ω_A)›
by (*simp add*: *cont-fun*)

lemma P_{SKIPS}*-nd-step-constructive-bis* : ‹*constructive* (P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*))› **by** *simp*

lemma P_{SKIPS}*-nd-step-cont-bis* [*simp*] : ‹*finite-trans A* ⟹ *cont* (P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*))›
by (*simp add*: *finite-trans-def*)

lemma P_{SKIPS}*-nd-rec*: ‹P_{SKIPS}⟪*A*⟫$_{nd}$ = (λσ. P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) P_{SKIPS}⟪*A*⟫$_{nd}$ σ)›
by (*unfold* P_{SKIPS}*-nd-def*, *rule ext*, *subst restriction-fix-eq*, *simp-all*)

lemma P_{SKIPS}*-nd-is-fix* : ‹*finite-trans A* ⟹ P_{SKIPS}⟪*A*⟫$_{nd}$ = (μ *X*. P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) *X*)›
by (*simp add*: P_{SKIPS}*-nd-def restriction-fix-is-fix*)

lemma P_{SKIPS}*-nd-induct* [*case-names adm base step*] :
‹$adm_{\downarrow}$ *P* ⟹ *P* σ ⟹ (⋀*X*. *P X* ⟹ *P* (P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) *X*)) ⟹ *P* P_{SKIPS}⟪*A*⟫$_{nd}$›
unfolding P_{SKIPS}*-nd-def*
by (*rule restriction-fix-ind*[*OF* P_{SKIPS}*-nd-step-constructive-bis*]) *simp-all*

lemma P_{SKIPS}*-nd-induct-iterated* [*consumes 1*, *case-names adm base step*] :
‹⟦*0* < *k*; $adm_{\downarrow}$ *P*; *P* σ; ⋀*X*. *P X* ⟹ *P* ((P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) ^^ *k*) *X*)⟧ ⟹ *P* P_{SKIPS}⟪*A*⟫$_{nd}$›
unfolding P_{SKIPS}*-nd-def*
by (*rule restriction-fix-ind-iterated*[**where** *f* = ‹P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*)›]) *auto*

Correspondence when we always have ω *A* σ = {}.

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

lemma P_{SKIPS}*-nd-updated-*ω: ‹$P\langle\!\langle A\rangle\!\rangle_{nd} = P_{SKIPS}\langle\!\langle A(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}$›
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-*$\mathcal{R}_{nd}$:
‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$› **if** ‹$\varrho\ A \cap \mathcal{R}_{nd}\ A\ \sigma = \{\}$›
proof −
have ‹$\sigma' \in \mathcal{R}_{nd}\ A\ \sigma \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma' = P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma'$› **for** σ'
proof (*induct A arbitrary*: σ' *rule*: P_{SKIPS}*-nd-induct*)
case *adm* **show** *?case* **by** *simp*
next
case *base* **show** ‹$P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma' = P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma'$› **..**
next
case (*step X*)
from *step.prems*(*1*) *that* **have** ‹$\sigma' \notin \varrho\ A$› **by** *blast*
hence ‹$\omega\ A\ \sigma' = \{\}$› **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.prems*)
qed
thus ‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$› **by** (*simp add*: $\mathcal{R}_{nd}$*.init*)
qed

lemma P_{SKIPS}*-nd-rec-notin-*ϱ:
‹$\sigma \notin \varrho\ A \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = P\text{-}nd\text{-}step\ (\varepsilon\ A)\ (\tau\ A)\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
by (*subst* P_{SKIPS}*-nd-rec*) (*simp add*: ϱ*-simps*)

lemma P_{SKIPS}*-nd-rec-in-*ϱ: ‹$\sigma \in \varrho\ A \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = SKIPS\ (\omega\ A\ \sigma)$›
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* :: ‹$[('\sigma, 'a)\ enabl, ('\sigma, 'a)\ trans_d, '\sigma \Rightarrow ('a, 'r)\ process_{ptick}, '\sigma] \Rightarrow ('a, 'r)\ process_{ptick}$›
where ‹$P\text{-}d\text{-}step\ \varepsilon_A\ \tau_A\ X\ s \equiv \Box\ e \in \varepsilon_A\ s \rightarrow X\ \lceil\tau_A\ s\ e\rceil$›

definition *P-d* :: ‹$('\sigma, 'a, 'r, '\alpha)\ A_d\text{-}scheme \Rightarrow '\sigma \Rightarrow ('a, 'r)\ process_{ptick}$› (‹$P\langle\!\langle\text{-}\rangle\!\rangle_d$› *1000*)
where ‹$P\langle\!\langle A\rangle\!\rangle_d \equiv \upsilon\ X.\ P\text{-}d\text{-}step\ (\varepsilon\ A)\ (\tau\ A)\ X$›

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

lemmas *P-d-step-constructive-bis* = *P-d-step-constructive*[*of* ‹ε A› ‹τ A›] **for** A

lemma *P-d-step-cont*[*simp*]: ‹*cont* (*P-d-step* ε_A τ_A)›
by (*simp add*: *cont-fun*)

lemmas *P-d-step-cont-bis* = *P-d-step-cont*[*of* ‹ε A› ‹τ A›] **for** A

lemma *P-d-rec*: ‹$P\langle\!\langle A\rangle\!\rangle_d$ = (λs. *P-d-step* (ε A) (τ A) $P\langle\!\langle A\rangle\!\rangle_d$ s)›
by (*unfold P-d-def*, *subst restriction-fix-eq*) *simp-all*

lemma *P-d-is-fix* : ‹$P\langle\!\langle A\rangle\!\rangle_d$ = (μ X. *P-d-step* (ε A) (τ A) X)›
by (*simp add*: *P-d-def restriction-fix-is-fix*)

lemma *P-d-ω-useless*: ‹$P\langle\!\langle A\rangle\!\rangle_d$ = $P\langle\!\langle A(\!|\omega$:= *some-ω*$|\!)\rangle\!\rangle_d$›
by (*simp add*: *P-d-def ε-simps*)

lemma *P-d-ω-useless-bis*: ‹$P\langle\!\langle A\rangle\!\rangle_d$ = $P\langle\!\langle A(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d$›
by (*fact P-d-ω-useless*)

lemma *P-d-induct* [*case-names adm base step*] :
‹$[\![adm_\downarrow\ P;\ P\ \sigma;\ \bigwedge X.\ P\ X \Longrightarrow P$ (*P-d-step* (ε A) (τ A) X)$]\!] \Longrightarrow P\ P\langle\!\langle A\rangle\!\rangle_d$›
unfolding *P-d-def*
by (*rule restriction-fix-ind*[*OF P-d-step-constructive-bis*]) *simp-all*

lemma *P-d-induct-iterated* [*consumes 1*, *case-names adm base step*] :
‹$[\![0 < k;\ adm_\downarrow\ P;\ P\ \sigma;\ \bigwedge X.\ P\ X \Longrightarrow P$ ((*P-d-step* (ε A) (τ A) $^{\frown\frown}$ k) X)$]\!] \Longrightarrow P\ P\langle\!\langle A\rangle\!\rangle_d$›
unfolding *P-d-def*
by (*rule restriction-fix-ind-iterated*[**where** f = ‹*P-d-step* (ε A) (τ A)›]) *auto*

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

abbreviation P_{SKIPS}*-d-step* ::
‹[($'\sigma$, $'a$) *enabl*, ($'\sigma$, $'a$) $trans_d$, $'\sigma \Rightarrow 'r$ *option*, $'\sigma \Rightarrow$ ($'a$, $'r$) $process_{ptick}$, $'\sigma$] $\Rightarrow$ ($'a$, $'r$) $process_{ptick}$›
where ‹P_{SKIPS}*-d-step* ε_A τ_A ω_A X σ $\equiv$ *case* ω_A σ *of* $\lfloor r\rfloor \Rightarrow$ *SKIP* r | $\Diamond \Rightarrow$ *P-d-step* ε_A τ_A X σ›

definition P_{SKIPS}*-d* :: ‹($'\sigma$, $'a$, $'r$, $'\alpha$) A_d*-scheme* $\Rightarrow$ $'\sigma \Rightarrow$ ($'a$, $'r$) $process_{ptick}$› (‹$P_{SKIPS}\langle\!\langle$-$\rangle\!\rangle_d$› *1000*)
where ‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d \equiv \upsilon$ X. P_{SKIPS}*-d-step* (ε A) (τ A) (ω A) X›

lemma P_{SKIPS}*-d-step-constructive*[*simp*]: ‹*constructive* (P_{SKIPS}*-d-step* ε_A τ_A $\mathcal{S}_{FA}$)›
by (*auto simp add*: *option.case-eq-if*)

lemmas P_{SKIPS}*-d-step-constructive-bis* = P_{SKIPS}*-d-step-constructive*[*of* ‹ε A› ‹τ A› ‹ω A›] **for** A

lemma P_{SKIPS}-*d-step-cont*[*simp*]: ‹*cont* (P_{SKIPS}-*d-step* ε_A τ_A $\mathcal{S}_{FA}$)›
by (*simp add*: *cont-fun option.case-eq-if*)

lemmas P_{SKIPS}-*d-step-cont-bis* = P_{SKIPS}-*d-step-cont*[*of* ‹ε A› ‹τ A› ‹ω A›]
for A

lemma P_{SKIPS}-*d-rec*: ‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$ = ($\lambda\sigma$. P_{SKIPS}-*d-step* (ε A) (τ A) (ω A) $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$ σ)›
by (*unfold* P_{SKIPS}-*d-def*, *subst restriction-fix-eq*) *auto*

lemma P_{SKIPS}-*d-is-fix* : ‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$ = (μ X. P_{SKIPS}-*d-step* (ε A) (τ A) (ω A) X)›
by (*simp add*: P_{SKIPS}-*d-def restriction-fix-is-fix*)

lemma P_{SKIPS}-*d-induct* [*case-names adm base step*] :
‹$adm_\downarrow$ P $\Longrightarrow$ P σ $\Longrightarrow$ ($\bigwedge X$. P X $\Longrightarrow$ P (P_{SKIPS}-*d-step* (ε A) (τ A) (ω A) X)) $\Longrightarrow$ P $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$›
unfolding P_{SKIPS}-*d-def*
by (*rule restriction-fix-ind*[*OF* P_{SKIPS}-*d-step-constructive-bis*]) *simp-all*

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

Correspondence when we always have ω A σ = {}.

lemma P_{SKIPS}-*d-empty-*ϱ : ‹ϱ A = {} $\Longrightarrow$ $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$ = $P\langle\!\langle A\rangle\!\rangle_d$›
by (*simp add*: ϱ-*simps P-d-def* P_{SKIPS}-*d-def*)

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

lemma P_{SKIPS}-*d-empty-*ϱ*-inter-*$\mathcal{R}_d$:
‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$ σ = $P\langle\!\langle A\rangle\!\rangle_d$ σ› **if** ‹ϱ A $\cap$ $\mathcal{R}_d$ A σ = {}›
proof −
have ‹$\sigma' \in \mathcal{R}_d$ A σ $\Longrightarrow$ $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d$ σ' = $P\langle\!\langle A\rangle\!\rangle_d$ σ'› **for** σ'
proof (*induct A arbitrary*: σ' *rule*: P_{SKIPS}-*d-induct*)
case *adm* **show** *?case* **by** *simp*
next
case *base* **show** ‹$P\langle\!\langle A\rangle\!\rangle_d$ σ' = $P\langle\!\langle A\rangle\!\rangle_d$ σ'› **..**
next
case (*step X*)
from *step.prems*(*1*) *that* **have** ‹$\sigma' \notin \varrho$ A› **by** *blast*
hence ‹ω A σ' = $\Diamond$› **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.prems*)
qed
thus ‹$P_{SKIPS}\langle\langle A\rangle\rangle_d\ \sigma = P\langle\langle A\rangle\rangle_d\ \sigma$› **by** (*simp add*: $\mathcal{R}_d$*.init*)
qed

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

lemma P_{SKIPS}*-d-rec-in-ϱ*: ‹$\sigma \in \varrho\ A \Longrightarrow P_{SKIPS}\langle\langle A\rangle\rangle_d\ \sigma = SKIP\ \lceil\omega\ A\ \sigma\rceil$›
by (*subst* P_{SKIPS}*-d-rec*, *simp add*: *ϱ-simps split*: *option.split*)

3.3.3 Link between deterministic and non-deterministic ProcOmata

lemma P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d* : ‹$P_{SKIPS}\langle\langle\langle A\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd} = P_{SKIPS}\langle\langle A\rangle\rangle_d$›
proof (*subst* P_{SKIPS}*-nd-def*, *rule restriction-fix-unique*)
show ‹*constructive* (P_{SKIPS}*-nd-step* $(\varepsilon\ \langle\langle A\rangle\rangle_{d\hookrightarrow nd})\ (\tau\ \langle\langle A\rangle\rangle_{d\hookrightarrow nd})\ (\omega\ \langle\langle A\rangle\rangle_{d\hookrightarrow nd})$)›
by *simp*
next
show ‹P_{SKIPS}*-nd-step* $(\varepsilon\ \langle\langle A\rangle\rangle_{d\hookrightarrow nd})\ (\tau\ \langle\langle A\rangle\rangle_{d\hookrightarrow nd})\ (\omega\ \langle\langle A\rangle\rangle_{d\hookrightarrow nd})\ P_{SKIPS}\langle\langle A\rangle\rangle_d$
$= P_{SKIPS}\langle\langle A\rangle\rangle_d$›
by (*subst* (*3*) P_{SKIPS}*-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* : ‹$P\langle\langle\langle A\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd} = P\langle\langle A\rangle\rangle_d$›
proof −
have ‹$P\langle\langle\langle A\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd} = P_{SKIPS}\langle\langle\langle A\rangle\rangle_{d\hookrightarrow nd}(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\rangle_{nd}$›
by (*fact* P_{SKIPS}*-nd-updated-ω*)
also have ‹$\langle\langle A\rangle\rangle_{d\hookrightarrow nd}(\!|\omega := \lambda\sigma.\ \{\}|\!) = \langle\langle A(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle_{d\hookrightarrow nd}$›
by (*simp add*: *from-det-to-ndet-def*)
finally show ‹$P\langle\langle\langle A\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd} = P\langle\langle A\rangle\rangle_d$›
by (*simp add*: P_{SKIPS}*-d-updated-ω* P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*)
qed

3.3.4 Prove Equality between ProcOmata

This is the easiest method we can think about.

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

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

by (*simp add*: *P-nd-def* ε*-simps*)

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

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

We establish now a much more powerful theorem.

theorem P_{SKIPS}*-nd-eqI-strong*:

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

theorem *P-nd-eqI-strong*:
‹$[\![$*inj-on f* $(\mathcal{R}_{nd}\ A_0\ \sigma_0)$;
$\bigwedge\sigma_0'\ a.\ \sigma_0' \in \mathcal{R}_{nd}\ A_0\ \sigma_0 \Longrightarrow \tau\ A_1\ (f\ \sigma_0')\ a = f\ `\ (\tau\ A_0\ \sigma_0'\ a)]\!]$

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

theorem P_{SKIPS}*-d-eqI-strong*:
assumes ‹*inj-on f* $(\mathcal{R}_d\ A_0\ \sigma_0)$›
and ‹$\bigwedge\sigma_0{}'\ a.\ \sigma_0{}' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow \tau\ A_1\ (f\ \sigma_0{}')\ a = \mathit{map\text{-}option}\ f\ (\tau\ A_0\ \sigma_0{}'\ a)$›
and ‹$\bigwedge\sigma_0{}'.\ \sigma_0{}' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow \omega\ A_1\ (f\ \sigma_0{}') = \omega\ A_0\ \sigma_0{}'$›
shows ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0 = P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ (f\ \sigma_0)$›
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*:
‹$[\![$*inj-on f* $(\mathcal{R}_d\ A_0\ \sigma_0)$;
$\bigwedge\sigma_0{}'\ a.\ \sigma_0{}' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow \tau\ A_1\ (f\ \sigma_0{}')\ a = \mathit{map\text{-}option}\ f\ (\tau\ A_0\ \sigma_0{}'\ a)]\!]$
$\Longrightarrow P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0 = P\langle\!\langle A_1\rangle\!\rangle_d\ (f\ \sigma_0)$›
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* : ‹$P_{SKIPS}\langle\!\langle_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd}\ [\sigma] = P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
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* : ‹$P_{SKIPS}\langle\!\langle_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_d\ [\sigma] = P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
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* : ‹$P\langle\!\langle_{nd}\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_{nd}\ [\sigma] = P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
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* : ‹$P\langle\!\langle_d\langle\!\langle A\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\rangle\!\rangle_d\ [\sigma] = P\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
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*$_d$ :: ‹[('σ × 'a) ⇒ 'σ *option*, (('σ × 'a) × 'σ *option*) *list*] ⇒ ('σ × 'a) ⇒ 'σ *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*$_d$ :: ‹[(('σ × 'a) × 'σ *option*) *list*, 'σ ⇒ 'r *option*] ⇒ ('σ, 'a, 'r) A_d›
where ‹*recursive-constructor-A*$_d$ $\mathcal{G}_A$ ω_A ≡ ⦇τ = *curry* (*recursive-modifier-fun*$_d$ (λ(s, e). ◊) $\mathcal{G}_A$), ω = ω_A⦈›

lemma *ε-det-breaker*:
‹ε (⦇τ = *curry* (g((σ'::'σ, a) ↦ σ''::'σ)), ω = *some-ω*, ... = *some-more*⦈) σ =
(*if* σ = σ' *then* {a} ∪ ε ⦇τ = *curry* g, ω = *some-ω*⦈ σ' *else* ε ⦇τ = *curry* g, ω = *some-ω*⦈ 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 :: ‹('σ, 'a, 'r, 'α) A_{nd}*-scheme*› **and** f :: ‹'a ⇒ 'b› **and** g :: ‹'r ⇒ 's›
assumes ‹*bij* f›
defines *B-def* : ‹B ≡ ⦇τ = $\lambda\sigma$ b. τ A σ (*inv* f b), ω = $\lambda\sigma$. g ‘ (ω A σ)⦈›
shows ‹*Renaming* (P_{SKIPS}⟪A⟫$_{nd}$ σ) f g = P_{SKIPS}⟪B⟫$_{nd}$ σ› (**is** ‹*?lhs* σ = -›)
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* (ε B) (τ B) (ω B) *?lhs* = *?lhs*›
proof (*rule ext*)
have ∗ : ‹ε B σ = f ‘ ε A σ› **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* (ε B) (τ B) (ω B) *?lhs* σ = *?lhs* σ› **for** σ

by (*subst* (*2*) P_{SKIPS}*-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_{SKIPS}*-d* :
‹*bij f* $\Longrightarrow$ *Renaming* ($P_{SKIPS}\langle\langle A\rangle\rangle_d$ σ) *f g* =
$P_{SKIPS}\langle\langle(\!|\tau = \lambda\sigma\ b.\ \tau\ A\ \sigma\ (inv\ f\ b),\ \omega = \lambda\sigma.\ map\text{-}option\ g\ (\omega\ A\ \sigma)|\!)\rangle\rangle_d$
σ›
by (*subst* (*1 2*) P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*[*symmetric*],
subst bij-Renaming-P_{SKIPS}*-nd*, *assumption*)
(*rule fun-cong*[*of - -* σ], *rule* P_{SKIPS}*-nd-eqI*,
simp-all add: *from-det-to-ndet-def split*: *option.split*)

lemma *RenamingTick-*P_{SKIPS}*-nd* :
‹*RenamingTick* ($P_{SKIPS}\langle\langle A\rangle\rangle_{nd}$ σ) g = $P_{SKIPS}\langle\langle(\!|\tau = \tau\ A,\ \omega = \lambda\sigma.\ g\ `\ \omega\ A\ \sigma|\!)\rangle\rangle_{nd}$ σ›
by (*simp add*: *bij-Renaming-*P_{SKIPS}*-nd*)

lemma *RenamingTick-*P_{SKIPS}*-d* :
‹*RenamingTick* ($P_{SKIPS}\langle\langle A\rangle\rangle_d$ σ) g = $P_{SKIPS}\langle\langle(\!|\tau = \tau\ A,\ \omega = \lambda\sigma.\ map\text{-}option\ g\ (\omega\ A\ \sigma)|\!)\rangle\rangle_d$ σ›
by (*simp add*: *bij-Renaming-*P_{SKIPS}*-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* :: ‹$'a$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set*›
where ‹*combine-sets-Sync* S_0 E S_1 $\equiv$ $(S_0 - E - S_1) \cup (S_1 - E - S_0) \cup (S_0 \cap S_1 - E) \cup S_0 \cap S_1 \cap E$›

definition *combine-Sync-ε* ::
‹[($'\sigma_0$, $'e$, $'\sigma_0{}'$, $'r_0$, $'\alpha_0$) *A-scheme*, $'e$ *set*,
($'\sigma_1$, $'e$, $'\sigma_1{}'$, $'r_1$, $'\alpha_1$) *A-scheme*, $'\sigma \Rightarrow '\sigma_0$, $'\sigma \Rightarrow '\sigma_1$, $'\sigma$] $\Rightarrow$ $'e$ *set*›
where ‹*combine-Sync-ε* A_0 E A_1 i_0 i_1 σs $\equiv$ *combine-sets-Sync* (ε A_0 (i_0 σs)) E (ε A_1 (i_1 σs))›

lemma *combine-Sync-ε-def-bis* :
‹*combine-Sync-ε* A_0 E A_1 i_0 i_1 σs =
ε A_0 (i_0 σs) $\cup$ ε A_1 (i_1 σs) $-$ E $\cup$ ε A_0 (i_0 σs) $\cap$ ε A_1 (i_1 σs)›
by (*auto simp add*: *combine-Sync-ε-def*)

fun *combine$_d$-Sync-τ* ::
‹[($'\sigma_0$, $'e$, $'r_0$, $'\alpha_0$) A_d*-scheme*, $'e$ *set*, ($'\sigma_1$, $'e$, $'r_1$, $'\alpha_1$) A_d*-scheme*,
$'\sigma \Rightarrow '\sigma_0$, $'\sigma \Rightarrow '\sigma_1$, $'\sigma_0 \Rightarrow '\sigma_1 \Rightarrow '\sigma$] $\Rightarrow$ ($'\sigma$, $'e$) *trans$_d$*›
and *combine$_{nd}$-Sync-τ* ::
‹[($'\sigma_0$, $'e$, $'r_0$, $'\alpha_0$) A_{nd}*-scheme*, $'e$ *set*, ($'\sigma_1$, $'e$, $'r_1$, $'\alpha_1$) A_{nd}*-scheme*,
$'\sigma \Rightarrow '\sigma_0$, $'\sigma \Rightarrow '\sigma_1$, $'\sigma_0 \Rightarrow '\sigma_1 \Rightarrow '\sigma$] $\Rightarrow$ ($'\sigma$, $'e$) *trans$_{nd}$*›
where ‹*combine$_d$-Sync-τ* A_0 E A_1 i_0 i_1 f σs e =
(*if* $e \in \varepsilon$ A_0 (i_0 σs) $\cap$ ε A_1 (i_1 σs)
then update-both A_0 A_1 (i_0 σs) (i_1 σs) e (*f-up-opt* f)
else *if* $e \in \varepsilon$ A_0 (i_0 σs) $-$ E $-$ ε A_1 (i_1 σs)
then update-left A_0 (i_0 σs) (i_1 σs) e (*f-up-opt* f) ($\lambda s.$ $\lfloor s \rfloor$)
else *if* $e \in \varepsilon$ A_1 (i_1 σs) $-$ E $-$ ε A_0 (i_0 σs)

then update-right A_1 $(i_0\ \sigma s)$ $(i_1\ \sigma s)$ e (*f-up-opt f*) $(\lambda s.\ \lfloor s \rfloor)$
else $\Diamond$)›
| ‹*combine*$_{nd}$*-Sync-*τ A_0 E A_1 i_0 i_1 f σs e =
(*if* $e \in \varepsilon\ A_0\ (i_0\ \sigma s) \cap \varepsilon\ A_1\ (i_1\ \sigma s) \cap E$
then update-both A_0 A_1 $(i_0\ \sigma s)$ $(i_1\ \sigma s)$ e (*f-up-set f*)
else *if* $e \in \varepsilon\ A_0\ (i_0\ \sigma s) \cap \varepsilon\ A_1\ (i_1\ \sigma s) - E$
then update-left A_0 $(i_0\ \sigma s)$ $(i_1\ \sigma s)$ e (*f-up-set f*) $(\lambda s.\ \{s\})$
$\cup$ *update-right* A_1 $(i_0\ \sigma s)$ $(i_1\ \sigma s)$ e (*f-up-set f*) $(\lambda s.\ \{s\})$
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-up-set f*) $(\lambda s.\ \{s\})$
else *if* $e \in \varepsilon\ A_1\ (i_1\ \sigma s) - E - \varepsilon\ A_0\ (i_0\ \sigma s)$
then update-right A_1 $(i_0\ \sigma s)$ $(i_1\ \sigma s)$ e (*f-up-set f*) $(\lambda s.\ \{s\})$
else $\{\}$)›

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

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

5.1.2 Specializations

definition *combine*$_{dPairlist}$*-Sync* ::
‹[$('\sigma,\ 'e,\ 'r,\ '\alpha)$ A_d*-scheme*, $'e$ *set*, $('\sigma,\ 'e,\ 'r,\ '\beta)$ A_d*-scheme*] $\Rightarrow$ ($'\sigma$ *list*, $'e$, $'r$) A_d›
where ‹*combine*$_{dPairlist}$*-Sync* A_0 E A_1 $\equiv$
combine$_d$*-Sync* A_0 E A_1 *hd* $(\lambda \sigma s.\ hd\ (tl\ \sigma s))$ $(\lambda s\ t.\ [s,\ t])$ $(\lambda s\ t.\ if\ s = t\ then\ \lfloor s \rfloor\ else\ \Diamond)$›
definition *combine*$_{ndPairlist}$*-Sync* ::

‹[('σ, 'e, 'r, 'α) A_{nd}-scheme, 'e set, ('σ, 'e, 'r, 'β) A_{nd}-scheme] ⇒ ('σ list, 'e, 'r) A_{nd}›
where ‹$combine_{ndPairlist}$-Sync A_0 E A_1 ≡
$combine_{nd}$-Sync A_0 E A_1 hd (λσs. hd (tl σs)) (λs t. [s, t]) (λs t. if s = t then ⌊s⌋ else ◊)›

definition $combine_{dPair}$-Sync ::
‹[('σ_0, 'e, 'r, 'α) A_d-scheme, 'e set, ('σ_1, 'e, 'r, 'β) A_d-scheme] ⇒ ('σ_0 × 'σ_1, 'e, 'r) A_d›
where ‹$combine_{dPair}$-Sync A_0 E A_1 ≡
$combine_d$-Sync A_0 E A_1 fst snd Pair (λs t. if s = t then ⌊s⌋ else ◊)›
definition $combine_{ndPair}$-Sync ::
‹[('σ_0, 'e, 'r, 'α) A_{nd}-scheme, 'e set, ('σ_1, 'e, 'r, 'β) A_{nd}-scheme] ⇒ ('σ_0 × 'σ_1, 'e, 'r) A_{nd}›
where ‹$combine_{ndPair}$-Sync A_0 E A_1 ≡
$combine_{nd}$-Sync A_0 E A_1 fst snd Pair (λs t. if s = t then ⌊s⌋ else ◊)›

definition $combine_{dListslenL}$-Sync ::
‹[('σ list, 'e, 'r, 'α) A_d-scheme, nat, 'e set, ('σ list, 'e, 'r, 'β) A_d-scheme] ⇒ ('σ list, 'e, 'r) A_d›
where ‹$combine_{dListslenL}$-Sync A_0 len_0 E A_1 ≡
$combine_d$-Sync A_0 E A_1 (take len_0) (drop len_0) (@) (λs t. if s = t then ⌊s⌋ else ◊)›
definition $combine_{ndListslenL}$-Sync ::
‹[('σ list, 'e, 'r, 'α) A_{nd}-scheme, nat, 'e set, ('σ list, 'e, 'r, 'β) A_{nd}-scheme] ⇒ ('σ list, 'e, 'r) A_{nd}›
where ‹$combine_{ndListslenL}$-Sync A_0 len_0 E A_1 ≡
$combine_{nd}$-Sync A_0 E A_1 (take len_0) (drop len_0) (@) (λs t. if s = t then ⌊s⌋ else ◊)›

definition $combine_{dRlist}$-Sync ::
‹[('σ, 'e, 'r, 'α) A_d-scheme, 'e set, ('σ list, 'e, 'r, 'β) A_d-scheme] ⇒ ('σ list, 'e, 'r) A_d›
where ‹$combine_{dRlist}$-Sync A_0 E A_1 ≡
$combine_d$-Sync A_0 E A_1 hd tl (#) (λs t. if s = t then ⌊s⌋ else ◊)›
definition $combine_{ndRlist}$-Sync ::
‹[('σ, 'e, 'r, 'α) A_{nd}-scheme, 'e set, ('σ list, 'e, 'r, 'β) A_{nd}-scheme] ⇒ ('σ list, 'e, 'r) A_{nd}›
where ‹$combine_{ndRlist}$-Sync A_0 E A_1 ≡
$combine_{nd}$-Sync A_0 E A_1 hd tl (#) (λs t. if s = t then ⌊s⌋ else ◊)›

lemmas $combine_{Pairlist}$-Sync-defs = $combine_{dPairlist}$-Sync-def $combine_{ndPairlist}$-Sync-def
and $combine_{Pair}$-Sync-defs = $combine_{dPair}$-Sync-def $combine_{ndPair}$-Sync-def
and $combine_{ListslenL}$-Sync-defs = $combine_{dListslenL}$-Sync-def $combine_{ndListslenL}$-Sync-def
and $combine_{Rlist}$-Sync-defs = $combine_{dRlist}$-Sync-def $combine_{ndRlist}$-Sync-def

lemmas combine-Sync-defs =
$combine_{Pairlist}$-Sync-defs $combine_{Pair}$-Sync-defs $combine_{ListslenL}$-Sync-defs $combine_{Rlist}$-Sync-defs

bundle *combine$_{nd}$-Sync-syntax* **begin**

notation *combine$_{dPairlist}$-Sync* (‹⟪- $_{d}\otimes$⟦-⟧$_{Pairlist}$ -⟫› [0, 0, 0])
notation *combine$_{ndPairlist}$-Sync* (‹⟪- $_{nd}\otimes$⟦-⟧$_{Pairlist}$ -⟫› [0, 0, 0])
notation *combine$_{dPair}$-Sync* (‹⟪- $_{d}\otimes$⟦-⟧$_{Pair}$ -⟫› [0, 0, 0])
notation *combine$_{ndPair}$-Sync* (‹⟪- $_{nd}\otimes$⟦-⟧$_{Pair}$ -⟫› [0, 0, 0])
notation *combine$_{dListslenL}$-Sync* (‹⟪- $_{d}\otimes$⟦-, -⟧$_{ListslenL}$ -⟫› [0, 0, 0, 0])
notation *combine$_{ndListslenL}$-Sync* (‹⟪- $_{nd}\otimes$⟦-, -⟧$_{ListslenL}$ -⟫› [0, 0, 0, 0])
notation *combine$_{dRlist}$-Sync* (‹⟪- $_{d}\otimes$⟦-⟧$_{Rlist}$ -⟫› [0, 0, 0])
notation *combine$_{ndRlist}$-Sync* (‹⟪- $_{nd}\otimes$⟦-⟧$_{Rlist}$ -⟫› [0, 0, 0])

end

unbundle *combine$_{nd}$-Sync-syntax*

5.2 First Properties

lemma *finite-trans-combine$_{nd}$-Sync-simps* [*simp*] :
‹*finite-trans* A_0 ⟹ *finite-trans* A_1 ⟹ *finite-trans* ⟪A_0 $_{nd}\otimes$⟦E⟧$_{Pairlist}$ A_1⟫›
‹*finite-trans* B_0 ⟹ *finite-trans* B_1 ⟹ *finite-trans* ⟪B_0 $_{nd}\otimes$⟦E⟧$_{Pair}$ B_1⟫›
‹*finite-trans* C_0 ⟹ *finite-trans* C_1 ⟹ *finite-trans* ⟪C_0 $_{nd}\otimes$⟦len_0, E⟧$_{ListslenL}$ C_1⟫›
‹*finite-trans* D_0 ⟹ *finite-trans* D_1 ⟹ *finite-trans* ⟪D_0 $_{nd}\otimes$⟦E⟧$_{Rlist}$ D_1⟫›
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*:
‹ε ⟪A_0 $_{d}\otimes$⟦E⟧$_{Pairlist}$ A_1⟫ σs = *combine-Sync-ε* A_0 E A_1 *hd* (*hd* ∘ *tl*) σs›
‹ε ⟪B_0 $_{nd}\otimes$⟦E⟧$_{Pairlist}$ B_1⟫ σs = *combine-Sync-ε* B_0 E B_1 *hd* (*hd* ∘ *tl*) σs›
by (*auto simp add*: *combine-Sync-ε-def-bis combine$_{Pairlist}$-Sync-defs ε-simps*)

lemma *ε-combine$_{Pair}$-Sync*:
‹ε ⟪A_0 $_{d}\otimes$⟦E⟧$_{Pair}$ A_1⟫ σs = *combine-Sync-ε* A_0 E A_1 *fst snd* σs›
‹ε ⟪B_0 $_{nd}\otimes$⟦E⟧$_{Pair}$ B_1⟫ σs = *combine-Sync-ε* B_0 E B_1 *fst snd* σs›
by (*auto simp add*: *combine-Sync-ε-def-bis combine$_{Pair}$-Sync-defs ε-simps*)

lemma *ε-combine$_{ListslenL}$-Sync*:
‹ε ⟪A_0 $_{d}\otimes$⟦len_0, E⟧$_{ListslenL}$ A_1⟫ σs = *combine-Sync-ε* A_0 E A_1 (*take* len_0) (*drop* len_0) σs›
‹ε ⟪B_0 $_{nd}\otimes$⟦len_0, E⟧$_{ListslenL}$ B_1⟫ σs = *combine-Sync-ε* B_0 E B_1 (*take* len_0) (*drop* len_0) σs›
by (*auto simp add*: *combine-Sync-ε-def-bis combine$_{ListslenL}$-Sync-defs ε-simps*)

lemma *ε-combine$_{Rlist}$-Sync*:
‹ε ⟪A_0 $_{d}\otimes$⟦E⟧$_{Rlist}$ A_1⟫ σs = *combine-Sync-ε* A_0 E A_1 *hd tl* σs›
‹ε ⟪B_0 $_{nd}\otimes$⟦E⟧$_{Rlist}$ B_1⟫ σs = *combine-Sync-ε* B_0 E B_1 *hd tl* σs›

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

lemma ϱ-$combine_{Pairlist}$*-Sync*:
‹ϱ $\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\rangle = \{\sigma s.\ hd\ \sigma s \in \varrho\ A_0 \wedge hd\ (tl\ \sigma s) \in \varrho\ A_1 \wedge \omega\ A_0\ (hd\ \sigma s) = \omega\ A_1\ (hd\ (tl\ \sigma s))\}$›
‹ϱ $\langle\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ B_1\rangle\rangle = \{\sigma s.\ hd\ \sigma s \in \varrho\ B_0 \wedge hd\ (tl\ \sigma s) \in \varrho\ B_1 \wedge \omega\ B_0\ (hd\ \sigma s) \cap \omega\ B_1\ (hd\ (tl\ \sigma s)) \neq \{\}\}$›
by (*auto simp add*: $combine_{Pairlist}$*-Sync-defs ϱ-simps split*: *option.split*)

lemma ϱ-$combine_{Pair}$*-Sync*:
‹ϱ $\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle = \{(\sigma_0, \sigma_1).\ \sigma_0 \in \varrho\ A_0 \wedge \sigma_1 \in \varrho\ A_1 \wedge \omega\ A_0\ \sigma_0 = \omega\ A_1\ \sigma_1\}$›
‹ϱ $\langle\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ B_1\rangle\rangle = \{(\sigma_0, \sigma_1).\ \sigma_0 \in \varrho\ B_0 \wedge \sigma_1 \in \varrho\ B_1 \wedge \omega\ B_0\ \sigma_0 \cap \omega\ B_1\ \sigma_1 \neq \{\}\}$›
by (*auto simp add*: $combine_{Pair}$*-Sync-defs ϱ-simps split*: *option.split*)

lemma ϱ-$combine_{ListslenL}$*-Sync*:
‹ϱ $\langle\langle A_0\ {}_d\otimes[\![len_0, E]\!]_{ListslenL}\ A_1\rangle\rangle =$
$\{\sigma s.\ take\ len_0\ \sigma s \in \varrho\ A_0 \wedge drop\ len_0\ \sigma s \in \varrho\ A_1 \wedge \omega\ A_0\ (take\ len_0\ \sigma s) = \omega\ A_1\ (drop\ len_0\ \sigma s)\}$›
‹ϱ $\langle\langle B_0\ {}_{nd}\otimes[\![len_0, E]\!]_{ListslenL}\ B_1\rangle\rangle =$
$\{\sigma s.\ take\ len_0\ \sigma s \in \varrho\ B_0 \wedge drop\ len_0\ \sigma s \in \varrho\ B_1 \wedge \omega\ B_0\ (take\ len_0\ \sigma s) \cap \omega\ B_1\ (drop\ len_0\ \sigma s) \neq \{\}\}$›
by (*auto simp add*: $combine_{ListslenL}$*-Sync-defs ϱ-simps split*: *option.split*)

lemma ϱ-$combine_{Rlist}$*-Sync*:
‹ϱ $\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ A_1\rangle\rangle =$
$\{\sigma s.\ hd\ \sigma s \in \varrho\ A_0 \wedge tl\ \sigma s \in \varrho\ A_1 \wedge \omega\ A_0\ (hd\ \sigma s) = \omega\ A_1\ (tl\ \sigma s)\}$›
‹ϱ $\langle\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ B_1\rangle\rangle =$
$\{\sigma s.\ hd\ \sigma s \in \varrho\ B_0 \wedge tl\ \sigma s \in \varrho\ B_1 \wedge \omega\ B_0\ (hd\ \sigma s) \cap \omega\ B_1\ (tl\ \sigma s) \neq \{\}\}$›
by (*auto simp add*: $combine_{Rlist}$*-Sync-defs ϱ-simps split*: *option.split*)

lemma *combine-Sync-τ* [*simp*] :
‹$combine_d$*-Sync-τ* $(A_0(\!|\omega := some\text{-}\omega_0|\!))\ E\ (A_1(\!|\omega := some\text{-}\omega_1|\!)) = combine_d$*-Sync-τ* $A_0\ E\ A_1$›
‹$combine_{nd}$*-Sync-τ* $(B_0(\!|\omega := some\text{-}\omega_0'|\!))\ E\ (B_1(\!|\omega := some\text{-}\omega_1'|\!)) = combine_{nd}$*-Sync-τ* $B_0\ E\ B_1$›
for A :: ‹$('\sigma, 'a, '\sigma\ option, 'r\ option, '\alpha)$ *A-scheme*›
and B :: ‹$('\sigma, 'a, '\sigma\ set, 'r\ set, '\alpha)$ *A-scheme*›
by (*intro ext*, *simp*)+

5.3 Reachability

lemma $\mathcal{R}_d$-$combine_{dListslenL}$*-Sync-subset*:
‹$\mathcal{R}_d$ $\langle\langle A_0\ {}_d\otimes[\![len_0, E]\!]_{ListslenL}\ A_1\rangle\rangle\ (s_0\ @\ s_1) \subseteq \{t_0\ @\ t_1 |\ t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1\}$› (**is** ‹$?S_A \subseteq$ -›)

if *same-length-*$\mathcal{R}_d$: ‹$\bigwedge t_0.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \Longrightarrow$ *length* $t_0 = len_0$›
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = t_0\ @\ t_1 \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1$› **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*:
‹$\mathcal{R}_{nd}\ \langle\!\langle B_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ B_1\rangle\!\rangle\ (s_0\ @\ s_1) \subseteq \{t_0\ @\ t_1 |\ t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1\}$› (**is** ‹$?S_B \subseteq$ -›)
if *same-length-*$\mathcal{R}_{nd}$: ‹$\bigwedge t_0.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \Longrightarrow$ *length* $t_0 = len_0$›
proof *safe*
show ‹$t \in ?S_B \Longrightarrow \exists\, t_0\ t_1.\ t = t_0\ @\ t_1 \wedge t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1$› **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*:
‹$\mathcal{R}_d\ \langle\!\langle A_0\ {}_{d}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\ [s_0,\ s_1] \subseteq \{[t_0,\ t_1] |\ t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1\}$› (**is** ‹$?S_A \subseteq$ -›)
and $\mathcal{R}_{nd}$*-combine*$_{ndPairlist}$*-Sync-subset*:
‹$\mathcal{R}_{nd}\ \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ B_1\rangle\!\rangle\ [s_0,\ s_1] \subseteq \{[t_0,\ t_1] |\ t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1\}$› (**is** ‹$?S_B \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = [t_0,\ t_1] \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1$› **for** t
by ($\mathcal{R}_d$*-subset-method defs*: *combine*$_{Pairlist}$*-Sync-defs*)
show ‹$t \in ?S_B \Longrightarrow \exists\, t_0\ t_1.\ t = [t_0,\ t_1] \wedge t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1$› **for** t
by ($\mathcal{R}_{nd}$*-subset-method defs*: *combine*$_{Pairlist}$*-Sync-defs*)
qed

lemma $\mathcal{R}_d$*-combine*$_{dPair}$*-Sync-subset*:
‹$\mathcal{R}_d\ \langle\!\langle A_0\ {}_{d}\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\ (s_0,\ s_1) \subseteq \mathcal{R}_d\ A_0\ s_0 \times \mathcal{R}_d\ A_1\ s_1$› (**is** ‹$?S_A \subseteq$ -›)
and $\mathcal{R}_{nd}$*-combine*$_{ndPair}$*-Sync-subset*:
‹$\mathcal{R}_{nd}\ \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ B_1\rangle\!\rangle\ (s_0,\ s_1) \subseteq \mathcal{R}_{nd}\ B_0\ s_0 \times \mathcal{R}_{nd}\ B_1\ s_1$› (**is** ‹$?S_B \subseteq$ -›)
proof −
have ‹$t \in ?S_A \Longrightarrow$ *fst* $t \in \mathcal{R}_d\ A_0\ s_0 \wedge$ *snd* $t \in \mathcal{R}_d\ A_1\ s_1$› **for** t
by ($\mathcal{R}_d$*-subset-method defs*: *combine*$_{Pair}$*-Sync-defs*)
thus ‹$?S_A \subseteq \mathcal{R}_d\ A_0\ s_0 \times \mathcal{R}_d\ A_1\ s_1$› **by** *force*
next
have ‹$t \in ?S_B \Longrightarrow$ *fst* $t \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge$ *snd* $t \in \mathcal{R}_{nd}\ B_1\ s_1$› **for** t
by ($\mathcal{R}_{nd}$*-subset-method defs*: *combine*$_{Pair}$*-Sync-defs*)
thus ‹$?S_B \subseteq \mathcal{R}_{nd}\ B_0\ s_0 \times \mathcal{R}_{nd}\ B_1\ s_1$› **by** *force*

qed

lemma $\mathcal{R}_d$-*combine*$_{dRlist}$-*Sync-subset*:
‹$\mathcal{R}_d$ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ A_1\rangle\!\rangle$ $(s_0\ \#\ \sigma s) \subseteq \{t_0\ \#\ \sigma t|\ t_0\ \sigma t.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge \sigma t \in \mathcal{R}_d\ A_1\ \sigma s\}$› (**is** ‹$?S_A \subseteq$ -›)
and $\mathcal{R}_{nd}$-*combine*$_{ndRlist}$-*Sync-subset*:
‹$\mathcal{R}_{nd}$ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ B_1\rangle\!\rangle$ $(s_0\ \#\ \sigma s) \subseteq \{t_0\ \#\ \sigma t|\ t_0\ \sigma t.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge \sigma t \in \mathcal{R}_{nd}\ B_1\ \sigma s\}$› (**is** ‹$?S_B \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ \sigma t.\ t = t_0\ \#\ \sigma t \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge \sigma t \in \mathcal{R}_d\ A_1\ \sigma s$› **for** t
by ($\mathcal{R}_d$-*subset-method defs*: *combine*$_{Rlist}$-*Sync-defs*)
next
show ‹$t \in ?S_B \Longrightarrow \exists\, t_0\ \sigma t.\ t = t_0\ \#\ \sigma t \wedge t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge \sigma t \in \mathcal{R}_{nd}\ B_1\ \sigma s$› **for** t
by ($\mathcal{R}_{nd}$-*subset-method defs*: *combine*$_{Rlist}$-*Sync-defs*)
qed

5.4 Normalization

lemma ω-*combine*$_{Pairlist}$-*Sync-behaviour*:
‹ω $\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $[s_0,\ s_1] = \omega$ $\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle$ $[s_0,\ s_1]$›
by (*auto simp add*: *combine*$_{Pairlist}$-*Sync-defs det-ndet-conv-defs option.case-eq-if*)

lemma ω-*combine*$_{Pair}$-*Sync-behaviour*:
‹ω $\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $(s_0,\ s_1) = \omega$ $\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pair}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle$ $(s_0,\ s_1)$›
by (*auto simp add*: *combine*$_{Pair}$-*Sync-defs det-ndet-conv-defs option.case-eq-if*)

lemma ω-*combine*$_{ListslenL}$-*Sync-behaviour*:
‹ω $\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $(\sigma s_0\ @\ \sigma s_1) = \omega$ $\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle$ $(\sigma s_0\ @\ \sigma s_1)$›
by (*auto simp add*: *combine*$_{ListslenL}$-*Sync-defs det-ndet-conv-defs option.case-eq-if*)

lemma ω-*combine*$_{Rlist}$-*Sync-behaviour*:
‹ω $\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $(s_0\ \#\ \sigma s_1) = \omega$ $\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle$ $(s_0\ \#\ \sigma s_1)$›
by (*auto simp add*: *combine*$_{Rlist}$-*Sync-defs det-ndet-conv-defs option.case-eq-if*)

lemma τ-*combine*$_{Pairlist}$-*Sync-behaviour-when-indep*:
‹$\varepsilon\ A_0\ s_0 \cap \varepsilon\ A_1\ s_1 \subseteq E \Longrightarrow$
τ $\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $[s_0,\ s_1]\ e = \tau$ $\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle$ $[s_0,\ s_1]\ e$›
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*:
‹$\varepsilon\ A_0\ s_0 \cap \varepsilon\ A_1\ s_1 \subseteq E \Longrightarrow$
$\tau\ \langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ (s_0,\ s_1)\ e = \tau\ \langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pair}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\ (s_0,\ s_1)\ e$›
by (*auto simp add*: *combine*$_{Pair}$-*Sync-defs det-ndet-conv-defs option.case-eq-if* ε-*simps*)

lemma τ-*combine*$_{ListslenL}$-*Sync-behaviour-when-indep*:
‹$\varepsilon\ A_0\ \sigma s_0 \cap \varepsilon\ A_1\ \sigma s_1 \subseteq E \Longrightarrow$ *length* $\sigma s_0 = len_0 \Longrightarrow$
$\tau\ \langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ (\sigma s_0\ @\ \sigma s_1)\ e = \tau\ \langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\ (\sigma s_0\ @\ \sigma s_1)\ e$›
by (*auto simp add*: *combine*$_{ListslenL}$-*Sync-defs det-ndet-conv-defs option.case-eq-if* ε-*simps*)

lemma τ-*combine*$_{Rlist}$-*Sync-behaviour-when-indep*:
‹$\varepsilon\ A_0\ s_0 \cap \varepsilon\ A_1\ \sigma s_1 \subseteq E \Longrightarrow$
$\tau\ \langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ (s_0\ \#\ \sigma s_1)\ e = \tau\ \langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\ (s_0\ \#\ \sigma s_1)\ e$›
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 ‹*drule set-mp*[*OF R-d-subset*, *rotated*]›

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 ‹*drule set-mp*[*OF R-d-subset*, *rotated*]›

lemma P_{SKIPS}-*combine*$_{Pairlist}$-*Sync-behaviour-when-indep*:
‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [s_0,\ s_1] = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\rangle\!\rangle_{nd}\ [s_0,\ s_1]$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ s_1$›
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*:
‹$P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [s_0,\ s_1] = P\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\rangle\!\rangle_{nd}\ [s_0,\ s_1]$›

if ‹*indep-enabl* A_0 s_0 E A_1 s_1›
by (*P-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*)

lemma P_{SKIPS}-*combine*$_{Pair}$-*Sync-behaviour-when-indep*:
‹$P_{SKIPS}\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle\rangle_d\ (s_0,\ s_1) = P_{SKIPS}\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pair}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (s_0,\ s_1)$›
if ‹*indep-enabl* A_0 s_0 E A_1 s_1›
by (P_{SKIPS}-*when-indep-method R-d-subset*: $\mathcal{R}_d$-*combine*$_{dPair}$-*Sync-subset*, *all* ‹*elim SigmaE*›)
(*metis* τ-*combine*$_{Pair}$-*Sync-behaviour-when-indep indep-enablD that*,
auto simp add: ω-*combine*$_{Pair}$-*Sync-behaviour option.case-eq-if*)

lemma *P-combine*$_{Pair}$-*Sync-behaviour-when-indep*:
‹$P\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle\rangle_d\ (s_0,\ s_1) = P\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pair}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (s_0,\ s_1)$›
if ‹*indep-enabl* A_0 s_0 E A_1 s_1›
by (*P-when-indep-method R-d-subset*: $\mathcal{R}_d$-*combine*$_{dPair}$-*Sync-subset*, *elim SigmaE*)
(*metis* τ-*combine*$_{Pair}$-*Sync-behaviour-when-indep indep-enablD that*)

lemma P_{SKIPS}-*combine*$_{ListslenL}$-*Sync-behaviour-when-indep*:
‹$P_{SKIPS}\langle\langle\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\rangle\rangle_d\ (\sigma s_0\ @\ \sigma s_1) = P_{SKIPS}\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (\sigma s_0\ @\ \sigma s_1)$›
if ‹*indep-enabl* A_0 σs_0 E A_1 σs_1› **and** ‹$\bigwedge\sigma t_0.\ \sigma t_0 \in \mathcal{R}_d\ A_0\ \sigma s_0 \Longrightarrow$ *length* $\sigma t_0 = len_0$›
by (P_{SKIPS}-*when-indep-method R-d-subset*: $\mathcal{R}_d$-*combine*$_{dListslenL}$-*Sync-subset*, *simp-all add*: *that*(*2*))
(*metis* τ-*combine*$_{ListslenL}$-*Sync-behaviour-when-indep indep-enablD that*,
metis ω-*combine*$_{ListslenL}$-*Sync-behaviour*)

lemma *P-combine*$_{ListslenL}$-*Sync-behaviour-when-indep*:
‹$P\langle\langle\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\rangle\rangle_d\ (\sigma s_0\ @\ \sigma s_1) = P\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (\sigma s_0\ @\ \sigma s_1)$›
if ‹*indep-enabl* A_0 σs_0 E A_1 σs_1› **and** ‹$\bigwedge\sigma t_0.\ \sigma t_0 \in \mathcal{R}_d\ A_0\ \sigma s_0 \Longrightarrow$ *length* $\sigma t_0 = len_0$›
by (*P-when-indep-method R-d-subset*: $\mathcal{R}_d$-*combine*$_{dListslenL}$-*Sync-subset*, *simp-all add*: *that*(*2*))
(*metis* τ-*combine*$_{ListslenL}$-*Sync-behaviour-when-indep indep-enablD that*)

lemma P_{SKIPS}-*combine*$_{Rlist}$-*Sync-behaviour-when-indep*:
‹$P_{SKIPS}\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ A_1\rangle\rangle\rangle_d\ (s_0\ \#\ \sigma s_1) = P_{SKIPS}\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (s_0\ \#\ \sigma s_1)$›
if ‹*indep-enabl* A_0 s_0 E A_1 σs_1›
by (P_{SKIPS}-*when-indep-method R-d-subset*: $\mathcal{R}_d$-*combine*$_{dRlist}$-*Sync-subset*, *simp-all*)
(*metis* τ-*combine*$_{Rlist}$-*Sync-behaviour-when-indep indep-enablD that*,

metis ω-*combine*$_{Rlist}$-*Sync-behaviour*)

lemma *P-combine*$_{Rlist}$-*Sync-behaviour-when-indep*:
‹$P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\!\rangle_d\ (s_0\ \#\ \sigma s_1) = P\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\!\rangle_{nd}$ $(s_0\ \#\ \sigma s_1)$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ \sigma s_1$›
by (*P-when-indep-method R-d-subset*: $\mathcal{R}_d$-*combine*$_{dRlist}$-*Sync-subset*, *simp*)
(*metis* τ-*combine*$_{Rlist}$-*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* :: ‹$'e$ *set* $\Rightarrow$ $('\sigma,\ 'e,\ 'r)\ A_d$ *list* $\Rightarrow$ $('\sigma$ *list*, $'e,\ 'r)\ A_d$› (‹$\langle\!\langle{}_d\bigotimes [\![-]\!]\ -\rangle\!\rangle$› $[0,\ 0]$)
  where ‹$\langle\!\langle{}_d\bigotimes [\![E]\!]\ []\rangle\!\rangle = (\!|\tau = \lambda\sigma s\ a.\ \Diamond,\ \omega = \lambda\sigma s.\ \Diamond|\!)$›
  | ‹$\langle\!\langle{}_d\bigotimes [\![E]\!]\ [A_0]\rangle\!\rangle = {}_d\langle\!\langle A_0\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›
  | ‹$\langle\!\langle{}_d\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle = \langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ \langle\!\langle{}_d\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle$›

fun *iterated-combine*$_{nd}$*-Sync* :: ‹$'e$ *set* $\Rightarrow$ $('\sigma,\ 'e,\ 'r)\ A_{nd}$ *list* $\Rightarrow$ $('\sigma$ *list*, $'e,\ 'r)\ A_{nd}$› (‹$\langle\!\langle{}_{nd}\bigotimes [\![-]\!]\ -\rangle\!\rangle$› $[0,\ 0]$)
  where ‹$\langle\!\langle{}_{nd}\bigotimes [\![E]\!]\ []\rangle\!\rangle = (\!|\tau = \lambda\sigma s\ a.\ \{\},\ \omega = \lambda\sigma s.\ \{\}|\!)$›
  | ‹$\langle\!\langle{}_{nd}\bigotimes [\![E]\!]\ [A_0]\rangle\!\rangle = {}_{nd}\langle\!\langle A_0\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›
  | ‹$\langle\!\langle{}_{nd}\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle = \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle{}_{nd}\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle$›

lemma *iterated-combine*$_d$*-Sync-simps-bis*: ‹$As \neq [] \Longrightarrow \langle\!\langle{}_d\bigotimes [\![E]\!]\ A_0\ \#\ As\rangle\!\rangle = \langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Rlist}\ \langle\!\langle{}_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle$›
  and *iterated-combine*$_{nd}$*-Sync-simps-bis*: ‹$Bs \neq [] \Longrightarrow \langle\!\langle{}_{nd}\bigotimes [\![E]\!]\ B_0\ \#\ Bs\rangle\!\rangle = \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle{}_{nd}\bigotimes [\![E]\!]\ Bs\rangle\!\rangle\rangle\!\rangle$›
  by (*induct As*, *simp-all*) (*induct Bs*, *simp-all*)

fun *iterated-combine*$_d$*-Sync-*ε :: ‹$('\sigma,\ 'e,\ 'r,\ '\alpha)\ A_d$*-scheme list* $\Rightarrow$ $'e$ *set* $\Rightarrow$ $'\sigma$ *list* $\Rightarrow$ $'e$ *set*›
  where ‹*iterated-combine*$_d$*-Sync-*ε $[]\ E\ \sigma s = \{\}$›
  | ‹*iterated-combine*$_d$*-Sync-*ε $[A_0]\ E\ \sigma s = \varepsilon\ A_0\ (hd\ \sigma s)$›
  | ‹*iterated-combine*$_d$*-Sync-*ε $(A_0\ \#\ A_1\ \#\ As)\ E\ \sigma s =$
    combine-sets-Sync $(\varepsilon\ A_0\ (hd\ \sigma s))\ E$ (*iterated-combine*$_d$*-Sync-*ε $(A_1\ \#\ As)\ E\ (tl\ \sigma s)$)›

fun *iterated-combine*$_{nd}$*-Sync-ε* :: ‹($'\sigma$, $'e$, $'r$, $'\alpha$) A_{nd}*-scheme list* ⇒ $'e$ *set* ⇒ $'\sigma$ *list* ⇒ $'e$ *set*›
where ‹*iterated-combine*$_{nd}$*-Sync-ε* [] *E σs* = {}›
| ‹*iterated-combine*$_{nd}$*-Sync-ε* [A_0] *E σs* = *ε* A_0 (*hd σs*)›
| ‹*iterated-combine*$_{nd}$*-Sync-ε* (A_0 # A_1 # *As*) *E σs* =
combine-sets-Sync (*ε* A_0 (*hd σs*)) *E* (*iterated-combine*$_{nd}$*-Sync-ε* (A_1 # *As*) *E* (*tl σs*))›

lemma *iterated-combine*$_d$*-Sync-ε-simps-bis*:
‹*As* ≠ [] ⟹ *iterated-combine*$_d$*-Sync-ε* (A_0 # *As*) *E σs* =
combine-sets-Sync (*ε* A_0 (*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* (B_0 # *Bs*) *E σs* =
combine-sets-Sync (*ε* B_0 (*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* ⟪$_d$⟪A_0⟫$_{\sigma\hookrightarrow\sigma}$ $_d$⨂⟦*1*, *E*⟧$_{ListslenL}$ A_1⟫ *σs* = *ε* ⟪A_0 $_d$⨂⟦*E*⟧$_{Rlist}$ A_1⟫ *σs*›
‹*τ* ⟪$_d$⟪A_0⟫$_{\sigma\hookrightarrow\sigma}$ $_d$⨂⟦*1*, *E*⟧$_{ListslenL}$ A_1⟫ (s_0 # *σs*) *e* = *τ* ⟪A_0 $_d$⨂⟦*E*⟧$_{Rlist}$ A_1⟫ (s_0 # *σs*) *e*›
‹*ε* ⟪$_{nd}$⟪B_0⟫$_{\sigma\hookrightarrow\sigma}$ $_{nd}$⨂⟦*1*, *E*⟧$_{ListslenL}$ B_1⟫ *σs* = *ε* ⟪B_0 $_{nd}$⨂⟦*E*⟧$_{Rlist}$ B_1⟫ *σs*›
‹*τ* ⟪$_{nd}$⟪B_0⟫$_{\sigma\hookrightarrow\sigma}$ $_{nd}$⨂⟦*1*, *E*⟧$_{ListslenL}$ B_1⟫ (s_0 # *σs*) *e* = *τ* ⟪B_0 $_{nd}$⨂⟦*E*⟧$_{Rlist}$ B_1⟫ (s_0 # *σs*) *e*›
by (*simp-all add*: *ε-combine*$_{ListslenL}$*-Sync ε-combine*$_{Rlist}$*-Sync drop-Suc combine-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 *combine$_{Pairlist}$-Sync-and-iterated-combine$_{nd}$-Sync-eq*:
‹ε ⟪A_0 $_d\otimes[\![E]\!]_{Pairlist}$ A_1⟫ $[s_0,\ s_1] = \varepsilon$ ⟪$_d\bigotimes$ $[\![E]\!]$ $[A_0,\ A_1]$⟫ $[s_0,\ s_1]$›
‹τ ⟪A_0 $_d\otimes[\![E]\!]_{Pairlist}$ A_1⟫ $[s_0,\ s_1]$ $e = \tau$ ⟪$_d\bigotimes$ $[\![E]\!]$ $[A_0,\ A_1]$⟫ $[s_0,\ s_1]$ e›
‹ε ⟪B_0 $_{nd}\otimes[\![E]\!]_{Pairlist}$ B_1⟫ $[s_0,\ s_1] = \varepsilon$ ⟪$_{nd}\bigotimes$ $[\![E]\!]$ $[B_0,\ B_1]$⟫ $[s_0,\ s_1]$›
‹τ ⟪B_0 $_{nd}\otimes[\![E]\!]_{Pairlist}$ B_1⟫ $[s_0,\ s_1]$ $e = \tau$ ⟪$_{nd}\bigotimes$ $[\![E]\!]$ $[B_0,\ B_1]$⟫ $[s_0,\ s_1]$ e›
by (*simp-all add*: *ε-combine$_{Pairlist}$-Sync ε-combine$_{Rlist}$-Sync*)
(*auto simp add*: *combine$_{Pairlist}$-Sync-defs combine$_{Rlist}$-Sync-defs σ-σs-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*:
‹*length σs = length As* $\Longrightarrow$ $\sigma s' \in \mathcal{R}_d$ ⟪$_d\bigotimes$ $[\![E]\!]$ *As*⟫ $\sigma s \Longrightarrow$
length $\sigma s'$ = length As $\wedge$ ($\forall$ *i* < *length As*. $\sigma s'$! *i* $\in \mathcal{R}_d$ (*As* ! *i*) (σs ! *i*))›
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-σs-description*)
next
case (*Cons $\sigma 1$ $\sigma 2$ σs A1 A2 As*)
from *set-mp*[*OF $\mathcal{R}_d$-combine$_{dRlist}$-Sync-subset*, *OF Cons.prems*[*simplified*]]
obtain *$\sigma 1'$ $\sigma s''$* **where** ‹$\sigma s' = \sigma 1'$ # $\sigma s''$› ‹$\sigma 1' \in \mathcal{R}_d$ *A1* $\sigma 1$›
‹$\sigma s'' \in \mathcal{R}_d$ ⟪$_d\bigotimes$ $[\![E]\!]$ *A2* # *As*⟫ ($\sigma 2$ # σs)› **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*:
‹*length σs = length Bs* $\Longrightarrow$ $\sigma s' \in \mathcal{R}_{nd}$ ⟪$_{nd}\bigotimes$ $[\![E]\!]$ *Bs*⟫ $\sigma s \Longrightarrow$
length $\sigma s'$ = length Bs $\wedge$ ($\forall$ *i* < *length Bs*. $\sigma s'$! *i* $\in \mathcal{R}_{nd}$ (*Bs* ! *i*) (σs ! *i*))›
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-σs-description*)
next
case (*Cons $\sigma 1$ $\sigma 2$ σs A1 A2 As*)
from *set-mp*[*OF $\mathcal{R}_{nd}$-combine$_{ndRlist}$-Sync-subset*, *OF Cons.prems*[*simplified*]]
obtain *$\sigma 1'$ $\sigma s''$* **where** ‹$\sigma s' = \sigma 1'$ # $\sigma s''$› ‹$\sigma 1' \in \mathcal{R}_{nd}$ *A1* $\sigma 1$›
‹$\sigma s'' \in \mathcal{R}_{nd}$ ⟪$_{nd}\bigotimes$ $[\![E]\!]$ *A2* # *As*⟫ ($\sigma 2$ # σs)› **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*:
‹$(\bigwedge A.\ A \in set\ As \Longrightarrow finite\text{-}trans\ A) \Longrightarrow finite\text{-}trans\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle$›
by (*induct As rule*: *induct-list012*)
(*auto simp add*: *σ-σs-conv-defs combine*$_{Rlist}$*-Sync-defs finite-trans-def finite-image-set2*)

lemma *ϱ-disjoint-ε-transmission-to-iterated-combine*$_{d}$*-Sync*:
‹$(\bigwedge A.\ A \in set\ As \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A) \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ \langle\!\langle_{d}\bigotimes [\![E]\!]\ As\rangle\!\rangle$›
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*:
‹$(\bigwedge A.\ A \in set\ As \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A) \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle$›
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*:
‹$(\bigwedge A.\ A \in set\ As \Longrightarrow at\text{-}most\text{-}1\text{-}elem\text{-}term\ A) \Longrightarrow at\text{-}most\text{-}1\text{-}elem\text{-}term\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle$›
by (*induct As rule*: *induct-list012*,
simp-all add: *at-most-1-elem-term-def σ-σs-conv-defs combine*$_{Rlist}$*-Sync-defs*)
fastforce

lemma *same-length-indep-transmission-to-iterated-combine*$_{d}$*-Sync*:
‹$length\ \sigma s = length\ As \Longrightarrow$
$(\bigwedge i\ j.\ i \le length\ As \Longrightarrow j \le length\ As \Longrightarrow i \ne j \Longrightarrow$
$indep\text{-}enabl\ ((A_0\ \#\ As)\ !\ i)\ ((s_0\ \#\ \sigma s)\ !\ i)\ E\ ((A_0\ \#\ As)\ !\ j)\ ((s_0\ \#\ \sigma s)\ !\ j))$
$\Longrightarrow$
$indep\text{-}enabl\ A_0\ s_0\ E\ \langle\!\langle_{d}\bigotimes [\![E]\!]\ As\rangle\!\rangle\ \sigma s$›
proof (*induct σs As rule*: *induct-2-lists012*)
case *Nil*
then show *?case* **by** (*simp add*: *indep-enablI*)
next
case (*single* σ_1 A_1)
from *single.prems*[*of 0 1*] **show** *?case*
by (*auto simp add*: $\mathcal{R}_d$*-from-σ-to-σs-description*
intro!: *indep-enablI dest*!: *indep-enablD*)
next
case (*Cons* σ_1 σ_2 σs A_1 A_2 *As*)
show *?case*
proof (*rule indep-enablI*)
fix t_0 *ts* **assume** *assms* : ‹$t_0 \in \mathcal{R}_d\ A_0\ s_0$› ‹$ts \in \mathcal{R}_d\ \langle\!\langle_{d}\bigotimes [\![E]\!]\ A_1\ \#\ A_2\ \#\ As\rangle\!\rangle$

$(\sigma_1 \# \sigma_2 \# \sigma s)$›
from *assms*(*2*) **obtain** t_1 ts' **where** ‹$ts = t_1 \# ts'$›
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*$_{dRlist}$*-Sync-subset*]]
have ‹$ts' \in \mathcal{R}_d$ $\langle\!\langle_d\bigotimes [\![E]\!]\ A_2 \# As\rangle\!\rangle$ $(\sigma_2 \# \sigma s)$› **by** *simp*

have ‹*indep-enabl* A_0 s_0 E $\langle\!\langle_d\bigotimes [\![E]\!]\ A_2 \# As\rangle\!\rangle$ $(\sigma_2 \# \sigma s)$›
proof (*rule Cons.hyps*(*3*))
show ‹$i \leq$ *length* $(A_2 \# As) \Longrightarrow j \leq$ *length* $(A_2 \# As) \Longrightarrow i \neq j \Longrightarrow$ *indep-enabl* $((A_0 \# A_2 \# As)\ !\ i)$ $((s_0 \# \sigma_2 \# \sigma s)\ !\ i)$ E $((A_0 \# A_2 \# As)\ !\ j)$ $((s_0 \# \sigma_2 \# \sigma s)\ !\ j)$› **for** i j
using *Cons.prems*[*of* ‹*if* $i = 0$ *then* 0 *else Suc* i› ‹*if* $j = 0$ *then* 0 *else Suc* j›]
by (*cases* i; *cases* j) *simp-all*
qed
from *this*[*THEN indep-enablD*, *OF assms*(*1*) ‹$ts' \in \mathcal{R}_d$ $\langle\!\langle_d\bigotimes [\![E]\!]\ A_2 \# As\rangle\!\rangle$ $(\sigma_2 \# \sigma s)$›]
have ‹$\varepsilon\ A_0\ t_0 \cap \varepsilon$ $\langle\!\langle_d\bigotimes [\![E]\!]\ A_2 \# As\rangle\!\rangle$ $ts' \subseteq E$› **.**
moreover from *Cons.prems*[*THEN indep-enablD*, *of* 0 ‹*Suc* 0› t_0 t_1] *assms*(*2*)[*simplified*, *THEN set-mp*[*OF* $\mathcal{R}_d$*-combine*$_{dRlist}$*-Sync-subset*]] *assms*(*1*)
have ‹$\varepsilon\ A_0\ t_0 \cap \varepsilon\ A_1\ t_1 \subseteq E$› **by** (*simp add*: ‹$ts = t_1 \# ts'$›)
ultimately show ‹$\varepsilon\ A_0\ t_0 \cap \varepsilon$ $\langle\!\langle_d\bigotimes [\![E]\!]\ A_1 \# A_2 \# As\rangle\!\rangle$ $ts \subseteq E$›
by (*auto simp add*: ε*-combine*$_{Rlist}$*-Sync* ‹$ts = t_1 \# ts'$› *combine-Sync-*ε*-def*)
qed
qed

lemma ω*-iterated-combine*$_d$*-Sync* :
‹*length* σs = *length* $As \Longrightarrow$
ω $\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle$ σs = (*case those* (*map2* ω As σs) *of* $\Diamond \Rightarrow \Diamond$
| $\lfloor$*terms*$\rfloor \Rightarrow$ *if card* (*set terms*) = *Suc* 0 *then* $\lfloor$*THE* r. *set terms* = $\{r\}\rfloor$ *else* $\Diamond$)›
by (*induct* σs As *rule*: *induct-2-lists012*)
(*auto simp add*: σ*-*σs*-conv-defs combine*$_{Rlist}$*-Sync-defs card-1-singleton-iff the-equality split*: *option.split*)

lemma ω*-iterated-combine*$_{nd}$*-Sync* :
‹*length* σs = *length* $As \Longrightarrow$
ω $\langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle$ σs = (*if* As = [] *then* {} *else* $\{r.\ \forall i <$ *length* $As.\ r \in \omega\ (As\ !\ i)\ (\sigma s\ !\ i)\}$)›
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*: σ*-*σs*-conv-defs*)
next
case (*Cons* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ As)
show *?case* (**is** ‹- = *?rhs* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ As›)

proof (*intro subset-antisym subsetI*)
fix r **assume** ‹$r \in \omega$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ $A1$ # $A2$ # $As\rangle\!\rangle$ ($\sigma 1$ # $\sigma 2$ # σs)›
hence ‹$r \in \omega$ $A1$ $\sigma 1$› ‹$r \in \omega$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ $A2$ # $As\rangle\!\rangle$ ($\sigma 2$ # σs)›
by (*simp-all add*: *combine*$_{Rlist}$*-Sync-defs split*: *if-split-asm*)
from *this*(*2*) **have** ‹$\forall$ *i*<*Suc* (*length As*). $r \in \omega$ (($A2$ # As) ! i) (($\sigma 2$ # σs) ! i)›
by (*simp add*: *Cons.hyps*(*3*))
with ‹$r \in \omega$ $A1$ $\sigma 1$› **show** ‹$r \in$ *?rhs* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ As›
by (*auto simp add*: *less-Suc-eq-0-disj*)
next
from *Cons.hyps*(*3*)
show ‹$r \in$ *?rhs* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ $As \Longrightarrow$
$r \in \omega$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ $A1$ # $A2$ # $As\rangle\!\rangle$ ($\sigma 1$ # $\sigma 2$ # σs)› **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*:
‹*length* σs = *length* $As \Longrightarrow$
ω $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *map* (λA. $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$) $As\rangle\!\rangle$ σs = ω $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ $As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ σs›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil*
show *?case* **by** (*simp add*: *base-trans-det-ndet-conv*(*1*))
next
case (*single* $\sigma 1$ $A1$)
show *?case* **by** (*simp add*: *from-det-to-ndet-*σ*-*σs*-conv-commute*)
next
case (*Cons* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ 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*:
‹*length* σs = *length* $As \Longrightarrow$
($\bigwedge i\ j$. $[\![i <$ *length* As; $j <$ *length* As; $i \neq j]\!]$
$\Longrightarrow$ *indep-enabl* (As ! i) (σs ! i) E (As ! j) (σs ! j)) $\Longrightarrow$
τ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ $As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ σs e = τ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *map* (λA. $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$) $As\rangle\!\rangle$ σs e›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil*
show *?case* **by** *simp*
next
case (*single* $\sigma 1$ $A1$)
show *?case* **by** (*simp add*: *from-det-to-ndet-*σ*-*σs*-conv-commute*(*1*))
next
case (*Cons* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ As)
have $*$: ‹τ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ $A2$ # $As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ ($\sigma 2$ # σs) e =
τ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *map* (λA. $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$) ($A2$ # As)$\rangle\!\rangle$ ($\sigma 2$ # σs) e›

proof (*rule Cons.hyps*(*3*))
show ‹$[\![i < length\ (A2\ \#\ As);\ j < length\ (A2\ \#\ As);\ i \neq j]\!]$
$\Longrightarrow$ *indep-enabl* $((A2\ \#\ As)\ !\ i)\ ((\sigma 2\ \#\ \sigma s)\ !\ i)\ E$
$((A2\ \#\ As)\ !\ j)\ ((\sigma 2\ \#\ \sigma s)\ !\ j)$› **for** *i j*
using *Cons.prems*[*of* ‹*Suc i*› ‹*Suc j*›] **by** *simp*
qed
have ‹$\tau\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ A1\ \#\ A2\ \#\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ (\sigma 1\ \#\ \sigma 2\ \#\ \sigma s)\ e =$
$\tau\ \langle\!\langle\langle\!\langle A1\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ A2\ \#\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\ (\sigma 1\ \#\ \sigma 2\ \#\ \sigma s)$
e›
proof (*subst iterated-combine*$_d$*-Sync.simps*(*3*), *rule* τ*-combine*$_{Rlist}$*-Sync-behaviour-when-indep*)
show ‹$\varepsilon\ A1\ \sigma 1 \cap \varepsilon\ \langle\!\langle_d\bigotimes [\![E]\!]\ A2\ \#\ As\rangle\!\rangle\ (\sigma 2\ \#\ \sigma s) \subseteq E$›
proof (*rule indep-enablD*[*OF* - $\mathcal{R}_d$*.init* $\mathcal{R}_d$*.init*])
show ‹*indep-enabl* $A1\ \sigma 1\ E\ \langle\!\langle_d\bigotimes [\![E]\!]\ A2\ \#\ As\rangle\!\rangle\ (\sigma 2\ \#\ \sigma s)$›
by (*simp add*: *Cons.hyps*(*1*) *Cons.prems order-le-less-trans*
same-length-indep-transmission-to-iterated-combine$_d$*-Sync*)
qed
qed
also have ‹$\ldots = \tau\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ map\ (\lambda A.\ \langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd})\ (A1\ \#\ A2\ \#\ As)\rangle\!\rangle\ (\sigma 1\ \#$
$\sigma 2\ \#\ \sigma s)\ e$›
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*: ‹$length\ \sigma s = length\ As$›
and *indep*: ‹$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
indep-enabl $(As\ !\ i)\ (\sigma s\ !\ i)\ E\ (As\ !\ j)\ (\sigma s\ !\ j)$›
shows ‹$P_{SKIPS}\langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_d\ \sigma s = P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]\ map\ (\lambda A.\ \langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd})$
$As\rangle\!\rangle\rangle\!\rangle_{nd}\ \sigma s$›
proof (*fold* P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*, *rule* P_{SKIPS}*-nd-eqI-strong-id*)
show ‹$\sigma s' \in \mathcal{R}_{nd}\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ \sigma s \Longrightarrow$
$\tau\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ map\ (\lambda A.\ \langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd})\ As\rangle\!\rangle\ \sigma s'\ e = \tau\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ \sigma s'$
e› **for** $\sigma s'\ e$
proof (*rule* τ*-iterated-combine*$_{nd}$*-Sync-behaviour-when-indep*[*symmetric*])
show ‹$\sigma s' \in \mathcal{R}_{nd}\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ \sigma s \Longrightarrow length\ \sigma s' = length\ As$›
by (*metis* $\mathcal{R}_{nd}$*-from-det-to-ndet same-length same-length-*$\mathcal{R}_d$*-iterated-combine*$_d$*-Sync-description*)
next
show ‹$[\![\sigma s' \in \mathcal{R}_{nd}\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ \sigma s;\ i < length\ As;\ j < length\ As;\ i \neq$
$j]\!]$
$\Longrightarrow$ *indep-enabl* $(As\ !\ i)\ (\sigma s'\ !\ i)\ E\ (As\ !\ j)\ (\sigma s'\ !\ j)$› **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 ‹$\sigma s' \in \mathcal{R}_{nd}\ \langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\ \sigma s \Longrightarrow$

ω $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *map* $(\lambda A.\ \langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd})$ *As*$\rangle\!\rangle$ $\sigma s' = \omega$ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $\sigma s'$›
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*: ‹*length* σs = *length As*›
and *indep*: ‹$\bigwedge i\ j.\ [\![i < \textit{length As};\ j < \textit{length As};\ i \neq j]\!] \Longrightarrow$
indep-enabl (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)›
shows ‹$P\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d}$ $\sigma s = P\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *map* $(\lambda A.\ \langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd})$ *As*$\rangle\!\rangle\rangle\!\rangle_{nd}$ σs›
proof (*fold P-nd-from-det-to-ndet-is-P-d*, *rule P-nd-eqI-strong-id*)
show ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$
τ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *map* $(\lambda A.\ \langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd})$ *As*$\rangle\!\rangle$ $\sigma s'$ *e* = τ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $\sigma s'$ *e*› **for** $\sigma s'$ *e*
proof (*rule* τ*-iterated-combine*$_{nd}$*-Sync-behaviour-when-indep*[*symmetric*])
show ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$ *length* $\sigma s'$ = *length As*›
by (*metis* $\mathcal{R}_{nd}$*-from-det-to-ndet same-length same-length-*$\mathcal{R}_d$*-iterated-combine*$_d$*-Sync-description*)
next
show ‹$[\![\sigma s' \in \mathcal{R}_{nd}$ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ σs; *i* < *length As*; *j* < *length As*; *i* $\neq$ *j*$]\!]$
$\Longrightarrow$ *indep-enabl* (*As* ! *i*) ($\sigma s'$! *i*) *E* (*As* ! *j*) ($\sigma s'$! *j*)› **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_{SKIPS}*-nd-combine*$_{Pair}$*-Sync* :
fixes *E* :: ‹*'a set*› **and** A_0 :: ‹($'\sigma$, *'a*, *'r*, $'\alpha$) A_{nd}*-scheme*›
assumes ϱ*-disjoint-*ε : ‹ϱ*-disjoint-*ε A_0› ‹ϱ*-disjoint-*ε A_1›
and *at-most-1-elem-term* : ‹*at-most-1-elem-term* A_0› ‹*at-most-1-elem-term* A_1›
defines *A-def*: ‹$A \equiv \langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Pair}\ A_1\rangle\!\rangle$›
defines *P-def*: ‹$P \equiv P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}$› **and** *Q-def*: ‹$Q \equiv P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}$› **and** *S-def*: ‹$S \equiv P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$›
shows ‹$P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1 = S\ (\sigma_0, \sigma_1)$›
proof −
let *?f* = ‹P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) ($\lambda\sigma'$. *case* σ' *of* (σ_0, σ_1) $\Rightarrow$ $P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1$)›
note *cartprod-rwrt* = *GlobalNdet-cartprod*[*of* - - ‹$\lambda x\ y$. - (x, y)›, *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_{SKIPS}*-nd-step-constructive-bis*[*of* A_0], *folded* P_{SKIPS}*-nd-def P-def*, *THEN fun-cong*]
note *Q-rec* = *restriction-fix-eq*[*OF* P_{SKIPS}*-nd-step-constructive-bis*[*of* A_1], *folded* P_{SKIPS}*-nd-def Q-def*, *THEN fun-cong*]
have ω*-A* : ‹ω A $(\sigma_0{}', \sigma_1{}')$ = ω A_0 $\sigma_0{}'$ $\cap$ ω A_1 $\sigma_1{}'$› **for** $\sigma_0{}'$ $\sigma_1{}'$
by (*auto simp add*: *A-def* $combine_{Pair}$*-Sync-defs*)
have ε*-A* : ‹ε A $(\sigma_0{}', \sigma_1{}')$ = *combine-sets-Sync* (ε A_0 $\sigma_0{}'$) E (ε A_1 $\sigma_1{}'$)› **for** $\sigma_0{}'$ $\sigma_1{}'$
by (*simp add*: *A-def* ε-$combine_{Pair}$*-Sync combine-Sync-*ε*-def*)
have *Mprefix-Sync-Mprefix-for-procomata* :
‹$\Box a{\in}A \to P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b$ =
$(\Box a{\in}(A - E - B) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b))$ $\Box$
$(\Box b{\in}(B - E - A) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b))$ $\Box$
$(\Box x{\in}(A \cap B - E) \to (P\ x\ [\![E]\!]\ \Box b{\in}B \to Q\ b) \sqcap (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ x))$ $\Box$
$(\Box x{\in}(A \cap B \cap E) \to (P\ x\ [\![E]\!]\ Q\ x))$› (**is** *?eq*) **for** A B **and** P Q :: ‹$'a \Rightarrow ('a, 'r)$ $process_{ptick}$›
proof −
have $*$: ‹$\Box a{\in}(A - E) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b)$ =
$(\Box a{\in}(A - E - B) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b))$ $\Box$
$(\Box a{\in}(A \cap B - E) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b))$›
by (*metis Diff-Int2 Diff-Int-distrib2 Mprefix-Un-distrib Un-Diff-Int*)
have $**$: ‹$\Box b{\in}(B - E) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b)$ =
$(\Box b{\in}(B - E - A) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b))$ $\Box$
$(\Box b{\in}(A \cap B - E) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b))$›
by (*metis* (*no-types*) *Int-Diff Int-commute Mprefix-Un-distrib Un-Diff-Int*)
have ‹$\Box a{\in}A \to P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b$ =
$(\Box a{\in}(A - E - B) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b))$ $\Box$
$(\Box b{\in}(B - E - A) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b))$ $\Box$
$((\Box a{\in}(A \cap B - E) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b))$ $\Box$
$(\Box b{\in}(A \cap B - E) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b)))$ $\Box$
$(\Box x{\in}(A \cap B \cap E) \to (P\ x\ [\![E]\!]\ Q\ x))$›
unfolding *Mprefix-Sync-Mprefix*
by (*auto simp add*: $**$ *Det-assoc intro*!: *arg-cong*[**where** f = ‹$\lambda P.\ P\ \Box$ -›])
(*subst* (*3*) *Det-commute*, *subst Det-assoc*,
auto simp add: $*$ *Det-commute intro*: *arg-cong*[**where** f = ‹$\lambda P.\ P\ \Box$ -›])
also have ‹$(\Box a{\in}(A \cap B - E) \to (P\ a\ [\![E]\!]\ \Box b{\in}B \to Q\ b))$ $\Box$
$(\Box b{\in}(A \cap B - E) \to (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ b))$ =
$\Box x{\in}(A \cap B - E) \to ((P\ x\ [\![E]\!]\ \Box b{\in}B \to Q\ b)) \sqcap (\Box a{\in}A \to P\ a\ [\![E]\!]\ Q\ x)$›
by (*simp add*: *Mprefix-Det-Mprefix*, *rule mono-Mprefix-eq*, *simp*)
finally show *?thesis* **.**
qed
show ‹$P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1 = S\ (\sigma_0, \sigma_1)$›
proof (*rule fun-cong*[*of* ‹$\lambda(\sigma_0, \sigma_1).\ P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1$› - ‹$(\sigma_0, \sigma_1)$›, *simplified*])
show ‹$(\lambda(\sigma_0, \sigma_1).\ P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1) = S$›
proof (*rule restriction-fix-unique*[*OF* P_{SKIPS}*-nd-step-constructive-bis*[*of A*],

symmetric, *folded* P_{SKIPS}*-nd-def S-def*])
show ‹*?f* = (λ(σ₀, σ₁). *P* σ₀ ⟦*E*⟧ *Q* σ₁)›
proof (*rule ext*, *clarify*)
have *ϱ-disjoint-ε-bis* : ‹ω *A*₀ σ₀ ≠ {} ⟹ ε *A*₀ σ₀ = {}›
‹ω *A*₁ σ₁ ≠ {} ⟹ ε *A*₁ σ₁ = {}› **for** σ₀ σ₁
by (*simp-all add*: *ϱ-simps ϱ-disjoint-εD ϱ-disjoint-ε*)
show ‹*?f* (σ₀, σ₁) = *P* σ₀ ⟦*E*⟧ *Q* σ₁› **for** σ₀ σ₁
proof (*cases* ‹ω *A*₀ σ₀ = {}›; *cases* ‹ω *A*₁ σ₁ = {}›)
assume ‹ω *A*₀ σ₀ = {}› ‹ω *A*₁ σ₁ = {}›
hence *P-rec′* : ‹*P* σ₀ = *P-nd-step* (ε *A*₀) (τ *A*₀) *P* σ₀›
and *Q-rec′* : ‹*Q* σ₁ = *P-nd-step* (ε *A*₁) (τ *A*₁) *Q* σ₁›
and *S-rec′* : ‹P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) (λ(σ₀, σ₁). *P* σ₀ ⟦*E*⟧ *Q* σ₁) (σ₀, σ₁) =
P-nd-step (ε *A*) (τ *A*) (λ(σ₀, σ₁). *P* σ₀ ⟦*E*⟧ *Q* σ₁) (σ₀, σ₁)›
by (*simp-all add*: *P-rec*[*of* σ₀] *Q-rec*[*of* σ₁] *ω-A*)
show ‹*?f* (σ₀, σ₁) = *P* σ₀ ⟦*E*⟧ *Q* σ₁›
unfolding *P-rec′ Q-rec′ S-rec′ Mprefix-Sync-Mprefix-for-procomata*
unfolding *ε-A Mprefix-Un-distrib*
by (*intro arg-cong2*[**where** *f* = ‹(□)›] *mono-Mprefix-eq*, *fold P-rec′ Q-rec′*,
auto simp add: *A-def Ndet-and-Sync cartprod-rwrt*
$combine_{Pair}$*-Sync-defs ε-simps GlobalNdet-sets-commute*[*of* ‹τ *A*₁ - -›]
simp flip: *GlobalNdet-factorization-union*
intro!: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹(⊓)›])
next
assume ‹ω *A*₀ σ₀ ≠ {}› ‹ω *A*₁ σ₁ = {}›
from *ϱ-disjoint-ε*(*1*) ‹ω *A*₀ σ₀ ≠ {}› **have** ‹ε *A*₀ σ₀ = {}›
by (*simp add*: *ϱ-disjoint-ε-def ϱ-simps*)
have ‹*?f* (σ₀, σ₁) = □*b*∈(ε *A*₁ σ₁ − *E*) → (⊓σ₁′∈τ *A*₁ σ₁ *b*. (*SKIPS* (ω *A*₀ σ₀) ⟦*E*⟧ *Q* σ₁′))›
by (*auto simp add*: *ω-A ε-A* ‹ω *A*₁ σ₁ = {}› ‹ε *A*₀ σ₀ = {}› *intro*!: *mono-Mprefix-eq*,
auto simp add: *A-def* $combine_{Pair}$*-Sync-defs* ‹ω *A*₀ σ₀ ≠ {}›
‹ε *A*₀ σ₀ = {}› *cartprod-rwrt P-rec*[*of* σ₀] *intro*!: *mono-GlobalNdet-eq*)
also have ‹... = *P* σ₀ ⟦*E*⟧ *Q* σ₁›
by (*unfold P-rec*[*of* σ₀] *Q-rec*[*of* σ₁])
(*auto simp add*: *SKIPS-def Ndet-and-Sync Mprefix-Sync-constant* ‹ω *A*₀ σ₀ ≠ {}›
GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[*of* ‹τ *A*₁ - -›] ‹ω *A*₁ σ₁ = {}›
intro!: *mono-Mprefix-eq mono-GlobalNdet-eq*)
finally show ‹*?f* (σ₀, σ₁) = ...› **.**
next
assume ‹ω *A*₀ σ₀ = {}› ‹ω *A*₁ σ₁ ≠ {}›
from *ϱ-disjoint-ε*(*2*) ‹ω *A*₁ σ₁ ≠ {}› **have** ‹ε *A*₁ σ₁ = {}›
by (*simp add*: *ϱ-disjoint-ε-def ϱ-simps*)
have ‹*?f* (σ₀, σ₁) = □*a*∈(ε *A*₀ σ₀ − *E*) → (⊓σ₀′∈τ *A*₀ σ₀ *a*. (*P* σ₀′ ⟦*E*⟧ *SKIPS* (ω *A*₁ σ₁)))›
by (*auto simp add*: *ω-A ε-A* ‹ω *A*₀ σ₀ = {}› ‹ε *A*₁ σ₁ = {}› *intro*!:

mono-Mprefix-eq,
auto simp add: *A-def combine*$_{Pair}$*-Sync-defs* ‹ω A_1 $\sigma_1 \neq \{\}$›
‹ε A_1 $\sigma_1 = \{\}$› *cartprod-rwrt Q-rec*[*of* σ_1] *intro*!: *mono-GlobalNdet-eq*)
also have ‹$\ldots = P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1$›
by (*unfold P-rec*[*of* σ_0] *Q-rec*[*of* σ_1])
(*auto simp add*: *SKIPS-def Ndet-and-Sync Mprefix-Sync-constant* ‹ω A_1 $\sigma_1 \neq \{\}$›
GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[*of* ‹τ A_0 - -›] ‹ω A_0 $\sigma_0 = \{\}$›
intro!: *mono-Mprefix-eq mono-GlobalNdet-eq*)
finally show ‹*?f* $(\sigma_0, \sigma_1) = \ldots$› **.**
next
assume ‹ω A_0 $\sigma_0 \neq \{\}$› ‹ω A_1 $\sigma_1 \neq \{\}$›
with *ϱ-disjoint-ε* **have** ‹ε A_0 $\sigma_0 = \{\}$› ‹ε A_1 $\sigma_1 = \{\}$›
by (*simp-all add*: *ϱ-disjoint-ε-def ϱ-simps*)
from *at-most-1-elem-term*
show ‹ω A_0 $\sigma_0 \neq \{\} \Longrightarrow \omega$ A_1 $\sigma_1 \neq \{\} \Longrightarrow$ *?f* $(\sigma_0, \sigma_1) = P\ \sigma_0\ [\![E]\!]\ Q\ \sigma_1$›
by (*auto simp add*: ‹ω A_0 $\sigma_0 \neq \{\}$› ‹ω A_1 $\sigma_1 \neq \{\}$› *ω-A P-rec*[*of* σ_0] *Q-rec*[*of* σ_1]
SKIPS-def Ndet-and-Sync cartprod-rwrt GlobalNdet-sets-commute[*of* ‹ω A_0 -›]
ε-A ‹ε A_0 $\sigma_0 = \{\}$› ‹ε A_1 $\sigma_1 = \{\}$› *elim*!: *at-most-1-elem-termE*)
qed
qed
qed
qed
qed

corollary *P-nd-combine*$_{Pair}$*-Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0, \sigma_1)$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp add*: P_{SKIPS}*-nd-updated-ω*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle\!\rangle_{nd}$ (σ_0, σ_1)›
by (*rule* P_{SKIPS}*-nd-combine*$_{Pair}$*-Sync*) *simp-all*
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)$›
by (*simp add*: *combine*$_{Pair}$*-Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd} = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\!\rangle_{nd}$›
by (*simp add*: P_{SKIPS}*-nd-updated-ω*)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}*-d-combine*$_{Pair}$*-Sync*:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\!\rangle_d\ (\sigma_0, \sigma_1)$›

if ‹ϱ-*disjoint*-ε A_0› **and** ‹ϱ-*disjoint*-ε A_1› **and** ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›
proof −
have ‹$P_{SKIPS}\langle\langle A_0\rangle\rangle_d$ σ_0 $[\![E]\!]$ $P_{SKIPS}\langle\langle A_1\rangle\rangle_d$ σ_1 = $P_{SKIPS}\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}$ σ_0 $[\![E]\!]$ $P_{SKIPS}\langle\langle\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}$ σ_1›
by (*simp flip*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also from *that*(*1*, *2*) **have** ‹... = $P_{SKIPS}\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pair}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle\rangle_{nd}$ (σ_0, σ_1)›
by (*auto intro*: P_{SKIPS}-*nd-combine*$_{Pair}$-*Sync*)
also have ‹... = $P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle\rangle\rangle_d$ (σ_0, σ_1)›
by (*simp add*: P_{SKIPS}-*combine*$_{Pair}$-*Sync-behaviour-when-indep* ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›)
finally show *?thesis* **.**
qed

corollary *P-d-combine*$_{Pair}$-*Sync*:
‹$P\langle\langle A_0\rangle\rangle_d$ σ_0 $[\![E]\!]$ $P\langle\langle A_1\rangle\rangle_d$ σ_1 = $P\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle\rangle\rangle_d$ (σ_0, σ_1)›
if ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›
proof −
have ‹$P\langle\langle A_0\rangle\rangle_d$ σ_0 $[\![E]\!]$ $P\langle\langle A_1\rangle\rangle_d$ σ_1 =
$P_{SKIPS}\langle\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle_d$ σ_0 $[\![E]\!]$ $P_{SKIPS}\langle\langle A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle_d$ σ_1›
by (*simp add*: P_{SKIPS}-*d-updated-*ω)
also have ‹... = $P_{SKIPS}\langle\langle\langle\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes[\![E]\!]_{Pair}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle\rangle\rangle_d$ (σ_0, σ_1)›
by (*subst* P_{SKIPS}-*d-combine*$_{Pair}$-*Sync*, *simp-all add*: ϱ-*simps* ϱ-*disjoint*-ε-*def*)
(*rule indep-enablI*,
use ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›[*THEN indep-enablD*] **in** ‹*simp add*: ε-*simps*›)
also have ‹$\langle\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes[\![E]\!]_{Pair}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle$ = $\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)$›
by (*simp add*: *combine*$_{Pair}$-*Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle_d$ = $P\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle\rangle\rangle_d$›
by (*simp add*: P_{SKIPS}-*d-updated-*ω)
finally show *?thesis* **.**
qed

Pairlist

theorem P_{SKIPS}-*nd-combine*$_{Pairlist}$-*Sync* :
‹$P_{SKIPS}\langle\langle A_0\rangle\rangle_{nd}$ σ_0 $[\![E]\!]$ $P_{SKIPS}\langle\langle A_1\rangle\rangle_{nd}$ σ_1 = $P_{SKIPS}\langle\langle\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\rangle\rangle\rangle_{nd}$ $[\sigma_0, \sigma_1]$›
if ‹ϱ-*disjoint*-ε A_0› ‹ϱ-*disjoint*-ε A_1› ‹*at-most-1-elem-term* A_0› ‹*at-most-1-elem-term* A_1›
proof −
from P_{SKIPS}-*nd-combine*$_{Pair}$-*Sync that*
have ‹$P_{SKIPS}\langle\langle A_0\rangle\rangle_{nd}$ σ_0 $[\![E]\!]$ $P_{SKIPS}\langle\langle A_1\rangle\rangle_{nd}$ σ_1 = $P_{SKIPS}\langle\langle\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1\rangle\rangle\rangle\rangle_{nd}$ (σ_0, σ_1)› **.**
also have ‹... = $P_{SKIPS}\langle\langle\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\rangle\rangle\rangle_{nd}$ $[\sigma_0, \sigma_1]$›
by (*auto intro*!: P_{SKIPS}-*nd-eqI-strong*[*of* ‹$\lambda(r, s).\ [r, s]$› - ‹(σ_0, σ_1)›, *simplified*] *inj-onI*)

(*auto simp add*: *combine-Sync-defs split*: *if-split-asm*)
finally show *?thesis* **.**
qed

corollary P-*nd-combine*$_{Pairlist}$-*Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0,\ \sigma_1]$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp only*: P_{SKIPS}-*nd-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0,\ \sigma_1]$›
by (*rule* P_{SKIPS}-*nd-combine*$_{Pairlist}$-*Sync*) *simp-all*
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)$›
by (*simp add*: *combine*$_{Pairlist}$-*Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd} = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}$›
by (*simp only*: P_{SKIPS}-*nd-updated-*ω)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}-*d-combine*$_{Pairlist}$-*Sync* :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
if ‹ϱ-*disjoint-*ε A_0› **and** ‹ϱ-*disjoint-*ε A_1› **and** ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp flip*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also from *that*(*1*, *2*) **have** ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0,\ \sigma_1]$›
by (*auto intro*: P_{SKIPS}-*nd-combine*$_{Pairlist}$-*Sync*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
by (*simp add*: P_{SKIPS}-*combine*$_{Pairlist}$-*Sync-behaviour-when-indep* ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›)
finally show *?thesis* **.**
qed

corollary P-*d-combine*$_{Pairlist}$-*Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
if ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_1$›
by (*simp only*: P_{SKIPS}-*d-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
by (*subst* P_{SKIPS}-*d-combine*$_{Pairlist}$-*Sync*, *simp-all*)

(*rule indep-enablI*,
use ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›[*THEN indep-enablD*] **in** *simp*)
also have ‹$\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes [\![E]\!]_{Pairlist}\ A_1 (\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \Diamond|\!)$›
by (*simp add*: *combine*$_{Pairlist}$*-Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d = P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d$›
by (*simp only*: P_{SKIPS}*-d-updated-ω*)
finally show *?thesis* **.**
qed

6.2.2 Rlist

theorem P_{SKIPS}*-nd-combine*$_{Rlist}$*-Sync* :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma_1)$›
if ‹*ϱ-disjoint-ε* A_0› ‹*ϱ-disjoint-ε* A_1› ‹*at-most-1-elem-term* A_0› ‹*at-most-1-elem-term* A_1›
proof −
from P_{SKIPS}*-nd-combine*$_{Pair}$*-Sync that*
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma_1)$› **.**
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma_1)$›
by (*auto intro*!: P_{SKIPS}*-nd-eqI-strong*[*of* ‹$\lambda(r,\ s).\ r\ \#\ s$› - ‹$(\sigma_0,\ \sigma_1)$›, *simplified*] *inj-onI*)
(*auto simp add*: *combine-Sync-defs split*: *if-split-asm*)
finally show *?thesis* **.**
qed

corollary *P-nd-combine*$_{Rlist}$*-Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma_1)$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp only*: P_{SKIPS}*-nd-updated-ω*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma_1)$›
by (*rule* P_{SKIPS}*-nd-combine*$_{Rlist}$*-Sync*) *simp-all*
also have ‹$\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \{\}|\!)$›
by (*simp add*: *combine*$_{Rlist}$*-Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd} = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}$›
by (*simp only*: P_{SKIPS}*-nd-updated-ω*)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}*-d-combine*$_{Rlist}$*-Sync* :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0$

$\# \ \sigma_1)$›
if ‹ϱ-*disjoint*-ε A_0› **and** ‹ϱ-*disjoint*-ε A_1› **and** ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d \ \sigma_0 \ [\![E]\!] \ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d \ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd} \ \sigma_0 \ [\![E]\!] \ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd} \ \sigma_1$›
by (*simp flip*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also from *that*(*1*, *2*) **have** ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd} \ {}_{nd}\otimes[\![E]\!]_{Rlist} \ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\rangle\!\rangle_{nd} \ (\sigma_0 \ \# \ \sigma_1)$›
by (*auto intro*: P_{SKIPS}-*nd-combine*$_{Rlist}$-*Sync*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0 \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1\rangle\!\rangle\rangle\!\rangle_d \ (\sigma_0 \ \# \ \sigma_1)$›
by (*simp add*: P_{SKIPS}-*combine*$_{Rlist}$-*Sync-behaviour-when-indep* ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›)
finally show *?thesis* **.**
qed

corollary *P-d-combine*$_{Rlist}$-*Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_d \ \sigma_0 \ [\![E]\!] \ P\langle\!\langle A_1\rangle\!\rangle_d \ \sigma_1 = P\langle\!\langle\langle\!\langle A_0 \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1\rangle\!\rangle\rangle\!\rangle_d \ (\sigma_0 \ \# \ \sigma_1)$›
if ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_d \ \sigma_0 \ [\![E]\!] \ P\langle\!\langle A_1\rangle\!\rangle_d \ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma. \ \Diamond|\!)\rangle\!\rangle_d \ \sigma_0 \ [\![E]\!] \ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma. \ \Diamond|\!)\rangle\!\rangle_d \ \sigma_1$›
by (*simp only*: P_{SKIPS}-*d-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma. \ \Diamond|\!) \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1(\!|\omega := \lambda\sigma. \ \Diamond|\!)\rangle\!\rangle\rangle\!\rangle_d \ (\sigma_0 \ \# \ \sigma_1)$›
by (*subst* P_{SKIPS}-*d-combine*$_{Rlist}$-*Sync*, *simp-all*)
(*rule indep-enablI*,
use ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›[*THEN indep-enablD*] **in** *simp*)
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma. \ \Diamond|\!) \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1(\!|\omega := \lambda\sigma. \ \Diamond|\!)\rangle\!\rangle = \langle\!\langle A_0 \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma. \ \Diamond|\!)$›
by (*simp add*: *combine*$_{Rlist}$-*Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0 \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma. \ \Diamond|\!)\rangle\!\rangle_d = P\langle\!\langle\langle\!\langle A_0 \ {}_d\otimes[\![E]\!]_{Rlist} \ A_1\rangle\!\rangle\rangle\!\rangle_d$›
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* : ‹$\bigwedge\sigma_0'. \ \sigma_0' \in \mathcal{R}_{nd} \ A_0 \ \sigma_0 \Longrightarrow$ *length* $\sigma_0' = len_0$›
and ϱ-*disjoint*-ε : ‹ϱ-*disjoint*-ε A_0› ‹ϱ-*disjoint*-ε A_1› ‹*at-most-1-elem-term* A_0› ‹*at-most-1-elem-term* A_1›
shows ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd} \ \sigma_0 \ [\![E]\!] \ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd} \ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0 \ {}_{nd}\otimes[\![len_0, E]\!]_{ListslenL} \ A_1\rangle\!\rangle\rangle\!\rangle_{nd} \ (\sigma_0 \ @ \ \sigma_1)$›
proof −
from *set-mp*[*OF* $\mathcal{R}_{nd}$-*combine*$_{ndPair}$-*Sync-subset*]
have $*$: ‹$\sigma' \in \mathcal{R}_{nd} \ \langle\!\langle A_0 \ {}_{nd}\otimes[\![E]\!]_{Pair} \ A_1\rangle\!\rangle \ (\sigma_0, \sigma_1) \Longrightarrow$
$\exists \sigma_0' \ \sigma_1'. \ \sigma' = (\sigma_0', \sigma_1') \wedge \sigma_0' \in \mathcal{R}_{nd} \ A_0 \ \sigma_0 \wedge \sigma_1' \in \mathcal{R}_{nd} \ A_1 \ \sigma_1$› **for** σ'
by *fast*

from P_{SKIPS}*-nd-combine*$_{Pair}$*-Sync assms*(*2*−*5*)
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma_1)$› **.**
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*auto intro*!: P_{SKIPS}*-nd-eqI-strong*[*of* ‹$\lambda(r,\ s).\ r\ @\ s$› - ‹$(\sigma_0,\ \sigma_1)$›, *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 SigmaI UnCI case-prod-conv insertI1*)+)
finally show *?thesis* **.**
qed

corollary *P-nd-combine*$_{ListslenL}$*-Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
if ‹$\bigwedge\sigma_0'.\ \sigma_0' \in \mathcal{R}_{nd}\ A_0\ \sigma_0 \Longrightarrow length\ \sigma_0' = len_0$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp only*: P_{SKIPS}*-nd-updated-ω*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*rule* P_{SKIPS}*-nd-combine*$_{ListslenL}$*-Sync*) (*simp-all add*: *that*)
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)$›
by (*simp add*: *combine*$_{ListslenL}$*-Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd} = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\!\rangle_{nd}$›
by (*simp only*: P_{SKIPS}*-nd-updated-ω*)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}*-d-combine*$_{ListslenL}$*-Sync* :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{d}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
if ‹$\bigwedge\sigma_0'.\ \sigma_0' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow length\ \sigma_0' = len_0$›
‹*ϱ-disjoint-ε* A_0› ‹*ϱ-disjoint-ε* A_1› ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp flip*: P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*)
also from *that*(*1*−*3*) **have** ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*auto simp add*: $\mathcal{R}_{nd}$*-from-det-to-ndet intro*!: P_{SKIPS}*-nd-combine*$_{ListslenL}$*-Sync*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{d}\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
by (*simp add*: P_{SKIPS}*-combine*$_{ListslenL}$*-Sync-behaviour-when-indep that*(*1*, *4*))
finally show *?thesis* **.**
qed

corollary *P-d-combine*$_{ListslenL}$*-Sync* :
‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
if ‹$\bigwedge\sigma_0'.\ \sigma_0' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow length\ \sigma_0' = len_0$› ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_1$›
by (*simp only*: P_{SKIPS}*-d-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
by (*subst* P_{SKIPS}*-d-combine*$_{ListslenL}$*-Sync*) (*simp-all add*: *that*)
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)$›
by (*simp add*: *combine*$_{ListslenL}$*-Sync-defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d = P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d$›
by (*simp only*: P_{SKIPS}*-d-updated-*ω)
finally show *?thesis* **.**
qed

6.2.4 Multiple

theorem P_{SKIPS}*-nd-compactification-Sync*:
‹$[\![length\ \sigma s = length\ As;\ \bigwedge A.\ A \in set\ As \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A;$
$\bigwedge A.\ A \in set\ As \Longrightarrow at\text{-}most\text{-}1\text{-}elem\text{-}term\ A]\!]$
$\Longrightarrow [\![E]\!]\ (\sigma,\ A) \in\#\ mset\ (zip\ \sigma s\ As).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{nd}\ \sigma s$›
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*: 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 ‹$[\![E]\!]\ (\sigma,\ A) \in\#\ mset\ (zip\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)\ (A_0\ \#\ A_1\ \#\ As)).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma =$
$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ [\![E]\!]\ (\sigma,\ A) \in\#\ mset\ (zip\ (\sigma_1\ \#\ \sigma s)\ (A_1\ \#\ As)).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$› **by** *simp*
also have ‹$[\![E]\!]\ (\sigma,\ A) \in\#\ mset\ (zip\ (\sigma_1\ \#\ \sigma s)\ (A_1\ \#\ As)).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma =$
$P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_1\ \#\ \sigma s)$›
by (*rule Cons.hyps*(*3*)) (*simp-all add*: *Cons.prems*)
also have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]\ P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_1\ \#\ \sigma s) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)$›
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.prems*)
also have ‹$\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{Rlist}\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle = \langle\!\langle_{nd}\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#$

As⟫› **by** *simp*
finally show *?case* **.**
qed

lemma *P-nd-compactification-Sync*:
‹*length* σs = *length As* $\Longrightarrow$
$[\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = P\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_{nd}\ \sigma s$›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst P-nd-rec*, *simp*)
next
case (*single* σ_0 A_0) **show** *?case*
by (*simp add*: *P-nd-from-*σ*-to-*σs*-is-P-nd*)
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 *As*) **thus** *?case*
by (*simp add*: *P-nd-combine*$_{Rlist}$*-Sync*)
qed

lemma P_{SKIPS}*-d-compactification-Sync*:
‹$[\![$*length* σs = *length As*; $\bigwedge A.\ A \in$ *set As* $\Longrightarrow$ ϱ*-disjoint-*ε *A*;
$\bigwedge i\ j.\ [\![i <$ *length As*; $j <$ *length As*; $i \neq j]\!] \Longrightarrow$
indep-enabl (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)$]\!] \Longrightarrow$
$[\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma = P_{SKIPS}\langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\rangle\!\rangle_d$
σs›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst* P_{SKIPS}*-d-rec*, *simp*)
next
case (*single* σ_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* σ_0 σ_1 σs A_0 A_1 *As*)
have ϱ*-disjoint-*ε : ‹$A \in$ *set* (A_1 # *As*) $\Longrightarrow$ ϱ*-disjoint-*ε *A*› **for** *A*
by (*simp add*: *Cons.prems*(*1*))
have *indep-enabl* :
‹$[\![i <$ *length* (A_1 # *As*); $j <$ *length* (A_1 # *As*); $i \neq j]\!] \Longrightarrow$
indep-enabl ((A_1 # *As*) ! *i*) ((σ_1 # σs) ! *i*) *E* ((A_1 # *As*) ! *j*) ((σ_1 # σs) ! *j*)›
for *i j*
by (*metis Cons.prems*(*2*) *Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)
have ‹ϱ*-disjoint-*ε A_0› **by** (*simp add*: *Cons.prems*(*1*))
moreover have ‹ϱ*-disjoint-*ε $\langle\!\langle_d\bigotimes [\![E]\!]\ A_1$ # *As*⟫›
by (*meson* ϱ*-disjoint-*ε ϱ*-disjoint-*ε*-transmission-to-iterated-combine*$_d$*-Sync*)
moreover have ‹*indep-enabl* A_0 σ_0 *E* $\langle\!\langle_d\bigotimes [\![E]\!]\ A_1$ # *As*⟫ (σ_1 # σs)›
by (*metis Cons.hyps*(*1*) *Cons.prems*(*2*) *length-Cons less-Suc-eq-le*
same-length-indep-transmission-to-iterated-combine$_d$*-Sync*)
ultimately show *?case*
by (*simp add*: P_{SKIPS}*-d-combine*$_{Rlist}$*-Sync*
Cons.hyps(*3*)[*OF* ϱ*-disjoint-*ε *indep-enabl*, *simplified*])
qed

lemma *P-d-compactification-Sync*:
‹$[\![$*length* σs = *length As*;
$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
indep-enabl (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)$]\!] \Longrightarrow$
$[\![E]\!]$ (σ, *A*) $\in\#$ *mset* (*zip* σs *As*). $P\langle\!\langle A\rangle\!\rangle_d\ \sigma = P\langle\!\langle\langle\!\langle_d \bigotimes [\![E]\!]\ As\rangle\!\rangle\!\rangle_d\ \sigma s$›
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* :
‹$[\![i < length\ (A_1\ \#\ As);\ j < length\ (A_1\ \#\ As);\ i \neq j]\!] \Longrightarrow$
indep-enabl ((A_1 # *As*) ! *i*) ((σ_1 # σs) ! *i*) *E* ((A_1 # *As*) ! *j*) ((σ_1 # σs) ! *j*)›
for *i j*
by (*metis Cons.prems Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)
have ‹*indep-enabl* A_0 σ_0 *E* $\langle\!\langle_d \bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle$ (σ_1 # σs)›
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*:
‹$P_{SKIPS}\langle\!\langle\langle\!\langle_{nd} \bigotimes [\![E]\!]\ As\rangle\!\rangle\!\rangle_{nd}\ \sigma s = P_{SKIPS}\langle\!\langle\langle\!\langle_{nd} \bigotimes [\![E]\!]\ As'\rangle\!\rangle\!\rangle_{nd}\ \sigma s'$›
if ‹*length* σs = *length As*› ‹*length* $\sigma s'$ = *length As'*›
‹*mset* (*zip* σs *As*) = *mset* (*zip* $\sigma s'$ *As'*)›
‹$\bigwedge A.\ A \in set\ As \Longrightarrow \varrho$*-disjoint-*$\varepsilon$ *A*›
‹$\bigwedge A.\ A \in set\ As \Longrightarrow$ *at-most-1-elem-term A*›
proof −
have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle_{nd} \bigotimes [\![E]\!]\ As\rangle\!\rangle\!\rangle_{nd}\ \sigma s = [\![E]\!]$ (σ, *A*)$\in\#$*mset* (*zip* σs *As*). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ›
by (*rule* P_{SKIPS}*-nd-compactification-Sync*[*OF that*(*1*, *4*, *5*), *symmetric*])
also have ‹... = $[\![E]\!]$ (σ, *A*)$\in\#$*mset* (*zip* $\sigma s'$ *As'*). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ›
by (*simp only*: *that*(*3*))
also have ‹... = $P_{SKIPS}\langle\!\langle\langle\!\langle_{nd} \bigotimes [\![E]\!]\ As'\rangle\!\rangle\!\rangle_{nd}\ \sigma s'$›
proof (*rule* P_{SKIPS}*-nd-compactification-Sync*[*OF that*(*2*)])
show ‹$A \in set\ As' \Longrightarrow \varrho$*-disjoint-*$\varepsilon$ *A*› **for** *A*
by (*metis in-set-impl-in-set-zip2 set-mset-mset set-zip-rightD that*(*2*−*4*))
next
show ‹$A \in set\ As' \Longrightarrow$ *at-most-1-elem-term A*› **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*:
‹$P\langle\langle\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\rangle\rangle_{nd}\ \sigma s = P\langle\langle\langle_{nd}\bigotimes [\![E]\!]\ As'\rangle\rangle\rangle_{nd}\ \sigma s'$›
if ‹$length\ \sigma s = length\ As$› ‹$length\ \sigma s' = length\ As'$›
‹$mset\ (zip\ \sigma s\ As) = mset\ (zip\ \sigma s'\ As')$›
proof −
have ‹$P\langle\langle\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\rangle\rangle_{nd}\ \sigma s = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ \sigma s\ As).\ P\langle\langle A\rangle\rangle_{nd}\ \sigma$›
by (*fact P-nd-compactification-Sync*[*OF that*(*1*), *symmetric*])
also have ‹$\ldots = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ \sigma s'\ As').\ P\langle\langle A\rangle\rangle_{nd}\ \sigma$›
by (*simp only*: *that*(*3*))
also have ‹$\ldots = P\langle\langle\langle_{nd}\bigotimes [\![E]\!]\ As'\rangle\rangle\rangle_{nd}\ \sigma s'$›
by (*fact P-nd-compactification-Sync*[*OF that*(*2*)])
finally show *?thesis* **.**
qed

corollary *P_{SKIPS}-d-compactification-Sync-order-is-arbitrary*:
‹$P_{SKIPS}\langle\langle\langle_{d}\bigotimes [\![E]\!]\ As\rangle\rangle\rangle_{d}\ \sigma s = P_{SKIPS}\langle\langle\langle_{d}\bigotimes [\![E]\!]\ As'\rangle\rangle\rangle_{d}\ \sigma s'$›
if ‹$length\ \sigma s = length\ As$› ‹$length\ \sigma s' = length\ As'$›
‹$mset\ (zip\ \sigma s\ As) = mset\ (zip\ \sigma s'\ As')$›
‹$\bigwedge A.\ A \in set\ As \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A$›
‹$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
$indep\text{-}enabl\ (As\ !\ i)\ (\sigma s\ !\ i)\ E\ (As\ !\ j)\ (\sigma s\ !\ j)$›
proof −
have ‹$P_{SKIPS}\langle\langle\langle_{d}\bigotimes [\![E]\!]\ As\rangle\rangle\rangle_{d}\ \sigma s = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ \sigma s\ As).\ P_{SKIPS}\langle\langle A\rangle\rangle_{d}\ \sigma$›
by (*rule P_{SKIPS}-d-compactification-Sync*[*OF that*(*1*, *4*, *5*), *symmetric*])
also have ‹$\ldots = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ \sigma s'\ As').\ P_{SKIPS}\langle\langle A\rangle\rangle_{d}\ \sigma$›
by (*simp only*: *that*(*3*))
also have ‹$\ldots = P_{SKIPS}\langle\langle\langle_{d}\bigotimes [\![E]\!]\ As'\rangle\rangle\rangle_{d}\ \sigma s'$›
proof (*rule P_{SKIPS}-d-compactification-Sync*[*OF that*(*2*)])
show ‹$A \in set\ As' \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A$› **for** *A*
by (*metis in-set-impl-in-set-zip2 set-mset-mset set-zip-rightD that*(*2*−*4*))
next
from ‹$length\ \sigma s = length\ As$› ‹$length\ \sigma s' = length\ As'$› ‹$mset\ (zip\ \sigma s\ As) = mset\ (zip\ \sigma s'\ As')$›
obtain *n* **where** $*$: ‹$length\ \sigma s = n$› ‹$length\ \sigma s' = n$› ‹$length\ As = n$› ‹$length\ As' = n$›
by (*metis length-zip min-less-iff-conj nat-neq-iff size-mset*)
from *that*(*3*)[*symmetric*, *THEN permutation-Ex-bij*] **obtain** *f*
where $**$: ‹$bij\text{-}betw\ f\ \{..<n\}\ \{..<n\}$›
‹$i < n \Longrightarrow zip\ \sigma s'\ As'\ !\ i = zip\ \sigma s\ As\ !\ f\ i$› **for** *i* **by** (*auto simp add*: $*$)
{ fix *i* **assume** ‹$i < n$›
hence ‹$f\ i < n$› **using** $**$(*1*) *bij-betwE* **by** *blast*
from ‹$i < n$› **have** ‹$zip\ \sigma s'\ As'\ !\ i = (\sigma s'\ !\ i,\ As'\ !\ i)$› **by** (*simp add*: $*$(*2*, *4*))
moreover from ‹$f\ i < n$› **have** ‹$zip\ \sigma s\ As\ !\ f\ i = (\sigma s\ !\ f\ i,\ As\ !\ f\ i)$›
by (*simp add*: $*$(*1*, *3*))

```
      ultimately have ‹σs' ! i = σs ! f i› ‹As' ! i = As ! f i›
        using **(2)[OF ‹i < n›] by simp-all
    } note *** = this
    fix i j assume ‹i < length As'› ‹j < length As'› ‹i ≠ j›
    hence ‹i < n› ‹j < n› by (simp-all add: *(2, 4))
    with bij-betw-imp-surj-on[OF **(1)] bij-betw-imp-inj-on[OF **(1)] ‹i ≠ j›
    have ‹f i < length As› ‹f j < length As› ‹f i ≠ f j›
      by (auto simp add: * dest: inj-onD)
    from that(5)[OF this]
    show ‹indep-enabl (As' ! i) (σs' ! i) E (As' ! j) (σs' ! j)›
      by (simp add: ***(1, 2) ‹i < n› ‹j < n›)
  qed
  finally show ?thesis .
qed

corollary P-d-compactification-Sync-order-is-arbitrary:
  ‹P⟪⟪d⨂ ⟦E⟧ As⟫⟫d σs = P⟪⟪d⨂ ⟦E⟧ As'⟫⟫d σs'›
  if ‹length σs = length As› ‹length σs' = length As'›
    ‹mset (zip σs As) = mset (zip σs' As')›
    ‹⋀i j. ⟦i < length As; j < length As; i ≠ j⟧ ⟹
          indep-enabl (As ! i) (σs ! i) E (As ! j) (σs ! j)›
proof −
  have ‹P⟪⟪d⨂ ⟦E⟧ As⟫⟫d σs = ⟦E⟧ (σ, A)∈#mset (zip σs As). P⟪A⟫d σ›
    by (rule P-d-compactification-Sync[OF that(1, 4), symmetric])
  also have ‹... = ⟦E⟧ (σ, A)∈#mset (zip σs' As'). P⟪A⟫d σ›
    by (simp only: that(3))
  also have ‹... = P⟪⟪d⨂ ⟦E⟧ As'⟫⟫d σs'›
  proof (rule P-d-compactification-Sync[OF that(2)])
    from ‹length σs = length As› ‹length σs' = length As'› ‹mset (zip σs As) =
mset (zip σs' As')›
    obtain n where * : ‹length σs = n› ‹length σs' = n› ‹length As = n› ‹length
As' = n›
      by (metis length-zip min-less-iff-conj nat-neq-iff size-mset)
    from that(3)[symmetric, THEN permutation-Ex-bij] obtain f
      where ** : ‹bij-betw f {..<n} {..<n}›
        ‹i < n ⟹ zip σs' As' ! i = zip σs As ! f i› for i by (auto simp add: *)
    { fix i assume ‹i < n›
      hence ‹f i < n› using **(1) bij-betwE by blast
     from ‹i < n› have ‹zip σs' As' ! i = (σs' ! i, As' ! i)› by (simp add: *(2, 4))
      moreover from ‹f i < n› have ‹zip σs As ! f i = (σs ! f i, As ! f i)›
        by (simp add: *(1, 3))
      ultimately have ‹σs' ! i = σs ! f i› ‹As' ! i = As ! f i›
        using **(2)[OF ‹i < n›] by simp-all
    } note *** = this
    fix i j assume ‹i < length As'› ‹j < length As'› ‹i ≠ j›
    hence ‹i < n› ‹j < n› by (simp-all add: *(2, 4))
    with bij-betw-imp-surj-on[OF **(1)] bij-betw-imp-inj-on[OF **(1)] ‹i ≠ j›
    have ‹f i < length As› ‹f j < length As› ‹f i ≠ f j›
      by (auto simp add: * dest: inj-onD)
```

from *that*(*4*)[*OF this*]
show ‹*indep-enabl* (*As'* ! *i*) (*σs'* ! *i*) *E* (*As'* ! *j*) (*σs'* ! *j*)›
by (*simp add*: ∗∗∗(*1*, *2*) ‹*i* < *n*› ‹*j* < *n*›)
qed
finally show *?thesis* **.**
qed

6.3 Derived Versions

lemma P_{SKIPS}*-nd-compactification-Sync-upt-version*:
‹⟦*E*⟧ *P* ∈# *mset* (*map Q* [*0*..<*n*]). *P* = P_{SKIPS}⟪⟪$_{nd}$⨂ ⟦*E*⟧ *map A* [*0*..<*n*]⟫⟫$_{nd}$ (*replicate n 0*)›
if ‹⋀*i*. *i* < *n* ⟹ *ϱ-disjoint-ε* (*A i*)›
‹⋀*i*. *i* < *n* ⟹ *at-most-1-elem-term* (*A i*)›
‹⋀*i*. *i* < *n* ⟹ P_{SKIPS}⟪*A i*⟫$_{nd}$ *0* = *Q i*›
proof −
have ‹⟦*E*⟧ *P* ∈# *mset* (*map Q* [*0*..<*n*]). *P* = ⟦*E*⟧ *i* ∈# (*mset-set* {*0*..<*n*}). *Q i*›
by (*auto intro*: *mono-MultiSync-eq2*)
also have ‹. . . = ⟦*E*⟧ *i* ∈# (*mset-set* {*0*..<*n*}). P_{SKIPS}⟪*A i*⟫$_{nd}$ *0*›
by (*auto simp add*: *that*(*3*) *intro*: *mono-MultiSync-eq*)
also have ‹. . . = ⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate n 0*) (*map A* [*0*..<*n*])). P_{SKIPS}⟪*A*⟫$_{nd}$ *σ*›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹⟦*E*⟧ *i*∈#*mset-set* {*0*..<*Suc k*}. P_{SKIPS}⟪*A i*⟫$_{nd}$ *0* =
P_{SKIPS}⟪*A k*⟫$_{nd}$ *0* ⟦*E*⟧ ⟦*E*⟧ *i*∈#*mset-set* {*0*..<*k*}. P_{SKIPS}⟪*A i*⟫$_{nd}$ *0*›
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 ‹. . . = P_{SKIPS}⟪*A k*⟫$_{nd}$ *0* ⟦*E*⟧
⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate k 0*) (*map A* [*0*..<*k*])). P_{SKIPS}⟪*A*⟫$_{nd}$ *σ*›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹. . . = ⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate* (*Suc k*) *0*) (*map A* [*0*..<*Suc k*])). P_{SKIPS}⟪*A*⟫$_{nd}$ *σ*›
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 ‹. . . = P_{SKIPS}⟪⟪$_{nd}$⨂ ⟦*E*⟧ *map A* [*0*..<*n*]⟫⟫$_{nd}$ (*replicate n 0*)›
by (*rule* P_{SKIPS}*-nd-compactification-Sync*)
(*auto simp add*: *that*(*1*, *2*))
finally show *?thesis* **.**
qed

lemma *P-nd-compactification-Sync-upt-version*:
‹⟦*E*⟧ *P* ∈# *mset* (*map Q* [*0*..<*n*]). *P* = *P*⟪⟪$_{nd}$⨂ ⟦*E*⟧ *map A* [*0*..<*n*]⟫⟫$_{nd}$ (*replicate n 0*)›

if ‹$\bigwedge i.\ i < n \Longrightarrow P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0 = Q\ i$›
proof −
have ‹$[\![E]\!]\ P \in\#\ mset\ (map\ Q\ [0..<n]).\ P = [\![E]\!]\ i \in\#\ (mset\text{-}set\ \{0..<n\}).\ Q\ i$›
by (*auto intro*: *mono-MultiSync-eq2*)
also have ‹$\ldots = [\![E]\!]\ i \in\#\ (mset\text{-}set\ \{0..<n\}).\ P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$›
by (*auto simp add*: *that*(*1*) *intro*: *mono-MultiSync-eq*)
also have ‹$\ldots = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ (replicate\ n\ 0)\ (map\ A\ [0..<n])).\ P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹$[\![E]\!]\ i\in\#mset\text{-}set\ \{0..<Suc\ k\}.\ P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0 =$
$P\langle\!\langle A\ k\rangle\!\rangle_{nd}\ 0\ [\![E]\!]\ [\![E]\!]\ i\in\#mset\text{-}set\ \{0..<k\}.\ P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$›
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 ‹$\ldots = P\langle\!\langle A\ k\rangle\!\rangle_{nd}\ 0\ [\![E]\!]\ [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ (replicate\ k\ 0)\ (map\ A\ [0..<k])).\ P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹$\ldots = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ (replicate\ (Suc\ k)\ 0)\ (map\ A\ [0..<Suc\ k])).\ P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
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 ‹$\ldots = P\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]\ map\ A\ [0..<n]\rangle\!\rangle\rangle\!\rangle_{nd}\ (replicate\ n\ 0)$›
by (*auto intro*: *P-nd-compactification-Sync*)
finally show *?thesis* **.**
qed

lemma P_{SKIPS}*-d-compactification-Sync-upt-version*:
‹$[\![E]\!]\ P \in\#\ mset\ (map\ Q\ [0..<n]).\ P = P_{SKIPS}\langle\!\langle\langle\!\langle_d\bigotimes [\![E]\!]\ map\ A\ [0..<n]\rangle\!\rangle\rangle\!\rangle_d\ (replicate\ n\ 0)$›
if ‹$\bigwedge i.\ i < n \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ (A\ i)$›
‹$\bigwedge i\ j.\ i < n \Longrightarrow j < n \Longrightarrow i \neq j \Longrightarrow indep\text{-}enabl\ (A\ i)\ 0\ E\ (A\ j)\ 0$›
‹$\bigwedge i.\ i < n \Longrightarrow P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0 = Q\ i$›
proof −
have ‹$[\![E]\!]\ P \in\#\ mset\ (map\ Q\ [0..<n]).\ P = [\![E]\!]\ i \in\#\ (mset\text{-}set\ \{0..<n\}).\ Q\ i$›
by (*auto intro*: *mono-MultiSync-eq2*)
also have ‹$\ldots = [\![E]\!]\ i \in\#\ (mset\text{-}set\ \{0..<n\}).\ P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0$›
by (*auto simp add*: *that*(*3*) *intro*: *mono-MultiSync-eq*)
also have ‹$\ldots = [\![E]\!]\ (\sigma,\ A)\in\#mset\ (zip\ (replicate\ n\ 0)\ (map\ A\ [0..<n])).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹$[\![E]\!]\ i\in\#mset\text{-}set\ \{0..<Suc\ k\}.\ P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0 =$
$P_{SKIPS}\langle\!\langle A\ k\rangle\!\rangle_d\ 0\ [\![E]\!]\ [\![E]\!]\ i\in\#mset\text{-}set\ \{0..<k\}.\ P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0$›
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 ‹... = P_{SKIPS}⟪*A k*⟫$_d$ *0* ⟦*E*⟧
⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate k 0*) (*map A* [*0*..<*k*])). P_{SKIPS}⟪*A*⟫$_d$ *σ*›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹... = ⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate* (*Suc k*) *0*) (*map A* [*0*..<*Suc k*])). P_{SKIPS}⟪*A*⟫$_d$ *σ*›
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 ‹... = P_{SKIPS}⟪⟪$_d$⨂ ⟦*E*⟧ *map A* [*0*..<*n*]⟫⟫$_d$ (*replicate n 0*)›
by (*rule* P_{SKIPS}*-d-compactification-Sync*)
(*auto simp add*: *that*(*1*, *2*))
finally show *?thesis* **.**
qed

lemma *P-d-compactification-Sync-upt-version*:
‹⟦*E*⟧ *P* ∈# *mset* (*map Q* [*0*..<*n*]). *P* = *P*⟪⟪$_d$⨂ ⟦*E*⟧ *map A* [*0*..<*n*]⟫⟫$_d$ (*replicate n 0*)›
if ‹⋀*i*. *i* < *n* ⟹ *P*⟪*A i*⟫$_d$ *0* = *Q i*›
‹⋀*i j*. *i* < *n* ⟹ *j* < *n* ⟹ *i* ≠ *j* ⟹ *indep-enabl* (*A i*) *0 E* (*A j*) *0*›
proof −
have ‹⟦*E*⟧ *P* ∈# *mset* (*map Q* [*0*..<*n*]). *P* = ⟦*E*⟧ *i* ∈# (*mset-set* {*0*..<*n*}). *Q i*›
by (*auto intro*: *mono-MultiSync-eq2*)
also have ‹... = ⟦*E*⟧ *i* ∈# (*mset-set* {*0*..<*n*}). *P*⟪*A i*⟫$_d$ *0*›
by (*auto simp add*: *that*(*1*) *intro*: *mono-MultiSync-eq*)
also have ‹... = ⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate n 0*) (*map A* [*0*..<*n*])). *P*⟪*A*⟫$_d$ *σ*›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹⟦*E*⟧ *i*∈#*mset-set* {*0*..<*Suc k*}. *P*⟪*A i*⟫$_d$ *0* =
P⟪*A k*⟫$_d$ *0* ⟦*E*⟧ ⟦*E*⟧ *i*∈#*mset-set* {*0*..<*k*}. *P*⟪*A i*⟫$_d$ *0*›
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 ‹... = *P*⟪*A k*⟫$_d$ *0* ⟦*E*⟧ ⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate k 0*) (*map A* [*0*..<*k*])). *P*⟪*A*⟫$_d$ *σ*›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹... = ⟦*E*⟧ (*σ*, *A*)∈#*mset* (*zip* (*replicate* (*Suc k*) *0*) (*map A* [*0*..<*Suc k*])). *P*⟪*A*⟫$_d$ *σ*›
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 ‹... = *P*⟪⟪$_d$⨂ ⟦*E*⟧ *map A* [*0*..<*n*]⟫⟫$_d$ (*replicate n 0*)›
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$_{nd}$-Sync-general-form*:
‹*length σs = length As* $\Longrightarrow$ *ε* ⟪$_{nd}\bigotimes$ ⟦*E*⟧ *As*⟫ *σs* =
(*if As* = [] *then* {}
else ($\bigcup$ *i* < *length As*. *ε* (*As* ! *i*) (*σs* ! *i*)) − *E* ∪ ($\bigcap$ *i* < *length As*. *ε* (*As* ! *i*) (*σs* ! *i*)))›
for *As* :: ‹('*σ*, '*e*, '*r*) A_{nd} *list*›
proof (*subst ε-iterated-combine$_{nd}$-Sync*, *assumption*, *induct σs As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** *simp*
next
case (*single* σ_0 A_0) **show** *?case* **by** *auto*
next
case (*Cons* σ_0 σ_1 *σs* A_0 A_1 *As*)
define *U* **and** *I*
where *U-def* : ‹*U As σs* ≡ $\bigcup$*i*<*length As*. *ε* (*As* ! *i*) (*σs* ! *i*)›
and *I-def* : ‹*I As σs* ≡ $\bigcap$*i*<*length As*. *ε* (*As* ! *i*) (*σs* ! *i*)›
for *As* :: ‹('*σ*, '*e*, '*r*) A_{nd} *list*› **and** *σs*
have ∗ : ‹*U* (A_0 # A_1 # *As*) (σ_0 # σ_1 # *σs*) = *ε* A_0 σ_0 ∪ *U* (A_1 # *As*) (σ_1 # *σs*)›
by (*auto simp add*: *U-def nth-Cons split*: *nat.split-asm*)
have ∗∗ : ‹*I* (A_0 # A_1 # *As*) (σ_0 # σ_1 # *σs*) = *ε* A_0 σ_0 ∩ *I* (A_1 # *As*) (σ_1 # *σs*)›
by (*auto simp add*: *I-def nth-Cons split*: *nat.splits*)
have ‹*iterated-combine$_{nd}$-Sync-ε* (A_0 # A_1 # *As*) *E* (σ_0 # σ_1 # *σs*) =
combine-sets-Sync (*ε* A_0 σ_0) *E* (*U* (A_1 # *As*) (σ_1 # *σs*) − *E* ∪ (*I* (A_1 # *As*) (σ_1 # *σs*)))›
by (*simp add*: *U-def I-def Cons.hyps*(*3*))
also have ‹... = *U* (A_0 # A_1 # *As*) (σ_0 # σ_1 # *σs*) − *E* ∪ *I* (A_0 # A_1 # *As*) (σ_0 # σ_1 # *σs*)›
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*:
‹*length σs = length As* $\Longrightarrow$ *ε* ⟪$_{d}\bigotimes$ ⟦*E*⟧ *As*⟫ *σs* =
(*if As* = [] *then* {}
else ($\bigcup$ *i* < *length As*. *ε* (*As* ! *i*) (*σs* ! *i*)) − *E* ∪ ($\bigcap$ *i* < *length As*. *ε* (*As* ! *i*) (*σs* ! *i*)))›
for *As* :: ‹('*σ*, '*e*, '*r*) A_d *list*›
proof (*subst ε-iterated-combine$_{d}$-Sync*, *assumption*, *induct σs As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** *simp*
next
case (*single* σ_0 A_0) **show** *?case* **by** *auto*
next
case (*Cons* σ_0 σ_1 *σs* A_0 A_1 *As*)

define U **and** I
where *U-def* : ‹$U\ As\ \sigma s \equiv \bigcup i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
and *I-def* : ‹$I\ As\ \sigma s \equiv \bigcap i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
for As :: ‹$('\sigma,\ 'e,\ 'r)\ A_d\ list$› **and** σs
have $*$: ‹$U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \varepsilon\ A_0\ \sigma_0 \cup U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)$›
by (*auto simp add*: *U-def nth-Cons split*: *nat.split-asm*)
have $**$: ‹$I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \varepsilon\ A_0\ \sigma_0 \cap I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)$›
by (*auto simp add*: *I-def nth-Cons split*: *nat.splits*)
have ‹$iterated\text{-}combine_d\text{-}Sync\text{-}\varepsilon\ (A_0\ \#\ A_1\ \#\ As)\ E\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) =$
$combine\text{-}sets\text{-}Sync\ (\varepsilon\ A_0\ \sigma_0)\ E\ (U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) - E \cup (I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)))$›
by (*simp add*: *U-def I-def Cons.hyps*(*3*))
also have ‹$\ldots = U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) - E \cup I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)$›
unfolding $*$ $**$ **by** (*auto simp add*: *U-def I-def*)
finally show *?case* **by** (*simp add*: *U-def I-def*)
qed

lemma τ*-iterated-combine*$_{nd}$*-Sync-general-form*:
‹$[\![length\ \sigma s = length\ As;\ \sigma s' \in \tau\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle\ \sigma s\ a;\ i < length\ As]\!] \Longrightarrow$
$\sigma s'\ !\ i \in insert\ (\sigma s\ !\ i)\ (\tau\ (As\ !\ i)\ (\sigma s\ !\ i)\ a)$›
proof (*induct* σs *As arbitrary*: $\sigma s'$ *i rule*: *induct-2-lists012*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*single* σ_0 A_0)
from *single.prems* **show** *?case* **by** (*auto simp add*: *behaviour-*σ*-*σs*-conv*)
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 As)
from ‹$length\ \sigma s = length\ As$› **have** ‹$length\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = length\ (A_0\ \#\ A_1\ \#\ As)$› **by** *simp*
from *same-length-*$\mathcal{R}_{nd}$*-iterated-combine*$_{nd}$*-Sync-description*[*OF this*]
have ‹$length\ \sigma s' = length\ (A_0\ \#\ A_1\ \#\ As)$›
by (*metis Cons.prems*(*1*) $\mathcal{R}_{nd}$*.init* $\mathcal{R}_{nd}$*.step*)
then obtain $\sigma_0'\ \sigma_1'\ \sigma s''$ **where** ‹$\sigma s' = \sigma_0'\ \#\ \sigma_1'\ \#\ \sigma s''$› **by** (*metis length-Suc-conv*)
with *Cons.prems Cons.hyps*(*3*)[*of* ‹$\sigma_1'\ \#\ \sigma s''$› ‹*nat.pred i*›] **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*:
‹$length\ \sigma s = length\ As \Longrightarrow$
$\tau\ \langle\!\langle_d\bigotimes [\![E]\!]\ As\rangle\!\rangle\ \sigma s\ a =$
$(if\ As = []\ then\ \Diamond\ else$
$if\ a \in (\bigcup i < length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)) - E \cup (\bigcap i < length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i))$
$then\ \lfloor map2\ (\lambda\sigma\ A.\ if\ a \in \varepsilon\ A\ \sigma\ then\ \lceil \tau\ A\ \sigma\ a\rceil\ else\ \sigma)\ \sigma s\ As\rfloor\ else\ \Diamond)$›

for As :: ‹$('\sigma, 'e, 'r)\ A_d\ list$›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** *simp*
next
case (*single* $\sigma_0\ A_0$) **show** *?case* **by** (*auto simp add*: *behaviour-σ-σs-conv ε-simps*)
next
case (*Cons* $\sigma_0\ \sigma_1\ \sigma s\ A_0\ A_1\ As$)
let $?U$ = ‹$\lambda As\ \sigma s.\ \bigcup i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
let $?I$ = ‹$\lambda As\ \sigma s.\ \bigcap i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
show *?case*
proof (*split if-split*, *split if-split*, *intro conjI impI*)
show ‹$A_0\ \#\ A_1\ \#\ As = [] \Longrightarrow \tau\ \langle\!\langle_d\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)\ a = \Diamond$›
and ‹$A_0\ \#\ A_1\ \#\ As = [] \Longrightarrow \tau\ \langle\!\langle_d\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)\ a = \Diamond$› **by** *simp-all*
next
assume ‹$a \notin ?U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) - E \cup ?I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)$›
hence ‹$a \notin \varepsilon\ \langle\!\langle_d\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)$›
by (*subst* ε-*iterated-combine*$_d$-*Sync-general-form*)
(*simp-all add*: *Cons.hyps*(*1*))
thus ‹$\tau\ \langle\!\langle_d\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)\ a = \Diamond$›
by (*simp add*: *ε-simps*)
next
assume $*$: ‹$a \in ?U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) - E \cup ?I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)$›
have $**$: ‹$?U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \varepsilon\ A_0\ \sigma_0 \cup ?U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)$›
by (*auto simp add*: *nth-Cons split*: *nat.split-asm*)
have $***$: ‹$?I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) = \varepsilon\ A_0\ \sigma_0 \cap ?I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)$›
by (*auto simp add*: *nth-Cons split*: *nat.splits*)
have $****$: ‹$?U\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) - E \cup ?I\ (A_0\ \#\ A_1\ \#\ As)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s) =$
combine-sets-Sync $(\varepsilon\ A_0\ \sigma_0)\ E\ (?U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) - E \cup ?I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s))$›
unfolding $**\ ***$ **by** *auto*
have \$: ‹$?U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) - E \cup ?I\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s) = \varepsilon\ \langle\!\langle_d\bigotimes [\![E]\!]\ A_1\ \#\ As\rangle\!\rangle\ (\sigma_1\ \#\ \sigma s)$›
by (*subst* ε-*iterated-combine*$_d$-*Sync-general-form*)
(*simp-all add*: *Cons.hyps*(*1*))

from *Cons.hyps*(*1*) **have** ‹$a \notin ?U\ As\ \sigma s \Longrightarrow$
$map2\ (\lambda\sigma\ A.\ if\ a \in \varepsilon\ A\ \sigma\ then\ \lceil\tau\ A\ \sigma\ a\rceil\ else\ \sigma)\ \sigma s\ As = \sigma s$›
by (*induct* σs *As rule*: *induct-2-lists012*)
(*auto simp add*: *ε-simps lessThan-def*, *fastforce*)
moreover have ‹$?U\ As\ \sigma s \subseteq ?U\ (A_1\ \#\ As)\ (\sigma_1\ \#\ \sigma s)$› **by** *force*
ultimately show ‹$\tau\ \langle\!\langle_d\bigotimes [\![E]\!]\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)\ a =$
$\lfloor map2\ (\lambda x\ y.\ if\ a \in \varepsilon\ y\ x\ then\ \lceil\tau\ y\ x\ a\rceil\ else\ x)\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s)$

$(A_0 \# A_1 \# As)\rfloor$›
using $*$ **unfolding** $****$ \$
by (*simp add*: ε-*combine*$_{Rlist}$-*Sync combine-Sync-*ε-*def*, *safe*,
auto simp add: *combine*$_{Rlist}$-*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'*:
‹$\exists! i.\ i < length\ As \wedge a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
if ‹$length\ \sigma s = length\ As$›
and ‹$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
$\varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i) \cap \varepsilon\ (As\ !\ j)\ (\sigma s\ !\ j) \subseteq E$›
and ‹$a \in (\bigcup i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)) - E$›
proof (*rule ex-ex1I*)
from *that*(*3*) **show** ‹$\exists\, i{<}length\ As.\ a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$› **by** *auto*
next
fix *i j* **assume** ‹$i < length\ As \wedge a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
‹$j < length\ As \wedge a \in \varepsilon\ (As\ !\ j)\ (\sigma s\ !\ j)$›
moreover from *that*(*3*) **have** ‹$a \notin E$› **by** *blast*
ultimately show ‹$i = j$› **using** *that*(*2*)[*of i j*] **by** *auto*
qed

lemma *indep-implies-only-one-enabled*:
‹$[\![length\ \sigma s = length\ As;$
$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
$indep\text{-}enabl\ (As\ !\ i)\ (\sigma s\ !\ i)\ E\ (As\ !\ j)\ (\sigma s\ !\ j);$
$a \in (\bigcup i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)) - E]\!] \Longrightarrow$
$\exists! i.\ i < length\ As \wedge a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
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*:
‹$\tau\ \langle\!\langle_d \bigotimes [\![E]\!]\ As\rangle\!\rangle\ \sigma s\ a =$
$(if\ As = []\ then\ \Diamond$
$else\quad if\ a \in (\bigcap i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i))$
$then\ \lfloor map2\ (\lambda\sigma\ A.\ \lceil\tau\ A\ \sigma\ a\rceil)\ \sigma s\ As\rfloor$
$else\quad if\ a \in (\bigcup i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)) - E$
$then\ let\ i = THE\ i.\ i < length\ As \wedge\ a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$
$in\ \lfloor\sigma s[i := \lceil\tau\ (As\ !\ i)\ (\sigma s\ !\ i)\ a\rceil]\rfloor$
$else\ \Diamond)$›
(**is** ‹- = (*if* $As = []$ *then* $\Diamond$ *else*
if $a \in$ *?I As* σs *then* $\lfloor$*map2* $(\lambda\sigma\ A.\ \lceil\tau\ A\ \sigma\ a\rceil)\ \sigma s\ As\rfloor$ *else*
if $a \in$ *?U As* $\sigma s - E$ *then ?upd As* σs *else* $\Diamond$)›)
if ‹$length\ \sigma s = length\ As$›

‹$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
$indep\text{-}enabl\ (As\ !\ i)\ (\sigma s\ !\ i)\ E\ (As\ !\ j)\ (\sigma s\ !\ j)$›
proof (*subst* τ*-iterated-combine*$_d$*-Sync-general-form*[*OF that*(*1*)], *rule if-cong*)
show ‹(*if* $a \in ?U\ As\ \sigma s - E \cup ?I\ As\ \sigma s$
then $\lfloor map2\ (\lambda x\ y.\ if\ a \in \varepsilon\ y\ x\ then\ \lceil \tau\ y\ x\ a\rceil\ else\ x)\ \sigma s\ As\rfloor$ *else* $\Diamond$) =
(*if* $a \in ?I\ As\ \sigma s$ *then* $\lfloor map2\ (\lambda x\ y.\ \lceil \tau\ y\ x\ a\rceil)\ \sigma s\ As\rfloor$ *else*
if $a \in ?U\ As\ \sigma s - E$ *then* $?upd\ As\ \sigma s$ *else* $\Diamond$)›
proof (*split if-split*, *intro conjI impI*)
assume $*$: ‹$a \in ?I\ As\ \sigma s$›
with *that*(*1*) **have** ‹$(\sigma,\ A) \in set\ (zip\ \sigma s\ As) \Longrightarrow a \in \varepsilon\ A\ \sigma$› **for** $\sigma\ A$
by (*induct* $\sigma s\ As$ *rule*: *list-induct2*) (*use lessThan-Suc-eq-insert-0* **in** *auto*)
with $*$ **show** ‹(*if* $a \in ?U\ As\ \sigma s - E \cup ?I\ As\ \sigma s$
then $\lfloor map2\ (\lambda x\ y.\ if\ a \in \varepsilon\ y\ x\ then\ \lceil \tau\ y\ x\ a\rceil\ else\ x)\ \sigma s\ As\rfloor$ *else* $\Diamond$) =
$\lfloor map2\ (\lambda x\ y.\ \lceil \tau\ y\ x\ a\rceil)\ \sigma s\ As\rfloor$› **by** *auto*
next
assume $*$: ‹$a \notin ?I\ As\ \sigma s$›
show ‹(*if* $a \in ?U\ As\ \sigma s - E \cup ?I\ As\ \sigma s$
then $\lfloor map2\ (\lambda x\ y.\ if\ a \in \varepsilon\ y\ x\ then\ \lceil \tau\ y\ x\ a\rceil\ else\ x)\ \sigma s\ As\rfloor$ *else* $\Diamond$) =
(*if* $a \in ?U\ As\ \sigma s - E$ *then* $?upd\ As\ \sigma s$ *else* $\Diamond$)›
proof (*rule if-cong*)
assume ‹$a \in ?U\ As\ \sigma s - E$›
from *indep-implies-only-one-enabled*[*OF that this*]
obtain i **where** \$: ‹$i < length\ As$› ‹$a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)$›
‹$j < length\ As \Longrightarrow j \neq i \Longrightarrow a \notin \varepsilon\ (As\ !\ j)\ (\sigma s\ !\ j)$› **for** j **by** *blast*
have \$\$: ‹$(THE\ i.\ i < length\ As \wedge a \in \varepsilon\ (As\ !\ i)\ (\sigma s\ !\ i)) = i$›
using \$ **by** *blast*
have ‹$\lfloor map2\ (\lambda x\ y.\ if\ a \in \varepsilon\ y\ x\ then\ \lceil \tau\ y\ x\ a\rceil\ else\ x)\ \sigma s\ As\rfloor =$
$\lfloor \sigma s[i := \lceil \tau\ (As\ !\ i)\ (\sigma s\ !\ i)\ a\rceil]\rfloor$›
by (*auto intro*!: *nth-equalityI simp add*: *that*(*1*))
(*use that*(*1*) \$(*3*) **in** *fastforce*, *metis* \$(*2*) *nth-list-update-neq*)
also have ‹$\ldots = ?upd\ As\ \sigma s$›
using \$\$ **by** *auto*
finally show ‹$\lfloor map2\ (\lambda x\ y.\ if\ a \in \varepsilon\ y\ x\ then\ \lceil \tau\ y\ x\ a\rceil\ else\ x)\ \sigma s\ As\rfloor = ?upd\ As\ \sigma s$› **.**
qed (*use* $*$ **in** *auto*)
qed
qed *simp-all*

6.5 More on Events

lemma *events-of-MultiSync-*P_{SKIPS}*-nd* :
‹$\alpha([\![E]\!]\ (\sigma,\ A) \in\#\ mset\ (zip\ \sigma s\ As).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma) =$
(*if* $As = []$ *then* $\{\}$ *else*
$\bigcup\ \sigma s' \in \mathcal{R}_{nd}\ \langle\!\langle_{nd}\bigotimes [\![E]\!]\ As\rangle\!\rangle\ \sigma s.\ (\bigcup i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s'\ !\ i)) - E \cup$
$(\bigcap i{<}length\ As.\ \varepsilon\ (As\ !\ i)\ (\sigma s'\ !\ i)))$›
(**is** ‹- = *?rhs*›) **if** ‹$length\ \sigma s = length\ As$›
‹$\bigwedge A.\ A \in set\ As \Longrightarrow \varrho\text{-}disjoint\text{-}\varepsilon\ A$› ‹$\bigwedge A.\ A \in set\ As \Longrightarrow at\text{-}most\text{-}1\text{-}elem\text{-}term\ A$›

proof −
have ‹$[\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ $\sigma = P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{nd}$ σs›
by (*simp only*: P_{SKIPS}-*nd-compactification-Sync*[*OF that*])
also have ‹$\alpha(\ldots) = \bigcup$ (ε $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ ‘ $\mathcal{R}_{nd}$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs)›
proof (*rule events-of-*P_{SKIPS}*-nd*)
show ‹ϱ-*disjoint-*ε $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$›
by (*simp only*: ϱ-*disjoint-*ε-*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 ‹$\ldots$ = *?rhs*› **by** (*auto simp add*: ε-*iterated-combine*$_{nd}$-*Sync-general-form*)
finally show *?thesis* **.**
qed

lemma *events-of-MultiSync-P-nd* :
‹$\alpha([\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P\langle\!\langle A\rangle\!\rangle_{nd}$ σ) =
(*if As* = [] *then* {} *else*
$\bigcup$ $\sigma s' \in \mathcal{R}_{nd}$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs. ($\bigcup$*i*<*length As*. ε (*As* ! *i*) ($\sigma s'$! *i*)) − *E* ∪
($\bigcap$*i*<*length As*. ε (*As* ! *i*) ($\sigma s'$! *i*)))›
(**is** ‹- = *?rhs*›) **if** ‹*length* σs = *length As*›
proof −
have ‹$[\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P\langle\!\langle A\rangle\!\rangle_{nd}$ $\sigma = P\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{nd}$ σs›
by (*fact P-nd-compactification-Sync*[*OF that*])
also have ‹$\alpha(\ldots) = \bigcup$ (ε $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ ‘ $\mathcal{R}_{nd}$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs)› **by** (*fact events-of-P-nd*)
also from *same-length-*$\mathcal{R}_{nd}$-*iterated-combine*$_{nd}$-*Sync-description*[*OF that*(*1*)]
have ‹$\ldots$ = *?rhs*› **by** (*auto simp add*: ε-*iterated-combine*$_{nd}$-*Sync-general-form*)
finally show *?thesis* **.**
qed

lemma *events-of-MultiSync-*P_{SKIPS}*-d* :
‹$\alpha([\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{d}$ σ) =
(*if As* = [] *then* {} *else*
$\bigcup$ $\sigma s' \in \mathcal{R}_{d}$ $\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs. ($\bigcup$*i*<*length As*. ε (*As* ! *i*) ($\sigma s'$! *i*)) − *E* ∪
($\bigcap$*i*<*length As*. ε (*As* ! *i*) ($\sigma s'$! *i*))))›
(**is** ‹- = *?rhs*›) **if** ‹*length* σs = *length As*› ‹$\bigwedge$*A*. *A* ∈ *set As* ⟹ ϱ-*disjoint-*ε *A*›
‹$\bigwedge$*i j*. $[\![$*i* < *length As*; *j* < *length As*; *i* ≠ *j*$]\!]$ ⟹
indep-enabl (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)›
proof −
have ‹$[\![E]\!]$ $(\sigma,\ A) \in\#$ *mset* (*zip* σs *As*). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{d}$ $\sigma = P_{SKIPS}\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_{d}$ σs›
by (*simp add*: P_{SKIPS}-*d-compactification-Sync*[*OF that*])
also have ‹$\alpha(\ldots) = \bigcup$ (ε $\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ ‘ $\mathcal{R}_{d}$ $\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs)›
proof (*rule events-of-*P_{SKIPS}*-d*)
show ‹ϱ-*disjoint-*ε $\langle\!\langle_{d}\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$›
by (*simp only*: ϱ-*disjoint-*ε-*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 ‹... = *?rhs*› **by** (*auto simp add*: ε*-iterated-combine*$_d$*-Sync-general-form*)
finally show *?thesis* **.**
qed

lemma *events-of-MultiSync-P-d* :
‹α($[\![E]\!]$ (σ, A) $\in\#$ *mset* (*zip* σs *As*). $P\langle\!\langle A\rangle\!\rangle_d$ σ) =
(*if As* = [] *then* {} *else*
$\bigcup$ $\sigma s' \in \mathcal{R}_d$ $\langle\!\langle{}_d\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs. ($\bigcup$ *i*<*length As*. ε (*As* ! *i*) ($\sigma s'$! *i*) − *E* ∪
($\bigcap$ *i*<*length As*. ε (*As* ! *i*) ($\sigma s'$! *i*))))›
(**is** ‹- = *?rhs*›) **if** ‹*length* σs = *length As*›
and ‹$\bigwedge$*i j*. $[\![i$ < *length As*; *j* < *length As*; *i* ≠ *j*$]\!]$ ⟹
indep-enabl (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)›
proof −
have ‹$[\![E]\!]$ (σ, A) $\in\#$ *mset* (*zip* σs *As*). $P\langle\!\langle A\rangle\!\rangle_d$ σ = $P\langle\!\langle\langle\!\langle{}_d\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle\rangle\!\rangle_d$ σs›
by (*simp add*: *P-d-compactification-Sync*[*OF that*])
also have ‹α(...) = $\bigcup$ (ε $\langle\!\langle{}_d\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ ‘ $\mathcal{R}_d$ $\langle\!\langle{}_d\bigotimes [\![E]\!]$ *As*$\rangle\!\rangle$ σs)› **by** (*fact events-of-P-d*)
also from *same-length-*$\mathcal{R}_d$*-iterated-combine*$_d$*-Sync-description*[*OF that*(*1*)]
have ‹... = *?rhs*› **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 $combine_{dPairlist}$-$Sync_{ptick}$ ::
‹[('σ, 'e, 'r, 'α) A_d-scheme, 'e set, ('σ, 'e, 'r, 'β) A_d-scheme] ⇒ ('σ list, 'e, 'r list) A_d›
where ‹$combine_{dPairlist}$-$Sync_{ptick}$ A_0 E A_1 ≡
$combine_d$-Sync A_0 E A_1 hd (λσs. hd (tl σs)) (λs t. [s, t]) (λs t. ⌊[s, t]⌋)›
definition $combine_{ndPairlist}$-$Sync_{ptick}$ ::
‹[('σ, 'e, 'r, 'α) A_{nd}-scheme, 'e set, ('σ, 'e, 'r, 'β) A_{nd}-scheme] ⇒ ('σ list, 'e, 'r list) A_{nd}›
where ‹$combine_{ndPairlist}$-$Sync_{ptick}$ A_0 E A_1 ≡ $combine_{nd}$-Sync A_0 E A_1 hd (λσs. hd (tl σs)) (λs t. [s, t]) (λs t. ⌊[s, t]⌋)›

definition $combine_{dPair}$-$Sync_{ptick}$ ::
‹[('σ_0, 'e, 'r_0, 'α) A_d-scheme, 'e set, ('σ_1, 'e, 'r_1, 'β) A_d-scheme] ⇒ ('σ_0 × 'σ_1, 'e, 'r_0 × 'r_1) A_d›
where ‹$combine_{dPair}$-$Sync_{ptick}$ A_0 E A_1 ≡ $combine_d$-Sync A_0 E A_1 fst snd Pair (λs r. ⌊(s, r)⌋)›
definition $combine_{ndPair}$-$Sync_{ptick}$ ::
‹[('σ_0, 'e, 'r_0, 'α) A_{nd}-scheme, 'e set, ('σ_1, 'e, 'r_1, 'β) A_{nd}-scheme] ⇒ ('σ_0 × 'σ_1, 'e, 'r_0 × 'r_1) A_{nd}›
where ‹$combine_{ndPair}$-$Sync_{ptick}$ A_0 E A_1 ≡ $combine_{nd}$-Sync A_0 E A_1 fst snd Pair (λs r. ⌊(s, r)⌋)›

definition $combine_{dListslenL}$-$Sync_{ptick}$ ::
‹[('σ list, 'e, 'r list, 'α) A_d-scheme, nat, 'e set, ('σ list, 'e, 'r list, 'β) A_d-scheme] ⇒ ('σ list, 'e, 'r list) A_d›

where ‹$combine_{dListslenL}$-$Sync_{ptick}$ A_0 len_0 E A_1 ≡ $combine_d$-$Sync$ A_0 E A_1 ($take$ len_0) ($drop$ len_0) (@) (λs r. $\lfloor s$ @ $r\rfloor$)›
definition $combine_{ndListslenL}$-$Sync_{ptick}$ ::
‹[('σ $list$, 'e, 'r $list$, 'α) A_{nd}-$scheme$, nat, 'e set, ('σ $list$, 'e, 'r $list$, 'β) A_{nd}-$scheme$] ⇒ ('σ $list$, 'e, 'r $list$) A_{nd}›
where ‹$combine_{ndListslenL}$-$Sync_{ptick}$ A_0 len_0 E A_1 ≡ $combine_{nd}$-$Sync$ A_0 E A_1 ($take$ len_0) ($drop$ len_0) (@) (λs r. $\lfloor s$ @ $r\rfloor$)›

definition $combine_{dRlist}$-$Sync_{ptick}$ ::
‹[('σ, 'e, 'r, 'α) A_d-$scheme$, 'e set, ('σ $list$, 'e, 'r $list$, 'β) A_d-$scheme$] ⇒ ('σ $list$, 'e, 'r $list$) A_d›
where ‹$combine_{dRlist}$-$Sync_{ptick}$ A_0 E A_1 ≡ $combine_d$-$Sync$ A_0 E A_1 hd tl (#) (λs r. $\lfloor s$ # $r\rfloor$)›
definition $combine_{ndRlist}$-$Sync_{ptick}$ ::
‹[('σ, 'e, 'r, 'α) A_{nd}-$scheme$, 'e set, ('σ $list$, 'e, 'r $list$, 'β) A_{nd}-$scheme$] ⇒ ('σ $list$, 'e, 'r $list$) A_{nd}›
where ‹$combine_{ndRlist}$-$Sync_{ptick}$ A_0 E A_1 ≡ $combine_{nd}$-$Sync$ A_0 E A_1 hd tl (#) (λs r. $\lfloor s$ # $r\rfloor$)›

lemmas $combine_{Pairlist}$-$Sync_{ptick}$-$defs$ = $combine_{dPairlist}$-$Sync_{ptick}$-def $combine_{ndPairlist}$-$Sync_{ptick}$-
and $combine_{Pair}$-$Sync_{ptick}$-$defs$ = $combine_{dPair}$-$Sync_{ptick}$-def $combine_{ndPair}$-$Sync_{ptick}$-def
and $combine_{ListslenL}$-$Sync_{ptick}$-$defs$ = $combine_{dListslenL}$-$Sync_{ptick}$-def $combine_{ndListslenL}$-$Sync_{ptick}$-def
and $combine_{Rlist}$-$Sync_{ptick}$-$defs$ = $combine_{dRlist}$-$Sync_{ptick}$-def $combine_{ndRlist}$-$Sync_{ptick}$-def

lemmas $combine$-$Sync_{ptick}$-$defs$ =
$combine_{Pairlist}$-$Sync_{ptick}$-$defs$ $combine_{Pair}$-$Sync_{ptick}$-$defs$ $combine_{ListslenL}$-$Sync_{ptick}$-$defs$ $combine_{Rlist}$-$Sync_{ptick}$-$defs$

bundle $combine_{nd}$-$Sync_{ptick}$-$syntax$ **begin**

notation $combine_{dPairlist}$-$Sync_{ptick}$ (‹⟪- $_d$⊗⟦-⟧$_{\checkmark Pairlist}$ -⟫› [0, 0, 0])
notation $combine_{ndPairlist}$-$Sync_{ptick}$ (‹⟪- $_{nd}$⊗⟦-⟧$_{\checkmark Pairlist}$ -⟫› [0, 0, 0])
notation $combine_{dPair}$-$Sync_{ptick}$ (‹⟪- $_d$⊗⟦-⟧$_{\checkmark Pair}$ -⟫› [0, 0, 0])
notation $combine_{ndPair}$-$Sync_{ptick}$ (‹⟪- $_{nd}$⊗⟦-⟧$_{\checkmark Pair}$ -⟫› [0, 0, 0])
notation $combine_{dListslenL}$-$Sync_{ptick}$ (‹⟪- $_d$⊗⟦-, -⟧$_{\checkmark ListslenL}$ -⟫› [0, 0, 0, 0])
notation $combine_{ndListslenL}$-$Sync_{ptick}$ (‹⟪- $_{nd}$⊗⟦-, -⟧$_{\checkmark ListslenL}$ -⟫› [0, 0, 0, 0])
notation $combine_{dRlist}$-$Sync_{ptick}$ (‹⟪- $_d$⊗⟦-⟧$_{\checkmark Rlist}$ -⟫› [0, 0, 0])
notation $combine_{ndRlist}$-$Sync_{ptick}$ (‹⟪- $_{nd}$⊗⟦-⟧$_{\checkmark Rlist}$ -⟫› [0, 0, 0])

end

unbundle $combine_{nd}$-$Sync_{ptick}$-$syntax$

7.2 First Properties

lemma $finite$-$trans$-$combine_{nd}$-$Sync_{ptick}$-$simps$ [$simp$] :
‹$finite$-$trans$ A_0 ⟹ $finite$-$trans$ A_1 ⟹ $finite$-$trans$ ⟪A_0 $_{nd}$⊗⟦E⟧$_{\checkmark Pairlist}$ A_1⟫›

‹*finite-trans* $B_0 \Longrightarrow$ *finite-trans* $B_1 \Longrightarrow$ *finite-trans* $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ B_1\rangle\!\rangle$›
‹*finite-trans* $C_0 \Longrightarrow$ *finite-trans* $C_1 \Longrightarrow$ *finite-trans* $\langle\!\langle C_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ C_1\rangle\!\rangle$›
‹*finite-trans* $D_0 \Longrightarrow$ *finite-trans* $D_1 \Longrightarrow$ *finite-trans* $\langle\!\langle D_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ D_1\rangle\!\rangle$›
unfolding *combine*$_{ndPairlist}$-*Sync*$_{ptick}$-*def combine*$_{ndPair}$-*Sync*$_{ptick}$-*def combine*$_{ndListslenL}$-*Sync*$_{ptick}$-*def combine*$_{ndRlist}$-*Sync*$_{ptick}$-*def*
by (*simp-all add*: *finite-trans-def finite-image-set2*)

lemma *ε-combine*$_{Pairlist}$-*Sync*$_{ptick}$:
‹$\varepsilon\ \langle\!\langle A_0\ {}_{d}\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\ \sigma s =$ *combine-Sync-ε* $A_0\ E\ A_1$ *hd* (*hd* ∘ *tl*) σs›
‹$\varepsilon\ \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ B_1\rangle\!\rangle\ \sigma s =$ *combine-Sync-ε* $B_0\ E\ B_1$ *hd* (*hd* ∘ *tl*) σs›
by (*auto simp add*: *combine-Sync-ε-def-bis combine*$_{Pairlist}$-*Sync*$_{ptick}$-*defs ε-simps*)

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

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

lemma *ε-combine*$_{Rlist}$-*Sync*$_{ptick}$:
‹$\varepsilon\ \langle\!\langle A_0\ {}_{d}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\ \sigma s =$ *combine-Sync-ε* $A_0\ E\ A_1$ *hd tl* σs›
‹$\varepsilon\ \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ B_1\rangle\!\rangle\ \sigma s =$ *combine-Sync-ε* $B_0\ E\ B_1$ *hd tl* σs›
by (*auto simp add*: *combine-Sync-ε-def-bis combine*$_{Rlist}$-*Sync*$_{ptick}$-*defs ε-simps*)

lemma *ϱ-combine*$_{Pairlist}$-*Sync*$_{ptick}$:
‹$\varrho\ \langle\!\langle A_0\ {}_{d}\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle = \{\sigma s.\ hd\ \sigma s \in \varrho\ A_0 \wedge hd\ (tl\ \sigma s) \in \varrho\ A_1\}$›
‹$\varrho\ \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ B_1\rangle\!\rangle = \{\sigma s.\ hd\ \sigma s \in \varrho\ B_0 \wedge hd\ (tl\ \sigma s) \in \varrho\ B_1\}$›
by (*auto simp add*: *combine*$_{Pairlist}$-*Sync*$_{ptick}$-*defs ϱ-simps split*: *option.split*)

lemma *ϱ-combine*$_{Pair}$-*Sync*$_{ptick}$:
‹$\varrho\ \langle\!\langle A_0\ {}_{d}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle = \{(\sigma_0,\ \sigma_1).\ \sigma_0 \in \varrho\ A_0 \wedge \sigma_1 \in \varrho\ A_1\}$›
‹$\varrho\ \langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ B_1\rangle\!\rangle = \{(\sigma_0,\ \sigma_1).\ \sigma_0 \in \varrho\ B_0 \wedge \sigma_1 \in \varrho\ B_1\}$›
by (*auto simp add*: *combine*$_{Pair}$-*Sync*$_{ptick}$-*defs ϱ-simps split*: *option.split*)

lemma *ϱ-combine*$_{ListslenL}$-*Sync*$_{ptick}$:
‹$\varrho\ \langle\!\langle A_0\ {}_{d}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle = \{\sigma s.\ take\ len_0\ \sigma s \in \varrho\ A_0 \wedge drop\ len_0\ \sigma s \in \varrho\ A_1\}$›
‹$\varrho\ \langle\!\langle B_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ B_1\rangle\!\rangle = \{\sigma s.\ take\ len_0\ \sigma s \in \varrho\ B_0 \wedge drop\ len_0\ \sigma s \in \varrho\ B_1\}$›
by (*auto simp add*: *combine*$_{ListslenL}$-*Sync*$_{ptick}$-*defs ϱ-simps split*: *option.split*)

lemma *ϱ-combine*$_{Rlist}$-*Sync*$_{ptick}$:

‹ϱ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle = \{\sigma s.\ hd\ \sigma s \in \varrho\ A_0 \wedge tl\ \sigma s \in \varrho\ A_1\}$›
‹ϱ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ B_1\rangle\!\rangle = \{\sigma s.\ hd\ \sigma s \in \varrho\ B_0 \wedge tl\ \sigma s \in \varrho\ B_1\}$›
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*$_{ptick}$:
‹τ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle = \tau$ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pairlist}\ A_1\rangle\!\rangle$›
‹τ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ B_1\rangle\!\rangle = \tau$ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Pairlist}\ B_1\rangle\!\rangle$›
by (*simp-all add*: *combine-Sync-defs combine-Sync*$_{ptick}$-*defs*)

lemma τ-*combine*$_{Pair}$-*Sync*$_{ptick}$:
‹τ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle = \tau$ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{Pair}\ A_1\rangle\!\rangle$›
‹τ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ B_1\rangle\!\rangle = \tau$ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{Pair}\ B_1\rangle\!\rangle$›
by (*simp-all add*: *combine-Sync-defs combine-Sync*$_{ptick}$-*defs*)

lemma τ-*combine*$_{ListslenL}$-*Sync*$_{ptick}$:
‹τ $\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle = \tau$ $\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle$›
‹τ $\langle\!\langle B_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ B_1\rangle\!\rangle = \tau$ $\langle\!\langle B_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ B_1\rangle\!\rangle$›
by (*simp-all add*: *combine-Sync-defs combine-Sync*$_{ptick}$-*defs*)

τ $\langle\!\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle$ and τ $\langle\!\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ B_1\rangle\!\rangle$ cannot be obtained that easily because of the types of terminations.

7.4 Reachability

lemma $\mathcal{R}_d$-*combine*$_{dListslenL}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_d$ $\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle$ $(s_0\ @\ s_1) \subseteq \{t_0\ @\ t_1 |\ t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1\}$› (**is** ‹$?S_A \subseteq$ -›)
if ‹$\bigwedge t_0.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \Longrightarrow length\ t_0 = len_0$›
by (*subst same-*τ-*implies-same-*$\mathcal{R}_d$[*of* - - ‹$\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{ListslenL}\ A_1\rangle\!\rangle$›])
(*simp-all add*: τ-*combine*$_{ListslenL}$-*Sync*$_{ptick}$ $\mathcal{R}_d$-*combine*$_{dListslenL}$-*Sync-subset that*)

lemma $\mathcal{R}_{nd}$-*combine*$_{ndListslenL}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_{nd}$ $\langle\!\langle B_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ B_1\rangle\!\rangle$ $(s_0\ @\ s_1) \subseteq \{t_0\ @\ t_1 |\ t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1\}$› (**is** ‹$?S_B \subseteq$ -›)
if ‹$\bigwedge t_0.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \Longrightarrow length\ t_0 = len_0$›
by (*subst same-*τ-*implies-same-*$\mathcal{R}_{nd}$[*of* - - ‹$\langle\!\langle B_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{ListslenL}\ B_1\rangle\!\rangle$›])
(*simp-all add*: τ-*combine*$_{ListslenL}$-*Sync*$_{ptick}$ $\mathcal{R}_{nd}$-*combine*$_{ndListslenL}$-*Sync-subset that*)

lemma $\mathcal{R}_d$-*combine*$_{dPairlist}$-*Sync*$_{ptick}$-*subset*:

‹$\mathcal{R}_d$ $\langle\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Pairlist}$ $A_1\rangle\rangle$ $[s_0,\ s_1] \subseteq \{[t_0,\ t_1] |\ t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1\}$› (**is** ‹$?S_A \subseteq$ -›)
and $\mathcal{R}_{nd}$-*combine*$_{ndPairlist}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_{nd}$ $\langle\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}$ $B_1\rangle\rangle$ $[s_0,\ s_1] \subseteq \{[t_0,\ t_1] |\ t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1\}$› (**is** ‹$?S_B \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = [t_0,\ t_1] \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge t_1 \in \mathcal{R}_d\ A_1\ s_1$› **for** t
by ($\mathcal{R}_d$-*subset-method defs*: *combine*$_{Pairlist}$-*Sync*$_{ptick}$-*defs*)
show ‹$t \in ?S_B \Longrightarrow \exists\, t_0\ t_1.\ t = [t_0,\ t_1] \wedge t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge t_1 \in \mathcal{R}_{nd}\ B_1\ s_1$› **for** t
by ($\mathcal{R}_{nd}$-*subset-method defs*: *combine*$_{Pairlist}$-*Sync*$_{ptick}$-*defs*)
qed

lemma $\mathcal{R}_d$-*combine*$_{dPair}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_d$ $\langle\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Pair}$ $A_1\rangle\rangle$ $(s_0,\ s_1) \subseteq \mathcal{R}_d\ A_0\ s_0 \times \mathcal{R}_d\ A_1\ s_1$› (**is** ‹$?S_A \subseteq$ -›)
and $\mathcal{R}_{nd}$-*combine*$_{ndPair}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_{nd}$ $\langle\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Pair}$ $B_1\rangle\rangle$ $(s_0,\ s_1) \subseteq \mathcal{R}_{nd}\ B_0\ s_0 \times \mathcal{R}_{nd}\ B_1\ s_1$› (**is** ‹$?S_B \subseteq$ -›)
proof −
have ‹$t \in ?S_A \Longrightarrow$ *fst* $t \in \mathcal{R}_d\ A_0\ s_0 \wedge$ *snd* $t \in \mathcal{R}_d\ A_1\ s_1$› **for** t
by ($\mathcal{R}_d$-*subset-method defs*: *combine*$_{Pair}$-*Sync*$_{ptick}$-*defs*)
thus ‹$?S_A \subseteq \mathcal{R}_d\ A_0\ s_0 \times \mathcal{R}_d\ A_1\ s_1$› **by** *force*
next
have ‹$t \in ?S_B \Longrightarrow$ *fst* $t \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge$ *snd* $t \in \mathcal{R}_{nd}\ B_1\ s_1$› **for** t
by ($\mathcal{R}_{nd}$-*subset-method defs*: *combine*$_{Pair}$-*Sync*$_{ptick}$-*defs*)
thus ‹$?S_B \subseteq \mathcal{R}_{nd}\ B_0\ s_0 \times \mathcal{R}_{nd}\ B_1\ s_1$› **by** *force*
qed

lemma $\mathcal{R}_d$-*combine*$_{dRlist}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_d$ $\langle\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Rlist}$ $A_1\rangle\rangle$ $(s_0\ \#\ \sigma s) \subseteq \{t_0\ \#\ \sigma t |\ t_0\ \sigma t.\ t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge \sigma t \in \mathcal{R}_d\ A_1\ \sigma s\}$› (**is** ‹$?S_A \subseteq$ -›)
and $\mathcal{R}_{nd}$-*combine*$_{ndRlist}$-*Sync*$_{ptick}$-*subset*:
‹$\mathcal{R}_{nd}$ $\langle\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Rlist}$ $B_1\rangle\rangle$ $(s_0\ \#\ \sigma s) \subseteq \{t_0\ \#\ \sigma t |\ t_0\ \sigma t.\ t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge \sigma t \in \mathcal{R}_{nd}\ B_1\ \sigma s\}$› (**is** ‹$?S_B \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ \sigma t.\ t = t_0\ \#\ \sigma t \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0 \wedge \sigma t \in \mathcal{R}_d\ A_1\ \sigma s$› **for** t
by ($\mathcal{R}_d$-*subset-method defs*: *combine*$_{Rlist}$-*Sync*$_{ptick}$-*defs*)
next
show ‹$t \in ?S_B \Longrightarrow \exists\, t_0\ \sigma t.\ t = t_0\ \#\ \sigma t \wedge t_0 \in \mathcal{R}_{nd}\ B_0\ s_0 \wedge \sigma t \in \mathcal{R}_{nd}\ B_1\ \sigma s$› **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*:
‹ω $\langle\langle\langle\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Pairlist}$ $A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $[s_0,\ s_1] = \omega$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}$ $_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $[s_0,\ s_1]$›

by (*simp add*: *combine*$_{Pairlist}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if*)

lemma ω-*combine*$_{Pair}$-*Sync*$_{ptick}$-*behaviour*:
‹ω $\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $(s_0,\ s_1) = \omega$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $(s_0,\ s_1)$›
by (*simp add*: *combine*$_{Pair}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if*)

lemma ω-*combine*$_{ListslenL}$-*Sync*$_{ptick}$-*behaviour*:
‹ω $\langle\langle\langle\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $(\sigma s_0\ @\ \sigma s_1) = \omega$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $(\sigma s_0\ @\ \sigma s_1)$›
by (*simp add*: *combine*$_{ListslenL}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if*)

lemma ω-*combine*$_{Rlist}$-*Sync*$_{ptick}$-*behaviour*:
‹ω $\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $(s_0\ \#\ \sigma s_1) = \omega$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $(s_0\ \#\ \sigma s_1)$›
by (*simp add*: *combine*$_{Rlist}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if*)

lemma τ-*combine*$_{Pairlist}$-*Sync*$_{ptick}$-*behaviour-when-indep*:
‹$\varepsilon\ A_0\ s_0 \cap \varepsilon\ A_1\ s_1 \subseteq E \Longrightarrow$
τ $\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $[s_0,\ s_1]\ e = \tau$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $[s_0,\ s_1]\ e$›
by (*auto simp add*: *combine*$_{Pairlist}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps*)

lemma τ-*combine*$_{Pair}$-*Sync*$_{ptick}$-*behaviour-when-indep*:
‹$\varepsilon\ A_0\ s_0 \cap \varepsilon\ A_1\ s_1 \subseteq E \Longrightarrow$
τ $\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $(s_0,\ s_1)\ e = \tau$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $(s_0,\ s_1)\ e$›
by (*auto simp add*: *combine*$_{Pair}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps*)

lemma τ-*combine*$_{ListslenL}$-*Sync*$_{ptick}$-*behaviour-when-indep*:
‹$\varepsilon\ A_0\ \sigma s_0 \cap \varepsilon\ A_1\ \sigma s_1 \subseteq E \Longrightarrow$ *length* $\sigma s_0 = len_0 \Longrightarrow$
τ $\langle\langle\langle\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $(\sigma s_0\ @\ \sigma s_1)\ e = \tau$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $(\sigma s_0\ @\ \sigma s_1)\ e$›
by (*auto simp add*: *combine*$_{ListslenL}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps*)

lemma τ-*combine*$_{Rlist}$-*Sync*$_{ptick}$-*behaviour-when-indep*:
‹$\varepsilon\ A_0\ s_0 \cap \varepsilon\ A_1\ \sigma s_1 \subseteq E \Longrightarrow$
τ $\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\rangle\rangle\rangle_{d\hookrightarrow nd}$ $(s_0\ \#\ \sigma s_1)\ e = \tau$ $\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle$ $(s_0\ \#\ \sigma s_1)\ e$›
by (*auto simp add*: *combine*$_{Rlist}$-*Sync*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps*)

lemma P_{SKIPS}-*combine*$_{Pairlist}$-*Sync*$_{ptick}$-*behaviour-when-indep*:

‹$P_{SKIPS}\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\rangle_d\ [s_0,\ s_1] = P_{SKIPS}\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}\ [s_0,\ s_1]$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ s_1$›
by (P_{SKIPS}*-when-indep-method R-d-subset*: $\mathcal{R}_d$*-combine*$_{dPairlist}$*-Sync*$_{ptick}$*-subset*, *simp-all*)
(*metis* τ*-combine*$_{Pairlist}$*-Sync*$_{ptick}$*-behaviour-when-indep indep-enablD that*,
metis ω*-combine*$_{Pairlist}$*-Sync*$_{ptick}$*-behaviour*)

lemma *P-combine*$_{Pairlist}$*-Sync*$_{ptick}$*-behaviour-when-indep*:
‹$P\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\rangle_d\ [s_0,\ s_1] = P\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}$ $[s_0,\ s_1]$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ s_1$›
by (*P-when-indep-method R-d-subset*: $\mathcal{R}_d$*-combine*$_{dPairlist}$*-Sync*$_{ptick}$*-subset*, *simp-all*)
(*metis* τ*-combine*$_{Pairlist}$*-Sync*$_{ptick}$*-behaviour-when-indep indep-enablD that*)

lemma P_{SKIPS}*-combine*$_{Pair}$*-Sync*$_{ptick}$*-behaviour-when-indep*:
‹$P_{SKIPS}\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle_d\ (s_0,\ s_1) = P_{SKIPS}\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}$ $\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}\ (s_0,\ s_1)$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ s_1$›
by (P_{SKIPS}*-when-indep-method R-d-subset*: $\mathcal{R}_d$*-combine*$_{dPair}$*-Sync*$_{ptick}$*-subset*, *all* ‹*elim SigmaE*›)
(*metis* τ*-combine*$_{Pair}$*-Sync*$_{ptick}$*-behaviour-when-indep indep-enablD that*,
auto simp add: ω*-combine*$_{Pair}$*-Sync*$_{ptick}$*-behaviour option.case-eq-if*)

lemma *P-combine*$_{Pair}$*-Sync*$_{ptick}$*-behaviour-when-indep*:
‹$P\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle_d\ (s_0,\ s_1) = P\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}$ $(s_0,\ s_1)$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ s_1$›
by (*P-when-indep-method R-d-subset*: $\mathcal{R}_d$*-combine*$_{dPair}$*-Sync*$_{ptick}$*-subset*, *elim SigmaE*)
(*metis* τ*-combine*$_{Pair}$*-Sync*$_{ptick}$*-behaviour-when-indep indep-enablD that*)

lemma P_{SKIPS}*-combine*$_{ListslenL}$*-Sync*$_{ptick}$*-behaviour-when-indep*:
‹$P_{SKIPS}\langle\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\rangle_d\ (\sigma s_0\ @\ \sigma s_1) = P_{SKIPS}\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}$ ${}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}\ (\sigma s_0\ @\ \sigma s_1)$›
if ‹*indep-enabl* $A_0\ \sigma s_0\ E\ A_1\ \sigma s_1$› **and** ‹$\bigwedge\sigma t_0.\ \sigma t_0 \in \mathcal{R}_d\ A_0\ \sigma s_0 \Longrightarrow$ *length* $\sigma t_0 = len_0$›
by (P_{SKIPS}*-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*,
metis ω*-combine*$_{ListslenL}$*-Sync*$_{ptick}$*-behaviour*)

lemma *P-combine*$_{ListslenL}$*-Sync*$_{ptick}$*-behaviour-when-indep*:
‹$P\langle\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\rangle_d\ (\sigma s_0\ @\ \sigma s_1) = P\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![len_0,$ $E]\!]_{\checkmark ListslenL}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}\ (\sigma s_0\ @\ \sigma s_1)$›
if ‹*indep-enabl* $A_0\ \sigma s_0\ E\ A_1\ \sigma s_1$› **and** ‹$\bigwedge\sigma t_0.\ \sigma t_0 \in \mathcal{R}_d\ A_0\ \sigma s_0 \Longrightarrow$ *length* $\sigma t_0 = len_0$›

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_{SKIPS}-*combine*$_{Rlist}$-*Sync*$_{ptick}$-*behaviour-when-indep*:
‹$P_{SKIPS}\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\rangle\rangle_d\ (s_0\ \#\ \sigma s_1) = P_{SKIPS}\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (s_0\ \#\ \sigma s_1)$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ \sigma s_1$›
by (P_{SKIPS}-*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*:
‹$P\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\rangle\rangle_d\ (s_0\ \#\ \sigma s_1) = P\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle_{nd}\ (s_0\ \#\ \sigma s_1)$›
if ‹*indep-enabl* $A_0\ s_0\ E\ A_1\ \sigma s_1$›
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* $\Rightarrow$ ($'\sigma$, $'e$, $'r$) A_d *list* $\Rightarrow$ ($'\sigma$ *list*, $'e$, $'r$ *list*) A_d› (‹$\langle\langle{}_d\bigotimes [\![-]\!]_{\checkmark}$ -$\rangle\rangle$› [*0*, *0*])
  where ‹$\langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}$ []$\rangle\rangle$ = $(\!|\tau = \lambda\sigma s\ a.\ \Diamond,\ \omega = \lambda\sigma s.\ \Diamond|\!)$›
  | ‹$\langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}$ [A_0]$\rangle\rangle$ = ${}_d\langle\langle A_0\rangle\rangle_{singl\hookrightarrow list}$›
  | ‹$\langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}$ A_0 # A_1 # *As*$\rangle\rangle$ = $\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}\ A_1$ # *As*$\rangle\rangle\rangle\rangle$›

fun *iterated-combine*$_{nd}$*-Sync*$_{ptick}$ :: ‹$'e$ *set* $\Rightarrow$ ($'\sigma$, $'e$, $'r$) A_{nd} *list* $\Rightarrow$ ($'\sigma$ *list*, $'e$, $'r$ *list*) A_{nd}› (‹$\langle\langle{}_{nd}\bigotimes [\![-]\!]_{\checkmark}$ -$\rangle\rangle$› [*0*, *0*])
  where ‹$\langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}$ []$\rangle\rangle$ = $(\!|\tau = \lambda\sigma s\ a.\ \{\},\ \omega = \lambda\sigma s.\ \{\}|\!)$›
  | ‹$\langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}$ [A_0]$\rangle\rangle$ = ${}_{nd}\langle\langle A_0\rangle\rangle_{singl\hookrightarrow list}$›
  | ‹$\langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}$ A_0 # A_1 # *As*$\rangle\rangle$ = $\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}\ A_1$ # *As*$\rangle\rangle\rangle\rangle$›

lemma *iterated-combine*$_d$*-Sync*$_{ptick}$*-simps-bis*: ‹*As* $\neq$ [] $\Longrightarrow$ $\langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}$ A_0 # *As*$\rangle\rangle$ = $\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}$ *As*$\rangle\rangle\rangle\rangle$›
  and *iterated-combine*$_{nd}$*-Sync*$_{ptick}$*-simps-bis*: ‹*Bs* $\neq$ [] $\Longrightarrow$ $\langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}$ B_0 # *Bs*$\rangle\rangle$ = $\langle\langle B_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *Bs*$\rangle\rangle\rangle\rangle$›
  by (*induct As*, *simp-all*) (*induct Bs*, *simp-all*)

8.1.2 First Results

lemma τ*-iterated-combine-Sync*$_{ptick}$:
  ‹τ $\langle\langle{}_d\bigotimes [\![E]\!]_{\checkmark}$ *As*$\rangle\rangle$ = τ $\langle\langle{}_d\bigotimes [\![E]\!]$ *As*$\rangle\rangle$› ‹τ $\langle\langle{}_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *Bs*$\rangle\rangle$ = τ $\langle\langle{}_{nd}\bigotimes [\![E]\!]$ *Bs*$\rangle\rangle$›
  by (*intro ext*, *induct rule*: *induct-list012*;
    simp add: σ*-*σ*s-conv-defs singl-list-conv-defs*
    combine$_{Rlist}$*-Sync-defs combine*$_{Rlist}$*-Sync*$_{ptick}$*-defs* ε*-simps*)+

corollary *ε-iterated-combine-Sync$_{ptick}$*:
‹ε $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\!\rangle$ $\sigma s = \varepsilon$ $\langle\!\langle_d\bigotimes [\![E]\!]$ $As\rangle\!\rangle$ σs›
‹ε $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $Bs\rangle\!\rangle$ $\sigma s = \varepsilon$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ $Bs\rangle\!\rangle$ σs›
by (*simp-all add*: *ε-simps τ-iterated-combine-Sync$_{ptick}$*)

corollary *$\mathcal{R}$-iterated-combine-Sync$_{ptick}$*:
‹$\mathcal{R}_d$ $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\!\rangle = \mathcal{R}_d$ $\langle\!\langle_d\bigotimes [\![E]\!]$ $As\rangle\!\rangle$› ‹$\mathcal{R}_{nd}$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $Bs\rangle\!\rangle = \mathcal{R}_{nd}$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]$ $Bs\rangle\!\rangle$›
by (*intro ext same-τ-implies-same-$\mathcal{R}_d$ same-τ-implies-same-$\mathcal{R}_{nd}$*,
simp add: *τ-iterated-combine-Sync$_{ptick}$*)+

lemma *combine$_{ListslenL}$-Sync$_{ptick}$-combine$_{Rlist}$-Sync$_{ptick}$-eq*:
‹ε $\langle\!\langle_d\langle\!\langle A_0\rangle\!\rangle_{singl\hookrightarrow list}$ $_d\otimes[\![1,\ E]\!]_{\checkmark ListslenL}$ $A_1\rangle\!\rangle$ $\sigma s = \varepsilon$ $\langle\!\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Rlist}$ $A_1\rangle\!\rangle$ σs›
‹τ $\langle\!\langle_d\langle\!\langle A_0\rangle\!\rangle_{singl\hookrightarrow list}$ $_d\otimes[\![1,\ E]\!]_{\checkmark ListslenL}$ $A_1\rangle\!\rangle$ $(s_0 \# \sigma s)$ $e = \tau$ $\langle\!\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Rlist}$ $A_1\rangle\!\rangle$ $(s_0 \# \sigma s)$ e›
‹ε $\langle\!\langle_{nd}\langle\!\langle B_0\rangle\!\rangle_{singl\hookrightarrow list}$ $_{nd}\otimes[\![1,\ E]\!]_{\checkmark ListslenL}$ $B_1\rangle\!\rangle$ $\sigma s = \varepsilon$ $\langle\!\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Rlist}$ $B_1\rangle\!\rangle$ σs›
‹τ $\langle\!\langle_{nd}\langle\!\langle B_0\rangle\!\rangle_{singl\hookrightarrow list}$ $_{nd}\otimes[\![1,\ E]\!]_{\checkmark ListslenL}$ $B_1\rangle\!\rangle$ $(s_0 \# \sigma s)$ $e = \tau$ $\langle\!\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Rlist}$ $B_1\rangle\!\rangle$ $(s_0 \# \sigma s)$ e›
by (*simp-all add*: *ε-combine$_{ListslenL}$-Sync$_{ptick}$ ε-combine$_{Rlist}$-Sync$_{ptick}$ drop-Suc combine-Sync-ε-def*,
auto simp add: *combine$_{ListslenL}$-Sync$_{ptick}$-defs combine$_{Rlist}$-Sync$_{ptick}$-defs singl-list-conv-defs ε-simps*)
(*metis append-Cons append-Nil*)

lemma *combine$_{Pairlist}$-Sync$_{ptick}$-and-iterated-combine$_{nd}$-Sync$_{ptick}$-eq*:
‹ε $\langle\!\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Pairlist}$ $A_1\rangle\!\rangle$ $[s_0,\ s_1] = \varepsilon$ $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $[A_0,\ A_1]\rangle\!\rangle$ $[s_0,\ s_1]$›
‹τ $\langle\!\langle A_0$ $_d\otimes[\![E]\!]_{\checkmark Pairlist}$ $A_1\rangle\!\rangle$ $[s_0,\ s_1]$ $e = \tau$ $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $[A_0,\ A_1]\rangle\!\rangle$ $[s_0,\ s_1]$ e›
‹ε $\langle\!\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}$ $B_1\rangle\!\rangle$ $[s_0,\ s_1] = \varepsilon$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $[B_0,\ B_1]\rangle\!\rangle$ $[s_0,\ s_1]$›
‹τ $\langle\!\langle B_0$ $_{nd}\otimes[\![E]\!]_{\checkmark Pairlist}$ $B_1\rangle\!\rangle$ $[s_0,\ s_1]$ $e = \tau$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $[B_0,\ B_1]\rangle\!\rangle$ $[s_0,\ s_1]$ e›
by (*simp-all add*: *ε-combine$_{Pairlist}$-Sync$_{ptick}$ ε-combine$_{Rlist}$-Sync$_{ptick}$*)
(*auto simp add*: *combine$_{Pairlist}$-Sync$_{ptick}$-defs combine$_{Rlist}$-Sync$_{ptick}$-defs singl-list-conv-defs*
option.case-eq-if ε-simps combine-Sync-ε-def)

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$_{ptick}$*:
‹$(\bigwedge A.\ A \in set\ As \Longrightarrow finite\text{-}trans\ A) \Longrightarrow finite\text{-}trans$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\!\rangle$›
by (*induct As rule*: *induct-list012*)

(*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-*$combine_d$-$Sync_{ptick}$:
‹($\bigwedge A.\ A \in set\ As \Longrightarrow$ *ϱ-disjoint-ε* A) $\Longrightarrow$ *ϱ-disjoint-ε* $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}\ As\rangle\!\rangle$›
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-*$combine_{nd}$-$Sync_{ptick}$:
‹$\forall B \in set\ Bs.$ *ϱ-disjoint-ε* $B \Longrightarrow$ *ϱ-disjoint-ε* $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}\ Bs\rangle\!\rangle$›
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-*$combine_d$-$Sync_{ptick}$:
‹*indep-enabl* A_0 s_0 E $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}\ As\rangle\!\rangle$ σs›
if ‹*length* σs = *length As*›
‹$\bigwedge i\ j.\ [\![i \le$ *length As*; $j \le$ *length As*; $i \ne j]\!] \Longrightarrow$
indep-enabl $((A_0\ \#\ As)\ !\ i)$ $((s_0\ \#\ \sigma s)\ !\ i)$ E $((A_0\ \#\ As)\ !\ j)$ $((s_0\ \#\ \sigma s)\ !\ j)$›
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-*$combine_d$-$Sync_{ptick}$:
‹*length* σs = *length As* $\Longrightarrow$
ω $\langle\!\langle_d\bigotimes [\![E]\!]_{\checkmark}\ As\rangle\!\rangle$ σs = (*if As* = [] *then* $\Diamond$ *else those* (*map2* ω *As* σs))›
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-*$combine_{nd}$-$Sync_{ptick}$:
‹*length* σs = *length As* $\Longrightarrow$
ω $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}\ As\rangle\!\rangle$ σs =
(*if As* = [] *then* {} *else* {*rs*. *length rs* = *length As* $\wedge$ ($\forall i <$ *length As*. *rs* ! $i \in \omega$ (*As* ! i) (σs ! i))}›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** *simp*
next
case (*single σ1 A1*)
from *length-Suc-conv* **show** *?case*
by (*auto simp add*: *singl-list-conv-defs*)
next
case (*Cons σ1 σ2 σs A1 A2 As*)
show *?case* (**is** ‹- = *?rhs σ1 σ2 σs A1 A2 As*›)
proof (*intro subset-antisym subsetI*)

fix *rs* **assume** ‹$rs \in \omega$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $A1$ # $A2$ # $As\rangle\!\rangle$ ($\sigma 1$ # $\sigma 2$ # σs)›
then obtain *r1 rs′* **where** ‹$rs = r1$ # rs'› ‹$r1 \in \omega$ $A1$ $\sigma 1$›
‹$rs' \in \omega$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $A2$ # $As\rangle\!\rangle$ ($\sigma 2$ # σs)›
by (*auto simp add*: $combine_{Rlist}$-$Sync_{ptick}$-*defs*)
from *this*(*3*) *Cons.hyps*(*3*) **obtain** *r2 rs″*
where ‹$rs' = r2$ # rs''› ‹*length rs″* = *length As*›
‹$\forall$ *i*<*Suc* (*length As*). (*r2* # *rs″*) ! *i* $\in \omega$ ((*A2* # *As*) ! *i*) (($\sigma 2$ # σs) ! *i*)›
by *simp* (*metis* (*no-types*, *lifting*) *length-Suc-conv*)
with ‹$r1 \in \omega$ $A1$ $\sigma 1$› **show** ‹$rs \in$ *?rhs* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ As›
by (*auto simp add*: ‹$rs = r1$ # rs'› *less-Suc-eq-0-disj*)
next
from *Cons.hyps*(*3*)
show ‹$rs \in$ *?rhs* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ $As \Longrightarrow$
$rs \in \omega$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ $A1$ # $A2$ # $As\rangle\!\rangle$ ($\sigma 1$ # $\sigma 2$ # σs)› **for** *rs*
by (*cases rs*; *cases* ‹*tl rs*›, *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*:
‹*length* σs = *length As* $\Longrightarrow$
ω $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *map* (λA. $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$) $As\rangle\!\rangle$ $\sigma s = \omega$ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ σs›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil*
show *?case* **by** (*simp add*: *base-trans-det-ndet-conv*(*1*))
next
case (*single* $\sigma 1$ $A1$)
show *?case* **by** (*simp add*: *from-det-to-ndet-singl-list-conv-commute*)
next
case (*Cons* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ 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*:
‹*length* σs = *length As* $\Longrightarrow$
($\bigwedge i\ j$. $[\![i <$ *length As*; $j <$ *length As*; $i \neq j]\!]$
$\Longrightarrow$ *indep-enabl* (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)) $\Longrightarrow$
τ $\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}$ σs $e = \tau$ $\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *map* (λA. $\langle\!\langle A\rangle\!\rangle_{d\hookrightarrow nd}$) $As\rangle\!\rangle$ σs e›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil*
show *?case* **by** *simp*
next
case (*single* $\sigma 1$ $A1$)
show *?case* **by** (*simp add*: *from-det-to-ndet-singl-list-conv-commute*(*1*))
next
case (*Cons* $\sigma 1$ $\sigma 2$ σs $A1$ $A2$ As)

have $*$: ‹τ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $A2$ # $As\rangle\rangle\rangle_{d\hookrightarrow nd}$ ($\sigma 2$ # σs) e =
τ $\langle\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ map ($\lambda A.$ $\langle\langle A\rangle\rangle_{d\hookrightarrow nd}$) ($A2$ # As)$\rangle\rangle$ ($\sigma 2$ # σs) e›
proof (*rule Cons.hyps*(*3*))
show ‹$[\![i < length\ (A2\ \#\ As);\ j < length\ (A2\ \#\ As);\ i \neq j]\!]$
$\Longrightarrow$ *indep-enabl* (($A2$ # As) ! i) (($\sigma 2$ # σs) ! i) E
(($A2$ # As) ! j) (($\sigma 2$ # σs) ! j)› **for** i j
using *Cons.prems*[*of* ‹*Suc i*› ‹*Suc j*›] **by** *simp*
qed
have ‹τ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $A1$ # $A2$ # $As\rangle\rangle\rangle_{d\hookrightarrow nd}$ ($\sigma 1$ # $\sigma 2$ # σs) e =
τ $\langle\langle\langle A1\rangle\rangle_{d\hookrightarrow nd}$ $_{nd}\otimes[\![E]\!]_{\checkmark Rlist}$ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $A2$ # $As\rangle\rangle\rangle_{d\hookrightarrow nd}\rangle$ ($\sigma 1$ # $\sigma 2$ # σs) e›
proof (*subst iterated-combine*$_d$*-Sync*$_{ptick}$*.simps*(*3*), *rule* τ*-combine*$_{Rlist}$*-Sync*$_{ptick}$*-behaviour-when-indep*)
show ‹ε $A1$ $\sigma 1$ $\cap$ ε $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $A2$ # $As\rangle$ ($\sigma 2$ # σs) $\subseteq$ E›
proof (*rule indep-enablD*[*OF* - $\mathcal{R}_d$*.init* $\mathcal{R}_d$*.init*])
show ‹*indep-enabl* $A1$ $\sigma 1$ E $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $A2$ # $As\rangle$ ($\sigma 2$ # σs)›
by (*simp add*: *Cons.hyps*(*1*) *Cons.prems order-le-less-trans*
same-length-indep-transmission-to-iterated-combine$_d$*-Sync*$_{ptick}$)
qed
qed
also have ‹$\ldots$ = τ $\langle\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ map ($\lambda A.$ $\langle\langle A\rangle\rangle_{d\hookrightarrow nd}$) ($A1$ # $A2$ # As)$\rangle\rangle$ ($\sigma 1$ # $\sigma 2$ # σs) e›
by (*use* $*$ **in** ‹*simp add*: *combine*$_{Rlist}$*-Sync*$_{ptick}$*-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*$_{ptick}$*-behaviour-when-indep*:
assumes *same-length*: ‹*length* σs = *length* As›
and *indep*: ‹$\bigwedge i\ j.$ $[\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
indep-enabl (As ! i) (σs ! i) E (As ! j) (σs ! j)›
shows ‹$P_{SKIPS}\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle\rangle_d$ σs = $P_{SKIPS}\langle\langle\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ map ($\lambda A.$ $\langle\langle A\rangle\rangle_{d\hookrightarrow nd}$) $As\rangle\rangle\rangle_{nd}$ σs›
proof (*fold* P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*, *rule* P_{SKIPS}*-nd-eqI-strong-id*)
show ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$
τ $\langle\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ map ($\lambda A.$ $\langle\langle A\rangle\rangle_{d\hookrightarrow nd}$) $As\rangle$ $\sigma s'$ e = τ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s'$ e› **for** $\sigma s'$ e
proof (*rule* τ*-iterated-combine*$_{nd}$*-Sync*$_{ptick}$*-behaviour-when-indep*[*symmetric*])
show ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$ *length* $\sigma s'$ = *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 ‹$[\![\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle\rangle_{d\hookrightarrow nd}$ σs; $i < length\ As$; $j < length\ As$; $i \neq j]\!]$
$\Longrightarrow$ *indep-enabl* (As ! i) ($\sigma s'$! i) E (As ! j) ($\sigma s'$! j)› **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 ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$
ω $\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *map* ($\lambda A.$ $\langle A\rangle_{d\hookrightarrow nd}$) $As\rangle$ $\sigma s' = \omega$ $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s'$›
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*: ‹*length* σs = *length* As›
and *indep*: ‹$\bigwedge i\ j.$ $[\![i <$ *length* As; $j <$ *length* As; $i \neq j]\!] \Longrightarrow$
indep-enabl (As ! i) (σs ! i) E (As ! j) (σs ! j)›
shows ‹$P\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_d$ $\sigma s = P\langle\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *map* ($\lambda A.$ $\langle A\rangle_{d\hookrightarrow nd}$) $As\rangle\rangle_{nd}$ σs›
proof (*fold P-nd-from-det-to-ndet-is-P-d*, *rule P-nd-eqI-strong-id*)
show ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$
τ $\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}$ *map* ($\lambda A.$ $\langle A\rangle_{d\hookrightarrow nd}$) $As\rangle$ $\sigma s'$ $e = \tau$ $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s'$ e› **for** $\sigma s'$ e
proof (*rule* τ*-iterated-combine*$_{nd}$*-Sync*$_{ptick}$*-behaviour-when-indep*[*symmetric*])
show ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$ *length* $\sigma s'$ = *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 ‹$[\![\sigma s' \in \mathcal{R}_{nd}$ $\langle\langle_d\bigotimes [\![E]\!]_{\checkmark}$ $As\rangle\rangle_{d\hookrightarrow nd}$ σs; $i <$ *length* As; $j <$ *length* As; $i \neq j]\!]$
$\Longrightarrow$ *indep-enabl* (As ! i) ($\sigma s'$! i) E (As ! j) ($\sigma s'$! j)› **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 :: ‹*'a set*›
assumes ϱ*-disjoint-*ε : ‹ϱ*-disjoint-*ε A_0› ‹ϱ*-disjoint-*ε A_1›
defines *A-def*: ‹$A \equiv \langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle$›
defines *P-def*: ‹$P \equiv P_{SKIPS}\langle A_0\rangle_{nd}$› **and** *Q-def*: ‹$Q \equiv P_{SKIPS}\langle A_1\rangle_{nd}$› **and** *S-def*: ‹$S \equiv P_{SKIPS}\langle A\rangle_{nd}$›
shows ‹P σ_0 $[\![E]\!]_{\checkmark Pair}$ Q $\sigma_1 = S$ (σ_0, σ_1)›
proof −

let *?f* = ‹P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) ($\lambda\sigma'$. *case* σ' *of* (σ_0, σ_1) $\Rightarrow$ *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1)›
note *cartprod-rwrt* = *GlobalNdet-cartprod*[*of* - - ‹λx *y*. - (*x*, *y*)›, *simplified*]
note *Ndet-and-Sync*$_{Pair}$ = *Sync*$_{Pair}$.*Sync*$_{ptick}$*-distrib-GlobalNdet-left*
Sync$_{Pair}$.*Sync*$_{ptick}$*-distrib-GlobalNdet-right*
note *Mprefix-Sync*$_{Pair}$*-constant* =
Sync$_{Pair}$.*SKIP-Sync*$_{ptick}$*-Mprefix Sync*$_{Pair}$.*Mprefix-Sync*$_{ptick}$*-SKIP*
Sync$_{Pair}$.*STOP-Sync*$_{ptick}$*-Mprefix Sync*$_{Pair}$.*Mprefix-Sync*$_{ptick}$*-STOP*
note *P-rec* = *restriction-fix-eq*[*OF* P_{SKIPS}*-nd-step-constructive-bis*[*of* A_0], *folded* P_{SKIPS}*-nd-def P-def*, *THEN fun-cong*]
note *Q-rec* = *restriction-fix-eq*[*OF* P_{SKIPS}*-nd-step-constructive-bis*[*of* A_1], *folded* P_{SKIPS}*-nd-def Q-def*, *THEN fun-cong*]
have ω*-A* : ‹ω *A* (σ_0', σ_1') = ω A_0 σ_0' $\times$ ω A_1 σ_1'› **for** σ_0' σ_1'
by (*auto simp add*: *A-def combine*$_{Pair}$*-Sync*$_{ptick}$*-defs*)
have ε*-A* : ‹ε *A* (σ_0', σ_1') = *combine-sets-Sync* (ε A_0 σ_0') *E* (ε A_1 σ_1')› **for** σ_0' σ_1'
by (*simp add*: *A-def* ε*-combine*$_{Pair}$*-Sync*$_{ptick}$ *combine-Sync-*ε*-def*)
show ‹*P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1 = *S* (σ_0, σ_1)›
proof (*rule fun-cong*[*of* ‹λ(σ_0, σ_1). *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1› - ‹(σ_0, σ_1)›, *simplified*])
show ‹(λ(σ_0, σ_1). *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1) = *S*›
proof (*rule restriction-fix-unique*[*OF* P_{SKIPS}*-nd-step-constructive-bis*[*of A*],
symmetric, *folded* P_{SKIPS}*-nd-def S-def*])
show ‹*?f* = (λ(σ_0, σ_1). *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1)›
proof (*rule ext*, *clarify*)
have ϱ*-disjoint-*ε*-bis* : ‹ω A_0 σ_0 $\neq$ {} $\Longrightarrow$ ε A_0 σ_0 = {}›
‹ω A_1 σ_1 $\neq$ {} $\Longrightarrow$ ε A_1 σ_1 = {}› **for** σ_0 σ_1
by (*simp-all add*: ϱ*-simps* ϱ*-disjoint-*ε*D* ϱ*-disjoint-*ε)
show ‹*?f* (σ_0, σ_1) = *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1› **for** σ_0 σ_1
proof (*cases* ‹ω A_0 σ_0 = {}›; *cases* ‹ω A_1 σ_1 = {}›)
assume ‹ω A_0 σ_0 = {}› ‹ω A_1 σ_1 = {}›
hence *P-rec*' : ‹*P* σ_0 = *P-nd-step* (ε A_0) (τ A_0) *P* σ_0›
and *Q-rec*' : ‹*Q* σ_1 = *P-nd-step* (ε A_1) (τ A_1) *Q* σ_1›
and *S-rec*' : ‹P_{SKIPS}*-nd-step* (ε *A*) (τ *A*) (ω *A*) (λ(σ_0, σ_1). *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1) (σ_0, σ_1) =
P-nd-step (ε *A*) (τ *A*) (λ(σ_0, σ_1). *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1) (σ_0, σ_1)›
by (*simp-all add*: *P-rec*[*of* σ_0] *Q-rec*[*of* σ_1] ω*-A*)
show ‹*?f* (σ_0, σ_1) = *P* σ_0 $[\![E]\!]_{\checkmark Pair}$ *Q* σ_1›
unfolding *P-rec*' *Q-rec*' *S-rec*' *Sync*$_{Pair}$.*Mprefix-Sync*$_{ptick}$*-Mprefix-for-procomata*
unfolding ε*-A Mprefix-Un-distrib*
by (*intro arg-cong2*[**where** *f* = ‹($\square$)›] *mono-Mprefix-eq*, *fold P-rec*' *Q-rec*',
auto simp add: *A-def Ndet-and-Sync*$_{Pair}$ *cartprod-rwrt*
combine$_{Pair}$*-Sync*$_{ptick}$*-defs* ε*-simps GlobalNdet-sets-commute*[*of* ‹τ A_1 - -›]
simp flip: *GlobalNdet-factorization-union*
intro!: *mono-GlobalNdet-eq arg-cong2*[**where** *f* = ‹($\sqcap$)›])
next
assume ‹ω A_0 σ_0 $\neq$ {}› ‹ω A_1 σ_1 = {}›
from ϱ*-disjoint-*ε*(1)* ‹ω A_0 σ_0 $\neq$ {}› **have** ‹ε A_0 σ_0 = {}›

by (*simp add*: *ϱ-disjoint-ε-def ϱ-simps*)
have ‹*?f* $(\sigma_0, \sigma_1) = \Box b \in (\varepsilon\ A_1\ \sigma_1 - E) \rightarrow (\sqcap \sigma_1' \in \tau\ A_1\ \sigma_1\ b.\ (SKIPS\ (\omega\ A_0\ \sigma_0)\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1'))$›
by (*auto simp add*: *ω-A ε-A* ‹$\omega\ A_1\ \sigma_1 = \{\}$› ‹$\varepsilon\ A_0\ \sigma_0 = \{\}$› *intro*!: *mono-Mprefix-eq*,
auto simp add: *A-def* $combine_{Pair}$-$Sync_{ptick}$-*defs* ‹$\omega\ A_0\ \sigma_0 \neq \{\}$›
‹$\varepsilon\ A_0\ \sigma_0 = \{\}$› *cartprod-rwrt P-rec*[*of* σ_0] *intro*!: *mono-GlobalNdet-eq*)
also have ‹$\ldots = P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1$›
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* ‹$\omega\ A_0\ \sigma_0 \neq \{\}$›
GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[*of* ‹$\tau\ A_1$ - -›] ‹$\omega\ A_1\ \sigma_1 = \{\}$›
intro!: *mono-Mprefix-eq mono-GlobalNdet-eq*)
finally show ‹*?f* $(\sigma_0, \sigma_1) = \ldots$› **.**
next
assume ‹$\omega\ A_0\ \sigma_0 = \{\}$› ‹$\omega\ A_1\ \sigma_1 \neq \{\}$›
from *ϱ-disjoint-ε*(*2*) ‹$\omega\ A_1\ \sigma_1 \neq \{\}$› **have** ‹$\varepsilon\ A_1\ \sigma_1 = \{\}$›
by (*simp add*: *ϱ-disjoint-ε-def ϱ-simps*)
have ‹*?f* $(\sigma_0, \sigma_1) = \Box a \in (\varepsilon\ A_0\ \sigma_0 - E) \rightarrow (\sqcap \sigma_0' \in \tau\ A_0\ \sigma_0\ a.\ (P\ \sigma_0'\ [\![E]\!]_{\checkmark Pair}\ SKIPS\ (\omega\ A_1\ \sigma_1)))$›
by (*auto simp add*: *ω-A ε-A* ‹$\omega\ A_0\ \sigma_0 = \{\}$› ‹$\varepsilon\ A_1\ \sigma_1 = \{\}$› *intro*!: *mono-Mprefix-eq*,
auto simp add: *A-def* $combine_{Pair}$-$Sync_{ptick}$-*defs* ‹$\omega\ A_1\ \sigma_1 \neq \{\}$›
‹$\varepsilon\ A_1\ \sigma_1 = \{\}$› *cartprod-rwrt Q-rec*[*of* σ_1] *intro*!: *mono-GlobalNdet-eq*)
also have ‹$\ldots = P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1$›
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* ‹$\omega\ A_1\ \sigma_1 \neq \{\}$›
GlobalNdet-Mprefix-distr GlobalNdet-sets-commute[*of* ‹$\tau\ A_0$ - -›] ‹$\omega\ A_0\ \sigma_0 = \{\}$›
intro!: *mono-Mprefix-eq mono-GlobalNdet-eq*)
finally show ‹*?f* $(\sigma_0, \sigma_1) = \ldots$› **.**
next
assume ‹$\omega\ A_0\ \sigma_0 \neq \{\}$› ‹$\omega\ A_1\ \sigma_1 \neq \{\}$›
with *ϱ-disjoint-ε* **have** ‹$\varepsilon\ A_0\ \sigma_0 = \{\}$› ‹$\varepsilon\ A_1\ \sigma_1 = \{\}$›
by (*simp-all add*: *ϱ-disjoint-ε-def ϱ-simps*)
show ‹$\omega\ A_0\ \sigma_0 \neq \{\} \Longrightarrow \omega\ A_1\ \sigma_1 \neq \{\} \Longrightarrow$ *?f* $(\sigma_0, \sigma_1) = P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1$›
by (*simp add*: ‹$\omega\ A_0\ \sigma_0 \neq \{\}$› ‹$\omega\ A_1\ \sigma_1 \neq \{\}$› *ω-A P-rec*[*of* σ_0] *Q-rec*[*of* σ_1] *SKIPS-def*
Ndet-and-$Sync_{Pair}$ *cartprod-rwrt GlobalNdet-sets-commute*[*of* ‹$\omega\ A_0$ -›])
qed
qed
qed
qed
qed

corollary *P-nd-combine*$_{Pair}$*-Sync*$_{ptick}$:
‹$P\langle\langle A_0\rangle\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P\langle\langle A_1\rangle\rangle_{nd}\ \sigma_1 = P\langle\langle\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle\rangle\rangle_{nd}\ (\sigma_0,\ \sigma_1)$›
proof −
have ‹$P\langle\langle A_0\rangle\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P\langle\langle A_1\rangle\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P_{SKIPS}\langle\langle A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\rangle_{nd}$
σ_1›
by (*simp add*: P_{SKIPS}*-nd-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\langle\langle\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\rangle\rangle\rangle_{nd}$
$(\sigma_0,\ \sigma_1)$›
by (*rule* P_{SKIPS}*-nd-combine*$_{Pair}$*-Sync*$_{ptick}$) *simp-all*
also have ‹$\langle\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\rangle = \langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}$
$A_1\rangle\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)$›
by (*auto simp add*: *combine*$_{Pair}$*-Sync*$_{ptick}$*-defs*)
also have ‹$P_{SKIPS}\langle\langle\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\rangle_{nd} = P\langle\langle\langle\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}$
$A_1\rangle\rangle\rangle\rangle_{nd}$›
by (*simp add*: P_{SKIPS}*-nd-updated-*ω)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}*-d-combine*$_{Pair}$*-Sync*$_{ptick}$:
‹$P_{SKIPS}\langle\langle A_0\rangle\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P_{SKIPS}\langle\langle A_1\rangle\rangle_d\ \sigma_1 = P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}$
$A_1\rangle\rangle\rangle\rangle_d\ (\sigma_0,\ \sigma_1)$›
if ‹*ϱ-disjoint-ε* A_0› **and** ‹*ϱ-disjoint-ε* A_1› **and** ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P_{SKIPS}\langle\langle A_0\rangle\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P_{SKIPS}\langle\langle A_1\rangle\rangle_d\ \sigma_1 = P_{SKIPS}\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}$
$\sigma_0\ [\![E]\!]_{\checkmark Pair}\ P_{SKIPS}\langle\langle\langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle_{nd}\ \sigma_1$›
by (*simp flip*: P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*)
also have ‹$\ldots = P_{SKIPS}\langle\langle\langle\langle\langle\langle A_0\rangle\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ \langle\langle A_1\rangle\rangle_{d\hookrightarrow nd}\rangle\rangle\rangle\rangle_{nd}\ (\sigma_0,\ \sigma_1)$›
by (*rule* P_{SKIPS}*-nd-combine*$_{Pair}$*-Sync*$_{ptick}$)
(*metis ϱ-disjoint-ε-def det-ndet-conv-ε*(*1*) *det-ndet-conv-ϱ*(*1*) ‹*ϱ-disjoint-ε* A_0›,
metis ϱ-disjoint-ε-def det-ndet-conv-ε(*1*) *det-ndet-conv-ϱ*(*1*) ‹*ϱ-disjoint-ε*
A_1›)
also have ‹$\ldots = P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle\rangle\rangle_d\ (\sigma_0,\ \sigma_1)$›
by (*simp add*: P_{SKIPS}*-combine*$_{Pair}$*-Sync*$_{ptick}$*-behaviour-when-indep* ‹*indep-enabl*
$A_0\ \sigma_0\ E\ A_1\ \sigma_1$›)
finally show *?thesis* **.**
qed

corollary *P-d-combine*$_{Pair}$*-Sync*$_{ptick}$:
‹$P\langle\langle A_0\rangle\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P\langle\langle A_1\rangle\rangle_d\ \sigma_1 = P\langle\langle\langle\langle A_0\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\rangle\rangle\rangle_d\ (\sigma_0,\ \sigma_1)$›
if ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P\langle\langle A_0\rangle\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P\langle\langle A_1\rangle\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P_{SKIPS}\langle\langle A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle_d\ \sigma_1$›
by (*simp add*: P_{SKIPS}*-d-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\langle\langle\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes[\![E]\!]_{\checkmark Pair}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\rangle\rangle\rangle_d$
$(\sigma_0,\ \sigma_1)$›
by (*subst* P_{SKIPS}*-d-combine*$_{Pair}$*-Sync*$_{ptick}$, *simp-all add*: *ϱ-simps ϱ-disjoint-ε-def*)
(*rule indep-enablI*,

use ‹*indep-enabl* A_0 σ_0 E A_1 σ_1›[*THEN indep-enablD*] **in** ‹*simp add*: ε-*simps*›)
also have ‹$\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \Diamond|\!)\ {}_d\otimes [\![E]\!]_{\checkmark Pair}\ A_1 (\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \Diamond|\!)$›
by (*simp add*: $combine_{Pair}$-$Sync_{ptick}$-*defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d = P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_d$›
by (*simp add*: P_{SKIPS}-*d-updated-*ω)
finally show *?thesis* **.**
qed

Pairlist

theorem P_{SKIPS}-*nd-*$combine_{Pairlist}$-$Sync_{ptick}$:
fixes E :: ‹*'a set*›
assumes ϱ-*disjoint-*ε : ‹ϱ-*disjoint-*ε A_0› ‹ϱ-*disjoint-*ε A_1›
shows ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›
proof −
let *?A* = ‹$(\!|\tau = \tau\ \langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle,\ \omega = \lambda\sigma.\ (\lambda(r,\ s).\ [r,\ s])\ `\ \omega\ \langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma|\!)$›
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
RenamingTick $(P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1)\ (\lambda(r,\ s).\ [r,\ s])$›
by (*simp only*: $Sync_{Pair}$-*to-*$Sync_{Pairlist}$)
also have ‹$\ldots$ = *RenamingTick* $(P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0, \sigma_1))\ (\lambda(r,\ s).\ [r,\ s])$›
by (*simp add*: P_{SKIPS}-*nd-*$combine_{Pair}$-$Sync_{ptick}$ ϱ-*disjoint-*ε)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle ?A\rangle\!\rangle_{nd}\ (\sigma_0, \sigma_1)$›
by (*simp only*: *RenamingTick-*P_{SKIPS}-*nd*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›
by (*auto intro*!: P_{SKIPS}-*nd-eqI-strong*[*of* ‹$\lambda(r,\ s).\ [r,\ s]$› - ‹(σ_0, σ_1)›, *simplified*] *inj-onI*)
(*auto simp add*: *combine-*$Sync_{ptick}$-*defs split*: *if-split-asm*)
finally show *?thesis* **.**
qed

corollary *P-nd-*$combine_{Pairlist}$-$Sync_{ptick}$:
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle A_1 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp add*: P_{SKIPS}-*nd-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pairlist}\ A_1 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›
by (*rule* P_{SKIPS}-*nd-*$combine_{Pairlist}$-$Sync_{ptick}$) *simp-all*
also have ‹$\langle\!\langle A_0 (\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pairlist}\ A_1 (\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle = \langle\!\langle A_0\ {}_{nd}\otimes [\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma.\ \{\}|\!)$›

by (*simp add*: $combine_{Pairlist}$-$Sync_{ptick}$-*defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd} = P\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}$›
by (*simp add*: P_{SKIPS}-*nd-updated-ω*)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}-*d*-$combine_{Pairlist}$-$Sync_{ptick}$:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
if ‹*ϱ-disjoint-ε* A_0› **and** ‹*ϱ-disjoint-ε* A_1› **and** ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp flip*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ _{nd}\otimes[\![E]\!]_{\checkmark Pairlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ [\sigma_0,\ \sigma_1]$›
by (*rule* P_{SKIPS}-*nd-*$combine_{Pairlist}$-$Sync_{ptick}$)
(*metis ϱ-disjoint-ε-def det-ndet-conv-ε*(*1*) *det-ndet-conv-ϱ*(*1*) ‹*ϱ-disjoint-ε* A_0›,
metis ϱ-disjoint-ε-def det-ndet-conv-ε(*1*) *det-ndet-conv-ϱ*(*1*) ‹*ϱ-disjoint-ε* A_1›)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
by (*simp add*: P_{SKIPS}-$combine_{Pairlist}$-$Sync_{ptick}$-*behaviour-when-indep* ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›)
finally show *?thesis* **.**
qed

corollary *P-d-*$combine_{Pairlist}$-$Sync_{ptick}$:
‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
if ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Pairlist}\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_1$›
by (*simp add*: P_{SKIPS}-*d-updated-ω*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0,\ \sigma_1]$›
by (*subst* P_{SKIPS}-*d-*$combine_{Pairlist}$-$Sync_{ptick}$, *simp-all*)
(*rule indep-enablI*,
use ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›[*THEN indep-enablD*] **in** *simp*)
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle = \langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)$›
by (*simp add*: $combine_{Pairlist}$-$Sync_{ptick}$-*defs*, *intro ext*, *simp*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d = P\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d$›
by (*simp add*: P_{SKIPS}-*d-updated-ω*)
finally show *?thesis* **.**
qed

8.2.2 Rlist

theorem P_{SKIPS}-*nd-combine*$_{Rlist}$-*Sync*$_{ptick}$:
fixes E :: ‹$'a$ *set*›
assumes *ϱ-disjoint-ε* : ‹*ϱ-disjoint-ε* A_0› ‹*ϱ-disjoint-ε* A_1›
defines *A-def*: ‹$A \equiv \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle$›
defines *P-def*: ‹$P \equiv P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}$› **and** *Q-def*: ‹$Q \equiv P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}$› **and** *S-def*: ‹$S \equiv P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$›
shows ‹$P\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ Q\ \sigma s = S\ (\sigma_0\ \#\ \sigma s)$›
proof −
let $?A' =$ ‹$(\!|\tau = \tau\ \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle,\ \omega = \lambda\sigma.\ (\lambda(x,\ y).\ x\ \#\ y)\ `\ \omega\ \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma|\!)$›
from P_{SKIPS}-*nd-combine*$_{Pair}$-*Sync*$_{ptick}$[*OF ϱ-disjoint-ε*]
have ‹$P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma s = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma s)$› **by** (*simp add*: *P-def Q-def*)
hence ‹*RenamingTick* $(P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma s)\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s) =$
RenamingTick $(P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma s))\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s)$› **by** *simp*
also have ‹*RenamingTick* $(P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma s)\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s) = P\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ Q\ \sigma s$›
by (*auto intro*: *inj-onI* $Sync_{Pair}$.*inj-on-RenamingTick-Sync*$_{ptick}$
[*of* ‹$\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s$›, *simplified*])
also have ‹*RenamingTick* $(P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma s))\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s) = S\ (\sigma_0\ \#\ \sigma s)$›
proof (*unfold RenamingTick*-P_{SKIPS}-*nd S-def*,
rule P_{SKIPS}-*nd-eqI-strong*[*of* ‹$\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s$› $?A'$ ‹$(\sigma_0,\ \sigma s)$›, *simplified*])
show ‹*inj-on* $(\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ \#\ \sigma s)\ (\mathcal{R}_{nd}\ ?A'\ (\sigma_0,\ \sigma s))$› **by** (*auto intro*: *inj-onI*)
next
show ‹$\tau\ A\ (case\ \sigma'\ of\ (\sigma_0,\ \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s)\ e =$
$(\lambda\sigma''.\ case\ \sigma''\ of\ (\sigma_0,\ \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s)\ `\ \tau\ \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma'\ e$› **for** $\sigma'\ e$
by (*cases* σ') (*auto simp add*: *A-def combine*$_{Rlist}$-*Sync*$_{ptick}$-*defs combine*$_{Pair}$-*Sync*$_{ptick}$-*defs*)
next
show ‹$\omega\ A\ (case\ \sigma'\ of\ (\sigma_0,\ \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s) =$
$(\lambda\sigma''.\ case\ \sigma''\ of\ (\sigma_0,\ \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s)\ `\ \omega\ \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma'$› **for** σ'
by (*cases* σ') (*auto simp add*: *A-def combine*$_{Rlist}$-*Sync*$_{ptick}$-*defs combine*$_{Pair}$-*Sync*$_{ptick}$-*defs*)
qed
finally show ‹$P\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ Q\ \sigma s = S\ (\sigma_0\ \#\ \sigma s)$› **.**
qed

corollary *P-nd-combine*$_{Rlist}$-*Sync*$_{ptick}$:
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma s = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma s)$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma s =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma s$›
by (*simp add*: P_{SKIPS}-*nd-updated-ω*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ {}_{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma s)$›

by (*rule* P_{SKIPS}*-nd-combine*$_{Rlist}$*-Sync*$_{ptick}$)
(*simp-all add*: *ϱ-simps ϱ-disjoint-ε-def*)
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \{\}|\!)\ _{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle = \langle\!\langle A_0\ _{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)$›
by (*simp add*: *combine*$_{Rlist}$*-Sync*$_{ptick}$*-defs*, *intro ext*, *simp add*: *ε-simps*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \{\}|\!)\rangle\!\rangle_{nd} = P\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}$›
by (*simp add*: P_{SKIPS}*-nd-updated-ω*)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}*-d-combine*$_{Rlist}$*-Sync*$_{ptick}$:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma s = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ \#\ \sigma s)$›
if ‹*ϱ-disjoint-ε* A_0› **and** ‹*ϱ-disjoint-ε* A_1› **and** ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma s$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma s = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma s$›
by (*simp flip*: P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ _{nd}\otimes[\![E]\!]_{\checkmark Rlist}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma s)$›
by (*rule* P_{SKIPS}*-nd-combine*$_{Rlist}$*-Sync*$_{ptick}$)
(*metis ϱ-disjoint-ε-def det-ndet-conv-ε*(*1*) *det-ndet-conv-ϱ*(*1*) ‹*ϱ-disjoint-ε* A_0›,
metis ϱ-disjoint-ε-def det-ndet-conv-ε(*1*) *det-ndet-conv-ϱ*(*1*) ‹*ϱ-disjoint-ε* A_1›)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ \#\ \sigma s)$›
by (*simp add*: P_{SKIPS}*-combine*$_{Rlist}$*-Sync*$_{ptick}$*-behaviour-when-indep* ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma s$›)
finally show *?thesis* **.**
qed

corollary *P-d-combine*$_{Rlist}$*-Sync*$_{ptick}$:
‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma s = P\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ \#\ \sigma s)$›
if ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma s$›
proof −
have ‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma s =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_0\ [\![E]\!]_{\checkmark Rlist}\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma s$›
by (*simp add*: P_{SKIPS}*-d-updated-ω*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ \#\ \sigma s)$›
by (*rule* P_{SKIPS}*-d-combine*$_{Rlist}$*-Sync*$_{ptick}$, *simp-all add*: *ϱ-simps ϱ-disjoint-ε-def*)
(*rule indep-enablI*,
use ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma s$›[*THEN indep-enablD*] **in** ‹*simp add*: *ε-simps*›)
also have ‹$\langle\!\langle A_0(\!|\omega := \lambda\sigma.\ \Diamond|\!)\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle = \langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)$›
by (*simp add*: *combine*$_{Rlist}$*-Sync*$_{ptick}$*-defs*, *intro ext*, *simp add*: *ε-simps*)
also have ‹$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma.\ \Diamond|\!)\rangle\!\rangle_d = P\langle\!\langle\langle\!\langle A_0\ _d\otimes[\![E]\!]_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d$›
by (*simp add*: P_{SKIPS}*-d-updated-ω*)
finally show *?thesis* **.**

qed

8.2.3 ListslenL

theorem P_{SKIPS}-*nd-combine*$_{ListslenL}$-*Sync*$_{ptick}$:
fixes E :: ‹*'a set*›
assumes *same-length-reach0* : ‹$\bigwedge\sigma_0{}'.\ \sigma_0{}' \in \mathcal{R}_{nd}\ A_0\ \sigma_0 \Longrightarrow$ *length* $\sigma_0{}' = len_0$›
and *same-length-term0* : ‹$\bigwedge\sigma_0{}'$ *rs*. $\sigma_0{}' \in \mathcal{R}_{nd}\ A_0\ \sigma_0 \Longrightarrow rs \in \omega\ A_0\ \sigma_0{}' \Longrightarrow$ *length rs* $= len_0$›
and ϱ-*disjoint*-ε : ‹ϱ-*disjoint*-ε A_0› ‹ϱ-*disjoint*-ε A_1›
defines *A-def*: ‹$A \equiv \langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle$›
defines *P-def*: ‹$P \equiv P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}$› **and** *Q-def*: ‹$Q \equiv P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_{nd}$› **and** *S-def*: ‹$S \equiv P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$›
shows ‹$P\ \sigma_0\ {}_{len_0}[\![E]\!]_{\checkmark ListslenL}\ Q\ \sigma_1 = S\ (\sigma_0\ @\ \sigma_1)$›
proof −
let $?A' =$ ‹$(\!|\tau = \tau\ \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle,\ \omega = \lambda\sigma.\ (\lambda(x,\ y).\ x\ @\ y)\ `\ \omega\ \langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma|\!)$›
let $?RT = RenamingTick$
have $*$: ‹$\sigma' \in \mathcal{R}_{nd}\ ?A'\ (\sigma_0,\ \sigma_1) \Longrightarrow$ *length* (*fst* σ') $= len_0$› **for** σ'
by (*metis* (*no-types*, *lifting*) $\mathcal{R}_{nd}$-*combine*$_{ndPair}$-*Sync*$_{ptick}$-*subset mem-Sigma-iff prod.collapse*
same-τ*-implies-same-*$\mathcal{R}_{nd}$ *same-length-reach0 select-convs*(*1*) *subset-iff*)

from P_{SKIPS}-*nd-combine*$_{Pair}$-*Sync*$_{ptick}$[*OF* ϱ-*disjoint*-ε]
have ‹$P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1 = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma_1)$›
by (*simp add*: *P-def Q-def*)

hence ‹$?RT\ (P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1)\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ @\ \sigma s) =$
$?RT\ (P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma_1))\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ @\ \sigma s)$›
by *simp*
also have ‹$?RT\ (P\ \sigma_0\ [\![E]\!]_{\checkmark Pair}\ Q\ \sigma_1)\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ @\ \sigma s) = P\ \sigma_0\ {}_{len_0}[\![E]\!]_{\checkmark ListslenL}\ Q\ \sigma_1$›
by (*rule* $Sync_{Pair}$-*to-*$Sync_{ListslenL}$[*OF is-ticks-length-*P_{SKIPS}-*nd*[*of* A_0, *folded P-def*]])
(*fact same-length-term0*)
also have ‹$?RT\ (P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![E]\!]_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0,\ \sigma_1))\ (\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ @\ \sigma s) = S\ (\sigma_0\ @\ \sigma_1)$›
by (*auto simp add*: *RenamingTick-*P_{SKIPS}-*nd S-def dest*!: $*$
intro!: P_{SKIPS}-*nd-eqI-strong*[*of* ‹$\lambda(\sigma_0,\ \sigma s).\ \sigma_0\ @\ \sigma s$› $?A'$ ‹$(\sigma_0,\ \sigma_1)$›, *simplified*] *inj-onI*)
(*force simp add*: *image-iff A-def* $combine_{ListslenL}$-$Sync_{ptick}$-*defs* $combine_{Pair}$-$Sync_{ptick}$-*defs split*: *if-split-asm*)+
finally show *?thesis* **.**
qed

corollary *P-nd-*$combine_{ListslenL}$-$Sync_{ptick}$:
‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ {}_{len_0}[\![E]\!]_{\checkmark ListslenL}\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 = P\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
if *same-length-reach0* : ‹$\bigwedge\sigma_0{}'.\ \sigma_0{}' \in \mathcal{R}_{nd}\ A_0\ \sigma_0 \Longrightarrow$ *length* $\sigma_0{}' = len_0$›

proof −
have $*$: ‹$\sigma s \in \mathcal{R}_{nd}$ $\langle\!\langle A_0 (\!|\omega := \lambda\sigma_0.\ \{\}|\!)\ _{nd}\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1 (\!|\omega := \lambda\sigma_1.\ \{\}|\!)\rangle\!\rangle$ $(\sigma_0\ @\ \sigma_1) \Longrightarrow len_0 \leq length\ \sigma s$› **for** σs
by (*auto dest*: *set-rev-mp*[*OF* - $\mathcal{R}_{nd}$-*combine*$_{ndListslenL}$-*Sync*$_{ptick}$-*subset*]
simp add: *same-length-reach0*)
have ‹$P\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ _{len_0}[\![E]\!]_{\checkmark ListslenL}\ P\langle\!\langle A_1\rangle\!\rangle_{nd}\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0 (\!|\omega := \lambda\sigma_0.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_0\ _{len_0}[\![E]\!]_{\checkmark ListslenL}\ P_{SKIPS}\langle\!\langle A_1 (\!|\omega := \lambda\sigma_1.\ \{\}|\!)\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp only*: P_{SKIPS}-*nd-updated-*ω)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0 (\!|\omega := \lambda\sigma_0.\ \{\}|\!)\ _{nd}\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1 (\!|\omega := \lambda\sigma_1.\ \{\}|\!)\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*auto intro*: P_{SKIPS}-*nd-combine*$_{ListslenL}$-*Sync*$_{ptick}$ *simp add*: *same-length-reach0*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle (\!|\omega := \lambda\sigma_1.\ \{\}|\!)\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*auto intro*!: P_{SKIPS}-*nd-eqI-strong-id dest*!: $*$)
(*auto simp add*: *combine*$_{ListslenL}$-*Sync*$_{ptick}$-*defs split*: *if-split-asm*)
also have ‹$\ldots = P\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*simp only*: P_{SKIPS}-*nd-updated-*ω)
finally show *?thesis* **.**
qed

corollary P_{SKIPS}-*d-combine*$_{ListslenL}$-*Sync*$_{ptick}$:
assumes *same-length-reach0* : ‹$\bigwedge\sigma_0'.\ \sigma_0' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow length\ \sigma_0' = len_0$›
and *same-length-term0* : ‹$\bigwedge\sigma_0'.\ \sigma_0' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow \omega\ A_0\ \sigma_0' \neq \Diamond \Longrightarrow length\ \lceil\omega\ A_0\ \sigma_0'\rceil = len_0$›
and ϱ-*disjoint-*ε : ‹ϱ-*disjoint-*ε A_0› ‹ϱ-*disjoint-*ε A_1›
and *indep-enabl* : ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
shows ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ _{len_0}[\![E]\!]_{\checkmark ListslenL}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
proof −
have $*$: ‹$\sigma s \in \mathcal{R}_{nd}$ $\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ _{nd}\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\ (\sigma_0\ @\ \sigma_1)$
$\Longrightarrow$
$\exists\sigma_0'\ \sigma_1'.\ \sigma s = \sigma_0'\ @\ \sigma_1' \wedge \sigma_0' \in \mathcal{R}_d\ A_0\ \sigma_0 \wedge \sigma_1' \in \mathcal{R}_d\ A_1\ \sigma_1$› **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 ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ _{len_0}[\![E]\!]_{\checkmark ListslenL}\ P_{SKIPS}\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ _{len_0}[\![E]\!]_{\checkmark ListslenL}\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_1$›
by (*simp only*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also from *same-length-term0* **have** ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ _{nd}\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ \langle\!\langle A_1\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*auto intro*!: P_{SKIPS}-*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** ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\ _d\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ (\sigma_0\ @\ \sigma_1)$›
by (*auto intro*!: P_{SKIPS}-*nd-eqI-strong-id*
τ-*combine*$_{ListslenL}$-*Sync*$_{ptick}$-*behaviour-when-indep* ω-*combine*$_{ListslenL}$-*Sync*$_{ptick}$-*behaviour*
simp add: *same-length-reach0 dest*!: $*$ *indep-enablD*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes [\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›

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* : ‹$\bigwedge\sigma_0{}'.\ \sigma_0{}' \in \mathcal{R}_d\ A_0\ \sigma_0 \Longrightarrow$ *length* $\sigma_0{}' = len_0$›
and *indep-enabl* : ‹*indep-enabl* $A_0\ \sigma_0\ E\ A_1\ \sigma_1$›
shows ‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ {}_{len_0}[\![E]\!]_{\checkmark ListslenL}\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
proof −
have $*$: ‹$\sigma s \in \mathcal{R}_d\ \langle\!\langle A_0(\!|\omega := \lambda\sigma_0.\ \Diamond|\!)\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1(\!|\omega := \lambda\sigma_1.\ \Diamond|\!)\rangle\!\rangle$ $(\sigma_0\ @\ \sigma_1) \Longrightarrow len_0 \le$ *length* σs› **for** σs
by (*auto dest*: *set-rev-mp*[*OF* - $\mathcal{R}_d$-*combine*$_{dListslenL}$-*Sync*$_{ptick}$-*subset*]
simp add: *same-length-reach0*)
have ‹$P\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ {}_{len_0}[\![E]\!]_{\checkmark ListslenL}\ P\langle\!\langle A_1\rangle\!\rangle_d\ \sigma_1 =$
$P_{SKIPS}\langle\!\langle A_0(\!|\omega := \lambda\sigma_0.\ \Diamond|\!)\rangle\!\rangle_d\ \sigma_0\ {}_{len_0}[\![E]\!]_{\checkmark ListslenL}\ P_{SKIPS}\langle\!\langle A_1(\!|\omega := \lambda\sigma_1.$ $\Diamond|\!)\rangle\!\rangle_d\ \sigma_1$›
by (*simp only*: P_{SKIPS}-*d-updated-*ω)
also from *indep-enabl*
have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0(\!|\omega := \lambda\sigma_0.\ \Diamond|\!)\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1(\!|\omega := \lambda\sigma_1.$ $\Diamond|\!)\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
by (*auto intro*: P_{SKIPS}-*d-combine*$_{ListslenL}$-*Sync*$_{ptick}$ *simp add*: *same-length-reach0*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle(\!|\omega := \lambda\sigma_0.\ \Diamond|\!)\rangle\!\rangle_d\ (\sigma_0$ $@\ \sigma_1)$›
by (*auto intro*!: P_{SKIPS}-*d-eqI-strong-id dest*!: $*$)
(*auto simp add*: *combine*$_{ListslenL}$-*Sync*$_{ptick}$-*defs split*: *if-split-asm*)
also have ‹$\ldots = P\langle\!\langle\langle\!\langle A_0\ {}_d\otimes[\![len_0,\ E]\!]_{\checkmark ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ @\ \sigma_1)$›
by (*simp only*: P_{SKIPS}-*d-updated-*ω)
finally show *?thesis* **.**
qed

8.2.4 Multiple

theorem P_{SKIPS}-*nd-compactification-Sync*$_{ptick}$:
‹*length* σs = *length* $As \Longrightarrow (\bigwedge A.\ A \in$ *set* $As \Longrightarrow \varrho$-*disjoint-*$\varepsilon$ $A) \Longrightarrow$
$[\![E]\!]_{\checkmark}\ (\sigma,\ A) \in@$ *zip* $\sigma s\ As.\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma = P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}\ As\rangle\!\rangle\rangle\!\rangle_{nd}$ σs›
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 ϱ-*disjoint-*ε : ‹$A \in$ *set* $(A_1\ \#\ As) \Longrightarrow \varrho$-*disjoint-*$\varepsilon$ A› **for** A
by (*simp add*: *Cons.prems*)
show *?case*
by (*simp add*: *Cons.hyps*(*3*)[*OF* ϱ-*disjoint-*ε, *simplified*] *Cons.prems*

P_{SKIPS}-nd-combine$_{Rlist}$-Sync$_{ptick}$ ϱ-disjoint-ε-transmission-to-iterated-combine$_{nd}$-Sync$_{ptick}$)
qed

corollary *P-nd-compactification-Sync$_{ptick}$*:
‹*length σs = length As* ⟹ ⟦*E*⟧$_{\checkmark}$ (σ, *A*) ∈@ *zip σs As*. *P*⟪*A*⟫$_{nd}$ σ = *P*⟪⟪$_{nd}$⨂ ⟦*E*⟧$_{\checkmark}$ *As*⟫⟫$_{nd}$ *σs*›
proof (*induct σs As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst P-nd-rec*, *simp*)
next
case (*single* σ_0 A_0) **show** *?case*
by (*simp*, *subst* (*1 2*) P_{SKIPS}-nd-updated-ω)
(*auto simp add*: *RenamingTick-*P_{SKIPS}*-nd singl-list-conv-defs*
intro!: *inj-onI* P_{SKIPS}*-nd-eqI-strong*)
next
case (*Cons* σ_0 σ_1 *σs* A_0 A_1 *As*)
show *?case*
by (*simp add*: *Cons.hyps*(*3*)[*simplified*] *Cons.prems*
P-nd-combine$_{Rlist}$-Sync$_{ptick}$ ϱ-disjoint-ε-transmission-to-iterated-combine$_{nd}$-Sync$_{ptick}$)
qed

corollary P_{SKIPS}*-d-compactification-Sync$_{ptick}$*:
‹⟦*length σs = length As*; ⋀*A*. *A* ∈ *set As* ⟹ *ϱ-disjoint-ε A*;
⋀*i j*. ⟦*i* < *length As*; *j* < *length As*; *i* ≠ *j*⟧ ⟹
indep-enabl (*As* ! *i*) (*σs* ! *i*) *E* (*As* ! *j*) (*σs* ! *j*)⟧ ⟹
⟦*E*⟧$_{\checkmark}$ (σ, *A*) ∈@ *zip σs As*. P_{SKIPS}⟪*A*⟫$_d$ σ = P_{SKIPS}⟪⟪$_d$⨂ ⟦*E*⟧$_{\checkmark}$ *As*⟫⟫$_d$ *σs*›
proof (*induct σs As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp*, *subst* P_{SKIPS}*-d-rec*, *simp*)
next
case (*single* σ_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* σ_0 σ_1 *σs* A_0 A_1 *As*)
have *ϱ-disjoint-ε* : ‹*A* ∈ *set* (A_1 # *As*) ⟹ *ϱ-disjoint-ε A*› **for** *A*
by (*simp add*: *Cons.prems*(*1*))
have *indep-enabl* :
‹⟦*i* < *length* (A_1 # *As*); *j* < *length* (A_1 # *As*); *i* ≠ *j*⟧ ⟹
indep-enabl ((A_1 # *As*) ! *i*) ((σ_1 # *σs*) ! *i*) *E* ((A_1 # *As*) ! *j*) ((σ_1 # *σs*) ! *j*)›
for *i j*
by (*metis Cons.prems*(*2*) *Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)
have ‹*ϱ-disjoint-ε* A_0› **by** (*simp add*: *Cons.prems*(*1*))
moreover have ‹*ϱ-disjoint-ε* ⟪$_d$⨂ ⟦*E*⟧$_{\checkmark}$ A_1 # *As*⟫›
by (*meson ϱ-disjoint-ε ϱ-disjoint-ε-transmission-to-iterated-combine$_d$-Sync$_{ptick}$*)
moreover have ‹*indep-enabl* A_0 σ_0 *E* ⟪$_d$⨂ ⟦*E*⟧$_{\checkmark}$ A_1 # *As*⟫ (σ_1 # *σs*)›
by (*metis Cons.hyps*(*1*) *Cons.prems*(*2*) *length-Cons less-Suc-eq-le*
same-length-indep-transmission-to-iterated-combine$_d$-Sync$_{ptick}$)
ultimately show *?case*
by (*simp add*: P_{SKIPS}*-d-combine$_{Rlist}$-Sync$_{ptick}$*
Cons.hyps(*3*)[*OF ϱ-disjoint-ε indep-enabl*, *simplified*])

qed

corollary *P-d-compactification-Sync*$_{ptick}$:
‹$[\![$*length* σs = *length As*;
$\bigwedge i\ j.\ [\![i < length\ As;\ j < length\ As;\ i \neq j]\!] \Longrightarrow$
indep-enabl (*As* ! *i*) (σs ! *i*) *E* (*As* ! *j*) (σs ! *j*)$]\!] \Longrightarrow$
$[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* σs *As*. $P\langle\!\langle A\rangle\!\rangle_d\ \sigma = P\langle\!\langle\langle\!\langle_d \bigotimes [\![E]\!]_{\checkmark}\ As\rangle\!\rangle\!\rangle_d\ \sigma s$›
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 singl-list-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* :
‹$[\![i < length\ (A_1\ \#\ As);\ j < length\ (A_1\ \#\ As);\ i \neq j]\!] \Longrightarrow$
indep-enabl ((A_1 # *As*) ! *i*) ((σ_1 # σs) ! *i*) *E* ((A_1 # *As*) ! *j*) ((σ_1 # σs) ! *j*)›
for *i j*
by (*metis Cons.prems Suc-less-eq length-Cons nat.inject nth-Cons-Suc*)
have ‹*indep-enabl* A_0 σ_0 *E* $\langle\!\langle_d \bigotimes [\![E]\!]_{\checkmark}\ A_1\ \#\ As\rangle\!\rangle$ (σ_1 # σs)›
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*:
‹$[\![E]\!]_{\checkmark}$ *P* ∈@ *map Q* [*0*..<*n*]. *P* = $P_{SKIPS}\langle\!\langle\langle\!\langle_{nd} \bigotimes [\![E]\!]_{\checkmark}$ *map A* [*0*..<*n*]$\rangle\!\rangle\!\rangle_{nd}$ (*replicate n 0*)›
if ‹$\bigwedge i.\ i < n \Longrightarrow \varrho$*-disjoint-*$\varepsilon$ (*A i*)›
‹$\bigwedge i.\ i < n \Longrightarrow P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0 = Q\ i$›
proof −
have ‹$[\![E]\!]_{\checkmark}$ *P* ∈@ *map Q* [*0*..<*n*]. *P* = $[\![E]\!]_{\checkmark}$ *i* ∈@ [*0*..<*n*]. *Q i*›
by (*auto intro*: *mono-MultiSync*$_{ptick}$*-eq2*)
also have ‹... = $[\![E]\!]_{\checkmark}$ *i* ∈@ [*0*..<*n*]. $P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$›
by (*auto simp add*: *that*(*2*) *intro*: *mono-MultiSync*$_{ptick}$*-eq*)
also have ‹... = $[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* (*replicate n 0*) (*map A* [*0*..<*n*]). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹$[\![E]\!]_{\checkmark}$ *i*∈@ [*0*..<*Suc k*]. $P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$ =
$[\![E]\!]_{\checkmark}$ *i*∈@ [*0*..<*k*]. $P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$ $[\![E]\!]_{\checkmark Llist}$ $P_{SKIPS}\langle\!\langle A\ k\rangle\!\rangle_{nd}\ 0$›
using *Suc.hyps*(*1*) **by** (*simp add*: *MultiSync*$_{ptick}$*-snoc*)
also have ‹... = $[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* (*replicate k 0*) (*map A* [*0*..<*k*]).

$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma\ [\![E]\!]_{\checkmark Llist}\ P_{SKIPS}\langle\!\langle A\ k\rangle\!\rangle_{nd}\ 0$›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹... = $[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* (*replicate* (*Suc k*) *0*) (*map A* [*0*..<*Suc k*]). $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma$›
using *Suc.hyps*(*1*) **by** (*simp flip*: *replicate-append-same add*: $MultiSync_{ptick}$-*snoc*)
finally show *?case* **.**
qed *simp-all*
also have ‹... = $P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\rangle\!\rangle_{nd}$ (*replicate n 0*)›
by (*rule* P_{SKIPS}-*nd-compactification-*$Sync_{ptick}$) (*auto simp add*: *that*(*1*))
finally show *?thesis* **.**
qed

lemma *P-nd-compactification-*$Sync_{ptick}$-*upt-version*:
‹$[\![E]\!]_{\checkmark}$ *P* ∈@ *map Q* [*0*..<*n*]. *P* = $P\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\rangle\!\rangle_{nd}$ (*replicate n 0*)›
if ‹$\bigwedge i.\ i < n \Longrightarrow P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0 = Q\ i$›
proof −
have ‹$[\![E]\!]_{\checkmark}$ *P* ∈@ *map Q* [*0*..<*n*]. *P* = $[\![E]\!]_{\checkmark}$ *i* ∈@ [*0*..<*n*]. *Q i*›
by (*auto intro*: *mono-*$MultiSync_{ptick}$-*eq2*)
also have ‹... = $[\![E]\!]_{\checkmark}$ *i* ∈@ [*0*..<*n*]. $P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$›
by (*auto simp add*: *that intro*: *mono-*$MultiSync_{ptick}$-*eq*)
also have ‹... = $[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* (*replicate n 0*) (*map A* [*0*..<*n*]). $P\langle\!\langle A\rangle\!\rangle_{nd}$ σ›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹$[\![E]\!]_{\checkmark}$ *i*∈@ [*0*..<*Suc k*]. $P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0$ =
$[\![E]\!]_{\checkmark}$ *i*∈@ [*0*..<*k*]. $P\langle\!\langle A\ i\rangle\!\rangle_{nd}\ 0\ [\![E]\!]_{\checkmark Llist}\ P\langle\!\langle A\ k\rangle\!\rangle_{nd}\ 0$›
using *Suc.hyps*(*1*) **by** (*simp add*: $MultiSync_{ptick}$-*snoc*)
also have ‹... = $[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* (*replicate k 0*) (*map A* [*0*..<*k*]). $P\langle\!\langle A\rangle\!\rangle_{nd}$ σ $[\![E]\!]_{\checkmark Llist}$ $P\langle\!\langle A\ k\rangle\!\rangle_{nd}\ 0$›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹... = $[\![E]\!]_{\checkmark}$ (σ, *A*) ∈@ *zip* (*replicate* (*Suc k*) *0*) (*map A* [*0*..<*Suc k*]). $P\langle\!\langle A\rangle\!\rangle_{nd}$ σ›
using *Suc.hyps*(*1*) **by** (*simp flip*: *replicate-append-same add*: $MultiSync_{ptick}$-*snoc*)
finally show *?case* **.**
qed *simp-all*
also have ‹... = $P\langle\!\langle\langle\!\langle_{nd}\bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\rangle\!\rangle_{nd}$ (*replicate n 0*)›
by (*rule P-nd-compactification-*$Sync_{ptick}$) *simp*
finally show *?thesis* **.**
qed

lemma P_{SKIPS}-*d-compactification-*$Sync_{ptick}$-*upt-version*:
‹$[\![E]\!]_{\checkmark}$ *P* ∈@ *map Q* [*0*..<*n*]. *P* = $P_{SKIPS}\langle\!\langle\langle\!\langle_{d}\bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\rangle\!\rangle_{d}$ (*replicate n 0*)›
if ‹$\bigwedge i.\ i < n \Longrightarrow$ *ϱ-disjoint-ε* (*A i*)›
‹$\bigwedge i\ j.\ i < n \Longrightarrow j < n \Longrightarrow i \neq j \Longrightarrow$ *indep-enabl* (*A i*) *0 E* (*A j*) *0*›
‹$\bigwedge i.\ i < n \Longrightarrow P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_{d}\ 0 = Q\ i$›
proof −
have ‹$[\![E]\!]_{\checkmark}$ *P* ∈@ *map Q* [*0*..<*n*]. *P* = $[\![E]\!]_{\checkmark}$ *i* ∈@ [*0*..<*n*]. *Q i*›

by (*auto intro*: *mono-MultiSync*$_{ptick}$*-eq2*)
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ i \in@ [0..<n].\ P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0$›
by (*auto simp add*: *that*(*3*) *intro*: *mono-MultiSync*$_{ptick}$*-eq*)
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ (\sigma, A) \in@ zip\ (replicate\ n\ 0)\ (map\ A\ [0..<n]).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹$[\![E]\!]_{\checkmark}\ i \in@ [0..<Suc\ k].\ P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0 =$
$[\![E]\!]_{\checkmark}\ i \in@ [0..<k].\ P_{SKIPS}\langle\!\langle A\ i\rangle\!\rangle_d\ 0\ [\![E]\!]_{\checkmark Llist}\ P_{SKIPS}\langle\!\langle A\ k\rangle\!\rangle_d\ 0$›
using *Suc.hyps*(*1*) **by** (*simp add*: *MultiSync*$_{ptick}$*-snoc*)
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ (\sigma, A) \in@ zip\ (replicate\ k\ 0)\ (map\ A\ [0..<k]).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma\ [\![E]\!]_{\checkmark Llist}\ P_{SKIPS}\langle\!\langle A\ k\rangle\!\rangle_d\ 0$›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ (\sigma, A) \in@ zip\ (replicate\ (Suc\ k)\ 0)\ (map\ A\ [0..<Suc\ k]).\ P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
using *Suc.hyps*(*1*) **by** (*simp flip*: *replicate-append-same add*: *MultiSync*$_{ptick}$*-snoc*)
finally show *?case* **.**
qed *simp-all*
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle_d \bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\!\rangle_d\ (replicate\ n\ 0)$›
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*:
‹$[\![E]\!]_{\checkmark}\ P \in@ map\ Q\ [0..<n].\ P = P\langle\!\langle\langle\!\langle_d \bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\!\rangle_d\ (replicate\ n\ 0)$›
if ‹$\bigwedge i\ j.\ i < n \Longrightarrow j < n \Longrightarrow i \neq j \Longrightarrow indep\text{-}enabl\ (A\ i)\ 0\ E\ (A\ j)\ 0$›
‹$\bigwedge i.\ i < n \Longrightarrow P\langle\!\langle A\ i\rangle\!\rangle_d\ 0 = Q\ i$›
proof −
have ‹$[\![E]\!]_{\checkmark}\ P \in@ map\ Q\ [0..<n].\ P = [\![E]\!]_{\checkmark}\ i \in@ [0..<n].\ Q\ i$›
by (*auto intro*: *mono-MultiSync*$_{ptick}$*-eq2*)
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ i \in@ [0..<n].\ P\langle\!\langle A\ i\rangle\!\rangle_d\ 0$›
by (*auto simp add*: *that intro*: *mono-MultiSync*$_{ptick}$*-eq*)
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ (\sigma, A) \in@ (zip\ (replicate\ n\ 0)\ (map\ A\ [0..<n])).\ P\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
proof (*induct n rule*: *nat-induct-012*)
case (*Suc k*)
have ‹$[\![E]\!]_{\checkmark}\ i \in@ [0..<Suc\ k].\ P\langle\!\langle A\ i\rangle\!\rangle_d\ 0 =$
$[\![E]\!]_{\checkmark}\ i \in@ [0..<k].\ P\langle\!\langle A\ i\rangle\!\rangle_d\ 0\ [\![E]\!]_{\checkmark Llist}\ P\langle\!\langle A\ k\rangle\!\rangle_d\ 0$›
using *Suc.hyps*(*1*) **by** (*simp add*: *MultiSync*$_{ptick}$*-snoc*)
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ (\sigma, A) \in@ zip\ (replicate\ k\ 0)\ (map\ A\ [0..<k]).\ P\langle\!\langle A\rangle\!\rangle_d\ \sigma\ [\![E]\!]_{\checkmark Llist}\ P\langle\!\langle A\ k\rangle\!\rangle_d\ 0$›
by (*simp only*: *Suc.hyps*(*2*))
also have ‹$\ldots = [\![E]\!]_{\checkmark}\ (\sigma, A) \in@ zip\ (replicate\ (Suc\ k)\ 0)\ (map\ A\ [0..<Suc\ k]).\ P\langle\!\langle A\rangle\!\rangle_d\ \sigma$›
using *Suc.hyps*(*1*) **by** (*simp flip*: *replicate-append-same add*: *MultiSync*$_{ptick}$*-snoc*)
finally show *?case* **.**
qed *simp-all*
also have ‹$\ldots = P\langle\!\langle\langle\!\langle_d \bigotimes [\![E]\!]_{\checkmark}\ map\ A\ [0..<n]\rangle\!\rangle\!\rangle_d\ (replicate\ n\ 0)$›

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-P*

Chapter 9

Combining Automata for Sequential Composition Generalized

9.1 Definitions

9.2 General Patterns

definition *combine$_d$-Seq-ε* ::
‹[('σ$_0$, 'a, 'r, 'α$_0$) A_d-*scheme*, 'r ⇒ ('σ$_1$, 'a, 's, 'α$_1$) A_d-*scheme*, 'σ ⇒ 'σ$_0$, 'σ ⇒ 'σ$_1$, 'σ] ⇒ 'a *set*›
where ‹*combine$_d$-Seq-ε* A_0 A_1 i_0 i_1 σs ≡
if i_0 σs ∈ ϱ A_0
then *if* i_1 σs ∈ ϱ (A_1 ⌈ω A_0 (i_0 σs)⌉)
then {}
else ε (A_1 ⌈ω A_0 (i_0 σs)⌉) (i_1 σs)
else ε A_0 (i_0 σs)›

definition *combine$_{nd}$-Seq-ε* ::
‹[('σ$_0$, 'a, 'r, 'α$_0$) A_{nd}-*scheme*, 'r ⇒ ('σ$_1$, 'a, 's, 'α$_1$) A_{nd}-*scheme*, 'σ ⇒ 'σ$_0$, 'σ ⇒ 'σ$_1$, 'σ] ⇒ 'a *set*›
where ‹*combine$_{nd}$-Seq-ε* A_0 A_1 i_0 i_1 σs ≡
if i_0 σs ∈ ϱ A_0
then *if* i_1 σs ∈ (⋃r∈ω A_0 (i_0 σs). ϱ (A_1 r))
then {}
else (⋃r ∈ ω A_0 (i_0 σs). ε (A_1 r) (i_1 σs))
else ε A_0 (i_0 σs)›

lemmas *combine-Seq-ε-defs* = *combine$_d$-Seq-ε-def* *combine$_{nd}$-Seq-ε-def*

fun *combine$_d$-Seq* ::
‹[('σ$_0$, 'e, 'r, 'α$_0$) A_d-*scheme*, 'r ⇒ ('σ$_1$, 'e, 's, 'α$_1$) A_d-*scheme*,

$'\sigma \Rightarrow '\sigma_0,\ '\sigma \Rightarrow '\sigma_1,\ '\sigma_0 \Rightarrow '\sigma_1 \Rightarrow '\sigma] \Rightarrow ('\sigma,\ 'e,\ 's)\ A_d$›
and *combine*$_{nd}$*-Seq* ::
‹$[('\sigma_0,\ 'e,\ 'r,\ '\alpha_0)\ A_{nd}$*-scheme*, $'r \Rightarrow ('\sigma_1,\ 'e,\ 's,\ '\alpha_1)\ A_{nd}$*-scheme*,
$'\sigma \Rightarrow '\sigma_0,\ '\sigma \Rightarrow '\sigma_1,\ '\sigma_0 \Rightarrow '\sigma_1 \Rightarrow '\sigma] \Rightarrow ('\sigma,\ 'e,\ 's)\ A_{nd}$›
where ‹*combine*$_d$*-Seq* $A_0\ A_1\ i_0\ i_1\ f =$
$(\!|\tau = \lambda\sigma s\ e.$ *if* $i_0\ \sigma s \in \varrho\ A_0$
then *if* $i_1\ \sigma s \in \varrho\ (A_1\ \lceil\omega\ A_0\ (i_0\ \sigma s)\rceil)$
then $\Diamond$
else update-right $(A_1\ \lceil\omega\ A_0\ (i_0\ \sigma s)\rceil)\ (i_0\ \sigma s)\ (i_1\ \sigma s)\ e$ (*f-up-opt* f) $(\lambda\sigma.\ \lfloor\sigma\rfloor)$
else update-left $A_0\ (i_0\ \sigma s)\ (i_1\ \sigma s)\ e$ (*f-up-opt* f) $(\lambda\sigma.\ \lfloor\sigma\rfloor)$,
$\omega = \lambda\sigma s.$ *case* $\omega\ A_0\ (i_0\ \sigma s)$ *of* $\Diamond \Rightarrow \Diamond \mid \lfloor r\rfloor \Rightarrow \omega\ (A_1\ r)\ (i_1\ \sigma s)|\!)$›
| ‹*combine*$_{nd}$*-Seq* $A_0\ A_1\ i_0\ i_1\ f =$
$(\!|\tau = \lambda\sigma s\ e.$ *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 $(\bigcup r{\in}\omega\ A_0\ (i_0\ \sigma s).$ *update-right* $(A_1\ r)\ (i_0\ \sigma s)\ (i_1\ \sigma s)\ e$ (*f-up-set* f) $(\lambda\sigma.\ \{\sigma\}))$
else *update-left* $A_0\ (i_0\ \sigma s)\ (i_1\ \sigma s)\ e$ (*f-up-set* f) $(\lambda\sigma.\ \{\sigma\})$,
$\omega = \lambda\sigma s.\ \bigcup r{\in}\omega\ A_0\ (i_0\ \sigma s).\ \omega\ (A_1\ r)\ (i_1\ \sigma s)|\!)$›

9.3 Specializations

definition *combine*$_{dPairlist}$*-Seq*$_{ptick}$ ::
‹$[('\sigma,\ 'e,\ 'r,\ '\alpha)\ A_d$*-scheme*, $'r \Rightarrow ('\sigma,\ 'e,\ 's,\ '\beta)\ A_d$*-scheme*$] \Rightarrow ('\sigma$ *list*, $'e,\ 's)\ A_d$›
where ‹*combine*$_{dPairlist}$*-Seq*$_{ptick}$ $A_0\ A_1 \equiv$ *combine*$_d$*-Seq* $A_0\ A_1$ *hd* $(\lambda\sigma s.$ *hd* (*tl* σs)) $(\lambda s\ t.\ [s,\ t])$›
definition *combine*$_{ndPairlist}$*-Seq*$_{ptick}$ ::
‹$[('\sigma,\ 'e,\ 'r,\ '\alpha)\ A_{nd}$*-scheme*, $'r \Rightarrow ('\sigma,\ 'e,\ 's,\ '\beta)\ A_{nd}$*-scheme*$] \Rightarrow ('\sigma$ *list*, $'e,\ 's)$ A_{nd}›
where ‹*combine*$_{ndPairlist}$*-Seq*$_{ptick}$ $A_0\ A_1 \equiv$ *combine*$_{nd}$*-Seq* $A_0\ A_1$ *hd* $(\lambda\sigma s.$ *hd* (*tl* σs)) $(\lambda s\ t.\ [s,\ t])$›

definition *combine*$_{dPair}$*-Seq*$_{ptick}$ ::
‹$[('\sigma_0,\ 'e,\ 'r,\ '\alpha)\ A_d$*-scheme*, $'r \Rightarrow ('\sigma_1,\ 'e,\ 's,\ '\beta)\ A_d$*-scheme*$] \Rightarrow ('\sigma_0 \times '\sigma_1,\ 'e,$ $'s)\ A_d$›
where ‹*combine*$_{dPair}$*-Seq*$_{ptick}$ $A_0\ A_1 \equiv$ *combine*$_d$*-Seq* $A_0\ A_1$ *fst snd Pair*›
definition *combine*$_{ndPair}$*-Seq*$_{ptick}$ ::
‹$[('\sigma_0,\ 'e,\ 'r,\ '\alpha)\ A_{nd}$*-scheme*, $'r \Rightarrow ('\sigma_1,\ 'e,\ 's,\ '\beta)\ A_{nd}$*-scheme*$] \Rightarrow ('\sigma_0 \times '\sigma_1,$ $'e,\ 's)\ A_{nd}$›
where ‹*combine*$_{ndPair}$*-Seq*$_{ptick}$ $A_0\ A_1 \equiv$ *combine*$_{nd}$*-Seq* $A_0\ A_1$ *fst snd Pair*›

definition *combine*$_{dListslenL}$*-Seq*$_{ptick}$ ::
‹$[('\sigma$ *list*, $'e,\ 'r,\ '\alpha)\ A_d$*-scheme*, *nat*, $'r \Rightarrow ('\sigma$ *list*, $'e,\ 's,\ '\beta)\ A_d$*-scheme*$] \Rightarrow ('\sigma$ *list*, $'e,\ 's)\ A_d$›
where ‹*combine*$_{dListslenL}$*-Seq*$_{ptick}$ A_0 *len*$_0$ $A_1 \equiv$ *combine*$_d$*-Seq* $A_0\ A_1$ (*take* *len*$_0$) (*drop* *len*$_0$) (@)›
definition *combine*$_{ndListslenL}$*-Seq*$_{ptick}$ ::
‹$[('\sigma$ *list*, $'e,\ 'r,\ '\alpha)\ A_{nd}$*-scheme*, *nat*, $'r \Rightarrow ('\sigma$ *list*, $'e,\ 's,\ '\beta)\ A_{nd}$*-scheme*$] \Rightarrow ('\sigma$

list, *'e*, *'s*) A_{nd}›
where ‹$combine_{ndListslenL}$-Seq_{ptick} A_0 len_0 A_1 ≡ $combine_{nd}$-*Seq* A_0 A_1 (*take* len_0) (*drop* len_0) (@)›

definition $combine_{dRlist}$-Seq_{ptick} ::
‹[(*'σ*, *'e*, *'r*, *'α*) A_d-*scheme*, *'r* ⇒ (*'σ list*, *'e*, *'s*, *'β*) A_d-*scheme*] ⇒ (*'σ list*, *'e*, *'s*) A_d›
where ‹$combine_{dRlist}$-Seq_{ptick} A_0 A_1 ≡ $combine_d$-*Seq* A_0 A_1 *hd tl* (#)›
definition $combine_{ndRlist}$-Seq_{ptick} ::
‹[(*'σ*, *'e*, *'r*, *'α*) A_{nd}-*scheme*, *'r* ⇒ (*'σ list*, *'e*, *'s*, *'β*) A_{nd}-*scheme*] ⇒ (*'σ list*, *'e*, *'s*) A_{nd}›
where ‹$combine_{ndRlist}$-Seq_{ptick} A_0 A_1 ≡ $combine_{nd}$-*Seq* A_0 A_1 *hd tl* (#)›

lemmas $combine_{Pairlist}$-Seq_{ptick}-*defs* = $combine_{dPairlist}$-Seq_{ptick}-*def* $combine_{ndPairlist}$-Seq_{ptick}-*def*
and $combine_{Pair}$-Seq_{ptick}-*defs* = $combine_{dPair}$-Seq_{ptick}-*def* $combine_{ndPair}$-Seq_{ptick}-*def*
and $combine_{ListslenL}$-Seq_{ptick} = $combine_{dListslenL}$-Seq_{ptick}-*def* $combine_{ndListslenL}$-Seq_{ptick}-*def*
and $combine_{Rlist}$-Seq_{ptick}-*defs* = $combine_{dRlist}$-Seq_{ptick}-*def* $combine_{ndRlist}$-Seq_{ptick}-*def*

lemmas *combine-Seq-defs* =
$combine_{Pairlist}$-Seq_{ptick}-*defs* $combine_{Pair}$-Seq_{ptick}-*defs* $combine_{ListslenL}$-Seq_{ptick} $combine_{Rlist}$-Seq_{ptick}-*defs*

bundle *combine-Seq-syntax* **begin**

notation $combine_{dPairlist}$-Seq_{ptick} (‹⟪- $_d$⊗;✓$_{Pairlist}$ -⟫› [*0*, *0*])
notation $combine_{ndPairlist}$-Seq_{ptick} (‹⟪- $_{nd}$⊗;✓$_{Pairlist}$ -⟫› [*0*, *0*])
notation $combine_{dPair}$-Seq_{ptick} (‹⟪- $_d$⊗;✓$_{Pair}$ -⟫› [*0*, *0*])
notation $combine_{ndPair}$-Seq_{ptick} (‹⟪- $_{nd}$⊗;✓$_{Pair}$ -⟫› [*0*, *0*])
notation $combine_{dListslenL}$-Seq_{ptick} (‹⟪- $_d$⊗-;$_{ListslenL}$ -⟫› [*0*, *0*, *0*])
notation $combine_{ndListslenL}$-Seq_{ptick} (‹⟪- $_{nd}$⊗-;$_{ListslenL}$ -⟫› [*0*, *0*, *0*])
notation $combine_{dRlist}$-Seq_{ptick} (‹⟪- $_d$⊗;✓$_{Rlist}$ -⟫› [*0*, *0*])
notation $combine_{ndRlist}$-Seq_{ptick} (‹⟪- $_{nd}$⊗;✓$_{Rlist}$ -⟫› [*0*, *0*])

end

unbundle *combine-Seq-syntax*

9.4 First Properties

lemma *ε*-$combine_{Pairlist}$-Seq_{ptick} :
‹*ε* ⟪A_0 $_d$⊗;✓$_{Pairlist}$ A_1⟫ *σs* = $combine_d$-*Seq*-*ε* A_0 A_1 *hd* (*hd* ∘ *tl*) *σs*›
‹*ε* ⟪B_0 $_{nd}$⊗;✓$_{Pairlist}$ B_1⟫ *σs* = $combine_{nd}$-*Seq*-*ε* B_0 B_1 *hd* (*hd* ∘ *tl*) *σs*›
by (*auto simp add*: *combine-Seq-ε-defs* $combine_{Pairlist}$-Seq_{ptick}-*defs* *ε-simps* *ϱ-simps* *option.case-eq-if*)

lemma *ε*-$combine_{Pair}$-Seq_{ptick} :

‹ε ⦃A_0 $_d\otimes;_{\checkmark Pair}$ A_1⦄ σs = *combine*$_d$*-Seq-*ε A_0 A_1 *fst snd* σs›
‹ε ⦃B_0 $_{nd}\otimes;_{\checkmark Pair}$ B_1⦄ σs = *combine*$_{nd}$*-Seq-*ε B_0 B_1 *fst snd* σs›
by (*auto simp add*: *combine-Seq-*ε*-defs combine*$_{Pair}$*-Seq*$_{ptick}$*-defs*
ε*-simps* ϱ*-simps option.case-eq-if*)

lemma ε*-combine*$_{ListslenL}$*-Seq*$_{ptick}$:
‹ε ⦃A_0 $_d\otimes len_0;_{ListslenL}$ A_1⦄ σs = *combine*$_d$*-Seq-*ε A_0 A_1 (*take* len_0) (*drop* len_0) σs›
‹ε ⦃B_0 $_{nd}\otimes len_0;_{ListslenL}$ B_1⦄ σs = *combine*$_{nd}$*-Seq-*ε B_0 B_1 (*take* len_0) (*drop* len_0) σs›
by (*auto simp add*: *combine-Seq-*ε*-defs combine*$_{ListslenL}$*-Seq*$_{ptick}$
ε*-simps* ϱ*-simps option.case-eq-if*)

lemma ε*-combine*$_{Rlist}$*-Seq*$_{ptick}$:
‹ε ⦃A_0 $_d\otimes;_{\checkmark Rlist}$ A_1⦄ σs = *combine*$_d$*-Seq-*ε A_0 A_1 *hd tl* σs›
‹ε ⦃B_0 $_{nd}\otimes;_{\checkmark Rlist}$ B_1⦄ σs = *combine*$_{nd}$*-Seq-*ε B_0 B_1 *hd tl* σs›
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*$_{dListslenL}$*-Seq*$_{ptick}$*-subset*:
‹$\mathcal{R}_d$ ⦃A_0 $_d\otimes len_0;_{ListslenL}$ A_1⦄ (s_0 @ s_1) $\subseteq$ {t_0 @ t_1 |t_0 t_1. $t_0 \in \mathcal{R}_d$ A_0 s_0}› (**is** ‹$?S_A \subseteq$ -›)
if *same-length-*$\mathcal{R}_d$: ‹$\bigwedge t_0$. $t_0 \in \mathcal{R}_d$ A_0 $s_0 \Longrightarrow$ *length* t_0 = len_0›
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = t_0$ @ $t_1 \wedge t_0 \in \mathcal{R}_d$ A_0 s_0› **for** t
proof (*induct rule*: $\mathcal{R}_d$*.induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_d$*.init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) *same-length-*$\mathcal{R}_d$ **obtain** t_0 t_1
where ‹σ' = t_0 @ t_1› ‹$t_0 \in \mathcal{R}_d$ A_0 s_0› ‹*length* t_0 = len_0› **by** *blast*
with *step.hyps*(*3*) **show** *?case*
by (*auto simp add*: *combine*$_{ListslenL}$*-Seq*$_{ptick}$ *map-option-case*
split: *if-split-asm option.splits*) (*metis* $\mathcal{R}_d$*.step*)
qed
qed

lemma $\mathcal{R}_{nd}$*-combine*$_{ndListslenL}$*-Seq*$_{ptick}$*-subset*:
‹$\mathcal{R}_{nd}$ ⦃A_0 $_{nd}\otimes len_0;_{ListslenL}$ A_1⦄ (s_0 @ s_1) $\subseteq$ {t_0 @ t_1 |t_0 t_1. $t_0 \in \mathcal{R}_{nd}$ A_0 s_0}› (**is** ‹$?S_A \subseteq$ -›)
if *same-length-*$\mathcal{R}_{nd}$: ‹$\bigwedge t_0$. $t_0 \in \mathcal{R}_{nd}$ A_0 $s_0 \Longrightarrow$ *length* t_0 = len_0›
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = t_0$ @ $t_1 \wedge t_0 \in \mathcal{R}_{nd}$ A_0 s_0› **for** t
proof (*induct rule*: $\mathcal{R}_{nd}$*.induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_{nd}$*.init*)
next
case (*step* σ' σ'' *a*)

from *step.hyps*(*2*) *same-length-*$\mathcal{R}_{nd}$ **obtain** t_0 t_1
where ‹$\sigma' = t_0$ @ t_1› ‹$t_0 \in \mathcal{R}_{nd}$ A_0 s_0› ‹*length* t_0 = len_0› **by** *blast*
with *step.hyps*(*3*) **show** *?case*
by (*auto simp add*: $combine_{ListslenL}$-Seq_{ptick} *split*: *if-split-asm*) (*metis* $\mathcal{R}_{nd}$.*step*)
qed
qed

lemma $\mathcal{R}_d$-$combine_{dPairlist}$-Seq_{ptick}-*subset*:
‹$\mathcal{R}_d$ ⟪A_0 $_d\otimes$;✓$_{Pairlist}$ A_1⟫ $[s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0\}$› (**is** ‹$?S_A \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0$ › **for** t
proof (*induct rule*: $\mathcal{R}_d$.*induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_d$.*init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) **obtain** t_0 t_1 **where** ‹$\sigma' = [t_0, t_1]$› ‹$t_0 \in \mathcal{R}_d$ A_0 s_0› **by** *blast*
with *step.hyps*(*3*) **show** *?case*
by (*auto simp add*: $combine_{Pairlist}$-Seq_{ptick}-*defs map-option-case*
split: *if-split-asm option.split-asm*) (*metis* $\mathcal{R}_d$.*step*)
qed
qed

lemma $\mathcal{R}_{nd}$-$combine_{ndPairlist}$-Seq_{ptick}-*subset*:
‹$\mathcal{R}_{nd}$ ⟪A_0 $_{nd}\otimes$;✓$_{Pairlist}$ A_1⟫ $[s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ A_0\ s_0\}$› (**is** ‹$?S_A \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = [t_0, t_1] \wedge t_0 \in \mathcal{R}_{nd}\ A_0\ s_0$ › **for** t
proof (*induct rule*: $\mathcal{R}_{nd}$.*induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_{nd}$.*init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) **obtain** t_0 t_1 **where** ‹$\sigma' = [t_0, t_1]$› ‹$t_0 \in \mathcal{R}_{nd}$ A_0 s_0› **by** *blast*
with *step.hyps*(*3*) **show** *?case*
by (*auto simp add*: $combine_{Pairlist}$-Seq_{ptick}-*defs split*: *if-split-asm*) (*metis* $\mathcal{R}_{nd}$.*step*)
qed
qed

lemma $\mathcal{R}_d$-$combine_{dPair}$-Seq_{ptick}-*subset*:
‹$\mathcal{R}_d$ ⟪A_0 $_d\otimes$;✓$_{Pair}$ A_1⟫ $(s_0, s_1) \subseteq \{(t_0, t_1) \mid t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0\}$› (**is** ‹$?S_A \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = (t_0, t_1) \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0$ › **for** t
proof (*induct rule*: $\mathcal{R}_d$.*induct*)

case *init* **show** *?case* **by** (*metis* $\mathcal{R}_d$*.init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) **obtain** t_0 t_1 **where** ‹$\sigma' = (t_0, t_1)$› ‹$t_0 \in \mathcal{R}_d\ A_0\ s_0$› **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*:
‹$\mathcal{R}_{nd}$ ⟪A_0 $_{nd}\otimes;_{\checkmark Pair}$ A_1⟫ $(s_0, s_1) \subseteq \{(t_0, t_1) \mid t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ A_0\ s_0\}$› (**is** ‹$?S_A \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = (t_0, t_1) \wedge t_0 \in \mathcal{R}_{nd}\ A_0\ s_0$ › **for** *t*
proof (*induct rule*: $\mathcal{R}_{nd}$*.induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_{nd}$*.init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) **obtain** t_0 t_1 **where** ‹$\sigma' = (t_0, t_1)$› ‹$t_0 \in \mathcal{R}_{nd}\ A_0\ s_0$› **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*:
‹$\mathcal{R}_d$ ⟪A_0 $_d\otimes;_{\checkmark Rlist}$ A_1⟫ $(s_0 \# s_1) \subseteq \{t_0 \# t_1 \mid t_0\ t_1.\ t_0 \in \mathcal{R}_d\ A_0\ s_0\}$› (**is** ‹$?S_A \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = t_0 \# t_1 \wedge t_0 \in \mathcal{R}_d\ A_0\ s_0$ › **for** *t*
proof (*induct rule*: $\mathcal{R}_d$*.induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_d$*.init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) **obtain** t_0 t_1 **where** ‹$\sigma' = t_0 \# t_1$› ‹$t_0 \in \mathcal{R}_d\ A_0\ s_0$› **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*:
‹$\mathcal{R}_{nd}$ ⟪A_0 $_{nd}\otimes;_{\checkmark Rlist}$ A_1⟫ $(s_0 \# s_1) \subseteq \{t_0 \# t_1 \mid t_0\ t_1.\ t_0 \in \mathcal{R}_{nd}\ A_0\ s_0\}$› (**is** ‹$?S_A \subseteq$ -›)
proof *safe*
show ‹$t \in ?S_A \Longrightarrow \exists\, t_0\ t_1.\ t = t_0 \# t_1 \wedge t_0 \in \mathcal{R}_{nd}\ A_0\ s_0$ › **for** *t*

proof (*induct rule*: $\mathcal{R}_{nd}$.*induct*)
case *init* **show** *?case* **by** (*metis* $\mathcal{R}_{nd}$.*init*)
next
case (*step* σ' σ'' *a*)
from *step.hyps*(*2*) **obtain** t_0 t_1 **where** ‹$\sigma' = t_0$ # t_1› ‹$t_0 \in \mathcal{R}_{nd}$ A_0 s_0› **by** *blast*
with *step.hyps*(*3*) **show** *?case*
by (*auto simp add*: *combine*$_{Rlist}$-*Seq*$_{ptick}$-*defs split*: *if-split-asm*) (*metis* $\mathcal{R}_{nd}$.*step*)
qed
qed

9.5 Normalization

lemma τ-*combine*$_{Pairlist}$-*Seq*$_{ptick}$-*behaviour*:
‹τ ⟪⟪A_0 $_{d}\otimes$;✓$_{Pairlist}$ A_1⟫⟫$_{d\hookrightarrow nd}$ [s_0, s_1] e = τ ⟪⟪A_0⟫$_{d\hookrightarrow nd}$ $_{nd}\otimes$;✓$_{Pairlist}$ (λr. ⟪A_1 r⟫$_{d\hookrightarrow nd}$)⟫ [s_0, s_1] e›
by (*auto simp add*: *combine*$_{Pairlist}$-*Seq*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps* ϱ-*simps*)

lemma τ-*combine*$_{Pair}$-*Seq*$_{ptick}$-*behaviour*:
‹τ ⟪⟪A_0 $_{d}\otimes$;✓$_{Pair}$ A_1⟫⟫$_{d\hookrightarrow nd}$ (s_0, s_1) e = τ ⟪⟪A_0⟫$_{d\hookrightarrow nd}$ $_{nd}\otimes$;✓$_{Pair}$ (λr. ⟪A_1 r⟫$_{d\hookrightarrow nd}$)⟫ (s_0, s_1) e›
by (*auto simp add*: *combine*$_{Pair}$-*Seq*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps* ϱ-*simps*)

lemma τ-*combine*$_{ListslenL}$-*Seq*$_{ptick}$-*behaviour*:
‹τ ⟪⟪A_0 $_{d}\otimes len_0$;$_{ListslenL}$ A_1⟫⟫$_{d\hookrightarrow nd}$ (σs_0 @ σs_1) e = τ ⟪⟪A_0⟫$_{d\hookrightarrow nd}$ $_{nd}\otimes len_0$;$_{ListslenL}$ (λr. ⟪A_1 r⟫$_{d\hookrightarrow nd}$)⟫ (σs_0 @ σs_1) e›
by (*auto simp add*: *combine*$_{ListslenL}$-*Seq*$_{ptick}$ *det-ndet-conv-defs option.case-eq-if* ε-*simps* ϱ-*simps*)

lemma τ-*combine*$_{Rlist}$-*Seq*$_{ptick}$-*behaviour*:
‹τ ⟪⟪A_0 $_{d}\otimes$;✓$_{Rlist}$ A_1⟫⟫$_{d\hookrightarrow nd}$ (s_0 # σs_1) e = τ ⟪⟪A_0⟫$_{d\hookrightarrow nd}$ $_{nd}\otimes$;✓$_{Rlist}$ (λr. ⟪A_1 r⟫$_{d\hookrightarrow nd}$)⟫ (s_0 # σs_1) e›
by (*auto simp add*: *combine*$_{Rlist}$-*Seq*$_{ptick}$-*defs det-ndet-conv-defs option.case-eq-if* ε-*simps* ϱ-*simps*)

Behaviour of normalisations

lemma P_{SKIPS}-*combine*$_{Pairlist}$-*Seq*$_{ptick}$-*behaviour*:
‹P_{SKIPS}⟪⟪A_0 $_{d}\otimes$;✓$_{Pairlist}$ A_1⟫⟫$_d$ [s_0, s_1] = P_{SKIPS}⟪⟪⟪A_0⟫$_{d\hookrightarrow nd}$ $_{nd}\otimes$;✓$_{Pairlist}$ (λr. ⟪A_1 r⟫$_{d\hookrightarrow nd}$)⟫⟫$_{nd}$ [s_0, s_1]›
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* ‹*drule set-mp*[*OF* $\mathcal{R}_d$-*combine*$_{dPairlist}$-*Seq*$_{ptick}$-*subset*]›,
auto simp add: *combine*$_{Pairlist}$-*Seq*$_{ptick}$-*defs from-det-to-ndet-def* ϱ-*simps split*: *option.split*)

lemma *P-combine*$_{Pairlist}$-*Seq*$_{ptick}$-*behaviour*:

‹$P\langle\langle A_0\ {}_d\otimes;_{\checkmark Pairlist}\ A_1\rangle\rangle_d\ [s_0,\ s_1] = P\langle\langle\langle\langle A_0\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Pairlist}\ (\lambda r.\ \langle\langle A_1\ r\rangle_{d\hookrightarrow nd}\rangle)\rangle\rangle_{nd}\ [s_0,\ s_1]$›
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*$_{dPairlist}$*-Seq*$_{ptick}$*-subset*],
auto simp add: *combine*$_{Pairlist}$*-Seq*$_{ptick}$*-defs from-det-to-ndet-def* ϱ*-simps split*: *option.split*)

lemma P_{SKIPS}*-combine*$_{Pair}$*-Seq*$_{ptick}$*-behaviour*:
‹$P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes;_{\checkmark Pair}\ A_1\rangle\rangle_d\ (s_0,\ s_1) = P_{SKIPS}\langle\langle\langle\langle A_0\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Pair}\ (\lambda r.\ \langle\langle A_1\ r\rangle_{d\hookrightarrow nd}\rangle)\rangle\rangle_{nd}\ (s_0,\ s_1)$›
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* ‹*drule set-mp*[*OF* $\mathcal{R}_d$*-combine*$_{dPair}$*-Seq*$_{ptick}$*-subset*]›,
auto simp add: *combine*$_{Pair}$*-Seq*$_{ptick}$*-defs from-det-to-ndet-def* ϱ*-simps split*: *option.split*)

lemma *P-combine*$_{Pair}$*-Seq*$_{ptick}$*-behaviour*:
‹$P\langle\langle\langle\langle A_0\ {}_d\otimes;_{\checkmark Pair}\ A_1\rangle\rangle_d\ (s_0,\ s_1) = P\langle\langle\langle\langle A_0\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Pair}\ (\lambda r.\ \langle\langle A_1\ r\rangle_{d\hookrightarrow nd}\rangle)\rangle\rangle_{nd}\ (s_0,\ s_1)$›
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*$_{dPair}$*-Seq*$_{ptick}$*-subset*],
auto simp add: *combine*$_{Pair}$*-Seq*$_{ptick}$*-defs from-det-to-ndet-def* ϱ*-simps split*: *option.split*)

lemma P_{SKIPS}*-combine*$_{Rlist}$*-Seq*$_{ptick}$*-behaviour*:
‹$P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes;_{\checkmark Rlist}\ A_1\rangle\rangle_d\ (s_0\ \#\ s_1) = P_{SKIPS}\langle\langle\langle\langle A_0\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Rlist}\ (\lambda r.\ \langle\langle A_1\ r\rangle_{d\hookrightarrow nd}\rangle)\rangle\rangle_{nd}\ (s_0\ \#\ s_1)$›
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* ‹*drule set-mp*[*OF* $\mathcal{R}_d$*-combine*$_{dRlist}$*-Seq*$_{ptick}$*-subset*]›,
auto simp add: *combine*$_{Rlist}$*-Seq*$_{ptick}$*-defs from-det-to-ndet-def* ϱ*-simps split*: *option.split*)

lemma *P-combine*$_{Rlist}$*-Seq*$_{ptick}$*-behaviour*:
‹$P\langle\langle\langle\langle A_0\ {}_d\otimes;_{\checkmark Rlist}\ A_1\rangle\rangle_d\ (s_0\ \#\ s_1) = P\langle\langle\langle\langle A_0\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Rlist}\ (\lambda r.\ \langle\langle A_1\ r\rangle_{d\hookrightarrow nd}\rangle)\rangle\rangle_{nd}\ (s_0\ \#\ s_1)$›
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*$_{dRlist}$*-Seq*$_{ptick}$*-subset*],
auto simp add: *combine*$_{Rlist}$*-Seq*$_{ptick}$*-defs from-det-to-ndet-def* ϱ*-simps split*: *option.split*)

lemma P_{SKIPS}*-combine*$_{ListslenL}$*-Seq*$_{ptick}$*-behaviour*:
‹$P_{SKIPS}\langle\langle\langle\langle A_0\ {}_d\otimes len_0;_{ListslenL}\ A_1\rangle\rangle_d\ (\sigma s_0\ @\ \sigma s_1) = P_{SKIPS}\langle\langle\langle\langle A_0\rangle_{d\hookrightarrow nd}$

$_{nd}\otimes len_0;_{ListslenL}$ $(\lambda r.\ \langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd})\rangle\!\rangle_{nd}$ $(\sigma s_0$ @ $\sigma s_1)$›
if ‹$\bigwedge\sigma s_0{}'.\ \sigma s_0{}' \in \mathcal{R}_d\ A_0\ \sigma s_0 \Longrightarrow$ *length* $\sigma s_0{}' = len_0$›
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* ‹*drule set-mp*[*OF* $\mathcal{R}_d$*-combine*$_{dListslenL}$*-Seq*$_{ptick}$*-subset*[*OF that*], *rotated*]›,
auto simp add: *combine*$_{ListslenL}$*-Seq*$_{ptick}$ *from-det-to-ndet-def* ϱ*-simps split*: *option.split*)

lemma *P-combine*$_{ListslenL}$*-Seq*$_{ptick}$*-behaviour*:
‹$P\langle\!\langle\langle\!\langle A_0\ _d\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle_d\ (\sigma s_0$ @ $\sigma s_1) = P\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ _{nd}\otimes len_0;_{ListslenL}$ $(\lambda r.\ \langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd})\rangle\!\rangle_{nd}$ $(\sigma s_0$ @ $\sigma s_1)$›
if ‹$\bigwedge\sigma t_0.\ \sigma t_0 \in \mathcal{R}_d\ A_0\ \sigma s_0 \Longrightarrow$ *length* $\sigma t_0 = len_0$›
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*$_{dListslenL}$*-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*$_{ptick}$ :: ‹$[('r \Rightarrow ('\sigma, 'e, 'r)\ A_d)\ list, 'r] \Rightarrow ('\sigma\ list, 'e, 'r)\ A_d$› (‹$\langle\!\langle_d\bigotimes;_{\checkmark}$ -$\rangle\!\rangle$› [*0*])
  where ‹$\langle\!\langle_d\bigotimes;_{\checkmark}\ []\rangle\!\rangle\ r = (\!|\tau = \lambda\sigma s\ a.\ \Diamond, \omega = \lambda\sigma s.\ \lfloor r\rfloor|\!)$›
  | ‹$\langle\!\langle_d\bigotimes;_{\checkmark}\ [A_0]\rangle\!\rangle\ r = {}_d\langle\!\langle A_0\ r\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›
  | ‹$\langle\!\langle_d\bigotimes;_{\checkmark}\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ r = \langle\!\langle A_0\ r\ {}_d\otimes;_{\checkmark Rlist}\ \langle\!\langle_d\bigotimes;_{\checkmark}\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle$›

fun *iterated-combine*$_{nd}$*-Seq*$_{ptick}$ :: ‹$[('r \Rightarrow ('\sigma, 'e, 'r)\ A_{nd})\ list, 'r] \Rightarrow ('\sigma\ list, 'e, 'r)\ A_{nd}$› (‹$\langle\!\langle_{nd}\bigotimes;_{\checkmark}$ -$\rangle\!\rangle$› [*0*])
  where ‹$\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ []\rangle\!\rangle\ r = (\!|\tau = \lambda\sigma s\ a.\ \{\}, \omega = \lambda\sigma s.\ \{r\}|\!)$›
  | ‹$\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ [A_0]\rangle\!\rangle\ r = {}_{nd}\langle\!\langle A_0\ r\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}$›
  | ‹$\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ A_0\ \#\ A_1\ \#\ As\rangle\!\rangle\ r = \langle\!\langle A_0\ r\ {}_{nd}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{nd}\bigotimes;_{\checkmark}\ A_1\ \#\ As\rangle\!\rangle\rangle\!\rangle$›

lemma *iterated-combine*$_d$*-Seq*$_{ptick}$*-simps-bis*: ‹$As \neq [] \Longrightarrow \langle\!\langle_d\bigotimes;_{\checkmark}\ A_0\ \#\ As\rangle\!\rangle\ r = \langle\!\langle A_0\ r\ {}_d\otimes;_{\checkmark Rlist}\ \langle\!\langle_d\bigotimes;_{\checkmark}\ As\rangle\!\rangle\rangle\!\rangle$›
  and *iterated-combine*$_{nd}$*-Seq*$_{ptick}$*-simps-bis*: ‹$Bs \neq [] \Longrightarrow \langle\!\langle_{nd}\bigotimes;_{\checkmark}\ B_0\ \#\ Bs\rangle\!\rangle\ r = \langle\!\langle B_0\ r\ {}_{nd}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{nd}\bigotimes;_{\checkmark}\ Bs\rangle\!\rangle\rangle\!\rangle$›
  by (*induct As*, *simp-all*) (*induct Bs*, *simp-all*)

10.1.2 First Results

lemma *combine*$_{ListslenL}$*-Seq*$_{ptick}$*-combine*$_{Rlist}$*-Seq*$_{ptick}$*-eq*:
  ‹$\varepsilon\ \langle\!\langle_d\langle\!\langle A_0\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\ {}_d\otimes 1;_{ListslenL}\ A_1\rangle\!\rangle\ (s_0\ \#\ \sigma s) = \varepsilon\ \langle\!\langle A_0\ {}_d\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\ (s_0\ \#\ \sigma s)$›
  ‹$\tau\ \langle\!\langle_d\langle\!\langle A_0\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\ {}_d\otimes 1;_{ListslenL}\ A_1\rangle\!\rangle\ (s_0\ \#\ \sigma s)\ e = \tau\ \langle\!\langle A_0\ {}_d\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\ (s_0\ \#\ \sigma s)\ e$›
  ‹$\varepsilon\ \langle\!\langle_{nd}\langle\!\langle B_0\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\ {}_{nd}\otimes 1;_{ListslenL}\ B_1\rangle\!\rangle\ (s_0\ \#\ \sigma s) = \varepsilon\ \langle\!\langle B_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ B_1\rangle\!\rangle\ (s_0\ \#$

$\sigma s)$›
‹τ $\langle\!\langle_{nd}\langle\!\langle B_0\rangle\!\rangle_{\sigma\hookrightarrow\sigma s}\ {}_{nd}\otimes 1;_{ListslenL}\ B_1\rangle\!\rangle$ $(s_0\ \#\ \sigma s)$ $e = \tau$ $\langle\!\langle B_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ B_1\rangle\!\rangle$ $(s_0\ \#\ \sigma s)$ e›
by (*simp-all add*: *ε-simps ϱ-simps* $combine_{ListslenL}$-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*:
‹ε $\langle\!\langle A_0\ r\ {}_{d}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle$ $[s_0,\ s_1] = \varepsilon$ $(\langle\!\langle_{d}\bigotimes;_{\checkmark}\ [A_0,\ A_1]\rangle\!\rangle\ r)$ $[s_0,\ s_1]$›
‹τ $\langle\!\langle A_0\ r\ {}_{d}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle$ $[s_0,\ s_1]$ $e = \tau$ $(\langle\!\langle_{d}\bigotimes;_{\checkmark}\ [A_0,\ A_1]\rangle\!\rangle\ r)$ $[s_0,\ s_1]$ e›
‹ε $\langle\!\langle B_0\ r\ {}_{nd}\otimes;_{\checkmark Pairlist}\ B_1\rangle\!\rangle$ $[s_0,\ s_1] = \varepsilon$ $(\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ [B_0,\ B_1]\rangle\!\rangle\ r)$ $[s_0,\ s_1]$›
‹τ $\langle\!\langle B_0\ r\ {}_{nd}\otimes;_{\checkmark Pairlist}\ B_1\rangle\!\rangle$ $[s_0,\ s_1]$ $e = \tau$ $(\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ [B_0,\ B_1]\rangle\!\rangle\ r)$ $[s_0,\ s_1]$ e›
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* :
‹ε $\langle\!\langle A_0\ {}_{d}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle$ $[s_0,\ s_1] = \varepsilon$ $\langle\!\langle A_0\ {}_{d}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{d}\bigotimes;_{\checkmark}\ [A_1]\rangle\!\rangle\rangle\!\rangle$ $[s_0,\ s_1]$›
‹τ $\langle\!\langle A_0\ {}_{d}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle$ $[s_0,\ s_1]$ $e = \tau$ $\langle\!\langle A_0\ {}_{d}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{d}\bigotimes;_{\checkmark}\ [A_1]\rangle\!\rangle\rangle\!\rangle$ $[s_0,\ s_1]$ e›
‹ε $\langle\!\langle B_0\ {}_{nd}\otimes;_{\checkmark Pairlist}\ B_1\rangle\!\rangle$ $[s_0,\ s_1] = \varepsilon$ $\langle\!\langle B_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{nd}\bigotimes;_{\checkmark}\ [B_1]\rangle\!\rangle\rangle\!\rangle$ $[s_0,\ s_1]$›
‹τ $\langle\!\langle B_0\ {}_{nd}\otimes;_{\checkmark Pairlist}\ B_1\rangle\!\rangle$ $[s_0,\ s_1]$ $e = \tau$ $\langle\!\langle B_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{nd}\bigotimes;_{\checkmark}\ [B_1]\rangle\!\rangle\rangle\!\rangle$ $[s_0,\ s_1]$ e›
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} :
‹*length* σs = *length As* $\Longrightarrow$ $\lfloor\sigma t\rfloor = \tau$ $(\langle\!\langle_{d}\bigotimes;_{\checkmark}\ As\rangle\!\rangle\ r)$ σs a $\Longrightarrow$ *length* σt = *length As*›
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* (*metis length-1-trans*$_d$*E length-1-trans-from-σ-to-σs*(*1*) *length-Cons list.size*(*3*))
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 *As*)
from *Cons.prems 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} :
‹*length* σs = *length As* $\Longrightarrow$ $\sigma t \in \tau$ $(\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ As\rangle\!\rangle\ r)$ σs a $\Longrightarrow$ *length* σt = *length As*›

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.prems 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*$_{ptick}$:
‹$\sigma t \in \mathcal{R}_d$ ($\langle\!\langle_d \bigotimes ;_{\checkmark}$ *As*$\rangle\!\rangle$ *r*) $\sigma s \Longrightarrow$ *length* σs = *length As* $\Longrightarrow$ *length* σt = *length As*›
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*$_{ptick}$:
‹$\sigma t \in \mathcal{R}_{nd}$ ($\langle\!\langle_{nd} \bigotimes ;_{\checkmark}$ *As*$\rangle\!\rangle$ *r*) $\sigma s \Longrightarrow$ *length* σs = *length As* $\Longrightarrow$ *length* σt = *length As*›
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*$_{ptick}$:
‹*As* $\neq$ [] $\Longrightarrow$ ϱ*-disjoint-*ε ((*last As*) *r*) $\Longrightarrow$ ϱ*-disjoint-*ε ($\langle\!\langle_d \bigotimes ;_{\checkmark}$ *As*$\rangle\!\rangle$ *r*)›
‹*Bs* $\neq$ [] $\Longrightarrow$ ϱ*-disjoint-*ε ((*last Bs*) *r*) $\Longrightarrow$ ϱ*-disjoint-*ε ($\langle\!\langle_{nd} \bigotimes ;_{\checkmark}$ *Bs*$\rangle\!\rangle$ *r*)›
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*:
‹ω ($\langle\!\langle_{nd} \bigotimes ;_{\checkmark}$ *map* (λA *r*. $\langle\!\langle A\ r\rangle\!\rangle_{d\hookrightarrow nd}$) *As*$\rangle\!\rangle$ *r*) σs = ω $\langle\!\langle\langle\!\langle_d \bigotimes ;_{\checkmark}$ *As*$\rangle\!\rangle$ *r*$\rangle\!\rangle_{d\hookrightarrow nd}$ σs›
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* :
‹ϱ ($\langle\!\langle_{nd} \bigotimes ;_{\checkmark}$ *map* (λA *r*. $\langle\!\langle A\ r\rangle\!\rangle_{d\hookrightarrow nd}$) *As*$\rangle\!\rangle$ *r*) = ϱ $\langle\!\langle\langle\!\langle_d \bigotimes ;_{\checkmark}$ *As*$\rangle\!\rangle$ *r*$\rangle\!\rangle_{d\hookrightarrow nd}$›
by (*simp add*: ϱ*-simps* ω*-iterated-combine-Seq*$_{ptick}$*-det-ndet-conv*)

lemma τ*-iterated-combine-Seq*$_{ptick}$*-behaviour*:
‹*length* σs = *length As* $\Longrightarrow$
τ $\langle\!\langle\langle\!\langle_d \bigotimes ;_{\checkmark}$ *As*$\rangle\!\rangle$ *r*$\rangle\!\rangle_{d\hookrightarrow nd}$ σs *e* = τ ($\langle\!\langle_{nd} \bigotimes ;_{\checkmark}$ *map* (λA *r*. $\langle\!\langle A\ r\rangle\!\rangle_{d\hookrightarrow nd}$) *As*$\rangle\!\rangle$ *r*) σs *e*›

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*-*σs*-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-*ϱ(*1*)
list.simps(*9*))
qed

lemma P_{SKIPS}*-iterated-combine-Seq*$_{ptick}$*-behaviour*:
assumes *same-length*: ‹*length* σs = *length As*›
shows ‹$P_{SKIPS}\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_d$ σs = $P_{SKIPS}\langle\langle\langle_{nd}\bigotimes;_{\checkmark}$ *map* (λA *r*. $\langle\langle A$ *r*$\rangle\rangle_{d\hookrightarrow nd}$) *As*$\rangle\rangle$ *r*$\rangle_{nd}$ σs›
proof (*fold* P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*, *rule* P_{SKIPS}*-nd-eqI-strong-id*)
show ‹$\sigma t \in \mathcal{R}_{nd}$ $\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$
τ ($\langle\langle_{nd}\bigotimes;_{\checkmark}$ *map* (λA *r*. $\langle\langle A$ *r*$\rangle\rangle_{d\hookrightarrow nd}$) *As*$\rangle\rangle$ *r*) σt *a* = τ $\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_{d\hookrightarrow nd}$
σt *a*› **for** σt *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 ‹ω ($\langle\langle_{nd}\bigotimes;_{\checkmark}$ *map* (λA *r*. $\langle\langle A$ *r*$\rangle\rangle_{d\hookrightarrow nd}$) *As*$\rangle\rangle$ *r*) σt = ω $\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_{d\hookrightarrow nd}$
σt› **for** σt
by (*simp add*: ω*-iterated-combine-Seq*$_{ptick}$*-det-ndet-conv*)
qed

lemma *P-iterated-combine-Seq*$_{ptick}$*-behaviour*:
assumes *same-length*: ‹*length* σs = *length As*›
shows ‹$P\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_d$ σs = $P\langle\langle\langle_{nd}\bigotimes;_{\checkmark}$ *map* (λA *r*. $\langle\langle A$ *r*$\rangle\rangle_{d\hookrightarrow nd}$) *As*$\rangle\rangle$ *r*$\rangle_{nd}$
σs›
proof (*fold P-nd-from-det-to-ndet-is-P-d*, *rule P-nd-eqI-strong-id*)
show ‹$\sigma t \in \mathcal{R}_{nd}$ $\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_{d\hookrightarrow nd}$ $\sigma s \Longrightarrow$
τ ($\langle\langle_{nd}\bigotimes;_{\checkmark}$ *map* (λA *r*. $\langle\langle A$ *r*$\rangle\rangle_{d\hookrightarrow nd}$) *As*$\rangle\rangle$ *r*) σt *a* = τ $\langle\langle\langle_d\bigotimes;_{\checkmark}$ *As*$\rangle\rangle$ *r*$\rangle_{d\hookrightarrow nd}$
σt *a*› **for** σt *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}-*nd-combine*$_{Pair}$-*Seq*$_{ptick}$:
fixes A_0 A_1
assumes *at-most-1-elem-term* : ‹*at-most-1-elem-term* A_0›
— This assumption is necessary in the new setup, otherwise the result is not always a Procomaton (for example if ω A_0 σ_0 = *UNIV*, we have P σ_0 $;_{\checkmark}$ Q σ_1 = *GlobalNdet UNIV* (Q σ_1)).

defines *A-def*: ‹$A \equiv \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle$›
defines *P-def*: ‹$P \equiv P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}$›
and *Q-def*: ‹$Q \equiv \lambda\sigma_1\ r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma_1$›
and *S-def*: ‹$S \equiv P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$›
shows ‹P σ_0 $;_{\checkmark}$ Q σ_1 = S (σ_0, σ_1)›
proof −
let *?f* = ‹P_{SKIPS}-*nd-step* (ε A) (τ A) (ω A) ($\lambda\sigma'$. *case* σ' *of* $(\sigma_0, \sigma_1) \Rightarrow P$ σ_0 $;_{\checkmark}$ Q σ_1)›
note *cartprod-rwrt* = *GlobalNdet-cartprod*[*of* - - ‹$\lambda x\ y$. - (x, y)›, *simplified*]
note *Ndet-and-Seq* = *Seq*$_{ptick}$-*distrib-GlobalNdet-left Seq*$_{ptick}$-*distrib-GlobalNdet-right*
have *P-rec* : ‹P σ_0 = P_{SKIPS}-*nd-step* (ε A_0) (τ A_0) (ω A_0) P σ_0› **for** σ_0
by (*fact restriction-fix-eq*[*OF* P_{SKIPS}-*nd-step-constructive-bis*[*of* A_0], *folded* P_{SKIPS}-*nd-def P-def*, *THEN fun-cong*])
have *Q-rec* : ‹Q σ_1 = (λr. P_{SKIPS}-*nd-step* (ε (A_1 r)) (τ (A_1 r)) (ω (A_1 r)) ($\lambda\sigma_1$. Q σ_1 r) σ_1)› **for** σ_1
by (*rule ext*, *simp add*: *Q-def* P_{SKIPS}-*nd-rec*)
show ‹P σ_0 $;_{\checkmark}$ Q σ_1 = S (σ_0, σ_1)›
proof (*rule fun-cong*[*of* ‹$\lambda(\sigma_0, \sigma_1)$.$P$ σ_0 $;_{\checkmark}$ Q σ_1› - ‹(σ_0, σ_1)›, *simplified*])
show ‹($\lambda(\sigma_0, \sigma_1)$. P σ_0 $;_{\checkmark}$ Q σ_1) = S›
proof (*rule restriction-fix-unique*[*OF* P_{SKIPS}-*nd-step-constructive-bis*[*of* A], *symmetric*, *folded* P_{SKIPS}-*nd-def S-def*])
show ‹*?f* = ($\lambda(\sigma_0, \sigma_1)$. P σ_0 $;_{\checkmark}$ Q σ_1)›
proof (*rule ext*, *clarify*)
show ‹*?f* (σ_0, σ_1) = P σ_0 $;_{\checkmark}$ Q σ_1› **for** σ_0 σ_1
proof (*cases* ‹$\sigma_0 \in \varrho$ A_0›)
show ‹$\sigma_0 \notin \varrho$ $A_0 \Longrightarrow$ *?f* (σ_0, σ_1) = P σ_0 $;_{\checkmark}$ Q σ_1›
by (*subst* (*2*) *P-rec*)
(*auto simp add*: *combine*$_{Pair}$-*Seq*$_{ptick}$-*defs* ε-*simps A-def* ϱ-*simps Mprefix-Seq*$_{ptick}$ *cartprod-rwrt Ndet-and-Seq intro*!: *mono-Mprefix-eq*)
next
assume ‹$\sigma_0 \in \varrho$ A_0›
hence ‹ω A_0 $\sigma_0 \neq \{\}$› **by** (*simp add*: ϱ-*simps*)
then obtain r **where** ‹ω A_0 $\sigma_0 = \{r\}$›
by (*meson at-most-1-elem-term at-most-1-elem-termE*)
from ‹$\sigma_0 \in \varrho$ A_0› **have** *P-rec'* : ‹P σ_0 = *SKIPS* (ω A_0 σ_0)› **by** (*simp add*: *P-rec* ϱ-*simps*)
have ε-*A* : ‹ε A (σ_0, σ_1) = (*if* $\sigma_1 \in \varrho$ (A_1 r) *then* $\{\}$ *else* ε (A_1 r) σ_1)›

by (*simp add*: *A-def* ε-*combine*$_{Pair}$-*Seq*$_{ptick}$ *combine-Seq-*ε-*defs* ‹$\sigma_0 \in \varrho\ A_0$› ‹$\omega\ A_0\ \sigma_0 = \{r\}$›)
from ‹$\sigma_0 \in \varrho\ A_0$› **have** ω-A : ‹$\omega\ A\ (\sigma_0, \sigma_1) = \omega\ (A_1\ r)\ \sigma_1$›
by (*simp add*: *A-def combine*$_{Pair}$-*Seq*$_{ptick}$-*defs* ‹$\omega\ A_0\ \sigma_0 = \{r\}$›)
show ‹*?f* $(\sigma_0, \sigma_1) = P\ \sigma_0\ ;_{\checkmark}\ Q\ \sigma_1$›
proof (*cases* ‹$\sigma_1 \in \varrho\ (A_1\ r)$›)
show ‹$\sigma_1 \in \varrho\ (A_1\ r) \Longrightarrow$ *?f* $(\sigma_0, \sigma_1) = P\ \sigma_0\ ;_{\checkmark}\ Q\ \sigma_1$›
by (*subst* (*2*) *Q-rec*)
(*simp add*: *P-rec'* ε-*A* ω-*A SKIPS-def Ndet-and-Seq* ‹$\omega\ A_0\ \sigma_0 = \{r\}$› ϱ-*simps*)
next
show ‹$\sigma_1 \notin \varrho\ (A_1\ r) \Longrightarrow$ *?f* $(\sigma_0, \sigma_1) = P\ \sigma_0\ ;_{\checkmark}\ Q\ \sigma_1$›
by (*subst* (*2*) *Q-rec*, *unfold* ε-*A* ω-*A SKIPS-def*)
(*auto simp add*: *A-def P-rec' Ndet-and-Seq* ‹$\omega\ A_0\ \sigma_0 = \{r\}$› ϱ-*simps cartprod-rwrt combine*$_{Pair}$-*Seq*$_{ptick}$-*defs intro*!: *mono-Mprefix-eq*)
qed
qed
qed
qed
qed
qed

corollary P_{SKIPS}-*d-combine*$_{Pair}$-*Seq*$_{ptick}$:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d\ \sigma_1) = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\!\rangle_d\ (\sigma_0, \sigma_1)$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d\ \sigma_1) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_1)$›
by (*simp add*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Pair}\ (\lambda r.\ \langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd})\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0, \sigma_1)$›
by (*metis* P_{SKIPS}-*nd-combine*$_{Pair}$-*Seq*$_{ptick}$ *at-most-1-elem-from-det-to-ndet at-most-1-elem-imp-at-most-1-elem-term*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\!\rangle_d\ (\sigma_0, \sigma_1)$›
by (*simp add*: P_{SKIPS}-*combine*$_{Pair}$-*Seq*$_{ptick}$-*behaviour*)
finally show *?thesis* **.**
qed

Pairlist

lemma P_{SKIPS}-*nd-combine*$_{Pairlist}$-*Seq*$_{ptick}$:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma_1) = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›
if ‹*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 ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma_1) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\!\rangle_{nd}\ (\sigma_0, \sigma_1)$› **.**
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›

proof (*rule* P_{SKIPS}*-nd-eqI-strong*[*of* ‹$\lambda(\sigma_0, \sigma_1).\ [\sigma_0, \sigma_1]$› - ‹$(\sigma_0, \sigma_1)$›, *simplified*])
show ‹*inj-on* $(\lambda(\sigma_0, \sigma_1).\ [\sigma_0, \sigma_1])$ $(\mathcal{R}_{nd}\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ (\sigma_0, \sigma_1))$›
by (*auto intro*: *inj-onI*)
next
show ‹$\tau\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle$ (*case* σ' *of* $(\sigma_0, \sigma_1) \Rightarrow [\sigma_0, \sigma_1])\ a =$
$(\lambda\sigma'.$ *case* σ' *of* $(\sigma_0, \sigma_1) \Rightarrow [\sigma_0, \sigma_1])$ ‘$\tau\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma'\ a$› **for** $\sigma'\ a$
by (*cases* σ') (*auto simp add*: *combine*$_{Pair}$*-Seq*$_{ptick}$*-defs combine*$_{Pairlist}$*-Seq*$_{ptick}$*-defs*)
next
show ‹$\omega\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle$ (*case* σ' *of* $(\sigma_0, \sigma_1) \Rightarrow [\sigma_0, \sigma_1]) = \omega\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma'$› **for** σ'
by (*cases* σ') (*auto simp add*: *combine*$_{Pair}$*-Seq*$_{ptick}$*-defs combine*$_{Pairlist}$*-Seq*$_{ptick}$*-defs*)
qed
finally show *?thesis* **.**
qed

corollary P_{SKIPS}*-d-combine*$_{Pairlist}$*-Seq*$_{ptick}$:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d\ \sigma_1) = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0, \sigma_1]$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d\ \sigma_1) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_1)$›
by (*simp add*: P_{SKIPS}*-nd-from-det-to-ndet-is-*P_{SKIPS}*-d*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes;_{\checkmark Pairlist}\ (\lambda r.\ \langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd})\rangle\!\rangle\rangle\!\rangle_{nd}\ [\sigma_0, \sigma_1]$›
by (*metis* P_{SKIPS}*-nd-combine*$_{Pairlist}$*-Seq*$_{ptick}$ *at-most-1-elem-from-det-to-ndet*
at-most-1-elem-imp-at-most-1-elem-term)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes;_{\checkmark Pairlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ [\sigma_0, \sigma_1]$›
by (*simp add*: P_{SKIPS}*-combine*$_{Pairlist}$*-Seq*$_{ptick}$*-behaviour*)
finally show *?thesis* **.**
qed

Rlist

lemma P_{SKIPS}*-nd-combine*$_{Rlist}$*-Seq*$_{ptick}$:
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma s) = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma s)$›
if ‹*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 ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma s) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0, \sigma s)$› **.**
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma s)$›
proof (*rule* P_{SKIPS}*-nd-eqI-strong*[*of* ‹$\lambda(\sigma_0, \sigma s).\ \sigma_0\ \#\ \sigma s$› - ‹$(\sigma_0, \sigma s)$›, *simplified*])
show ‹*inj-on* $(\lambda(\sigma_0, \sigma s).\ \sigma_0\ \#\ \sigma s)$ $(\mathcal{R}_{nd}\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ (\sigma_0, \sigma s))$›
by (*auto intro*: *inj-onI*)
next
show ‹$\tau\ \langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle$ (*case* σ' *of* $(\sigma_0, \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s)\ a =$

$(\lambda\sigma'.\ case\ \sigma'\ of\ (\sigma_0,\ \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s)$ ‘ $\tau\ \langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma'\ a$› **for** $\sigma'\ a$
by (*cases* σ') (*auto simp add*: $combine_{Pair}$-Seq_{ptick}-*defs* $combine_{Rlist}$-Seq_{ptick}-*defs*)
next
show ‹$\omega\ \langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\ (case\ \sigma'\ of\ (\sigma_0,\ \sigma s) \Rightarrow \sigma_0\ \#\ \sigma s) = \omega\ \langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ \sigma'$› **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} :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d\ \sigma s) = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ \#\ \sigma s)$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d\ \sigma s) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}\ \sigma s)$›
by (*simp add*: P_{SKIPS}-*nd-from-det-to-ndet-is-*P_{SKIPS}-*d*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ _{nd}\otimes;_{\checkmark Rlist}\ (\lambda r.\ \langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd})\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma_0\ \#\ \sigma s)$›
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 ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _d\otimes;_{\checkmark Rlist}\ A_1\rangle\!\rangle\rangle\!\rangle_d\ (\sigma_0\ \#\ \sigma s)$›
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} :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma s_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma s_1) = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma s_0\ @\ \sigma s_1)$›
if *same-length-reach* : ‹$\bigwedge\sigma s_0'.\ \sigma s_0' \in \mathcal{R}_{nd}\ A_0\ \sigma s_0 \Longrightarrow length\ \sigma s_0' = len_0$›
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 ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_{nd}\ \sigma s_0\ ;_{\checkmark}\ (\lambda r.\ P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_{nd}\ \sigma s_1) =$
$P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma s_0,\ \sigma s_1)$› **.**
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ _{nd}\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_{nd}\ (\sigma s_0\ @\ \sigma s_1)$›
proof (*rule* P_{SKIPS}-*nd-eqI-strong*[*of* ‹$\lambda(\sigma s_0,\ \sigma s_1).\ \sigma s_0\ @\ \sigma s_1$› - ‹$(\sigma s_0,\ \sigma s_1)$›, *simplified*])
show ‹*inj-on* $(\lambda(\sigma s_0,\ \sigma s_1).\ \sigma s_0\ @\ \sigma s_1)\ (\mathcal{R}_{nd}\ \langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ (\sigma s_0,\ \sigma s_1))$›
proof (*rule inj-onI*, *clarify*)
fix $\sigma s_0'\ \sigma s_1'\ \sigma s_0''\ \sigma s_1''$
assume *assms* : ‹$(\sigma s_0',\ \sigma s_1') \in \mathcal{R}_{nd}\ \langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ (\sigma s_0,\ \sigma s_1)$›
‹$(\sigma s_0'',\ \sigma s_1'') \in \mathcal{R}_{nd}\ \langle\!\langle A_0\ _{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle\ (\sigma s_0,\ \sigma s_1)$›
‹$\sigma s_0'\ @\ \sigma s_1' = \sigma s_0''\ @\ \sigma s_1''$›
from $\mathcal{R}_{nd}$-$combine_{ndPair}$-Seq_{ptick}-*subset assms*(*1*, *2*)
have ‹$\sigma s_0' \in \mathcal{R}_{nd}\ A_0\ \sigma s_0$› ‹$\sigma s_0'' \in \mathcal{R}_{nd}\ A_0\ \sigma s_0$› **by** *fast+*
with *same-length-reach* **have** ‹$length\ \sigma s_0' = length\ \sigma s_0''$› **by** *blast*

with *assms*(*3*) **show** ‹$\sigma s_0{}' = \sigma s_0{}'' \wedge \sigma s_1{}' = \sigma s_1{}''$› **by** *simp*
qed
next
fix $\sigma s'$ a
assume ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle$ $(\sigma s_0,\ \sigma s_1)$›
with $\mathcal{R}_{nd}$-$combine_{ndPair}$-Seq_{ptick}-*subset*
obtain $\sigma s_0{}'$ $\sigma s_1{}'$ **where** ‹$\sigma s' = (\sigma s_0{}',\ \sigma s_1{}')$› ‹$\sigma s_0{}' \in \mathcal{R}_{nd}$ A_0 σs_0› **by** *fast*
from ‹$\sigma s_0{}' \in \mathcal{R}_{nd}$ A_0 σs_0› *same-length-reach* **have** ‹*length* $\sigma s_0{}' = len_0$› **by** *blast*
show ‹τ $\langle\!\langle A_0\ {}_{nd}\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle$ (*case* $\sigma s'$ *of* $(\sigma s_0,\ \sigma s_1) \Rightarrow \sigma s_0$ @ σs_1) a = ($\lambda\sigma s'$. *case* $\sigma s'$ *of* $(\sigma s_0,\ \sigma s_1) \Rightarrow \sigma s_0$ @ σs_1) ‘ τ $\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle$ $\sigma s'$ a›
by (*auto simp add*: ‹$\sigma s' = (\sigma s_0{}',\ \sigma s_1{}')$› ‹*length* $\sigma s_0{}' = len_0$› $combine_{Pair}$-Seq_{ptick}-*defs* $combine_{ListslenL}$-Seq_{ptick})
next
fix $\sigma s'$ a
assume ‹$\sigma s' \in \mathcal{R}_{nd}$ $\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle$ $(\sigma s_0,\ \sigma s_1)$›
with $\mathcal{R}_{nd}$-$combine_{ndPair}$-Seq_{ptick}-*subset*
obtain $\sigma s_0{}'$ $\sigma s_1{}'$ **where** ‹$\sigma s' = (\sigma s_0{}',\ \sigma s_1{}')$› ‹$\sigma s_0{}' \in \mathcal{R}_{nd}$ A_0 σs_0› **by** *fast*
from ‹$\sigma s_0{}' \in \mathcal{R}_{nd}$ A_0 σs_0› *same-length-reach* **have** ‹*length* $\sigma s_0{}' = len_0$› **by** *blast*
show ‹ω $\langle\!\langle A_0\ {}_{nd}\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle$ (*case* $\sigma s'$ *of* $(\sigma s_0,\ \sigma s_1) \Rightarrow \sigma s_0$ @ σs_1) = ω $\langle\!\langle A_0\ {}_{nd}\otimes;_{\checkmark Pair}\ A_1\rangle\!\rangle$ $\sigma s'$›
by (*auto simp add*: ‹$\sigma s' = (\sigma s_0{}',\ \sigma s_1{}')$› ‹*length* $\sigma s_0{}' = len_0$› $combine_{Pair}$-Seq_{ptick}-*defs* $combine_{ListslenL}$-Seq_{ptick})
qed
finally show *?thesis* **.**
qed

corollary P_{SKIPS}-*d*-$combine_{ListslenL}$-Seq_{ptick} :
‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d$ σs_0 $;_{\checkmark}$ (λr. $P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d$ σs_1) = $P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d$ (σs_0 @ σs_1)›
if *same-length-reach* : ‹$\bigwedge\sigma s_0{}'$. $\sigma s_0{}' \in \mathcal{R}_d$ A_0 $\sigma s_0 \Longrightarrow$ *length* $\sigma s_0{}' = len_0$›
proof −
have ‹$P_{SKIPS}\langle\!\langle A_0\rangle\!\rangle_d$ σs_0 $;_{\checkmark}$ (λr. $P_{SKIPS}\langle\!\langle A_1\ r\rangle\!\rangle_d$ σs_1) = $P_{SKIPS}\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}$ σs_0 $;_{\checkmark}$ (λr. $P_{SKIPS}\langle\!\langle\langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd}\rangle\!\rangle_{nd}$ σs_1)›
by (*simp add*: P_{SKIPS}-*nd*-*from*-*det*-*to*-*ndet*-*is*-P_{SKIPS}-*d*)
also have ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle\langle\!\langle A_0\rangle\!\rangle_{d\hookrightarrow nd}\ {}_{nd}\otimes len_0;_{ListslenL}$ (λr. $\langle\!\langle A_1\ r\rangle\!\rangle_{d\hookrightarrow nd}$)$\rangle\!\rangle\rangle\!\rangle_{nd}$ (σs_0 @ σs_1)›
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 ‹$\ldots = P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ {}_d\otimes len_0;_{ListslenL}\ A_1\rangle\!\rangle\rangle\!\rangle_d$ (σs_0 @ σs_1)›
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}:

‹$[\![$*length* σs = *length As*; $\bigwedge A\ r.\ A \in$ *set* (*butlast As*) $\Longrightarrow$ *at-most-1-elem-term* (A r)$]\!] \Longrightarrow$
SEQ$_{\checkmark}$ (σ, A) $\in$@ *zip* σs *As*. ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{nd}\ \sigma$) = ($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes;_{\checkmark}$ *As*$\rangle\!\rangle$ $r\rangle\!\rangle_{nd}\ \sigma s$)›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp add*: P_{SKIPS}*-nd-rec*)
next
case (*single* σ_0 A_0) **show** *?case* **by** (*simp add*: P_{SKIPS}*-nd-from-*σ*-to-*σs*-is-*P_{SKIPS}*-nd*)
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 *As*)
have ‹*SEQ*$_{\checkmark}$ (σ, A) $\in$@ *zip* (σ_0 # σ_1 # σs) (A_0 # A_1 # *As*). ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{nd}\ \sigma$) =
($\lambda r.\ P_{SKIPS}\langle\!\langle A_0\ r\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}$ *SEQ*$_{\checkmark}$ (σ, A) $\in$@ *zip* (σ_1 # σs) (A_1 # *As*). ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{nd}\ \sigma$))› **by** *simp*
also have ‹*SEQ*$_{\checkmark}$ (σ, A) $\in$@ *zip* (σ_1 # σs) (A_1 # *As*). ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{nd}$ σ) =
($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ A_1$ # *As*$\rangle\!\rangle$ $r\rangle\!\rangle_{nd}$ (σ_1 # σs))›
by (*rule Cons.hyps*(*3*)) (*simp add*: *Cons.prems*)
also have ‹($\lambda r.\ P_{SKIPS}\langle\!\langle A_0\ r\rangle\!\rangle_{nd}\ \sigma_0\ ;_{\checkmark}$ ($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ A_1$ # *As* $\rangle\!\rangle$ $r\rangle\!\rangle_{nd}$ (σ_1 # σs))) =
($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ r\ _{nd}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{nd}\bigotimes;_{\checkmark}\ A_1$ # *As*$\rangle\!\rangle\rangle\!\rangle_{nd}$ (σ_0 # σ_1 # σs))›
by (*intro ext* P_{SKIPS}*-nd-combine*$_{Rlist}$*-Seq*$_{ptick}$) (*simp add*: *Cons.prems*)
also have ‹... = ($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{nd}\bigotimes;_{\checkmark}\ A_0$ # A_1 # *As*$\rangle\!\rangle$ $r\rangle\!\rangle_{nd}$ (σ_0 # σ_1 # σs))› **by** *simp*
finally show *?case* **.**
qed

corollary P_{SKIPS}*-d-compactification-Seq*$_{ptick}$:
‹*length* σs = *length As* $\Longrightarrow$
SEQ$_{\checkmark}$ (σ, A) $\in$@ *zip* σs *As*. ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{d}\ \sigma$) = ($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{d}\bigotimes;_{\checkmark}$ *As*$\rangle\!\rangle$ $r\rangle\!\rangle_{d}\ \sigma s$)›
proof (*induct* σs *As rule*: *induct-2-lists012*)
case *Nil* **show** *?case* **by** (*simp add*: P_{SKIPS}*-d-rec*)
next
case (*single* σ_0 A_0) **show** *?case* **by** (*simp add*: P_{SKIPS}*-d-from-*σ*-to-*σs*-is-*P_{SKIPS}*-d*)
next
case (*Cons* σ_0 σ_1 σs A_0 A_1 *As*)
have ‹*SEQ*$_{\checkmark}$ (σ, A) $\in$@ *zip* (σ_0 # σ_1 # σs) (A_0 # A_1 # *As*). ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{d}$ σ) =
($\lambda r.\ P_{SKIPS}\langle\!\langle A_0\ r\rangle\!\rangle_{d}\ \sigma_0\ ;_{\checkmark}$ *SEQ*$_{\checkmark}$ (σ, A) $\in$@ *zip* (σ_1 # σs) (A_1 # *As*). ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{d}\ \sigma$))› **by** *simp*
also have ‹*SEQ*$_{\checkmark}$ (σ, A) $\in$@ *zip* (σ_1 # σs) (A_1 # *As*). ($\lambda r.\ P_{SKIPS}\langle\!\langle A\ r\rangle\!\rangle_{d}\ \sigma$) =
($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{d}\bigotimes;_{\checkmark}\ A_1$ # *As*$\rangle\!\rangle$ $r\rangle\!\rangle_{d}$ (σ_1 # σs))›
by (*fact Cons.hyps*(*3*))
also have ‹($\lambda r.\ P_{SKIPS}\langle\!\langle A_0\ r\rangle\!\rangle_{d}\ \sigma_0\ ;_{\checkmark}$ ($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle_{d}\bigotimes;_{\checkmark}\ A_1$ # *As*$\rangle\!\rangle$ $r\rangle\!\rangle_{d}$ (σ_1 # σs))) =
($\lambda r.\ P_{SKIPS}\langle\!\langle\langle\!\langle A_0\ r\ _{d}\otimes;_{\checkmark Rlist}\ \langle\!\langle_{d}\bigotimes;_{\checkmark}\ A_1$ # *As*$\rangle\!\rangle\rangle\!\rangle_{d}$ (σ_0 # σ_1 # σs))›

by (*intro ext* P_{SKIPS}-*d-combine*$_{Rlist}$-*Seq*$_{ptick}$)
also have ‹$\ldots = (\lambda r.\ P_{SKIPS}\langle\langle_d \bigotimes ;_{\checkmark}\ A_0\ \#\ A_1\ \#\ As\rangle\ r\rangle_d\ (\sigma_0\ \#\ \sigma_1\ \#\ \sigma s))$›
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*: ‹$n > 1 \Longrightarrow i \neq Suc\ i\ mod\ n$›
  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*: ‹$n > 1 \Longrightarrow \neg\ n \leq Suc\ 0$›
  by *simp*

lemma *minus-suc*: ‹$n > 0 \Longrightarrow n - Suc\ 0 \neq n$›
  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*:
  ‹$RPHIL\ i = (\upsilon\ X.\ picks\ i\ i \rightarrow picks\ i\ ((i - 1)\ mod\ N) \rightarrow$
    $putsdown\ i\ ((i - 1)\ mod\ N) \rightarrow putsdown\ i\ i \rightarrow X)$›
  by (*simp add*: *RPHIL-def restriction-fix-is-fix*)

lemma *LPHIL0-restriction-fix-def*:
  ‹$LPHIL0 = (\upsilon\ X.\ picks\ 0\ (N - 1) \rightarrow picks\ 0\ 0 \rightarrow$
    $putsdown\ 0\ 0 \rightarrow putsdown\ 0\ (N - 1) \rightarrow X)$›
  by (*simp add*: *LPHIL0-def restriction-fix-is-fix*)

lemma *FORK-restriction-fix-def*:
‹$FORK\ i = (\upsilon\ X.\ (picks\ i\ i \rightarrow putsdown\ i\ i \rightarrow X)\ \Box$
$(picks\ ((i + 1)\ mod\ N)\ i \rightarrow putsdown\ ((i + 1)\ mod\ N)\ i \rightarrow X))$›
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*]: ‹$N > 0$›
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 $\sigma_{fork} = nat$
type-synonym $id_{phil} = nat$
type-synonym $\sigma_{phil} = nat$

definition *fork-A* :: ‹$id_{fork} \Rightarrow (\sigma_{fork},\ dining\text{-}event,\ unit)\ A_d$› (‹$A_f$›)
where ‹$A_f\ i \equiv recursive\text{-}constructor\text{-}A_d$
$[((0,\ picks\ i\ i),\ \lfloor 1 \rfloor),\ ((0,\ picks\ ((i + 1)\ mod\ N)\ i),\ \lfloor 2 \rfloor),$
$((1,\ putsdown\ i\ i),\ \lfloor 0 \rfloor),\ ((2,\ putsdown\ ((i + 1)\ mod\ N)\ i),\ \lfloor 0 \rfloor)]$
$(\lambda\sigma.\ \Diamond)$›

definition *rphil-A* :: ‹$id_{phil} \Rightarrow (\sigma_{phil},\ dining\text{-}event,\ unit)\ A_d$› (‹$A_{rp}$›)
where ‹$A_{rp}\ i \equiv recursive\text{-}constructor\text{-}A_d$
$[((0,\ picks\ i\ i),\ \lfloor 1 \rfloor),\ ((1,\ picks\ i\ ((i-1)\ mod\ N)),\ \lfloor 2 \rfloor),$
$((2,\ putsdown\ i\ ((i-1)\ mod\ N)),\ \lfloor 3 \rfloor),\ ((3,\ putsdown\ i\ i),\ \lfloor 0 \rfloor)]$
$(\lambda\sigma.\ \Diamond)$›

definition *lphil0-A* :: ‹$(\sigma_{phil},\ dining\text{-}event,\ unit)\ A_d$› (‹$A_{lp}$›)
where ‹$A_{lp} \equiv recursive\text{-}constructor\text{-}A_d$

$[((0,\ picks\ 0\ (N-1)),\ \lfloor 1 \rfloor),\ ((1,\ picks\ 0\ 0),\ \lfloor 2 \rfloor),$
$((2,\ putsdown\ 0\ 0),\ \lfloor 3 \rfloor),\ ((3,\ putsdown\ 0\ (N-1)),\ \lfloor 0 \rfloor)]\ (\lambda\sigma.\ \Diamond)$›

Definition and first properties of associated normal processes

definition *fork-P-d* :: ‹$id_{fork} \Rightarrow \sigma_{fork} \Rightarrow$ *dining-event process*› **where** ‹*fork-P-d* $i \equiv P\langle\!\langle A_f\ i \rangle\!\rangle_d$›
definition *rphil-P-d* :: ‹$id_{phil} \Rightarrow \sigma_{phil} \Rightarrow$ *dining-event process*› **where** ‹*rphil-P-d* $i \equiv P\langle\!\langle A_{rp}\ i \rangle\!\rangle_d$›
definition *lphil0-P-d* :: ‹$\sigma_{phil} \Rightarrow$ *dining-event process*› **where** ‹*lphil0-P-d* $\equiv P\langle\!\langle A_{lp} \rangle\!\rangle_d$›

lemmas *fork-P-d-rec* = *P-d-rec*[*of* ‹A_f -›, *folded fork-P-d-def*]
and *rphil-P-d-rec* = *P-d-rec*[*of* ‹A_{rp} -›, *folded rphil-P-d-def*]
and *lphil0-P-d-rec* = *P-d-rec*[*of* ‹A_{lp}›, *folded lphil0-P-d-def*]

schematic-goal *fork-ε*: ‹$\varepsilon\ (A_f\ i)\ \sigma = ?S$›
and *rphil-ε*: ‹$\varepsilon\ (A_{rp}\ i)\ \sigma = ?T$›
and *lphil0-ε*: ‹$\varepsilon\ A_{lp}\ \sigma = ?U$›
unfolding *fork-A-def rphil-A-def lphil0-A-def* **by** (*all ε-det-calc*)

schematic-goal *fork-τ*: ‹$\tau\ (A_f\ i)\ \sigma = ?S$›
and *rphil-τ*: ‹$\tau\ (A_{rp}\ i)\ \sigma = ?T$›
and *lphil0-τ*: ‹$\tau\ A_{lp}\ \sigma = ?U$›
unfolding *fork-A-def rphil-A-def lphil0-A-def* **by** (*all τ-det-calc*)

corollary *ev-id*$_{fork}x$: ‹$e \in \varepsilon\ (A_f\ i)\ \sigma \Longrightarrow fork\ e = i$›
and *rphil-phil*: ‹$e \in \varepsilon\ (A_{rp}\ i)\ \sigma \Longrightarrow phil\ e = i$›
and *lphil0-phil*: ‹$e \in \varepsilon\ A_{lp}\ \sigma \Longrightarrow phil\ e = 0$›
by (*auto simp add*: *fork-ε rphil-ε lphil0-ε split*: *if-split-asm*)

corollary *ev-id*$_{phil}x$: ‹$i < n \Longrightarrow \sigma \in \varepsilon\ ((A_{lp}\ \#\ map\ A_{rp}\ [1..< n])\ !\ i)\ s \Longrightarrow phil\ \sigma = i$›
by (*cases* ‹$i = 0$›) (*simp-all add*: *lphil0-phil rphil-phil*)

lemma *indep-forks*: ‹$i \neq j \Longrightarrow \varepsilon\ (A_f\ i)\ \sigma \cap \varepsilon\ (A_f\ j)\ \sigma' = \{\}$›
and *indep-phils*: ‹$i \neq 0 \Longrightarrow \varepsilon\ A_{lp}\ \sigma \cap \varepsilon\ (A_{rp}\ i)\ \sigma' = \{\}$›
‹$i \neq j \Longrightarrow \varepsilon\ (A_{rp}\ i)\ \sigma \cap \varepsilon\ (A_{rp}\ j)\ \sigma' = \{\}$›
using *ev-id*$_{fork}x$ *lphil0-phil rphil-phil* **by** (*blast*, *simp*, *fastforce*, *blast*)

Equalities between *FORK*, *RPHIL*, *LPHIL0* and respectively *fork-P-d*, *rphil-P-d*, *lphil0-P-d*

lemma *FORK-is-fork-P-d*: ‹*FORK i* = *fork-P-d i 0*›
proof (*unfold fork-P-d-def*)

have ‹$(0 :: nat) < 2$› **by** *simp*
thus ‹$FORK\ i = P\langle\!\langle A_f\ i\rangle\!\rangle_d\ 0$›
proof (*induct rule*: *P-d-induct-iterated*)
show ‹$FORK\ i = X\ 0 \Longrightarrow FORK\ i = (P\text{-}d\text{-}step\ (\varepsilon\ (A_f\ i))\ (\tau\ (A_f\ i))\ \hat{}\hat{}\ 2)\ X\ 0$› **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*: ‹$RPHIL\ i = rphil\text{-}P\text{-}d\ i\ 0$›
proof (*unfold rphil-P-d-def*)
have ‹$(0 :: nat) < 4$› **by** *simp*
thus ‹$RPHIL\ i = P\langle\!\langle A_{rp}\ i\rangle\!\rangle_d\ 0$›
proof (*induct rule*: *P-d-induct-iterated*)
show ‹$RPHIL\ i = X\ 0 \Longrightarrow RPHIL\ i = (P\text{-}d\text{-}step\ (\varepsilon\ (A_{rp}\ i))\ (\tau\ (A_{rp}\ i))\ \hat{}\hat{}\ 4)\ X\ 0$› **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 *LPHIL0-is-lphil0-P-d*: ‹$LPHIL0 = lphil0\text{-}P\text{-}d\ 0$›
proof (*unfold lphil0-P-d-def*)
have ‹$(0 :: nat) < 4$› **by** *simp*
thus ‹$LPHIL0 = P\langle\!\langle A_{lp}\rangle\!\rangle_d\ 0$›
proof (*induct rule*: *P-d-induct-iterated*)
show ‹$LPHIL0 = X\ 0 \Longrightarrow LPHIL0 = (P\text{-}d\text{-}step\ (\varepsilon\ A_{lp})\ (\tau\ A_{lp})\ \hat{}\hat{}\ 4)\ X\ 0$› **for** X
by (*subst LPHIL0-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 σ_{forks} = ‹*nat list*›

definition *forks-A* :: ‹$(\sigma_{forks},\ dining\text{-}event,\ unit)\ A_d$› (‹$A_F$›) **where** ‹$A_F \equiv \langle\!\langle {}_d\bigotimes [\![\{\}]\!]\ map\ A_f\ [0..<N]\rangle\!\rangle$›

Definition and first properties of the associated normal process

definition *forks-P-d*:: ‹$\sigma_{forks} \Rightarrow dining\text{-}event\ process$› **where** ‹$forks\text{-}P\text{-}d \equiv P\langle\!\langle A_F\rangle\!\rangle_d$›

lemma *forks-ε*: ‹*length fs* = *N* ⟹ ε A_F *fs* = (⋃*i*<*N*. ε (A_f *i*) (*fs* ! *i*))›
unfolding *forks-A-def* **using** *N-pos* **by** (*subst ε-iterated-combine*$_d$*-Sync-general-form*) *force*+

Equality between *FORKS* and *forks-P-d*

lemma *NFORKS-is-forks-P-d*: ‹*FORKS* = *forks-P-d* (*replicate N 0*)›
unfolding *forks-P-d-def forks-A-def FORKS-def*
apply (*subst P-d-compactification-Sync-upt-version*)
by (*simp-all add*: *FORK-is-fork-P-d*[*unfolded fork-P-d-def*] *indep-forks indep-enabl-def*)

11.3.3 *PHILS* is normalizable

Definition of the all-philosophers automaton

type-synonym σ_{phils} = ‹*nat list*›

definition *phils-A* :: ‹(σ_{phils}, *dining-event*, *unit*) A_d› (‹A_P›) **where** ‹A_P ≡ ⟪$_d$⨂ ⟦{}⟧ A_{lp} # *map* A_{rp} [*1*..< *N*]⟫›

lemma *phils-A-def-bis*: ‹A_P = ⟪$_d$⨂ ⟦{}⟧ *map* (λ*i*. *if i* = *0 then* A_{lp} *else* A_{rp} *i*) [*0*..<*N*]⟫›
unfolding *phils-A-def* **apply** (*subst* (*2*) *upt-rec*, *simp*)
apply (*subgoal-tac* ‹*map* (λ*i*. *if i* = *0 then* A_{lp} *else* A_{rp} *i*) [*Suc 0*..<*N*] = *map* A_{rp} [*Suc 0*..<*N*]›)
by (*presburger*, *subst list-eq-iff-nth-eq*, *simp*)

Definition and first properties of the associated normal process

definition *phils-P-d*:: ‹σ_{phils} ⇒ *dining-event process*› **where** ‹*phils-P-d* ≡ *P*⟪A_P⟫$_d$›

lemma *phils-ε*: ‹*length ps* = *N* ⟹ ε A_P *ps* = ε A_{lp} (*ps* ! *0*) ∪ (⋃*i*∈{*1*..<*N*}. ε (A_{rp} *i*) (*ps* ! *i*))›
unfolding *phils-A-def-bis* **using** *N-pos*
by (*subst ε-iterated-combine*$_d$*-Sync-general-form*, (*auto split*: *if-splits*)+)

Equality between *PHILS* and *phils-P-d*

lemma *NPHILS-is-phils-P-d*: ‹*PHILS* = *phils-P-d* (*replicate N 0*)›
unfolding *phils-P-d-def phils-A-def-bis PHILS-def*
apply (*subst P-d-compactification-Sync-upt-version*[*symmetric*])
apply (*simp-all add*: *indep-enabl-def indep-phils*(*1*,*2*) *inf-sup-aci*(*1*))
apply (*subgoal-tac* ‹{*0*..<*N*} = *insert 0* {*1*..<*N*}›)
apply (*simp add*: *LPHIL0-is-lphil0-P-d lphil0-P-d-def*)
by (*rule arg-cong*[*OF image-mset-cong*]) (*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* :: ‹$(\sigma_{phils} \times \sigma_{forks}$, *dining-event*, *unit*) A_d› (‹A_D›) **where** ‹$A_D \equiv \langle\!\langle A_P\ {}_d\otimes [\![UNIV]\!]_{Pair}\ A_F \rangle\!\rangle$›

Definition and first properties of the associated normal process

definition *dining-P-d*:: ‹$\sigma_{phils} \times \sigma_{forks} \Rightarrow$ *dining-event process*› **where** ‹*dining-P-d* $\equiv P\langle\!\langle A_D \rangle\!\rangle_d$›

lemma *dining-ε*:
‹*length ps* $= N \Longrightarrow$ *length fs* $= N \Longrightarrow$
$\varepsilon\ A_D\ (ps, fs) = (\varepsilon\ A_{lp}\ (ps\ !\ 0) \cup (\bigcup i{\in}\{1..{<}N\}.\ \varepsilon\ (A_{rp}\ i)\ (ps\ !\ i))) \cap (\bigcup i{<}N.\ \varepsilon\ (A_f\ i)\ (fs\ !\ i))$›
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*: ‹*DINING* = *dining-P-d* (*replicate N 0*, *replicate N 0*)›
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 ‹*inv-dining ps fs* $\equiv$
$length\ fs = N \wedge length\ ps = N$
$\wedge\ (\forall i < N.\ fs\ !\ i = 0 \vee fs\ !\ i = 1 \vee fs\ !\ i = 2)$
$\wedge\ (\forall i < N.\ ps\ !\ i = 0 \vee ps\ !\ i = 1 \vee ps\ !\ i = 2 \vee ps\ !\ i = 3)$
$\wedge\ (\forall i.\ Suc\ i < N \longrightarrow ((fs\ !\ Suc\ i = 1) \longleftrightarrow ps\ !\ Suc\ i \neq 0)) \wedge (fs\ !\ (N - 1) = 2 \longleftrightarrow ps\ !\ 0 \neq 0)$
$\wedge\ (\forall i < N - 1.\ fs\ !\ i = 2 \longleftrightarrow ps\ !\ Suc\ i = 2) \wedge (fs\ !\ 0 = 1 \longleftrightarrow ps\ !\ 0 = 2)$›

lemma *show-inv-dining*:
‹$length\ fs = N \wedge length\ ps = N \Longrightarrow$
$(\forall i < N.\ fs\ !\ i = 0 \vee fs\ !\ i = 1 \vee fs\ !\ i = 2) \Longrightarrow$
$(\forall i < N.\ ps\ !\ i = 0 \vee ps\ !\ i = 1 \vee ps\ !\ i = 2 \vee ps\ !\ i = 3) \Longrightarrow$
$(\forall i.\ Suc\ i < N \longrightarrow (fs\ !\ Suc\ i = 1 \longleftrightarrow ps\ !\ Suc\ i \neq 0)) \Longrightarrow (fs\ !\ (N - 1) = 2 \longleftrightarrow ps\ !\ 0 \neq 0) \Longrightarrow$

$(\forall i < N - 1.\ fs\ !\ i = 2 \longleftrightarrow ps\ !\ Suc\ i = 2) \Longrightarrow (fs\ !\ 0\ = 1 \longleftrightarrow ps\ !\ 0 = 2)$
$\Longrightarrow$
inv-dining ps fs›
by (*simp add*: *inv-dining-def*)

lemma *inv-DINING*: ‹$s \in \mathcal{R}_d\ A_D$ (*replicate N 0*, *replicate N 0*) $\Longrightarrow$ *inv-dining* (*fst s*) (*snd s*)›
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*: ‹*t* = (*t-ps*, *t-fs*)› **and** *u-pair*: ‹*u* = (*u-ps*, *u-fs*)› **by** *fastforce*
hence *inv-hyp*: ‹*length t-fs* = *N*› ‹*length t-ps* = *N*›
‹$i < N \Longrightarrow$ *t-fs* ! *i* = *0* $\vee$ *t-fs* ! *i* = *1* $\vee$ *t-fs* ! *i* = *2*›
‹$i < N \Longrightarrow$ *t-ps* ! *i* = *0* $\vee$ *t-ps* ! *i* = *1* $\vee$ *t-ps* ! *i* = *2* $\vee$ *t-ps* ! *i* = *3*›
‹*Suc i* < *N* $\Longrightarrow$ (*t-fs* ! *Suc i* = *1*) = (*t-ps* ! *Suc i* $\neq$ *0*)›
‹(*t-fs* ! (*N* − *1*) = *2*) = (*t-ps* ! *0* $\neq$ *0*)›
‹$i < N - 1 \Longrightarrow$ (*t-fs* ! *i* = *2*) = (*t-ps* ! *Suc i* = *2*)›
‹(*t-fs* ! *0* = *1*) = (*t-ps* ! *0* = *2*)› **for** *i*
using *step.hyps*(*2*)[*simplified inv-dining-def*] **by** *simp-all*

have *u-in-*$\mathcal{R}_d$*-prem*: ‹(*u-ps*, *u-fs*) $\in \mathcal{R}_d\ A_P$ (*replicate N 0*) $\times\ \mathcal{R}_d\ A_F$ (*replicate N 0*)›
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*: ‹*length u-ps* = *N*› ‹*length u-fs* = *N*›
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*: ‹$\lfloor$*u-ps*$\rfloor$ = $\tau\ A_P$ *t-ps e*› ‹$\lfloor$*u-fs*$\rfloor$ = $\tau\ A_F$ *t-fs e*›
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*: ‹$e \in \varepsilon\ A_{lp}$ (*t-ps* ! *0*) $\cup$ ($\bigcup i{\in}\{1..{<}N\}.\ \varepsilon\ (A_{rp}\ i)$ (*t-ps* ! *i*))›
‹$e \in (\bigcup i{<}N.\ \varepsilon\ (A_f\ i)$ (*t-fs* ! *i*))›
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*:

‹$i < N \Longrightarrow$ *u-ps* ! i =
(*if* $i = 0$ *then* (*if* $e \in \varepsilon$ A_{lp} (*t-ps* ! *0*) *then* ⌈τ A_{lp} (*t-ps* ! *0*) e⌉ *else t-ps* ! *0*)
else if $e \in \varepsilon$ (A_{rp} i) (*t-ps* ! i) *then* ⌈τ (A_{rp} i) (*t-ps* ! i) e⌉ *else t-ps* ! i)›
‹$i < N \Longrightarrow$ *u-fs* ! i =
(*if* $e \in \varepsilon$ (A_f i) (*t-fs* ! i) *then* ⌈τ (A_f i) (*t-fs* ! i) e⌉ *else t-fs* ! i)› **for** i
by (*insert u-is*(*1*), *unfold phils-A-def*, *subst* (*asm*) τ*-iterated-combine*$_d$*-Sync-general-form*,
simp-all add: *inv-hyp*(*2*) *split*: *if-splits*)
(*insert u-is*(*2*), *unfold forks-A-def*, *subst* (*asm*) τ*-iterated-combine*$_d$*-Sync-general-form*,
simp-all add: *inv-hyp*(*1*) *split*: *if-splits*)
have ‹$e \in \varepsilon$ A_P *t-ps*› ‹$e \in \varepsilon$ A_F *t-fs*› **using** *u-is* ε*-simps* **by** *fastforce*+
hence *e-equiv*: ‹$e \in \varepsilon$ A_{lp} (*t-ps* ! *0*) $\longleftrightarrow$ *phil* e = *0*›
‹*Suc* $i < N \Longrightarrow e \in \varepsilon$ (A_{rp} (*Suc* i)) (*t-ps* ! *Suc* i) $\longleftrightarrow$ *phil* e = *Suc* i›
‹$i < N \Longrightarrow e \in \varepsilon$ (A_f i) (*t-fs* ! i) $\longleftrightarrow$ *fork* e = i› **for** i
apply (*simp-all add*: *phils-*ε[*OF inv-hyp*(*2*)] *forks-*ε[*OF inv-hyp*(*1*)])
using *rphil-phil lphil0-phil ev-id*$_{fork}$*x* **by** *auto* (*metis Suc-le-eq less-irrefl-nat*, *blast*, *metis*)

show *?case*
proof (*simp add*: *u-pair*, *rule show-inv-dining*[*rule-format*], *simp add*: *same-length-u*, *goal-cases*)
case (*1 i*) **thus** *?case* **using** *u-nth*(*2*)[*of i*] *inv-hyp*(*3*) **by** ε*-sets-simp A-defs-simp*
next
case (*2 i*) **thus** *?case* **using** *u-nth*(*1*)[*of i*] *inv-hyp*(*4*) **by** ε*-sets-simp A-defs-simp*
next
case (*3 i*)
with *u-nth*(*1*)[*of* ‹*Suc i*›] *u-nth*(*2*)[*of* ‹*Suc i*›] **show** *?case*
using *inv-hyp*(*5*)[*of i*] **apply** ε*-sets-simp* **apply** *A-defs-simp*
using *e-equiv*(*3*) *fork-*ε *e-equiv*(*2*) *rphil-*ε **by** *fastforce*+
next
case *4*
with *u-nth*(*1*)[*of 0*] *u-nth*(*2*) **show** *?case* **using** *inv-hyp*(*6*) *N-g1* **apply** ε*-sets-simp* **apply** *A-defs-simp*
apply (*metis N-pos One-nat-def Suc-pred fork-*ε *dining-event.sel*(*3*)
dining-event.simps(*3*) *inv-hyp*(*3*) *lessI singletonD e-equiv*(*3*))
using *lphil0-*ε *e-equiv*(*1*) **by** *force*+
next
case (*5 i*)
hence ‹*Suc* $i < N$› **by** *linarith*
with *u-nth*(*1*)[*of* ‹*Suc i*›] *u-nth*(*2*)[*of i*] *5* **show** *?case*
using *inv-hyp*(*7*)[*of i*] **apply** ε*-sets-simp* **apply** *A-defs-simp*
apply (*metis Suc-lessD fork-*ε *dining-event.sel*(*3*) *dining-event.simps*(*3*) *singletonD e-equiv*(*3*))
apply (*metis One-nat-def Suc-lessD bot-nat-0.not-eq-extremum inv-hyp*(*3*))
using *rphil-*ε *e-equiv*(*2*) **by** *force*+
next
case *6*
with *u-nth*(*1*)[*of 0*] *u-nth*(*2*)[*of 0*] **show** *?case*
using *N-g1 inv-hyp*(*8*) **apply** (*simp split*: *if-split-asm*) **apply** ε*-sets-simp* **apply** *A-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 *deadlock-free*

```
method nonempty-Int-by-common-element for x = rule-tac ex-in-conv[THEN iffD1,
OF exI, OF IntI, of x]

lemma inv-implies-DF: ‹ε A_D (ps, fs) ≠ {}› if hyp-inv: ‹inv-dining ps fs›
  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 ‹*RPHILS* ≡ ||| *P* ∈# *mset* (*map RPHIL* [*0*..< *N*]). *P*›

corollary ‹*N* = *3* ⟹ *RPHILS* = (*RPHIL 0* ||| *RPHIL 1* ||| *RPHIL 2*)›
unfolding *RPHILS-def* **by** (*simp add*: *eval-nat-numeral upt-rec Sync-assoc*)

definition *RDINING* :: ‹*dining-event process*›
where ‹*RDINING* = (*FORKS* || *RPHILS*)›

11.5.2 Normalization

definition *rphils-A* :: ‹(σ_{phils}, *dining-event*, *unit*) A_d› (‹A_{RP}›) **where** ‹A_{RP} ≡ $\langle\!\langle_d \bigotimes [\![\{\}]\!]$ *map* A_{rp} [*0*..< *N*]$\rangle\!\rangle$›

definition *rphils-P-d*:: ‹σ_{phils} ⇒ *dining-event process*› **where** ‹*rphils-P-d* ≡ $P\langle\!\langle A_{RP}\rangle\!\rangle_d$›

definition *rdining-A* :: ‹(σ_{phils} × σ_{forks}, *dining-event*, *unit*) A_d› (‹A_{RD}›) **where** ‹A_{RD} ≡ $\langle\!\langle A_{RP}\ {}_d\otimes[\![UNIV]\!]_{Pair}\ A_F\rangle\!\rangle$›

definition *rdining-P-d*:: ‹σ_{phils} × σ_{forks} ⇒ *dining-event process*› **where** ‹*rdining-P-d* ≡ $P\langle\!\langle A_{RD}\rangle\!\rangle_d$›

11.5.3 Correspondance between our normalized processes and the previous definitions

lemma *rphils-ε*: ‹*length ps* = *N* ⟹ ε A_{RP} *ps* = (⋃*i*∈{*0*..<*N*}. ε (A_{rp} *i*) (*ps* ! *i*))›
unfolding *rphils-A-def* **using** *N-pos*
by (*subst ε-iterated-combine*$_d$*-Sync-general-form*, (*auto split*: *if-splits*)+)

lemma *NRPHILS-is-rphils-P-d*: ‹*RPHILS* = *rphils-P-d* (*replicate N 0*)›
unfolding *rphils-P-d-def rphils-A-def RPHILS-def*
apply (*subst P-d-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-ε*:
‹*length ps* = *N* ⟹ *length fs* = *N* ⟹
ε A_{RD} (*ps*, *fs*) = (⋃*i*∈{*0*..<*N*}. ε (A_{rp} *i*) (*ps* ! *i*)) ∩ (⋃*i*<*N*. ε (A_f *i*) (*fs* ! *i*))›
by (*simp add*: *rdining-A-def ε-combine*$_{Pair}$*-Sync combine-Sync-ε-def forks-ε rphils-ε*)

lemma *RDINING-is-rdining-P-d*: ‹*RDINING* = *rdining-P-d* (*replicate N 0*, *replicate N 0*)›

apply (*unfold rdining-P-d-def rdining-A-def*)
apply (*subst P-d-combine*$_{Pair}$*-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*: ‹ε A_{RD} (*replicate N 1*, *replicate N 1*) = {}›
by (*subst rdining-ε*, *auto simp add*: *rphil-ε fork-ε*)

corollary *non-dealock-free-rdining*: ‹$\neg$ *deadlock-free* (*rdining-P-d* (*replicate N 1*, *replicate N 1*))›
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} (*replicate N 0*, *replicate N 0*)

lemma *rdining-τ*: ‹*length ps* = *N* $\Longrightarrow$ *length fs* = *N* $\Longrightarrow$ *e* $\in$ ε A_{RD} (*ps*, *fs*) $\Longrightarrow$
τ A_{RD} (*ps*, *fs*) *e* = $\lfloor(\lceil\tau\ A_{RP}\ ps\ e\rceil,\ \lceil\tau\ A_F\ fs\ e\rceil)\rfloor$›
by (*auto simp add*: *rdining-A-def combine*$_{Pair}$*-Sync-defs ε-simps split*: *if-split-asm*)

lemma *replicate1-reachable-from-replicate0-prelim*:
‹*n* $\leq$ *N* $\Longrightarrow$ (*replicate n 1* @ *replicate* (*N* − *n*) *0*, *replicate n 1* @ *replicate* (*N* − *n*) *0*) $\in \mathcal{R}_d$ A_{RD} (*replicate N 0*, *replicate N 0*)›
proof (*induct n*, *simp add*: $\mathcal{R}_d$*.init*)
case (*Suc n*)
have ‹*n* $\leq$ *N*› **by** (*simp add*: *Suc.prems Suc-leD*)
define σs σt **where** *a1*: ‹σs $\equiv$ *replicate n* (*1*::*nat*) @ *replicate* (*N* − *n*) *0*›
and *a2*: ‹σt $\equiv$ *replicate* (*Suc n*) (*1*::*nat*) @ *replicate* (*N* − *Suc n*) *0*›
have ‹*length* σs = *N*› ‹*length* σt = *N*› ‹*length* σs = *length* σt›
by (*simp-all add*: ‹*n* $\leq$ *N*› *a1 a2 Suc.prems*)
have *f1*: ‹(σs, σs) $\in \mathcal{R}_d$ A_{RD} (*replicate N 0*, *replicate N 0*)›
using *Suc.hyps*(*1*) *a1 Suc.prems Suc-leD* **by** *presburger*
have *f2*: ‹*picks n n* $\in$ ε A_{RP} σs› ‹*picks n n* $\in$ ε A_F σs› ‹*picks n n* $\in$ ε A_{RD} (σs, σs)›
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 ‹*a* $\in$ ε (A_{rp} *i*) (σs ! *i*) $\Longrightarrow$ *i* < *N* $\Longrightarrow$ *a* $\in$ ε (A_{rp} *j*) (σs ! *j*) $\Longrightarrow$ *j* < *N* $\Longrightarrow$ *j* = *i*› **for** *i j a*
by (*metis rphil-phil*)
hence $*$: ‹*a* $\in$ ε (A_{rp} *i*) (σs ! *i*) $\Longrightarrow$ *i* < *N* $\Longrightarrow$
(*THE j*. *j* < *N* $\wedge$ *a* $\in$ ε (*map* A_{rp} [*0*..<*N*] ! *j*) (σs ! *j*)) = *i*› **for** *i a* **by**

auto

have ‹$a \in \varepsilon\ (A_f\ i)\ (\sigma s\ !\ i) \Longrightarrow i < N \Longrightarrow a \in \varepsilon\ (A_f\ j)\ (\sigma s\ !\ j) \Longrightarrow j < N \Longrightarrow j = i$› **for** *i j a*
by (*metis ev-id*$_{fork}$*x*)
hence $**$: ‹$a \in \varepsilon\ (A_f\ i)\ (\sigma s\ !\ i) \Longrightarrow i < N \Longrightarrow$
$(THE\ j.\ j < N \wedge a \in \varepsilon\ (map\ A_f\ [0..<N]\ !\ j)\ (\sigma s\ !\ j)) = i$› **for** *i a* **by** *auto*

have *f3*: ‹$\sigma t = \lceil \tau\ A_{RP}\ \sigma s\ (picks\ n\ n)\rceil$›
apply (*unfold rphils-A-def*, *subst* τ*-iterated-combine*$_d$*-Sync-general-form-when-indep*)
subgoal by (*simp add*: ‹*length* $\sigma s = N$›)
subgoal by (*simp add*: *indep-enablI 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*: ‹*length* σs = *length* σt› *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 ‹*length* $\sigma s = N$› *atLeast0LessThan f2*(*1*) **by** *force*

have *f4*: ‹$\sigma t = \lceil \tau\ A_F\ \sigma s\ (picks\ n\ n)\rceil$›
apply (*unfold forks-A-def*, *subst* τ*-iterated-combine*$_d$*-Sync-general-form-when-indep*)
subgoal by (*simp add*: ‹*length* $\sigma s = N$›)
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*: ‹*length* σs = *length* σt› *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 ‹*length* $\sigma s = N$› *atLeast0LessThan f2*(*2*) **by** *force*
show ‹$(\sigma t, \sigma t) \in \mathcal{R}_d\ A_{RD}\ (replicate\ N\ 0,\ replicate\ N\ 0)$›
apply (*rule* $\mathcal{R}_d$*.step*[*OF f1*], *subst rdining-*τ)
using *Suc.prems a1* **apply** *fastforce*
apply (*rule f2*)
using *f3 f4* **by** *blast*
qed

corollary *replicate1-reachable-from-replicate0*: ‹$(replicate\ N\ 1,\ replicate\ N\ 1) \in \mathcal{R}_d\ A_{RD}\ (replicate\ N\ 0,\ replicate\ N\ 0)$›
by (*simp add*: *replicate1-reachable-from-replicate0-prelim*[*of N*, *simplified*])

theorem *not-deadlock-free-RDINING*: ‹¬ *deadlock-free RDINING*›
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* :
‹$(\Box a \in A \to P\ a) \Box (\Box b \in B \to Q\ b) =$
$(\Box x \in (A \cap B) \to P\ x \sqcap Q\ x) \Box (\Box a \in (A - B) \to P\ a) \Box (\Box b \in (B - A) \to Q\ b)$›
(**is** ‹*?lhs* = *?rhs*›)
proof (*subst Process-eq-spec*, *safe*)
show ‹$(t, X) \in \mathcal{F}$ *?lhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?rhs*› **for** $t\ X$
by (*cases t*) (*auto simp add*: *F-Det F-Mprefix F-Ndet*)
next
show ‹$(t, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?lhs*› **for** $t\ X$
by (*cases t*, *simp-all add*: *F-Mprefix F-Ndet F-Det D-Det T-Det disjoint-iff*) *blast+*
next
show ‹$t \in \mathcal{D}$ *?lhs* $\Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** t
by (*auto simp add*: *D-Det D-Mprefix image-iff D-Ndet*)
next
show ‹$t \in \mathcal{D}$ *?rhs* $\Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** t
by (*auto simp add*: *D-Mprefix D-Ndet D-Det split*: *if-split-asm*)
qed

lemma *GlobalNdet-Ndet-GlobalNdet*:
‹$A \neq \{\} \Longrightarrow B \neq \{\} \Longrightarrow (\sqcap a \in A.\ P\ a) \sqcap (\sqcap b \in B.\ Q\ b) =$
$\sqcap x \in (A \cup B).$ (*if* $x \in A \cap B$ *then* $P\ x \sqcap Q\ x$ *else if* $x \in A$ *then* $P\ x$ *else* $Q\ x$)›
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*:
‹$A \cap B \neq \{\} \Longrightarrow A - B \neq \{\} \Longrightarrow B - A \neq \{\} \Longrightarrow$
$(\sqcap a \in A.\ P\ a) \sqcap (\sqcap b \in B.\ Q\ b) =$
$(\sqcap x \in (A \cap B).\ P\ x \sqcap Q\ x) \sqcap (\sqcap a \in (A - B).\ P\ a) \sqcap (\sqcap b \in (B - A).\ Q\ b)$›
by (*auto simp add*: *Process-eq-spec F-Ndet D-Ndet F-GlobalNdet D-GlobalNdet*)

lemma *GlobalNdet-GlobalNdet*:
‹$(\sqcap a \in A.\ \sqcap b \in B\ a.\ P\ b) =$
(*if* $\forall a \in A.\ B\ a \neq \{\}$ *then* $\sqcap b \in (\bigcup a \in A.\ B\ a).\ P\ b$ *else* $(\sqcap b \in (\bigcup a \in A.\ B\ a).\ P\ b) \sqcap STOP$)›
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* :
‹$(\Box\ (s,\ A) \in s\text{-}A\text{-}set.\ P\langle\!\langle A\rangle\!\rangle_{nd}\ s) =$
$\Box e \in (\bigcup (s,\ A) \in s\text{-}A\text{-}set.\ \varepsilon\ A\ s) \rightarrow$
$\sqcap (s,\ A) \in \{(s,\ A) \in s\text{-}A\text{-}set.\ e \in \varepsilon\ A\ s\}.$
$\sqcap s' \in \tau\ A\ s\ e.\ P\langle\!\langle A\rangle\!\rangle_{nd}\ s'$› (**is** ‹*?lhs* = *?rhs*›)
proof −
have ‹*?lhs* = $(\Box\ (s,\ A) \in s\text{-}A\text{-}set.\ P\text{-}nd\text{-}step\ (\varepsilon\ A)\ (\tau\ A)\ P\langle\!\langle A\rangle\!\rangle_{nd}\ s)$›
by (*auto intro*: *mono-GlobalDet-eq arg-cong*[*OF P-nd-rec*])
also have ‹$\ldots = \Box s\text{-}A \in s\text{-}A\text{-}set.\ P\text{-}nd\text{-}step\ (\varepsilon\ (snd\ s\text{-}A))\ (\tau\ (snd\ s\text{-}A))\ P\langle\!\langle snd\ s\text{-}A\rangle\!\rangle_{nd}\ (fst\ s\text{-}A)$›
by (*simp only*: *case-prod-beta′*)
also have ‹$\ldots = \Box e \in (\bigcup s\text{-}A \in s\text{-}A\text{-}set.\ \varepsilon\ (snd\ s\text{-}A)\ (fst\ s\text{-}A)) \rightarrow$
$\sqcap s\text{-}A \in \{s\text{-}A \in s\text{-}A\text{-}set.\ e \in \varepsilon\ (snd\ s\text{-}A)\ (fst\ s\text{-}A)\}.$
$GlobalNdet\ (\tau\ (snd\ s\text{-}A)\ (fst\ s\text{-}A)\ e)\ P\langle\!\langle snd\ s\text{-}A\rangle\!\rangle_{nd}$›
by (*simp add*: *GlobalDet-Mprefix*)
also have ‹$\ldots$ = *?rhs*› **by** (*simp add*: *case-prod-beta′*)
finally show ‹*?lhs* = *?rhs*› **.**
qed

lemma *P-nd-set-almost-compactification-Det-bis* :
‹$(\Box\ (s,\ A) \in s\text{-}A\text{-}set.\ P\langle\!\langle A\rangle\!\rangle_{nd}\ s) =$
$\Box e \in (\bigcup (s,\ A) \in s\text{-}A\text{-}set.\ \varepsilon\ A\ s) \rightarrow$
$\sqcap (s',\ A) \in \{(s',\ A) |\ s'\ s\ A.\ (s,\ A) \in s\text{-}A\text{-}set \wedge e \in \varepsilon\ A\ s \wedge s' \in \tau\ A\ s\ e\}.\ P\langle\!\langle A\rangle\!\rangle_{nd}\ s'$›
(**is** ‹- = *?rhs*›)
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 ‹$(\Box\ (s,\ A) \in$ *s-A-set*. $P\langle\!\langle A\rangle\!\rangle_d\ s) =$
$\Box e \in (\bigcup (s,\ A) \in$ *s-A-set*. $\varepsilon\ A\ s) \rightarrow$
$\sqcap (s,\ A) \in \{(s,\ A) \in$ *s-A-set*. $e \in \varepsilon\ A\ s\}$. $P\langle\!\langle A\rangle\!\rangle_d\ \lceil\tau\ A\ s\ e\rceil$› (**is** ‹*?lhs* = *?rhs*›)
proof −
have ‹*?lhs* = $(\Box\ (s,\ A) \in$ *s-A-set*. *P-d-step* $(\varepsilon\ A)\ (\tau\ A)\ P\langle\!\langle A\rangle\!\rangle_d\ s)$›
by (*auto intro*: *mono-GlobalDet-eq arg-cong*[*OF P-d-rec*])
also have ‹... = $\Box$*s-A* $\in$ *s-A-set*. *P-d-step* (ε (*snd s-A*)) (τ (*snd s-A*)) $P\langle\!\langle$*snd s-A*$\rangle\!\rangle_d$ (*fst s-A*)›
by (*simp only*: *case-prod-beta′*)
also have ‹... = $\Box e \in (\bigcup$ *s-A* $\in$ *s-A-set*. ε (*snd s-A*) (*fst s-A*)) $\rightarrow$
$\sqcap$*s-A* $\in \{$*s-A* $\in$ *s-A-set*. $e \in \varepsilon$ (*snd s-A*) (*fst s-A*)$\}$.
$P\langle\!\langle$*snd s-A*$\rangle\!\rangle_d\ \lceil\tau$ (*snd s-A*) (*fst s-A*) $e\rceil$›
by (*simp add*: *GlobalDet-Mprefix*)
also have ‹... = *?rhs*› **by** (*simp add*: *case-prod-beta′*)
finally show ‹*?lhs* = *?rhs*› **.**
qed

lemma *P-d-set-almost-compactification-Det-bis*:
shows ‹$(\Box\ (s,\ A) \in$ *s-A-set*. $P\langle\!\langle A\rangle\!\rangle_d\ s) =$
$\Box e \in (\bigcup (s,\ A) \in$ *s-A-set*. $\varepsilon\ A\ s) \rightarrow$
$\sqcap (s',\ A) \in \{(\lceil\tau\ A\ s\ e\rceil,\ A) |\ s\ A.\ (s,\ A) \in$ *s-A-set* $\wedge\ e \in \varepsilon\ A\ s\}$. $P\langle\!\langle A\rangle\!\rangle_d\ s'$›
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_{SKIPS}*-nd* :
‹$(P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma)^0 =$ (*if* $\sigma \in \varrho\ A$ *then tick* ‘ $\omega\ A\ \sigma$ *else ev* ‘ $\varepsilon\ A\ \sigma$)›
by (*subst* P_{SKIPS}*-nd-rec*) (*simp add*: *initials-Mprefix* ϱ*-simps*)

lemma *initials-*P_{SKIPS}*-d* :
‹$(P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma)^0 =$ (*if* $\sigma \in \varrho\ A$ *then* $\{\checkmark(\lceil\omega\ A\ \sigma\rceil)\}$ *else ev* ‘ $\varepsilon\ A\ \sigma$)›
by (*subst* P_{SKIPS}*-d-rec*) (*auto simp add*: *initials-Mprefix* ϱ*-simps*)

lemma *initials-P-nd* : ‹$(P\langle\!\langle A\rangle\!\rangle_{nd}\ s)^0 =$ *ev* ‘ $\varepsilon\ A\ s$›
by (*subst P-nd-rec*) (*simp-all add*: *initials-Mprefix*)

lemma *initials-P-d* : ‹$(P\langle\!\langle A\rangle\!\rangle_d\ s)^0 =$ *ev* ‘ $\varepsilon\ A\ s$›

by (*subst P-d-rec*) (*simp add*: *initials-Mprefix*)

12.4.2 *Throw*

lemma *Throw-*P_{SKIPS}*-nd* :
‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ Θ $b \in B.$ Q b =
(*if* $\sigma \in \varrho$ A *then SKIPS* (ω A σ) *else*
$\Box a{\in}\varepsilon$ A $\sigma \rightarrow$ (*if* $a \in B$ *then* Q a *else* $\sqcap\sigma' \in \tau$ A σ $a.$ ($P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ' Θ b $\in B.$ Q b)))›
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-*P_{SKIPS}*-d* :
‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{d}$ σ Θ $b \in B.$ Q b =
(*if* $\sigma \in \varrho$ A *then SKIP* $\lceil\omega$ A $\sigma\rceil$ *else*
$\Box a{\in}\varepsilon$ A $\sigma \rightarrow$ (*if* $a \in B$ *then* Q a *else* $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{d}$ $\lceil\tau$ A σ $a\rceil$ Θ $b \in B.$ Q b))›
by (*simp add*: P_{SKIPS}*-d-rec-in-*ϱ P_{SKIPS}*-d-rec-notin-*ϱ *Throw-Mprefix*)

lemma *Throw-P-nd* :
‹$P\langle\!\langle A\rangle\!\rangle_{nd}$ σ Θ $b \in B.$ Q b =
$\Box a{\in}\varepsilon$ A $\sigma \rightarrow$ (*if* $a \in B$ *then* Q a *else* $\sqcap\sigma' \in \tau$ A σ $a.$ ($P\langle\!\langle A\rangle\!\rangle_{nd}$ σ' Θ $b \in B.$ Q b))›
by (*subst P-nd-rec*)
(*auto simp add*: *Throw-Mprefix Throw-distrib-GlobalNdet-right intro*: *mono-Mprefix-eq*)

lemma *Throw-P-d* :
‹$P\langle\!\langle A\rangle\!\rangle_{d}$ σ Θ $b \in B.$ Q b =
$\Box a{\in}\varepsilon$ A $\sigma \rightarrow$ (*if* $a \in B$ *then* Q a *else* $P\langle\!\langle A\rangle\!\rangle_{d}$ $\lceil\tau$ A σ $a\rceil$ Θ $b \in B.$ Q b)›
by (*subst P-d-rec*) (*simp add*: *Throw-Mprefix*)

12.4.3 ($\triangle$)

lemma *SKIPS-Interrupt-is-SKIPS-Det* :
‹*SKIPS* R $\triangle$ P = *SKIPS* R $\Box$ P›
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-*P_{SKIPS}*-nd* :
‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ $\triangle$ Q =
Q $\Box$ (*if* $\sigma \in \varrho$ A *then SKIPS* (ω A σ) *else* $\Box a \in \varepsilon$ A $\sigma \rightarrow \sqcap\sigma' \in \tau$ A σ $a.$ $P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}$ σ' $\triangle$ Q)›
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 = ‹($\Box$)›] *mono-Mprefix-eq split*: *if-split-asm*)

lemma *Interrupt-*P_{SKIPS}*-d* :
‹$P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{d}$ σ $\triangle$ Q =

$Q \,\Box\, (\textit{if } \sigma \in \varrho\ A \textit{ then SKIP } \lceil \omega\ A\ \sigma \rceil \textit{ else } \Box a \in \varepsilon\ A\ \sigma \rightarrow P_{SKIPS}\langle\!\langle A \rangle\!\rangle_d\ \lceil \tau\ A\ \sigma\ a \rceil \triangle Q)$›
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* :
‹$P\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma \triangle Q = Q \,\Box\, (\Box a \in \varepsilon\ A\ \sigma \rightarrow \sqcap \sigma' \in \tau\ A\ \sigma\ a.\ P\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma' \triangle Q)$›
by (*subst P-nd-rec*)
(*auto simp add*: *Interrupt-Mprefix Interrupt-distrib-GlobalNdet-right* ε*-simps*
intro!: *arg-cong2*[**where** f = ‹($\Box$)›] *mono-Mprefix-eq split*: *if-split-asm*)

lemma *Interrupt-P-d* :
‹$P\langle\!\langle A \rangle\!\rangle_d\ \sigma \triangle Q = Q \,\Box\, (\Box a \in \varepsilon\ A\ \sigma \rightarrow P\langle\!\langle A \rangle\!\rangle_d\ \lceil \tau\ A\ \sigma\ a \rceil \triangle Q)$›
by (*metis Interrupt-Mprefix P-d-rec*)

12.4.4 After

context *After*
begin

lemma *After-SKIPS* : ‹*SKIPS R after a* = Ψ (*SKIPS R*) *a*›
by (*simp add*: *Process-eq-spec After-projs image-iff*)

lemma *After-*P_{SKIPS}*-nd* :
‹$P_{SKIPS}\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma$ *after a* =
(*if* $\sigma \in \varrho\ A$ *then* Ψ (*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* Ψ ($P_{SKIPS}\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma$) *a*)›
by (*subst* (*1 4*) P_{SKIPS}*-nd-rec*)
(*simp add*: ε*-simps* ϱ*-simps After-SKIPS After-Mprefix*)

lemma *After-*P_{SKIPS}*-d* :
‹$P_{SKIPS}\langle\!\langle A \rangle\!\rangle_d\ \sigma$ *after a* =
(*if* $\sigma \in \varrho\ A$ *then* Ψ (*SKIP* $\lceil \omega\ A\ \sigma \rceil$) *a else*
if $a \in \varepsilon\ A\ \sigma$ *then* $P_{SKIPS}\langle\!\langle A \rangle\!\rangle_d\ \lceil \tau\ A\ \sigma\ a \rceil$ *else* Ψ ($P_{SKIPS}\langle\!\langle A \rangle\!\rangle_d\ \sigma$) *a*)›
by (*subst* (*1 2*) P_{SKIPS}*-d-rec*)
(*auto simp add*: ε*-simps* ϱ*-simps After-SKIP After-Mprefix*)

lemma *After-P-nd* :
‹$P\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma$ *after a* = (*if* $a \in \varepsilon\ A\ \sigma$ *then* $\sqcap \sigma' \in \tau\ A\ \sigma\ a.\ P\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma'$ *else* Ψ ($P\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma$) *a*)›
by (*subst* (*1 4*) *P-nd-rec*) (*simp add*: ε*-simps After-Mprefix*)

lemma *After-P-d* :
‹$P\langle\!\langle A \rangle\!\rangle_d\ \sigma$ *after a* = (*if* $a \in \varepsilon\ A\ \sigma$ *then* $P\langle\!\langle A \rangle\!\rangle_d\ \lceil \tau\ A\ \sigma\ a \rceil$ *else* Ψ ($P\langle\!\langle A \rangle\!\rangle_d\ \sigma$) *a*)›
by (*subst* (*1 2*) *P-d-rec*)
(*simp add*: ε*-simps After-SKIP After-Mprefix*)

end

context *AfterExt*
begin

lemma $After_{tick}$-*SKIPS* :
‹*SKIPS R after*$_{\checkmark}$ *e* = (*case e of ev a* ⇒ Ψ (*SKIPS R*) *a* | ✓(*r*) ⇒ Ω (*SKIPS R*) *r*)›
by (*cases e*) (*simp-all add*: $After_{tick}$-*def After-SKIPS*)

lemma $After_{tick}$-P_{SKIPS}-*nd* :
‹P_{SKIPS}⟪*A*⟫$_{nd}$ σ *after*$_{\checkmark}$ *e* =
(*case e of ev a* ⇒ *if* σ ∈ ϱ *A then* Ψ (*SKIPS* (ω *A* σ)) *a else*
if a ∈ ε *A* σ *then* ⊓σ′ ∈ τ *A* σ *a*. P_{SKIPS}⟪*A*⟫$_{nd}$ σ′ *else* Ψ (P_{SKIPS}⟪*A*⟫$_{nd}$ σ) *a*
| ✓(*r*) ⇒ *if* σ ∈ ϱ *A then* Ω (*SKIPS* (ω *A* σ)) *r else* Ω (P_{SKIPS}⟪*A*⟫$_{nd}$ σ) *r*)›
proof (*cases e*)
show ‹*e* = *ev a* ⟹ *?thesis*› **for** *a*
by (*simp add*: *After*-P_{SKIPS}-*nd* $After_{tick}$-*def*)
next
show ‹*e* = ✓(*r*) ⟹ *?thesis*› **for** *r*
by (*subst* (*1 5*) P_{SKIPS}-*nd-rec*) (*simp add*: $After_{tick}$-*def* ϱ-*simps*)
qed

lemma $After_{tick}$-P_{SKIPS}-*d* :
‹P_{SKIPS}⟪*A*⟫$_{d}$ σ *after*$_{\checkmark}$ *e* =
(*case e of ev a* ⇒ *if* σ ∈ ϱ *A then* Ψ (*SKIP* ⌈ω *A* σ⌉) *a else*
if a ∈ ε *A* σ *then* P_{SKIPS}⟪*A*⟫$_{d}$ ⌈τ *A* σ *a*⌉ *else* Ψ (P_{SKIPS}⟪*A*⟫$_{d}$ σ) *a*
| ✓(*r*) ⇒ *if* σ ∈ ϱ *A then* Ω (*SKIP* ⌈ω *A* σ⌉) *r else* Ω (P_{SKIPS}⟪*A*⟫$_{d}$ σ) *r*)›
proof (*cases e*)
show ‹*e* = *ev a* ⟹ *?thesis*› **for** *a*
by (*simp add*: *After*-P_{SKIPS}-*d* $After_{tick}$-*def*)
next
show ‹*e* = ✓(*r*) ⟹ *?thesis*› **for** *r*
by (*subst* (*1 2*) P_{SKIPS}-*d-rec*) (*auto simp add*: $After_{tick}$-*def* ϱ-*simps*)
qed

lemma $After_{tick}$-*P-nd* :
‹*P*⟪*A*⟫$_{nd}$ σ *after*$_{\checkmark}$ *e* =
(*case e of ev a* ⇒ *if a* ∈ ε *A* σ *then* ⊓σ′ ∈ τ *A* σ *a*. *P*⟪*A*⟫$_{nd}$ σ′ *else* Ψ (*P*⟪*A*⟫$_{nd}$ σ) *a*
| ✓(*r*) ⇒ Ω (*P*⟪*A*⟫$_{nd}$ σ) *r*)›
proof (*cases e*)
show ‹*e* = *ev a* ⟹ *?thesis*› **for** *a*
by (*simp add*: *After-P-nd* $After_{tick}$-*def*)
next

show ‹$e = \checkmark(r) \Longrightarrow$ *?thesis*› **for** r
by (*subst* (*1 2*) *P-nd-rec*) (*auto simp add*: $After_{tick}$*-def*)
qed

lemma $After_{tick}$*-P-d* :
‹$P\langle\!\langle A\rangle\!\rangle_d\ \sigma\ after_{\checkmark}\ e =$
(*case e of ev a* $\Rightarrow$ *if* $a \in \varepsilon\ A\ \sigma$ *then* $P\langle\!\langle A\rangle\!\rangle_d\ \lceil\tau\ A\ \sigma\ a\rceil$ *else* $\Psi\ (P\langle\!\langle A\rangle\!\rangle_d\ \sigma)\ a$
$\mid \checkmark(r) \Rightarrow \Omega\ (P\langle\!\langle A\rangle\!\rangle_d\ \sigma)\ r$)›
proof (*cases e*)
show ‹$e = ev\ a \Longrightarrow$ *?thesis*› **for** a
by (*simp add*: *After-P-d* $After_{tick}$*-def*)
next
show ‹$e = \checkmark(r) \Longrightarrow$ *?thesis*› **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* : ‹$r \in R \Longrightarrow SKIPS\ R \rightsquigarrow_\tau SKIP\ r$›
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* :
‹$r \in \omega\ A\ \sigma \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma \rightsquigarrow_\tau SKIP\ r$›
by (*subst* P_{SKIPS}*-nd-rec*) (*auto simp add*: *SKIPS-*τ*-trans-SKIP*)

lemma τ*-trans-*P_{SKIPS}*-d* :
‹$\sigma \in \varrho\ A \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma \rightsquigarrow_{\checkmark\lceil\omega\ A\ \sigma\rceil}\ \Omega\ (SKIP\ \lceil\omega\ A\ \sigma\rceil)\ \lceil\omega\ A\ \sigma\rceil$›
by (*auto simp add*: P_{SKIPS}*-d-rec SKIP-trans-tick-*Ω*-SKIP* ϱ*-simps*)

lemma *ev-trans-*P_{SKIPS}*-nd* :
‹$\sigma \notin \varrho\ A \Longrightarrow \sigma' \in \tau\ A\ \sigma\ a \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma \rightsquigarrow_a P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma'$›
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* :
‹$\sigma \notin \varrho\ A \Longrightarrow a \in \varepsilon\ A\ \sigma \Longrightarrow P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \sigma \rightsquigarrow_a P_{SKIPS}\langle\!\langle A\rangle\!\rangle_d\ \lceil\tau\ A\ \sigma\ a\rceil$›
by (*subst* P_{SKIPS}*-d-rec*)

(*auto simp add*: *ϱ-simps intro*: *ev-trans-Mprefix*)

lemma *ev-trans-P-nd* :
‹$\sigma' \in \tau\ A\ \sigma\ a \Longrightarrow P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma \rightsquigarrow_a P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma'$›
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* :
‹$a \in \varepsilon\ A\ \sigma \Longrightarrow P\langle\!\langle A\rangle\!\rangle_{d}\ \sigma \rightsquigarrow_a P\langle\!\langle A\rangle\!\rangle_{d}\ \lceil\tau\ A\ \sigma\ a\rceil$›
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:

$$\frac{\textit{deadlock-free}\ (P\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma) = (\forall\,\sigma'\!\in\!\mathcal{R}_{nd}\ A\ \sigma.\ \varepsilon\ A\ \sigma' \neq \emptyset) \qquad \varrho\text{-}\textit{disjoint-}\varepsilon\ A}{\textit{deadlock-free}_{SKIPS}\ (P_{SKIPS}\langle\!\langle A\rangle\!\rangle_{nd}\ \sigma) = (\forall\,\sigma'\!\in\!\mathcal{R}_{nd}\ A\ \sigma.\ \sigma' \in \varrho\ A \vee \varepsilon\ A\ \sigma' \neq \emptyset)}$$

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 $P\ [\![A]\!]\ Q$ to $P\ [\![A]\!]_{\checkmark}\ Q$.

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 $\upsilon\ x.\ f\ x$ in all cases.

The resulting framework yields compactification theorems that support invariant-based reasoning over large process architectures:

$$\frac{|\sigma s| = |As|}{[\![E]\!]\ (\sigma,\ A) \in \#mset\ (zip\ \sigma s\ As).\ P\langle\!\langle A \rangle\!\rangle_{nd}\ \sigma = P\langle\!\langle\!\langle_{nd} \bigotimes [\![E]\!]\ As \rangle\!\rangle\!\rangle_{nd}\ \sigma s}$$

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 ICTAC 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.