

# HOL-CSP\_Proc-Omata: A Bridge between CSP Processes and Functional Automata

Benoît Ballenghien      Burkhart Wolff

February 3, 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, enable-ness), 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

<b>1</b>	<b>Introduction</b>	<b>9</b>
<b>2</b>	<b>An Excursion into Determinism</b>	<b>11</b>
2.1	Accepts initials . . . . .	11
2.1.1	Definition . . . . .	11
2.1.2	First properties . . . . .	12
2.1.3	Monotonicity . . . . .	12
2.1.4	Behaviour on Operators . . . . .	12
2.1.5	Characterizations with After . . . . .	14
2.2	Deterministic process . . . . .	16
2.2.1	Definition . . . . .	16
2.2.2	Monotonicity . . . . .	16
2.2.3	Characterization as Maximal . . . . .	16
2.2.4	Characterization with After . . . . .	19
2.2.5	Operators preserving Determinism . . . . .	19
2.2.6	Operators not (always) preserving Determinism . . . . .	20
2.3	Application to Operational Semantics . . . . .	20
<b>3</b>	<b>ProcOmata: Functional Automata embedded into CSP Processes</b>	<b>23</b>
3.1	Definitions . . . . .	24
3.1.1	Non-deterministic and deterministic Automata . . . . .	24
3.1.2	Enableness . . . . .	24
3.1.3	States allowing Termination . . . . .	25
3.1.4	Reachability . . . . .	25
3.1.5	Morphisms . . . . .	25
3.1.6	Generic update Functions . . . . .	27
3.1.7	Assumptions on Automata . . . . .	28
3.2	First Properties . . . . .	31
3.2.1	$\varepsilon$ , $\rho$ and $\omega$ first equalities . . . . .	31
3.2.2	Properties of our morphisms . . . . .	32
3.2.3	Reachability results (for $\mathcal{R}_d$ and $\mathcal{R}_{nd}$ ) . . . . .	36
3.3	Normalization . . . . .	38

3.3.1	Non-deterministic Case . . . . .	38
3.3.2	Deterministic Case . . . . .	41
3.3.3	Link between deterministic and non-deterministic ProcOmata . . . . .	43
3.3.4	Prove Equality between ProcOmata . . . . .	43
<b>4</b>	<b>Advanced Properties of ProcOmata</b>	<b>47</b>
<b>5</b>	<b>Combining Automata for Synchronization Product</b>	<b>49</b>
5.1	Definitions . . . . .	49
5.1.1	General Patterns . . . . .	49
5.1.2	Specializations . . . . .	50
5.2	First Properties . . . . .	52
5.3	Reachability . . . . .	53
5.4	Normalization . . . . .	54
<b>6</b>	<b>Compactification of Synchronization Product</b>	<b>57</b>
6.1	Iterated Combine . . . . .	57
6.1.1	Definitions . . . . .	57
6.1.2	First Results . . . . .	58
6.1.3	Reachability . . . . .	59
6.1.4	Transmission of Properties . . . . .	59
6.1.5	Normalization . . . . .	60
6.2	Compactification Theorems . . . . .	60
6.2.1	Binary . . . . .	60
6.2.2	Rlist . . . . .	62
6.2.3	ListslenL . . . . .	62
6.2.4	Multiple . . . . .	63
6.3	Derived Versions . . . . .	64
6.4	More on Iterated Combine . . . . .	65
6.5	More on Events . . . . .	66
<b>7</b>	<b>Combining Automata for Generalized Synchronization Product</b>	<b>69</b>
7.1	Definitions . . . . .	69
7.1.1	Specializations . . . . .	69
7.2	First Properties . . . . .	70
7.3	Transitions are unchanged in the Generalization . . . . .	72
7.4	Reachability . . . . .	72
7.5	Normalization . . . . .	73
<b>8</b>	<b>Compactification of Synchronization Product Generalized</b>	<b>77</b>
8.1	Iterated Combine . . . . .	77
8.1.1	Definitions . . . . .	77

8.1.2	First Results . . . . .	77
8.1.3	Transmission of Properties . . . . .	78
8.1.4	Normalization . . . . .	79
8.2	Compactification Theorems . . . . .	80
8.2.1	Binary . . . . .	80
8.2.2	Rlist . . . . .	81
8.2.3	ListslenL . . . . .	81
8.2.4	Multiple . . . . .	82
8.3	Derived Versions . . . . .	83
8.4	More on Iterated Combine and Events . . . . .	83
<b>9</b>	<b>Combining Automata for Sequential Composition Generalized</b>	<b>85</b>
9.1	Definitions . . . . .	85
9.2	General Patterns . . . . .	85
9.3	Specializations . . . . .	86
9.4	First Properties . . . . .	87
9.4.1	Reachability . . . . .	88
9.5	Normalization . . . . .	89
<b>10</b>	<b>Compactification of Sequential Composition Generalized</b>	<b>91</b>
10.1	Iterated Combine . . . . .	91
10.1.1	Definitions . . . . .	91
10.1.2	First Results . . . . .	91
10.1.3	Reachability . . . . .	92
10.1.4	Transmission of Properties . . . . .	92
10.1.5	Normalization . . . . .	93
10.2	Compactification Theorems . . . . .	93
10.2.1	Binary . . . . .	93
10.2.2	ListslenL . . . . .	94
10.2.3	Multiple . . . . .	95
<b>11</b>	<b>Application : May Philosophers dine ?</b>	<b>97</b>
11.1	Preliminaries . . . . .	97
11.1.1	Preliminary lemmas for proof automation . . . . .	97
11.2	The dining processes definition . . . . .	97
11.2.1	Unfolding rules . . . . .	98
11.3	Translation into normal form . . . . .	98
11.3.1	<i>FORK</i> , <i>LPHILO</i> and <i>RPHIL</i> are normalizable . . . . .	98
11.3.2	<i>FORKS</i> is normalizable . . . . .	100
11.3.3	<i>PHILS</i> is normalizable . . . . .	100
11.3.4	The complete process <i>DINING</i> is normalizable . . . . .	101
11.4	And finally: Philosophers may dine ! Always ! . . . . .	101
11.4.1	Construction of an invariant for the dining automaton . . . . .	101

11.4.2	The invariant <i>inv-dining</i> implies that <i>DINING</i> is <i>deadlock-free</i> . . . . .	102
11.4.3	Conclusion . . . . .	102
11.5	Alternative version with only right-handed philosophers (in order to show that it's not <i>deadlock-free</i> ) . . . . .	102
11.5.1	Setup . . . . .	102
11.5.2	Normalization . . . . .	103
11.5.3	Correspondance between our normalized processes and the previous definitions . . . . .	103
11.5.4	Proof that we have a deadlock in the state ( <i>replicate N 1, replicate N 1</i> ) . . . . .	103
11.5.5	Proof that this state is reachable from our initial state, i.e. ( <i>replicate N 1, replicate N 1</i> ) $\in \mathcal{R}_d$ <i>ARD</i> ( <i>replicate N 0, replicate N 0</i> ) . . . . .	103
<b>12</b>	<b>Other Results similar to Compactification</b>	<b>105</b>
12.1	Some preliminary Results . . . . .	105
12.2	Results for <i>Det</i> . . . . .	106
12.3	Results for <i>Ndet</i> . . . . .	106
12.4	Other Operators . . . . .	106
12.4.1	<i>initials</i> . . . . .	106
12.4.2	<i>Throw</i> . . . . .	107
12.4.3	( $\Delta$ ) . . . . .	107
12.4.4	<i>After</i> . . . . .	108
12.5	<i>OpSem</i> . . . . .	109
<b>13</b>	<b>Conclusion</b>	<b>111</b>
13.1	Entry Point . . . . .	111
13.2	Conclusion . . . . .	111

# Chapter 1

## Introduction

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

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

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



## Chapter 2

# An Excursion into Determinism

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

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

### 2.1 Accepts initials

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

#### 2.1.1 Definition

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

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

**lemma** *accepts-initialsI* :  $\langle (\bigwedge e. e \in P^0 \Longrightarrow \{e\} \notin \mathcal{R} P) \Longrightarrow \text{determ}^0 P \rangle$   
**and** *accepts-initialsD* :  $\langle \text{determ}^0 P \Longrightarrow e \in P^0 \Longrightarrow \{e\} \notin \mathcal{R} P \rangle$   
*<proof>*

**lemma** *accepts-initials-def-bis*:

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

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

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

### 2.1.2 First properties

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

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

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

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

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

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

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

### 2.1.3 Monotonicity

**lemma** *mono-accepts-initials-F*:  $\langle P \sqsubseteq_F Q \implies \text{determ}^0 P \implies \text{determ}^0 Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-accepts-initials-FD*:  $\langle P \sqsubseteq_{FD} Q \implies \text{determ}^0 P \implies \text{determ}^0 Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-accepts-initials*:  $\langle P \sqsubseteq Q \implies \text{determ}^0 P \implies \text{determ}^0 Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *restriction-adm-accepts-initials* [*restriction-adm-process<sub>ptick</sub>-simpset, simp*]

:

$\langle \text{adm}_\downarrow (\lambda x. \text{determ}^0 (f x)) \rangle$  **if**  $\langle \text{cont}_\downarrow f \rangle$

**for**  $f :: \langle 'b :: \text{restriction} \Rightarrow ('a, 'r) \text{process}_{ptick} \rangle$

$\langle \text{proof} \rangle$

### 2.1.4 Behaviour on Operators

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

$\langle \text{proof} \rangle$

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

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

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

**lemma** *accepts-initials-Ndet-iff*:  
 $\langle \text{determ}^0 (P \sqcap Q) \longleftrightarrow \text{determ}^0 P \wedge \text{determ}^0 Q \wedge P^0 = Q^0 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-GlobalNdet-iff*:  
 $\langle \text{determ}^0 (\sqcap a \in A. P a) \longleftrightarrow$   
 $(\forall a \in A. \text{determ}^0 (P a) \wedge (\forall b \in A. (P a)^0 = (P b)^0)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-Mndetprefix-iff*:  
 $\langle \text{determ}^0 (\sqcap a \in A \rightarrow P a) \longleftrightarrow (\exists a. A \subseteq \{a\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-ndet-write-iff*:  
 $\langle \text{determ}^0 (c!a \in A \rightarrow P a) \longleftrightarrow (\exists b. c \in A \subseteq \{b\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-SKIPS-iff* :  
 $\langle \text{determ}^0 (\text{SKIPS } R) \longleftrightarrow R = \{\} \vee (\exists r. R = \{r\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-Det* :  
 $\langle \text{determ}^0 (P \sqcap Q) \longleftrightarrow P = \text{STOP} \vee Q = \text{STOP} \vee \text{range tick} \cap P^0 \cap Q^0 \neq \{\}$   
 $\vee$   
 $\text{range tick} \cap (P^0 \cup Q^0) = \{\} \rangle$   
**(is  $\langle - \longleftrightarrow ?rhs \rangle$  if *accepts-initials* :  $\langle \text{determ}^0 P \rangle \langle \text{determ}^0 Q \rangle$ )**  
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-GlobalDet* :  
 $\langle \text{determ}^0 (\sqcap a \in A. P a) \rangle$  **if**  $\langle \bigwedge a. a \in A \implies \text{determ}^0 (P a) \rangle$   
 $\langle \text{range tick} \cap (\bigcap a \in A. (P a)^0) \neq \{\} \vee \text{range tick} \cap (\bigcup a \in A. (P a)^0) = \{\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-Seq<sub>ptick</sub>* :  
 $\langle \text{determ}^0 (P ; \checkmark Q) \longleftrightarrow (\forall r. \checkmark(r) \in P^0 \longrightarrow \text{determ}^0 (Q r)) \rangle$  **if**  $\langle \text{determ}^0 P \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *accepts-initials-Seq* :  
 $\langle \text{determ}^0 (P ; Q) \longleftrightarrow (P^0 \cap \text{range tick} = \{\} \vee \text{determ}^0 Q) \rangle$  **if**  $\langle \text{determ}^0 P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *Sync<sub>ptick</sub>-locale*) *accepts-initials-Sync<sub>ptick</sub>* :  
 $\langle \text{determ}^0 (P \llbracket S \rrbracket \checkmark Q) \rangle$  **if**  $\langle \text{determ}^0 P \rangle \langle \text{determ}^0 Q \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *accepts-initials-Sync*:  
 $\langle \text{determ}^0 P \implies \text{determ}^0 Q \implies \text{determ}^0 (P \llbracket S \rrbracket Q) \rangle$   
 $\langle \text{proof} \rangle$

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

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

**lemma** *accepts-initials-Sliding*:  
 $\langle \text{determ}^0 P \implies \text{determ}^0 Q \implies \text{determ}^0 (P \triangleright Q) \longleftrightarrow$   
 $P = \text{STOP} \vee P^0 \subseteq Q^0 \wedge (\text{range tick} \cap P^0 \neq \{\} \vee \text{range tick} \cap Q^0 = \{\}) \rangle$   
 $\langle \text{proof} \rangle$

## 2.1.5 Characterizations with After

**context** *After*  
**begin**

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

**lemma** *leFD-SKIPS-Det-Mprefix-After*:  
 $\langle P \sqsubseteq_{FD} \text{SKIPS} \{r. \checkmark(r) \in P^0\} \square (\square a \in \{a. \text{ev } a \in P^0\} \rightarrow P \text{ after } a) \rangle$  (**is**  $\langle P \sqsubseteq_{FD} ?rhs \rangle$ )  
 $\langle \text{proof} \rangle$

**lemma** *accepts-initials-imp-eq-Mprefix-After*:  
 $\langle P = ( \text{if } \exists r. \checkmark(r) \in P^0 \text{ then } \text{SKIP } (\text{THE } r. \checkmark(r) \in P^0)$   
 $\text{else } \square a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a) \rangle$  (**is**  $\langle P = ?rhs \rangle$ )

**if**  $\langle \text{determ}^0 P \rangle$   
 $\langle \text{proof} \rangle$

**theorem** *is-some-Mprefix-iff*:  
 $\langle (\exists A Q. P = \Box a \in A \rightarrow Q a) \longleftrightarrow \text{range tick} \cap P^0 = \{\} \wedge \text{accepts-initials } P \rangle$   
**for**  $P :: \langle ('a, 'r) \text{ process}_{ptick} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *tick-not-initial-imp-STOP-Ndet-Mndetprefix-After-FD*:  
 $\langle \text{range tick} \cap P^0 = \{\} \implies \text{STOP} \sqcap (\Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a) \sqsubseteq_{FD} P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *lifelock-free*  $P \longleftrightarrow \mathcal{D} P = \{\} \wedge (\forall t \in \mathcal{T} P. tF t)$   
 $\langle \text{proof} \rangle$

**lemma** *STOP-Ndet-SKIPS-Ndet-Mprefix-After-leF* :  
 $\langle \text{STOP} \sqcap \text{SKIPS } \{r. \checkmark(r) \in P^0\} \sqcap (\Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a) \sqsubseteq_F P \rangle$   
**(is**  $\langle - \sqcap ?lhs1 \sqcap ?lhs2 \sqsubseteq_F P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *non-BOT-imp-Mprefix-After-leD* :  
 $\langle \Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a \sqsubseteq_D P \rangle$  **(is**  $\langle - ?lhs \sqsubseteq_D P \rangle$  **if**  $\langle P \neq \perp \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *non-BOT-imp-STOP-Ndet-SKIPS-Ndet-Mprefix-After-leFD* :  
 $\langle P \neq \perp \implies \text{STOP} \sqcap \text{SKIPS } \{r. \checkmark(r) \in P^0\} \sqcap (\Box a \in \{e. \text{ev } e \in P^0\} \rightarrow P \text{ after } a) \sqsubseteq_{FD} P \rangle$   
 $\langle \text{proof} \rangle$

**theorem** *singl-initial-imp-equals-prefix-After*:  
 $\langle P = (\text{if } \text{UNIV} \notin \mathcal{R} P \text{ then } a \rightarrow P \text{ after } a \text{ else } \text{STOP} \sqcap (a \rightarrow P \text{ after } a)) \rangle$   
**if** *initials-is* :  $\langle \text{initials } P = \{\text{ev } a\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\langle \{\text{ev } e\} \notin \mathcal{R} P \implies \text{ev } e \in P^0 \rangle$   
 $\langle \text{proof} \rangle$

**end**

## 2.2 Deterministic process

### 2.2.1 Definition

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

**lemma** *deterministicI* :  $\langle (\bigwedge t e. t @ [e] \in \mathcal{T} P \Longrightarrow (t, \{e\}) \notin \mathcal{F} (P)) \Longrightarrow \text{determ } P \rangle$

**and** *deterministicD* :  $\langle \text{determ } P \Longrightarrow t @ [e] \in \mathcal{T} P \Longrightarrow (t, \{e\}) \notin \mathcal{F} (P) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deterministic-STOP* [*simp*] :  $\langle \text{determ } \text{STOP} \rangle$   
**and** *deterministic-SKIP* [*simp*] :  $\langle \text{determ } (\text{SKIP } r) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deterministic-div-free* :  $\langle \text{determ } P \Longrightarrow \mathcal{D} P = \{\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *not-deterministic-BOT* [*simp*] :  $\langle \neg \text{determ } \perp \rangle$   
 $\langle \text{proof} \rangle$

### 2.2.2 Monotonicity

**lemma** *mono-deterministic-F*:  $\langle P \sqsubseteq_F Q \Longrightarrow \text{determ } P \Longrightarrow \text{determ } Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-deterministic-FD*:  $\langle P \sqsubseteq_{FD} Q \Longrightarrow \text{determ } P \Longrightarrow \text{determ } Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-deterministic*:  $\langle P \sqsubseteq Q \Longrightarrow \text{determ } P \Longrightarrow \text{determ } Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *restriction-adm-deterministic* [*restriction-adm-process<sub>ptick</sub>-simpset, simp*]

:

$\langle \text{adm}_\downarrow (\lambda x. \text{determ } (f x)) \rangle$  **if**  $\langle \text{cont}_\downarrow f \rangle$

**for**  $f :: \langle 'b :: \text{restriction} \Rightarrow ('a, 'r) \text{ process}_{ptick} \rangle$

$\langle \text{proof} \rangle$

### 2.2.3 Characterization as Maximal

#### Some preliminary work

**definition** *is-process<sub>T</sub>* ::  $\langle ('a, 'r) \text{ trace}_{ptick} \text{ set} \Rightarrow \text{bool} \rangle$

**where**  $\langle \text{is-process}_T T \equiv$

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

**typedef**  $\langle ('a, 'r) \text{ process}_T = \langle \{ T :: ('a, 'r) \text{ trace}_{ptick} \text{ set} . \text{is-process}_T T \} \rangle$

$\langle proof \rangle$

**setup-lifting** *type-definition-process<sub>T</sub>*

**lift-definition** *Traces<sub>T</sub>* ::

$\langle ('a, 'r) process_T \Rightarrow ('a, 'r) trace_{ptick} set \rangle \langle \mathcal{T}_T \rangle$   
**is**  $\langle \lambda P. Rep-process_T P \rangle \langle proof \rangle$

**lemma** *Process<sub>T</sub>-eq-spec* :  $\langle T = U \longleftrightarrow \mathcal{T}_T T = \mathcal{T}_T U \rangle$   
 $\langle proof \rangle$

**lemma** *is-process<sub>T</sub>-1* :  $\langle [] \in \mathcal{T}_T P \rangle$

**and** *is-process<sub>T</sub>-2* :  $\langle s \in \mathcal{T}_T P \Longrightarrow ftF s \rangle$

**and** *is-process<sub>T</sub>-3* :  $\langle s @ t \in \mathcal{T}_T P \Longrightarrow s \in \mathcal{T}_T P \rangle$

**and** *is-process<sub>T</sub>-4* :  $\langle s @ [\checkmark(r)] \in \mathcal{T}_T P \Longrightarrow e \neq \checkmark(r) \Longrightarrow s @ [e] \notin \mathcal{T}_T P \rangle$   
 $\langle proof \rangle$

**lemmas** *is-process<sub>T</sub>-def-bis = is-process<sub>T</sub>-def[of  $\langle Rep-process_T \rightarrow \rangle$ , folded *Traces<sub>T</sub>.rep-eq*]*

**lift-definition** *process<sub>ptick</sub>-of-process<sub>T</sub>* ::

$\langle ('a, 'r) process_T \Rightarrow ('a, 'r) process_{ptick} \rangle$

**is**  $\langle \lambda T. (\{(s, X). s \in \mathcal{T}_T T \wedge X \subseteq - \{e. s @ [e] \in \mathcal{T}_T T\}\}, \{\}) \rangle$   
 $\langle proof \rangle$

**lemma** *F-process<sub>ptick</sub>-of-process<sub>T</sub>* :

$\langle \mathcal{F} (process_{ptick}\text{-of-process}_T T) = \{(s, X). s \in \mathcal{T}_T T \wedge X \subseteq - \{e. s @ [e] \in \mathcal{T}_T T\}\} \rangle$

**and** *D-process<sub>ptick</sub>-of-process<sub>T</sub>* :

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

**and** *T-process<sub>ptick</sub>-of-process<sub>T</sub>* :

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

$\langle proof \rangle$

**lemmas** *process<sub>ptick</sub>-of-process<sub>T</sub>-projs = F-process<sub>ptick</sub>-of-process<sub>T</sub>*

*D-process<sub>ptick</sub>-of-process<sub>T</sub> T-process<sub>ptick</sub>-of-process<sub>T</sub>*

**Now the big results**

**lemma** *bij-betw-det* :

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

**(is**  $\langle bij\text{-betw } process_{ptick}\text{-of-process}_T ?S1 ?S2 \rangle$

$\langle proof \rangle$

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

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

**theorem** *P-FD-some-det* :

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

**fixes** *termination-choice* ::  $\langle ('a, 'r) \text{trace}_{\text{ptick}} \Rightarrow 'r \rangle$

**assumes**  $\langle \bigwedge t. \exists r. t @ [\checkmark(r)] \in \mathcal{T} P \implies \text{termination-choice } t \in \{r. t @ [\checkmark(r)] \in \mathcal{T} P\} \rangle$

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

**shows**  $\langle P \sqsubseteq_{FD} \text{process}_{\text{ptick-of-process}} T (\text{Abs-process}_T T) \rangle$   
 $\langle \text{proof} \rangle$

**theorem** *deterministic-iff-maximal-for-leFD*:

$\langle \text{determ } P \iff (\forall Q. P \sqsubseteq_{FD} Q \longrightarrow P = Q) \rangle$  **for**  $P :: \langle ('a, 'r) \text{process}_{\text{ptick}} \rangle$

— see TPC, chapter 9)

$\langle \text{proof} \rangle$

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

We have the immediate powerful corollaries.

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

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

**corollary** (in *After*) *deterministic-process-eq-SKIP-or-Mprefix-After* :  
 $\langle \text{determ } P \implies P = (\text{if } \exists r. \checkmark(r) \in P^0 \text{ then SKIP (THE } r. P^0 = \{\checkmark(r)\})$   
 $\text{else } \Box a \in \{a. \text{ev } a \in P^0\} \rightarrow P \text{ after } a) \rangle$   
 $\langle \text{proof} \rangle$

## 2.2.4 Characterization with After

**lemma** (in *AfterExt*) *deterministic-iff-accepts-initials-After<sub>trace</sub>*:  
 $\langle \text{determ } P \iff (\forall t \in \mathcal{T} P. tF t \longrightarrow \text{determ}^0 (P \text{ after}_{\mathcal{T}} t)) \rangle$   
 $\langle \text{proof} \rangle$

## 2.2.5 Operators preserving Determinism

**lemma** *deterministic-Mprefix-iff* :  
 $\langle \text{determ } (\Box a \in A \rightarrow P a) \iff (\forall a \in A. \text{determ } (P a)) \rangle$   
 $\langle \text{proof} \rangle$

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

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

**corollary** *deterministic-inj-on-read-iff* :  
 $\langle \text{inj-on } c A \implies \text{determ } (c?a \in A \rightarrow P a) \iff (\forall a \in A. \text{determ } (P a)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deterministic-inj-Renaming* :  
 $\langle \text{determ } (\text{Renaming } P f g) \text{ if } \langle \text{inj } f \rangle \langle \text{inj } g \rangle \langle \text{determ } P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deterministic-bij-Renaming-iff* :  
 $\langle \text{determ } (\text{Renaming } P f g) \iff \text{determ } P \rangle \text{ if } \langle \text{bij } f \rangle \text{ and } \langle \text{bij } g \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deterministic-Throw* :  $\langle \text{determ } (P \Theta a \in A. Q a) \rangle$   
 $\text{if } \langle \text{determ } P \rangle \langle \bigwedge a. a \in A \implies a \in \alpha(P) \implies \text{determ } (Q a) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *T-snoc-tick-imp-no-continuation-if-deterministic* :  
 $\langle u = [] \wedge e = \checkmark(r) \rangle \text{ if } \langle \text{determ } P \rangle \langle t @ u @ [e] \in \mathcal{T} P \rangle \langle t @ [\checkmark(r)] \in \mathcal{T} P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *T-snoc-ev-imp-no-tick-continuation-if-deterministic* :  
 $\langle u \neq [] \wedge \text{is-ev } (\text{hd } u) \vee \text{is-ev } e \rangle \text{ if } \langle \text{determ } P \rangle \langle t @ u @ [e] \in \mathcal{T} P \rangle \langle t @ [ev a] \in \mathcal{T} P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deterministic-Seq<sub>ptick</sub>* :  $\langle \text{determ } (P ; \checkmark Q) \rangle$   
**if**  $\langle \text{determ } P \rangle \langle \bigwedge r. r \in \checkmark \mathbf{s}(P) \implies \text{determ } (Q r) \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *deterministic-Seq* :  $\langle \text{determ } P \implies \text{determ } Q \implies \text{determ } (P ; Q) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** (*in After*) *initial-imp-deterministic-After*:  
 $\langle ev e \in P^0 \implies \text{determ } P \implies \text{determ } (P \text{ after } e) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** (*in AfterExt*) *initial-imp-deterministic-After<sub>tick</sub>*:  
 $\langle e \in P^0 \implies (\text{case } e \text{ of } \checkmark(r) \Rightarrow \text{determ } (\Omega P r)) \implies \text{determ } P \implies \text{determ } (P \text{ after } \checkmark e) \rangle$   
 $\langle \text{proof} \rangle$

## 2.2.6 Operators not (always) preserving Determinism

**lemma** *deterministic-imp-accepts-initials* :  $\langle \text{determ } P \implies \text{determ}^0 P \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma** *deterministic-Det*:  
 $\langle \text{determ } P \implies \text{determ } Q \implies \text{range tick} \cap P^0 \cap Q^0 \neq \{\} \vee P^0 \cap Q^0 = \{\} \wedge \text{range tick} \cap (P^0 \cup Q^0) = \{\} \implies \text{determ } (P \square Q) \rangle$   
 $\langle \text{proof} \rangle$

## 2.3 Application to Operational Semantics

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

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

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

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

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



## 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*] :  
⟨ $\llbracket \text{length } xs = \text{length } ys; P \ [] \ []; \bigwedge x1 \ y1. P \ [x1] \ [y1];$   
 $\bigwedge x1 \ x2 \ xs \ y1 \ y2 \ ys. \text{length } xs = \text{length } ys \implies P \ xs \ ys \implies$   
 $P \ (x2 \ \# \ xs) \ (y2 \ \# \ ys) \implies P \ (x1 \ \# \ x2 \ \# \ xs) \ (y1 \ \# \ y2 \ \# \ ys) \rrbracket$   
 $\implies P \ xs \ ys$ ⟩  
⟨*proof*⟩

**lemma** *nat-induct-012* [*case-names 0 1 2 Suc*]:  
⟨ $\llbracket P \ 0; P \ (Suc \ 0); P \ (Suc \ (Suc \ 0)); \bigwedge k. Suc \ (Suc \ 0) \leq k \implies P \ k \implies P \ (Suc \ k) \rrbracket$   
 $\implies P \ n$ ⟩  
⟨*proof*⟩

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

**lemma** *restriction-shift-iterated* :  
⟨*restriction-shift* ( $f \ \widehat{\ \ } \ k$ ) (*int*  $k \ * \ m$ )⟩  
**if** ⟨*restriction-shift*  $f \ m$ ⟩ **for**  $f :: \langle 'a \Rightarrow 'a :: \text{restriction-space} \rangle$   
⟨*proof*⟩

**lemma** *non-destructive-iterated* :  
⟨*non-destructive*  $f \implies \text{non-destructive} \ (f \ \widehat{\ \ } \ k)$ ⟩  
**for**  $f :: \langle 'a \Rightarrow 'a :: \text{restriction-space} \rangle$   
⟨*proof*⟩

**lemma** *constructive-iterated* :  
⟨*constructive* ( $f \ \widehat{\ \ } \ k$ )⟩ **if**  $\langle 0 < k \rangle$  ⟨*constructive*  $f$ ⟩  
**for**  $f :: \langle 'a \Rightarrow 'a :: \text{restriction-space} \rangle$   
⟨*proof*⟩

**lemma** *restriction-fix-unique-iterated* :

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

**lemma** *restriction-fix-iterated* :

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

**corollary** *restriction-fix-ind-iterated*

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

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

## 3.1 Definitions

### 3.1.1 Non-deterministic and deterministic Automata

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

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

**type-synonym**  $\langle 'a, 'a' \rangle$  *trans* =  $\langle 'a \Rightarrow 'a \Rightarrow 'a' \rangle$

**type-synonym**  $\langle 'a, 'a' \rangle$  *trans<sub>d</sub>* =  $\langle ('a, 'a, 'a' \text{ option}) \text{ trans} \rangle$

**type-synonym**  $\langle 'a, 'a' \rangle$  *trans<sub>nd</sub>* =  $\langle ('a, 'a, 'a' \text{ set}) \text{ trans} \rangle$

**record**  $\langle 'a, 'a', 'a', 'a' \rangle$  *A* =

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

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

**type-synonym**  $\langle 'a, 'a, 'a' \rangle$  *A<sub>d</sub>* =  $\langle ('a, 'a, 'a' \text{ option}, 'a' \text{ option}) A \rangle$

**type-synonym**  $\langle 'a, 'a, 'a, 'a' \rangle$  *A<sub>d</sub>-scheme* =  $\langle ('a, 'a, 'a' \text{ option}, 'a' \text{ option}, 'a') A\text{-scheme} \rangle$

**type-synonym**  $\langle 'a, 'a, 'a' \rangle$  *A<sub>nd</sub>* =  $\langle ('a, 'a, 'a' \text{ set}, 'a' \text{ set}) A \rangle$

**type-synonym**  $\langle 'a, 'a, 'a, 'a' \rangle$  *A<sub>nd</sub>-scheme* =  $\langle ('a, 'a, 'a' \text{ set}, 'a' \text{ set}, 'a') A\text{-scheme} \rangle$

### 3.1.2 Enableness

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

**overloading**

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

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

**begin**

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

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

```

fun  $\varepsilon_{nd} :: \langle (' \sigma, ' a, ' \sigma \text{ set}, ' r', ' \alpha) A\text{-scheme} \Rightarrow (' \sigma, ' a) \text{ enabl} \rangle$ 
  where  $\langle \varepsilon_{nd} A \sigma = \{ a. \tau A \sigma a \neq \{\} \} \rangle$ 
end

```

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

### 3.1.3 States allowing Termination

```

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

```

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

### 3.1.4 Reachability

```

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

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

```

**lemma**  $\mathcal{R}_d\text{-trans}$ :  $\langle \sigma'' \in \mathcal{R}_d A \sigma' \Longrightarrow \sigma' \in \mathcal{R}_d A \sigma \Longrightarrow \sigma'' \in \mathcal{R}_d A \sigma \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\mathcal{R}_{nd}\text{-trans}$ :  $\langle \sigma'' \in \mathcal{R}_{nd} A \sigma' \Longrightarrow \sigma' \in \mathcal{R}_{nd} A \sigma \Longrightarrow \sigma'' \in \mathcal{R}_{nd} A \sigma \rangle$   
 $\langle \text{proof} \rangle$

### 3.1.5 Morphisms

Our morphisms are defined considering that, except from  $\tau$ , the fields remain unchanged.

```

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

```

**definition** *from-ndet-to-det* ::

$\langle ('σ, 'a, 'r, 'α) A_{nd}\text{-scheme} \Rightarrow ('σ, 'a, 'r, 'α) A_d\text{-scheme} \rangle$

**where**  $\langle \text{from-ndet-to-det } A \equiv$

$(\tau = \lambda\sigma a. \text{ if } \tau A \sigma a = \{\} \text{ then } \diamond \text{ else } [THE \sigma'. \sigma' \in \tau A \sigma a],$

$\omega = \lambda\sigma. \text{ if } \omega A \sigma = \{\} \text{ then } \diamond \text{ else } [THE r. r \in \omega A \sigma], \dots = \text{more } A) \rangle$

**definition** *from-σ-to-σ<sub>s<sub>d</sub></sub>* ::

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

**where**  $\langle \text{from-σ-to-σ}_{s_d} A \equiv$

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

$\omega = \lambda\sigma s. \omega A (hd \sigma s), \dots = \text{more } A) \rangle$

**definition** *from-σ-to-σ<sub>s<sub>nd</sub></sub>* ::

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

**where**  $\langle \text{from-σ-to-σ}_{s_{nd}} A \equiv$

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

$\omega = \lambda\sigma s. \omega A (hd \sigma s), \dots = \text{more } A) \rangle$

**definition** *from-σ<sub>s</sub>-to-σ<sub>d</sub>* ::

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

**where**  $\langle \text{from-σ}_s\text{-to-σ}_d A \equiv$

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

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

**definition** *from-σ<sub>s</sub>-to-σ<sub>nd</sub>* ::

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

**where**  $\langle \text{from-σ}_s\text{-to-σ}_{nd} A \equiv$

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

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

**definition** *from-singl-to-list<sub>d</sub>* ::

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

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

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

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

**definition** *from-singl-to-list<sub>nd</sub>* ::

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

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

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

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

**definition** *from-list-to-singl<sub>d</sub>* ::

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

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

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

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

**definition** *from-list-to-singl<sub>nd</sub>* ::

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

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

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

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

**lemmas** *det-ndet-conv-defs* = *from-det-to-ndet-def from-ndet-to-det-def*  
**and**  $\sigma$ - $\sigma s$ -*conv-defs* = *from- $\sigma$ -to- $\sigma s_d$ -def from- $\sigma$ -to- $\sigma s_{nd}$ -def*  
*from- $\sigma s$ -to- $\sigma_d$ -def from- $\sigma s$ -to- $\sigma_{nd}$ -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*  $\langle \langle \_ \rangle_{d \hookrightarrow nd} \rangle [0]$   
**notation** *from-ndet-to-det*  $\langle \langle \_ \rangle_{nd \rightsquigarrow d} \rangle [0]$   
**notation** *from- $\sigma$ -to- $\sigma s_d$*   $\langle \langle \_ \rangle_{\sigma \hookrightarrow \sigma s} \rangle [0]$   
**notation** *from- $\sigma$ -to- $\sigma s_{nd}$*   $\langle \langle \_ \rangle_{\sigma \hookrightarrow \sigma s} \rangle [0]$   
**notation** *from- $\sigma s$ -to- $\sigma_d$*   $\langle \langle \_ \rangle_{\sigma s \rightsquigarrow \sigma} \rangle [0]$   
**notation** *from- $\sigma s$ -to- $\sigma_{nd}$*   $\langle \langle \_ \rangle_{\sigma s \rightsquigarrow \sigma} \rangle [0]$   
**notation** *from-singl-to-list $_d$*   $\langle \langle \_ \rangle_{singl \hookrightarrow list} \rangle [0]$   
**notation** *from-singl-to-list $_{nd}$*   $\langle \langle \_ \rangle_{singl \hookrightarrow list} \rangle [0]$   
**notation** *from-list-to-singl $_d$*   $\langle \langle \_ \rangle_{list \rightsquigarrow singl} \rangle [0]$   
**notation** *from-list-to-singl $_{nd}$*   $\langle \langle \_ \rangle_{list \rightsquigarrow singl} \rangle [0]$

**end**

**unbundle** *functional-automata-morphisms-syntax*

**lemma** *morphisms-A-scheme-more-simps* [*simp*] :  
 $\langle \text{more } \langle \langle A \rangle_{d \hookrightarrow nd} = \text{more } A \rangle \langle \text{more } \langle \langle B \rangle_{nd \rightsquigarrow d} = \text{more } B \rangle$   
 $\langle \text{more } d \langle \langle C \rangle_{\sigma \hookrightarrow \sigma s} = \text{more } C \rangle \langle \text{more } nd \langle \langle D \rangle_{\sigma \hookrightarrow \sigma s} = \text{more } D \rangle$   
 $\langle \text{more } d \langle \langle E \rangle_{\sigma s \rightsquigarrow \sigma} = \text{more } E \rangle \langle \text{more } nd \langle \langle F \rangle_{\sigma s \rightsquigarrow \sigma} = \text{more } F \rangle$   
 $\langle \text{more } d \langle \langle G \rangle_{singl \hookrightarrow list} = \text{more } G \rangle \langle \text{more } nd \langle \langle H \rangle_{singl \hookrightarrow list} = \text{more } H \rangle$   
 $\langle \text{more } d \langle \langle I \rangle_{list \rightsquigarrow singl} = \text{more } I \rangle \langle \text{more } nd \langle \langle J \rangle_{list \rightsquigarrow singl} = \text{more } J \rangle$   
 $\langle \text{proof} \rangle$

### 3.1.6 Generic update Functions

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

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

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

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

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

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

### 3.1.7 Assumptions on Automata

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

**lemma**  $\text{finite-trans-morphisms-simps}[\text{simp}]$ :

$\langle \text{finite-trans } \langle A \rangle_{d \hookrightarrow nd} \rangle$   
 $\langle \text{finite-trans } B \Longrightarrow \text{finite-trans}_{nd} \langle B \rangle_{\sigma \hookrightarrow \sigma s} \rangle$   
 $\langle \text{finite-trans } C \Longrightarrow \text{finite-trans}_{nd} \langle C \rangle_{\sigma s \rightsquigarrow \sigma} \rangle$   
 $\langle \text{finite-trans } D \Longrightarrow \text{finite-trans}_{nd} \langle D \rangle_{\text{singl} \hookrightarrow \text{list}} \rangle$   
 $\langle \text{finite-trans } E \Longrightarrow \text{finite-trans}_{nd} \langle E \rangle_{\text{list} \rightsquigarrow \text{singl}} \rangle$   
 $\langle \text{proof} \rangle$

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

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

**lemma**  $\text{at-most-1-elem-def-bis}$  :

$\langle \text{at-most-1-elem } A \longleftrightarrow (\forall \sigma a. \exists \sigma'. \tau A \sigma a \subseteq \{\sigma'\}) \wedge (\forall \sigma. \exists r. \omega A \sigma \subseteq \{r\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{at-most-1-elemI}$  :

$\langle [(\forall \sigma a. \tau A \sigma a = \{\}) \vee (\exists \sigma'. \tau A \sigma a = \{\sigma'\});$   
 $\quad \wedge \sigma. \omega A \sigma = \{\} \vee (\exists r. \omega A \sigma = \{r\})] \Longrightarrow \text{at-most-1-elem } A \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{at-most-1-elemE}$  :

$\langle [(\tau A \sigma a = \{\}) \Longrightarrow \text{thesis}; \wedge \sigma'. \tau A \sigma a = \{\sigma'\} \Longrightarrow \text{thesis}] \Longrightarrow \text{thesis} \rangle$   
 $\langle [(\omega A \sigma = \{\}) \Longrightarrow \text{thesis}; \wedge r. \omega A \sigma = \{r\} \Longrightarrow \text{thesis}] \Longrightarrow \text{thesis} \rangle$   
**if**  $\langle \text{at-most-1-elem } A \rangle$   
 $\langle \text{proof} \rangle$

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

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

**lemma**  $\text{at-most-1-elem-trans-def-bis}$  :

$\langle \text{at-most-1-elem-trans } A \longleftrightarrow (\forall \sigma a. \exists \sigma'. \tau A \sigma a \subseteq \{\sigma'\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{at-most-1-elem-transI}$  :

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

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

$\langle \llbracket \tau A \sigma a = \{\} \implies thesis; \bigwedge \sigma'. \tau A \sigma a = \{\sigma'\} \implies thesis \rrbracket \implies thesis \rangle$   
**if**  $\langle at-most-1-elem-trans A \rangle$   
 $\langle proof \rangle$

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

$\langle at-most-1-elem A \implies at-most-1-elem-trans A \rangle$   
 $\langle proof \rangle$

**definition** *length-1-trans<sub>d</sub>* ::  $\langle (' \sigma list, 'a, 'r, ' \alpha) A_d\text{-scheme} \implies bool \rangle$

**where**  $\langle length-1-trans_d A \equiv$

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

**lemma** *length-1-trans<sub>d</sub>I* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \tau A \sigma s a = \llbracket \sigma s' \rrbracket \implies length \sigma s' = Suc 0 \rrbracket \implies length-1-trans_d A \rangle$   
 $\langle proof \rangle$

**lemma** *length-1-trans<sub>d</sub>E* :

$\langle \llbracket length-1-trans_d A; \tau A \sigma s a = \llbracket \sigma s' \rrbracket; \bigwedge \sigma. \sigma s' = [\sigma] \implies thesis \rrbracket \implies thesis \rangle$   
 $\langle proof \rangle$

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

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

**lemma** *length-1-trans<sub>nd</sub>I* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \sigma s' \in \tau A \sigma s a \implies length \sigma s' = Suc 0 \rrbracket \implies length-1-trans_{nd} A \rangle$   
 $\langle proof \rangle$

**lemma** *length-1-trans<sub>nd</sub>E* :

$\langle \llbracket length-1-trans_{nd} A; \sigma s' \in \tau A \sigma s a; \bigwedge \sigma. \sigma s' = [\sigma] \implies thesis \rrbracket \implies thesis \rangle$   
 $\langle proof \rangle$

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

**where**  $\langle length-1_d A \equiv$

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

**lemma** *length-1<sub>d</sub>I* :

$\langle \llbracket \bigwedge \sigma s a \sigma s'. \tau A \sigma s a = \llbracket \sigma s' \rrbracket \implies length \sigma s' = Suc 0; \bigwedge \sigma s rs. \omega A \sigma s = \llbracket rs \rrbracket \implies length rs = Suc 0 \rrbracket \implies length-1_d A \rangle$   
 $\langle proof \rangle$

**lemma** *length-1<sub>d</sub>E* :

$\langle \llbracket length-1_d A; \tau A \sigma s a = \llbracket \sigma s' \rrbracket; \bigwedge \sigma. \sigma s' = [\sigma] \implies thesis \rrbracket \implies thesis \rangle$

$\langle \llbracket \text{length-1}_d A; \omega A \sigma s = [rs]; \bigwedge r. rs = [r] \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\text{length-1}_{nd}I :$   
 $\langle \llbracket \bigwedge \sigma s a \sigma s'. \sigma s' \in \tau A \sigma s a \implies \text{length } \sigma s' = \text{Suc } 0;$   
 $\llbracket \bigwedge \sigma s rs. rs \in \omega A \sigma s \implies \text{length } rs = \text{Suc } 0 \rrbracket \implies \text{length-1}_{nd} A \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{length-1}_{nd}E :$   
 $\langle \llbracket \text{length-1}_{nd} A; \sigma s' \in \tau A \sigma s a; \bigwedge \sigma. \sigma s' = [\sigma] \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$   
 $\langle \llbracket \text{length-1}_{nd} A; rs \in \omega A \sigma s; \bigwedge r. rs = [r] \implies \text{thesis} \rrbracket \implies \text{thesis} \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\text{indep-enabl}I :$   
 $\langle (\bigwedge t_0 t_1. t_0 \in \mathcal{R}_d A_0 \sigma_0 \implies t_1 \in \mathcal{R}_d A_1 \sigma_1 \implies \varepsilon A_0 t_0 \cap \varepsilon A_1 t_1 \subseteq E)$   
 $\implies \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
**and**  $\text{indep-enabl}D :$   
 $\langle \llbracket \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1; t_0 \in \mathcal{R}_d A_0 \sigma_0; t_1 \in \mathcal{R}_d A_1 \sigma_1 \rrbracket \implies \varepsilon A_0 t_0 \cap \varepsilon$   
 $A_1 t_1 \subseteq E \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\varrho\text{-disjoint-}\varepsilon I : \langle (\bigwedge \sigma. \sigma \in \varrho A \implies \varepsilon A \sigma = \{\}) \implies \varrho\text{-disjoint-}\varepsilon A \rangle$   
**and**  $\varrho\text{-disjoint-}\varepsilon D : \langle \varrho\text{-disjoint-}\varepsilon A \implies \sigma \in \varrho A \implies \varepsilon A \sigma = \{\} \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\text{at-most-1-elem-term-def-bis} :$

$\langle \text{at-most-1-elem-term } A \longleftrightarrow (\forall \sigma. \exists r. \omega A \sigma \subseteq \{r\}) \rangle$   
 $\langle \text{proof} \rangle$

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

$\langle \llbracket \bigwedge \sigma. \omega A \sigma = \{\} \vee (\exists r. \omega A \sigma = \{r\}) \rrbracket \Longrightarrow \text{at-most-1-elem-term } A \rangle$   
 $\langle \text{proof} \rangle$

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

$\langle \llbracket \omega A \sigma = \{\} \Longrightarrow \text{thesis}; \bigwedge r. \omega A \sigma = \{r\} \Longrightarrow \text{thesis} \rrbracket \Longrightarrow \text{thesis} \rangle$   
**if**  $\langle \text{at-most-1-elem-term } A \rangle$   
 $\langle \text{proof} \rangle$

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

$\langle \text{at-most-1-elem } A \Longrightarrow \text{at-most-1-elem-term } A \rangle$   
 $\langle \text{proof} \rangle$

## 3.2 First Properties

### 3.2.1 $\varepsilon$ , $\varrho$ and $\omega$ first equalities

**lemma** *base-trans- $\varepsilon$ [simp]*:

$\langle \varepsilon ((\tau = \lambda \sigma a. \diamond, \omega = \lambda \sigma. \diamond, \dots = \text{some}) :: ('\sigma, 'a, 'r, 'a) A_d\text{-scheme}) \sigma = \{\} \rangle$   
 $\langle \varepsilon ((\tau = \lambda \sigma a. \{\}, \omega = \lambda \sigma. \{\}, \dots = \text{some}) :: ('\sigma, 'a, 'r, 'a) A_{nd}\text{-scheme}) \sigma = \{\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *base-trans- $\varrho$ [simp]*:

$\langle \varrho ((\tau = \lambda \sigma a. \diamond, \omega = \lambda \sigma. \diamond, \dots = \text{some}) :: ('\sigma, 'a, 'r, 'a) A_d\text{-scheme}) = \{\} \rangle$   
 $\langle \varrho ((\tau = \lambda \sigma a. \{\}, \omega = \lambda \sigma. \{\}, \dots = \text{some}) :: ('\sigma, 'a, 'r, 'a) A_{nd}\text{-scheme}) = \{\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\sigma$ - $\sigma$ s-conv- $\varepsilon$ [simp]*:

$\langle \varepsilon d \langle \langle A \rangle \rangle_{\sigma \mapsto \sigma s} \sigma s = \varepsilon A (hd \sigma s) \rangle \langle \varepsilon_{nd} \langle \langle B \rangle \rangle_{\sigma \mapsto \sigma s} \sigma s = \varepsilon B (hd \sigma s) \rangle$   
 $\langle \varepsilon d \langle \langle C \rangle \rangle_{\sigma s \rightsquigarrow \sigma} \sigma = \varepsilon C [\sigma] \rangle \langle \varepsilon_{nd} \langle \langle D \rangle \rangle_{\sigma s \rightsquigarrow \sigma} \sigma = \varepsilon D [\sigma] \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\sigma$ - $\sigma$ s-conv- $\varrho$ [simp]*:

$\langle \varrho d \langle \langle A \rangle \rangle_{\sigma \mapsto \sigma s} = \{\sigma s. hd \sigma s \in \varrho A\} \rangle \langle \varrho_{nd} \langle \langle B \rangle \rangle_{\sigma \mapsto \sigma s} = \{\sigma s. hd \sigma s \in \varrho B\} \rangle$   
 $\langle \varrho d \langle \langle C \rangle \rangle_{\sigma s \rightsquigarrow \sigma} = \{\sigma. [\sigma] \in \varrho C\} \rangle \langle \varrho_{nd} \langle \langle D \rangle \rangle_{\sigma s \rightsquigarrow \sigma} = \{\sigma. [\sigma] \in \varrho D\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *singl-list-conv- $\varepsilon$ [simp]*:

$\langle \varepsilon d \langle \langle A \rangle \rangle_{singl \mapsto list} \sigma s = \varepsilon A (hd \sigma s) \rangle \langle \varepsilon_{nd} \langle \langle B \rangle \rangle_{singl \mapsto list} \sigma s = \varepsilon B (hd \sigma s) \rangle$   
 $\langle \varepsilon d \langle \langle C \rangle \rangle_{list \rightsquigarrow singl} \sigma = \varepsilon C [\sigma] \rangle \langle \varepsilon_{nd} \langle \langle D \rangle \rangle_{list \rightsquigarrow singl} \sigma = \varepsilon D [\sigma] \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *singl-list-conv- $\varrho$ [simp]*:

$\langle \varrho \ d \langle \langle A \rangle \rangle_{\text{singl} \hookrightarrow \text{list}} = \{ \sigma s. \text{hd } \sigma s \in \varrho \ A \} \rangle \langle \varrho \ nd \langle \langle B \rangle \rangle_{\text{singl} \hookrightarrow \text{list}} = \{ \sigma s. \text{hd } \sigma s \in \varrho \ B \} \rangle$   
 $\langle \varrho \ d \langle \langle C \rangle \rangle_{\text{list} \rightsquigarrow \text{singl}} = \{ \sigma. [\sigma] \in \varrho \ C \} \rangle \langle \varrho \ nd \langle \langle D \rangle \rangle_{\text{list} \rightsquigarrow \text{singl}} = \{ \sigma. [\sigma] \in \varrho \ D \} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *det-ndet-conv- $\varepsilon$*  [*simp*]:  $\langle \varepsilon \ \langle \langle A \rangle \rangle_{d \hookrightarrow nd} = \varepsilon \ A \rangle \langle \varepsilon \ \langle \langle B \rangle \rangle_{nd \rightsquigarrow d} = \varepsilon \ B \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *det-ndet-conv- $\varrho$*  [*simp*]:  $\langle \varrho \ \langle \langle A \rangle \rangle_{d \hookrightarrow nd} = \varrho \ A \rangle \langle \varrho \ \langle \langle B \rangle \rangle_{nd \rightsquigarrow d} = \varrho \ B \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\omega$ -from-det-to-ndet* :  
 $\langle \omega \ \langle \langle A \rangle \rangle_{d \hookrightarrow nd} = (\lambda \sigma. \text{case } \omega \ A \ \sigma \ \text{of } [r] \Rightarrow \{r\} \mid \diamond \Rightarrow \{\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\varepsilon$ - $\omega$ -useless* [*simp*] :  
 $\langle \varepsilon \ (A(\omega := \text{some-}\omega)) = \varepsilon \ A \rangle \langle \varepsilon \ (B(\omega := \text{some-}\omega')) = \varepsilon \ B \rangle$   
**for**  $A :: \langle (' \sigma, 'a, ' \sigma \ \text{option}, 'r \ \text{option}, ' \alpha) \ A\text{-scheme} \rangle$   
**and**  $B :: \langle (' \sigma, 'a, ' \sigma \ \text{set}, 'r \ \text{set}, ' \alpha) \ A\text{-scheme} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\varrho$ -disjoint- $\varepsilon$ -updated- $\omega$*  [*simp*] :  
 $\langle \varrho\text{-disjoint-}\varepsilon \ (A(\omega := \lambda \sigma. \diamond)) \rangle$   
 $\langle \varrho\text{-disjoint-}\varepsilon \ (B(\omega := \lambda \sigma. \{\})) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\varrho$ -disjoint- $\varepsilon$ -det-ndet-conv-iff* [*simp*] :  
 $\langle \varrho\text{-disjoint-}\varepsilon \ \langle \langle A \rangle \rangle_{d \hookrightarrow nd} \longleftrightarrow \varrho\text{-disjoint-}\varepsilon \ A \rangle$   
 $\langle \varrho\text{-disjoint-}\varepsilon \ \langle \langle B \rangle \rangle_{nd \rightsquigarrow d} \longleftrightarrow \varrho\text{-disjoint-}\varepsilon \ B \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *at-most-1-elem-term-updated- $\omega$*  [*simp*] :  
 $\langle \text{at-most-1-elem-term} \ (A(\omega := \lambda \sigma. \{\})) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *at-most-1-elem-term-from-det-to-ndet* [*simp*] :  
 $\langle \text{at-most-1-elem-term} \ \langle \langle A \rangle \rangle_{d \hookrightarrow nd} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *at-most-1-elem-term-unit* [*simp*] :  
 $\langle \text{at-most-1-elem-term} \ (A :: (' \sigma, 'a, \text{unit}, ' \alpha) \ A_{nd}\text{-scheme}) \rangle$   
 $\langle \text{proof} \rangle$

### 3.2.2 Properties of our morphisms

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

⟨cases A, cases B⟩

**lemma** *base-trans-det-ndet-conv:*

⟨⟨⟨τ = λσ a. ◇, ω = λσ. ◇, ... = some⟩⟩<sub>d↔nd</sub> =  
 ⟨τ = λσ a. {}, ω = λσ. {}, ... = some⟩  
 ⟨⟨⟨τ = λσ a. {}, ω = λσ. {}, ... = some⟩⟩<sub>nd↔d</sub> =  
 ⟨τ = λσ a. ◇, ω = λσ. ◇, ... = some⟩  
 ⟨proof⟩

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

⟨<sub>nd</sub>⟨⟨A⟩<sub>d↔nd</sub>⟩<sub>σ↔σs</sub> = ⟨<sub>d</sub>⟨⟨A⟩<sub>σ↔σs</sub>⟩<sub>d↔nd</sub>⟩ ⟨<sub>nd</sub>⟨⟨B⟩<sub>d↔nd</sub>⟩<sub>σs↔σ</sub> = ⟨<sub>d</sub>⟨⟨B⟩<sub>σs↔σ</sub>⟩<sub>d↔nd</sub>⟩  
 ⟨proof⟩

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

⟨<sub>nd</sub>⟨⟨A⟩<sub>d↔nd</sub>⟩<sub>singl↔list</sub> = ⟨<sub>d</sub>⟨⟨A⟩<sub>singl↔list</sub>⟩<sub>d↔nd</sub>⟩ ⟨<sub>nd</sub>⟨⟨B⟩<sub>d↔nd</sub>⟩<sub>list↔singl</sub> =  
 ⟨<sub>d</sub>⟨⟨B⟩<sub>list↔singl</sub>⟩<sub>d↔nd</sub>⟩  
 ⟨proof⟩

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

⟨at-most-1-elem-trans A ⇒ <sub>d</sub>⟨⟨A⟩<sub>nd↔d</sub>⟩<sub>σ↔σs</sub> = ⟨<sub>nd</sub>⟨⟨A⟩<sub>σ↔σs</sub>⟩<sub>nd↔d</sub>⟩  
 ⟨at-most-1-elem-trans B ⇒ <sub>d</sub>⟨⟨B⟩<sub>nd↔d</sub>⟩<sub>σs↔σ</sub> = ⟨<sub>nd</sub>⟨⟨B⟩<sub>σs↔σ</sub>⟩<sub>nd↔d</sub>⟩  
 ⟨proof⟩

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

⟨at-most-1-elem A ⇒ <sub>d</sub>⟨⟨A⟩<sub>nd↔d</sub>⟩<sub>singl↔list</sub> = ⟨<sub>nd</sub>⟨⟨A⟩<sub>singl↔list</sub>⟩<sub>nd↔d</sub>⟩  
 ⟨at-most-1-elem B ⇒ <sub>d</sub>⟨⟨B⟩<sub>nd↔d</sub>⟩<sub>list↔singl</sub> = ⟨<sub>nd</sub>⟨⟨B⟩<sub>list↔singl</sub>⟩<sub>nd↔d</sub>⟩  
 ⟨proof⟩

**lemma** *behaviour-σ-σ-s-conv:*

⟨ε <sub>d</sub>⟨⟨A⟩<sub>σ↔σs</sub> [σ] = ε A σ⟩  
 ⟨τ <sub>d</sub>⟨⟨A⟩<sub>σ↔σs</sub> [σ] a = (case τ A σ a of ◇ ⇒ ◇ | [t] ⇒ [[t]])⟩  
 ⟨ρ <sub>d</sub>⟨⟨A⟩<sub>σ↔σs</sub> = {σs. hd σs ∈ ρ A}⟩  
 ⟨ω <sub>d</sub>⟨⟨A⟩<sub>σ↔σs</sub> [σ] = ω A σ⟩  
 ⟨ε <sub>nd</sub>⟨⟨B⟩<sub>σ↔σs</sub> [σ] = ε B σ⟩  
 ⟨τ <sub>nd</sub>⟨⟨B⟩<sub>σ↔σs</sub> [σ] a = {σ' | σ'. σ' ∈ τ B σ a}⟩  
 ⟨ρ <sub>nd</sub>⟨⟨B⟩<sub>σ↔σs</sub> = {σs. hd σs ∈ ρ B}⟩  
 ⟨ω <sub>nd</sub>⟨⟨B⟩<sub>σ↔σs</sub> [σ] = ω B σ⟩  
 ⟨ε <sub>d</sub>⟨⟨C⟩<sub>σs↔σ</sub> σ = ε C [σ]⟩  
 ⟨τ <sub>d</sub>⟨⟨C⟩<sub>σs↔σ</sub> σ a = (case τ C [σ] a of ◇ ⇒ ◇ | [σs'] ⇒ [hd σs'])⟩  
 ⟨ρ <sub>d</sub>⟨⟨C⟩<sub>σs↔σ</sub> = {σ. [σ] ∈ ρ C}⟩  
 ⟨ω <sub>d</sub>⟨⟨C⟩<sub>σs↔σ</sub> σ = ω C [σ]⟩  
 ⟨ε <sub>nd</sub>⟨⟨D⟩<sub>σs↔σ</sub> σ = ε D [σ]⟩  
 ⟨τ <sub>nd</sub>⟨⟨D⟩<sub>σs↔σ</sub> σ a = {hd σs' | σs'. σs' ∈ τ D [σ] a}⟩  
 ⟨ρ <sub>nd</sub>⟨⟨D⟩<sub>σs↔σ</sub> = {σ. [σ] ∈ ρ D}⟩ ⟨ω <sub>nd</sub>⟨⟨D⟩<sub>σs↔σ</sub> σ = ω D [σ]⟩

*<proof>*

**lemma** *behaviour-singl-list-conv*:

$\langle \varepsilon_d \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = \varepsilon A \sigma \rangle$   
 $\langle \tau_d \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] a = (\text{case } \tau A \sigma a \text{ of } \diamond \Rightarrow \diamond \mid [t] \Rightarrow [[t]]) \rangle$   
 $\langle \varrho_d \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} = \{\sigma s. \text{hd } \sigma s \in \varrho A\} \rangle$   
 $\langle \omega_d \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = (\text{case } \omega A \sigma \text{ of } \diamond \Rightarrow \diamond \mid [r] \Rightarrow [[r]]) \rangle$   
 $\langle \varepsilon_{nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = \varepsilon B \sigma \rangle$   
 $\langle \tau_{nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] a = \{[\sigma'] \mid \sigma'. \sigma' \in \tau B \sigma a\} \rangle$   
 $\langle \varrho_{nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} = \{\sigma s. \text{hd } \sigma s \in \varrho B\} \rangle$   
 $\langle \omega_{nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} [\sigma] = \{[r] \mid r. r \in \omega B \sigma\} \rangle$   
 $\langle \varepsilon_d \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = \varepsilon C [\sigma] \rangle$   
 $\langle \tau_d \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma a = (\text{case } \tau C [\sigma] a \text{ of } \diamond \Rightarrow \diamond \mid [\sigma s'] \Rightarrow [\text{hd } \sigma s']) \rangle$   
 $\langle \varrho_d \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} = \{\sigma. [\sigma] \in \varrho C\} \rangle$   
 $\langle \omega_d \langle C \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = (\text{case } \omega C [\sigma] \text{ of } \diamond \Rightarrow \diamond \mid [rs] \Rightarrow [\text{hd } rs]) \rangle$   
 $\langle \varepsilon_{nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = \varepsilon D [\sigma] \rangle$   
 $\langle \tau_{nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma a = \{\text{hd } \sigma s' \mid \sigma s'. \sigma s' \in \tau D [\sigma] a\} \rangle$   
 $\langle \varrho_{nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} = \{\sigma. [\sigma] \in \varrho D\} \rangle$   
 $\langle \omega_{nd} \langle D \rangle_{\text{list} \rightsquigarrow \text{singl}} \sigma = \{\text{hd } rs \mid rs. rs \in \omega D [\sigma]\} \rangle$   
*<proof>*

**lemma** *empty-from-det-to-ndet-is-None-trans* [simp] :  $\langle \tau \langle A \rangle_{d \leftrightarrow nd} \sigma a = \{\} \longleftrightarrow \tau A \sigma a = \diamond \rangle$   
*<proof>*

**lemma** *at-most-1-elem-from-det-to-ndet* [simp] :  $\langle \text{at-most-1-elem } \langle A \rangle_{d \leftrightarrow nd} \rangle$   
*<proof>*

**lemma** *from-ndet-to-det-from-det-to-ndet* [simp] :  $\langle \langle \langle A \rangle_{d \leftrightarrow nd} \rangle_{nd \rightsquigarrow d} = A \rangle$   
*<proof>*

**lemma** *from-det-to-ndet-from-ndet-to-det* [simp] :  
 $\langle \langle \langle A \rangle_{nd \rightsquigarrow d} \rangle_{d \leftrightarrow nd} = A \rangle$  **if**  $\langle \text{at-most-1-elem } A \rangle$   
*<proof>*

**theorem** *bij-betw-from-det-to-ndet* :  
 $\langle \text{bij-betw } (\lambda A. \langle A \rangle_{d \leftrightarrow nd}) \text{ UNIV } \{A. \text{at-most-1-elem } A\} \rangle$   
*<proof>*

**lemma** *bij-betw-from-ndet-to-det* :  
 $\langle \text{bij-betw } (\lambda A. \langle A \rangle_{nd \rightsquigarrow d}) \{A. \text{at-most-1-elem } A\} \text{ UNIV} \rangle$   
*<proof>*

**lemma** *length-1-trans-from-σ-to-σs* [simp] :  
 ⟨length-1-trans<sub>d</sub> d⟨A⟩<sub>σ↪σs</sub>⟩ ⟨length-1-trans<sub>nd</sub> nd⟨B⟩<sub>σ↪σs</sub>⟩  
 ⟨proof⟩

**lemma** *τ-hd-from-σ-to-σs-eq* [simp] :  
 ⟨τ d⟨A⟩<sub>σ↪σs</sub> [hd σs] a = τ d⟨A⟩<sub>σ↪σs</sub> σs a⟩  
 ⟨τ nd⟨B⟩<sub>σ↪σs</sub> [hd σs] a = τ nd⟨B⟩<sub>σ↪σs</sub> σs a⟩  
 ⟨proof⟩

**lemma** *ω-hd-from-σ-to-σs-eq* [simp] :  
 ⟨ω d⟨A⟩<sub>σ↪σs</sub> [hd σs] = ω d⟨A⟩<sub>σ↪σs</sub> σs⟩  
 ⟨ω nd⟨B⟩<sub>σ↪σs</sub> [hd σs] = ω nd⟨B⟩<sub>σ↪σs</sub> σs⟩  
 ⟨proof⟩

**lemma** *from-σs-to-σ-from-σ-to-σs* [simp] :  
 ⟨d⟨d⟨A⟩<sub>σ↪σs</sub>⟩<sub>σs↪σ</sub> = A⟩ ⟨nd⟨nd⟨B⟩<sub>σ↪σs</sub>⟩<sub>σs↪σ</sub> = B⟩  
 ⟨proof⟩

**lemma** *from-σ-to-σs-from-σs-to-σ* [simp] :  
 ⟨[length-1-trans<sub>d</sub> A; ∧σs a. τ A [hd σs] a = τ A σs a;  
 ∧σs. ω A [hd σs] = ω A σs] ⇒ d⟨d⟨A⟩<sub>σs↪σ</sub>⟩<sub>σ↪σs</sub> = A⟩  
 ⟨[length-1-trans<sub>nd</sub> B; ∧σs a. τ B [hd σs] a = τ B σs a;  
 ∧σs. ω B [hd σs] = ω B σs] ⇒ nd⟨nd⟨B⟩<sub>σs↪σ</sub>⟩<sub>σ↪σs</sub> = B⟩  
 ⟨proof⟩

**theorem** *bij-betw-from-σ-to-σs* :  
 ⟨bij-betw (λA. d⟨A⟩<sub>σ↪σs</sub>) UNIV  
 {A. length-1-trans<sub>d</sub> A ∧ (∀σs a. τ A [hd σs] a = τ A σs a) ∧ (∀σs. ω A [hd  
 σs] = ω A σs)}⟩  
 (is ⟨bij-betw (λA. d⟨A⟩<sub>σ↪σs</sub>) UNIV ?S<sub>d</sub>⟩)  
 ⟨bij-betw (λB. nd⟨B⟩<sub>σ↪σs</sub>) UNIV  
 {B. length-1-trans<sub>nd</sub> B ∧ (∀σs a. τ B σs a = τ B [hd σs] a) ∧ (∀σs. ω B [hd  
 σs] = ω B σs)}⟩  
 ⟨proof⟩

**lemma** *bij-betw-from-σs-to-σ* :  
 ⟨bij-betw (λA. d⟨A⟩<sub>σs↪σ</sub>)  
 {A. length-1-trans<sub>d</sub> A ∧ (∀σs a. τ A [hd σs] a = τ A σs a) ∧ (∀σs. ω A [hd  
 σs] = ω A σs)} UNIV⟩  
 ⟨bij-betw (λB. nd⟨B⟩<sub>σs↪σ</sub>)  
 {B. length-1-trans<sub>nd</sub> B ∧ (∀σs a. τ B σs a = τ B [hd σs] a) ∧ (∀σs. ω B [hd  
 σs] = ω B σs)} UNIV⟩  
 ⟨proof⟩

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

$\langle \text{length-1}_d \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} \rangle \langle \text{length-1}_{nd} \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\tau$ -hd-from-singl-to-list-eq [simp] :

$\langle \tau \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} [\text{hd } \sigma s] a = \tau \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} \sigma s a \rangle$   
 $\langle \tau \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} [\text{hd } \sigma s] a = \tau \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} \sigma s a \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\omega$ -hd-from-singl-to-list-eq [simp] :

$\langle \omega \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} [\text{hd } \sigma s] = \omega \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} \sigma s \rangle$   
 $\langle \omega \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} [\text{hd } \sigma s] = \omega \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} \sigma s \rangle$   
 $\langle \text{proof} \rangle$

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

$\langle \langle \langle A \rangle_{\text{singl} \leftrightarrow \text{list}} \rangle_{\text{list} \rightsquigarrow \text{singl}} = A \rangle \langle \langle \langle B \rangle_{\text{singl} \leftrightarrow \text{list}} \rangle_{\text{list} \rightsquigarrow \text{singl}} = B \rangle$   
 $\langle \text{proof} \rangle$

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

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

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

$\langle \text{bij-betw } (\lambda A. \langle A \rangle_{\text{singl} \leftrightarrow \text{list}}) \text{ UNIV} \{A. \text{length-1}_d A \wedge (\forall \sigma s a. \tau A [\text{hd } \sigma s] a = \tau A \sigma s a) \wedge (\forall \sigma s. \omega A [\text{hd } \sigma s] = \omega A \sigma s)\} \rangle$   
 $\langle \text{is } \langle \text{bij-betw } (\lambda A. \langle A \rangle_{\text{singl} \leftrightarrow \text{list}}) \text{ UNIV } ?S_d \rangle \rangle$   
 $\langle \text{bij-betw } (\lambda B. \langle B \rangle_{\text{singl} \leftrightarrow \text{list}}) \text{ UNIV} \{B. \text{length-1}_{nd} B \wedge (\forall \sigma s a. \tau B [\text{hd } \sigma s] a = \tau B \sigma s a) \wedge (\forall \sigma s. \omega B [\text{hd } \sigma s] = \omega B \sigma s)\} \rangle$   
 $\langle \text{proof} \rangle$

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

$\langle \text{bij-betw } (\lambda A. \langle A \rangle_{\text{list} \rightsquigarrow \text{singl}}) \{A. \text{length-1}_d A \wedge (\forall \sigma s a. \tau A [\text{hd } \sigma s] a = \tau A \sigma s a) \wedge (\forall \sigma s. \omega A [\text{hd } \sigma s] = \omega A \sigma s)\} \text{ UNIV} \rangle$   
 $\langle \text{bij-betw } (\lambda B. \langle B \rangle_{\text{list} \rightsquigarrow \text{singl}}) \{B. \text{length-1}_{nd} B \wedge (\forall \sigma s a. \tau B [\text{hd } \sigma s] a = \tau B \sigma s a) \wedge (\forall \sigma s. \omega B [\text{hd } \sigma s] = \omega B \sigma s)\} \text{ UNIV} \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma**  $\mathcal{R}$ -base-trans[simp]:  $\langle \mathcal{R}_d \ (\lceil \tau = \lambda \sigma a. \diamond, \omega = \lambda \sigma. \diamond, \dots = \text{some} \rceil) = (\lambda \sigma. \{\sigma\}) \rangle$

$\langle \mathcal{R}_{nd} (\tau = \lambda \sigma a. \{\}, \omega = \lambda \sigma. \{\}, \dots = some) = (\lambda \sigma. \{\sigma\}) \rangle$   
 $\langle proof \rangle$

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

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

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

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

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

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

**corollary** *indep-enabl- $\omega$ -useless* [*simp*] :  
 $\langle indep\text{-enabl } (A_0(\omega := some\text{-}\omega)) \sigma_0 E A_1 \sigma_1 \longleftrightarrow indep\text{-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle indep\text{-enabl } A_0 \sigma_0 E (A_1(\omega := some\text{-}\omega)) \sigma_1 \longleftrightarrow indep\text{-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle proof \rangle$

**method**  $\mathcal{R}$ -subset-method **uses** *defs opt induct init simps* =  
*induct rule: induct, auto simp add: init defs  $\varepsilon$ -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- $\sigma$ -to- $\sigma s$ -description :  $\langle \mathcal{R}_{nd} \langle B \rangle_{\sigma \rightarrow \sigma s} [\sigma] = \{[\sigma'] \mid \sigma'. \sigma' \in \mathcal{R}_{nd}$

$B \sigma \rangle$   
 $\langle proof \rangle$

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

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

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

**lemma** length- $\mathcal{R}_d$ -from- $\sigma$ -to- $\sigma s$ :  
 $\langle \sigma s' \in \mathcal{R}_d \ d \langle A \rangle_{\sigma \hookrightarrow \sigma s} \ \sigma s \implies \sigma s' = \sigma s \vee length \ \sigma s' = 1 \rangle$   
 $\langle proof \rangle$

**lemma** length- $\mathcal{R}_{nd}$ -from- $\sigma$ -to- $\sigma s$ :  
 $\langle \sigma s' \in \mathcal{R}_{nd} \ nd \langle B \rangle_{\sigma \hookrightarrow \sigma s} \ \sigma s \implies \sigma s' = \sigma s \vee length \ \sigma s' = 1 \rangle$   
 $\langle proof \rangle$

**lemma** length- $\mathcal{R}_d$ -from-singl-to-list:  
 $\langle \sigma s' \in \mathcal{R}_d \ d \langle A \rangle_{singl \hookrightarrow list} \ \sigma s \implies \sigma s' = \sigma s \vee length \ \sigma s' = 1 \rangle$   
 $\langle proof \rangle$

**lemma** length- $\mathcal{R}_{nd}$ -from-singl-to-list:  
 $\langle \sigma s' \in \mathcal{R}_{nd} \ nd \langle B \rangle_{singl \hookrightarrow list} \ \sigma s \implies \sigma s' = \sigma s \vee length \ \sigma s' = 1 \rangle$   
 $\langle proof \rangle$

### 3.3 Normalization

#### 3.3.1 Non-deterministic Case

First version, without final state notion

**abbreviation**  $P$ -nd-step ::  $\langle [(\sigma, 'a) \ enabl, (\sigma, 'a) \ trans_{nd}, \sigma \Rightarrow ('a, 'r) \ process_{ptick}, \sigma] \Rightarrow ('a, 'r) \ process_{ptick} \rangle$   
**where**  $\langle P$ -nd-step  $\varepsilon_A \ \tau_A \ X \ \sigma \equiv \square \ e \in \varepsilon_A \ \sigma \rightarrow \sqcap \ \sigma' \in \tau_A \ \sigma \ e. \ X \ \sigma' \rangle$

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

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

**lemma** *P-nd-step-cont* [simp] :  $\langle \forall \sigma a. \text{finite } (\tau_A \sigma a) \implies \text{cont } (P\text{-nd-step } \varepsilon_A \tau_A) \rangle \langle \text{proof} \rangle$

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

**lemma** *P-nd-step-cont-bis* [simp] :  $\langle \text{finite-trans } A \implies \text{cont } (P\text{-nd-step } (\varepsilon A) (\tau A)) \rangle \langle \text{proof} \rangle$

**lemma** *P-nd-rec*:  $\langle P\langle A \rangle_{nd} = (\lambda \sigma. P\text{-nd-step } (\varepsilon A) (\tau A) P\langle A \rangle_{nd} \sigma) \rangle \langle \text{proof} \rangle$

**lemma** *P-nd-is-fix* :  $\langle \text{finite-trans } A \implies P\langle A \rangle_{nd} = (\mu X. P\text{-nd-step } (\varepsilon A) (\tau A) X) \rangle \langle \text{proof} \rangle$

**lemma** *non-destructive-imp-restriction-cont* [simp] :  
 $\langle \text{non-destructive } f \implies \text{restriction-cont } f \rangle \langle \text{proof} \rangle$

**lemma** *P-nd- $\omega$ -useless*:  $\langle P\langle A \rangle_{nd} = P\langle A(\omega := \text{some-}\omega) \rangle_{nd} \rangle \langle \text{proof} \rangle$

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

**lemma** *P-nd-induct* [case-names adm base step] :  
 $\langle \text{adm}_\downarrow P \implies P \sigma \implies (\bigwedge X. P X \implies P (P\text{-nd-step } (\varepsilon A) (\tau A) X)) \implies P P\langle A \rangle_{nd} \rangle \langle \text{proof} \rangle$

**lemma** *P-nd-induct-iterated* [consumes 1, case-names adm base step] :  
 $\langle [\!| 0 < k; \text{adm}_\downarrow P; P \sigma; \bigwedge X. P X \implies P ((P\text{-nd-step } (\varepsilon A) (\tau A) \overset{\sim}{\sim} k) X) \!|] \implies P P\langle A \rangle_{nd} \rangle \langle \text{proof} \rangle$

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

**abbreviation** *P<sub>SKIPS</sub>-nd-step* ::

$\langle [(\sigma, 'a) \text{enabl}, (\sigma, 'a) \text{trans}_{nd}, \sigma \Rightarrow 'r \text{set}, \sigma \Rightarrow ('a, 'r) \text{process}_{ptick}, \sigma] \Rightarrow ('a, 'r) \text{process}_{ptick} \rangle$

**where**  $\langle P_{SKIPS}\text{-nd-step } \varepsilon_A \tau_A \omega_A X \sigma \equiv \text{if } \omega_A \sigma = \{\} \text{ then } P\text{-nd-step } \varepsilon_A \tau_A X \sigma \text{ else } SKIPS (\omega_A \sigma) \rangle$

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

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

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

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

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

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

**lemma**  $P_{SKIPS}\text{-nd-rec} : \langle P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} = (\lambda \sigma. P_{SKIPS}\text{-nd-step} (\varepsilon A) (\tau A) (\omega A) P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} \sigma) \rangle \langle \text{proof} \rangle$

**lemma**  $P_{SKIPS}\text{-nd-is-fix} : \langle \text{finite-trans} A \Longrightarrow P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} = (\mu X. P_{SKIPS}\text{-nd-step} (\varepsilon A) (\tau A) (\omega A) X) \rangle \langle \text{proof} \rangle$

**lemma**  $P_{SKIPS}\text{-nd-induct} [case-names adm \text{ base step}] : \langle \text{adm}_\downarrow P \Longrightarrow P \sigma \Longrightarrow (\bigwedge X. P X \Longrightarrow P (P_{SKIPS}\text{-nd-step} (\varepsilon A) (\tau A) (\omega A) X)) \Longrightarrow P P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} \rangle \langle \text{proof} \rangle$

**lemma**  $P_{SKIPS}\text{-nd-induct-iterated} [consumes 1, case-names adm \text{ base step}] : \langle \llbracket 0 < k; \text{adm}_\downarrow P; P \sigma; \bigwedge X. P X \Longrightarrow P ((P_{SKIPS}\text{-nd-step} (\varepsilon A) (\tau A) (\omega A) \overset{\sim}{\sim} k) X) \rrbracket \Longrightarrow P P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} \rangle \langle \text{proof} \rangle$

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

**lemma**  $P_{SKIPS}\text{-nd-empty-}\varrho : \langle \varrho A = \{\} \Longrightarrow P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} = P\langle\!\langle\!A\!\rangle\!\rangle_{nd} \rangle \langle \text{proof} \rangle$

**lemma**  $P_{SKIPS}\text{-nd-updated-}\omega : \langle P\langle\!\langle\!A\!\rangle\!\rangle_{nd} = P_{SKIPS}\langle\!\langle\!A(\omega := \lambda \sigma. \{\})\!\rangle\!\rangle_{nd} \rangle \langle \text{proof} \rangle$

**lemma**  $P_{SKIPS}\text{-nd-empty-}\varrho\text{-inter-}\mathcal{R}_{nd} : \langle P_{SKIPS}\langle\!\langle\!A\!\rangle\!\rangle_{nd} \sigma = P\langle\!\langle\!A\!\rangle\!\rangle_{nd} \sigma \rangle \text{ if } \langle \varrho A \cap \mathcal{R}_{nd} A \sigma = \{\} \rangle \langle \text{proof} \rangle$

**lemma** *P<sub>SKIPS</sub>-nd-rec-notin- $\varrho$* :

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

**lemma** *P<sub>SKIPS</sub>-nd-rec-in- $\varrho$* :  $\langle \sigma \in \varrho A \implies P_{SKIPS}\langle\langle A \rangle\rangle_{nd} \sigma = SKIPS (\omega A \sigma) \rangle$   
 $\langle \text{proof} \rangle$

### 3.3.2 Deterministic Case

First version, without final state notion.

**abbreviation** *P-d-step* ::  $\langle [(\sigma, 'a) \text{enabl}, (\sigma, 'a) \text{trans}_d, ' \sigma \Rightarrow ('a, 'r) \text{process}_{ptick}, ' \sigma \Rightarrow ('a, 'r) \text{process}_{ptick}] \rangle$   
**where**  $\langle P\text{-d-step} \varepsilon_A \tau_A X s \equiv \square e \in \varepsilon_A s \rightarrow X [\tau_A s e] \rangle$

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

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

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

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

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

**lemma** *P-d-rec*:  $\langle P\langle\langle A \rangle\rangle_d = (\lambda s. P\text{-d-step} (\varepsilon A) (\tau A) P\langle\langle A \rangle\rangle_d s) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-d-is-fix* :  $\langle P\langle\langle A \rangle\rangle_d = (\mu X. P\text{-d-step} (\varepsilon A) (\tau A) X) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-d- $\omega$ -useless*:  $\langle P\langle\langle A \rangle\rangle_d = P\langle\langle A(\omega := \text{some-}\omega) \rangle\rangle_d \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma** *P-d-induct [case-names adm base step]* :  
 $\langle [ \text{adm}_\downarrow P; P \sigma; \bigwedge X. P X \implies P (P\text{-d-step} (\varepsilon A) (\tau A) X) ] \implies P P\langle\langle A \rangle\rangle_d \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-d-induct-iterated [consumes 1, case-names adm base step]* :

$\langle \llbracket 0 < k; adm_{\downarrow} P; P \sigma; \bigwedge X. P X \implies P ((P\text{-}d\text{-}step (\varepsilon A) (\tau A) \overset{\sim}{\sim} k) X) \rrbracket \implies P P\langle A \rangle_d \rangle$   
 $\langle proof \rangle$

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

**abbreviation**  $P_{SKIP\text{-}d\text{-}step} ::$

$\langle [(\sigma, 'a) \text{ enabl}, (' \sigma, 'a) \text{ trans}_d, ' \sigma \Rightarrow 'r \text{ option}, ' \sigma \Rightarrow ('a, 'r) \text{ process}_{ptick}, ' \sigma \Rightarrow ('a, 'r) \text{ process}_{ptick}] \rangle$

**where**  $\langle P_{SKIP\text{-}d\text{-}step \varepsilon_A \tau_A \omega_A X \sigma \equiv \text{case } \omega_A \sigma \text{ of } [r] \Rightarrow SKIP r \mid \diamond \Rightarrow P\text{-}d\text{-}step \varepsilon_A \tau_A X \sigma \rangle$

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

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

**lemma**  $P_{SKIP\text{-}d\text{-}step\text{-}constructive[simp]$ :  $\langle \text{constructive } (P_{SKIP\text{-}d\text{-}step \varepsilon_A \tau_A \mathcal{S}_{FA}}) \rangle$

$\langle proof \rangle$

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

**lemma**  $P_{SKIP\text{-}d\text{-}step\text{-}cont[simp]$ :  $\langle \text{cont } (P_{SKIP\text{-}d\text{-}step \varepsilon_A \tau_A \mathcal{S}_{FA}}) \rangle$

$\langle proof \rangle$

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

**lemma**  $P_{SKIP\text{-}d\text{-}rec$ :  $\langle P_{SKIP\text{-}d} \langle A \rangle_d = (\lambda \sigma. P_{SKIP\text{-}d\text{-}step (\varepsilon A) (\tau A) (\omega A) P_{SKIP\text{-}d} \langle A \rangle_d \sigma) \rangle$

$\langle proof \rangle$

**lemma**  $P_{SKIP\text{-}d\text{-}is\text{-}fix$  :  $\langle P_{SKIP\text{-}d} \langle A \rangle_d = (\mu X. P_{SKIP\text{-}d\text{-}step (\varepsilon A) (\tau A) (\omega A) X) \rangle$

$\langle proof \rangle$

**lemma**  $P_{SKIP\text{-}d\text{-}induct [case\text{-}names \text{ adm base step}]$  :

$\langle adm_{\downarrow} P \implies P \sigma \implies (\bigwedge X. P X \implies P (P_{SKIP\text{-}d\text{-}step (\varepsilon A) (\tau A) (\omega A) X)) \implies P P_{SKIP\text{-}d} \langle A \rangle_d \rangle$

$\langle proof \rangle$

**lemma**  $P_{SKIP\text{-}d\text{-}induct\text{-}iterated [consumes 1, case\text{-}names \text{ adm base step}]$  :

$\langle \llbracket 0 < k; adm_{\downarrow} P; P \sigma; \bigwedge X. P X \implies P ((P_{SKIP\text{-}d\text{-}step (\varepsilon A) (\tau A) (\omega A) \overset{\sim}{\sim} k) X) \rrbracket \implies P P_{SKIP\text{-}d} \langle A \rangle_d \rangle$

$\langle proof \rangle$

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

**lemma**  $P_{SKIP\text{S-d-empty-}\varrho}$  :  $\langle \varrho A = \{\} \implies P_{SKIP\text{S}}\langle\langle A \rangle\rangle_d = P\langle\langle A \rangle\rangle_d \rangle$   
*<proof>*

**lemma**  $P_{SKIP\text{S-d-updated-}\omega}$  :  $\langle P\langle\langle A \rangle\rangle_d = P_{SKIP\text{S}}\langle\langle A(\omega := \lambda\sigma. \diamond) \rangle\rangle_d \rangle$   
*<proof>*

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

**lemma**  $P_{SKIP\text{S-d-rec-notin-}\varrho}$  :  
 $\langle \sigma \notin \varrho A \implies P_{SKIP\text{S}}\langle\langle A \rangle\rangle_d \sigma = P\text{-d-step}(\varepsilon A)(\tau A) P_{SKIP\text{S}}\langle\langle A \rangle\rangle_d \sigma \rangle$   
*<proof>*

**lemma**  $P_{SKIP\text{S-d-rec-in-}\varrho}$  :  $\langle \sigma \in \varrho A \implies P_{SKIP\text{S}}\langle\langle A \rangle\rangle_d \sigma = SKIP[\omega A \sigma] \rangle$   
*<proof>*

### 3.3.3 Link between deterministic and non-deterministic ProcOmata

**lemma**  $P_{SKIP\text{S-nd-from-det-to-ndet-is-}P_{SKIP\text{S-d}}$  :  $\langle P_{SKIP\text{S}}\langle\langle\langle A \rangle\rangle_{d \hookrightarrow nd} \rangle_{nd} = P_{SKIP\text{S}}\langle\langle A \rangle\rangle_d \rangle$   
*<proof>*

**corollary**  $P\text{-nd-from-det-to-ndet-is-}P\text{-d}$  :  $\langle P\langle\langle\langle A \rangle\rangle_{d \hookrightarrow nd} \rangle_{nd} = P\langle\langle A \rangle\rangle_d \rangle$   
*<proof>*

### 3.3.4 Prove Equality between ProcOmata

**This is the easiest method we can think about.**

**lemma**  $P\text{-d-eqI}$  :  $\langle (\bigwedge \sigma a. \tau A \sigma a = \tau B \sigma a) \implies P\langle\langle A \rangle\rangle_d = P\langle\langle B \rangle\rangle_d \rangle$   
*<proof>*

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

**lemma**  $P_{SKIP\text{S-d-eqI}}$  :  
 $\langle (\bigwedge \sigma a. \sigma \notin \varrho A \implies \tau A \sigma a = \tau B \sigma a) \implies (\bigwedge \sigma. \omega A \sigma = \omega B \sigma) \implies P_{SKIP\text{S}}\langle\langle A \rangle\rangle_d = P_{SKIP\text{S}}\langle\langle B \rangle\rangle_d \rangle$   
*<proof>*

**lemma**  $P_{SKIP\text{S-nd-eqI}}$  :  
 $\langle (\bigwedge \sigma a. \sigma \notin \varrho A \implies \tau A \sigma a = \tau B \sigma a) \implies (\bigwedge \sigma. \omega A \sigma = \omega B \sigma) \implies P_{SKIP\text{S}}\langle\langle A \rangle\rangle_{nd} = P_{SKIP\text{S}}\langle\langle B \rangle\rangle_{nd} \rangle$   
*<proof>*

We establish now a much more powerful theorem.

**theorem** *P<sub>SKIP</sub>S-nd-eqI-strong*:

assumes *inj-on-f* :  $\langle \text{inj-on } f \ (\mathcal{R}_{nd} \ A_0 \ \sigma_0) \rangle$   
and *eq-trans* :  $\langle \bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_{nd} \ A_0 \ \sigma_0 \implies \tau \ A_1 \ (f \ \sigma_0') \ a = f \ ' \ (\tau \ A_0 \ \sigma_0' \ a) \rangle$   
and *eq-fin* :  $\langle \bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_{nd} \ A_0 \ \sigma_0 \implies \omega \ A_1 \ (f \ \sigma_0') \ a = \omega \ A_0 \ \sigma_0' \ a \rangle$   
shows  $\langle P_{SKIP}S\langle A_0 \rangle_{nd} \ \sigma_0 = P_{SKIP}S\langle A_1 \rangle_{nd} \ (f \ \sigma_0) \rangle$   
*<proof>*

**theorem** *P-nd-eqI-strong*:

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

**theorem** *P<sub>SKIP</sub>S-d-eqI-strong*:

assumes  $\langle \text{inj-on } f \ (\mathcal{R}_d \ A_0 \ \sigma_0) \rangle$   
and  $\langle \bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \tau \ A_1 \ (f \ \sigma_0') \ a = \text{map-option } f \ (\tau \ A_0 \ \sigma_0' \ a) \rangle$   
and  $\langle \bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \omega \ A_1 \ (f \ \sigma_0') \ a = \omega \ A_0 \ \sigma_0' \ a \rangle$   
shows  $\langle P_{SKIP}S\langle A_0 \rangle_d \ \sigma_0 = P_{SKIP}S\langle A_1 \rangle_d \ (f \ \sigma_0) \rangle$   
*<proof>*

**theorem** *P-d-eqI-strong*:

$\langle [\text{inj-on } f \ (\mathcal{R}_d \ A_0 \ \sigma_0);$   
 $\bigwedge \sigma_0' \ a. \ \sigma_0' \in \mathcal{R}_d \ A_0 \ \sigma_0 \implies \tau \ A_1 \ (f \ \sigma_0') \ a = \text{map-option } f \ (\tau \ A_0 \ \sigma_0' \ a)] \rangle$   
 $\implies P\langle A_0 \rangle_d \ \sigma_0 = P\langle A_1 \rangle_d \ (f \ \sigma_0) \rangle$   
*<proof>*

**lemmas** *P<sub>SKIP</sub>S-nd-eqI-strong-id* = *P<sub>SKIP</sub>S-nd-eqI-strong*[*of id, simplified*]

and *P<sub>SKIP</sub>S-d-eqI-strong-id* = *P<sub>SKIP</sub>S-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<sub>SKIP</sub>S-nd-from-σ-to-σs-is-P<sub>SKIP</sub>S-nd* :  $\langle P_{SKIP}S\langle A \rangle_{nd} \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_{nd} \ [\sigma]$   
=  $P_{SKIP}S\langle A \rangle_{nd} \ \sigma$

*<proof>*

**corollary** *P<sub>SKIP</sub>S-d-from-σ-to-σs-is-P<sub>SKIP</sub>S-d* :  $\langle P_{SKIP}S\langle A \rangle_d \langle A \rangle_{\sigma \mapsto \sigma s} \rangle_d \ [\sigma]$  =  
 $P_{SKIP}S\langle A \rangle_d \ \sigma$

*<proof>*

**corollary**  $P$ -nd-from- $\sigma$ -to- $\sigma$ s-is- $P$ -nd :  $\langle P\langle\langle A \rangle\rangle_{\sigma \hookrightarrow \sigma s}\rangle_{nd} [\sigma] = P\langle A \rangle_{nd} \sigma$   
 ⟨proof⟩

**corollary**  $P$ -d-from- $\sigma$ -to- $\sigma$ s-is- $P$ -d :  $\langle P\langle\langle A \rangle\rangle_{\sigma \hookrightarrow \sigma s}\rangle_d [\sigma] = P\langle A \rangle_d \sigma$   
 ⟨proof⟩

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

**fun** *recursive-modifier-fun<sub>d</sub>* ::  $\langle [(\sigma \times 'a) \Rightarrow \sigma \text{ option}, ((\sigma \times 'a) \times \sigma \text{ option}) \text{ list}] \Rightarrow (\sigma \times 'a) \Rightarrow \sigma \text{ option} \rangle$   
**where**  $\langle \text{recursive-modifier-fun}_d f [] = f \rangle$   
 $| \langle \text{recursive-modifier-fun}_d f (((s, e), t) \# \mathcal{G}_A) = \text{recursive-modifier-fun}_d (f((s, e) := t)) \mathcal{G}_A \rangle$

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

**lemma**  $\varepsilon$ -det-breaker:

$\langle \varepsilon (\tau = \text{curry} (g((\sigma'::\sigma, a) \mapsto \sigma''::\sigma)), \omega = \text{some-}\omega, \dots = \text{some-more}) \rangle \sigma =$   
 $\langle \text{if } \sigma = \sigma' \text{ then } \{a\} \cup \varepsilon (\tau = \text{curry } g, \omega = \text{some-}\omega) \sigma' \text{ else } \varepsilon (\tau = \text{curry } g, \omega = \text{some-}\omega) \sigma \rangle$   
 ⟨proof⟩

**method**  $\varepsilon$ -det-calc = (unfold recursive-modifier-fun<sub>d</sub>.simps  $\varepsilon$ -det-breaker, simp cong: if-cong)[1]

**method**  $\tau$ -det-calc = (unfold recursive-modifier-fun<sub>d</sub>.simps, simp cong: if-cong)[1]

**lemma** *bij-Renaming-PSKIPS-nd* :

**fixes**  $A :: \langle (\sigma, 'a, 'r, 's) A_{nd}\text{-scheme} \rangle$  **and**  $f :: \langle 'a \Rightarrow 'b \rangle$  **and**  $g :: \langle 'r \Rightarrow 's \rangle$   
**assumes**  $\langle \text{bij } f \rangle$   
**defines**  $B\text{-def} : \langle B \equiv (\tau = \lambda\sigma. b. \tau A \sigma (\text{inv } f b), \omega = \lambda\sigma. g (\omega A \sigma)) \rangle$   
**shows**  $\langle \text{Renaming} (PSKIPS\langle A \rangle_{nd} \sigma) f g = PSKIPS\langle B \rangle_{nd} \sigma \rangle$  (**is**  $\langle ?lhs \sigma = \cdot \rangle$ )  
 ⟨proof⟩

**lemma** *bij-Renaming-PSKIPS-d* :

$\langle \text{bij } f \implies \text{Renaming} (PSKIPS\langle A \rangle_d \sigma) f g =$   
 $PSKIPS\langle (\tau = \lambda\sigma. b. \tau A \sigma (\text{inv } f b), \omega = \lambda\sigma. \text{map-option } g (\omega A \sigma)) \rangle_d \sigma \rangle$

*<proof>*

**lemma** *RenamingTick-PSKIPS-nd* :

$\langle \text{RenamingTick } (PSKIPS\langle A \rangle_{nd} \sigma) g = PSKIPS\langle (\tau = \tau A, \omega = \lambda\sigma. g \text{ ' } \omega A \sigma) \rangle_{nd} \sigma \rangle$   
*<proof>*

**lemma** *RenamingTick-PSKIPS-d* :

$\langle \text{RenamingTick } (PSKIPS\langle A \rangle_d \sigma) g = PSKIPS\langle (\tau = \tau A, \omega = \lambda\sigma. \text{map-option } g (\omega A \sigma)) \rangle_d \sigma \rangle$   
*<proof>*





## Chapter 5

# Combining Automata for Synchronization Product

### 5.1 Definitions

#### 5.1.1 General Patterns

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

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

**definition** *combine-Sync- $\varepsilon$*  ::

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

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

**lemma** *combine-Sync- $\varepsilon$ -def-bis* :

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

**fun** *combine<sub>d</sub>-Sync- $\tau$*  ::

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

**and** *combine<sub>nd</sub>-Sync- $\tau$*  ::

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

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

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

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

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

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

## 5.1.2 Specializations

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

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

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

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

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

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

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

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

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

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

**bundle** *combine<sub>nd</sub>-Sync-syntax* **begin**

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

**end**

**unbundle** *combine<sub>nd</sub>-Sync-syntax*

## 5.2 First Properties

**lemma** *finite-trans-combine<sub>nd</sub>-Sync-simps* [*simp*] :

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

**lemma**  $\varepsilon$ -*combine<sub>Pairlist</sub>-Sync*:

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

**lemma**  $\varepsilon$ -*combine<sub>Pair</sub>-Sync*:

$\langle \varepsilon \langle \langle A_0 \ d \otimes [E]_{Pair} \ A_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ \text{fst} \ \text{snd} \ \sigma s \rangle$   
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes [E]_{Pair} \ B_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ B_0 \ E \ B_1 \ \text{fst} \ \text{snd} \ \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -*combine<sub>ListslenL</sub>-Sync*:

$\langle \varepsilon \langle \langle A_0 \ d \otimes [len_0, E]_{ListslenL} \ A_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ A_0 \ E \ A_1 \ (\text{take } len_0) \ (\text{drop } len_0) \ \sigma s \rangle$   
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes [len_0, E]_{ListslenL} \ B_1 \rangle \rangle \sigma s = \text{combine-Sync-}\varepsilon \ B_0 \ E \ B_1 \ (\text{take } len_0) \ (\text{drop } len_0) \ \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -*combine<sub>Rlist</sub>-Sync*:

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

**lemma**  $\varrho$ -combine $_{Pairlist}$ -Sync:

$\langle \varrho \langle \langle A_0 \ d \otimes [E]_{Pairlist} A_1 \rangle \rangle = \{ \sigma s. hd \ \sigma s \in \varrho A_0 \wedge hd \ (tl \ \sigma s) \in \varrho A_1 \wedge \omega A_0 \ (hd \ \sigma s) = \omega A_1 \ (hd \ (tl \ \sigma s)) \} \rangle$   
 $\langle \varrho \langle \langle B_0 \ nd \otimes [E]_{Pairlist} B_1 \rangle \rangle = \{ \sigma s. hd \ \sigma s \in \varrho B_0 \wedge hd \ (tl \ \sigma s) \in \varrho B_1 \wedge \omega B_0 \ (hd \ \sigma s) \cap \omega B_1 \ (hd \ (tl \ \sigma s)) \neq \{ \} \} \rangle$   
 $\langle proof \rangle$

**lemma**  $\varrho$ -combine $_{Pair}$ -Sync:

$\langle \varrho \langle \langle A_0 \ d \otimes [E]_{Pair} A_1 \rangle \rangle = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho A_0 \wedge \sigma_1 \in \varrho A_1 \wedge \omega A_0 \ \sigma_0 = \omega A_1 \ \sigma_1 \} \rangle$   
 $\langle \varrho \langle \langle B_0 \ nd \otimes [E]_{Pair} B_1 \rangle \rangle = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho B_0 \wedge \sigma_1 \in \varrho B_1 \wedge \omega B_0 \ \sigma_0 \cap \omega B_1 \ \sigma_1 \neq \{ \} \} \rangle$   
 $\langle proof \rangle$

**lemma**  $\varrho$ -combine $_{ListslenL}$ -Sync:

$\langle \varrho \langle \langle A_0 \ d \otimes [len_0, E]_{ListslenL} A_1 \rangle \rangle =$   
 $\{ \sigma s. take \ len_0 \ \sigma s \in \varrho A_0 \wedge drop \ len_0 \ \sigma s \in \varrho A_1 \wedge \omega A_0 \ (take \ len_0 \ \sigma s) = \omega A_1 \ (drop \ len_0 \ \sigma s) \} \rangle$   
 $\langle \varrho \langle \langle B_0 \ nd \otimes [len_0, E]_{ListslenL} B_1 \rangle \rangle =$   
 $\{ \sigma s. take \ len_0 \ \sigma s \in \varrho B_0 \wedge drop \ len_0 \ \sigma s \in \varrho B_1 \wedge \omega B_0 \ (take \ len_0 \ \sigma s) \cap \omega B_1 \ (drop \ len_0 \ \sigma s) \neq \{ \} \} \rangle$   
 $\langle proof \rangle$

**lemma**  $\varrho$ -combine $_{Rlist}$ -Sync:

$\langle \varrho \langle \langle A_0 \ d \otimes [E]_{Rlist} A_1 \rangle \rangle =$   
 $\{ \sigma s. hd \ \sigma s \in \varrho A_0 \wedge tl \ \sigma s \in \varrho A_1 \wedge \omega A_0 \ (hd \ \sigma s) = \omega A_1 \ (tl \ \sigma s) \} \rangle$   
 $\langle \varrho \langle \langle B_0 \ nd \otimes [E]_{Rlist} B_1 \rangle \rangle =$   
 $\{ \sigma s. hd \ \sigma s \in \varrho B_0 \wedge tl \ \sigma s \in \varrho B_1 \wedge \omega B_0 \ (hd \ \sigma s) \cap \omega B_1 \ (tl \ \sigma s) \neq \{ \} \} \rangle$   
 $\langle proof \rangle$

**lemma** combine-Sync- $\tau$  [simp] :

$\langle combine_d\text{-Sync-}\tau \ (A_0(\omega := some\ \omega_0)) \ E \ (A_1(\omega := some\ \omega_1)) = combine_d\text{-Sync-}\tau \ A_0 \ E \ A_1 \rangle$   
 $\langle combine_{nd}\text{-Sync-}\tau \ (B_0(\omega := some\ \omega_0')) \ E \ (B_1(\omega := some\ \omega_1')) = combine_{nd}\text{-Sync-}\tau \ B_0 \ E \ B_1 \rangle$   
**for**  $A :: \langle ('\sigma, 'a, '\sigma \text{ option}, 'r \text{ option}, '\alpha) \ A\text{-scheme} \rangle$   
**and**  $B :: \langle ('\sigma, 'a, '\sigma \text{ set}, 'r \text{ set}, '\alpha) \ A\text{-scheme} \rangle$   
 $\langle proof \rangle$

## 5.3 Reachability

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

$\langle \mathcal{R}_d \langle \langle A_0 \ d \otimes [len_0, E]_{ListslenL} A_1 \rangle \rangle \ (s_0 @ s_1) \subseteq \{ t_0 @ t_1 \mid t_0 \ t_1. t_0 \in \mathcal{R}_d \ A_0 \ s_0 \wedge t_1 \in \mathcal{R}_d \ A_1 \ s_1 \} \rangle$  (**is**  $\langle ?S_A \subseteq \cdot \rangle$ )  
**if** same-length- $\mathcal{R}_d$ :  $\langle \bigwedge t_0. t_0 \in \mathcal{R}_d \ A_0 \ s_0 \implies length \ t_0 = len_0 \rangle$   
 $\langle proof \rangle$

**lemma**  $\mathcal{R}_{nd}$ -combine $_{ndListslenL}$ -Sync-subset:

$\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [len_0, E]_{ListslenL} B_1 \rangle \rangle [s_0 @ s_1] \subseteq \{t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$  (is  $\langle ?S_B \subseteq - \rangle$ )

if same-length- $\mathcal{R}_{nd}$ :  $\langle \bigwedge t_0. t_0 \in \mathcal{R}_{nd} B_0 s_0 \implies \text{length } t_0 = len_0 \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\mathcal{R}_d$ -combine $_dPairlist$ -Sync-subset:

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

and  $\mathcal{R}_{nd}$ -combine $_{ndPairlist}$ -Sync-subset:

$\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [E]_{Pairlist} B_1 \rangle \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$  (is  $\langle ?S_B \subseteq - \rangle$ )

$\langle \text{proof} \rangle$

**lemma**  $\mathcal{R}_d$ -combine $_dPair$ -Sync-subset:

$\langle \mathcal{R}_d \langle \langle A_0 \text{ d} \otimes [E]_{Pair} A_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_d A_0 s_0 \times \mathcal{R}_d A_1 s_1 \rangle$  (is  $\langle ?S_A \subseteq - \rangle$ )

and  $\mathcal{R}_{nd}$ -combine $_{ndPair}$ -Sync-subset:

$\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [E]_{Pair} B_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_{nd} B_0 s_0 \times \mathcal{R}_{nd} B_1 s_1 \rangle$  (is  $\langle ?S_B \subseteq - \rangle$ )  
 $\langle \text{proof} \rangle$

**lemma**  $\mathcal{R}_d$ -combine $_dRlist$ -Sync-subset:

$\langle \mathcal{R}_d \langle \langle A_0 \text{ d} \otimes [E]_{Rlist} A_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \sigma t. t_0 \in \mathcal{R}_d A_0 s_0 \wedge \sigma t \in \mathcal{R}_d A_1 \sigma s\} \rangle$  (is  $\langle ?S_A \subseteq - \rangle$ )

and  $\mathcal{R}_{nd}$ -combine $_{ndRlist}$ -Sync-subset:

$\langle \mathcal{R}_{nd} \langle \langle B_0 \text{ nd} \otimes [E]_{Rlist} B_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \sigma t. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge \sigma t \in \mathcal{R}_{nd} B_1 \sigma s\} \rangle$  (is  $\langle ?S_B \subseteq - \rangle$ )

$\langle \text{proof} \rangle$

## 5.4 Normalization

**lemma**  $\omega$ -combine $_{Pairlist}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ d} \otimes [E]_{Pairlist} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} [s_0, s_1] = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \text{ nd} \otimes [E]_{Pairlist} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle [s_0, s_1] \rangle$

$\langle \text{proof} \rangle$

**lemma**  $\omega$ -combine $_{Pair}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ d} \otimes [E]_{Pair} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (s_0, s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \text{ nd} \otimes [E]_{Pair} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle (s_0, s_1) \rangle$

$\langle \text{proof} \rangle$

**lemma**  $\omega$ -combine $_{ListslenL}$ -Sync-behaviour:

$\langle \omega \langle \langle \langle A_0 \text{ d} \otimes [len_0, E]_{ListslenL} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 @ \sigma s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \text{ nd} \otimes [len_0, E]_{ListslenL} \langle \langle A_1 \rangle \rangle_{d \hookrightarrow nd} \rangle (\sigma s_0 @ \sigma s_1) \rangle$

$\langle \text{proof} \rangle$

**lemma**  $\omega$ -combine $_{Rlist}$ -Sync-behaviour:

$\langle \omega \langle \langle A_0 \ d \otimes [E]_{Rlist} \ A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0 \ \# \ \sigma s_1) = \omega \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Rlist} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (s_0 \ \# \ \sigma s_1) \rangle$   
*<proof>*

**lemma**  $\tau$ -combine<sub>Pairlist</sub>-Sync-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ s_0 \ \cap \ \varepsilon \ A_1 \ s_1 \ \subseteq \ E \implies$   
 $\tau \langle \langle A_0 \ d \otimes [E]_{Pairlist} \ A_1 \rangle \rangle_{d \hookrightarrow nd} [s_0, s_1] \ e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Pairlist} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle [s_0, s_1] \ e \rangle$   
*<proof>*

**lemma**  $\tau$ -combine<sub>Pair</sub>-Sync-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ s_0 \ \cap \ \varepsilon \ A_1 \ s_1 \ \subseteq \ E \implies$   
 $\tau \langle \langle A_0 \ d \otimes [E]_{Pair} \ A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0, s_1) \ e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Pair} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (s_0, s_1) \ e \rangle$   
*<proof>*

**lemma**  $\tau$ -combine<sub>ListstlenL</sub>-Sync-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ \sigma s_0 \ \cap \ \varepsilon \ A_1 \ \sigma s_1 \ \subseteq \ E \implies \text{length } \sigma s_0 = \text{len}_0 \implies$   
 $\tau \langle \langle A_0 \ d \otimes [\text{len}_0, E]_{ListstlenL} \ A_1 \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 \ @ \ \sigma s_1) \ e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [\text{len}_0, E]_{ListstlenL} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (\sigma s_0 \ @ \ \sigma s_1) \ e \rangle$   
*<proof>*

**lemma**  $\tau$ -combine<sub>Rlist</sub>-Sync-behaviour-when-indep:

$\langle \varepsilon \ A_0 \ s_0 \ \cap \ \varepsilon \ A_1 \ \sigma s_1 \ \subseteq \ E \implies$   
 $\tau \langle \langle A_0 \ d \otimes [E]_{Rlist} \ A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0 \ \# \ \sigma s_1) \ e = \tau \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Rlist} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (s_0 \ \# \ \sigma s_1) \ e \rangle$   
*<proof>*

**method** *P<sub>SKIPs</sub>-when-indep-method* **uses** *R-d-subset* =

*fold P<sub>SKIPs</sub>-nd-from-det-to-ndet-is-P<sub>SKIPs</sub>-d,*  
*rule P<sub>SKIPs</sub>-nd-eqI-strong-id,*  
*unfold R<sub>nd</sub>-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 R<sub>nd</sub>-from-det-to-ndet,*  
*all <drule set-mp[OF R-d-subset, rotated]>*

**lemma** *P<sub>SKIPs</sub>-combine<sub>Pairlist</sub>-Sync-behaviour-when-indep*:

$\langle P_{SKIPs} \langle \langle A_0 \ d \otimes [E]_{Pairlist} \ A_1 \rangle \rangle_d [s_0, s_1] = P_{SKIPs} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E]_{Pairlist} \ \langle A_1 \rangle_{d \hookrightarrow nd} \rangle_{nd} [s_0, s_1] \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$

*<proof>*

**lemma** *P-combine<sub>Pairlist</sub>-Sync-behaviour-when-indep:*

$\langle P\langle\langle A_0 \text{ d} \otimes [E]_{\text{Pairlist}} A_1 \rangle\rangle_d [s_0, s_1] = P\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [E]_{\text{Pairlist}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} [s_0, s_1] \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
*<proof>*

**lemma** *P<sub>SKIPS</sub>-combine<sub>Pair</sub>-Sync-behaviour-when-indep:*

$\langle P_{SKIPS}\langle\langle A_0 \text{ d} \otimes [E]_{\text{Pair}} A_1 \rangle\rangle_d (s_0, s_1) = P_{SKIPS}\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [E]_{\text{Pair}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} (s_0, s_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
*<proof>*

**lemma** *P-combine<sub>Pair</sub>-Sync-behaviour-when-indep:*

$\langle P\langle\langle A_0 \text{ d} \otimes [E]_{\text{Pair}} A_1 \rangle\rangle_d (s_0, s_1) = P\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [E]_{\text{Pair}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} (s_0, s_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
*<proof>*

**lemma** *P<sub>SKIPS</sub>-combine<sub>ListstlenL</sub>-Sync-behaviour-when-indep:*

$\langle P_{SKIPS}\langle\langle A_0 \text{ d} \otimes [len_0, E]_{\text{ListstlenL}} A_1 \rangle\rangle_d (\sigma s_0 \ @ \ \sigma s_1) = P_{SKIPS}\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [len_0, E]_{\text{ListstlenL}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} (\sigma s_0 \ @ \ \sigma s_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ \sigma s_0 \ E \ A_1 \ \sigma s_1 \rangle$  **and**  $\langle \wedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d \ A_0 \ \sigma s_0 \implies \text{length } \sigma t_0 = len_0 \rangle$   
*<proof>*

**lemma** *P-combine<sub>ListstlenL</sub>-Sync-behaviour-when-indep:*

$\langle P\langle\langle A_0 \text{ d} \otimes [len_0, E]_{\text{ListstlenL}} A_1 \rangle\rangle_d (\sigma s_0 \ @ \ \sigma s_1) = P\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [len_0, E]_{\text{ListstlenL}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} (\sigma s_0 \ @ \ \sigma s_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ \sigma s_0 \ E \ A_1 \ \sigma s_1 \rangle$  **and**  $\langle \wedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d \ A_0 \ \sigma s_0 \implies \text{length } \sigma t_0 = len_0 \rangle$   
*<proof>*

**lemma** *P<sub>SKIPS</sub>-combine<sub>Rlist</sub>-Sync-behaviour-when-indep:*

$\langle P_{SKIPS}\langle\langle A_0 \text{ d} \otimes [E]_{\text{Rlist}} A_1 \rangle\rangle_d (s_0 \ # \ \sigma s_1) = P_{SKIPS}\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [E]_{\text{Rlist}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} (s_0 \ # \ \sigma s_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ \sigma s_1 \rangle$   
*<proof>*

**lemma** *P-combine<sub>Rlist</sub>-Sync-behaviour-when-indep:*

$\langle P\langle\langle A_0 \text{ d} \otimes [E]_{\text{Rlist}} A_1 \rangle\rangle_d (s_0 \ # \ \sigma s_1) = P\langle\langle\langle A_0 \rangle\rangle_{d \rightarrow nd} \text{ nd} \otimes [E]_{\text{Rlist}} \langle\langle A_1 \rangle\rangle_{d \rightarrow nd} \rangle\rangle_{nd} (s_0 \ # \ \sigma s_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \ s_0 \ E \ A_1 \ \sigma s_1 \rangle$   
*<proof>*

## Chapter 6

# Compactification of Synchronization Product

### 6.1 Iterated Combine

#### 6.1.1 Definitions

**fun** *iterated-combine<sub>d</sub>-Sync* :: ⟨'e set ⇒ ('σ, 'e, 'r) A<sub>d</sub> list ⇒ ('σ list, 'e, 'r) A<sub>d</sub>⟩  
 (⟨⟨<sub>d</sub>⊗ [-] -⟩ [θ, θ]⟩  
**where** ⟨⟨<sub>d</sub>⊗ [E] []⟩ = (τ = λσs a. ◇, ω = λσs. ◇)⟩  
 | ⟨⟨<sub>d</sub>⊗ [E] [A<sub>0</sub>]⟩ = <sub>d</sub>⟨A<sub>0</sub>⟩<sub>σ↔σs</sub>⟩  
 | ⟨⟨<sub>d</sub>⊗ [E] A<sub>0</sub> # A<sub>1</sub> # As⟩ = ⟨A<sub>0</sub> <sub>d</sub>⊗ [E] *Rlist* ⟨<sub>d</sub>⊗ [E] A<sub>1</sub> # As⟩⟩⟩

**fun** *iterated-combine<sub>nd</sub>-Sync* :: ⟨'e set ⇒ ('σ, 'e, 'r) A<sub>nd</sub> list ⇒ ('σ list, 'e, 'r) A<sub>nd</sub>⟩  
 (⟨⟨<sub>nd</sub>⊗ [-] -⟩ [θ, θ]⟩  
**where** ⟨⟨<sub>nd</sub>⊗ [E] []⟩ = (τ = λσs a. {}, ω = λσs. {})⟩  
 | ⟨⟨<sub>nd</sub>⊗ [E] [A<sub>0</sub>]⟩ = <sub>nd</sub>⟨A<sub>0</sub>⟩<sub>σ↔σs</sub>⟩  
 | ⟨⟨<sub>nd</sub>⊗ [E] A<sub>0</sub> # A<sub>1</sub> # As⟩ = ⟨A<sub>0</sub> <sub>nd</sub>⊗ [E] *Rlist* ⟨<sub>nd</sub>⊗ [E] A<sub>1</sub> # As⟩⟩⟩

**lemma** *iterated-combine<sub>d</sub>-Sync-simps-bis*: ⟨As ≠ [] ⇒ ⟨<sub>d</sub>⊗ [E] A<sub>0</sub> # As⟩ = ⟨A<sub>0</sub> <sub>d</sub>⊗ [E] *Rlist* ⟨<sub>d</sub>⊗ [E] As⟩⟩⟩  
**and** *iterated-combine<sub>nd</sub>-Sync-simps-bis*: ⟨Bs ≠ [] ⇒ ⟨<sub>nd</sub>⊗ [E] B<sub>0</sub> # Bs⟩ = ⟨B<sub>0</sub> <sub>nd</sub>⊗ [E] *Rlist* ⟨<sub>nd</sub>⊗ [E] Bs⟩⟩⟩  
 ⟨*proof*⟩

**fun** *iterated-combine<sub>d</sub>-Sync-ε* :: ⟨('σ, 'e, 'r, 'α) A<sub>d</sub>-scheme list ⇒ 'e set ⇒ 'σ list ⇒ 'e set⟩  
**where** ⟨*iterated-combine<sub>d</sub>-Sync-ε* [] E σs = {}⟩  
 | ⟨*iterated-combine<sub>d</sub>-Sync-ε* [A<sub>0</sub>] E σs = ε A<sub>0</sub> (hd σs)⟩  
 | ⟨*iterated-combine<sub>d</sub>-Sync-ε* (A<sub>0</sub> # A<sub>1</sub> # As) E σs =  
   *combine-sets-Sync* (ε A<sub>0</sub> (hd σs)) E (*iterated-combine<sub>d</sub>-Sync-ε* (A<sub>1</sub> # As)  
 E (tl σs))⟩

**fun** *iterated-combine<sub>nd</sub>-Sync-ε* :: ⟨('σ, 'e, 'r, 'α) *A<sub>nd</sub>-scheme list* ⇒ 'e set ⇒ 'σ list ⇒ 'e set⟩  
**where** ⟨*iterated-combine<sub>nd</sub>-Sync-ε* [] *E* σ s = {}⟩  
| ⟨*iterated-combine<sub>nd</sub>-Sync-ε* [*A*<sub>0</sub>] *E* σ s = ε *A*<sub>0</sub> (*hd* σ s)⟩  
| ⟨*iterated-combine<sub>nd</sub>-Sync-ε* (*A*<sub>0</sub> # *A*<sub>1</sub> # *As*) *E* σ s =  
*combine-sets-Sync* (ε *A*<sub>0</sub> (*hd* σ s)) *E* (*iterated-combine<sub>nd</sub>-Sync-ε* (*A*<sub>1</sub> # *As*)  
*E* (*tl* σ s))⟩

**lemma** *iterated-combine<sub>d</sub>-Sync-ε-simps-bis*:  
⟨*As* ≠ [] ⇒ *iterated-combine<sub>d</sub>-Sync-ε* (*A*<sub>0</sub> # *As*) *E* σ s =  
*combine-sets-Sync* (ε *A*<sub>0</sub> (*hd* σ s)) *E* (*iterated-combine<sub>d</sub>-Sync-ε* *As* *E*  
(*tl* σ s))⟩  
**and** *iterated-combine<sub>nd</sub>-Sync-ε-simps-bis*:  
⟨*Bs* ≠ [] ⇒ *iterated-combine<sub>nd</sub>-Sync-ε* (*B*<sub>0</sub> # *Bs*) *E* σ s =  
*combine-sets-Sync* (ε *B*<sub>0</sub> (*hd* σ s)) *E* (*iterated-combine<sub>nd</sub>-Sync-ε* *Bs*  
*E* (*tl* σ s))⟩  
⟨*proof*⟩

## 6.1.2 First Results

**lemma** *ε-iterated-combine<sub>d</sub>-Sync*:  
⟨*length* σ s = *length* *As* ⇒ ε ⟨*d*⊗ [E] *As*⟩ σ s = *iterated-combine<sub>d</sub>-Sync-ε* *As* *E*  
σ s⟩  
⟨*proof*⟩

**lemma** *ε-iterated-combine<sub>nd</sub>-Sync*:  
⟨*length* σ s = *length* *Bs* ⇒ ε ⟨*nd*⊗ [E] *Bs*⟩ σ s = *iterated-combine<sub>nd</sub>-Sync-ε* *Bs*  
*E* σ s⟩  
⟨*proof*⟩

**lemma** *combine<sub>ListslenL</sub>-Sync-combine<sub>Rlist</sub>-Sync-eq*:  
⟨ε ⟨*d*⟨*A*<sub>0</sub>⟩<sub>σ↔σs</sub> *d*⊗ [1, E] *ListslenL* *A*<sub>1</sub>⟩ σ s = ε ⟨*A*<sub>0</sub> *d*⊗ [E] *Rlist* *A*<sub>1</sub>⟩ σ s⟩  
⟨τ ⟨*d*⟨*A*<sub>0</sub>⟩<sub>σ↔σs</sub> *d*⊗ [1, E] *ListslenL* *A*<sub>1</sub>⟩ (*s*<sub>0</sub> # σ s) *e* = τ ⟨*A*<sub>0</sub> *d*⊗ [E] *Rlist* *A*<sub>1</sub>⟩ (*s*<sub>0</sub>  
# σ s) *e*⟩  
⟨ε ⟨*nd*⟨*B*<sub>0</sub>⟩<sub>σ↔σs</sub> *nd*⊗ [1, E] *ListslenL* *B*<sub>1</sub>⟩ σ s = ε ⟨*B*<sub>0</sub> *nd*⊗ [E] *Rlist* *B*<sub>1</sub>⟩ σ s⟩  
⟨τ ⟨*nd*⟨*B*<sub>0</sub>⟩<sub>σ↔σs</sub> *nd*⊗ [1, E] *ListslenL* *B*<sub>1</sub>⟩ (*s*<sub>0</sub> # σ s) *e* = τ ⟨*B*<sub>0</sub> *nd*⊗ [E] *Rlist* *B*<sub>1</sub>⟩  
(*s*<sub>0</sub> # σ s) *e*⟩  
⟨*proof*⟩

**lemma** *combine<sub>Pairlist</sub>-Sync-and-iterated-combine<sub>nd</sub>-Sync-eq*:  
⟨ε ⟨*A*<sub>0</sub> *d*⊗ [E] *Pairlist* *A*<sub>1</sub>⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>] = ε ⟨*d*⊗ [E] [*A*<sub>0</sub>, *A*<sub>1</sub>]⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>]⟩  
⟨τ ⟨*A*<sub>0</sub> *d*⊗ [E] *Pairlist* *A*<sub>1</sub>⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>] *e* = τ ⟨*d*⊗ [E] [*A*<sub>0</sub>, *A*<sub>1</sub>]⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>] *e*⟩  
⟨ε ⟨*B*<sub>0</sub> *nd*⊗ [E] *Pairlist* *B*<sub>1</sub>⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>] = ε ⟨*nd*⊗ [E] [*B*<sub>0</sub>, *B*<sub>1</sub>]⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>]⟩  
⟨τ ⟨*B*<sub>0</sub> *nd*⊗ [E] *Pairlist* *B*<sub>1</sub>⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>] *e* = τ ⟨*nd*⊗ [E] [*B*<sub>0</sub>, *B*<sub>1</sub>]⟩ [*s*<sub>0</sub>, *s*<sub>1</sub>] *e*⟩  
⟨*proof*⟩

**lemmas** *combine<sub>Pairlist</sub>-Sync-and-combine<sub>Rlist</sub>-Sync-eq = combine<sub>Pairlist</sub>-Sync-and-iterated-combine<sub>nd</sub>-Sync-eq[simplified]*

### 6.1.3 Reachability

**lemma** *same-length- $\mathcal{R}_d$ -iterated-combine<sub>d</sub>-Sync-description:*  
 $\langle \text{length } \sigma s = \text{length } As \implies \sigma s' \in \mathcal{R}_d \langle \langle_d \otimes [E] \rangle As \rangle \sigma s \implies$   
 $\text{length } \sigma s' = \text{length } As \wedge (\forall i < \text{length } As. \sigma s' ! i \in \mathcal{R}_d (As ! i) (\sigma s ! i)) \rangle$   
*\langle proof \rangle*

**lemma** *same-length- $\mathcal{R}_{nd}$ -iterated-combine<sub>nd</sub>-Sync-description:*  
 $\langle \text{length } \sigma s = \text{length } Bs \implies \sigma s' \in \mathcal{R}_{nd} \langle \langle_{nd} \otimes [E] \rangle Bs \rangle \sigma s \implies$   
 $\text{length } \sigma s' = \text{length } Bs \wedge (\forall i < \text{length } Bs. \sigma s' ! i \in \mathcal{R}_{nd} (Bs ! i) (\sigma s ! i)) \rangle$   
*\langle proof \rangle*

### 6.1.4 Transmission of Properties

**lemma** *finite-trans-transmission-to-iterated-combine<sub>nd</sub>-Sync:*  
 $\langle (\bigwedge A. A \in \text{set } As \implies \text{finite-trans } A) \implies \text{finite-trans } \langle \langle_{nd} \otimes [E] \rangle As \rangle \rangle$   
*\langle proof \rangle*

**lemma**  *$\varrho$ -disjoint- $\varepsilon$ -transmission-to-iterated-combine<sub>d</sub>-Sync:*  
 $\langle (\bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A) \implies \varrho\text{-disjoint-}\varepsilon \langle \langle_d \otimes [E] \rangle As \rangle \rangle$   
*\langle proof \rangle*

**lemma**  *$\varrho$ -disjoint- $\varepsilon$ -transmission-to-iterated-combine<sub>nd</sub>-Sync:*  
 $\langle (\bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A) \implies \varrho\text{-disjoint-}\varepsilon \langle \langle_{nd} \otimes [E] \rangle As \rangle \rangle$   
*\langle proof \rangle*

**lemma** *at-most-1-elem-term-transmission-to-iterated-combine<sub>nd</sub>-Sync:*  
 $\langle (\bigwedge A. A \in \text{set } As \implies \text{at-most-1-elem-term } A) \implies \text{at-most-1-elem-term } \langle \langle_{nd} \otimes [E] \rangle As \rangle \rangle$   
*\langle proof \rangle*

**lemma** *same-length-indep-transmission-to-iterated-combine<sub>d</sub>-Sync:*  
 $\langle \text{length } \sigma s = \text{length } As \implies$   
 $(\bigwedge i j. i \leq \text{length } As \implies j \leq \text{length } As \implies i \neq j \implies$   
 $\text{indep-enabl } ((A_0 \# As) ! i) ((s_0 \# \sigma s) ! i) E ((A_0 \# As) ! j) ((s_0 \# \sigma s) ! j))$   
 $\implies$   
 $\text{indep-enabl } A_0 s_0 E \langle \langle_d \otimes [E] \rangle As \rangle \sigma s \rangle$   
*\langle proof \rangle*

**lemma**  *$\omega$ -iterated-combine<sub>d</sub>-Sync :*  
 $\langle \text{length } \sigma s = \text{length } As \implies$

$\omega \langle \langle_d \otimes [E] As \rangle \sigma s = (\text{case those } (\text{map2 } \omega As \sigma s) \text{ of } \diamond \Rightarrow \diamond$   
 $| [terms] \Rightarrow \text{if card (set terms)} = \text{Suc } 0 \text{ then } [THE r. \text{ set terms} = \{r\}] \text{ else}$   
 $\diamond) \rangle$   
 (proof)

**lemma**  $\omega$ -iterated-combine<sub>nd</sub>-Sync :

$\langle \text{length } \sigma s = \text{length } As \Rightarrow$   
 $\omega \langle \langle_{nd} \otimes [E] As \rangle \sigma s = (\text{if } As = [] \text{ then } \{\} \text{ else } \{r. \forall i < \text{length } As. r \in \omega (As !$   
 $i) (\sigma s ! i)\}) \rangle$   
 (proof)

### 6.1.5 Normalization

**lemma**  $\omega$ -iterated-combine<sub>nd</sub>-Sync-det-ndet-conv:

$\langle \text{length } \sigma s = \text{length } As \Rightarrow$   
 $\omega \langle \langle_{nd} \otimes [E] \text{map } (\lambda A. \langle A \rangle_{d \leftrightarrow nd}) As \rangle \sigma s = \omega \langle \langle_d \otimes [E] As \rangle \rangle_{d \leftrightarrow nd} \sigma s \rangle$   
 (proof)

**lemma**  $\tau$ -iterated-combine<sub>nd</sub>-Sync-behaviour-when-indep:

$\langle \text{length } \sigma s = \text{length } As \Rightarrow$   
 $(\bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket$   
 $\Rightarrow \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j)) \Rightarrow$   
 $\tau \langle \langle_d \otimes [E] As \rangle \rangle_{d \leftrightarrow nd} \sigma s e = \tau \langle \langle_{nd} \otimes [E] \text{map } (\lambda A. \langle A \rangle_{d \leftrightarrow nd}) As \rangle \sigma s e \rangle$   
 (proof)

**lemma**  $P_{SKIPS}$ -iterated-combine<sub>nd</sub>-Sync-behaviour-when-indep:

**assumes** same-length:  $\langle \text{length } \sigma s = \text{length } As \rangle$   
**and** indep:  $\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \Rightarrow$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$   
**shows**  $\langle P_{SKIPS} \langle \langle_d \otimes [E] As \rangle \rangle_d \sigma s = P_{SKIPS} \langle \langle_{nd} \otimes [E] \text{map } (\lambda A. \langle A \rangle_{d \leftrightarrow nd})$   
 $As \rangle \rangle_{nd} \sigma s \rangle$   
 (proof)

**lemma**  $P$ -d-iterated-combine<sub>nd</sub>-Sync-behaviour-when-indep:

**assumes** same-length:  $\langle \text{length } \sigma s = \text{length } As \rangle$   
**and** indep:  $\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \Rightarrow$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$   
**shows**  $\langle P \langle \langle_d \otimes [E] As \rangle \rangle_d \sigma s = P \langle \langle_{nd} \otimes [E] \text{map } (\lambda A. \langle A \rangle_{d \leftrightarrow nd}) As \rangle \rangle_{nd} \sigma s \rangle$   
 (proof)

## 6.2 Compactification Theorems

### 6.2.1 Binary

**Pair**

**theorem**  $P_{SKIPS}$ -nd-combine<sub>Pair</sub>-Sync :

**fixes**  $E :: \langle 'a \text{ set} \rangle$  **and**  $A_0 :: \langle ('\sigma, 'a, 'r, 'a) A_{nd}\text{-scheme} \rangle$   
**assumes**  $\varrho\text{-disjoint-}\varepsilon : \langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$   
**and**  $at\text{-most-1-elem-term} : \langle at\text{-most-1-elem-term } A_0 \rangle \langle at\text{-most-1-elem-term } A_1 \rangle$   
**defines**  $A\text{-def}: \langle A \equiv \langle A_0 \text{ nd} \otimes [E]_{Pair} A_1 \rangle \rangle$   
**defines**  $P\text{-def}: \langle P \equiv P_{SKIPS} \langle A_0 \rangle_{nd} \rangle$  **and**  $Q\text{-def}: \langle Q \equiv P_{SKIPS} \langle A_1 \rangle_{nd} \rangle$  **and**  
 $S\text{-def}: \langle S \equiv P_{SKIPS} \langle A \rangle_{nd} \rangle$   
**shows**  $\langle P \sigma_0 [E] Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$   
 $\langle proof \rangle$

**corollary**  $P\text{-nd-combine}_{Pair}\text{-Sync} :$   
 $\langle P \langle A_0 \rangle_{nd} \sigma_0 [E] P \langle A_1 \rangle_{nd} \sigma_1 = P \langle \langle A_0 \text{ nd} \otimes [E]_{Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1) \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}\text{-d-combine}_{Pair}\text{-Sync}:$   
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E] P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{ d} \otimes [E]_{Pair} A_1 \rangle \rangle_d (\sigma_0, \sigma_1) \rangle$   
**if**  $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$  **and**  $\langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$  **and**  $\langle indep\text{-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle proof \rangle$

**corollary**  $P\text{-d-combine}_{Pair}\text{-Sync}:$   
 $\langle P \langle A_0 \rangle_d \sigma_0 [E] P \langle A_1 \rangle_d \sigma_1 = P \langle \langle A_0 \text{ d} \otimes [E]_{Pair} A_1 \rangle \rangle_d (\sigma_0, \sigma_1) \rangle$   
**if**  $\langle indep\text{-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle proof \rangle$

## Pairlist

**theorem**  $P_{SKIPS}\text{-nd-combine}_{Pairlist}\text{-Sync} :$   
 $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 [E] P_{SKIPS} \langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{ nd} \otimes [E]_{Pairlist} A_1 \rangle \rangle_{nd} [\sigma_0, \sigma_1] \rangle$   
**if**  $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle \langle at\text{-most-1-elem-term } A_0 \rangle \langle at\text{-most-1-elem-term } A_1 \rangle$   
 $\langle proof \rangle$

**corollary**  $P\text{-nd-combine}_{Pairlist}\text{-Sync} :$   
 $\langle P \langle A_0 \rangle_{nd} \sigma_0 [E] P \langle A_1 \rangle_{nd} \sigma_1 = P \langle \langle A_0 \text{ nd} \otimes [E]_{Pairlist} A_1 \rangle \rangle_{nd} [\sigma_0, \sigma_1] \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}\text{-d-combine}_{Pairlist}\text{-Sync} :$   
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E] P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle A_0 \text{ d} \otimes [E]_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1] \rangle$   
**if**  $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$  **and**  $\langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$  **and**  $\langle indep\text{-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle proof \rangle$

**corollary**  $P\text{-d-combine}_{Pairlist}\text{-Sync} :$   
 $\langle P \langle A_0 \rangle_d \sigma_0 [E] P \langle A_1 \rangle_d \sigma_1 = P \langle \langle A_0 \text{ d} \otimes [E]_{Pairlist} A_1 \rangle \rangle_d [\sigma_0, \sigma_1] \rangle$   
**if**  $\langle indep\text{-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle proof \rangle$

## 6.2.2 Rlist

**theorem**  $P_{SKIPS}$ - $nd$ - $combine_{Rlist}$ - $Sync$  :

$\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 [E] P_{SKIPS}\langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes [E]_{Rlist} A_1 \rangle\rangle_{nd}$   
 $(\sigma_0 \# \sigma_1) \rangle$

**if**  $\langle \varrho$ - $disjoint$ - $\varepsilon A_0 \rangle \langle \varrho$ - $disjoint$ - $\varepsilon A_1 \rangle \langle at$ - $most$ - $1$ - $elem$ - $term A_0 \rangle \langle at$ - $most$ - $1$ - $elem$ - $term$   
 $A_1 \rangle$

$\langle proof \rangle$

**corollary**  $P$ - $nd$ - $combine_{Rlist}$ - $Sync$  :

$\langle P\langle A_0 \rangle_{nd} \sigma_0 [E] P\langle A_1 \rangle_{nd} \sigma_1 = P\langle\langle A_0 \text{ nd} \otimes [E]_{Rlist} A_1 \rangle\rangle_{nd} (\sigma_0 \# \sigma_1) \rangle$

$\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ - $combine_{Rlist}$ - $Sync$  :

$\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 [E] P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes [E]_{Rlist} A_1 \rangle\rangle_d (\sigma_0$   
 $\# \sigma_1) \rangle$

**if**  $\langle \varrho$ - $disjoint$ - $\varepsilon A_0 \rangle$  **and**  $\langle \varrho$ - $disjoint$ - $\varepsilon A_1 \rangle$  **and**  $\langle indep$ - $enabl A_0 \sigma_0 E A_1 \sigma_1 \rangle$

$\langle proof \rangle$

**corollary**  $P$ - $d$ - $combine_{Rlist}$ - $Sync$  :

$\langle P\langle A_0 \rangle_d \sigma_0 [E] P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \text{ d} \otimes [E]_{Rlist} A_1 \rangle\rangle_d (\sigma_0 \# \sigma_1) \rangle$

**if**  $\langle indep$ - $enabl A_0 \sigma_0 E A_1 \sigma_1 \rangle$

$\langle proof \rangle$

## 6.2.3 ListslenL

**theorem**  $P_{SKIPS}$ - $nd$ - $combine_{ListslenL}$ - $Sync$  :

**assumes**  $same$ - $length$ - $reach0$  :  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies length \sigma_0' = len_0 \rangle$   
**and**  $\varrho$ - $disjoint$ - $\varepsilon$  :  $\langle \varrho$ - $disjoint$ - $\varepsilon A_0 \rangle \langle \varrho$ - $disjoint$ - $\varepsilon A_1 \rangle \langle at$ - $most$ - $1$ - $elem$ - $term A_0 \rangle$   
 $\langle at$ - $most$ - $1$ - $elem$ - $term A_1 \rangle$

**shows**  $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 [E] P_{SKIPS}\langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ nd} \otimes [len_0,$   
 $E]_{ListslenL} A_1 \rangle\rangle_{nd} (\sigma_0 @ \sigma_1) \rangle$

$\langle proof \rangle$

**corollary**  $P$ - $nd$ - $combine_{ListslenL}$ - $Sync$  :

$\langle P\langle A_0 \rangle_{nd} \sigma_0 [E] P\langle A_1 \rangle_{nd} \sigma_1 = P\langle\langle A_0 \text{ nd} \otimes [len_0, E]_{ListslenL} A_1 \rangle\rangle_{nd} (\sigma_0 @$   
 $\sigma_1) \rangle$

**if**  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies length \sigma_0' = len_0 \rangle$

$\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ - $combine_{ListslenL}$ - $Sync$  :

$\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 [E] P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes [len_0, E]_{ListslenL}$   
 $A_1 \rangle\rangle_d (\sigma_0 @ \sigma_1) \rangle$

**if**  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies length \sigma_0' = len_0 \rangle$

$\langle \varrho$ - $disjoint$ - $\varepsilon A_0 \rangle \langle \varrho$ - $disjoint$ - $\varepsilon A_1 \rangle \langle indep$ - $enabl A_0 \sigma_0 E A_1 \sigma_1 \rangle$

$\langle proof \rangle$

**corollary**  $P$ - $d$ - $combine_{ListslenL}$ - $Sync$  :

$\langle P\langle A_0 \rangle_d \sigma_0 [E] P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \text{ d} \otimes [len_0, E]_{ListslenL} A_1 \rangle\rangle_d (\sigma_0 @ \sigma_1) \rangle$

**if**  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies length \sigma_0' = len_0 \rangle \langle indep$ - $enabl A_0 \sigma_0 E A_1 \sigma_1 \rangle$

$\langle \text{proof} \rangle$

## 6.2.4 Multiple

**theorem** *P<sub>SKIPS</sub>-nd-compactification-Sync:*

$\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A;$   
 $\bigwedge A. A \in \text{set } As \implies \text{at-most-1-elem-term } A \rrbracket$   
 $\implies \llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P_{SKIPS} \langle A \rangle_{nd} \sigma = P_{SKIPS} \langle \langle_{nd} \otimes \llbracket E \rrbracket$   
 $As \rangle \rangle_{nd} \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-nd-compactification-Sync:*

$\langle \text{length } \sigma s = \text{length } As \implies$   
 $\llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P \langle A \rangle_{nd} \sigma = P \langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle \rangle_{nd} \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P<sub>SKIPS</sub>-d-compactification-Sync:*

$\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A;$   
 $\bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies$   
 $\llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P_{SKIPS} \langle A \rangle_d \sigma = P_{SKIPS} \langle \langle_d \otimes \llbracket E \rrbracket As \rangle \rangle_d$   
 $\sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-d-compactification-Sync:*

$\langle \llbracket \text{length } \sigma s = \text{length } As;$   
 $\bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies$   
 $\llbracket E \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P \langle A \rangle_d \sigma = P \langle \langle_d \otimes \llbracket E \rrbracket As \rangle \rangle_d \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *P<sub>SKIPS</sub>-nd-compactification-Sync-order-is-arbitrary:*

$\langle P_{SKIPS} \langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle \rangle_{nd} \sigma s = P_{SKIPS} \langle \langle_{nd} \otimes \llbracket E \rrbracket As' \rangle \rangle_{nd} \sigma s' \rangle$   
**if**  $\langle \text{length } \sigma s = \text{length } As \rangle \langle \text{length } \sigma s' = \text{length } As' \rangle$   
 $\langle \text{mset } (\text{zip } \sigma s As) = \text{mset } (\text{zip } \sigma s' As') \rangle$   
 $\langle \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A \rangle$   
 $\langle \bigwedge A. A \in \text{set } As \implies \text{at-most-1-elem-term } A \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *P-nd-compactification-Sync-order-is-arbitrary:*

$\langle P \langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle \rangle_{nd} \sigma s = P \langle \langle_{nd} \otimes \llbracket E \rrbracket As' \rangle \rangle_{nd} \sigma s' \rangle$   
**if**  $\langle \text{length } \sigma s = \text{length } As \rangle \langle \text{length } \sigma s' = \text{length } As' \rangle$   
 $\langle \text{mset } (\text{zip } \sigma s As) = \text{mset } (\text{zip } \sigma s' As') \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *P<sub>SKIPS</sub>-d-compactification-Sync-order-is-arbitrary:*

$\langle P_{SKIPS} \langle \langle \langle d \otimes [E] \text{ As} \rangle \rangle_d \sigma s = P_{SKIPS} \langle \langle \langle d \otimes [E] \text{ As}' \rangle \rangle_d \sigma s' \rangle$   
**if**  $\langle \text{length } \sigma s = \text{length } \text{As} \rangle \langle \text{length } \sigma s' = \text{length } \text{As}' \rangle$   
 $\langle \text{mset } (\text{zip } \sigma s \text{ As}) = \text{mset } (\text{zip } \sigma s' \text{ As}') \rangle$   
 $\langle \bigwedge A. A \in \text{set } \text{As} \implies \varrho\text{-disjoint-}\varepsilon \text{ A} \rangle$   
 $\langle \bigwedge i j. \llbracket i < \text{length } \text{As}; j < \text{length } \text{As}; i \neq j \rrbracket \implies$   
 $\quad \text{indep-enabl } (\text{As } ! i) (\sigma s ! i) E (\text{As } ! j) (\sigma s ! j) \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *P-d-compactification-Sync-order-is-arbitrary:*

$\langle P \langle \langle \langle d \otimes [E] \text{ As} \rangle \rangle_d \sigma s = P \langle \langle \langle d \otimes [E] \text{ As}' \rangle \rangle_d \sigma s' \rangle$   
**if**  $\langle \text{length } \sigma s = \text{length } \text{As} \rangle \langle \text{length } \sigma s' = \text{length } \text{As}' \rangle$   
 $\langle \text{mset } (\text{zip } \sigma s \text{ As}) = \text{mset } (\text{zip } \sigma s' \text{ As}') \rangle$   
 $\langle \bigwedge i j. \llbracket i < \text{length } \text{As}; j < \text{length } \text{As}; i \neq j \rrbracket \implies$   
 $\quad \text{indep-enabl } (\text{As } ! i) (\sigma s ! i) E (\text{As } ! j) (\sigma s ! j) \rangle$   
 $\langle \text{proof} \rangle$

### 6.3 Derived Versions

**lemma** *P<sub>SKIPS</sub>-nd-compactification-Sync-upt-version:*

$\langle \llbracket [E] P \in \# \text{mset } (\text{map } Q [0..<n]) \rrbracket. P = P_{SKIPS} \langle \langle \langle \langle nd \otimes [E] \text{ map } A [0..<n] \rangle \rangle \rangle_{nd}$   
 $(\text{replicate } n \ 0) \rangle$   
**if**  $\langle \bigwedge i. i < n \implies \varrho\text{-disjoint-}\varepsilon (A \ i) \rangle$   
 $\langle \bigwedge i. i < n \implies \text{at-most-1-elem-term } (A \ i) \rangle$   
 $\langle \bigwedge i. i < n \implies P_{SKIPS} \langle \langle A \ i \rangle \rangle_{nd} \ 0 = Q \ i \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-nd-compactification-Sync-upt-version:*

$\langle \llbracket [E] P \in \# \text{mset } (\text{map } Q [0..<n]) \rrbracket. P = P \langle \langle \langle \langle nd \otimes [E] \text{ map } A [0..<n] \rangle \rangle \rangle_{nd} (\text{replicate}$   
 $n \ 0) \rangle$   
**if**  $\langle \bigwedge i. i < n \implies P \langle \langle A \ i \rangle \rangle_{nd} \ 0 = Q \ i \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P<sub>SKIPS</sub>-d-compactification-Sync-upt-version:*

$\langle \llbracket [E] P \in \# \text{mset } (\text{map } Q [0..<n]) \rrbracket. P = P_{SKIPS} \langle \langle \langle \langle d \otimes [E] \text{ map } A [0..<n] \rangle \rangle \rangle_d$   
 $(\text{replicate } n \ 0) \rangle$   
**if**  $\langle \bigwedge i. i < n \implies \varrho\text{-disjoint-}\varepsilon (A \ i) \rangle$   
 $\langle \bigwedge i j. i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) \ 0 \ E \ (A \ j) \ 0 \rangle$   
 $\langle \bigwedge i. i < n \implies P_{SKIPS} \langle \langle A \ i \rangle \rangle_d \ 0 = Q \ i \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-d-compactification-Sync-upt-version:*

$\langle \llbracket [E] P \in \# \text{mset } (\text{map } Q [0..<n]) \rrbracket. P = P \langle \langle \langle \langle d \otimes [E] \text{ map } A [0..<n] \rangle \rangle \rangle_d (\text{replicate}$   
 $n \ 0) \rangle$   
**if**  $\langle \bigwedge i. i < n \implies P \langle \langle A \ i \rangle \rangle_d \ 0 = Q \ i \rangle$   
 $\langle \bigwedge i j. i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) \ 0 \ E \ (A \ j) \ 0 \rangle$   
 $\langle \text{proof} \rangle$

## 6.4 More on Iterated Combine

**lemma**  $\varepsilon$ -iterated-combine <sub>$n_d$</sub> -Sync-general-form:

$\langle \text{length } \sigma s = \text{length } As \implies \varepsilon \llbracket_{n_d} \otimes [E] \rrbracket As \rangle \sigma s =$   
 $\langle \text{if } As = [] \text{ then } \{\} \rangle$   
 $\langle \text{else } (\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \cup (\bigcap i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) \rangle$   
**for**  $As :: \langle ('\sigma, 'e, 'r) A_{n_d} \text{ list} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -iterated-combine <sub>$d$</sub> -Sync-general-form:

$\langle \text{length } \sigma s = \text{length } As \implies \varepsilon \llbracket_d \otimes [E] \rrbracket As \rangle \sigma s =$   
 $\langle \text{if } As = [] \text{ then } \{\} \rangle$   
 $\langle \text{else } (\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \cup (\bigcap i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) \rangle$   
**for**  $As :: \langle ('\sigma, 'e, 'r) A_d \text{ list} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\tau$ -iterated-combine <sub>$n_d$</sub> -Sync-general-form:

$\langle \llbracket \text{length } \sigma s = \text{length } As; \sigma s' \in \tau \llbracket_{n_d} \otimes [E] \rrbracket As \rangle \sigma s a; i < \text{length } As \implies$   
 $\langle \sigma s' ! i \in \text{insert } (\sigma s ! i) (\tau (As ! i) (\sigma s ! i) a) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\tau$ -iterated-combine <sub>$d$</sub> -Sync-general-form:

$\langle \text{length } \sigma s = \text{length } As \implies$   
 $\tau \llbracket_d \otimes [E] \rrbracket As \rangle \sigma s a =$   
 $\langle \text{if } As = [] \text{ then } \diamond \text{ else}$   
 $\langle \text{if } a \in (\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \cup (\bigcap i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i))$   
 $\langle \text{then } [\text{map2 } (\lambda \sigma A. \text{if } a \in \varepsilon A \sigma \text{ then } [\tau A \sigma a] \text{ else } \sigma) \sigma s As] \text{ else } \diamond \rangle$   
**for**  $As :: \langle ('\sigma, 'e, 'r) A_d \text{ list} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *indep-implies-only-one-enabled'*:

$\langle \exists ! i. i < \text{length } As \wedge a \in \varepsilon (As ! i) (\sigma s ! i) \rangle$   
**if**  $\langle \text{length } \sigma s = \text{length } As \rangle$   
**and**  $\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$   
 $\langle \varepsilon (As ! i) (\sigma s ! i) \cap \varepsilon (As ! j) (\sigma s ! j) \subseteq E \rangle$   
**and**  $\langle a \in (\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *indep-implies-only-one-enabled*:

$\langle \llbracket \text{length } \sigma s = \text{length } As;$   
 $\bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j);$   
 $a \in (\bigcup i < \text{length } As. \varepsilon (As ! i) (\sigma s ! i)) - E \rrbracket \implies$

$\exists! i. i < \text{length } As \wedge a \in \varepsilon (As ! i) (\sigma s ! i)$   
 ⟨proof⟩

**lemma**  $\tau$ -iterated-combine<sub>d</sub>-Sync-general-form-when-indep:

$\langle \tau \llbracket_d \otimes [E] \rrbracket As \rangle \sigma s a =$   
 (if  $As = []$  then  $\diamond$   
 else if  $a \in (\bigcap_{i < \text{length } As} \varepsilon (As ! i) (\sigma s ! i))$   
 then  $\llbracket \text{map2 } (\lambda \sigma A. \lceil \tau A \sigma a \rceil) \sigma s As \rrbracket$   
 else if  $a \in (\bigcup_{i < \text{length } As} \varepsilon (As ! i) (\sigma s ! i)) - E$   
 then let  $i = \text{THE } i. i < \text{length } As \wedge a \in \varepsilon (As ! i) (\sigma s ! i)$   
 in  $\llbracket \sigma s [i := \lceil \tau (As ! i) (\sigma s ! i) a \rceil] \rrbracket$   
 else  $\diamond$ )  
 (is  $\langle - = \text{if } As = [] \text{ then } \diamond \text{ else}$   
 $\text{if } a \in ?I \text{ } As \text{ } \sigma s \text{ then } \llbracket \text{map2 } (\lambda \sigma A. \lceil \tau A \sigma a \rceil) \sigma s As \rrbracket$  else  
 $\text{if } a \in ?U \text{ } As \text{ } \sigma s - E \text{ then } ?\text{upd } As \text{ } \sigma s \text{ else } \diamond$ )  
 if  $\langle \text{length } \sigma s = \text{length } As \rangle$   
 $\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$   
 ⟨proof⟩

## 6.5 More on Events

**lemma** *events-of-MultiSync-P<sub>SKIPS</sub>-nd* :

$\langle \alpha(\llbracket [E] \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P_{SKIPS} \langle A \rangle_{nd} \sigma) =$   
 (if  $As = []$  then  $\{\}$  else  
 $\bigcup \sigma s' \in \mathcal{R}_{nd} \llbracket_{nd} \otimes [E] \rrbracket As \rangle \sigma s. (\bigcup_{i < \text{length } As} \varepsilon (As ! i) (\sigma s' ! i)) - E \cup$   
 $(\bigcap_{i < \text{length } As} \varepsilon (As ! i) (\sigma s' ! i)) \rangle$   
 (is  $\langle - = ?rhs \rangle$ ) if  $\langle \text{length } \sigma s = \text{length } As \rangle$   
 $\langle \bigwedge A. A \in \text{set } As \implies \rho\text{-disjoint-}\varepsilon A \rangle \langle \bigwedge A. A \in \text{set } As \implies \text{at-most-1-elem-term}$   
 $A \rangle$   
 ⟨proof⟩

**lemma** *events-of-MultiSync-P-nd* :

$\langle \alpha(\llbracket [E] \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P \langle A \rangle_{nd} \sigma) =$   
 (if  $As = []$  then  $\{\}$  else  
 $\bigcup \sigma s' \in \mathcal{R}_{nd} \llbracket_{nd} \otimes [E] \rrbracket As \rangle \sigma s. (\bigcup_{i < \text{length } As} \varepsilon (As ! i) (\sigma s' ! i)) - E \cup$   
 $(\bigcap_{i < \text{length } As} \varepsilon (As ! i) (\sigma s' ! i)) \rangle$   
 (is  $\langle - = ?rhs \rangle$ ) if  $\langle \text{length } \sigma s = \text{length } As \rangle$   
 ⟨proof⟩

**lemma** *events-of-MultiSync-P<sub>SKIPS</sub>-d* :

$\langle \alpha(\llbracket [E] \rrbracket (\sigma, A) \in \# \text{mset } (\text{zip } \sigma s As). P_{SKIPS} \langle A \rangle_d \sigma) =$   
 (if  $As = []$  then  $\{\}$  else  
 $\bigcup \sigma s' \in \mathcal{R}_d \llbracket_d \otimes [E] \rrbracket As \rangle \sigma s. (\bigcup_{i < \text{length } As} \varepsilon (As ! i) (\sigma s' ! i)) - E \cup$   
 $(\bigcap_{i < \text{length } As} \varepsilon (As ! i) (\sigma s' ! i)) \rangle$

**(is**  $\langle - = ?rhs \rangle$  **if**  $\langle length \ \sigma s = length \ As \rangle$   $\langle \bigwedge A. A \in set \ As \implies \rho\text{-disjoint-}\varepsilon \ A \rangle$   
 $\langle \bigwedge i \ j. \llbracket i < length \ As; j < length \ As; i \neq j \rrbracket \implies$   
 $\quad indep\text{-enabl} \ (As \ ! \ i) \ (\sigma s \ ! \ i) \ E \ (As \ ! \ j) \ (\sigma s \ ! \ j) \rangle$   
**\langle proof \rangle**

**lemma** *events-of-MultiSync-P-d* :

$\langle \alpha(\llbracket E \rrbracket \ (\sigma, A) \in \# \ mset \ (zip \ \sigma s \ As). \ P \langle A \rangle_d \ \sigma) =$   
 $\quad (if \ As = [] \ then \ \{\} \ else$   
 $\quad \bigcup \ \sigma s' \in \mathcal{R}_d \ \langle \langle A \otimes \llbracket E \rrbracket \ As \rangle \ \sigma s. \ (\bigcup_{i < length \ As. \ \varepsilon \ (As \ ! \ i) \ (\sigma s' \ ! \ i)} - E \cup$   
 $\quad \quad (\bigcap_{i < length \ As. \ \varepsilon \ (As \ ! \ i) \ (\sigma s' \ ! \ i)}) \rangle \rangle$   
**(is**  $\langle - = ?rhs \rangle$  **if**  $\langle length \ \sigma s = length \ As \rangle$   
**and**  $\langle \bigwedge i \ j. \llbracket i < length \ As; j < length \ As; i \neq j \rrbracket \implies$   
 $\quad indep\text{-enabl} \ (As \ ! \ i) \ (\sigma s \ ! \ i) \ E \ (As \ ! \ j) \ (\sigma s \ ! \ j) \rangle$   
**\langle proof \rangle**



## Chapter 7

# Combining Automata for Generalized Synchronization Product

### 7.1 Definitions

#### 7.1.1 Specializations

**definition**  $combine_{dPairlist-Sync_{ptick}} ::$   
 $\langle [(\sigma, 'e, 'r, 'α) A_d\text{-scheme}, 'e \text{ set}, ('σ, 'e, 'r, 'β) A_d\text{-scheme}] \Rightarrow ('σ \text{ list}, 'e, 'r \text{ list}) A_d \rangle$

**where**  $\langle combine_{dPairlist-Sync_{ptick}} A_0 E A_1 \equiv$   
 $combine_{d-Sync} A_0 E A_1 hd (\lambda s s. hd (tl \sigma s)) (\lambda s t. [s, t]) (\lambda s t. [[s, t]]) \rangle$

**definition**  $combine_{ndPairlist-Sync_{ptick}} ::$   
 $\langle [(\sigma, 'e, 'r, 'α) A_{nd}\text{-scheme}, 'e \text{ set}, ('σ, 'e, 'r, 'β) A_{nd}\text{-scheme}] \Rightarrow ('σ \text{ list}, 'e, 'r \text{ list}) A_{nd} \rangle$

**where**  $\langle combine_{ndPairlist-Sync_{ptick}} A_0 E A_1 \equiv combine_{nd-Sync} A_0 E A_1 hd$   
 $(\lambda s s. hd (tl \sigma s)) (\lambda s t. [s, t]) (\lambda s t. [[s, t]]) \rangle$

**definition**  $combine_{dPair-Sync_{ptick}} ::$   
 $\langle [(\sigma_0, 'e, 'r_0, 'α) A_d\text{-scheme}, 'e \text{ set}, ('σ_1, 'e, 'r_1, 'β) A_d\text{-scheme}] \Rightarrow ('σ_0 \times 'σ_1,$   
 $'e, 'r_0 \times 'r_1) A_d \rangle$

**where**  $\langle combine_{dPair-Sync_{ptick}} A_0 E A_1 \equiv combine_{d-Sync} A_0 E A_1 fst snd Pair$   
 $(\lambda s r. [(s, r)]) \rangle$

**definition**  $combine_{ndPair-Sync_{ptick}} ::$   
 $\langle [(\sigma_0, 'e, 'r_0, 'α) A_{nd}\text{-scheme}, 'e \text{ set}, ('σ_1, 'e, 'r_1, 'β) A_{nd}\text{-scheme}] \Rightarrow ('σ_0 \times$   
 $'σ_1, 'e, 'r_0 \times 'r_1) A_{nd} \rangle$

**where**  $\langle combine_{ndPair-Sync_{ptick}} A_0 E A_1 \equiv combine_{nd-Sync} A_0 E A_1 fst snd$   
 $Pair (\lambda s r. [(s, r)]) \rangle$

**definition**  $combine_{dListslenL-Sync_{ptick}} ::$   
 $\langle [(\sigma \text{ list}, 'e, 'r \text{ list}, 'α) A_d\text{-scheme}, nat, 'e \text{ set}, ('σ \text{ list}, 'e, 'r \text{ list}, 'β) A_d\text{-scheme}]$   
 $\Rightarrow ('σ \text{ list}, 'e, 'r \text{ list}) A_d \rangle$

**where**  $\langle \text{combine}_{d\text{ListslenL-Sync}_{\text{ptick}}} A_0 \text{ len}_0 E A_1 \equiv \text{combine}_d\text{-Sync} A_0 E A_1$   
 $(\text{take len}_0) (\text{drop len}_0) (\text{@}) (\lambda s r. [s \text{@} r]) \rangle$

**definition**  $\text{combine}_{nd\text{ListslenL-Sync}_{\text{ptick}}} ::$

$\langle [(\sigma \text{ list}, 'e, 'r \text{ list}, ' \alpha) A_{nd}\text{-scheme}, \text{ nat}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, ' \beta) A_{nd}\text{-scheme}]$   
 $\Rightarrow (\sigma \text{ list}, 'e, 'r \text{ list}) A_{nd} \rangle$

**where**  $\langle \text{combine}_{nd\text{ListslenL-Sync}_{\text{ptick}}} A_0 \text{ len}_0 E A_1 \equiv \text{combine}_{nd}\text{-Sync} A_0 E A_1$   
 $(\text{take len}_0) (\text{drop len}_0) (\text{@}) (\lambda s r. [s \text{@} r]) \rangle$

**definition**  $\text{combine}_{d\text{Rlist-Sync}_{\text{ptick}}} ::$

$\langle [(\sigma, 'e, 'r, ' \alpha) A_d\text{-scheme}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, ' \beta) A_d\text{-scheme}] \Rightarrow (\sigma \text{ list},$   
 $'e, 'r \text{ list}) A_d \rangle$

**where**  $\langle \text{combine}_{d\text{Rlist-Sync}_{\text{ptick}}} A_0 E A_1 \equiv \text{combine}_d\text{-Sync} A_0 E A_1 \text{ hd tl } (\#)$   
 $(\lambda s r. [s \# r]) \rangle$

**definition**  $\text{combine}_{nd\text{Rlist-Sync}_{\text{ptick}}} ::$

$\langle [(\sigma, 'e, 'r, ' \alpha) A_{nd}\text{-scheme}, 'e \text{ set}, (\sigma \text{ list}, 'e, 'r \text{ list}, ' \beta) A_{nd}\text{-scheme}] \Rightarrow (\sigma$   
 $\text{ list}, 'e, 'r \text{ list}) A_{nd} \rangle$

**where**  $\langle \text{combine}_{nd\text{Rlist-Sync}_{\text{ptick}}} A_0 E A_1 \equiv \text{combine}_{nd}\text{-Sync} A_0 E A_1 \text{ hd tl } (\#)$   
 $(\lambda s r. [s \# r]) \rangle$

**lemmas**  $\text{combine}_{\text{Pairlist-Sync}_{\text{ptick}}}\text{-defs} = \text{combine}_{d\text{Pairlist-Sync}_{\text{ptick}}}\text{-def } \text{combine}_{nd\text{Pairlist-Sync}_{\text{ptick}}}\text{-def}$

**and**  $\text{combine}_{\text{Pair-Sync}_{\text{ptick}}}\text{-defs} = \text{combine}_{d\text{Pair-Sync}_{\text{ptick}}}\text{-def } \text{combine}_{nd\text{Pair-Sync}_{\text{ptick}}}\text{-def}$

**and**  $\text{combine}_{\text{ListslenL-Sync}_{\text{ptick}}}\text{-defs} = \text{combine}_{d\text{ListslenL-Sync}_{\text{ptick}}}\text{-def } \text{combine}_{nd\text{ListslenL-Sync}_{\text{ptick}}}\text{-def}$

**and**  $\text{combine}_{\text{Rlist-Sync}_{\text{ptick}}}\text{-defs} = \text{combine}_{d\text{Rlist-Sync}_{\text{ptick}}}\text{-def } \text{combine}_{nd\text{Rlist-Sync}_{\text{ptick}}}\text{-def}$

**lemmas**  $\text{combine-Sync}_{\text{ptick}}\text{-defs} =$

$\text{combine}_{\text{Pairlist-Sync}_{\text{ptick}}}\text{-defs } \text{combine}_{\text{Pair-Sync}_{\text{ptick}}}\text{-defs } \text{combine}_{\text{ListslenL-Sync}_{\text{ptick}}}\text{-defs}$   
 $\text{combine}_{\text{Rlist-Sync}_{\text{ptick}}}\text{-defs}$

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

**notation**  $\text{combine}_{d\text{Pairlist-Sync}_{\text{ptick}}} (\langle \langle - \text{d} \otimes [-] \checkmark_{\text{Pairlist}} - \rangle \rangle [0, 0, 0])$

**notation**  $\text{combine}_{nd\text{Pairlist-Sync}_{\text{ptick}}} (\langle \langle - \text{nd} \otimes [-] \checkmark_{\text{Pairlist}} - \rangle \rangle [0, 0, 0])$

**notation**  $\text{combine}_{d\text{Pair-Sync}_{\text{ptick}}} (\langle \langle - \text{d} \otimes [-] \checkmark_{\text{Pair}} - \rangle \rangle [0, 0, 0])$

**notation**  $\text{combine}_{nd\text{Pair-Sync}_{\text{ptick}}} (\langle \langle - \text{nd} \otimes [-] \checkmark_{\text{Pair}} - \rangle \rangle [0, 0, 0])$

**notation**  $\text{combine}_{d\text{ListslenL-Sync}_{\text{ptick}}} (\langle \langle - \text{d} \otimes [-, -] \checkmark_{\text{ListslenL}} - \rangle \rangle [0, 0, 0, 0])$

**notation**  $\text{combine}_{nd\text{ListslenL-Sync}_{\text{ptick}}} (\langle \langle - \text{nd} \otimes [-, -] \checkmark_{\text{ListslenL}} - \rangle \rangle [0, 0, 0, 0])$

**notation**  $\text{combine}_{d\text{Rlist-Sync}_{\text{ptick}}} (\langle \langle - \text{d} \otimes [-] \checkmark_{\text{Rlist}} - \rangle \rangle [0, 0, 0])$

**notation**  $\text{combine}_{nd\text{Rlist-Sync}_{\text{ptick}}} (\langle \langle - \text{nd} \otimes [-] \checkmark_{\text{Rlist}} - \rangle \rangle [0, 0, 0])$

**end**

**unbundle**  $\text{combine}_{nd}\text{-Sync}_{\text{ptick}}\text{-syntax}$

## 7.2 First Properties

**lemma**  $\text{finite-trans-combine}_{nd}\text{-Sync}_{\text{ptick}}\text{-simps } [\text{simp}] :$

$\langle \text{finite-trans } A_0 \Rightarrow \text{finite-trans } A_1 \Rightarrow \text{finite-trans } \langle \langle A_0 \text{ nd} \otimes [E] \checkmark_{\text{Pairlist}} A_1 \rangle \rangle \rangle$

$\langle \text{finite-trans } B_0 \implies \text{finite-trans } B_1 \implies \text{finite-trans } \ll B_0 \text{ nd} \otimes [E] \checkmark_{\text{Pair}} B_1 \gg \rangle$   
 $\langle \text{finite-trans } C_0 \implies \text{finite-trans } C_1 \implies \text{finite-trans } \ll C_0 \text{ nd} \otimes [len_0, E] \checkmark_{\text{ListslenL}} C_1 \gg \rangle$   
 $\langle \text{finite-trans } D_0 \implies \text{finite-trans } D_1 \implies \text{finite-trans } \ll D_0 \text{ nd} \otimes [E] \checkmark_{\text{Rlist}} D_1 \gg \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -combine<sub>Pairlist-Syncptick</sub>:

$\langle \varepsilon \ll A_0 \text{ d} \otimes [E] \checkmark_{\text{Pairlist}} A_1 \gg \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 \text{ hd } (hd \circ tl) \sigma s \rangle$   
 $\langle \varepsilon \ll B_0 \text{ nd} \otimes [E] \checkmark_{\text{Pairlist}} B_1 \gg \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 \text{ hd } (hd \circ tl) \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -combine<sub>Pair-Syncptick</sub>:

$\langle \varepsilon \ll A_0 \text{ d} \otimes [E] \checkmark_{\text{Pair}} A_1 \gg \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 \text{ fst snd } \sigma s \rangle$   
 $\langle \varepsilon \ll B_0 \text{ nd} \otimes [E] \checkmark_{\text{Pair}} B_1 \gg \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 \text{ fst snd } \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -combine<sub>ListslenL-Syncptick</sub>:

$\langle \varepsilon \ll A_0 \text{ d} \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \gg \sigma s = \text{combine-Sync-}\varepsilon A_0 E A_1 (\text{take } len_0) (\text{drop } len_0) \sigma s \rangle$   
 $\langle \varepsilon \ll B_0 \text{ nd} \otimes [len_0, E] \checkmark_{\text{ListslenL}} B_1 \gg \sigma s = \text{combine-Sync-}\varepsilon B_0 E B_1 (\text{take } len_0) (\text{drop } len_0) \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon$ -combine<sub>Rlist-Syncptick</sub>:

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

**lemma**  $\varrho$ -combine<sub>Pairlist-Syncptick</sub>:

$\langle \varrho \ll A_0 \text{ d} \otimes [E] \checkmark_{\text{Pairlist}} A_1 \gg = \{ \sigma s. \text{hd } \sigma s \in \varrho A_0 \wedge \text{hd } (tl \sigma s) \in \varrho A_1 \} \rangle$   
 $\langle \varrho \ll B_0 \text{ nd} \otimes [E] \checkmark_{\text{Pairlist}} B_1 \gg = \{ \sigma s. \text{hd } \sigma s \in \varrho B_0 \wedge \text{hd } (tl \sigma s) \in \varrho B_1 \} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varrho$ -combine<sub>Pair-Syncptick</sub>:

$\langle \varrho \ll A_0 \text{ d} \otimes [E] \checkmark_{\text{Pair}} A_1 \gg = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho A_0 \wedge \sigma_1 \in \varrho A_1 \} \rangle$   
 $\langle \varrho \ll B_0 \text{ nd} \otimes [E] \checkmark_{\text{Pair}} B_1 \gg = \{ (\sigma_0, \sigma_1). \sigma_0 \in \varrho B_0 \wedge \sigma_1 \in \varrho B_1 \} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varrho$ -combine<sub>ListslenL-Syncptick</sub>:

$\langle \varrho \ll A_0 \text{ d} \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \gg = \{ \sigma s. \text{take } len_0 \sigma s \in \varrho A_0 \wedge \text{drop } len_0 \sigma s \in \varrho A_1 \} \rangle$   
 $\langle \varrho \ll B_0 \text{ nd} \otimes [len_0, E] \checkmark_{\text{ListslenL}} B_1 \gg = \{ \sigma s. \text{take } len_0 \sigma s \in \varrho B_0 \wedge \text{drop } len_0 \sigma s \in \varrho B_1 \} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varrho$ -combine<sub>Rlist-Syncptick</sub>:

$\langle \varrho \ll A_0 \text{ d} \otimes [E] \checkmark_{\text{Rlist}} A_1 \gg = \{ \sigma s. \text{hd } \sigma s \in \varrho A_0 \wedge \text{tl } \sigma s \in \varrho A_1 \} \rangle$   
 $\langle \varrho \ll B_0 \text{ nd} \otimes [E] \checkmark_{\text{Rlist}} B_1 \gg = \{ \sigma s. \text{hd } \sigma s \in \varrho B_0 \wedge \text{tl } \sigma s \in \varrho B_1 \} \rangle$

*<proof>*

### 7.3 Transitions are unchanged in the Generalization

In the generalization, only the  $\omega$  function is modified.

**lemma**  $\tau$ -combine<sub>Pairlist-Syncptick</sub> :

$\langle \tau \langle A_0 \text{ }_d \otimes [E] \checkmark_{\text{Pairlist}} A_1 \rangle = \tau \langle A_0 \text{ }_d \otimes [E]_{\text{Pairlist}} A_1 \rangle \rangle$   
 $\langle \tau \langle B_0 \text{ }_{nd} \otimes [E] \checkmark_{\text{Pairlist}} B_1 \rangle = \tau \langle B_0 \text{ }_{nd} \otimes [E]_{\text{Pairlist}} B_1 \rangle \rangle$   
*<proof>*

**lemma**  $\tau$ -combine<sub>Pair-Syncptick</sub> :

$\langle \tau \langle A_0 \text{ }_d \otimes [E] \checkmark_{\text{Pair}} A_1 \rangle = \tau \langle A_0 \text{ }_d \otimes [E]_{\text{Pair}} A_1 \rangle \rangle$   
 $\langle \tau \langle B_0 \text{ }_{nd} \otimes [E] \checkmark_{\text{Pair}} B_1 \rangle = \tau \langle B_0 \text{ }_{nd} \otimes [E]_{\text{Pair}} B_1 \rangle \rangle$   
*<proof>*

**lemma**  $\tau$ -combine<sub>ListslenL-Syncptick</sub> :

$\langle \tau \langle A_0 \text{ }_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \rangle = \tau \langle A_0 \text{ }_d \otimes [len_0, E]_{\text{ListslenL}} A_1 \rangle \rangle$   
 $\langle \tau \langle B_0 \text{ }_{nd} \otimes [len_0, E] \checkmark_{\text{ListslenL}} B_1 \rangle = \tau \langle B_0 \text{ }_{nd} \otimes [len_0, E]_{\text{ListslenL}} B_1 \rangle \rangle$   
*<proof>*

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

### 7.4 Reachability

**lemma**  $\mathcal{R}_d$ -combine<sub>dListslenL-Syncptick-subset</sub>:

$\langle \mathcal{R}_d \langle A_0 \text{ }_d \otimes [len_0, E] \checkmark_{\text{ListslenL}} A_1 \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 \text{ }_t_1. t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )  
**if**  $\langle \bigwedge t_0. t_0 \in \mathcal{R}_d A_0 s_0 \implies \text{length } t_0 = len_0 \rangle$   
*<proof>*

**lemma**  $\mathcal{R}_{nd}$ -combine<sub>ndListslenL-Syncptick-subset</sub>:

$\langle \mathcal{R}_{nd} \langle B_0 \text{ }_{nd} \otimes [len_0, E] \checkmark_{\text{ListslenL}} B_1 \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 \text{ }_t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$  (is  $\langle ?S_B \subseteq \rightarrow \rangle$ )  
**if**  $\langle \bigwedge t_0. t_0 \in \mathcal{R}_{nd} B_0 s_0 \implies \text{length } t_0 = len_0 \rangle$   
*<proof>*

**lemma**  $\mathcal{R}_d$ -combine<sub>dPairlist-Syncptick-subset</sub>:

$\langle \mathcal{R}_d \langle A_0 \text{ }_d \otimes [E] \checkmark_{\text{Pairlist}} A_1 \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 \text{ }_t_1. t_0 \in \mathcal{R}_d A_0 s_0 \wedge t_1 \in \mathcal{R}_d A_1 s_1\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )  
**and**  $\mathcal{R}_{nd}$ -combine<sub>ndPairlist-Syncptick-subset</sub>:  
 $\langle \mathcal{R}_{nd} \langle B_0 \text{ }_{nd} \otimes [E] \checkmark_{\text{Pairlist}} B_1 \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 \text{ }_t_1. t_0 \in \mathcal{R}_{nd} B_0 s_0 \wedge t_1 \in \mathcal{R}_{nd} B_1 s_1\} \rangle$  (is  $\langle ?S_B \subseteq \rightarrow \rangle$ )  
*<proof>*

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

$$\langle \mathcal{R}_d \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_d A_0 \ s_0 \times \mathcal{R}_d A_1 \ s_1 \rangle \text{ (is } \langle ?S_A \subseteq - \rangle)$$

and  $\mathcal{R}_{nd}$ -combine $_{nd}$ Pair-Sync $_{ptick}$ -subset:

$$\langle \mathcal{R}_{nd} \langle \langle B_0 \ nd \otimes [E] \checkmark_{Pair} B_1 \rangle \rangle (s_0, s_1) \subseteq \mathcal{R}_{nd} B_0 \ s_0 \times \mathcal{R}_{nd} B_1 \ s_1 \rangle \text{ (is } \langle ?S_B \subseteq - \rangle)$$

$\langle proof \rangle$

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

$$\langle \mathcal{R}_d \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \ \sigma t. t_0 \in \mathcal{R}_d A_0 \ s_0 \wedge \sigma t \in \mathcal{R}_d A_1 \ \sigma s\} \rangle \text{ (is } \langle ?S_A \subseteq - \rangle)$$

and  $\mathcal{R}_{nd}$ -combine $_{nd}$ Rlist-Sync $_{ptick}$ -subset:

$$\langle \mathcal{R}_{nd} \langle \langle B_0 \ nd \otimes [E] \checkmark_{Rlist} B_1 \rangle \rangle (s_0 \# \sigma s) \subseteq \{t_0 \# \sigma t \mid t_0 \ \sigma t. t_0 \in \mathcal{R}_{nd} B_0 \ s_0 \wedge \sigma t \in \mathcal{R}_{nd} B_1 \ \sigma s\} \rangle \text{ (is } \langle ?S_B \subseteq - \rangle)$$

$\langle proof \rangle$

## 7.5 Normalization

**lemma**  $\omega$ -combine $_{Pairlist}$ -Sync $_{ptick}$ -behaviour:

$$\langle \omega \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle \rangle_{d \mapsto nd} [s_0, s_1] = \omega \langle \langle \langle A_0 \rangle \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pairlist} \langle \langle A_1 \rangle \rangle_{d \mapsto nd} [s_0, s_1] \rangle$$

$\langle proof \rangle$

**lemma**  $\omega$ -combine $_{Pair}$ -Sync $_{ptick}$ -behaviour:

$$\langle \omega \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle \rangle_{d \mapsto nd} (s_0, s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pair} \langle \langle A_1 \rangle \rangle_{d \mapsto nd} (s_0, s_1) \rangle$$

$\langle proof \rangle$

**lemma**  $\omega$ -combine $_{ListslenL}$ -Sync $_{ptick}$ -behaviour:

$$\langle \omega \langle \langle \langle A_0 \ d \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle \rangle_{d \mapsto nd} (\sigma s_0 @ \sigma s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \mapsto nd} \ nd \otimes [len_0, E] \checkmark_{ListslenL} \langle \langle A_1 \rangle \rangle_{d \mapsto nd} (\sigma s_0 @ \sigma s_1) \rangle$$

$\langle proof \rangle$

**lemma**  $\omega$ -combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour:

$$\langle \omega \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle \rangle_{d \mapsto nd} (s_0 \# \sigma s_1) = \omega \langle \langle \langle A_0 \rangle \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Rlist} \langle \langle A_1 \rangle \rangle_{d \mapsto nd} (s_0 \# \sigma s_1) \rangle$$

$\langle proof \rangle$

**lemma**  $\tau$ -combine $_{Pairlist}$ -Sync $_{ptick}$ -behaviour-when-indep:

$$\langle \varepsilon A_0 \ s_0 \cap \varepsilon A_1 \ s_1 \subseteq E \implies$$

$$\tau \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle \rangle_{d \mapsto nd} [s_0, s_1] \ e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pairlist} \langle \langle A_1 \rangle \rangle_{d \mapsto nd} [s_0, s_1] \ e \rangle$$

$\langle proof \rangle$

**lemma**  $\tau$ -combine $_{Pair}$ -Sync $_{ptick}$ -behaviour-when-indep:

$$\langle \varepsilon A_0 \ s_0 \cap \varepsilon A_1 \ s_1 \subseteq E \implies$$

$$\tau \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle \rangle_{d \mapsto nd} (s_0, s_1) \ e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \mapsto nd} \ nd \otimes [E] \checkmark_{Pair} \langle \langle A_1 \rangle \rangle_{d \mapsto nd} (s_0, s_1) \ e \rangle$$

$\langle proof \rangle$

**lemma**  $\tau$ -combine $_{ListslenL}$ -Sync $_{ptick}$ -behaviour-when-indep:

$\langle \varepsilon A_0 \sigma s_0 \cap \varepsilon A_1 \sigma s_1 \subseteq E \implies length \sigma s_0 = len_0 \implies$   
 $\tau \langle \langle \langle A_0 \ d \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 \ @ \ \sigma s_1) \ e = \tau \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd}$   
 $nd \otimes [len_0, E] \checkmark_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle (\sigma s_0 \ @ \ \sigma s_1) \ e \rangle$   
 $\langle proof \rangle$

**lemma**  $\tau$ -combine $_{Rlist}$ -Sync $_{ptick}$ -behaviour-when-indep:

$\langle \varepsilon A_0 \ s_0 \cap \varepsilon A_1 \ \sigma s_1 \subseteq E \implies$   
 $\tau \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_{d \hookrightarrow nd} (s_0 \ \# \ \sigma s_1) \ e = \tau \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Rlist}$   
 $\langle A_1 \rangle_{d \hookrightarrow nd} \rangle (s_0 \ \# \ \sigma s_1) \ e \rangle$   
 $\langle proof \rangle$

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

$\langle P_{SKIPS} \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [s_0, s_1] = P_{SKIPS} \langle \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Pairlist}$   
 $\langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} [s_0, s_1] \rangle$   
**if**  $\langle indep-enabl \ A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
 $\langle proof \rangle$

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

$\langle P \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pairlist} A_1 \rangle \rangle_d [s_0, s_1] = P \langle \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Pairlist} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd}$   
 $[s_0, s_1] \rangle$   
**if**  $\langle indep-enabl \ A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
 $\langle proof \rangle$

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

$\langle P_{SKIPS} \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_d (s_0, s_1) = P_{SKIPS} \langle \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Pair}$   
 $\langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0, s_1) \rangle$   
**if**  $\langle indep-enabl \ A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
 $\langle proof \rangle$

**lemma**  $P$ -combine $_{Pair}$ -Sync $_{ptick}$ -behaviour-when-indep:

$\langle P \langle \langle \langle A_0 \ d \otimes [E] \checkmark_{Pair} A_1 \rangle \rangle_d (s_0, s_1) = P \langle \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Pair} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd}$   
 $(s_0, s_1) \rangle$   
**if**  $\langle indep-enabl \ A_0 \ s_0 \ E \ A_1 \ s_1 \rangle$   
 $\langle proof \rangle$

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

$\langle P_{SKIPS} \langle \langle \langle A_0 \ d \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_d (\sigma s_0 \ @ \ \sigma s_1) = P_{SKIPS} \langle \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd}$   
 $nd \otimes [len_0, E] \checkmark_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 \ @ \ \sigma s_1) \rangle$   
**if**  $\langle indep-enabl \ A_0 \ \sigma s_0 \ E \ A_1 \ \sigma s_1 \rangle$  **and**  $\langle \wedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d \ A_0 \ \sigma s_0 \implies length \ \sigma t_0$   
 $= len_0 \rangle$   
 $\langle proof \rangle$

**lemma**  $P$ -combine $_{ListslenL}$ -Sync $_{ptick}$ -behaviour-when-indep:

$\langle P \langle \langle A_0 \ d \otimes [len_0, E] \checkmark_{ListslenL} A_1 \rangle \rangle_d (\sigma s_0 \ @ \ \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [len_0, E] \checkmark_{ListslenL} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (\sigma s_0 \ @ \ \sigma s_1) \rangle$   
**if**  $\langle indep-enabl \ A_0 \ \sigma s_0 \ E \ A_1 \ \sigma s_1 \rangle$  **and**  $\langle \wedge \sigma t_0. \ \sigma t_0 \in \mathcal{R}_d \ A_0 \ \sigma s_0 \implies length \ \sigma t_0 = len_0 \rangle$   
 $\langle proof \rangle$

**lemma** *P<sub>SKIPS</sub>-combine<sub>Rlist-Sync<sub>ptick</sub></sub>*-behaviour-when-indep:

$\langle P_{SKIPS} \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_d (s_0 \ # \ \sigma s_1) = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Rlist} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0 \ # \ \sigma s_1) \rangle$   
**if**  $\langle indep-enabl \ A_0 \ s_0 \ E \ A_1 \ \sigma s_1 \rangle$   
 $\langle proof \rangle$

**lemma** *P-combine<sub>Rlist-Sync<sub>ptick</sub></sub>*-behaviour-when-indep:

$\langle P \langle \langle A_0 \ d \otimes [E] \checkmark_{Rlist} A_1 \rangle \rangle_d (s_0 \ # \ \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \ nd \otimes [E] \checkmark_{Rlist} \langle A_1 \rangle_{d \hookrightarrow nd} \rangle \rangle_{nd} (s_0 \ # \ \sigma s_1) \rangle$   
**if**  $\langle indep-enabl \ A_0 \ s_0 \ E \ A_1 \ \sigma s_1 \rangle$   
 $\langle proof \rangle$



## Chapter 8

# Compactification of Synchronization Product Generalized

### 8.1 Iterated Combine

#### 8.1.1 Definitions

**fun** *iterated-combine<sub>d</sub>-Sync<sub>ptick</sub>* ::  $\langle 'e \text{ set} \Rightarrow (' \sigma, 'e, 'r) A_d \text{ list} \Rightarrow (' \sigma \text{ list}, 'e, 'r \text{ list}) A_d \rangle$  ( $\langle \langle d \otimes [-] \checkmark - \rangle \rangle [0, 0]$ )  
**where**  $\langle \langle d \otimes [E] \checkmark [] \rangle \rangle = \langle \tau = \lambda \sigma s a. \diamond, \omega = \lambda \sigma s. \diamond \rangle$   
 $| \quad \langle \langle d \otimes [E] \checkmark [A_0] \rangle \rangle = d \langle \langle A_0 \rangle \rangle_{\text{singl} \hookrightarrow \text{list}}$   
 $| \quad \langle \langle d \otimes [E] \checkmark A_0 \# A_1 \# As \rangle \rangle = \langle \langle A_0 \ d \otimes [E] \checkmark Rlist \ \langle \langle d \otimes [E] \checkmark A_1 \# As \rangle \rangle \rangle$

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

**lemma** *iterated-combine<sub>d</sub>-Sync<sub>ptick</sub>-simps-bis*:  $\langle As \neq [] \implies \langle \langle d \otimes [E] \checkmark A_0 \# As \rangle \rangle = \langle \langle A_0 \ d \otimes [E] \checkmark Rlist \ \langle \langle d \otimes [E] \checkmark As \rangle \rangle \rangle$   
**and** *iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-simps-bis*:  $\langle Bs \neq [] \implies \langle \langle nd \otimes [E] \checkmark B_0 \# Bs \rangle \rangle = \langle \langle B_0 \ nd \otimes [E] \checkmark Rlist \ \langle \langle nd \otimes [E] \checkmark Bs \rangle \rangle \rangle$   
*<proof>*

#### 8.1.2 First Results

**lemma**  $\tau$ -*iterated-combine-Sync<sub>ptick</sub>*:  
 $\langle \tau \ \langle \langle d \otimes [E] \checkmark As \rangle \rangle = \tau \ \langle \langle d \otimes [E] \checkmark As \rangle \rangle \rangle \ \langle \tau \ \langle \langle nd \otimes [E] \checkmark Bs \rangle \rangle = \tau \ \langle \langle nd \otimes [E] \checkmark Bs \rangle \rangle \rangle$   
*<proof>*

**corollary**  $\varepsilon$ -*iterated-combine-Sync<sub>ptick</sub>*:

$\langle \varepsilon \langle d \otimes [E] \checkmark As \rangle \sigma s = \varepsilon \langle d \otimes [E] As \rangle \sigma s \rangle$   
 $\langle \varepsilon \langle nd \otimes [E] \checkmark Bs \rangle \sigma s = \varepsilon \langle nd \otimes [E] Bs \rangle \sigma s \rangle$   
 (proof)

**corollary**  $\mathcal{R}$ -iterated-combine-Sync<sub>ptick</sub>:

$\langle \mathcal{R}_d \langle d \otimes [E] \checkmark As \rangle = \mathcal{R}_d \langle d \otimes [E] As \rangle \rangle \langle \mathcal{R}_{nd} \langle nd \otimes [E] \checkmark Bs \rangle = \mathcal{R}_{nd} \langle nd \otimes [E] Bs \rangle \rangle$   
 (proof)

**lemma** combine<sub>ListLenL</sub>-Sync<sub>ptick</sub>-combine<sub>Rlist</sub>-Sync<sub>ptick</sub>-eq:

$\langle \varepsilon \langle d \langle A_0 \rangle_{\text{singl} \rightarrow \text{list}} d \otimes [1, E] \checkmark \text{ListLenL } A_1 \rangle \sigma s = \varepsilon \langle A_0 d \otimes [E] \checkmark \text{Rlist } A_1 \rangle \sigma s \rangle$   
 $\langle \tau \langle d \langle A_0 \rangle_{\text{singl} \rightarrow \text{list}} d \otimes [1, E] \checkmark \text{ListLenL } A_1 \rangle (s_0 \# \sigma s) e = \tau \langle A_0 d \otimes [E] \checkmark \text{Rlist } A_1 \rangle (s_0 \# \sigma s) e \rangle$   
 $\langle \varepsilon \langle nd \langle B_0 \rangle_{\text{singl} \rightarrow \text{list}} nd \otimes [1, E] \checkmark \text{ListLenL } B_1 \rangle \sigma s = \varepsilon \langle B_0 nd \otimes [E] \checkmark \text{Rlist } B_1 \rangle \sigma s \rangle$   
 $\langle \tau \langle nd \langle B_0 \rangle_{\text{singl} \rightarrow \text{list}} nd \otimes [1, E] \checkmark \text{ListLenL } B_1 \rangle (s_0 \# \sigma s) e = \tau \langle B_0 nd \otimes [E] \checkmark \text{Rlist } B_1 \rangle (s_0 \# \sigma s) e \rangle$   
 (proof)

**lemma** combine<sub>Pairlist</sub>-Sync<sub>ptick</sub>-and-iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-eq:

$\langle \varepsilon \langle A_0 d \otimes [E] \checkmark \text{Pairlist } A_1 \rangle [s_0, s_1] = \varepsilon \langle d \otimes [E] \checkmark [A_0, A_1] \rangle [s_0, s_1] \rangle$   
 $\langle \tau \langle A_0 d \otimes [E] \checkmark \text{Pairlist } A_1 \rangle [s_0, s_1] e = \tau \langle d \otimes [E] \checkmark [A_0, A_1] \rangle [s_0, s_1] e \rangle$   
 $\langle \varepsilon \langle B_0 nd \otimes [E] \checkmark \text{Pairlist } B_1 \rangle [s_0, s_1] = \varepsilon \langle nd \otimes [E] \checkmark [B_0, B_1] \rangle [s_0, s_1] \rangle$   
 $\langle \tau \langle B_0 nd \otimes [E] \checkmark \text{Pairlist } B_1 \rangle [s_0, s_1] e = \tau \langle nd \otimes [E] \checkmark [B_0, B_1] \rangle [s_0, s_1] e \rangle$   
 (proof)

**lemmas** combine<sub>Pairlist</sub>-Sync<sub>ptick</sub>-and-combine<sub>Rlist</sub>-Sync<sub>ptick</sub>-eq =  
 combine<sub>Pairlist</sub>-Sync<sub>ptick</sub>-and-iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-eq[simplified]

### 8.1.3 Transmission of Properties

**lemma** finite-trans-transmission-to-iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>:

$\langle (\bigwedge A. A \in \text{set } As \implies \text{finite-trans } A) \implies \text{finite-trans } \langle nd \otimes [E] \checkmark As \rangle \rangle$   
 (proof)

**lemma**  $\varrho$ -disjoint- $\varepsilon$ -transmission-to-iterated-combine<sub>d</sub>-Sync<sub>ptick</sub>:

$\langle (\bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A) \implies \varrho\text{-disjoint-}\varepsilon \langle d \otimes [E] \checkmark As \rangle \rangle$   
 (proof)

**lemma**  $\varrho$ -disjoint- $\varepsilon$ -transmission-to-iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>:

$\langle \forall B \in \text{set } Bs. \varrho\text{-disjoint-}\varepsilon B \implies \varrho\text{-disjoint-}\varepsilon \langle nd \otimes [E] \checkmark Bs \rangle \rangle$   
 (proof)

**lemma** *same-length-indep-transmission-to-iterated-combine<sub>d</sub>-Sync<sub>ptick</sub>*:  
 $\langle \text{indep-enabl } A_0 \ s_0 \ E \ \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} \ As \rrbracket \ \sigma s \rangle$   
**if**  $\langle \text{length } \sigma s = \text{length } As \rangle$   
 $\langle \bigwedge i \ j. \llbracket i \leq \text{length } As; j \leq \text{length } As; i \neq j \rrbracket \implies$   
 $\text{indep-enabl } ((A_0 \# As) ! i) ((s_0 \# \sigma s) ! i) \ E \ ((A_0 \# As) ! j) ((s_0 \# \sigma s)$   
 $! j) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\omega$ -iterated-combine<sub>d</sub>-Sync<sub>ptick</sub>* :  
 $\langle \text{length } \sigma s = \text{length } As \implies$   
 $\omega \ \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} \ As \rrbracket \ \sigma s = (\text{if } As = [] \text{ then } \diamond \text{ else those } (\text{map2 } \omega \ As \ \sigma s)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\omega$ -iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>* :  
 $\langle \text{length } \sigma s = \text{length } As \implies$   
 $\omega \ \llbracket_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \ As \rrbracket \ \sigma s =$   
 $(\text{if } As = [] \text{ then } \{\} \text{ else } \{rs. \text{length } rs = \text{length } As \wedge (\forall i < \text{length } As. rs ! i \in \omega$   
 $(As ! i) (\sigma s ! i))\}) \rangle$   
 $\langle \text{proof} \rangle$

## 8.1.4 Normalization

**lemma**  *$\omega$ -iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-det-ndet-conv*:  
 $\langle \text{length } \sigma s = \text{length } As \implies$   
 $\omega \ \llbracket_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \ \text{map } (\lambda A. \llbracket A \rrbracket_{d \mapsto nd}) \ As \rrbracket \ \sigma s = \omega \ \llbracket \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} \ As \rrbracket \rrbracket_{d \mapsto nd} \ \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  *$\tau$ -iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-behaviour-when-indep*:  
 $\langle \text{length } \sigma s = \text{length } As \implies$   
 $(\bigwedge i \ j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket$   
 $\implies \text{indep-enabl } (As ! i) (\sigma s ! i) \ E \ (As ! j) (\sigma s ! j)) \implies$   
 $\tau \ \llbracket \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} \ As \rrbracket \rrbracket_{d \mapsto nd} \ \sigma s \ e = \tau \ \llbracket_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \ \text{map } (\lambda A. \llbracket A \rrbracket_{d \mapsto nd}) \ As \rrbracket \ \sigma s \ e \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P<sub>SKIPS</sub>-iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-behaviour-when-indep*:  
**assumes** *same-length*:  $\langle \text{length } \sigma s = \text{length } As \rangle$   
**and** *indep*:  $\langle \bigwedge i \ j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies$   
 $\text{indep-enabl } (As ! i) (\sigma s ! i) \ E \ (As ! j) (\sigma s ! j) \rangle$   
**shows**  $\langle P_{SKIPS} \llbracket \llbracket_d \otimes \llbracket E \rrbracket_{\checkmark} \ As \rrbracket \rrbracket_d \ \sigma s = P_{SKIPS} \llbracket \llbracket_{nd} \otimes \llbracket E \rrbracket_{\checkmark} \ \text{map } (\lambda A. \llbracket A \rrbracket_{d \mapsto nd})$   
 $As \rrbracket \rrbracket_{nd} \ \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *P-d-iterated-combine<sub>nd</sub>-Sync<sub>ptick</sub>-behaviour-when-indep*:  
**assumes** *same-length*:  $\langle \text{length } \sigma s = \text{length } As \rangle$

**and indep:**  $\langle \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rangle$   
**shows**  $\langle P \langle \langle \langle d \otimes [E]_{\checkmark} As \rangle \rangle_d \sigma s = P \langle \langle \langle nd \otimes [E]_{\checkmark} \text{map } (\lambda A. \langle A \rangle_{d \leftrightarrow nd}) As \rangle \rangle_{nd} \sigma s \rangle$   
 $\langle \text{proof} \rangle$

## 8.2 Compactification Theorems

### 8.2.1 Binary

#### Pair

**theorem**  $P_{SKIPS}\text{-nd-combine}_{Pair}\text{-Sync}_{ptick}$  :  
**fixes**  $E :: \langle 'a \text{ set} \rangle$   
**assumes**  $\varrho\text{-disjoint-}\varepsilon : \langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$   
**defines**  $A\text{-def: } \langle A \equiv \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle$   
**defines**  $P\text{-def: } \langle P \equiv P_{SKIPS} \langle A_0 \rangle_{nd} \rangle$  **and**  $Q\text{-def: } \langle Q \equiv P_{SKIPS} \langle A_1 \rangle_{nd} \rangle$  **and**  
 $S\text{-def: } \langle S \equiv P_{SKIPS} \langle A \rangle_{nd} \rangle$   
**shows**  $\langle P \sigma_0 [E]_{\checkmark Pair} Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P\text{-nd-combine}_{Pair}\text{-Sync}_{ptick}$  :  
 $\langle P \langle A_0 \rangle_{nd} \sigma_0 [E]_{\checkmark Pair} P \langle A_1 \rangle_{nd} \sigma_1 = P \langle \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle_{nd} (\sigma_0, \sigma_1) \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P_{SKIPS}\text{-d-combine}_{Pair}\text{-Sync}_{ptick}$ :  
 $\langle P_{SKIPS} \langle A_0 \rangle_d \sigma_0 [E]_{\checkmark Pair} P_{SKIPS} \langle A_1 \rangle_d \sigma_1 = P_{SKIPS} \langle \langle \langle A_0 \text{ d} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle_d (\sigma_0, \sigma_1) \rangle$   
**if**  $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle$  **and**  $\langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$  **and**  $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P\text{-d-combine}_{Pair}\text{-Sync}_{ptick}$ :  
 $\langle P \langle A_0 \rangle_d \sigma_0 [E]_{\checkmark Pair} P \langle A_1 \rangle_d \sigma_1 = P \langle \langle \langle A_0 \text{ d} \otimes [E]_{\checkmark Pair} A_1 \rangle \rangle_d (\sigma_0, \sigma_1) \rangle$   
**if**  $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
 $\langle \text{proof} \rangle$

#### Pairlist

**theorem**  $P_{SKIPS}\text{-nd-combine}_{Pairlist}\text{-Sync}_{ptick}$  :  
**fixes**  $E :: \langle 'a \text{ set} \rangle$   
**assumes**  $\varrho\text{-disjoint-}\varepsilon : \langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$   
**shows**  $\langle P_{SKIPS} \langle A_0 \rangle_{nd} \sigma_0 [E]_{\checkmark Pairlist} P_{SKIPS} \langle A_1 \rangle_{nd} \sigma_1 = P_{SKIPS} \langle \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pairlist} A_1 \rangle \rangle_{nd} [\sigma_0, \sigma_1] \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P\text{-nd-combine}_{Pairlist}\text{-Sync}_{ptick}$  :  
 $\langle P \langle A_0 \rangle_{nd} \sigma_0 [E]_{\checkmark Pairlist} P \langle A_1 \rangle_{nd} \sigma_1 = P \langle \langle \langle A_0 \text{ nd} \otimes [E]_{\checkmark Pairlist} A_1 \rangle \rangle_{nd} [\sigma_0, \sigma_1] \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{Pairlist}$ -Sync $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pairlist} P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \ d \otimes \llbracket E \rrbracket_{\checkmark Pairlist} A_1 \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$   
**if**  $\langle \varrho$ -disjoint- $\varepsilon$   $A_0 \rangle$  **and**  $\langle \varrho$ -disjoint- $\varepsilon$   $A_1 \rangle$  **and**  $\langle indep$ -enabl  $A_0 \ \sigma_0 \ E \ A_1 \ \sigma_1 \rangle$   
 $\langle proof \rangle$

**corollary**  $P$ - $d$ -combine $_{Pairlist}$ -Sync $_{ptick}$  :  
 $\langle P\langle A_0 \rangle_d \sigma_0 \llbracket E \rrbracket_{\checkmark Pairlist} P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \ d \otimes \llbracket E \rrbracket_{\checkmark Pairlist} A_1 \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$   
**if**  $\langle indep$ -enabl  $A_0 \ \sigma_0 \ E \ A_1 \ \sigma_1 \rangle$   
 $\langle proof \rangle$

## 8.2.2 Rlist

**theorem**  $P_{SKIPS}$ - $nd$ -combine $_{Rlist}$ -Sync $_{ptick}$  :  
**fixes**  $E :: \langle 'a \ set \rangle$   
**assumes**  $\varrho$ -disjoint- $\varepsilon$  :  $\langle \varrho$ -disjoint- $\varepsilon$   $A_0 \rangle \ \langle \varrho$ -disjoint- $\varepsilon$   $A_1 \rangle$   
**defines**  $A$ -def:  $\langle A \equiv \langle A_0 \ nd \otimes \llbracket E \rrbracket_{\checkmark Rlist} A_1 \rangle \rangle$   
**defines**  $P$ -def:  $\langle P \equiv P_{SKIPS}\langle A_0 \rangle_{nd} \rangle$  **and**  $Q$ -def:  $\langle Q \equiv P_{SKIPS}\langle A_1 \rangle_{nd} \rangle$  **and**  
 $S$ -def:  $\langle S \equiv P_{SKIPS}\langle A \rangle_{nd} \rangle$   
**shows**  $\langle P \ \sigma_0 \llbracket E \rrbracket_{\checkmark Rlist} \ Q \ \sigma_s = S \ (\sigma_0 \ \# \ \sigma_s) \rangle$   
 $\langle proof \rangle$

**corollary**  $P$ - $nd$ -combine $_{Rlist}$ -Sync $_{ptick}$  :  
 $\langle P\langle A_0 \rangle_{nd} \ \sigma_0 \llbracket E \rrbracket_{\checkmark Rlist} \ P\langle A_1 \rangle_{nd} \ \sigma_s = P\langle\langle A_0 \ nd \otimes \llbracket E \rrbracket_{\checkmark Rlist} A_1 \rangle\rangle_{nd} \ (\sigma_0 \ \# \ \sigma_s) \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{Rlist}$ -Sync $_{ptick}$ :  
 $\langle P_{SKIPS}\langle A_0 \rangle_d \ \sigma_0 \llbracket E \rrbracket_{\checkmark Rlist} \ P_{SKIPS}\langle A_1 \rangle_d \ \sigma_s = P_{SKIPS}\langle\langle A_0 \ d \otimes \llbracket E \rrbracket_{\checkmark Rlist} A_1 \rangle\rangle_d \ (\sigma_0 \ \# \ \sigma_s) \rangle$   
**if**  $\langle \varrho$ -disjoint- $\varepsilon$   $A_0 \rangle$  **and**  $\langle \varrho$ -disjoint- $\varepsilon$   $A_1 \rangle$  **and**  $\langle indep$ -enabl  $A_0 \ \sigma_0 \ E \ A_1 \ \sigma_s \rangle$   
 $\langle proof \rangle$

**corollary**  $P$ - $d$ -combine $_{Rlist}$ -Sync $_{ptick}$ :  
 $\langle P\langle A_0 \rangle_d \ \sigma_0 \llbracket E \rrbracket_{\checkmark Rlist} \ P\langle A_1 \rangle_d \ \sigma_s = P\langle\langle A_0 \ d \otimes \llbracket E \rrbracket_{\checkmark Rlist} A_1 \rangle\rangle_d \ (\sigma_0 \ \# \ \sigma_s) \rangle$   
**if**  $\langle indep$ -enabl  $A_0 \ \sigma_0 \ E \ A_1 \ \sigma_s \rangle$   
 $\langle proof \rangle$

## 8.2.3 ListslenL

**theorem**  $P_{SKIPS}$ - $nd$ -combine $_{ListslenL}$ -Sync $_{ptick}$  :  
**fixes**  $E :: \langle 'a \ set \rangle$   
**assumes** same-length-reach0 :  $\langle \bigwedge \sigma_0'. \ \sigma_0' \in \mathcal{R}_{nd} \ A_0 \ \sigma_0 \implies length \ \sigma_0' = len_0 \rangle$   
**and** same-length-term0 :  $\langle \bigwedge \sigma_0' \ rs. \ \sigma_0' \in \mathcal{R}_{nd} \ A_0 \ \sigma_0 \implies rs \in \omega \ A_0 \ \sigma_0' \implies length \ rs = len_0 \rangle$   
**and**  $\varrho$ -disjoint- $\varepsilon$  :  $\langle \varrho$ -disjoint- $\varepsilon$   $A_0 \rangle \ \langle \varrho$ -disjoint- $\varepsilon$   $A_1 \rangle$   
**defines**  $A$ -def:  $\langle A \equiv \langle A_0 \ nd \otimes \llbracket len_0, E \rrbracket_{\checkmark ListslenL} A_1 \rangle \rangle$   
**defines**  $P$ -def:  $\langle P \equiv P_{SKIPS}\langle A_0 \rangle_{nd} \rangle$  **and**  $Q$ -def:  $\langle Q \equiv P_{SKIPS}\langle A_1 \rangle_{nd} \rangle$  **and**  
 $S$ -def:  $\langle S \equiv P_{SKIPS}\langle A \rangle_{nd} \rangle$   
**shows**  $\langle P \ \sigma_0 \ len_0 \llbracket E \rrbracket_{\checkmark ListslenL} \ Q \ \sigma_1 = S \ (\sigma_0 \ @ \ \sigma_1) \rangle$   
 $\langle proof \rangle$

**corollary**  $P$ - $nd$ -combine $_{ListslenL}$ -Sync $_{ptick}$  :  
 $\langle P\langle A_0 \rangle_{nd} \sigma_0 \text{ len}_0 \llbracket E \rrbracket_{\checkmark ListslenL} P\langle A_1 \rangle_{nd} \sigma_1 = P\langle\langle A_0 \text{ nd} \otimes \llbracket len_0, E \rrbracket_{\checkmark ListslenL} A_1 \rangle\rangle_{nd} (\sigma_0 \text{ @ } \sigma_1) \rangle$   
**if** same-length-reach $0$  :  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_{nd} A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{ListslenL}$ -Sync $_{ptick}$  :  
**assumes** same-length-reach $0$  :  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$   
**and** same-length-term $0$  :  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies \omega A_0 \sigma_0' \neq \diamond \implies \text{length } \llbracket \omega A_0 \sigma_0' \rrbracket = \text{len}_0 \rangle$   
**and**  $\varrho$ -disjoint- $\varepsilon$  :  $\langle \varrho\text{-disjoint-}\varepsilon A_0 \rangle \langle \varrho\text{-disjoint-}\varepsilon A_1 \rangle$   
**and** indep-enabl :  $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
**shows**  $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 \text{ len}_0 \llbracket E \rrbracket_{\checkmark ListslenL} P_{SKIPS}\langle A_1 \rangle_d \sigma_1 = P_{SKIPS}\langle\langle A_0 \text{ d} \otimes \llbracket len_0, E \rrbracket_{\checkmark ListslenL} A_1 \rangle\rangle_d (\sigma_0 \text{ @ } \sigma_1) \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P$ - $d$ -combine $_{ListslenL}$ -Sync $_{ptick}$  :  
**assumes** same-length-reach $0$  :  $\langle \bigwedge \sigma_0'. \sigma_0' \in \mathcal{R}_d A_0 \sigma_0 \implies \text{length } \sigma_0' = \text{len}_0 \rangle$   
**and** indep-enabl :  $\langle \text{indep-enabl } A_0 \sigma_0 E A_1 \sigma_1 \rangle$   
**shows**  $\langle P\langle A_0 \rangle_d \sigma_0 \text{ len}_0 \llbracket E \rrbracket_{\checkmark ListslenL} P\langle A_1 \rangle_d \sigma_1 = P\langle\langle A_0 \text{ d} \otimes \llbracket len_0, E \rrbracket_{\checkmark ListslenL} A_1 \rangle\rangle_d (\sigma_0 \text{ @ } \sigma_1) \rangle$   
 $\langle \text{proof} \rangle$

## 8.2.4 Multiple

**theorem**  $P_{SKIPS}$ - $nd$ -compactification-Sync $_{ptick}$  :  
 $\langle \text{length } \sigma s = \text{length } As \implies (\bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A) \implies \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in \text{@ zip } \sigma s As. P_{SKIPS}\langle A \rangle_{nd} \sigma = P_{SKIPS}\langle\langle nd \otimes \llbracket E \rrbracket_{\checkmark} As \rangle\rangle_{nd} \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P$ - $nd$ -compactification-Sync $_{ptick}$  :  
 $\langle \text{length } \sigma s = \text{length } As \implies \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in \text{@ zip } \sigma s As. P\langle A \rangle_{nd} \sigma = P\langle\langle nd \otimes \llbracket E \rrbracket_{\checkmark} As \rangle\rangle_{nd} \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -compactification-Sync $_{ptick}$  :  
 $\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A. A \in \text{set } As \implies \varrho\text{-disjoint-}\varepsilon A; \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in \text{@ zip } \sigma s As. P_{SKIPS}\langle A \rangle_d \sigma = P_{SKIPS}\langle\langle d \otimes \llbracket E \rrbracket_{\checkmark} As \rangle\rangle_d \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**corollary**  $P$ - $d$ -compactification-Sync $_{ptick}$  :  
 $\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge i j. \llbracket i < \text{length } As; j < \text{length } As; i \neq j \rrbracket \implies \text{indep-enabl } (As ! i) (\sigma s ! i) E (As ! j) (\sigma s ! j) \rrbracket \implies \llbracket E \rrbracket_{\checkmark} (\sigma, A) \in \text{@ zip } \sigma s As. P\langle A \rangle_d \sigma = P\langle\langle d \otimes \llbracket E \rrbracket_{\checkmark} As \rangle\rangle_d \sigma s \rangle$

$\langle proof \rangle$

### 8.3 Derived Versions

**lemma**  *$P_{SKIPs}$ -nd-compactification- $Sync_{ptick}$ -upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n]. P = P_{SKIPs} \llbracket \llbracket nd \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rrbracket \rrbracket_{nd}$   
 $(\text{replicate } n \ 0) \rangle$

**if**  $\langle \bigwedge i. i < n \implies \varrho\text{-disjoint-}\varepsilon (A \ i) \rangle$   
 $\langle \bigwedge i. i < n \implies P_{SKIPs} \llbracket A \ i \rrbracket_{nd} \ 0 = Q \ i \rangle$

$\langle proof \rangle$

**lemma**  *$P$ -nd-compactification- $Sync_{ptick}$ -upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n]. P = P \llbracket \llbracket nd \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rrbracket \rrbracket_{nd} (\text{replicate}$   
 $n \ 0) \rangle$

**if**  $\langle \bigwedge i. i < n \implies P \llbracket A \ i \rrbracket_{nd} \ 0 = Q \ i \rangle$

$\langle proof \rangle$

**lemma**  *$P_{SKIPs}$ -d-compactification- $Sync_{ptick}$ -upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n]. P = P_{SKIPs} \llbracket \llbracket d \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rrbracket \rrbracket_d (\text{replicate}$   
 $n \ 0) \rangle$

**if**  $\langle \bigwedge i. i < n \implies \varrho\text{-disjoint-}\varepsilon (A \ i) \rangle$   
 $\langle \bigwedge i \ j. i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) \ 0 \ E \ (A \ j) \ 0 \rangle$   
 $\langle \bigwedge i. i < n \implies P_{SKIPs} \llbracket A \ i \rrbracket_d \ 0 = Q \ i \rangle$

$\langle proof \rangle$

**lemma**  *$P$ -d-compactification- $Sync_{ptick}$ -upt-version:*

$\langle \llbracket E \rrbracket_{\checkmark} P \in @ \text{map } Q [0..<n]. P = P \llbracket \llbracket d \otimes \llbracket E \rrbracket_{\checkmark} \text{map } A [0..<n] \rrbracket \rrbracket_d (\text{replicate } n$   
 $0) \rangle$

**if**  $\langle \bigwedge i \ j. i < n \implies j < n \implies i \neq j \implies \text{indep-enabl } (A \ i) \ 0 \ E \ (A \ j) \ 0 \rangle$   
 $\langle \bigwedge i. i < n \implies P \llbracket A \ i \rrbracket_d \ 0 = Q \ i \rangle$

$\langle proof \rangle$

### 8.4 More on Iterated Combine and Events

Through  $\tau$ -iterated-combine- $Sync_{ptick}$   $\varepsilon$ -iterated-combine- $Sync_{ptick}$   $\mathcal{R}$ -iterated-combine- $Sync_{ptick}$ ,  
we immediately recover the results proven in *HOL-CSP-Proc-Omata.Compactification-Synchronization-F*



## Chapter 9

# Combining Automata for Sequential Composition Generalized

### 9.1 Definitions

### 9.2 General Patterns

**definition** *combine<sub>d</sub>-Seq-ε* ::

$\langle [(\sigma_0, 'a, 'r, '\alpha_0) A_d\text{-scheme}, 'r \Rightarrow (\sigma_1, 'a, 's, '\alpha_1) A_d\text{-scheme}, '\sigma \Rightarrow \sigma_0, '\sigma \Rightarrow \sigma_1, '\sigma] \Rightarrow 'a \text{ set} \rangle$

**where**  $\langle \text{combine}_d\text{-Seq-}\varepsilon A_0 A_1 i_0 i_1 \sigma s \equiv$   
    if  $i_0 \sigma s \in \varrho A_0$   
    then if  $i_1 \sigma s \in \varrho (A_1 [\omega A_0 (i_0 \sigma s)])$   
        then {}  
        else  $\varepsilon (A_1 [\omega A_0 (i_0 \sigma s)]) (i_1 \sigma s)$   
    else  $\varepsilon A_0 (i_0 \sigma s) \rangle$

**definition** *combine<sub>nd</sub>-Seq-ε* ::

$\langle [(\sigma_0, 'a, 'r, '\alpha_0) A_{nd}\text{-scheme}, 'r \Rightarrow (\sigma_1, 'a, 's, '\alpha_1) A_{nd}\text{-scheme}, '\sigma \Rightarrow \sigma_0, '\sigma \Rightarrow \sigma_1, '\sigma] \Rightarrow 'a \text{ set} \rangle$

**where**  $\langle \text{combine}_{nd}\text{-Seq-}\varepsilon A_0 A_1 i_0 i_1 \sigma s \equiv$   
    if  $i_0 \sigma s \in \varrho A_0$   
    then if  $i_1 \sigma s \in (\bigcup r \in \omega A_0 (i_0 \sigma s). \varrho (A_1 r))$   
        then {}  
        else  $(\bigcup r \in \omega A_0 (i_0 \sigma s). \varepsilon (A_1 r) (i_1 \sigma s))$   
    else  $\varepsilon A_0 (i_0 \sigma s) \rangle$

**lemmas** *combine-Seq-ε-defs* = *combine<sub>d</sub>-Seq-ε-def combine<sub>nd</sub>-Seq-ε-def*

**fun** *combine<sub>d</sub>-Seq* ::

$\langle [(\sigma_0, 'e, 'r, '\alpha_0) A_d\text{-scheme}, 'r \Rightarrow (\sigma_1, 'e, 's, '\alpha_1) A_d\text{-scheme},$

$\langle ' \sigma \Rightarrow ' \sigma_0, ' \sigma \Rightarrow ' \sigma_1, ' \sigma_0 \Rightarrow ' \sigma_1 \Rightarrow ' \sigma \Rightarrow (' \sigma, ' e, ' s) A_d \rangle$   
**and** *combine<sub>nd</sub>-Seq* ::  
 $\langle [ (' \sigma_0, ' e, ' r, ' \alpha_0) A_{nd}\text{-scheme}, ' r \Rightarrow (' \sigma_1, ' e, ' s, ' \alpha_1) A_{nd}\text{-scheme}, ' \sigma \Rightarrow ' \sigma_0, ' \sigma \Rightarrow ' \sigma_1, ' \sigma_0 \Rightarrow ' \sigma_1 \Rightarrow ' \sigma \Rightarrow (' \sigma, ' e, ' s) A_{nd} ] \rangle$   
**where**  $\langle \text{combine}_d\text{-Seq } A_0 A_1 i_0 i_1 f =$   
 $(\tau = \lambda \sigma s e. \text{ if } i_0 \sigma s \in \varrho A_0$   
 $\quad \text{then if } i_1 \sigma s \in \varrho (A_1 [\omega A_0 (i_0 \sigma s)])$   
 $\quad \quad \text{then } \diamond$   
 $\quad \quad \text{else update-right } (A_1 [\omega A_0 (i_0 \sigma s)]) (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-opt}$   
 $f) (\lambda \sigma. [\sigma])$   
 $\quad \quad \text{else update-left } A_0 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-opt } f) (\lambda \sigma. [\sigma]),$   
 $\quad \quad \omega = \lambda \sigma s. \text{ case } \omega A_0 (i_0 \sigma s) \text{ of } \diamond \Rightarrow \diamond \mid [r] \Rightarrow \omega (A_1 r) (i_1 \sigma s) \rangle$   
 $\mid \langle \text{combine}_{nd}\text{-Seq } A_0 A_1 i_0 i_1 f =$   
 $(\tau = \lambda \sigma s e. \text{ if } i_0 \sigma s \in \varrho A_0$   
 $\quad \text{then if } i_1 \sigma s \in (\bigcup r \in \omega A_0 (i_0 \sigma s). \varrho (A_1 r))$   
 $\quad \quad \text{then } \{ \}$   
 $\quad \quad \text{else } (\bigcup r \in \omega A_0 (i_0 \sigma s). \text{ update-right } (A_1 r) (i_0 \sigma s) (i_1 \sigma s) e$   
 $(f\text{-up-set } f) (\lambda \sigma. \{ \sigma \}))$   
 $\quad \quad \text{else update-left } A_0 (i_0 \sigma s) (i_1 \sigma s) e (f\text{-up-set } f) (\lambda \sigma. \{ \sigma \}),$   
 $\quad \quad \omega = \lambda \sigma s. \bigcup r \in \omega A_0 (i_0 \sigma s). \omega (A_1 r) (i_1 \sigma s) \rangle$

### 9.3 Specializations

**definition** *combine<sub>d</sub>Pairlist-Seq<sub>ptick</sub>* ::  
 $\langle [ (' \sigma, ' e, ' r, ' \alpha) A_d\text{-scheme}, ' r \Rightarrow (' \sigma, ' e, ' s, ' \beta) A_d\text{-scheme} ] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_d \rangle$   
**where**  $\langle \text{combine}_d\text{Pairlist-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_d\text{-Seq } A_0 A_1 \text{ hd } (\lambda \sigma s. \text{hd } (tl \sigma s)) (\lambda s t. [s, t]) \rangle$

**definition** *combine<sub>nd</sub>Pairlist-Seq<sub>ptick</sub>* ::  
 $\langle [ (' \sigma, ' e, ' r, ' \alpha) A_{nd}\text{-scheme}, ' r \Rightarrow (' \sigma, ' e, ' s, ' \beta) A_{nd}\text{-scheme} ] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_{nd} \rangle$   
**where**  $\langle \text{combine}_{nd}\text{Pairlist-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_{nd}\text{-Seq } A_0 A_1 \text{ hd } (\lambda \sigma s. \text{hd } (tl \sigma s)) (\lambda s t. [s, t]) \rangle$

**definition** *combine<sub>d</sub>Pair-Seq<sub>ptick</sub>* ::  
 $\langle [ (' \sigma_0, ' e, ' r, ' \alpha) A_d\text{-scheme}, ' r \Rightarrow (' \sigma_1, ' e, ' s, ' \beta) A_d\text{-scheme} ] \Rightarrow (' \sigma_0 \times ' \sigma_1, ' e, ' s) A_d \rangle$   
**where**  $\langle \text{combine}_d\text{Pair-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_d\text{-Seq } A_0 A_1 \text{ fst snd Pair} \rangle$

**definition** *combine<sub>nd</sub>Pair-Seq<sub>ptick</sub>* ::  
 $\langle [ (' \sigma_0, ' e, ' r, ' \alpha) A_{nd}\text{-scheme}, ' r \Rightarrow (' \sigma_1, ' e, ' s, ' \beta) A_{nd}\text{-scheme} ] \Rightarrow (' \sigma_0 \times ' \sigma_1, ' e, ' s) A_{nd} \rangle$   
**where**  $\langle \text{combine}_{nd}\text{Pair-Seq}_{ptick} A_0 A_1 \equiv \text{combine}_{nd}\text{-Seq } A_0 A_1 \text{ fst snd Pair} \rangle$

**definition** *combine<sub>d</sub>ListslenL-Seq<sub>ptick</sub>* ::  
 $\langle [ (' \sigma \text{ list}, ' e, ' r, ' \alpha) A_d\text{-scheme}, \text{nat}, ' r \Rightarrow (' \sigma \text{ list}, ' e, ' s, ' \beta) A_d\text{-scheme} ] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_d \rangle$   
**where**  $\langle \text{combine}_d\text{ListslenL-Seq}_{ptick} A_0 \text{ len}_0 A_1 \equiv \text{combine}_d\text{-Seq } A_0 A_1 (\text{take len}_0) (\text{drop len}_0) (@) \rangle$

**definition** *combine<sub>nd</sub>ListslenL-Seq<sub>ptick</sub>* ::  
 $\langle [ (' \sigma \text{ list}, ' e, ' r, ' \alpha) A_{nd}\text{-scheme}, \text{nat}, ' r \Rightarrow (' \sigma \text{ list}, ' e, ' s, ' \beta) A_{nd}\text{-scheme} ] \Rightarrow (' \sigma \text{ list}, ' e, ' s) A_{nd} \rangle$

*list, 'e, 's) A<sub>nd</sub>*  
**where**  $\langle \text{combine}_{nd\text{ListslenL-Seqptick}} A_0 \text{ len}_0 A_1 \equiv \text{combine}_{nd\text{-Seq}} A_0 A_1 (\text{take len}_0) (\text{drop len}_0) (@) \rangle$

**definition** *combine<sub>dRlist-Seqptick</sub>* ::  
 $\langle [(\sigma, 'e, 'r, 'a) A_d\text{-scheme}, 'r \Rightarrow (\sigma \text{ list}, 'e, 's, 'b) A_d\text{-scheme}] \Rightarrow (\sigma \text{ list}, 'e, 's) A_d \rangle$

**where**  $\langle \text{combine}_{dRlist-Seqptick} A_0 A_1 \equiv \text{combine}_{d\text{-Seq}} A_0 A_1 \text{ hd tl } (\#) \rangle$

**definition** *combine<sub>ndRlist-Seqptick</sub>* ::  
 $\langle [(\sigma, 'e, 'r, 'a) A_{nd}\text{-scheme}, 'r \Rightarrow (\sigma \text{ list}, 'e, 's, 'b) A_{nd}\text{-scheme}] \Rightarrow (\sigma \text{ list}, 'e, 's) A_{nd} \rangle$

**where**  $\langle \text{combine}_{ndRlist-Seqptick} A_0 A_1 \equiv \text{combine}_{nd\text{-Seq}} A_0 A_1 \text{ hd tl } (\#) \rangle$

**lemmas** *combine<sub>Pairlist-Seqptick-defs</sub>* = *combine<sub>dPairlist-Seqptick-def</sub>* *combine<sub>ndPairlist-Seqptick-def</sub>*  
**and** *combine<sub>Pair-Seqptick-defs</sub>* = *combine<sub>dPair-Seqptick-def</sub>* *combine<sub>ndPair-Seqptick-def</sub>*  
**and** *combine<sub>ListslenL-Seqptick</sub>* = *combine<sub>dListslenL-Seqptick-def</sub>* *combine<sub>ndListslenL-Seqptick-def</sub>*  
**and** *combine<sub>Rlist-Seqptick-defs</sub>* = *combine<sub>dRlist-Seqptick-def</sub>* *combine<sub>ndRlist-Seqptick-def</sub>*

**lemmas** *combine-Seq-defs* =  
*combine<sub>Pairlist-Seqptick-defs</sub>* *combine<sub>Pair-Seqptick-defs</sub>* *combine<sub>ListslenL-Seqptick</sub>*  
*combine<sub>Rlist-Seqptick-defs</sub>*

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

**notation** *combine<sub>dPairlist-Seqptick</sub>* ( $\langle \langle - \text{ d}\otimes; \checkmark_{\text{Pairlist}} - \rangle \rangle [0, 0]$ )  
**notation** *combine<sub>ndPairlist-Seqptick</sub>* ( $\langle \langle - \text{ nd}\otimes; \checkmark_{\text{Pairlist}} - \rangle \rangle [0, 0]$ )  
**notation** *combine<sub>dPair-Seqptick</sub>* ( $\langle \langle - \text{ d}\otimes; \checkmark_{\text{Pair}} - \rangle \rangle [0, 0]$ )  
**notation** *combine<sub>ndPair-Seqptick</sub>* ( $\langle \langle - \text{ nd}\otimes; \checkmark_{\text{Pair}} - \rangle \rangle [0, 0]$ )  
**notation** *combine<sub>dListslenL-Seqptick</sub>* ( $\langle \langle - \text{ d}\otimes; \checkmark_{\text{ListslenL}} - \rangle \rangle [0, 0, 0]$ )  
**notation** *combine<sub>ndListslenL-Seqptick</sub>* ( $\langle \langle - \text{ nd}\otimes; \checkmark_{\text{ListslenL}} - \rangle \rangle [0, 0, 0]$ )  
**notation** *combine<sub>dRlist-Seqptick</sub>* ( $\langle \langle - \text{ d}\otimes; \checkmark_{\text{Rlist}} - \rangle \rangle [0, 0]$ )  
**notation** *combine<sub>ndRlist-Seqptick</sub>* ( $\langle \langle - \text{ nd}\otimes; \checkmark_{\text{Rlist}} - \rangle \rangle [0, 0]$ )

**end**

**unbundle** *combine-Seq-syntax*

## 9.4 First Properties

**lemma**  $\varepsilon\text{-combine}_{\text{Pairlist-Seqptick}}$  :  
 $\langle \varepsilon \langle A_0 \text{ d}\otimes; \checkmark_{\text{Pairlist}} A_1 \rangle \sigma s = \text{combine}_{d\text{-Seq-}\varepsilon} A_0 A_1 \text{ hd } (\text{hd} \circ \text{tl}) \sigma s \rangle$   
 $\langle \varepsilon \langle B_0 \text{ nd}\otimes; \checkmark_{\text{Pairlist}} B_1 \rangle \sigma s = \text{combine}_{nd\text{-Seq-}\varepsilon} B_0 B_1 \text{ hd } (\text{hd} \circ \text{tl}) \sigma s \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\varepsilon\text{-combine}_{\text{Pair-Seqptick}}$  :  
 $\langle \varepsilon \langle A_0 \text{ d}\otimes; \checkmark_{\text{Pair}} A_1 \rangle \sigma s = \text{combine}_{d\text{-Seq-}\varepsilon} A_0 A_1 \text{ fst snd } \sigma s \rangle$

$\langle \varepsilon \langle B_0 \text{ nd} \otimes; \checkmark \text{Pair } B_1 \rangle \sigma s = \text{combine}_{\text{nd-Seq-}\varepsilon} B_0 B_1 \text{ fst snd } \sigma s \rangle$   
 (proof)

**lemma**  $\varepsilon$ -combine $_{\text{ListslenL-Seqptick}}$  :

$\langle \varepsilon \langle A_0 \text{ d} \otimes \text{len}_0; \text{ListslenL } A_1 \rangle \sigma s = \text{combine}_{\text{d-Seq-}\varepsilon} A_0 A_1 (\text{take len}_0) (\text{drop len}_0) \sigma s \rangle$

$\langle \varepsilon \langle B_0 \text{ nd} \otimes \text{len}_0; \text{ListslenL } B_1 \rangle \sigma s = \text{combine}_{\text{nd-Seq-}\varepsilon} B_0 B_1 (\text{take len}_0) (\text{drop len}_0) \sigma s \rangle$   
 (proof)

**lemma**  $\varepsilon$ -combine $_{\text{Rlist-Seqptick}}$  :

$\langle \varepsilon \langle A_0 \text{ d} \otimes; \checkmark \text{Rlist } A_1 \rangle \sigma s = \text{combine}_{\text{d-Seq-}\varepsilon} A_0 A_1 \text{ hd tl } \sigma s \rangle$

$\langle \varepsilon \langle B_0 \text{ nd} \otimes; \checkmark \text{Rlist } B_1 \rangle \sigma s = \text{combine}_{\text{nd-Seq-}\varepsilon} B_0 B_1 \text{ hd tl } \sigma s \rangle$   
 (proof)

### 9.4.1 Reachability

**lemma**  $\mathcal{R}_d$ -combine $_{\text{ListslenL-Seqptick-subset}}$ :

$\langle \mathcal{R}_d \langle A_0 \text{ d} \otimes \text{len}_0; \text{ListslenL } A_1 \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )

if same-length- $\mathcal{R}_d$ :  $\langle \bigwedge t_0. t_0 \in \mathcal{R}_d A_0 s_0 \implies \text{length } t_0 = \text{len}_0 \rangle$   
 (proof)

**lemma**  $\mathcal{R}_{\text{nd}}$ -combine $_{\text{ndListslenL-Seqptick-subset}}$ :

$\langle \mathcal{R}_{\text{nd}} \langle A_0 \text{ nd} \otimes \text{len}_0; \text{ListslenL } A_1 \rangle (s_0 @ s_1) \subseteq \{t_0 @ t_1 \mid t_0 t_1. t_0 \in \mathcal{R}_{\text{nd}} A_0 s_0\} \rangle$   
 (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )

if same-length- $\mathcal{R}_{\text{nd}}$ :  $\langle \bigwedge t_0. t_0 \in \mathcal{R}_{\text{nd}} A_0 s_0 \implies \text{length } t_0 = \text{len}_0 \rangle$   
 (proof)

**lemma**  $\mathcal{R}_d$ -combine $_{\text{Pairlist-Seqptick-subset}}$ :

$\langle \mathcal{R}_d \langle A_0 \text{ d} \otimes; \checkmark \text{Pairlist } A_1 \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )

(proof)

**lemma**  $\mathcal{R}_{\text{nd}}$ -combine $_{\text{ndPairlist-Seqptick-subset}}$ :

$\langle \mathcal{R}_{\text{nd}} \langle A_0 \text{ nd} \otimes; \checkmark \text{Pairlist } A_1 \rangle [s_0, s_1] \subseteq \{[t_0, t_1] \mid t_0 t_1. t_0 \in \mathcal{R}_{\text{nd}} A_0 s_0\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )

(proof)

**lemma**  $\mathcal{R}_d$ -combine $_{\text{Pair-Seqptick-subset}}$ :

$\langle \mathcal{R}_d \langle A_0 \text{ d} \otimes; \checkmark \text{Pair } A_1 \rangle (s_0, s_1) \subseteq \{(t_0, t_1) \mid t_0 t_1. t_0 \in \mathcal{R}_d A_0 s_0\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )

(proof)

**lemma**  $\mathcal{R}_{\text{nd}}$ -combine $_{\text{ndPair-Seqptick-subset}}$ :

$\langle \mathcal{R}_{\text{nd}} \langle A_0 \text{ nd} \otimes; \checkmark \text{Pair } A_1 \rangle (s_0, s_1) \subseteq \{(t_0, t_1) \mid t_0 t_1. t_0 \in \mathcal{R}_{\text{nd}} A_0 s_0\} \rangle$  (is  $\langle ?S_A \subseteq \rightarrow \rangle$ )

(proof)

$\langle \text{proof} \rangle$

**lemma**  $\mathcal{R}_d\text{-combine}_{dRlist}\text{-Seq}_{ptick}\text{-subset}$ :

$\langle \mathcal{R}_d \langle \langle A_0 \ d \otimes; \checkmark_{Rlist} A_1 \rangle \rangle (s_0 \# s_1) \subseteq \{t_0 \# t_1 \mid t_0 \ t_1. t_0 \in \mathcal{R}_d A_0 \ s_0\} \rangle$  (is  $\langle ?S_A \subseteq - \rangle$ )  
 $\langle \text{proof} \rangle$

**lemma**  $\mathcal{R}_{nd}\text{-combine}_{ndRlist}\text{-Seq}_{ptick}\text{-subset}$ :

$\langle \mathcal{R}_{nd} \langle \langle A_0 \ nd \otimes; \checkmark_{Rlist} A_1 \rangle \rangle (s_0 \# s_1) \subseteq \{t_0 \# t_1 \mid t_0 \ t_1. t_0 \in \mathcal{R}_{nd} A_0 \ s_0\} \rangle$  (is  $\langle ?S_A \subseteq - \rangle$ )  
 $\langle \text{proof} \rangle$

## 9.5 Normalization

**lemma**  $\tau\text{-combine}_{Pairlist}\text{-Seq}_{ptick}\text{-behaviour}$ :

$\langle \tau \langle \langle \langle A_0 \ d \otimes; \checkmark_{Pairlist} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} [s_0, s_1] \ e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \ nd \otimes; \checkmark_{Pairlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle \rangle [s_0, s_1] \ e \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\tau\text{-combine}_{Pair}\text{-Seq}_{ptick}\text{-behaviour}$ :

$\langle \tau \langle \langle \langle A_0 \ d \otimes; \checkmark_{Pair} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (s_0, s_1) \ e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \ nd \otimes; \checkmark_{Pair} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle \rangle (s_0, s_1) \ e \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\tau\text{-combine}_{ListslenL}\text{-Seq}_{ptick}\text{-behaviour}$ :

$\langle \tau \langle \langle \langle A_0 \ d \otimes; \text{len}_0; \text{ListslenL} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (\sigma s_0 \ @ \ \sigma s_1) \ e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \ nd \otimes; \text{len}_0; \text{ListslenL} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle \rangle (\sigma s_0 \ @ \ \sigma s_1) \ e \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\tau\text{-combine}_{Rlist}\text{-Seq}_{ptick}\text{-behaviour}$ :

$\langle \tau \langle \langle \langle A_0 \ d \otimes; \checkmark_{Rlist} A_1 \rangle \rangle \rangle_{d \hookrightarrow nd} (s_0 \# \sigma s_1) \ e = \tau \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \ nd \otimes; \checkmark_{Rlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle \rangle (s_0 \# \sigma s_1) \ e \rangle$   
 $\langle \text{proof} \rangle$

Behaviour of normalisations

**lemma**  $P_{SKIPS}\text{-combine}_{Pairlist}\text{-Seq}_{ptick}\text{-behaviour}$ :

$\langle P_{SKIPS} \langle \langle \langle A_0 \ d \otimes; \checkmark_{Pairlist} A_1 \rangle \rangle \rangle_d [s_0, s_1] = P_{SKIPS} \langle \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \ nd \otimes; \checkmark_{Pairlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} [s_0, s_1] \rangle$   
 $\langle \text{proof} \rangle$

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

$\langle P \langle \langle \langle A_0 \ d \otimes; \checkmark_{Pairlist} A_1 \rangle \rangle \rangle_d [s_0, s_1] = P \langle \langle \langle \langle A_0 \rangle \rangle_{d \hookrightarrow nd} \ nd \otimes; \checkmark_{Pairlist} (\lambda r. \langle A_1 \ r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} [s_0, s_1] \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $P_{SKIPS}\text{-combine}_{Pair}\text{-Seq}_{ptick}\text{-behaviour}$ :

$$\langle P_{SKIPS} \langle \langle A_0 \text{ } d \otimes; \checkmark_{Pair} A_1 \rangle \rangle_d (s_0, s_1) = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes; \checkmark_{Pair} (\lambda r. \langle A_1 \text{ } r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (s_0, s_1) \rangle$$

⟨proof⟩

**lemma** *P-combine<sub>Pair</sub>-Seq<sub>ptick</sub>-behaviour:*

$$\langle P \langle \langle A_0 \text{ } d \otimes; \checkmark_{Pair} A_1 \rangle \rangle_d (s_0, s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes; \checkmark_{Pair} (\lambda r. \langle A_1 \text{ } r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (s_0, s_1) \rangle$$

⟨proof⟩

**lemma** *P<sub>SKIPS</sub>-combine<sub>Rlist</sub>-Seq<sub>ptick</sub>-behaviour:*

$$\langle P_{SKIPS} \langle \langle A_0 \text{ } d \otimes; \checkmark_{Rlist} A_1 \rangle \rangle_d (s_0 \# s_1) = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes; \checkmark_{Rlist} (\lambda r. \langle A_1 \text{ } r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (s_0 \# s_1) \rangle$$

⟨proof⟩

**lemma** *P-combine<sub>Rlist</sub>-Seq<sub>ptick</sub>-behaviour:*

$$\langle P \langle \langle A_0 \text{ } d \otimes; \checkmark_{Rlist} A_1 \rangle \rangle_d (s_0 \# s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes; \checkmark_{Rlist} (\lambda r. \langle A_1 \text{ } r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (s_0 \# s_1) \rangle$$

⟨proof⟩

**lemma** *P<sub>SKIPS</sub>-combine<sub>List<sub>slenL</sub></sub>-Seq<sub>ptick</sub>-behaviour:*

$$\langle P_{SKIPS} \langle \langle A_0 \text{ } d \otimes \text{ } len_0; List_{slenL} A_1 \rangle \rangle_d (\sigma s_0 @ \sigma s_1) = P_{SKIPS} \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes \text{ } len_0; List_{slenL} (\lambda r. \langle A_1 \text{ } r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (\sigma s_0 @ \sigma s_1) \rangle$$

if  $\langle \bigwedge \sigma s_0'. \sigma s_0' \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma s_0' = len_0 \rangle$

⟨proof⟩

**lemma** *P-combine<sub>List<sub>slenL</sub></sub>-Seq<sub>ptick</sub>-behaviour:*

$$\langle P \langle \langle A_0 \text{ } d \otimes \text{ } len_0; List_{slenL} A_1 \rangle \rangle_d (\sigma s_0 @ \sigma s_1) = P \langle \langle \langle A_0 \rangle_{d \hookrightarrow nd} \text{ } nd \otimes \text{ } len_0; List_{slenL} (\lambda r. \langle A_1 \text{ } r \rangle_{d \hookrightarrow nd}) \rangle \rangle_{nd} (\sigma s_0 @ \sigma s_1) \rangle$$

if  $\langle \bigwedge \sigma t_0. \sigma t_0 \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma t_0 = len_0 \rangle$

⟨proof⟩

## Chapter 10

# Compactification of Sequential Composition Generalized

### 10.1 Iterated Combine

#### 10.1.1 Definitions

**fun** *iterated-combine<sub>d</sub>-Seq<sub>ptick</sub>* ::  $\langle [ ('r \Rightarrow (' \sigma, 'e, 'r) A_d) \text{ list}, 'r] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_d \rangle$  ( $\langle \langle d \otimes ; \checkmark - \rangle \rangle [0]$ )  
**where**  $\langle \langle d \otimes ; \checkmark [] \rangle \rangle r = \langle \tau = \lambda \sigma s a. \diamond, \omega = \lambda \sigma s. [r] \rangle$   
 $| \quad \langle \langle d \otimes ; \checkmark [A_0] \rangle \rangle r = d \langle \langle A_0 r \rangle \rangle_{\sigma \mapsto \sigma s}$   
 $| \quad \langle \langle d \otimes ; \checkmark A_0 \# A_1 \# A_s \rangle \rangle r = \langle \langle A_0 r \ d \otimes ; \checkmark Rlist \ \langle \langle d \otimes ; \checkmark A_1 \# A_s \rangle \rangle \rangle$

**fun** *iterated-combine<sub>nd</sub>-Seq<sub>ptick</sub>* ::  $\langle [ ('r \Rightarrow (' \sigma, 'e, 'r) A_{nd}) \text{ list}, 'r] \Rightarrow (' \sigma \text{ list}, 'e, 'r) A_{nd} \rangle$  ( $\langle \langle nd \otimes ; \checkmark - \rangle \rangle [0]$ )  
**where**  $\langle \langle nd \otimes ; \checkmark [] \rangle \rangle r = \langle \tau = \lambda \sigma s a. \{\}, \omega = \lambda \sigma s. \{r\} \rangle$   
 $| \quad \langle \langle nd \otimes ; \checkmark [A_0] \rangle \rangle r = nd \langle \langle A_0 r \rangle \rangle_{\sigma \mapsto \sigma s}$   
 $| \quad \langle \langle nd \otimes ; \checkmark A_0 \# A_1 \# A_s \rangle \rangle r = \langle \langle A_0 r \ nd \otimes ; \checkmark Rlist \ \langle \langle nd \otimes ; \checkmark A_1 \# A_s \rangle \rangle \rangle$

**lemma** *iterated-combine<sub>d</sub>-Seq<sub>ptick</sub>-simps-bis*:  $\langle As \neq [] \implies \langle \langle d \otimes ; \checkmark A_0 \# A_s \rangle \rangle r = \langle \langle A_0 r \ d \otimes ; \checkmark Rlist \ \langle \langle d \otimes ; \checkmark As \rangle \rangle \rangle$   
**and** *iterated-combine<sub>nd</sub>-Seq<sub>ptick</sub>-simps-bis*:  $\langle Bs \neq [] \implies \langle \langle nd \otimes ; \checkmark B_0 \# Bs \rangle \rangle r = \langle \langle B_0 r \ nd \otimes ; \checkmark Rlist \ \langle \langle nd \otimes ; \checkmark Bs \rangle \rangle \rangle$   
 $\langle proof \rangle$

#### 10.1.2 First Results

**lemma** *combine<sub>ListslenL</sub>-Seq<sub>ptick</sub>-combine<sub>Rlist</sub>-Seq<sub>ptick</sub>-eq*:  
 $\langle \varepsilon \langle \langle d \langle \langle A_0 \rangle \rangle_{\sigma \mapsto \sigma s} \ d \otimes ; \checkmark 1; ListslenL \ A_1 \rangle \rangle (s_0 \# \sigma s) = \varepsilon \langle \langle A_0 \ d \otimes ; \checkmark Rlist \ A_1 \rangle \rangle (s_0 \# \sigma s) \rangle$   
 $\langle \tau \langle \langle d \langle \langle A_0 \rangle \rangle_{\sigma \mapsto \sigma s} \ d \otimes ; \checkmark 1; ListslenL \ A_1 \rangle \rangle (s_0 \# \sigma s) \ e = \tau \langle \langle A_0 \ d \otimes ; \checkmark Rlist \ A_1 \rangle \rangle (s_0 \# \sigma s) \ e \rangle$   
 $\langle \varepsilon \langle \langle nd \langle \langle B_0 \rangle \rangle_{\sigma \mapsto \sigma s} \ nd \otimes ; \checkmark 1; ListslenL \ B_1 \rangle \rangle (s_0 \# \sigma s) = \varepsilon \langle \langle B_0 \ nd \otimes ; \checkmark Rlist \ B_1 \rangle \rangle (s_0 \# \sigma s) \rangle$

$\sigma s \rangle$   
 $\langle \tau \langle \langle nd \langle B_0 \rangle \sigma \hookrightarrow \sigma s \ nd \otimes 1; ListslenL \ B_1 \rangle (s_0 \# \sigma s) \ e = \tau \langle \langle B_0 \ nd \otimes; \checkmark Rlist \ B_1 \rangle (s_0 \# \sigma s) \ e \rangle$   
 $\langle proof \rangle$

**lemma** *combinePairlist-Seqptick-and-iterated-combine-Seqptick-eq:*

$\langle \varepsilon \langle \langle A_0 \ r \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] = \varepsilon \langle \langle d \otimes; \checkmark [A_0, A_1] \rangle r \rangle [s_0, s_1] \rangle$   
 $\langle \tau \langle \langle A_0 \ r \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] \ e = \tau \langle \langle d \otimes; \checkmark [A_0, A_1] \rangle r \rangle [s_0, s_1] \ e \rangle$   
 $\langle \varepsilon \langle \langle B_0 \ r \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] = \varepsilon \langle \langle nd \otimes; \checkmark [B_0, B_1] \rangle r \rangle [s_0, s_1] \rangle$   
 $\langle \tau \langle \langle B_0 \ r \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] \ e = \tau \langle \langle nd \otimes; \checkmark [B_0, B_1] \rangle r \rangle [s_0, s_1] \ e \rangle$   
 $\langle proof \rangle$

**lemma** *combinePairlist-Seqptick-and-combineRlist-Seqptick-eq :*

$\langle \varepsilon \langle \langle A_0 \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] = \varepsilon \langle \langle A_0 \ d \otimes; \checkmark Rlist \ \langle d \otimes; \checkmark [A_1] \rangle \rangle [s_0, s_1] \rangle$   
 $\langle \tau \langle \langle A_0 \ d \otimes; \checkmark Pairlist \ A_1 \rangle [s_0, s_1] \ e = \tau \langle \langle A_0 \ d \otimes; \checkmark Rlist \ \langle d \otimes; \checkmark [A_1] \rangle \rangle [s_0, s_1] \ e \rangle$   
 $\langle \varepsilon \langle \langle B_0 \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] = \varepsilon \langle \langle B_0 \ nd \otimes; \checkmark Rlist \ \langle nd \otimes; \checkmark [B_1] \rangle \rangle [s_0, s_1] \rangle$   
 $\langle \tau \langle \langle B_0 \ nd \otimes; \checkmark Pairlist \ B_1 \rangle [s_0, s_1] \ e = \tau \langle \langle B_0 \ nd \otimes; \checkmark Rlist \ \langle nd \otimes; \checkmark [B_1] \rangle \rangle [s_0, s_1] \ e \rangle$   
 $\langle proof \rangle$

### 10.1.3 Reachability

**lemma** *same-length- $\tau$ -iterated-combine $_d$ -Seqptick :*

$\langle length \ \sigma s = length \ As \implies [\sigma t] = \tau \langle \langle d \otimes; \checkmark As \rangle r \rangle \ \sigma s \ a \implies length \ \sigma t = length \ As \rangle$   
 $\langle proof \rangle$

**lemma** *same-length- $\tau$ -iterated-combine $_{nd}$ -Seqptick :*

$\langle length \ \sigma s = length \ As \implies \sigma t \in \tau \langle \langle nd \otimes; \checkmark As \rangle r \rangle \ \sigma s \ a \implies length \ \sigma t = length \ As \rangle$   
 $\langle proof \rangle$

**lemma** *same-length- $\mathcal{R}_d$ -iterated-combine $_d$ -Seqptick :*

$\langle \sigma t \in \mathcal{R}_d \langle \langle d \otimes; \checkmark As \rangle r \rangle \ \sigma s \implies length \ \sigma s = length \ As \implies length \ \sigma t = length \ As \rangle$   
 $\langle proof \rangle$

**lemma** *same-length- $\mathcal{R}_{nd}$ -iterated-combine $_{nd}$ -Seqptick :*

$\langle \sigma t \in \mathcal{R}_{nd} \langle \langle nd \otimes; \checkmark As \rangle r \rangle \ \sigma s \implies length \ \sigma s = length \ As \implies length \ \sigma t = length \ As \rangle$   
 $\langle proof \rangle$

### 10.1.4 Transmission of Properties

**lemma**  *$\varrho$ -disjoint- $\varepsilon$ -transmission-to-iterated-combine-Seqptick :*

$\langle As \neq [] \implies \varrho\text{-disjoint-}\varepsilon \langle \langle last \ As \rangle r \rangle \implies \varrho\text{-disjoint-}\varepsilon \langle \langle d \otimes; \checkmark As \rangle r \rangle \rangle$

$\langle Bs \neq [] \implies \varrho\text{-disjoint-}\varepsilon ((\text{last } Bs) r) \implies \varrho\text{-disjoint-}\varepsilon (\llbracket_{nd} \otimes ; \checkmark Bs \rrbracket r) \rangle$   
 ⟨proof⟩

### 10.1.5 Normalization

**lemma**  $\omega$ -iterated-combine-Seq<sub>ptick</sub>-det-ndet-conv:

$\langle \omega (\llbracket_{nd} \otimes ; \checkmark \text{map } (\lambda A r. \llbracket A r \rrbracket_{d \mapsto nd}) As \rrbracket r) \sigma s = \omega \llbracket \llbracket_d \otimes ; \checkmark As \rrbracket r \rrbracket_{d \mapsto nd} \sigma s \rangle$   
 ⟨proof⟩

**lemma**  $\varrho$ -iterated-combine-Seq<sub>ptick</sub>-det-ndet-conv :

$\langle \varrho (\llbracket_{nd} \otimes ; \checkmark \text{map } (\lambda A r. \llbracket A r \rrbracket_{d \mapsto nd}) As \rrbracket r) = \varrho \llbracket \llbracket_d \otimes ; \checkmark As \rrbracket r \rrbracket_{d \mapsto nd} \rangle$   
 ⟨proof⟩

**lemma**  $\tau$ -iterated-combine-Seq<sub>ptick</sub>-behaviour:

$\langle \text{length } \sigma s = \text{length } As \implies$   
 $\tau \llbracket \llbracket_d \otimes ; \checkmark As \rrbracket r \rrbracket_{d \mapsto nd} \sigma s e = \tau (\llbracket_{nd} \otimes ; \checkmark \text{map } (\lambda A r. \llbracket A r \rrbracket_{d \mapsto nd}) As \rrbracket r) \sigma s$   
 $e \rangle$   
 ⟨proof⟩

**lemma**  $P_{SKIPS}$ -iterated-combine-Seq<sub>ptick</sub>-behaviour:

**assumes** same-length:  $\langle \text{length } \sigma s = \text{length } As \rangle$   
**shows**  $\langle P_{SKIPS} \llbracket \llbracket_d \otimes ; \checkmark As \rrbracket r \rrbracket_d \sigma s = P_{SKIPS} \llbracket \llbracket_{nd} \otimes ; \checkmark \text{map } (\lambda A r. \llbracket A r \rrbracket_{d \mapsto nd}) As \rrbracket r \rrbracket_{nd} \sigma s \rangle$   
 ⟨proof⟩

**lemma**  $P$ -iterated-combine-Seq<sub>ptick</sub>-behaviour:

**assumes** same-length:  $\langle \text{length } \sigma s = \text{length } As \rangle$   
**shows**  $\langle P \llbracket \llbracket_d \otimes ; \checkmark As \rrbracket r \rrbracket_d \sigma s = P \llbracket \llbracket_{nd} \otimes ; \checkmark \text{map } (\lambda A r. \llbracket A r \rrbracket_{d \mapsto nd}) As \rrbracket r \rrbracket_{nd} \sigma s \rangle$   
 ⟨proof⟩

## 10.2 Compactification Theorems

### 10.2.1 Binary

#### Pair

**lemma**  $P_{SKIPS}$ -nd-combine<sub>Pair</sub>-Seq<sub>ptick</sub> :

**fixes**  $A_0 A_1$

**assumes** at-most-1-elem-term :  $\langle \text{at-most-1-elem-term } A_0 \rangle$

— This assumption is necessary in the new setup, otherwise the result is not always a Procomaton (for example if  $\omega A_0 \sigma_0 = UNIV$ , we have  $P \sigma_0 ; \checkmark Q \sigma_1 = GlobalNdet UNIV (Q \sigma_1)$ ).

**defines**  $A$ -def:  $\langle A \equiv \llbracket A_0 \text{ nd} \otimes ; \checkmark Pair A_1 \rrbracket \rangle$

**defines**  $P$ -def:  $\langle P \equiv P_{SKIPS} \llbracket A_0 \rrbracket_{nd} \rangle$

**and**  $Q$ -def:  $\langle Q \equiv \lambda \sigma_1 r. P_{SKIPS} \llbracket A_1 r \rrbracket_{nd} \sigma_1 \rangle$

**and**  $S$ -def:  $\langle S \equiv P_{SKIPS}\langle A \rangle_{nd} \rangle$   
**shows**  $\langle P \sigma_0 ; \checkmark Q \sigma_1 = S (\sigma_0, \sigma_1) \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{Pair}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_d \sigma_1) = P_{SKIPS}\langle\langle A_0 d \otimes ; \checkmark_{Pair} A_1 \rangle\rangle_d$   
 $(\sigma_0, \sigma_1) \rangle$   
 $\langle proof \rangle$

## Pairlist

**lemma**  $P_{SKIPS}$ - $nd$ -combine $_{Pairlist}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_{nd} \sigma_1) = P_{SKIPS}\langle\langle A_0 nd \otimes ; \checkmark_{Pairlist} A_1 \rangle\rangle_{nd} [\sigma_0, \sigma_1] \rangle$   
**if**  $\langle at-most-1-elem-term A_0 \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{Pairlist}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_d \sigma_1) = P_{SKIPS}\langle\langle A_0 d \otimes ; \checkmark_{Pairlist} A_1 \rangle\rangle_d [\sigma_0, \sigma_1] \rangle$   
 $\langle proof \rangle$

## Rlist

**lemma**  $P_{SKIPS}$ - $nd$ -combine $_{Rlist}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_{nd} \sigma s) = P_{SKIPS}\langle\langle A_0 nd \otimes ; \checkmark_{Rlist} A_1 \rangle\rangle_{nd} (\sigma_0 \# \sigma s) \rangle$   
**if**  $\langle at-most-1-elem-term A_0 \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{Rlist}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_d \sigma s) = P_{SKIPS}\langle\langle A_0 d \otimes ; \checkmark_{Rlist} A_1 \rangle\rangle_d (\sigma_0 \# \sigma s) \rangle$   
 $\langle proof \rangle$

## 10.2.2 ListslenL

**lemma**  $P_{SKIPS}$ - $nd$ -combine $_{ListslenL}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_{nd} \sigma s_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_{nd} \sigma s_1) = P_{SKIPS}\langle\langle A_0 nd \otimes len_0 ; ListslenL A_1 \rangle\rangle_{nd} (\sigma s_0 @ \sigma s_1) \rangle$   
**if** same-length-reach :  $\langle \bigwedge \sigma s_0'. \sigma s_0' \in \mathcal{R}_{nd} A_0 \sigma s_0 \implies length \sigma s_0' = len_0 \rangle$   
**and**  $\langle at-most-1-elem-term A_0 \rangle$   
 $\langle proof \rangle$

**corollary**  $P_{SKIPS}$ - $d$ -combine $_{ListslenL}$ -Seq $_{ptick}$  :  
 $\langle P_{SKIPS}\langle A_0 \rangle_d \sigma s_0 ; \checkmark (\lambda r. P_{SKIPS}\langle A_1 r \rangle_d \sigma s_1) = P_{SKIPS}\langle\langle A_0 d \otimes len_0 ; ListslenL A_1 \rangle\rangle_d (\sigma s_0 @ \sigma s_1) \rangle$   
**if** same-length-reach :  $\langle \bigwedge \sigma s_0'. \sigma s_0' \in \mathcal{R}_d A_0 \sigma s_0 \implies length \sigma s_0' = len_0 \rangle$   
 $\langle proof \rangle$

### 10.2.3 Multiple

**theorem** *P<sub>SKIP</sub>S-nd-compactification-Seq<sub>ptick</sub>*:

$\langle \llbracket \text{length } \sigma s = \text{length } As; \bigwedge A r. A \in \text{set } (\text{butlast } As) \implies \text{at-most-1-elem-term } (A r) \rrbracket \implies$   
 $SEQ_{\checkmark} (\sigma, A) \in @ \text{ zip } \sigma s As. (\lambda r. P_{SKIP}S \langle \langle A r \rangle \rangle_{nd} \sigma) = (\lambda r. P_{SKIP}S \langle \langle \langle nd \otimes \rangle \rangle_{\checkmark} As \rangle \rangle_{nd} \sigma s) \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *P<sub>SKIP</sub>S-d-compactification-Seq<sub>ptick</sub>*:

$\langle \text{length } \sigma s = \text{length } As \implies$   
 $SEQ_{\checkmark} (\sigma, A) \in @ \text{ zip } \sigma s As. (\lambda r. P_{SKIP}S \langle \langle A r \rangle \rangle_d \sigma) = (\lambda r. P_{SKIP}S \langle \langle \langle d \otimes \rangle \rangle_{\checkmark} As \rangle \rangle_d \sigma s) \rangle$   
 $\langle \text{proof} \rangle$



# Chapter 11

## Application : May Philosophers dine ?

### 11.1 Preliminaries

#### 11.1.1 Preliminary lemmas for proof automation

**lemma** *Suc-mod*:  $\langle n > 1 \implies i \neq \text{Suc } i \text{ mod } n \rangle$   
*<proof>*

**lemmas** *suc-mods* = *Suc-mod Suc-mod*[*symmetric*]

**lemma** *l-suc*:  $\langle n > 1 \implies \neg n \leq \text{Suc } 0 \rangle$   
*<proof>*

**lemma** *minus-suc*:  $\langle n > 0 \implies n - \text{Suc } 0 \neq n \rangle$   
*<proof>*

**declare** *Un-insert-right*[*simp del*] *Un-insert-left*[*simp del*]

### 11.2 The dining processes definition

**context** *DiningPhilosophers* **begin**

**lemma** *RPHIL-restriction-fix-def*:  
 $\langle \text{RPHIL } i = (\nu X. \text{picks } i \ i \rightarrow \text{picks } i \ ((i - 1) \text{ mod } N) \rightarrow$   
 $\text{putsdown } i \ ((i - 1) \text{ mod } N) \rightarrow \text{putsdown } i \ i \rightarrow X) \rangle$   
*<proof>*

**lemma** *LPHIL0-restriction-fix-def*:  
 $\langle \text{LPHIL0} = (\nu X. \text{picks } 0 \ (N - 1) \rightarrow \text{picks } 0 \ 0 \rightarrow$   
 $\text{putsdown } 0 \ 0 \rightarrow \text{putsdown } 0 \ (N - 1) \rightarrow X) \rangle$   
*<proof>*

**lemma** *FORK-restriction-fix-def*:

$\langle \text{FORK } i = (v \ X. (\text{picks } i \ i \rightarrow \text{putsdown } i \ i \rightarrow X) \ \square$   
 $\quad (\text{picks } ((i + 1) \ \text{mod } N) \ i \rightarrow \text{putsdown } ((i + 1) \ \text{mod } N) \ i \rightarrow X)) \rangle$   
 $\langle \text{proof} \rangle$

### 11.2.1 Unfolding rules

**lemmas** *RPHIL-rec* = *cont-process-rec*[*OF RPHIL-def*[*THEN meta-eq-to-obj-eq*],  
*simplified*]  
**and** *LPHIL0-rec* = *cont-process-rec*[*OF LPHIL0-def*[*THEN meta-eq-to-obj-eq*],  
*simplified*]  
**and** *FORK-rec* = *cont-process-rec*[*OF FORK-def*[*THEN meta-eq-to-obj-eq*],  
*simplified*]

## 11.3 Translation into normal form

**lemma** *N-pos[simp]*:  $\langle N > 0 \rangle$   
 $\langle \text{proof} \rangle$

**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<sub>fork</sub>* = *nat*

**type-synonym** *σ<sub>fork</sub>* = *nat*

**type-synonym** *id<sub>phil</sub>* = *nat*

**type-synonym** *σ<sub>phil</sub>* = *nat*

**definition** *fork-A* ::  $\langle \text{id}_{\text{fork}} \Rightarrow (\sigma_{\text{fork}}, \text{dining-event}, \text{unit}) \ A_d \rangle \ (\langle A_f \rangle)$

**where**  $\langle A_f \ i \equiv \text{recursive-constructor-}A_d$   
 $\quad [((0, \text{picks } i \ i), \ [1]), ((0, \text{picks } ((i + 1) \ \text{mod } N) \ i), \ [2]),$   
 $\quad ((1, \text{putsdown } i \ i), [0]), ((2, \text{putsdown } ((i + 1) \ \text{mod } N) \ i), [0])] \rangle$   
 $\langle \lambda \sigma. \diamond \rangle$

**definition** *rphil-A* ::  $\langle \text{id}_{\text{phil}} \Rightarrow (\sigma_{\text{phil}}, \text{dining-event}, \text{unit}) \ A_d \rangle \ (\langle A_{rp} \rangle)$

**where**  $\langle A_{rp} \ i \equiv \text{recursive-constructor-}A_d$   
 $\quad [((0, \text{picks } i \ i), \ [1]), ((1, \text{picks } i \ ((i - 1) \ \text{mod } N)), [2]),$   
 $\quad ((2, \text{putsdown } i \ ((i - 1) \ \text{mod } N)), [3]), ((3, \text{putsdown } i \ i), \ [0])] \rangle$   
 $\langle \lambda \sigma. \diamond \rangle$

**definition** *lphil0-A* ::  $\langle (\sigma_{\text{phil}}, \text{dining-event}, \text{unit}) \ A_d \rangle \ (\langle A_{lp} \rangle)$

**where**  $\langle A_{lp} \equiv \text{recursive-constructor-}A_d$

$$[(0, \text{picks } 0 (N - 1)), [1]], ((1, \text{picks } 0 0), [2]), \\ ((2, \text{putsdown } 0 0), [3]), ((3, \text{putsdown } 0 (N - 1)), [0])] (\lambda\sigma. \diamond)$$

Definition and first properties of associated normal processes

**definition** *fork-P-d* ::  $\langle id_{fork} \Rightarrow \sigma_{fork} \Rightarrow \text{dining-event process} \rangle$  **where**  $\langle \text{fork-P-d}$   
 $i \equiv P\langle A_f i \rangle_d \rangle$

**definition** *rphil-P-d* ::  $\langle id_{rphil} \Rightarrow \sigma_{rphil} \Rightarrow \text{dining-event process} \rangle$  **where**  $\langle \text{rphil-P-d}$   
 $i \equiv P\langle A_{rp} i \rangle_d \rangle$

**definition** *lphil0-P-d* ::  $\langle \sigma_{rphil} \Rightarrow \text{dining-event process} \rangle$  **where**  $\langle \text{lphil0-P-d}$   
 $\equiv P\langle A_{lp} \rangle_d \rangle$

**lemmas** *fork-P-d-rec* = *P-d-rec*[of  $\langle A_f - \rangle$ , *folded fork-P-d-def*]  
**and** *rphil-P-d-rec* = *P-d-rec*[of  $\langle A_{rp} - \rangle$ , *folded rphil-P-d-def*]  
**and** *lphil0-P-d-rec* = *P-d-rec*[of  $\langle A_{lp} \rangle$ , *folded lphil0-P-d-def*]

**schematic-goal** *fork-ε*:  $\langle \varepsilon (A_f i) \sigma = ?S \rangle$   
**and** *rphil-ε*:  $\langle \varepsilon (A_{rp} i) \sigma = ?T \rangle$   
**and** *lphil0-ε*:  $\langle \varepsilon A_{lp} \sigma = ?U \rangle$   
 $\langle \text{proof} \rangle$

**schematic-goal** *fork-τ*:  $\langle \tau (A_f i) \sigma = ?S \rangle$   
**and** *rphil-τ*:  $\langle \tau (A_{rp} i) \sigma = ?T \rangle$   
**and** *lphil0-τ*:  $\langle \tau A_{lp} \sigma = ?U \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *ev-id<sub>fork</sub>x*:  $\langle e \in \varepsilon (A_f i) \sigma \implies \text{fork } e = i \rangle$   
**and** *rphil-phil*:  $\langle e \in \varepsilon (A_{rp} i) \sigma \implies \text{phil } e = i \rangle$   
**and** *lphil0-phil*:  $\langle e \in \varepsilon A_{lp} \sigma \implies \text{phil } e = 0 \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *ev-id<sub>rphil</sub>x*:  $\langle i < n \implies \sigma \in \varepsilon ((A_{lp} \# \text{map } A_{rp} [1..< n]) ! i) s \implies \text{phil}$   
 $\sigma = i \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *indep-forks*:  $\langle i \neq j \implies \varepsilon (A_f i) \sigma \cap \varepsilon (A_f j) \sigma' = \{\} \rangle$   
**and** *indep-phils*:  $\langle i \neq 0 \implies \varepsilon A_{lp} \sigma \cap \varepsilon (A_{rp} i) \sigma' = \{\} \rangle$   
 $\langle i \neq j \implies \varepsilon (A_{rp} i) \sigma \cap \varepsilon (A_{rp} j) \sigma' = \{\} \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma** *FORK-is-fork-P-d*:  $\langle \text{FORK } i = \text{fork-P-d } i 0 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *RPHIL-is-rphil-P-d*:  $\langle RPHIL\ i = rphil\text{-}P\text{-}d\ i\ 0 \rangle$   
 $\langle proof \rangle$

**lemma** *LPHIL0-is-lphil0-P-d*:  $\langle LPHIL0 = lphil0\text{-}P\text{-}d\ 0 \rangle$   
 $\langle proof \rangle$

### 11.3.2 FORKS is normalizable

Definition of the all-forks automaton

**type-synonym**  $\sigma_{forks} = \langle nat\ list \rangle$

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

Definition and first properties of the associated normal process

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

**lemma** *forks-ε*:  $\langle length\ fs = N \implies \varepsilon\ A_F\ fs = (\bigcup_{i < N}. \varepsilon\ (A_f\ i)\ (fs\ !\ i)) \rangle$   
 $\langle proof \rangle$

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

**lemma** *NFORKS-is-forks-P-d*:  $\langle FORKS = forks\text{-}P\text{-}d\ (replicate\ N\ 0) \rangle$   
 $\langle proof \rangle$

### 11.3.3 PHILS is normalizable

Definition of the all-philosophers automaton

**type-synonym**  $\sigma_{phils} = \langle nat\ list \rangle$

**definition** *phils-A* ::  $\langle (\sigma_{phils}, dining\text{-}event, unit)\ A_d \rangle (\langle A_P \rangle)$  **where**  $\langle A_P \equiv \langle \langle_d \otimes [\{\}] \rangle A_{lp}\ \# \ map\ A_{rp}\ [1..<N] \rangle \rangle$

**lemma** *phils-A-def-bis*:  $\langle A_P = \langle \langle_d \otimes [\{\}] \rangle map\ (\lambda i. \text{if } i = 0 \text{ then } A_{lp} \text{ else } A_{rp}\ i)\ [0..<N] \rangle$   
 $\langle proof \rangle$

Definition and first properties of the associated normal process

**definition** *phils-P-d*::  $\langle \sigma_{phils} \Rightarrow dining\text{-}event\ process \rangle$  **where**  $\langle phils\text{-}P\text{-}d \equiv P \langle \langle A_P \rangle \rangle_d \rangle$

**lemma** *phils-ε*:  $\langle length\ ps = N \implies \varepsilon\ A_P\ ps = \varepsilon\ A_{lp}\ (ps\ !\ 0) \cup (\bigcup_{i \in \{1..<N\}}. \varepsilon\ (A_{rp}\ i)\ (ps\ !\ i)) \rangle$

*<proof>*

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

**lemma** *NPHILS-is-phils-P-d*:  $\langle PHILS = phils-P-d \text{ (replicate } N \ 0) \rangle$   
*<proof>*

### 11.3.4 The complete process *DINING* is normalizable

Definition of the dining automaton

**definition** *dining-A* ::  $\langle (\sigma_{phils} \times \sigma_{forks}, \text{dining-event}, \text{unit}) \ A_d \rangle \ (\langle A_D \rangle)$  **where**  
 $\langle A_D \equiv \langle \langle A_P \ d \otimes \llbracket UNIV \rrbracket_{Pair} \ A_F \rangle \rangle \rangle$

Definition and first properties of the associated normal process

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

**lemma** *dining-ε*:

$\langle \text{length } ps = N \implies \text{length } fs = N \implies$   
 $\varepsilon \ A_D \ (ps, fs) = (\varepsilon \ A_{I_p} \ (ps \ ! \ 0) \cup (\bigcup_{i \in \{1..<N\}}. \varepsilon \ (A_{r_p} \ i) \ (ps \ ! \ i))) \cap (\bigcup_{i < N.}$   
 $\varepsilon \ (A_f \ i) \ (fs \ ! \ i)) \rangle$   
*<proof>*

Equality between *DINING* and *dining-P-d*

**lemma** *DINING-is-dining-P-d*:  $\langle DINING = dining-P-d \text{ (replicate } N \ 0, \text{ replicate } N \ 0) \rangle$   
*<proof>*

## 11.4 And finally: Philosophers may dine ! Always !

**method** *ε-sets-simp uses opt = (simp-all split: if-split-asm)?,*  
*simp-all add: fork-ε lphil0-ε rphil-ε opt split: if-splits*

**method** *A-defs-simp uses opt = (simp-all split: if-split-asm)?,*  
*simp-all add: fork-A-def lphil0-A-def rphil-A-def opt split: if-splits*

### 11.4.1 Construction of an invariant for the dining automaton

**definition**  $\langle \text{inv-dining } ps \ fs \equiv$   
 $\text{length } fs = N \wedge \text{length } ps = N$   
 $\wedge (\forall i < N. fs \ ! \ i = 0 \vee fs \ ! \ i = 1 \vee fs \ ! \ i = 2)$   
 $\wedge (\forall i < N. ps \ ! \ i = 0 \vee ps \ ! \ i = 1 \vee ps \ ! \ i = 2 \vee ps \ ! \ i = 3)$   
 $\wedge (\forall i. \text{Suc } i < N \longrightarrow ((fs \ ! \ \text{Suc } i = 1) \longleftrightarrow ps \ ! \ \text{Suc } i \neq 0)) \wedge (fs \ ! \ (N$   
 $- 1) = 2 \longleftrightarrow ps \ ! \ 0 \neq 0)$   
 $\wedge (\forall i < N - 1. fs \ ! \ i = 2 \longleftrightarrow ps \ ! \ \text{Suc } i = 2) \wedge (fs \ ! \ 0 =$   
 $1 \longleftrightarrow ps \ ! \ 0 = 2) \rangle$

**lemma** *show-inv-dining*:

$\langle \text{length } fs = N \wedge \text{length } ps = N \implies$   
 $(\forall i < N. fs ! i = 0 \vee fs ! i = 1 \vee fs ! i = 2) \implies$   
 $(\forall i < N. ps ! i = 0 \vee ps ! i = 1 \vee ps ! i = 2 \vee ps ! i = 3) \implies$   
 $(\forall i. \text{Suc } i < N \longrightarrow (fs ! \text{Suc } i = 1 \longleftrightarrow ps ! \text{Suc } i \neq 0)) \implies (fs ! (N - 1) =$   
 $2 \longleftrightarrow ps ! 0 \neq 0) \implies$   
 $(\forall i < N - 1. fs ! i = 2 \longleftrightarrow ps ! \text{Suc } i = 2) \implies (fs ! 0 = 1 \longleftrightarrow ps ! 0 = 2)$   
 $\implies$   
*inv-dining ps fs*  
*proof*

**lemma** *inv-DINING*:  $\langle s \in \mathcal{R}_d A_D (\text{replicate } N \ 0, \text{replicate } N \ 0) \implies \text{inv-dining}$   
*(fst s) (snd s)*  
*proof*

### 11.4.2 The invariant *inv-dining* implies that *DINING* is *deadlock-free*

**method** *nonempty-Int-by-common-element* **for**  $x = \text{rule-tac } ex\text{-in-conv}[ \text{THEN } iffD1,$   
 $OF \ exI, OF \ IntI, \text{ of } x]$

**lemma** *inv-implies-DF*:  $\langle \varepsilon A_D (ps, fs) \neq \{\} \rangle$  **if** *hyp-inv*: *inv-dining ps fs*  
*proof*

### 11.4.3 Conclusion

**corollary** *deadlock-free-DINING*: *deadlock-free DINING*  
*proof*

## 11.5 Alternative version with only right-handed philosophers (in order to show that it's not *deadlock-free*)

### 11.5.1 Setup

**definition**  $\langle RPHILS \equiv ||| P \in \# \text{ mset } (\text{map } RPHIL [0..< N]). P \rangle$

**corollary**  $\langle N = 3 \implies RPHILS = (RPHIL \ 0 ||| RPHIL \ 1 ||| RPHIL \ 2) \rangle$   
*proof*

**definition** *RDINING* :: *dining-event process*  
**where**  $\langle RDINING = (\text{FORKS} || RPHILS) \rangle$

## 11.5.2 Normalization

**definition**  $rphils-A$  ::  $\langle (\sigma_{phils}, dining-event, unit) A_d \rangle (\langle A_{RP} \rangle)$  **where**  $\langle A_{RP} \equiv \langle \langle_d \otimes [\{\}] \rangle map A_{rp} [0..< N] \rangle \rangle$

**definition**  $rphils-P-d$ ::  $\langle \sigma_{phils} \Rightarrow dining-event process \rangle$  **where**  $\langle rphils-P-d \equiv P \langle \langle A_{RP} \rangle \rangle_d \rangle$

**definition**  $rdining-A$  ::  $\langle (\sigma_{phils} \times \sigma_{forks}, dining-event, unit) A_d \rangle (\langle A_{RD} \rangle)$  **where**  $\langle A_{RD} \equiv \langle \langle A_{RP} \rangle \otimes [UNIV]_{Pair} A_F \rangle \rangle$

**definition**  $rdining-P-d$ ::  $\langle \sigma_{phils} \times \sigma_{forks} \Rightarrow dining-event process \rangle$  **where**  $\langle rdining-P-d \equiv P \langle \langle A_{RD} \rangle \rangle_d \rangle$

## 11.5.3 Correspondance between our normalized processes and the previous definitions

**lemma**  $rphils-\varepsilon$ :  $\langle length ps = N \implies \varepsilon A_{RP} ps = (\bigcup i \in \{0..< N\}. \varepsilon (A_{rp} i) (ps ! i)) \rangle$   
 $\langle proof \rangle$

**lemma**  $NRPHILS-is-rphils-P-d$ :  $\langle RPHILS = rphils-P-d (replicate N 0) \rangle$   
 $\langle proof \rangle$

**lemma**  $rdining-\varepsilon$ :  
 $\langle length ps = N \implies length fs = N \implies \varepsilon A_{RD} (ps, fs) = (\bigcup i \in \{0..< N\}. \varepsilon (A_{rp} i) (ps ! i)) \cap (\bigcup i < N. \varepsilon (A_f i) (fs ! i)) \rangle$   
 $\langle proof \rangle$

**lemma**  $RDINING-is-rdining-P-d$ :  $\langle RDINING = rdining-P-d (replicate N 0, replicate N 0) \rangle$   
 $\langle proof \rangle$

## 11.5.4 Proof that we have a deadlock in the state (replicate N 1, replicate N 1)

**lemma**  $empty-enabl-replicate1$ :  $\langle \varepsilon A_{RD} (replicate N 1, replicate N 1) = \{\} \rangle$   
 $\langle proof \rangle$

**corollary**  $non-deadlock-free-rdining$ :  $\langle \neg deadlock-free (rdining-P-d (replicate N 1, replicate N 1)) \rangle$   
 $\langle proof \rangle$

## 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**  $rdining-\tau$ :  $\langle length ps = N \implies length fs = N \implies e \in \varepsilon A_{RD} (ps, fs) \implies \tau A_{RD} (ps, fs) e = [(\tau A_{RP} ps e), (\tau A_F fs e)] \rangle$   
 $\langle proof \rangle$

**lemma** *replicate1-reachable-from-replicate0-prelim:*

$\langle n \leq N \implies (\text{replicate } n \ 1 \ @ \ \text{replicate } (N - n) \ 0, \text{ replicate } n \ 1 \ @ \ \text{replicate } (N - n) \ 0) \in \mathcal{R}_d \ A_{RD} \ (\text{replicate } N \ 0, \text{ replicate } N \ 0) \rangle$   
*\langle proof \rangle*

**corollary** *replicate1-reachable-from-replicate0:*  $\langle (\text{replicate } N \ 1, \text{ replicate } N \ 1) \in \mathcal{R}_d \ A_{RD} \ (\text{replicate } N \ 0, \text{ replicate } N \ 0) \rangle$   
*\langle proof \rangle*

**theorem** *not-deadlock-free-RDINING:*  $\langle \neg \text{deadlock-free } RDINING \rangle$   
*\langle proof \rangle*

**end**

## Chapter 12

# Other Results similar to Compactification

Unlike *Sync* and  $(;)$ , some operators like *Det* do not enjoy a compactification result. Nevertheless, we still can prove some useful lemmas.

### 12.1 Some preliminary Results

**lemma** *Mprefix-Det-Mprefix-bis* :

$\langle (\Box a \in A \rightarrow P a) \Box (\Box b \in B \rightarrow Q b) =$   
 $(\Box x \in (A \cap B) \rightarrow P x \Box Q x) \Box (\Box a \in (A - B) \rightarrow P a) \Box (\Box b \in (B - A) \rightarrow$   
 $Q b) \rangle$   
(is  $\langle ?lhs = ?rhs \rangle$ )  
 $\langle proof \rangle$

**lemma** *GlobalNdet-Ndet-GlobalNdet*:

$\langle A \neq \{\} \implies B \neq \{\} \implies (\Box a \in A. P a) \Box (\Box b \in B. Q b) =$   
 $\Box x \in (A \cup B). (if x \in A \cap B then P x \Box Q x else if x \in A then P x else Q x) \rangle$   
 $\langle proof \rangle$

**lemma** *GlobalNdet-Ndet-GlobalNdet-bis*:

$\langle A \cap B \neq \{\} \implies A - B \neq \{\} \implies B - A \neq \{\} \implies$   
 $(\Box a \in A. P a) \Box (\Box b \in B. Q b) =$   
 $(\Box x \in (A \cap B). P x \Box Q x) \Box (\Box a \in (A - B). P a) \Box (\Box b \in (B - A). Q b) \rangle$   
 $\langle proof \rangle$

**lemma** *GlobalNdet-GlobalNdet*:

$\langle (\Box a \in A. \Box b \in B a. P b) =$   
 $(if \forall a \in A. B a \neq \{\} then \Box b \in (\bigcup a \in A. B a). P b else (\Box b \in (\bigcup a \in A. B$   
 $a). P b) \Box STOP) \rangle$   
 $\langle proof \rangle$

## 12.2 Results for *Det*

**lemma** *P-nd-set-almost-compactification-Det* :

$$\langle \Box (s, A) \in s\text{-}A\text{-set}. P\langle A \rangle_{nd} s) =$$

$$\Box e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$$

$$\Box (s, A) \in \{(s, A) \in s\text{-}A\text{-set}. e \in \varepsilon A s\}.$$

$$\Box s' \in \tau A s e. P\langle A \rangle_{nd} s' \rangle \text{ (is } \langle ?lhs = ?rhs \rangle)$$

*<proof>*

**lemma** *P-nd-set-almost-compactification-Det-bis* :

$$\langle \Box (s, A) \in s\text{-}A\text{-set}. P\langle A \rangle_{nd} s) =$$

$$\Box e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$$

$$\Box (s', A) \in \{(s', A) \mid s' s A. (s, A) \in s\text{-}A\text{-set} \wedge e \in \varepsilon A s \wedge s' \in \tau A s e\}. P\langle A \rangle_{nd}$$

$$s'\rangle$$

$$\text{ (is } \langle - = ?rhs \rangle)$$

*<proof>*

**lemma** *P-d-set-almost-compactification-Det*:

**shows**  $\langle \Box (s, A) \in s\text{-}A\text{-set}. P\langle A \rangle_d s) =$

$$\Box e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$$

$$\Box (s, A) \in \{(s, A) \in s\text{-}A\text{-set}. e \in \varepsilon A s\}. P\langle A \rangle_d [\tau A s e] \rangle \text{ (is } \langle ?lhs =$$

$$?rhs \rangle)$$

*<proof>*

**lemma** *P-d-set-almost-compactification-Det-bis*:

**shows**  $\langle \Box (s, A) \in s\text{-}A\text{-set}. P\langle A \rangle_d s) =$

$$\Box e \in (\bigcup (s, A) \in s\text{-}A\text{-set}. \varepsilon A s) \rightarrow$$

$$\Box (s', A) \in \{([\tau A s e], A) \mid s A. (s, A) \in s\text{-}A\text{-set} \wedge e \in \varepsilon A s\}. P\langle A \rangle_d s'\rangle$$

*<proof>*

## 12.3 Results for *Ndet*

## 12.4 Other Operators

### 12.4.1 *initials*

**lemma** *initials-P<sub>SKIP</sub>S-nd* :

$$\langle (P_{SKIP}S\langle A \rangle_{nd} \sigma)^0 = (\text{if } \sigma \in \varrho A \text{ then tick } \omega A \sigma \text{ else ev } \varepsilon A \sigma) \rangle$$

*<proof>*

**lemma** *initials-P<sub>SKIP</sub>S-d* :

$\langle (P_{SKIPS}\langle A \rangle_d \sigma)^0 = (\text{if } \sigma \in \varrho A \text{ then } \{\checkmark([\omega A \sigma])\} \text{ else } \text{ev } \epsilon A \sigma) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *initials-P-nd* :  $\langle (P\langle A \rangle_{nd} s)^0 = \text{ev } \epsilon A s \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *initials-P-d* :  $\langle (P\langle A \rangle_d s)^0 = \text{ev } \epsilon A s \rangle$   
 $\langle \text{proof} \rangle$

### 12.4.2 Throw

**lemma** *Throw-P<sub>SKIPS</sub>-nd* :  
 $\langle P_{SKIPS}\langle A \rangle_{nd} \sigma \Theta b \in B. Q b =$   
 $(\text{if } \sigma \in \varrho A \text{ then } SKIPS (\omega A \sigma) \text{ else}$   
 $\square a \in \varepsilon A \sigma \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } \sqcap \sigma' \in \tau A \sigma a. (P_{SKIPS}\langle A \rangle_{nd} \sigma' \Theta b$   
 $\in B. Q b))) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Throw-P<sub>SKIPS</sub>-d* :  
 $\langle P_{SKIPS}\langle A \rangle_d \sigma \Theta b \in B. Q b =$   
 $(\text{if } \sigma \in \varrho A \text{ then } SKIP [\omega A \sigma] \text{ else}$   
 $\square a \in \varepsilon A \sigma \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } P_{SKIPS}\langle A \rangle_d [\tau A \sigma a] \Theta b \in B. Q b)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Throw-P-nd* :  
 $\langle P\langle A \rangle_{nd} \sigma \Theta b \in B. Q b =$   
 $\square a \in \varepsilon A \sigma \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } \sqcap \sigma' \in \tau A \sigma a. (P\langle A \rangle_{nd} \sigma' \Theta b \in B. Q$   
 $b)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Throw-P-d* :  
 $\langle P\langle A \rangle_d \sigma \Theta b \in B. Q b =$   
 $\square a \in \varepsilon A \sigma \rightarrow (\text{if } a \in B \text{ then } Q a \text{ else } P\langle A \rangle_d [\tau A \sigma a] \Theta b \in B. Q b) \rangle$   
 $\langle \text{proof} \rangle$

### 12.4.3 ( $\Delta$ )

**lemma** *SKIPS-Interrupt-is-SKIPS-Det* :  
 $\langle SKIPS R \Delta P = SKIPS R \square P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Interrupt-P<sub>SKIPS</sub>-nd* :  
 $\langle P_{SKIPS}\langle A \rangle_{nd} \sigma \Delta Q =$   
 $Q \square (\text{if } \sigma \in \varrho A \text{ then } SKIPS (\omega A \sigma) \text{ else } \square a \in \varepsilon A \sigma \rightarrow \sqcap \sigma' \in \tau A \sigma a.$   
 $P_{SKIPS}\langle A \rangle_{nd} \sigma' \Delta Q) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Interrupt-P<sub>SKIPS</sub>-d* :  
 $\langle P_{SKIPS}\langle A \rangle_d \sigma \Delta Q =$

$Q \sqcap (\text{if } \sigma \in \varrho A \text{ then } \text{SKIP } [\omega A \sigma] \text{ else } \sqcap a \in \varepsilon A \sigma \rightarrow P_{\text{SKIPS}}\langle\langle A \rangle\rangle_d [\tau A \sigma a] \Delta Q) \rangle$   
 ⟨proof⟩

**lemma** *Interrupt-P-nd* :

$\langle P\langle\langle A \rangle\rangle_{nd} \sigma \Delta Q = Q \sqcap (\sqcap a \in \varepsilon A \sigma \rightarrow \sqcap \sigma' \in \tau A \sigma a. P\langle\langle A \rangle\rangle_{nd} \sigma' \Delta Q) \rangle$   
 ⟨proof⟩

**lemma** *Interrupt-P-d* :

$\langle P\langle\langle A \rangle\rangle_d \sigma \Delta Q = Q \sqcap (\sqcap a \in \varepsilon A \sigma \rightarrow P\langle\langle A \rangle\rangle_d [\tau A \sigma a] \Delta Q) \rangle$   
 ⟨proof⟩

#### 12.4.4 After

**context** *After*

**begin**

**lemma** *After-SKIPS* :  $\langle \text{SKIPS } R \text{ after } a = \Psi (\text{SKIPS } R) a \rangle$

⟨proof⟩

**lemma** *After-P<sub>SKIPS</sub>-nd* :

$\langle P_{\text{SKIPS}}\langle\langle A \rangle\rangle_{nd} \sigma \text{ after } a =$   
 (if  $\sigma \in \varrho A$  then  $\Psi (\text{SKIPS } (\omega A \sigma)) a$  else  
 if  $a \in \varepsilon A \sigma$  then  $\sqcap \sigma' \in \tau A \sigma a. P_{\text{SKIPS}}\langle\langle A \rangle\rangle_{nd} \sigma'$  else  $\Psi (P_{\text{SKIPS}}\langle\langle A \rangle\rangle_{nd} \sigma)$   
 $a) \rangle$   
 ⟨proof⟩

**lemma** *After-P<sub>SKIPS</sub>-d* :

$\langle P_{\text{SKIPS}}\langle\langle A \rangle\rangle_d \sigma \text{ after } a =$   
 (if  $\sigma \in \varrho A$  then  $\Psi (\text{SKIP } [\omega A \sigma]) a$  else  
 if  $a \in \varepsilon A \sigma$  then  $P_{\text{SKIPS}}\langle\langle A \rangle\rangle_d [\tau A \sigma a]$  else  $\Psi (P_{\text{SKIPS}}\langle\langle A \rangle\rangle_d \sigma) a) \rangle$   
 ⟨proof⟩

**lemma** *After-P-nd* :

$\langle P\langle\langle A \rangle\rangle_{nd} \sigma \text{ after } a = (\text{if } a \in \varepsilon A \sigma \text{ then } \sqcap \sigma' \in \tau A \sigma a. P\langle\langle A \rangle\rangle_{nd} \sigma' \text{ else } \Psi$   
 $(P\langle\langle A \rangle\rangle_{nd} \sigma) a) \rangle$   
 ⟨proof⟩

**lemma** *After-P-d* :

$\langle P\langle\langle A \rangle\rangle_d \sigma \text{ after } a = (\text{if } a \in \varepsilon A \sigma \text{ then } P\langle\langle A \rangle\rangle_d [\tau A \sigma a] \text{ else } \Psi (P\langle\langle A \rangle\rangle_d \sigma) a) \rangle$   
 ⟨proof⟩

**end**

**context** *AfterExt*

**begin**

**lemma** *After<sub>tick</sub>-SKIPS* :

$\langle \text{SKIPS } R \text{ after } \checkmark e = (\text{case } e \text{ of } ev \ a \Rightarrow \Psi (\text{SKIPS } R) \ a \mid \checkmark(r) \Rightarrow \Omega (\text{SKIPS } R) \ r) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *After<sub>tick</sub>-P<sub>SKIPS</sub>-nd :*

$\langle P_{\text{SKIPS}} \langle A \rangle_{nd} \ \sigma \text{ after } \checkmark e =$   
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } \sigma \in \varrho \ A \ \text{then } \Psi (\text{SKIPS } (\omega \ A \ \sigma)) \ a \ \text{else}$   
 $\text{if } a \in \varepsilon \ A \ \sigma \ \text{then } \sqcap \sigma' \in \tau \ A \ \sigma \ a. \ P_{\text{SKIPS}} \langle A \rangle_{nd} \ \sigma' \ \text{else } \Psi$   
 $(P_{\text{SKIPS}} \langle A \rangle_{nd} \ \sigma) \ a$   
 $\mid \checkmark(r) \Rightarrow \text{if } \sigma \in \varrho \ A \ \text{then } \Omega (\text{SKIPS } (\omega \ A \ \sigma)) \ r \ \text{else } \Omega (P_{\text{SKIPS}} \langle A \rangle_{nd}$   
 $\sigma) \ r) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *After<sub>tick</sub>-P<sub>SKIPS</sub>-d :*

$\langle P_{\text{SKIPS}} \langle A \rangle_d \ \sigma \text{ after } \checkmark e =$   
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } \sigma \in \varrho \ A \ \text{then } \Psi (\text{SKIP } [\omega \ A \ \sigma]) \ a \ \text{else}$   
 $\text{if } a \in \varepsilon \ A \ \sigma \ \text{then } P_{\text{SKIPS}} \langle A \rangle_d \ [\tau \ A \ \sigma \ a] \ \text{else } \Psi (P_{\text{SKIPS}} \langle A \rangle_d$   
 $\sigma) \ a$   
 $\mid \checkmark(r) \Rightarrow \text{if } \sigma \in \varrho \ A \ \text{then } \Omega (\text{SKIP } [\omega \ A \ \sigma]) \ r \ \text{else } \Omega (P_{\text{SKIPS}} \langle A \rangle_d \ \sigma)$   
 $r) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *After<sub>tick</sub>-P-nd :*

$\langle P \langle A \rangle_{nd} \ \sigma \text{ after } \checkmark e =$   
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } a \in \varepsilon \ A \ \sigma \ \text{then } \sqcap \sigma' \in \tau \ A \ \sigma \ a. \ P \langle A \rangle_{nd} \ \sigma' \ \text{else } \Psi (P \langle A \rangle_{nd}$   
 $\sigma) \ a$   
 $\mid \checkmark(r) \Rightarrow \Omega (P \langle A \rangle_{nd} \ \sigma) \ r) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *After<sub>tick</sub>-P-d :*

$\langle P \langle A \rangle_d \ \sigma \text{ after } \checkmark e =$   
 $(\text{case } e \text{ of } ev \ a \Rightarrow \text{if } a \in \varepsilon \ A \ \sigma \ \text{then } P \langle A \rangle_d \ [\tau \ A \ \sigma \ a] \ \text{else } \Psi (P \langle A \rangle_d \ \sigma) \ a$   
 $\mid \checkmark(r) \Rightarrow \Omega (P \langle A \rangle_d \ \sigma) \ r) \rangle$   
 $\langle \text{proof} \rangle$

end

## 12.5 OpSem

**context** *OpSemTransitions*

**begin**

**lemma** *SKIPS- $\tau$ -trans-SKIP :*  $\langle r \in R \Rightarrow \text{SKIPS } R \rightsquigarrow_{\tau} \text{SKIP } r \rangle$

$\langle \text{proof} \rangle$

In the ProcOmata, we will absorb the  $\tau$  transitions that appear when we unfold the fixed-point operator.

**lemma**  $\tau$ -trans- $P_{SKIP}$ -nd :

$$\langle r \in \omega A \sigma \implies P_{SKIP}\langle\langle A \rangle\rangle_{nd} \sigma \rightsquigarrow_{\tau} SKIP r \rangle$$

$\langle proof \rangle$

**lemma**  $\tau$ -trans- $P_{SKIP}$ -d :

$$\langle \sigma \in \varrho A \implies P_{SKIP}\langle\langle A \rangle\rangle_d \sigma \rightsquigarrow_{\checkmark} [\omega A \sigma] \Omega (SKIP [\omega A \sigma]) [\omega A \sigma] \rangle$$

$\langle proof \rangle$

**lemma**  $ev$ -trans- $P_{SKIP}$ -nd :

$$\langle \sigma \notin \varrho A \implies \sigma' \in \tau A \sigma a \implies P_{SKIP}\langle\langle A \rangle\rangle_{nd} \sigma \rightsquigarrow_a P_{SKIP}\langle\langle A \rangle\rangle_{nd} \sigma' \rangle$$

$\langle proof \rangle$

**lemma**  $ev$ -trans- $P_{SKIP}$ -d :

$$\langle \sigma \notin \varrho A \implies a \in \varepsilon A \sigma \implies P_{SKIP}\langle\langle A \rangle\rangle_d \sigma \rightsquigarrow_a P_{SKIP}\langle\langle A \rangle\rangle_d [\tau A \sigma a] \rangle$$

$\langle proof \rangle$

**lemma**  $ev$ -trans- $P$ -nd :

$$\langle \sigma' \in \tau A \sigma a \implies P\langle\langle A \rangle\rangle_{nd} \sigma \rightsquigarrow_a P\langle\langle A \rangle\rangle_{nd} \sigma' \rangle$$

$\langle proof \rangle$

**lemma**  $ev$ -trans- $P$ -d :

$$\langle a \in \varepsilon A \sigma \implies P\langle\langle A \rangle\rangle_d \sigma \rightsquigarrow_a P\langle\langle A \rangle\rangle_d [\tau A \sigma a] \rangle$$

$\langle proof \rangle$

**end**

# Chapter 13

## Conclusion

### 13.1 Entry Point

This is where `HOL-CSP_Proc-Omata` should be imported from.

### 13.2 Conclusion

In this entry we have developed the Proc-Omata framework on top of `HOL-CSP` and its extensions. Starting from functional automata, we introduced Proc-Omata in four variants: deterministic, terminating deterministic, non-deterministic, and terminating non-deterministic. They enjoy strong structural properties, for example deadlocks can be characterized directly and established by invariant reasoning:

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

We then lifted sequential composition and synchronization product to the automata level, by defining suitable combination functions and proving their correctness. A major generalization of our development is the treatment of parameterized termination. For sequential composition we worked directly with the generalized operator  $(;_{\checkmark})$ , since the standard one  $(;)$  is easily recovered (indeed  $P ;_{\checkmark} (\lambda r. Q) = P ; Q$ ). In contrast, for synchronization product we had to provide two distinct versions, as the handling of ticks prevents any straightforward reduction from  $P \llbracket A \rrbracket Q$  to  $P \llbracket A \rrbracket_{\checkmark} Q$ .

Another central ingredient is the library `Restriction_Spaces` [1]. Proc-Omata are indeed defined as fixed points of endofunctions which, in the non-deterministic case, are not always continuous due to global non-deterministic

choice. While deterministic prefix choice does not suffice to restore continuity under composition, it does guarantee constructiveness, allowing us to rely on the fixed-point operator  $\nu x. f x$  in all cases.

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

$$\frac{|\sigma s| = |As|}{\llbracket E \rrbracket (\sigma, A) \in \#mset (zip \sigma s As). P \langle\langle A \rangle\rangle_{nd} \sigma = P \langle\langle \langle_{nd} \otimes \llbracket E \rrbracket As \rangle\rangle_{nd} \sigma s}$$

Finally, we demonstrated the applicability of our approach with the Dining Philosophers case study, where Proc-Omata compactification enables proofs that scale to an arbitrary number of participants in this parameterized process architecture.

# Bibliography

- [1] B. Ballenghien, B. Puyobro, and B. Wolff. Restriction spaces: a fixed-point theory. *Archive of Formal Proofs*, May 2025. [https://isa-afp.org/entries/Restriction\\_Spaces.html](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](https://isa-afp.org/entries/HOL-CSP_OpSem.html), Formal proof development.
- [5] B. Ballenghien and B. Wolff. A theory of proc-omataand proof methods for process architectures. In *Theoretical Aspects of Computing IC-TAC 2024: 21st International Colloquium, Bangkok, Thailand, November 2529, 2024, Proceedings*, page 272289, Berlin, Heidelberg, 2024. Springer-Verlag.
- [6] B. Ballenghien and B. Wolff. Csp semantics over restriction spaces. *Archive of Formal Proofs*, May 2025. [https://isa-afp.org/entries/HOL-CSP\\_RS.html](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.