

# HOL-CSPM - Architectural operators for HOL-CSP

Benoît Ballenghien      Safouan Taha      Burkhart Wolff

May 26, 2024



# Abstract

Recently, a modern version of Roscoes and Brookes [1] Failure-Divergence Semantics for CSP has been formalized in Isabelle [3].

The session HOL-CSP introduces among other things some binary operators on processes that we will here generalize in a fully-abstract way.

On these "architectural operators", we will prove the properties of refinement, the rules of continuity and the laws of interaction so that they can be easily used.

Finally, we will give examples of their usefulness when trying to model complex systems.



# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Motivations . . . . .	9
1.2	The Global Architecture of HOL-CSPM . . . . .	11
<b>2</b>	<b>Some Preliminary Work</b>	<b>13</b>
2.1	Induction Rules for ' $\alpha$ set' . . . . .	13
2.2	Induction Rules for ' $\alpha$ multiset' . . . . .	14
2.3	Strong Induction for <i>nat</i> . . . . .	15
2.4	Preliminaries for Cartesian Product Results . . . . .	15
<b>3</b>	<b>Patch for Compatibility</b>	<b>17</b>
3.1	Results . . . . .	17
3.1.1	Continuity Rule . . . . .	17
3.1.2	Nice Properties . . . . .	18
3.1.3	Renaming Laws . . . . .	18
3.2	Assertions . . . . .	19
3.3	Liflock Freeness . . . . .	19
3.4	New Laws . . . . .	19
<b>4</b>	<b>The MultiDet Operator</b>	<b>21</b>
4.1	Definition . . . . .	21
4.2	First Properties . . . . .	21
4.3	Some Tests . . . . .	22
4.4	Continuity . . . . .	22
4.5	Factorization of $(\square)$ in front of <i>MultiDet</i> . . . . .	22
4.6	$\perp$ Absorbance . . . . .	22
4.7	First Properties . . . . .	22
4.8	Behaviour of <i>MultiDet</i> with $(\square)$ . . . . .	23
4.9	Commutativity . . . . .	23
4.10	Behaviour with Injectivity . . . . .	23
4.11	The Projections . . . . .	23
4.12	Cartesian Product Results . . . . .	23

<b>5</b>	<b>The MultiNdet Operator</b>	<b>25</b>
5.1	Definition . . . . .	25
5.2	First Properties . . . . .	26
5.3	Some Tests . . . . .	26
5.4	Continuity . . . . .	27
5.5	Factorization of $(\sqcap)$ in front of <i>MultiNdet</i> . . . . .	27
5.6	$\perp$ Absorbance . . . . .	27
5.7	First Properties . . . . .	27
5.8	Behaviour of <i>MultiNdet</i> with $(\sqcap)$ . . . . .	28
5.9	Commutativity . . . . .	28
5.10	Behaviour with Injectivity . . . . .	28
5.11	The Projections . . . . .	28
5.12	Cartesian Product Results . . . . .	28
<b>6</b>	<b>The MultiSync Operator</b>	<b>31</b>
6.1	Definition . . . . .	31
6.2	First Properties . . . . .	32
6.3	Some Tests . . . . .	33
6.4	Continuity . . . . .	34
6.5	Factorization of <i>Sync</i> in front of <i>MultiSync</i> . . . . .	34
6.6	$\perp$ Absorbance . . . . .	34
6.7	Other Properties . . . . .	35
6.8	Behaviour of <i>MultiSync</i> with <i>Sync</i> . . . . .	35
6.9	Commutativity . . . . .	35
6.10	Behaviour with Injectivity . . . . .	35
<b>7</b>	<b>The MultiSeq Operator</b>	<b>37</b>
7.1	Definition . . . . .	37
7.2	First Properties . . . . .	37
7.3	Some Tests . . . . .	37
7.4	Continuity . . . . .	38
7.5	Factorization of $(;)$ in front of <i>MultiSeq</i> . . . . .	38
7.6	$\perp$ Absorbance . . . . .	38
7.7	First Properties . . . . .	38
7.8	Commutativity . . . . .	39
7.9	Behaviour with Injectivity . . . . .	39
7.10	Definition of <i>first-elem</i> . . . . .	39
<b>8</b>	<b>The Global Non-Deterministic Choice</b>	<b>41</b>
8.1	General Non-Deterministic Choice Definition . . . . .	41
8.2	The Projections . . . . .	42
8.3	Factorization of $(\sqcap)$ in front of <i>GlobalNdet</i> . . . . .	42
8.4	$\perp$ Absorbance . . . . .	42
8.5	First Properties . . . . .	43

8.6	Behaviour of <i>GlobalNdet</i> with $(\sqcap)$ . . . . .	43
8.7	Commutativity . . . . .	43
8.8	Behaviour with Injectivity . . . . .	43
8.9	Cartesian Product Results . . . . .	43
8.10	Link with <i>MultiNdet</i> . . . . .	44
8.11	Link with <i>Mndetprefix</i> . . . . .	44
8.12	Properties . . . . .	44
<b>9</b>	<b>CSPM</b> . . . . .	<b>47</b>
9.1	Refinements Results . . . . .	47
9.2	Combination of Multi-Operators Laws . . . . .	51
9.3	Results on <i>Renaming</i> . . . . .	54
<b>10</b>	<b>Example: Dining Philosophers</b> . . . . .	<b>57</b>
10.1	Classic Version . . . . .	57
10.2	Formalization with <code>.-{ }fixrec</code> Package . . . . .	58
<b>11</b>	<b>Example: Plain Old Telephone System</b> . . . . .	<b>61</b>
11.1	The Alphabet and Basic Types of POTS . . . . .	62
11.2	Auxilliaris to Substructure the Specification . . . . .	63
11.3	A Telephone . . . . .	64
11.4	A Connector with the Network . . . . .	65
11.5	Combining NETWORK and TELEPHONES to a SYSTEM . . . . .	66
11.6	Simple Model of a User . . . . .	66
11.7	Toplevel Proof-Goals . . . . .	67
<b>12</b>	<b>Results on <i>events-of</i></b> . . . . .	<b>69</b>
12.1	With Operators of HOL-CSP . . . . .	69
12.2	With Architectural Operators of HOL-CSPM . . . . .	71
<b>13</b>	<b>Deadlock Results</b> . . . . .	<b>73</b>
13.1	Unfolding Lemmas for the Projections of <i>DF</i> and <i>DF<sub>SKIP</sub></i> . . . . .	73
13.2	Characterizations for <i>deadlock-free</i> , <i>deadlock-free<sub>SKIP</sub></i> . . . . .	74
13.3	Results on <i>Renaming</i> . . . . .	77
13.3.1	Behaviour with References Processes . . . . .	77
13.3.2	Corollaries on <i>deadlock-free</i> and <i>deadlock-free<sub>SKIP</sub></i> . . . . .	78
13.4	Big Results . . . . .	79
13.4.1	Interesting Equivalence . . . . .	79
13.4.2	<i>STOP</i> and <i>SKIP</i> Synchronized with <i>DF A</i> . . . . .	79
13.4.3	Finally, <i>deadlock-free</i> ( $P   Q$ ) . . . . .	79
<b>14</b>	<b>Conclusion</b> . . . . .	<b>83</b>





# Chapter 1

## Introduction

### 1.1 Motivations

HOL-CSP [3] is a formalization in Isabelle/HOL of the work of Hoare and Roscoe on the denotational semantics of the Failure/Divergence Model of CSP. It follows essentially the presentation of CSP in Roscoe's Book "Theory and Practice of Concurrency" [2] and the semantic details in a joint Paper of Roscoe and Brooks "An improved failures model for communicating processes" [1].

In the session HOL-CSP are introduced the type  $\alpha$  process, several classic CSP operators and number of laws that govern their interactions.

Four of them are binary operators: the non-deterministic choice  $P \sqcap Q$ , the deterministic choice  $P \sqcap Q$ , the synchronization  $P \llbracket S \rrbracket Q$  and the sequential composition  $P ; Q$ .

Analogously to the finite sum  $\sum_{i=0}^n a_i$  which is generalization of the addition  $a + b$ , we define generalisations of the binary operators of CSP.

The most straight-forward way to do so would be a fold on a list of processes. However, in many cases, we have additional properties, like commutativity, idempotency, etc. that allow for stronger/more abstract constructions. In particular, in several cases, generalization to unbounded and even infinite index-sets are possible.

The notations we choose are widely inspired by the  $\text{CSP}_M$  syntax of FDR: <https://cocotec.io/fdr/manual/cspm.html>.

In this session we therefore introduce the multi-operators associated respectively with  $P \sqcap Q$ ,  $P \sqcap Q$ ,  $P \llbracket S \rrbracket Q$  and  $P ; Q$ . We prove their continuity and refinements rules, as well as some laws governing their interactions.

We also give the definitions of the POTS and Dining Philosophers examples, which greatly benefit from the newly introduced generalized operators.

Since they appear naturally when modeling complex architectures, we may call them *architectural operators* of CSP.

Finally this session also includes results on the notion of *events-of*, and a very powerful result about *deadlock-free* and *Sync*: the interleaving  $P|||Q$  is *deadlock-free* if  $P$  and  $Q$  are.

## 1.2 The Global Architecture of HOL-CSPM

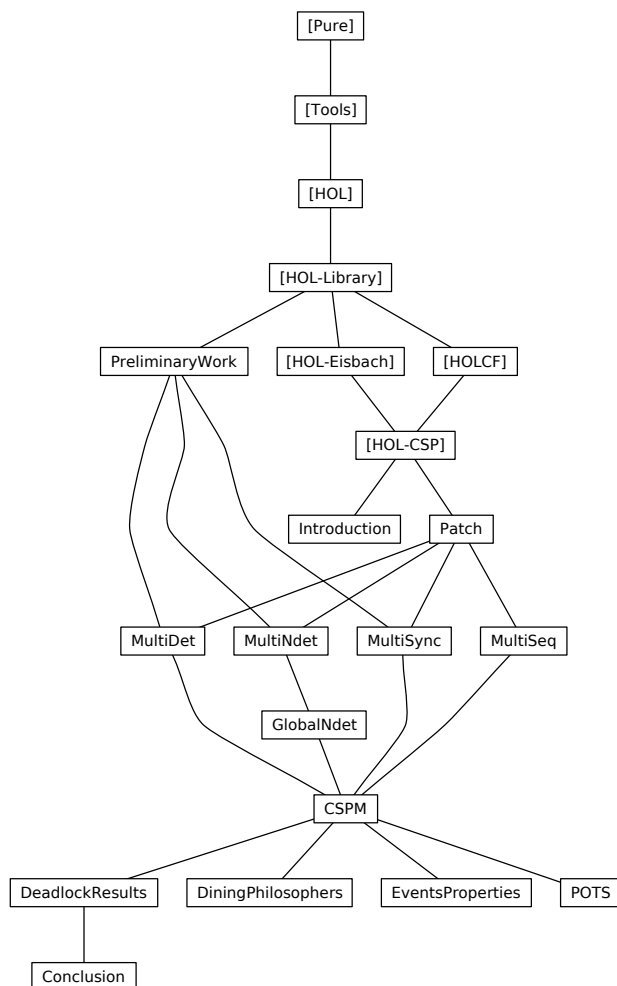


Figure 1.1: The overall architecture

The global architecture of HOL-CSPM is shown in [Figure 1.1](#). The entire package resides on:

1. HOL-Eisbach from the Isabelle/HOL distribution,
2. HOLCF from the Isabelle/HOL distribution, and
3. HOL-CSP 2.0 from the Isabelle Archive of Formal Proofs.



## Chapter 2

# Some Preliminary Work

```
theory PreliminaryWork
  imports HOL-Library.Multiset
begin
```

### 2.1 Induction Rules for $\alpha$ set

**lemma** *finite-subset-induct-singleton*

```
[consumes 3, case-names singleton insertion]:
⟨[[a ∈ A; finite F; F ⊆ A; P {a};
  ∧x F. finite F ⇒ x ∈ A ⇒ x ∉ (insert a F) ⇒ P (insert a F)
  ⇒ P (insert x (insert a F))] ⇒ P (insert a F)⟩
⟨proof⟩
```

**lemma** *finite-set-induct-nonempty*

```
[consumes 2, case-names singleton insertion]:
assumes ⟨A ≠ {}⟩ and ⟨finite A⟩
and singleton: ⟨∧a. a ∈ A ⇒ P {a}⟩
and insert: ⟨∧x F. [[F ≠ {}]; finite F; x ∈ A; x ∉ F; P F]
  ⇒ P (insert x F)⟩
shows ⟨P A⟩
⟨proof⟩
```

**lemma** *finite-subset-induct-singleton'*

```
[consumes 3, case-names singleton insertion]:
⟨[[a ∈ A; finite F; F ⊆ A; P {a};
  ∧x F. [[finite F; x ∈ A; insert a F ⊆ A; x ∉ insert a F; P (insert a F)]
  ⇒ P (insert x (insert a F))]
  ⇒ P (insert a F)⟩
⟨proof⟩
```

**lemma** *induct-subset-empty-single*[consumes 1]:

$\langle \llbracket \text{finite } A; P \{\}; \forall a \in A. P \{a\};$   
 $\bigwedge F a. \llbracket a \in A; \text{finite } F; F \subseteq A; F \neq \{\}; P F \rrbracket \implies P (\text{insert } a F) \rrbracket \implies P A \rangle$   
 $\langle \text{proof} \rangle$

## 2.2 Induction Rules for $\alpha$ multiset

The following rule comes directly from *HOL-Library.Multiset* but is written with *consumes 2* instead of *consumes 1*. I rewrite here a correct version.

**lemma** *msubset-induct* [*consumes 1, case-names empty add*]:  
 $\langle \llbracket F \subseteq\# A; P \{\#\}; \bigwedge a F. \llbracket a \in\# A; P F \rrbracket \implies P (\text{add-mset } a F) \rrbracket \implies P F \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *msubset-induct-singleton* [*consumes 2, case-names m-singleton add*]:  
 $\langle \llbracket a \in\# A; F \subseteq\# A; P \{\#a\#\};$   
 $\bigwedge x F. \llbracket x \in\# A; P (\text{add-mset } a F) \rrbracket \implies P (\text{add-mset } x (\text{add-mset } a F)) \rrbracket$   
 $\implies P (\text{add-mset } a F) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mset-induct-nonempty* [*consumes 1, case-names m-singleton add*]:  
**assumes**  $\langle A \neq \{\#\} \rangle$   
**and** *m-singleton*:  $\langle \bigwedge a. a \in\# A \implies P \{\#a\#\} \rangle$   
**and** *add*:  $\langle \bigwedge x F. \llbracket F \neq \{\#\}; x \in\# A; P F \rrbracket \implies P (\text{add-mset } x F) \rangle$   
**shows**  $\langle P A \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *msubset-induct'* [*consumes 2, case-names empty add*]:  
**assumes**  $\langle F \subseteq\# A \rangle$   
**and** *empty*:  $\langle P \{\#\} \rangle$   
**and** *insert*:  $\langle \bigwedge a F. \llbracket a \in\# A - F; F \subseteq\# A; P F \rrbracket \implies P (\text{add-mset } a F) \rangle$   
**shows**  $\langle P F \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *msubset-induct-singleton'* [*consumes 2, case-names m-singleton add*]:  
 $\langle \llbracket a \in\# A - F; F \subseteq\# A; P \{\#a\#\};$   
 $\bigwedge x F. \llbracket x \in\# A - F; F \subseteq\# A; P (\text{add-mset } a F) \rrbracket$   
 $\implies P (\text{add-mset } x (\text{add-mset } a F)) \rrbracket$   
 $\implies P (\text{add-mset } a F) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *msubset-induct-singleton''* [*consumes 1, case-names m-singleton add*]:  
 $\langle \llbracket \text{add-mset } a F \subseteq\# A; P \{\#a\#\};$   
 $\bigwedge x F. \llbracket \text{add-mset } x (\text{add-mset } a F) \subseteq\# A; P (\text{add-mset } a F) \rrbracket$   
 $\implies P (\text{add-mset } x (\text{add-mset } a F)) \rrbracket \rangle$

$\implies P (\text{add-mset } a \ F)$   
 $\langle \text{proof} \rangle$

**lemma** *mset-induct-nonempty'* [consumes 1, case-names m-singleton add]:  
**assumes** *nonempty*:  $\langle A \neq \{\#\} \rangle$  **and** *m-singleton*:  $\langle \bigwedge a. a \in\# A \implies P \{\#a\# \} \rangle$   
**and hyp**:  $\langle \bigwedge a \ x \ F. \llbracket a \in\# A; x \in\# A - \text{add-mset } a \ F; \text{add-mset } a \ F \subseteq\# A; P (\text{add-mset } a \ F) \rrbracket \implies P (\text{add-mset } x (\text{add-mset } a \ F)) \rangle$   
**shows**  $\langle P \ A \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *induct-subset-mset-empty-single*:  
 $\langle \llbracket P \ \{\#\}; \forall a \in\# M. P \ \{\#a\# \}; \bigwedge N \ a. \llbracket a \in\# M; N \subseteq\# M; N \neq \{\#\}; P \ N \rrbracket \implies P (\text{add-mset } a \ N) \rrbracket \implies P \ M \rangle$   
 $\langle \text{proof} \rangle$

## 2.3 Strong Induction for *nat*

**lemma** *strong-nat-induct*[consumes 0, case-names 0 Suc]:  
 $\langle \llbracket P \ 0; \bigwedge n. (\bigwedge m. m \leq n \implies P \ m) \rrbracket \implies P \ (\text{Suc } n) \rrbracket \implies P \ n \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *strong-nat-induct-non-zero*[consumes 1, case-names 1 Suc]:  
 $\langle \llbracket 0 < n; P \ 1; \bigwedge n. 0 < n \implies (\bigwedge m. 0 < m \wedge m \leq n \implies P \ m) \rrbracket \implies P \ (\text{Suc } n) \rrbracket \implies P \ n \rangle$   
 $\langle \text{proof} \rangle$

## 2.4 Preliminaries for Cartesian Product Results

**lemma** *prem-Multi-cartprod*:  
 $\langle (\lambda(x, y). x \ @ \ y) \ ' (A \times B) = \{s \ @ \ t \ | \ s \ t. (s, t) \in A \times B \} \rangle$   
 $\langle (\lambda(x, y). x \ \# \ y) \ ' (A' \times B) = \{s \ \# \ t \ | \ s \ t. (s, t) \in A' \times B \} \rangle$   
 $\langle (\lambda(x, y). [x, y]) \ ' (A' \times B') = \{[s, t] \ | \ s \ t. (s, t) \in A' \times B' \} \rangle$   
 $\langle \text{proof} \rangle$

**end**





## Chapter 3

# Patch for Compatibility

```
theory Patch
  imports HOL-CSP.Assertions
begin
```

HOL-CSP significantly changed during the past months, but not all the modifications appear in the current version on the AFP. This theory fixes the incompatibilities and will be removed in the next release.

### 3.1 Results

**lemma** *Mprefix-Det-distr*:

```
⟨(□ a ∈ A → P a) □ (□ b ∈ B → Q b) =
  □ x ∈ A ∪ B → ( if x ∈ A ∩ B then P x □ Q x
                    else if x ∈ A then P x else Q x)⟩
(is ⟨?lhs = ?rhs⟩)
⟨proof⟩
```

**lemma** *D-expand* :

```
⟨D P = {t1 @ t2 | t1 t2. t1 ∈ D P ∧ tickFree t1 ∧ front-tickFree t2}⟩
(is ⟨D P = ?rhs⟩)
⟨proof⟩
```

#### 3.1.1 Continuity Rule

**Monotonicity of Renaming.**

**lemma** *mono-Renaming[simp]* : ⟨(Renaming P f) ⊆ (Renaming Q f)⟩ if ⟨P ⊆ Q⟩  
⟨proof⟩

**Useful Results about finitary, and Preliminaries Lemmas for Continuity.**

**lemma** *le-snoc-is* : ⟨t ≤ s @ [x] ⟷ t = s @ [x] ∨ t ≤ s⟩  
⟨proof⟩

**lemma** *Cont-RenH5*:  $\langle \text{finite } (\bigcup t \in \{t. t \leq (s :: 'a \text{ trace})\}. \{s. t = \text{map } (\text{EvExt } f) s\}) \rangle$  **if**  $\langle \text{finitary } f \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Cont-RenH7*:  
 $\langle \text{finite } \{t. \exists t2. \text{tickFree } t \wedge \text{front-tickFree } t2 \wedge s = \text{map } (\text{EvExt } f) t @ t2\} \rangle$   
**if**  $\langle \text{finitary } f \rangle$   
 $\langle \text{proof} \rangle$

### Finally, Continuity !

**lemma** *Cont-Renaming-prem*:  
 $\langle (\bigsqcup i. \text{Renaming } (Y i) f) = (\text{Renaming } (\bigsqcup i. Y i) f) \rangle$  **if** *finitary*:  $\langle \text{finitary } f \rangle$   
**and** *chain*:  $\langle \text{chain } Y \rangle$   
 $\langle \text{proof} \rangle$

### 3.1.2 Nice Properties

**lemma** *Renaming-inv*:  $\langle \text{Renaming } (\text{Renaming } P f) (\text{inv } f) = P \rangle$  **if**  $\langle \text{inj } f \rangle$   
 $\langle \text{proof} \rangle$

### 3.1.3 Renaming Laws

**lemma** *Renaming-Mprefix-inj-on*:  
 $\langle \text{Renaming } (\text{Mprefix } A P) f = \square b \in f ' A \rightarrow \text{Renaming } (P (\text{THE } a. a \in A \wedge f a = b)) f \rangle$   
**if** *inj-on-f*:  $\langle \text{inj-on } f A \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *Renaming-Mprefix-inj*:  
 $\langle \text{Renaming } (\text{Mprefix } A P) f = \square b \in f ' A \rightarrow \text{Renaming } (P (\text{THE } a. f a = b)) f \rangle$  **if** *inj-f*:  $\langle \text{inj } f \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *Renaming-Mndetprefix-inj-on*:  
 $\langle \text{Renaming } (\text{Mndetprefix } A P) f = \square b \in f ' A \rightarrow \text{Renaming } (P (\text{THE } a. a \in A \wedge f a = b)) f \rangle$   
**if** *inj-on-f*:  $\langle \text{inj-on } f A \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *Renaming-Mndetprefix-inj*:

$\langle \text{Renaming } (\text{Mndetprefix } A \ P) \ f = \sqcap b \in f \ 'A \rightarrow \text{Renaming } (P \ (\text{THE } a. \ f \ a = b)) \ f \rangle$   
**if** *inj-f*:  $\langle \text{inj } f \rangle$   
 $\langle \text{proof} \rangle$

## 3.2 Assertions

**abbreviation** *deadlock-free<sub>SKIP</sub>* :: 'a process  $\Rightarrow$  bool

**where** *deadlock-free<sub>SKIP</sub>*  $\equiv$  *deadlock-free-v2*

**lemma** *deadlock-free-implies-lifelock-free*:  $\langle \text{deadlock-free } P \Longrightarrow \text{lifelock-free } P \rangle$

$\langle \text{proof} \rangle$

**lemmas** *deadlock-free<sub>SKIP</sub>-def* = *deadlock-free-v2-def*

**and** *deadlock-free<sub>SKIP</sub>-is-right* = *deadlock-free-v2-is-right*

**and** *deadlock-free<sub>SKIP</sub>-implies-div-free* = *deadlock-free-v2-implies-div-free*

**and** *deadlock-free<sub>SKIP</sub>-FD* = *deadlock-free-v2-FD*

**and** *deadlock-free<sub>SKIP</sub>-is-right-wrt-events* = *deadlock-free-v2-is-right-wrt-events*

**and** *deadlock-free-is-deadlock-free<sub>SKIP</sub>* = *deadlock-free-is-deadlock-free-v2*

**and** *deadlock-free<sub>SKIP</sub>-SKIP* = *deadlock-free-v2-SKIP*

**and** *non-deadlock-free<sub>SKIP</sub>-STOP* = *non-deadlock-free-v2-STOP*

## 3.3 Lifelock Freeness

**definition** *lifelock-free<sub>SKIP</sub>* :: 'a process  $\Rightarrow$  bool

**where** *lifelock-free<sub>SKIP</sub>*  $P \equiv \text{CHAOS}_{\text{SKIP}} \text{ UNIV } \sqsubseteq_{\text{FD}} P$

**lemma** *div-free-is-lifelock-free<sub>SKIP</sub>*:  $\langle \text{lifelock-free}_{\text{SKIP}} P \longleftrightarrow \mathcal{D} P = \{\} \rangle$

$\langle \text{proof} \rangle$

**lemma** *lifelock-free-is-lifelock-free<sub>SKIP</sub>*:  $\langle \text{lifelock-free } P \Longrightarrow \text{lifelock-free}_{\text{SKIP}} P \rangle$

$\langle \text{proof} \rangle$

**corollary** *deadlock-free<sub>SKIP</sub>-is-lifelock-free<sub>SKIP</sub>*:  $\langle \text{deadlock-free}_{\text{SKIP}} P \Longrightarrow \text{lifelock-free}_{\text{SKIP}} P \rangle$

$\langle \text{proof} \rangle$

## 3.4 New Laws

**lemma** *non-terminating-Sync*:

$\langle \text{non-terminating } P \Longrightarrow \text{lifelock-free}_{\text{SKIP}} Q \Longrightarrow \text{non-terminating } (P \llbracket A \rrbracket Q) \rangle$

$\langle \text{proof} \rangle$

**lemmas** *non-terminating-Par* = *non-terminating-Sync*[**where**  $A = \langle \text{UNIV} \rangle$ ]

**and** *non-terminating-Inter* = *non-terminating-Sync*[**where**  $A = \langle \{\} \rangle$ ]

**syntax**

*-writeS* :: [*'b* ⇒ *'a*, *pttrn*, *'b set*, *'a process*] ⇒ *'a process* ((*4*(*!**-*|*-*) /→ *-*)  
[*0,0,50,78*] *50*)

**translations**

*-writeS c p b P* ⇒ *CONST Mndetprefix* (*c* ‘ {*p*. *b*} ) (*λ*-. *P*)

**end**

## Chapter 4

# The MultiDet Operator

```
theory MultiDet
  imports Patch PreliminaryWork
begin
```

### 4.1 Definition

```
definition MultiDet :: ⟨'a set, 'a ⇒ 'b process⟩ ⇒ 'b process
  where MultiDet A P = Finite-Set.fold (λa r. r □ P a) STOP A
```

```
syntax -MultiDet :: ⟨[pttrn, 'a set, 'b process] ⇒ 'b process⟩ (⟨(ℱ□-∈-. / -)⟩ 75)
translations □ p ∈ A. P ⇒ CONST MultiDet A (λp. P)
```

### 4.2 First Properties

```
lemma MultiDet-rec0[simp]: ⟨(□ p ∈ {}). P p⟩ = STOP
  ⟨proof⟩
```

```
lemma MultiDet-rec1[simp]: ⟨(□ p ∈ {a}. P p) = P a⟩
  ⟨proof⟩
```

```
lemma MultiDet-in-id[simp]:
  ⟨a ∈ A ⇒ (□ p ∈ insert a A. P p) = □ p ∈ A. P p⟩
  ⟨proof⟩
```

```
lemma MultiDet-insert[simp]:
  ⟨finite A ⇒ (□ p ∈ insert a A. P p) = P a □ (□ p ∈ A - {a}. P p)⟩
  ⟨proof⟩
```

```
lemma MultiDet-insert'[simp]:
```

$\langle \text{finite } A \implies (\Box p \in \text{insert } a \ A. P \ p) = (P \ a \ \Box (\Box p \in A. P \ p)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiDet-eq*:

$\langle \text{finite } A \implies \forall x \in A. P \ x = Q \ x \implies \text{MultiDet } A \ P = \text{MultiDet } A \ Q \rangle$   
 $\langle \text{proof} \rangle$

### 4.3 Some Tests

**lemma** *test-MultiDet*:  $\langle (\Box p \in \{1::\text{int} \ .. \ 3\}. P \ p) = P \ 1 \ \Box P \ 2 \ \Box P \ 3 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *test-MultiDet'*:

$\langle (\Box p \in \{0::\text{nat} \ .. \ a\}. P \ p) = (\Box p \in \{a\} \cup \{1 \ .. \ a\} \cup \{0\}. P \ p) \rangle$   
 $\langle \text{proof} \rangle$

### 4.4 Continuity

**lemma** *MultiDet-cont[simp]*:

$\langle \llbracket \text{finite } A; \forall x \in A. \text{cont } (P \ x) \rrbracket \implies \text{cont } (\lambda y. \Box z \in A. P \ z \ y) \rangle$   
 $\langle \text{proof} \rangle$

### 4.5 Factorization of $(\Box)$ in front of *MultiDet*

**lemma** *MultiDet-factorization-union*:

$\langle \llbracket \text{finite } A; \text{finite } B \rrbracket \implies (\Box p \in A. P \ p) \ \Box (\Box p \in B. P \ p) = \Box p \in A \cup B. P \ p \rangle$   
 $\langle \text{proof} \rangle$

### 4.6 $\perp$ Absorbance

**lemma** *MultiDet-BOT-absorb*:

**assumes** *fin*:  $\langle \text{finite } A \rangle$  **and** *bot*:  $\langle P \ a = \perp \rangle$  **and** *dom*:  $\langle a \in A \rangle$   
**shows**  $\langle (\Box x \in A. P \ x) = \perp \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-is-BOT-iff*:

$\langle \text{finite } A \implies \text{MultiDet } A \ P = \perp \iff (\exists a \in A. P \ a = \perp) \rangle$   
 $\langle \text{proof} \rangle$

### 4.7 First Properties

**lemma** *MultiDet-id*:  $\langle A \neq \{\} \implies \text{finite } A \implies (\Box p \in A. P) = P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-STOP-id*:  $\langle \text{finite } A \implies (\Box p \in A. \text{STOP}) = \text{STOP} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-STOP-neutral*:  
 $\langle \text{finite } A \implies P a = \text{STOP} \implies (\Box z \in \text{insert } a A. P z) = \Box z \in A. P z \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-is-STOP-iff*:  
 $\langle \text{finite } A \implies (\Box a \in A. P a) = \text{STOP} \iff A = \{\} \vee (\forall a \in A. P a = \text{STOP}) \rangle$   
 $\langle \text{proof} \rangle$

## 4.8 Behaviour of *MultiDet* with $(\Box)$

**lemma** *MultiDet-Det*:  
 $\langle \text{finite } A \implies (\Box a \in A. P a) \Box (\Box a \in A. Q a) = \Box a \in A. P a \Box Q a \rangle$   
 $\langle \text{proof} \rangle$

## 4.9 Commutativity

**lemma** *MultiDet-sets-commute*:  
 $\langle \llbracket \text{finite } A; \text{finite } B \rrbracket \implies (\Box a \in A. \Box b \in B. P a b) = \Box b \in B. \Box a \in A. P a b \rangle$   
 $\langle \text{proof} \rangle$

## 4.10 Behaviour with Injectivity

**lemma** *inj-on-mapping-over-MultiDet*:  
 $\langle \llbracket \text{finite } A; \text{inj-on } f A \rrbracket \implies (\Box x \in A. P x) = \Box x \in f^{-1} A. P (\text{inv-into } A f x) \rangle$   
 $\langle \text{proof} \rangle$

## 4.11 The Projections

**lemma** *D-MultiDet*:  $\langle \text{finite } A \implies \mathcal{D} (\Box x \in A. P x) = (\bigcup p \in A. \mathcal{D} (P p)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *T-MultiDet*:  
 $\langle \text{finite } A \implies \mathcal{T} (\Box x \in A. P x) = (\text{if } A = \{\} \text{ then } \{\}\} \text{ else } \bigcup p \in A. \mathcal{T} (P p)) \rangle$   
 $\langle \text{proof} \rangle$

## 4.12 Cartesian Product Results

**lemma** *MultiDet-cartprod- $\sigma$ s-set- $\sigma$ s-set*:

$\langle \llbracket \text{finite } A; \text{ finite } B; \forall s \in A. \text{ length } s = \text{len}_1 \rrbracket \Longrightarrow$   
 $(\square (s, t) \in A \times B. P (s @ t)) = \square u \in \{s @ t \mid s t. (s, t) \in A \times B\}. P u \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-cartprod-s-set-σs-set:*

$\langle \llbracket \text{finite } A; \text{ finite } B \rrbracket \Longrightarrow$   
 $(\square (s, t) \in A \times B. P (s \# t)) = \square u \in \{s \# t \mid s t. (s, t) \in A \times B\}. P u \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-cartprod-s-set-s-set:*

$\langle \llbracket \text{finite } A; \text{ finite } B \rrbracket \Longrightarrow$   
 $(\square (s, t) \in A \times B. P [s, t]) = \square u \in \{[s, t] \mid s t. (s, t) \in A \times B\}. P u \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-cartprod:*

$\langle \text{finite } A \Longrightarrow \text{finite } B \Longrightarrow (\square (s, t) \in A \times B. P s t) = \square s \in A. \square t \in B. P s t \rangle$   
 $\langle \text{proof} \rangle$

**end**



## Chapter 5

# The MultiNdet Operator

```
theory MultiNdet
  imports Patch PreliminaryWork
begin
```

### 5.1 Definition

Defining the multi operator of  $(\sqcap)$  requires more work than with  $(\sqcup)$  since there is no neutral element. We will first build a version on ' $\alpha$  list' that we will generalize to ' $\alpha$  set'.

```
fun MultiNdet-list :: <['a list, 'a  $\Rightarrow$  'b process]  $\Rightarrow$  'b process>
  where <MultiNdet-list [] P = STOP>
  | <MultiNdet-list (a # l) P = fold ( $\lambda x r. r \sqcap P x$ ) l (P a)>
```

```
syntax -MultiNdet-list :: <[pttrn, 'a set, 'b process]  $\Rightarrow$  'b process>
      <( $\exists \sqcap_l$  - $\in$ -. / -)> 76)
```

```
translations  $\sqcap_l p \in l. P \Rightarrow \text{CONST MultiNdet-list } l (\lambda p. P)$ 
```

```
interpretation MultiNdet: comp-fun-idem where f= $\lambda x r. r \sqcap P x$ 
  <proof>
```

```
lemma MultiNdet-list-set:
  <set L = set L'  $\Longrightarrow$  MultiNdet-list L P = MultiNdet-list L' P>
  <proof>
```

```
definition MultiNdet :: <['a set, 'a  $\Rightarrow$  'b process]  $\Rightarrow$  'b process>
  where <MultiNdet A P = MultiNdet-list (SOME L. set L = A) P>
```

```
syntax -MultiNdet :: <[pttrn, 'a set, 'b process]  $\Rightarrow$  'b process> <( $\exists \sqcap$  - $\in$ -. / -)> 76)
```

translations  $\prod p \in A. P \equiv \text{CONST MultiNdet } A (\lambda p. P)$

## 5.2 First Properties

**lemma** *MultiNdet-rec0[simp]*:  $\langle (\prod p \in \{\}. P p) = \text{STOP} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiNdet-rec1[simp]*:  $\langle (\prod p \in \{a\}. P p) = P a \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiNdet-in-id[simp]*:  
 $\langle a \in A \implies (\prod p \in \text{insert } a \ A. P p) = \prod p \in A. P p \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiNdet-insert[simp]*:  
**assumes** *fin*:  $\langle \text{finite } A \rangle$  **and** *notempty*:  $\langle A \neq \{\} \rangle$  **and** *notin*:  $\langle a \notin A \rangle$   
**shows**  $\langle (\prod p \in \text{insert } a \ A. P p) = P a \sqcap (\prod p \in A. P p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiNdet-insert'[simp]*:  
 $\langle [\text{finite } A; A \neq \{\}] \implies (\prod p \in \text{insert } a \ A. P p) = P a \sqcap (\prod p \in A. P p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiNdet-eq*:  
 $\langle \text{finite } A \implies \forall x \in A. P x = Q x \implies \text{MultiNdet } A \ P = \text{MultiNdet } A \ Q \rangle$   
 $\langle \text{proof} \rangle$

## 5.3 Some Tests

**lemma**  $\langle (\prod_l p \in []. P p) = \text{STOP} \rangle$   
**and**  $\langle (\prod_l p \in [a]. P p) = P a \rangle$   
**and**  $\langle (\prod_l p \in [a, b]. P p) = P a \sqcap P b \rangle$   
**and**  $\langle (\prod_l p \in [a, b, c]. P p) = P a \sqcap P b \sqcap P c \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\langle (\prod p \in \{\}. P p) = \text{STOP} \rangle$   
**and**  $\langle (\prod p \in \{a\}. P p) = P a \rangle$   
**and**  $\langle (\prod p \in \{a, b\}. P p) = P a \sqcap P b \rangle$   
**and**  $\langle (\prod p \in \{a, b, c\}. P p) = P a \sqcap P b \sqcap P c \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *test-MultiNdet*:  $\langle (\prod p \in \{1::int .. 3\}. P p) = P 1 \sqcap P 2 \sqcap P 3 \rangle$   
 $\langle proof \rangle$

**lemma** *test-MultiNdet'*:  
 $\langle (\prod p \in \{0::nat .. a\}. P p) = (\prod p \in \{a\} \cup \{1 .. a\} \cup \{0\}. P p) \rangle$   
 $\langle proof \rangle$

## 5.4 Continuity

**lemma** *MultiNdet-cont[simp]*:  
 $\langle [finite A; \forall x \in A. cont (P x)] \implies cont (\lambda y. \prod z \in A. P z y) \rangle$   
 $\langle proof \rangle$

## 5.5 Factorization of $(\prod)$ in front of *MultiNdet*

**lemma** *MultiNdet-factorization-union*:  
 $\langle [A \neq \{\}; finite A; B \neq \{\}; finite B] \implies$   
 $(\prod p \in A. P p) \sqcap (\prod p \in B. P p) = \prod p \in A \cup B. P p \rangle$   
 $\langle proof \rangle$

## 5.6 $\perp$ Absorbance

**lemma** *MultiNdet-BOT-absorb*:  
**assumes** *fin*:  $\langle finite A \rangle$  **and** *bot*:  $\langle P a = \perp \rangle$  **and** *dom*:  $\langle a \in A \rangle$   
**shows**  $\langle (\prod x \in A. P x) = \perp \rangle$   
 $\langle proof \rangle$

**lemma** *MultiNdet-is-BOT-iff*:  
 $\langle finite A \implies (\prod p \in A. P p) = \perp \iff (\exists a \in A. P a = \perp) \rangle$   
 $\langle proof \rangle$

## 5.7 First Properties

**lemma** *MultiNdet-id*:  $\langle A \neq \{\} \implies finite A \implies (\prod p \in A. P) = P \rangle$   
 $\langle proof \rangle$

**lemma** *MultiNdet-STOP-id*:  $\langle finite A \implies (\prod p \in A. STOP) = STOP \rangle$   
 $\langle proof \rangle$

**lemma** *MultiNdet-is-STOP-iff*:  
 $\langle finite A \implies (\prod p \in A. P p) = STOP \iff A = \{\} \vee (\forall a \in A. P a = STOP) \rangle$   
 $\langle proof \rangle$

## 5.8 Behaviour of *MultiNdet* with $(\sqcap)$

**lemma** *MultiNdet-Ndet*:

$$\langle \llbracket \text{finite } A \rrbracket \implies (\sqcap a \in A. P a) \sqcap (\sqcap a \in A. Q a) = \sqcap a \in A. P a \sqcap Q a \rangle$$

*<proof>*

## 5.9 Commutativity

**lemma** *MultiNdet-sets-commute*:

$$\langle \llbracket \text{finite } A; \text{finite } B \rrbracket \implies (\sqcap a \in A. \sqcap b \in B. P a b) = \sqcap b \in B. \sqcap a \in A. P a b \rangle$$

*<proof>*

## 5.10 Behaviour with Injectivity

**lemma** *inj-on-mapping-over-MultiNdet*:

$$\langle \llbracket \text{finite } A; \text{inj-on } f A \rrbracket \implies (\sqcap x \in A. P x) = \sqcap x \in f^{-1} A. P (\text{inv-into } A f x) \rangle$$

*<proof>*

## 5.11 The Projections

**lemma** *D-MultiNdet*:  $\langle \text{finite } A \implies \mathcal{D} (\sqcap x \in A. P x) = (\bigcup p \in A. \mathcal{D} (P p)) \rangle$

*<proof>*

**lemma** *F-MultiNdet*:

$$\langle \text{finite } A \implies \mathcal{F} (\sqcap x \in A. P x) = (\text{if } A = \{\} \text{ then } \{(s, X). s = []\} \text{ else } \bigcup p \in A. \mathcal{F} (P p)) \rangle$$

*<proof>*

**lemma** *T-MultiNdet*:

$$\langle \text{finite } A \implies \mathcal{T} (\sqcap x \in A. P x) = (\text{if } A = \{\} \text{ then } \{[]\} \text{ else } \bigcup p \in A. \mathcal{T} (P p)) \rangle$$

*<proof>*

## 5.12 Cartesian Product Results

**lemma** *MultiNdet-cartprod- $\sigma$ s-set- $\sigma$ s-set*:

$$\langle \llbracket \text{finite } A; \text{finite } B; \forall s \in A. \text{length } s = \text{len}_1 \rrbracket \implies (\sqcap (s, t) \in A \times B. P (s @ t)) = \sqcap u \in \{s @ t \mid s t. (s, t) \in A \times B\}. P u \rangle$$

*<proof>*

**lemma** *MultiNdet-cartprod-s-set- $\sigma$ s-set*:

$$\langle \llbracket \text{finite } A; \text{finite } B \rrbracket \implies (\sqcap (s, t) \in A \times B. P (s \# t)) = \sqcap u \in \{s \# t \mid s t. (s, t) \in A \times B\}. P u \rangle$$

*<proof>*

**lemma** *MultiNdet-cartprod-s-set-s-set:*

$\langle \llbracket \text{finite } A; \text{finite } B \rrbracket \implies$

$(\prod (s, t) \in A \times B. P [s, t]) = \prod u \in \{[s, t] \mid s \ t. (s, t) \in A \times B\}. P u \rangle$

$\langle \text{proof} \rangle$

**lemma** *MultiNdet-cartprod:*

$\langle \llbracket \text{finite } A; \text{finite } B \rrbracket \implies (\prod (s, t) \in A \times B. P s t) = \prod s \in A. \prod t \in B. P s t \rangle$

$\langle \text{proof} \rangle$

**end**



## Chapter 6

# The MultiSync Operator

```
theory MultiSync
  imports HOL-Library.Multiset PreliminaryWork Patch
begin
```

### 6.1 Definition

As in the  $(\sqcap)$  case, we have no neutral element so we will also have to go through lists first. But the binary operator *Sync* is not idempotent either, so the generalization will be done on ' $\alpha$  multiset' and not on ' $\alpha$  set'.

Note that a ' $\alpha$  multiset' is by construction finite (cf. theorem *finite (set-mset M)*).

```
fun MultiSync-list :: <['b set, 'a list, 'a  $\Rightarrow$  'b process]  $\Rightarrow$  'b process>
  where <MultiSync-list S [] P = STOP>
  | <MultiSync-list S (l # L) P = fold ( $\lambda x r. r \llbracket S \rrbracket P x$ ) L (P l)>
```

```
syntax -MultiSync-list :: <[pttrn, 'b set, 'a list, 'b process]  $\Rightarrow$  'b process>
  (<( $\exists \llbracket - \rrbracket_l \in \cdot / \cdot$ ) 63>)
```

```
translations  $\llbracket S \rrbracket_l p \in L. P \Rightarrow \text{CONST MultiSync-list } S L (\lambda p. P)$ 
```

```
interpretation MultiSync: comp-fun-commute where f = < $\lambda x r. r \llbracket E \rrbracket P x$ >
  <proof>
```

```
lemma MultiSync-list-mset:
  <mset L = mset L'  $\Longrightarrow$  MultiSync-list S L P = MultiSync-list S L' P>
  <proof>
```

```
definition MultiSync :: <['b set, 'a multiset, 'a  $\Rightarrow$  'b process]  $\Rightarrow$  'b process>
  where <MultiSync S M P = MultiSync-list S (SOME L. mset L = M) P>
```

**syntax**  $\text{-MultiSync} :: \langle [pttrn, 'b \text{ set}, 'a \text{ multiset}, 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$   
 $(\langle (\exists [-] \text{-}\in\#\text{-} / \text{-}) \rangle 63)$

**translations**  $\llbracket S \rrbracket p \in\# M. P \equiv \text{CONST MultiSync } S M (\lambda p. P)$

Special case of  $\text{MultiSync } E P$  when  $E = \{\}$ .

**abbreviation**  $\text{MultiInter} :: \langle ['a \text{ multiset}, 'a \Rightarrow 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$   
**where**  $\langle \text{MultiInter } M P \equiv \text{MultiSync } \{\} M P \rangle$

**syntax**  $\text{-MultiInter} :: \langle [pttrn, 'a \text{ multiset}, 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$   
 $(\langle (\exists [||] \text{-}\in\#\text{-} / \text{-}) \rangle 77)$

**translations**  $||| p \in\# M. P \equiv \text{CONST MultiInter } M (\lambda p. P)$

Special case of  $\text{MultiSync } E P$  when  $E = \text{UNIV}$ .

**abbreviation**  $\text{MultiPar} :: \langle ['a \text{ multiset}, 'a \Rightarrow 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$   
**where**  $\langle \text{MultiPar } M P \equiv \text{MultiSync } \text{UNIV } M P \rangle$

**syntax**  $\text{-MultiPar} :: \langle [pttrn, 'a \text{ multiset}, 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$   
 $(\langle (\exists [|] \text{-}\in\#\text{-} / \text{-}) \rangle 77)$

**translations**  $|| p \in\# M. P \equiv \text{CONST MultiPar } M (\lambda p. P)$

## 6.2 First Properties

**lemma**  $\text{MultiSync-rec0}[\text{simp}]$ :  $\langle (\llbracket S \rrbracket p \in\# \{\#\}. P p) = \text{STOP} \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{MultiSync-rec1}[\text{simp}]$ :  $\langle (\llbracket S \rrbracket p \in\# \{\#a\#\}. P p) = P a \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{MultiSync-add}[\text{simp}]$ :  
 $\langle M \neq \{\#\} \implies (\llbracket S \rrbracket p \in\# \text{add-mset } m M. P p) = P m \llbracket S \rrbracket (\llbracket S \rrbracket p \in\# M. P p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{mono-MultiSync-eq}$ :  
 $\langle \forall x \in\# M. P x = Q x \implies \text{MultiSync } S M P = \text{MultiSync } S M Q \rangle$   
 $\langle \text{proof} \rangle$

**lemmas**  $\text{MultiInter-rec0} = \text{MultiSync-rec0}[\text{where } S = \langle \{\} \rangle]$   
**and**  $\text{MultiPar-rec0} = \text{MultiSync-rec0}[\text{where } S = \langle \text{UNIV} \rangle]$   
**and**  $\text{MultiInter-rec1} = \text{MultiSync-rec1}[\text{where } S = \langle \{\} \rangle]$   
**and**  $\text{MultiPar-rec1} = \text{MultiSync-rec1}[\text{where } S = \langle \text{UNIV} \rangle]$   
**and**  $\text{MultiInter-add} = \text{MultiSync-add}[\text{where } S = \langle \{\} \rangle]$   
**and**  $\text{MultiPar-add} = \text{MultiSync-add}[\text{where } S = \langle \text{UNIV} \rangle]$   
**and**  $\text{mono-MultiInter-eq} = \text{mono-MultiSync-eq}[\text{where } S = \langle \{\} \rangle]$   
**and**  $\text{mono-MultiPar-eq} = \text{mono-MultiSync-eq}[\text{where } S = \langle \text{UNIV} \rangle]$



### 6.3 Some Tests

**lemma**  $\langle \llbracket S \rrbracket_l p \in []. P p \rangle = STOP$   
**and**  $\langle \llbracket S \rrbracket_l p \in [a]. P p \rangle = P a$   
**and**  $\langle \llbracket S \rrbracket_l p \in [a, b]. P p \rangle = P a \llbracket S \rrbracket P b$   
**and**  $\langle \llbracket S \rrbracket_l p \in [a, b, c]. P p \rangle = P a \llbracket S \rrbracket P b \llbracket S \rrbracket P c$   
 $\langle proof \rangle$

**lemma** *test-MultiSync*:  
 $\langle \llbracket S \rrbracket p \in \# mset []. P p \rangle = STOP$   
 $\langle \llbracket S \rrbracket p \in \# mset [a]. P p \rangle = P a$   
 $\langle \llbracket S \rrbracket p \in \# mset [a, b]. P p \rangle = P a \llbracket S \rrbracket P b$   
 $\langle \llbracket S \rrbracket p \in \# mset [a, b, c]. P p \rangle = P a \llbracket S \rrbracket P b \llbracket S \rrbracket P c$   
 $\langle proof \rangle$

**lemma** *MultiSync-set1*:  $\langle MultiSync S (mset-set \{k::nat..<k\}) P \rangle = STOP$   
 $\langle proof \rangle$

**lemma** *MultiSync-set2*:  $\langle MultiSync S (mset-set \{k..<Suc k\}) P \rangle = P k$   
 $\langle proof \rangle$

**lemma** *MultiSync-set3*:  
 $\langle l < k \implies MultiSync S (mset-set \{l..< Suc k\}) P =$   
 $P l \llbracket S \rrbracket (MultiSync S (mset-set \{Suc l ..< Suc k\}) P) \rangle$   
 $\langle proof \rangle$

**lemma** *test-MultiSync'*:  
 $\langle \llbracket S \rrbracket p \in \# mset-set \{1::int .. 3\}. P p \rangle = P 1 \llbracket S \rrbracket P 2 \llbracket S \rrbracket P 3$   
 $\langle proof \rangle$

**lemma** *test-MultiSync''*:  
 $\langle \llbracket S \rrbracket p \in \# mset-set \{0::nat .. a\}. P p \rangle =$   
 $\llbracket S \rrbracket p \in \# mset-set (\{a\} \cup \{1 .. a\} \cup \{0\}) . P p$   
 $\langle proof \rangle$

**lemmas** *test-MultiInter* = *test-MultiSync*[**where**  $S = \langle \{\} \rangle$ ]  
**and** *test-MultiPar* = *test-MultiSync*[**where**  $S = \langle UNIV \rangle$ ]  
**and** *MultiInter-set1* = *MultiSync-set1*[**where**  $S = \langle \{\} \rangle$ ]  
**and** *MultiPar-set1* = *MultiSync-set1*[**where**  $S = \langle UNIV \rangle$ ]  
**and** *MultiInter-set2* = *MultiSync-set2*[**where**  $S = \langle \{\} \rangle$ ]  
**and** *MultiPar-set2* = *MultiSync-set2*[**where**  $S = \langle UNIV \rangle$ ]

**and**  $MultiInter\text{-}set3 = MultiSync\text{-}set3[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar\text{-}set3 = MultiSync\text{-}set3[\mathbf{where} S = \langle UNIV \rangle]$   
**and**  $test\text{-}MultiInter' = test\text{-}MultiSync'[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $test\text{-}MultiPar' = test\text{-}MultiSync'[\mathbf{where} S = \langle UNIV \rangle]$   
**and**  $test\text{-}MultiInter'' = test\text{-}MultiSync''[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $test\text{-}MultiPar'' = test\text{-}MultiSync''[\mathbf{where} S = \langle UNIV \rangle]$

## 6.4 Continuity

**lemma**  $MultiSync\text{-}cont[simp]$ :  
 $\langle \forall x \in \# M. cont (P x) \implies cont (\lambda y. \llbracket S \rrbracket z \in \# M. P z y) \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter\text{-}cont[simp] = MultiSync\text{-}cont[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar\text{-}cont[simp] = MultiSync\text{-}cont[\mathbf{where} S = \langle UNIV \rangle]$

## 6.5 Factorization of *Sync* in front of *MultiSync*

**lemma**  $MultiSync\text{-}factorization\text{-}union$ :  
 $\langle \llbracket M \neq \{\# \}; N \neq \{\# \} \rrbracket \implies$   
 $\langle \llbracket S \rrbracket z \in \# M. P z \rrbracket \llbracket S \rrbracket (\llbracket S \rrbracket z \in \# N. P z) = \llbracket S \rrbracket z \in \# M + N. P z \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter\text{-}factorization\text{-}union =$   
 $MultiSync\text{-}factorization\text{-}union[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar\text{-}factorization\text{-}union =$   
 $MultiSync\text{-}factorization\text{-}union[\mathbf{where} S = \langle UNIV \rangle]$

## 6.6 $\perp$ Absorbance

**lemma**  $MultiSync\text{-}BOT\text{-}absorb$ :  
 $\langle m \in \# M \implies P m = \perp \implies (\llbracket S \rrbracket z \in \# M. P z) = \perp \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter\text{-}BOT\text{-}absorb = MultiSync\text{-}BOT\text{-}absorb[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar\text{-}BOT\text{-}absorb = MultiSync\text{-}BOT\text{-}absorb[\mathbf{where} S = \langle UNIV \rangle]$

**lemma**  $MultiSync\text{-}is\text{-}BOT\text{-}iff$ :  
 $\langle (\llbracket S \rrbracket m \in \# M. P m) = \perp \longleftrightarrow (\exists m \in \# M. P m = \perp) \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter\text{-}is\text{-}BOT\text{-}iff = MultiSync\text{-}is\text{-}BOT\text{-}iff[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar\text{-}is\text{-}BOT\text{-}iff = MultiSync\text{-}is\text{-}BOT\text{-}iff[\mathbf{where} S = \langle UNIV \rangle]$

## 6.7 Other Properties

**lemma** *MultiSync-SKIP-id*:  $\langle M \neq \{\#\} \implies (\llbracket S \rrbracket z \in \# M. SKIP) = SKIP \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter-SKIP-id = MultiSync-SKIP-id[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar-SKIP-id = MultiSync-SKIP-id[\mathbf{where} S = \langle UNIV \rangle]$

**lemma** *MultiPar-prefix-two-distincts-STOP*:  
**assumes**  $\langle m \in \# M \rangle$  **and**  $\langle m' \in \# M \rangle$  **and**  $\langle fst\ m \neq fst\ m' \rangle$   
**shows**  $\langle (\| a \in \# M. (fst\ a \rightarrow P\ (snd\ a)) = STOP \rangle$   
 $\langle proof \rangle$

**lemma** *MultiPar-prefix-two-distincts-STOP'*:  
 $\langle \llbracket (m, n) \in \# M; (m', n') \in \# M; m \neq m' \rrbracket \implies$   
 $\langle (\| (m, n) \in \# M. (m \rightarrow P\ n)) = STOP \rangle$   
 $\langle proof \rangle$

## 6.8 Behaviour of *MultiSync* with *Sync*

**lemma** *Sync-STOP-STOP*:  $\langle STOP \llbracket S \rrbracket STOP = STOP \rangle$   
 $\langle proof \rangle$

**lemma** *MultiSync-Sync*:  
 $\langle (\llbracket S \rrbracket z \in \# M. P\ z) \llbracket S \rrbracket (\llbracket S \rrbracket z \in \# M. P'\ z) = \llbracket S \rrbracket z \in \# M. P\ z \llbracket S \rrbracket P'\ z \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter-Inter = MultiSync-Sync[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar-Par = MultiSync-Sync[\mathbf{where} S = \langle UNIV \rangle]$

## 6.9 Commutativity

**lemma** *MultiSync-sets-commute*:  
 $\langle (\llbracket S \rrbracket a \in \# M. \llbracket S \rrbracket b \in \# N. P\ a\ b) = \llbracket S \rrbracket b \in \# N. \llbracket S \rrbracket a \in \# M. P\ a\ b \rangle$   
 $\langle proof \rangle$

**lemmas**  $MultiInter-sets-commute = MultiSync-sets-commute[\mathbf{where} S = \langle \{\} \rangle]$   
**and**  $MultiPar-sets-commute = MultiSync-sets-commute[\mathbf{where} S = \langle UNIV \rangle]$

## 6.10 Behaviour with Injectivity

**lemma** *inj-on-mapping-over-MultiSync*:

$\langle inj\text{-on } f \text{ (set-mset } M) \implies$   
 $(\llbracket S \rrbracket x \in\# M. P x) = \llbracket S \rrbracket x \in\# \text{image-mset } f M. P (\text{inv-into (set-mset } M) f x)\rangle$   
 $\langle \text{proof} \rangle$

**lemmas** *inj-on-mapping-over-MultiInter* =  
*inj-on-mapping-over-MultiSync*[**where**  $S = \langle \{\} \rangle$ ]  
**and** *inj-on-mapping-over-MultiPar* =  
*inj-on-mapping-over-MultiSync*[**where**  $S = \langle UNIV \rangle$ ]

**end**

# Chapter 7

## The MultiSeq Operator

```
theory MultiSeq
  imports Patch
begin
```

### 7.1 Definition

```
definition MultiSeq :: ⟨['a list, 'a ⇒ 'b process] ⇒ 'b process⟩
  where ⟨MultiSeq S P = foldr (λa r. (P a) ; r ) S SKIP⟩
```

```
syntax -MultiSeq :: ⟨[pttrn, 'a list, 'b process] ⇒ 'b process⟩
      (⟨(3SEQ -∈@-./ -)⟩ 73)
```

```
translations SEQ i ∈@ A. P ⇒ CONST MultiSeq A (λi. P)
```

### 7.2 First Properties

```
lemma MultiSeq-rec0[simp]: ⟨(SEQ p ∈@ []. P p) = SKIP⟩
  ⟨proof⟩
```

```
lemma MultiSeq-rec1[simp]: ⟨(SEQ p ∈@ [a]. P p) = P a⟩
  ⟨proof⟩
```

```
lemma MultiSeq-Cons[simp]: ⟨(SEQ i ∈@ a # L. P i) = P a ; (SEQ i ∈@ L. P
i)⟩
  ⟨proof⟩
```

### 7.3 Some Tests

```
lemma ⟨(SEQ p ∈@ []. P p) = SKIP⟩
  and ⟨(SEQ p ∈@ [a]. P p) = P a⟩
  and ⟨(SEQ p ∈@ [a,b]. P p) = P a ; P b⟩
```

**and**  $\langle (SEQ\ p \in@ [a,b,c].\ P\ p) = P\ a ; P\ b ; P\ c \rangle$   
 $\langle proof \rangle$

**lemma** *test-MultiSeq*:  $\langle (SEQ\ p \in@ [1::int .. 3].\ P\ p) = P\ 1 ; P\ 2 ; P\ 3 \rangle$   
 $\langle proof \rangle$

## 7.4 Continuity

**lemma** *MultiSeq-cont[simp]*:  
 $\langle \forall x \in set\ L.\ cont\ (P\ x) \implies cont\ (\lambda y.\ SEQ\ z \in@ L.\ P\ z\ y) \rangle$   
 $\langle proof \rangle$

## 7.5 Factorization of (;) in front of MultiSeq

**lemma** *MultiSeq-factorization-append*:  
 $\langle (SEQ\ p \in@ A.\ P\ p) ; (SEQ\ p \in@ B.\ P\ p) = (SEQ\ p \in@ A\ @\ B.\ P\ p) \rangle$   
 $\langle proof \rangle$

## 7.6 $\perp$ Absorbance

**lemma** *MultiSeq-BOT-absorb*:  
 $\langle P\ a = \perp \implies (SEQ\ z \in@ l1\ @\ [a]\ @\ l2.\ P\ z) = (SEQ\ z \in@ l1.\ P\ z) ; \perp \rangle$   
 $\langle proof \rangle$

## 7.7 First Properties

**lemma** *MultiSeq-SKIP-neutral*:  
 $\langle P\ a = SKIP \implies (SEQ\ z \in@ l1\ @\ [a]\ @\ l2.\ P\ z) = SEQ\ z \in@ l1\ @\ l2.\ P\ z \rangle$   
 $\langle proof \rangle$

**lemma** *MultiSeq-STOP-absorb*:  
 $\langle P\ a = STOP \implies$   
 $(SEQ\ z \in@ l1\ @\ [a]\ @\ l2.\ P\ z) = (SEQ\ z \in@ l1.\ P\ z) ; STOP \rangle$   
 $\langle proof \rangle$

**lemma** *mono-MultiSeq-eq*:  
 $\langle \forall x \in set\ L.\ P\ x = Q\ x \implies MultiSeq\ L\ P = MultiSeq\ L\ Q \rangle$   
 $\langle proof \rangle$

**lemma** *MultiSeq-is-SKIP-iff*:  
 $\langle MultiSeq\ L\ P = SKIP \iff (\forall a \in set\ L.\ P\ a = SKIP) \rangle$   
 $\langle proof \rangle$

## 7.8 Commutativity

Of course, since the sequential composition  $P ; Q$  is not commutative, the result here is negative: the order of the elements of list  $L$  does matter in  $SEQ\ z \in @L. P\ z$ .

## 7.9 Behaviour with Injectivity

**lemma** *inj-on-mapping-over-MultiSeq*:

$\langle inj\text{-on}\ f\ (set\ C) \implies$   
 $(SEQ\ x \in @\ C. P\ x) = SEQ\ x \in @\ map\ f\ C. P\ (inv\text{-into}\ (set\ C)\ f\ x) \rangle$   
 $\langle proof \rangle$

## 7.10 Definition of *first-elm*

**primrec** *first-elm* ::  $\langle 'a \Rightarrow bool, 'a\ list \rangle \Rightarrow nat$   
**where**  $\langle first\text{-elm}\ P\ [] = 0 \rangle$   
 $| \quad \langle first\text{-elm}\ P\ (x \# L) = (if\ P\ x\ then\ 0\ else\ Suc\ (first\text{-elm}\ P\ L)) \rangle$

*first-elm* returns the first index  $i$  such that  $P\ (L\ !\ i) = True$  if it exists, *length*  $L$  otherwise.

This will be very useful later.

**value**  $\langle first\text{-elm}\ (\lambda x. 4 < x)\ [0::nat, 2, 5] \rangle$   
**lemma**  $\langle first\text{-elm}\ (\lambda x. 5 < x)\ [0::nat, 2, 5] = 3 \rangle \langle proof \rangle$   
**lemma**  $\langle P\ ' set\ L \subseteq \{False\} \implies first\text{-elm}\ P\ L = length\ L \rangle \langle proof \rangle$

**end**





## Chapter 8

# The Global Non-Deterministic Choice

```
theory GlobalNdet
  imports MultiNdet
begin
```

### 8.1 General Non-Deterministic Choice Definition

This is an experimental definition of a generalized non-deterministic choice  $a \sqcap b$  for an arbitrary set. The present version is "totalised" for the case of  $A = \{\}$  by *STOP*, which is not the neutral element of the  $(\sqcap)$  operator (because there is no neutral element for  $(\sqcap)$ ).

**lemma**  $\langle \# P. \forall Q. (P :: 'a \text{ process}) \sqcap Q = Q \rangle$   
*<proof>*

**lift-definition** *GlobalNdet* ::  $\langle ['a \text{ set}, 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$   
**is**  $\langle \lambda A P. \text{ if } A = \{\}$   
     $\text{ then } (\{(s, X). s = []\}, \{\})$   
     $\text{ else } (\bigcup_{a \in A}. \mathcal{F} (P a), \bigcup_{a \in A}. \mathcal{D} (P a)) \rangle$   
*<proof>*

**syntax** *-GlobalNdet* ::  $\langle [pttrn, 'a \text{ set}, 'b \text{ process}] \Rightarrow 'b \text{ process} \rangle$  ( $\langle (\exists \sqcap - \in -. / -) \rangle$  76)  
**translations**  $\sqcap p \in A. P \equiv \text{CONST } \textit{GlobalNdet } A (\lambda p. P)$

Note that the global non-deterministic choice  $\sqcap p \in A. P p$  is different from the multi-non-deterministic prefix  $\sqcap p \in A \rightarrow P p$  which guarantees continuity even when  $A$  is *infinite* due to the fact that it communicates its choice

via an internal prefix operator.

It is also subtly different from the multi-non-deterministic choice  $\sqcap_{p \in A} P p$  which is only defined when  $A$  is *finite*.

**lemma** *empty-GlobalNdet[simp]* :  $\langle \text{GlobalNdet } \{\} P = \text{STOP} \rangle$   
 $\langle \text{proof} \rangle$

## 8.2 The Projections

**lemma** *F-GlobalNdet*:  
 $\langle \mathcal{F} (\sqcap x \in A. P x) = (\text{if } A = \{\} \text{ then } \{(s, X). s = \{\}\} \text{ else } (\bigcup x \in A. \mathcal{F} (P x))) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *D-GlobalNdet*:  
 $\langle \mathcal{D} (\sqcap x \in A. P x) = (\text{if } A = \{\} \text{ then } \{\} \text{ else } (\bigcup x \in A. \mathcal{D} (P x))) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *T-GlobalNdet*:  
 $\langle \mathcal{T} (\sqcap x \in A. P x) = (\text{if } A = \{\} \text{ then } \{\{\}\} \text{ else } (\bigcup x \in A. \mathcal{T} (P x))) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-GlobalNdet-eq*:  
 $\langle \forall x \in A. P x = Q x \implies \text{GlobalNdet } A P = \text{GlobalNdet } A Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-GlobalNdet-eq2*:  
 $\langle \forall x \in A. P (f x) = Q x \implies \text{GlobalNdet } (f \text{ ` } A) P = \text{GlobalNdet } A Q \rangle$   
 $\langle \text{proof} \rangle$

## 8.3 Factorization of $(\sqcap)$ in front of *GlobalNdet*

**lemma** *GlobalNdet-factorization-union*:  
 $\langle \llbracket A \neq \{\}; B \neq \{\} \rrbracket \implies$   
 $(\sqcap p \in A. P p) \sqcap (\sqcap p \in B. P p) = (\sqcap p \in A \cup B. P p) \rangle$   
 $\langle \text{proof} \rangle$

## 8.4 $\perp$ Absorbance

**lemma** *GlobalNdet-BOT-absorb*:  $\langle P a = \perp \implies a \in A \implies (\sqcap x \in A. P x) = \perp \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *GlobalNdet-is-BOT-iff*:  $\langle (\sqcap x \in A. P x) = \perp \iff (\exists a \in A. P a = \perp) \rangle$   
 $\langle \text{proof} \rangle$

## 8.5 First Properties

**lemma** *GlobalNdet-id*:  $\langle A \neq \{\} \implies (\prod p \in A. P) = P \rangle$   
*<proof>*

**lemma** *GlobalNdet-STOP-id*:  $\langle (\prod p \in A. STOP) = STOP \rangle$   
*<proof>*

**lemma** *GlobalNdet-unit[simp]*:  $\langle (\prod x \in \{a\}. P x) = P a \rangle$   
*<proof>*

**lemma** *GlobalNdet-distrib-unit*:  
 $\langle A - \{a\} \neq \{\} \implies (\prod x \in insert\ a\ A. P x) = P a \prod (\prod x \in A - \{a\}. P x) \rangle$   
*<proof>*

## 8.6 Behaviour of *GlobalNdet* with $(\prod)$

**lemma** *GlobalNdet-Ndet*:  
 $\langle (\prod a \in A. P a) \prod (\prod a \in A. Q a) = \prod a \in A. P a \prod Q a \rangle$   
*<proof>*

## 8.7 Commutativity

**lemma** *GlobalNdet-sets-commute*:  
 $\langle (\prod a \in A. \prod b \in B. P a b) = \prod b \in B. \prod a \in A. P a b \rangle$   
*<proof>*

## 8.8 Behaviour with Injectivity

**lemma** *inj-on-mapping-over-GlobalNdet*:  
 $\langle inj\ on\ f\ A \implies (\prod x \in A. P x) = \prod x \in f\ ' A. P (inv\ into\ A\ f\ x) \rangle$   
*<proof>*

## 8.9 Cartesian Product Results

**lemma** *GlobalNdet-cartprod- $\sigma s$ -set- $\sigma s$ -set*:  
 $\langle (\prod (s, t) \in A \times B. P (s @ t)) = \prod u \in \{s @ t \mid s\ t. (s, t) \in A \times B\}. P u \rangle$   
*<proof>*

**lemma** *GlobalNdet-cartprod-s-set- $\sigma s$ -set*:  
 $\langle (\prod (s, t) \in A \times B. P (s \# t)) = \prod u \in \{s \# t \mid s\ t. (s, t) \in A \times B\}. P u \rangle$   
*<proof>*

**lemma** *GlobalNdet-cartprod-s-set-s-set*:

$$\langle (\prod (s, t) \in A \times B. P [s, t]) = \prod u \in \{[s, t] \mid s \ t. (s, t) \in A \times B\}. P u \rangle$$

*<proof>*

**lemma** *GlobalNdet-cartprod*:  $\langle (\prod (s, t) \in A \times B. P s t) = \prod s \in A. \prod t \in B. P s t \rangle$

*<proof>*

## 8.10 Link with *MultiNdet*

This operator is in fact an extension of *MultiNdet* to arbitrary sets: when  $A$  is *finite*, we have  $\prod a \in A. P a = \prod a \in A. P a$ .

**lemma** *finite-GlobalNdet-is-MultiNdet*:

$$\langle \text{finite } A \implies (\prod p \in A. P p) = \prod p \in A. P p \rangle$$

*<proof>*

We obtain immediately the continuity when  $A$  is *finite* (and this is a necessary hypothesis for continuity).

**lemma** *GlobalNdet-cont[simp]*:

$$\langle \llbracket \text{finite } A; \forall x. \text{cont } (f x) \rrbracket \implies \text{cont } (\lambda y. (\prod z \in A. (f z y))) \rangle$$

*<proof>*

## 8.11 Link with *Mndetprefix*

This is a trick to make proof of *Mndetprefix* using *GlobalNdet* as it has an easier denotational definition.

**lemma** *Mndetprefix-GlobalNdet*:  $\langle \prod x \in A \rightarrow P x = \prod x \in A. (x \rightarrow P x) \rangle$

*<proof>*

**lemma** *write0-GlobalNdet*:

$$\langle A \neq \{\} \implies (\prod x \in A. (a \rightarrow P x)) = (a \rightarrow (\prod x \in A. P x)) \rangle$$

*<proof>*

## 8.12 Properties

**lemma** *GlobalNdet-Det*:

$$\langle A \neq \{\} \implies (\prod a \in A. P a) \square Q = \prod a \in A. P a \square Q \rangle$$

*<proof>*

**lemma** *Mndetprefix-STOP*:  $\langle A \subseteq C \implies (\prod a \in A \rightarrow P a) \llbracket C \rrbracket \text{STOP} = \text{STOP} \rangle$

*<proof>*

**lemma** *GlobalNdet-Sync-distr*:

$\langle A \neq \{\} \implies (\sqcap x \in A. P x) \llbracket C \rrbracket Q = \sqcap x \in A. (P x \llbracket C \rrbracket Q) \rangle$   
 ⟨proof⟩

**lemma** *Mndetprefix-Mprefix-Sync-distr:*

$\langle [A \subseteq B; B \subseteq C] \implies (\sqcap a \in A \rightarrow P a) \llbracket C \rrbracket (\sqcap b \in B \rightarrow Q b) =$   
 $\quad \sqcap a \in A \rightarrow (P a \llbracket C \rrbracket Q a) \rangle$

— does not hold in general when  $A \subseteq C$

⟨proof⟩

**corollary** *Mndetprefix-Mprefix-Par-distr:*

$\langle A \subseteq B \implies ((\sqcap a \in A \rightarrow P a) \parallel (\sqcap b \in B \rightarrow Q b)) = \sqcap a \in A \rightarrow P a \parallel Q a \rangle$

⟨proof⟩

**lemma** *Mndetprefix-Sync-Det-distr:*

$\langle (\sqcap a \in A \rightarrow (P a \llbracket C \rrbracket (\sqcap b \in B \rightarrow Q b))) \sqcap$

$(\sqcap b \in B \rightarrow ((\sqcap a \in A \rightarrow P a) \llbracket C \rrbracket Q b))$

$\sqsubseteq_{FD} (\sqcap a \in A \rightarrow P a) \llbracket C \rrbracket (\sqcap b \in B \rightarrow Q b) \rangle$

**if** *set-hyps* :  $\langle A \neq \{\} \rangle \langle B \neq \{\} \rangle \langle A \cap C = \{\} \rangle \langle B \cap C = \{\} \rangle$

— both surprising: equality does not hold + deterministic choice

⟨proof⟩

**lemma** *GlobalNdet-Mprefix-distr:*

$\langle A \neq \{\} \implies (\sqcap a \in A. \sqcap b \in B \rightarrow P a b) = \sqcap b \in B \rightarrow (\sqcap a \in A. P a b) \rangle$

⟨proof⟩

**lemma** *GlobalNdet-Det-distrib:*

$\langle (\sqcap a \in A. P a \sqcap Q a) = (\sqcap a \in A. P a) \sqcap (\sqcap a \in A. Q a) \rangle$

**if**  $\langle \exists Q' b. \forall a. Q a = (b \rightarrow Q' a) \rangle$

⟨proof⟩

**end**



# Chapter 9

## CSPM

**theory** *CSPM*

**imports** *MultiDet MultiNdet MultiSync MultiSeq GlobalNdet HOL-CSP.Assertions*  
**begin**

From the binary laws of HOL-CSP, we immediately obtain refinement results and lemmas about the combination of multi-operators.

### 9.1 Refinements Results

**lemma** *mono-MultiDet-F*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_F Q\ x \implies \text{MultiDet } A\ P \sqsubseteq_F \text{MultiDet } A\ Q \rangle$   
*<proof>*

**lemma** *mono-MultiDet-D[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_D Q\ x \implies \text{MultiDet } A\ P \sqsubseteq_D \text{MultiDet } A\ Q \rangle$

**and** *mono-MultiDet-T[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_T Q\ x \implies \text{MultiDet } A\ P \sqsubseteq_T \text{MultiDet } A\ Q \rangle$

**and** *mono-MultiDet-DT[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_{DT} Q\ x \implies \text{MultiDet } A\ P \sqsubseteq_{DT} \text{MultiDet } A\ Q \rangle$

**and** *mono-MultiDet-FD[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_{FD} Q\ x \implies \text{MultiDet } A\ P \sqsubseteq_{FD} \text{MultiDet } A\ Q \rangle$

*<proof>*

**lemma** *mono-MultiNdet-F[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_F Q\ x \implies \text{MultiNdet } A\ P \sqsubseteq_F \text{MultiNdet } A\ Q \rangle$

**and** *mono-MultiNdet-D[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_D Q\ x \implies \text{MultiNdet } A\ P \sqsubseteq_D \text{MultiNdet } A\ Q \rangle$

**and** *mono-MultiNdet-T[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_T Q\ x \implies \text{MultiNdet } A\ P \sqsubseteq_T \text{MultiNdet } A\ Q \rangle$

**and** *mono-MultiNdet-DT[simp, elim]*:

$\langle \text{finite } A \implies \forall x \in A. P\ x \sqsubseteq_{DT} Q\ x \implies \text{MultiNdet } A\ P \sqsubseteq_{DT} \text{MultiNdet } A\ Q \rangle$

**and** *mono-MultiNdet-FD[simp, elim]*:  
 $\langle \text{finite } A \implies \forall x \in A. P \sqsubseteq_{FD} Q \implies \text{MultiNdet } A \ P \sqsubseteq_{FD} \text{MultiNdet } A \ Q \rangle$   
*<proof>*

**lemma** *mono-MultiNdet-F-single*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies \forall a \in A. P \sqsubseteq_F Q \ a \implies P \sqsubseteq_F \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-D-single*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies \forall a \in A. P \sqsubseteq_D Q \ a \implies P \sqsubseteq_D \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-T-single*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies \forall a \in A. P \sqsubseteq_T Q \ a \implies P \sqsubseteq_T \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-DT-single*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies \forall a \in A. P \sqsubseteq_{DT} Q \ a \implies P \sqsubseteq_{DT} \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-FD-single*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies \forall a \in A. P \sqsubseteq_{FD} Q \ a \implies P \sqsubseteq_{FD} \text{MultiNdet } A \ Q \rangle$   
*<proof>*

**lemma**  
**assumes**  $\langle A \neq \{\} \rangle$  **and**  $\langle \text{finite } B \rangle$  **and**  $\langle A \subseteq B \rangle$   
**shows** *mono-MultiNdet-F-left-absorb-subset*:  
 $\langle \forall x \in A. P \ x \sqsubseteq_F Q \ x \implies \text{MultiNdet } B \ P \sqsubseteq_F \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-D-left-absorb-subset*:  
 $\langle \forall x \in A. P \ x \sqsubseteq_D Q \ x \implies \text{MultiNdet } B \ P \sqsubseteq_D \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-T-left-absorb-subset*:  
 $\langle \forall x \in A. P \ x \sqsubseteq_T Q \ x \implies \text{MultiNdet } B \ P \sqsubseteq_T \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-FD-left-absorb-subset*:  
 $\langle \forall x \in A. P \ x \sqsubseteq_{FD} Q \ x \implies \text{MultiNdet } B \ P \sqsubseteq_{FD} \text{MultiNdet } A \ Q \rangle$   
**and** *mono-MultiNdet-DT-left-absorb-subset*:  
 $\langle \forall x \in A. P \ x \sqsubseteq_{DT} Q \ x \implies \text{MultiNdet } B \ P \sqsubseteq_{DT} \text{MultiNdet } A \ Q \rangle$   
*<proof>*

**corollary** *mono-MultiNdet-F-left-absorb[simp]*:  
 $\langle \text{finite } A \implies x \in A \implies P \ x \sqsubseteq_F Q \implies \text{MultiNdet } A \ P \sqsubseteq_F Q \rangle$   
**and** *mono-MultiNdet-D-left-absorb[simp]*:  
 $\langle \text{finite } A \implies x \in A \implies P \ x \sqsubseteq_D Q \implies \text{MultiNdet } A \ P \sqsubseteq_D Q \rangle$   
**and** *mono-MultiNdet-T-left-absorb[simp]*:  
 $\langle \text{finite } A \implies x \in A \implies P \ x \sqsubseteq_T Q \implies \text{MultiNdet } A \ P \sqsubseteq_T Q \rangle$   
**and** *mono-MultiNdet-FD-left-absorb[simp]*:  
 $\langle \text{finite } A \implies x \in A \implies P \ x \sqsubseteq_{FD} Q \implies \text{MultiNdet } A \ P \sqsubseteq_{FD} Q \rangle$   
**and** *mono-MultiNdet-DT-left-absorb[simp]*:  
 $\langle \text{finite } A \implies x \in A \implies P \ x \sqsubseteq_{DT} Q \implies \text{MultiNdet } A \ P \sqsubseteq_{DT} Q \rangle$   
*<proof>*



**lemma** *mono-MultiNdet-MultiDet-F*[*simp, elim*]:  
 $\langle \text{finite } A \implies \text{MultiNdet } A \ P \sqsubseteq_F \text{MultiDet } A \ P \rangle$   
**and** *mono-MultiNdet-MultiDet-D*[*simp, elim*]:  
 $\langle \text{finite } A \implies \text{MultiNdet } A \ P \sqsubseteq_D \text{MultiDet } A \ P \rangle$   
**and** *mono-MultiNdet-MultiDet-T*[*simp, elim*]:  
 $\langle \text{finite } A \implies \text{MultiNdet } A \ P \sqsubseteq_T \text{MultiDet } A \ P \rangle$   
**and** *mono-MultiNdet-MultiDet-FD*[*simp, elim*]:  
 $\langle \text{finite } A \implies \text{MultiNdet } A \ P \sqsubseteq_{FD} \text{MultiDet } A \ P \rangle$   
**and** *mono-MultiNdet-MultiDet-DT*[*simp, elim*]:  
 $\langle \text{finite } A \implies \text{MultiNdet } A \ P \sqsubseteq_{DT} \text{MultiDet } A \ P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSync-F*:  $\langle \forall x \in \# \ M. \ P \ x \sqsubseteq_F \ Q \ x \implies \text{MultiSync } S \ M \ P \sqsubseteq_F \text{MultiSync } S \ M \ Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSync-D*:  $\langle \forall x \in \# \ M. \ P \ x \sqsubseteq_D \ Q \ x \implies \text{MultiSync } S \ M \ P \sqsubseteq_D \text{MultiSync } S \ M \ Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSync-T*:  $\langle \forall x \in \# \ M. \ P \ x \sqsubseteq_T \ Q \ x \implies \text{MultiSync } S \ M \ P \sqsubseteq_T \text{MultiSync } S \ M \ Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSync-DT*[*simp, elim*]:  
 $\langle \forall x \in \# \ M. \ P \ x \sqsubseteq_{DT} \ Q \ x \implies \text{MultiSync } S \ M \ P \sqsubseteq_{DT} \text{MultiSync } S \ M \ Q \rangle$   
**and** *mono-MultiSync-FD*[*simp, elim*]:  
 $\langle \forall x \in \# \ M. \ P \ x \sqsubseteq_{FD} \ Q \ x \implies \text{MultiSync } S \ M \ P \sqsubseteq_{FD} \text{MultiSync } S \ M \ Q \rangle$   
 $\langle \text{proof} \rangle$

**find-theorems** *name: mset name: ind*

**lemmas** *mono-MultiInter-DT*[*simp, elim*] = *mono-MultiSync-DT*[**where**  $S = \langle \{ \} \rangle$ ]  
**and** *mono-MultiInter-FD*[*simp, elim*] = *mono-MultiSync-FD*[**where**  $S = \langle \{ \} \rangle$ ]  
**and** *mono-MultiPar-DT*[*simp, elim*] = *mono-MultiSync-DT*[**where**  $S = \langle \text{UNIV} \rangle$ ]  
**and** *mono-MultiPar-FD*[*simp, elim*] = *mono-MultiSync-FD*[**where**  $S = \langle \text{UNIV} \rangle$ ]

**lemma** *mono-MultiSeq-F*:  
 $\langle \forall x \in \text{set } L. \ P \ x \sqsubseteq_F \ Q \ x \implies \text{MultiSeq } L \ P \sqsubseteq_F \text{MultiSeq } L \ Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSeq-D*:  
 $\langle \forall x \in \text{set } L. \ P \ x \sqsubseteq_D \ Q \ x \implies \text{MultiSeq } L \ P \sqsubseteq_D \text{MultiSeq } L \ Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSeq-T*:

$\langle \forall x \in \text{set } L. P x \sqsubseteq_T Q x \implies \text{MultiSeq } L P \sqsubseteq_T \text{MultiSeq } L Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-MultiSeq-DT[simp, elim]*:

$\langle \forall x \in \text{set } L. P x \sqsubseteq_{DT} Q x \implies \text{MultiSeq } L P \sqsubseteq_{DT} \text{MultiSeq } L Q \rangle$   
**and** *mono-MultiSeq-FD[simp, elim]*:  
 $\langle \forall x \in \text{set } L. P x \sqsubseteq_{FD} Q x \implies \text{MultiSeq } L P \sqsubseteq_{FD} \text{MultiSeq } L Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-GlobalNdet[simp]* :  $\langle \text{GlobalNdet } A P \sqsubseteq \text{GlobalNdet } A Q \rangle$

**if**  $\langle \forall x \in A. P x \sqsubseteq Q x \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-GlobalNdet-F[simp, elim]*:

$\langle \forall x \in A. P x \sqsubseteq_F Q x \implies \text{GlobalNdet } A P \sqsubseteq_F \text{GlobalNdet } A Q \rangle$   
**and** *mono-GlobalNdet-D[simp, elim]*:  
 $\langle \forall x \in A. P x \sqsubseteq_D Q x \implies \text{GlobalNdet } A P \sqsubseteq_D \text{GlobalNdet } A Q \rangle$   
**and** *mono-GlobalNdet-T[simp, elim]*:  
 $\langle \forall x \in A. P x \sqsubseteq_T Q x \implies \text{GlobalNdet } A P \sqsubseteq_T \text{GlobalNdet } A Q \rangle$   
**and** *mono-GlobalNdet-DT[simp, elim]*:  
 $\langle \forall x \in A. P x \sqsubseteq_{DT} Q x \implies \text{GlobalNdet } A P \sqsubseteq_{DT} \text{GlobalNdet } A Q \rangle$   
**and** *mono-GlobalNdet-FD[simp, elim]*:  
 $\langle \forall x \in A. P x \sqsubseteq_{FD} Q x \implies \text{GlobalNdet } A P \sqsubseteq_{FD} \text{GlobalNdet } A Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *GlobalNdet-refine-FD-subset*:

$\langle A \neq \{\} \implies A \subseteq B \implies \text{GlobalNdet } B P \sqsubseteq_{FD} \text{GlobalNdet } A P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *GlobalNdet-refine-F-subset*:

$\langle A \neq \{\} \implies A \subseteq B \implies \text{GlobalNdet } B P \sqsubseteq_F \text{GlobalNdet } A P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *GlobalNdet-refine-FD*:  $\langle a \in A \implies \text{GlobalNdet } A P \sqsubseteq_{FD} P a \rangle$

$\langle \text{proof} \rangle$

**lemma** *GlobalNdet-refine-F*:  $\langle a \in A \implies \text{GlobalNdet } A P \sqsubseteq_F P a \rangle$

$\langle \text{proof} \rangle$

**lemma** *mono-GlobalNdet-FD-const*:

$\langle A \neq \{\} \implies \forall x \in A. P \sqsubseteq_{FD} Q x \implies P \sqsubseteq_{FD} \text{GlobalNdet } A Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mono-GlobalNdet-F-const*:

$\langle A \neq \{\} \implies \forall x \in A. P \sqsubseteq_F Q \ x \implies P \sqsubseteq_F \text{GlobalNdet } A \ Q \rangle$   
 $\langle \text{proof} \rangle$

## 9.2 Combination of Multi-Operators Laws

**lemma** *finite-Mprefix-is-MultiDet:*

$\langle \text{finite } A \implies (\Box p \in A \rightarrow P \ p) = \Box p \in A. (p \rightarrow P \ p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *finite-Mndetprefix-is-MultiNdet:*

$\langle \text{finite } A \implies \text{Mndetprefix } A \ P = \prod p \in A. (p \rightarrow P \ p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\langle Q \Box (\prod p \in \{\}. P \ p) = \prod p \in \{\}. (Q \Box P \ p) \implies Q = \text{STOP} \rangle$

$\langle \text{proof} \rangle$

**lemma** *Det-MultiNdet-distrib:*

$\langle A \neq \{\} \implies \text{finite } A \implies M \Box (\prod p \in A. P \ p) = \prod p \in A. M \Box P \ p \rangle$   
 $\langle \text{proof} \rangle$

**lemma**  $\langle M \Box (\Box p \in \{\}. P \ p) = \Box p \in \{\}. (M \Box P \ p) \implies M \Box \text{STOP} = \text{STOP} \rangle$

$\langle \text{proof} \rangle$

**lemma** *Ndet-MultiDet-distrib:*

$\langle A \neq \{\} \implies \text{finite } A \implies M \Box (\Box p \in A. P \ p) = \Box p \in A. M \Box P \ p \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiNdet-Sync-left-distrib:*

$\langle B \neq \{\} \implies \text{finite } B \implies (\prod a \in B. P \ a) \llbracket S \rrbracket M = \prod a \in B. (P \ a \llbracket S \rrbracket M) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiNdet-Sync-right-distrib:*

$\langle B \neq \{\} \implies \text{finite } B \implies M \llbracket S \rrbracket \text{MultiNdet } B \ P = \prod a \in B. (M \llbracket S \rrbracket P \ a) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Sync-MultiNdet-cartprod:*

$\langle A \neq \{\} \implies \text{finite } A \implies B \neq \{\} \implies \text{finite } B \implies$   
 $(\prod (s, t) \in A \times B. (x \ s \llbracket S \rrbracket y \ t)) = (\prod s \in A. x \ s) \llbracket S \rrbracket (\prod t \in B. y \ t) \rangle$   
 $\langle \text{proof} \rangle$

**lemmas** *Inter-MultiNdet-cartprod* = *Sync-MultiNdet-cartprod*[**where**  $S = \langle \{\} \rangle$ ]  
**and** *Par-MultiNdet-cartprod* = *Sync-MultiNdet-cartprod*[**where**  $S = UNIV$ ]

**lemmas** *MultiNdet-Inter-left-distrib* =  
*MultiNdet-Sync-left-distrib*[**where**  $S = \langle \{\} \rangle$ ]  
**and** *MultiNdet-Par-left-distrib* =  
*MultiNdet-Sync-left-distrib*[**where**  $S = \langle UNIV \rangle$ ]

**lemma** *Seq-MultiNdet-distribR*:  
 $\langle \text{finite } A \implies (\prod p \in A. P p) ; S = (\prod p \in A. (P p ; S)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Seq-MultiNdet-distribL*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies S ; (\prod p \in A. P p) = (\prod p \in A. (S ; P p)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *prefix-MultiNdet-distrib*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies (a \rightarrow (\prod p \in A. P p)) = \prod p \in A. (a \rightarrow P p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Mndetprefix-MultiNdet-distrib*:  
 $\langle (\prod q \in B \rightarrow (\prod p \in A. P p q)) = \prod p \in A. \prod q \in B \rightarrow P p q \rangle$   
**if** *finB*:  $\langle \text{finite } B \rangle$  **and** *nonemptyA*:  $\langle A \neq \{\} \rangle$  **and** *finA*:  $\langle \text{finite } A \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-Mprefix*:  
 $\langle \text{finite } A \implies (\prod a \in A. \prod x \in S a \rightarrow P a x) =$   
 $\quad \prod x \in (\bigcup a \in A. S a) \rightarrow \prod a \in \{a \in A. x \in S a\}. P a x \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiDet-prefix-is-MultiNdet-prefix*:  
 $\langle \text{finite } A \implies (\prod p \in A. (a \rightarrow P p)) = \prod p \in A. (a \rightarrow P p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *prefix-MultiNdet-is-MultiDet-prefix*:  
 $\langle A \neq \{\} \implies \text{finite } A \implies (a \rightarrow (\prod p \in A. P p)) = \prod p \in A. (a \rightarrow P p) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Mprefix-MultiNdet-distrib'*:

$\langle \text{finite } B \implies A \neq \{\} \implies \text{finite } A \implies$   
 $(\Box q \in B \rightarrow \prod p \in A. P p q) = \Box p \in A. \Box q \in B \rightarrow P p q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiSync-Hiding-pseudo-distrib*:

$\langle \text{finite } B \implies B \cap S = \{\} \implies$   
 $(\llbracket S \rrbracket p \in \# M. ((P p) \setminus B)) = (\llbracket S \rrbracket p \in \# M. P p) \setminus B \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiSync-prefix-pseudo-distrib*:

$\langle M \neq \{\# \} \implies a \in S \implies$   
 $(\llbracket S \rrbracket p \in \# M. (a \rightarrow P p)) = (a \rightarrow (\llbracket S \rrbracket p \in \# M. P p)) \rangle$   
 $\langle \text{proof} \rangle$

**lemmas** *MultiInter-Hiding-pseudo-distrib =*

*MultiSync-Hiding-pseudo-distrib*[**where**  $S = \langle \{\} \rangle$ , *simplified*]

**and** *MultiPar-prefix-pseudo-distrib =*

*MultiSync-prefix-pseudo-distrib*[**where**  $S = \langle UNIV \rangle$ , *simplified*]

**lemma** *Hiding-MultiNdet-distrib*:

$\langle \text{finite } A \implies (\prod p \in A. P p) \setminus B = (\prod p \in A. (P p \setminus B)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Mndetprefix-Hiding-is-MultiNdet-prefix-Hiding*:

$\langle \text{finite } A \implies \prod p \in A - B \rightarrow ((P p) \setminus B) = \prod p \in A - B. (p \rightarrow ((P p) \setminus B)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Hiding-Mndetprefix-is-MultiNdet-Hiding*:

$\langle \text{finite } A \implies A \subseteq B \implies (\prod a \in A \rightarrow P) \setminus B = \prod a \in A. (P \setminus B) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *MultiSync-Mprefix-pseudo-distrib*:

$\langle (\llbracket S \rrbracket B \in \# M. \Box x \in B \rightarrow P B x) =$   
 $\Box x \in (\prod B \in \text{set-mset } M. B) \rightarrow (\llbracket S \rrbracket B \in \# M. P B x) \rangle$   
**if** *nonempty*:  $\langle M \neq \{\# \} \rangle$  **and** *hyp*:  $\langle \forall B \in \# M. B \subseteq S \rangle$   
 $\langle \text{proof} \rangle$

**lemmas** *MultiPar-Mprefix-pseudo-distrib =*

*MultiSync-Mprefix-pseudo-distrib*[**where**  $S = \langle UNIV \rangle$ , *simplified*]

A result on Mndetprefix and Sync.

**lemma** *Mndetprefix-Sync-distr*:  $\langle A \neq \{\} \implies B \neq \{\} \implies$   
 $(\sqcap a \in A \rightarrow P a) \llbracket S \rrbracket (\sqcap b \in B \rightarrow Q b) =$   
 $\sqcap a \in A. \sqcap b \in B. (\sqcap c \in \{a\} - S \rightarrow (P a \llbracket S \rrbracket (b \rightarrow Q b))) \sqcap$   
 $(\sqcap d \in \{b\} - S \rightarrow ((a \rightarrow P a) \llbracket S \rrbracket Q b)) \sqcap$   
 $(\sqcap c \in \{a\} \cap \{b\} \cap S \rightarrow (P a \llbracket S \rrbracket Q b)) \rangle$   
*<proof>*

A result on *MultiSeq* with *non-terminating*.

**lemma** *non-terminating-MultiSeq*:  
 $\langle (SEQ a \in @ L. P a) =$   
 $SEQ a \in @ take (Suc (first-elem (\lambda a. non-terminating (P a)) L)) L. P a \rangle$   
*<proof>*

### 9.3 Results on Renaming

**lemma** *Renaming-GlobalNdet*:  
 $\langle Renaming (\sqcap a \in A. P (f a)) f = \sqcap b \in f ' A. Renaming (P b) f \rangle$   
*<proof>*

**lemma** *Renaming-GlobalNdet-inj-on*:  
 $\langle Renaming (\sqcap a \in A. P a) f =$   
 $\sqcap b \in f ' A. Renaming (P (THE a. a \in A \wedge f a = b)) f \rangle$   
**if** *inj-on-f*:  $\langle inj-on f A \rangle$   
*<proof>*

**corollary** *Renaming-GlobalNdet-inj*:  
 $\langle Renaming (\sqcap a \in A. P a) f =$   
 $\sqcap b \in f ' A. Renaming (P (THE a. f a = b)) f \rangle$  **if** *inj-f*:  $\langle inj f \rangle$   
*<proof>*

**lemma** *Renaming-MultiNdet*:  $\langle finite A \implies Renaming (\sqcap a \in A. P (f a)) f =$   
 $\sqcap b \in f ' A. Renaming (P b) f \rangle$   
*<proof>*

**lemma** *Renaming-MultiNdet-inj-on*:  
 $\langle finite A \implies inj-on f A \implies$   
 $Renaming (\sqcap a \in A. P a) f =$   
 $\sqcap b \in f ' A. Renaming (P (THE a. a \in A \wedge f a = b)) f \rangle$   
*<proof>*

**corollary** *Renaming-MultiNdet-inj*:

$\langle \text{finite } A \implies \text{inj } f \implies$   
 $\text{Renaming } (\prod a \in A. P a) f = \prod b \in f ' A. \text{Renaming } (P (\text{THE } a. f a = b)) f \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Renaming-MultiDet:*

$\langle \text{finite } A \implies \text{Renaming } (\prod a \in A. P (f a)) f =$   
 $\quad \prod b \in f ' A. \text{Renaming } (P b) f \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Renaming-MultiDet-inj-on:*

$\langle \text{Renaming } (\prod a \in A. P a) f =$   
 $\quad \prod b \in f ' A. \text{Renaming } (P (\text{THE } a. a \in A \wedge f a = b)) f \rangle$   
**if** *finite-A:*  $\langle \text{finite } A \rangle$  **and** *inj-on-f:*  $\langle \text{inj-on } f A \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *Renaming-MultiDet-inj:*

$\langle \text{Renaming } (\prod a \in A. P a) f = \prod b \in f ' A. \text{Renaming } (P (\text{THE } a. f a = b)) f \rangle$   
**if** *finite-A:*  $\langle \text{finite } A \rangle$  **and** *inj-f:*  $\langle \text{inj } f \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Renaming-MultiSeq:*

$\langle \text{Renaming } (\text{SEQ } l \in @ L. P (f l)) f = \text{SEQ } m \in @ \text{map } f L. \text{Renaming } (P m) f \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *Renaming-MultiSeq-inj-on:*

$\langle \text{Renaming } (\text{SEQ } l \in @ L. P l) f =$   
 $\quad \text{SEQ } m \in @ \text{map } f L. \text{Renaming } (P (\text{THE } l. l \in \text{set } L \wedge f l = m)) f \rangle$   
**if** *inj-on-f:*  $\langle \text{inj-on } f (\text{set } L) \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *Renaming-MultiSeq-inj:*

$\langle \text{Renaming } (\text{SEQ } l \in @ L. P l) f =$   
 $\quad \text{SEQ } m \in @ \text{map } f L. \text{Renaming } (P (\text{THE } l. f l = m)) f \rangle$  **if** *inj-f:*  $\langle \text{inj } f \rangle$   
 $\langle \text{proof} \rangle$

**end**





## Chapter 10

# Example: Dining Philosophers

```
theory DiningPhilosophers
  imports CSPM
begin
```

### 10.1 Classic Version

We formalize here the Dining Philosophers problem with a locale.

```
locale DiningPhilosophers =
```

```
  fixes  $N::nat$ 
```

```
  assumes  $N-g1[simp] : \langle N > 1 \rangle$ 
```

— We assume that we have at least one right handed philosophers (so at least two philosophers with the left handed one).

```
begin
```

We use a datatype for representing the dinner's events.

```
datatype dining-event = picks (phil:nat) (fork:nat)
  | putsdown (phil:nat) (fork:nat)
```

We introduce the right handed philosophers, the left handed philosopher and the forks.

```
definition RPHIL::  $\langle nat \Rightarrow dining-event process \rangle$ 
```

```
  where  $\langle RPHIL\ i \equiv \mu X. (picks\ i\ i \rightarrow (picks\ i\ ((i-1)\ mod\ N) \rightarrow$   
     $(putsdown\ i\ ((i-1)\ mod\ N) \rightarrow (putsdown\ i\ i \rightarrow X)))) \rangle$ 
```

```
definition LPHILO::  $\langle dining-event process \rangle$ 
```

```
  where  $\langle LPHILO \equiv \mu X. (picks\ 0\ (N-1) \rightarrow (picks\ 0\ 0 \rightarrow$   
     $(putsdown\ 0\ 0 \rightarrow (putsdown\ 0\ (N-1) \rightarrow X)))) \rangle$ 
```

**definition** *FORK* ::  $\langle \text{nat} \Rightarrow \text{dining-event process} \rangle$   
**where**  $\langle \text{FORK } i \equiv \mu X. (\text{picks } i \ i \rightarrow (\text{putsdown } i \ i \rightarrow X)) \square$   
 $(\text{picks } ((i+1) \ \text{mod } N) \ i \rightarrow (\text{putsdown } ((i+1) \ \text{mod } N) \ i \rightarrow X)) \rangle$

Now we use the architectural operators for modelling the interleaving of the philosophers, and the interleaving of the forks.

**definition**  $\langle \text{PHILS} \equiv ||| P \in\# \text{ add-mset } \text{LPHILO} \ (\text{mset } (\text{map } \text{RPHIL} \ [1..< N])) . P \rangle$

**definition**  $\langle \text{FORKS} \equiv ||| P \in\# \text{ mset } (\text{map } \text{FORK} \ [0..< N]) . P \rangle$

**corollary**  $\langle N = 3 \implies \text{PHILS} = (\text{LPHILO} \ ||| \ \text{RPHIL } 1 \ ||| \ \text{RPHIL } 2) \rangle$   
— just a test  
 $\langle \text{proof} \rangle$

Finally, the dinner is obtained by putting forks and philosophers in parallel.

**definition** *DINING* ::  $\langle \text{dining-event process} \rangle$   
**where**  $\langle \text{DINING} = (\text{FORKS} \ || \ \text{PHILS}) \rangle$

**end**

## 10.2 Formalization with fixrec Package

The **fixrec** package of HOLCF provides a more readable syntax (essentially, it allows us to "get rid of  $\mu$ " in equations like  $\mu x. P x$ ).

First, we need to see *nat* as *cpo*.

**instantiation** *nat* :: *discrete-cpo*  
**begin**

**definition** *below-nat-def*:  
 $(x::\text{nat}) \sqsubseteq y \longleftrightarrow x = y$

**instance**  $\langle \text{proof} \rangle$

**end**

**locale** *DiningPhilosophers-fixrec* =

**fixes**  $N::\text{nat}$

**assumes**  $N\text{-g1}[\text{simp}] : \langle N > 1 \rangle$

— We assume that we have at least one right handed philosophers (so at least two philosophers with the left handed one).

**begin**

We use a datatype for representing the dinner's events.

**datatype** *dining-event* = *picks* (*phil:nat*) (*fork:nat*)  
 | *putsdown* (*phil:nat*) (*fork:nat*)

We introduce the right handed philosophers, the left handed philosopher and the forks.

**fixrec** *RPHIL* ::  $\langle \text{nat} \rightarrow \text{dining-event process} \rangle$   
**and** *LPHILO* ::  $\langle \text{dining-event process} \rangle$   
**and** *FORK* ::  $\langle \text{nat} \rightarrow \text{dining-event process} \rangle$

**where**

*RPHIL-rec* [*simp del*] :  
 $\langle \text{RPHIL} \cdot i = (\text{picks } i \ i \rightarrow (\text{picks } i \ (i-1) \rightarrow$   
 $(\text{putsdown } i \ (i-1) \rightarrow (\text{putsdown } i \ i \rightarrow \text{RPHIL} \cdot i))) \rangle$   
 | *LPHILO-rec*[*simp del*] :  
 $\langle \text{LPHILO} = (\text{picks } 0 \ (N-1) \rightarrow (\text{picks } 0 \ 0 \rightarrow$   
 $(\text{putsdown } 0 \ 0 \rightarrow (\text{putsdown } 0 \ (N-1) \rightarrow \text{LPHILO}))) \rangle$   
 | *FORK-rec* [*simp del*] :  
 $\langle \text{FORK} \cdot i = (\text{picks } i \ i \rightarrow (\text{putsdown } i \ i \rightarrow \text{FORK} \cdot i)) \square$   
 $(\text{picks } ((i+1) \ \text{mod } N) \ i \rightarrow (\text{putsdown } ((i+1) \ \text{mod } N) \ i \rightarrow \text{FORK} \cdot i)) \rangle$

Now we use the architectural operators for modelling the interleaving of the philosophers, and the interleaving of the forks.

**definition**  $\langle \text{PHILS} \equiv ||| \ P \in\# \ \text{add-mset } \text{LPHILO} \ (\text{mset } (\text{map } (\lambda i. \ \text{RPHIL} \cdot i) [1..<N])) \ . \ P \rangle$

**definition**  $\langle \text{FORKS} \equiv ||| \ P \in\# \ \text{mset } (\text{map } (\lambda i. \ \text{FORK} \cdot i) [0..<N]) \ . \ P \rangle$

**corollary**  $\langle N = 3 \implies \text{PHILS} = (\text{LPHILO} \ ||| \ \text{RPHIL} \cdot 1 \ ||| \ \text{RPHIL} \cdot 2) \rangle$   
 — just a test  
 $\langle \text{proof} \rangle$

Finally, the dinner is obtained by putting forks and philosophers in parallel.

**definition** *DINING* ::  $\langle \text{dining-event process} \rangle$   
**where**  $\langle \text{DINING} = (\text{FORKS} \ ||| \ \text{PHILS}) \rangle$

**end**

**end**



## Chapter 11

# Example: Plain Old Telephone System

The "Plain Old Telephone Service is a standard medium-size example for architectural modeling of a concurrent system.

Plain old telephone service (POTS), or plain ordinary telephone system,[1] is a retronym for voice-grade telephone service employing analog signal transmission over copper loops. POTS was the standard service offering from telephone companies from 1876 until 1988[2] in the United States when the Integrated Services Digital Network (ISDN) Basic Rate Interface (BRI) was introduced, followed by cellular telephone systems, and voice over IP (VoIP). POTS remains the basic form of residential and small business service connection to the telephone network in many parts of the world. The term reflects the technology that has been available since the introduction of the public telephone system in the late 19th century, in a form mostly unchanged despite the introduction of Touch-Tone dialing, electronic telephone exchanges and fiber-optic communication into the public switched telephone network (PSTN).

C.f. wikipedia [https://en.wikipedia.org/wiki/Plain\\_old\\_telephone\\_service](https://en.wikipedia.org/wiki/Plain_old_telephone_service).

```
theory POTS
  imports CSPM
begin
```

We need to see *int* as a *cpo*.

```
instantiation int :: discrete-cpo
begin
```

```
definition below-int-def:
  (x::int)  $\sqsubseteq$  y  $\longleftrightarrow$  x = y
```

```
instance  $\langle$ proof $\rangle$ 
```

end

## 11.1 The Alphabet and Basic Types of POTS

Underlying terminology apparent in the acronyms:

1. T-side (target side, callee side)
2. O-side (originator (?) side, caller side)

```
datatype MtcO = Osetup | Odiscon-o  
datatype MctO = Obusy | Oalert | Oconnect | Odiscon-t  
datatype MtcT = Tbusy | Talert | Tconnect | Tdiscon-t  
datatype MctT = Tsetup | Tdiscon-o
```

```
type-synonym Phones =  $\langle int \rangle$ 
```

```
datatype channels = tcO  $\langle Phones \times MtcO \rangle$  —  
| ctO  $\langle Phones \times MctO \rangle$   
| tcT  $\langle Phones \times MtcT \times Phones \rangle$   
| ctT  $\langle Phones \times MctT \times Phones \rangle$   
| tcOdial  $\langle Phones \times Phones \rangle$   
| StartReject Phones — phone x rejects from now on to be  
called  
| EndReject Phones — phone x accepts from now on to be  
called  
| terminal Phones  
| off-hook Phones  
| on-hook Phones  
| digits  $\langle Phones \times Phones \rangle$  — communication relation: x calls y  
  
| tone-ring Phones  
| tone-quiet Phones  
| tone-busy Phones  
| tone-dial Phones  
| connected Phones
```

```
locale POTS =  
  fixes min-phones :: int  
  and max-phones :: int  
  and VisibleEvents ::  $\langle channels \text{ set} \rangle$   
  assumes min-phones-g-1[simp] :  $\langle 1 \leq min-phones \rangle$   
  and max-phones-g-min-phones[simp] :  $\langle min-phones < max-phones \rangle$   
begin
```

```
definition phones ::  $\langle Phones \text{ set} \rangle$  where  $\langle phones \equiv \{min-phones .. max-phones\} \rangle$ 
```

**lemma** *nonempty-phones*[simp]:  $\langle \text{phones} \neq \{\} \rangle$   
**and** *finite-phones*[simp]:  $\langle \text{finite phones} \rangle$   
**and** *at-least-two-phones*[simp]:  $\langle 2 \leq \text{card phones} \rangle$   
**and** *not-singl-phone*[simp]:  $\langle \text{phones} - \{p\} \neq \{\} \rangle$   
 $\langle \text{proof} \rangle$

**definition** *EventsIPhone* ::  $\langle \text{Phones} \Rightarrow \text{channels set} \rangle$   
**where**  $\langle \text{EventsIPhone } u \equiv \{ \text{tone-ring } u, \text{tone-quiet } u, \text{tone-busy } u, \text{tone-dial } u, \text{connected } u \} \rangle$   
**definition** *EventsUser* ::  $\langle \text{Phones} \Rightarrow \text{channels set} \rangle$   
**where**  $\langle \text{EventsUser } u \equiv \{ \text{off-hook } u, \text{on-hook } u \} \cup \{ x . \exists n. x = \text{digits } (u, n) \} \rangle$

## 11.2 Auxilliaris to Substructure the Specification

**abbreviation** *Sliding* ::  $\langle ' \alpha \text{ process} \Rightarrow ' \alpha \text{ process} \Rightarrow ' \alpha \text{ process} \rangle$  (**infixl**  $\langle \triangleright \rangle$  78)  
**where**  $\langle P \triangleright Q \equiv (P \square Q) \square Q \rangle$   
— This operator is also called Timeout, more studied in future theories.

**abbreviation**  
*Tside-connected* ::  $\langle \text{Phones} \Rightarrow \text{Phones} \Rightarrow \text{channels process} \rangle$   
**where**  $\langle \text{Tside-connected } ts \ os \equiv$   
 $(ctT!(ts, Tdiscon-o, os) \rightarrow tcT!(ts, Tdiscon-t, os) \rightarrow \text{EndReject!}ts \rightarrow \text{SKIP})$   
 $\triangleright (tcT!(ts, Tdiscon-t, os) \rightarrow ctT!(ts, Tdiscon-o, os) \rightarrow \text{EndReject!}ts \rightarrow \text{SKIP}) \rangle$

**abbreviation**  
*Oside-connected* ::  $\langle \text{Phones} \Rightarrow \text{channels process} \rangle$   
**where**  $\langle \text{Oside-connected } ts \equiv$   
 $(ctO!(ts, Odiscon-t) \rightarrow tcO!(ts, Odiscon-o) \rightarrow \text{EndReject!}ts \rightarrow \text{SKIP})$   
 $\triangleright (tcO!(ts, Odiscon-o) \rightarrow ctO!(ts, Odiscon-t) \rightarrow \text{EndReject!}ts \rightarrow \text{SKIP}) \rangle$

**abbreviation**  
*Oside1* ::  $\langle [\text{Phones}, \text{Phones}] \Rightarrow \text{channels process} \rangle$   
**where**  
 $\langle \text{Oside1 } ts \ p \equiv tcO\text{dial!}(ts, p)$   
 $\rightarrow (ctO!(ts, Oalert)$   
 $\rightarrow ctO!(ts, Oconnect)$   
 $\rightarrow (\text{Oside-connected } ts))$   
 $\square (ctO!(ts, Oconnect) \rightarrow (\text{Oside-connected } ts))$   
 $\square (ctO!(ts, Obusy) \rightarrow tcO!(ts, Odiscon-o) \rightarrow \text{EndReject!}ts \rightarrow$   
 $\text{SKIP}) \rangle$

**definition**

$$ITside-connected \quad :: \langle [Phones, Phones, channels process] \Rightarrow channels process \rangle$$
**where**

$$\begin{aligned} \langle ITside-connected \ ts \ os \ IT \equiv & (ctT(ts, Tdiscon-o, os) \\ & \rightarrow ( \text{tone-busy!}ts \\ & \quad \rightarrow \text{on-hook!}ts \\ & \quad \rightarrow tcT!(ts, Tdiscon-t, os) \\ & \quad \rightarrow \text{EndReject!}ts \\ & \quad \rightarrow IT) \\ & \square (\text{on-hook!}ts \\ & \quad \rightarrow tcT!(ts, Tdiscon-t, os) \\ & \quad \rightarrow \text{EndReject!}ts \\ & \quad \rightarrow IT) \\ & )) \\ & \square (\text{on-hook!}ts \\ & \quad \rightarrow tcT!(ts, Tdiscon-t, os) \\ & \quad \rightarrow ctT!(ts, Tdiscon-o, os) \\ & \quad \rightarrow \text{EndReject!}ts \\ & \quad \rightarrow IT) \rangle \end{aligned}$$

### 11.3 A Telephone

**fixrec**  $T \quad :: \langle Phones \rightarrow channels process \rangle$   
**and**  $Oside \quad :: \langle Phones \rightarrow channels process \rangle$   
**and**  $Tside \quad :: \langle Phones \rightarrow channels process \rangle$   
**and**  $NoReject \quad :: \langle Phones \rightarrow channels process \rangle$   
**and**  $Reject \quad :: \langle Phones \rightarrow channels process \rangle$

**where**

$$\begin{aligned} T-rec \quad [simp \ del]: \langle T \cdot ts & = (Tside \cdot ts ; T \cdot ts) \triangleright (Oside \cdot ts ; T \cdot ts) \rangle \\ | \ Oside-rec \quad [simp \ del]: \langle Oside \cdot ts & = \text{StartReject!}ts \\ & \rightarrow tcO!(ts, Osetup) \\ & \rightarrow (\prod p \in phones. \ Oside1 \ ts \ p) \rangle \\ | \ Tside-rec \quad [simp \ del]: \langle Tside \cdot ts & = ctT?(y, z, os) | ((y, z) = (ts, Tsetup)) \\ & \rightarrow \text{StartReject!}ts \\ & \rightarrow ( \quad tcT!(ts, Talert, os) \\ & \quad \rightarrow tcT!(ts, Tconnect, os) \\ & \quad \rightarrow (Tside-connected \ ts \ os) \\ & \quad \square (tcT!(ts, Tconnect, os) \\ & \quad \rightarrow (Tside-connected \ ts \ os))) \rangle \\ | \ NoReject-rec \ [simp \ del]: \langle NoReject \cdot ts & = \text{StartReject!}ts \rightarrow \text{Reject} \cdot ts \rangle \\ | \ Reject-rec \quad [simp \ del]: \langle Reject \cdot ts & = ctT?(y, z, os) | (y=ts \wedge z=Tsetup \wedge os \in phones \\ \wedge \ os \neq ts) \\ & \rightarrow (tcT!(ts, Tbusy, os) \rightarrow \text{Reject} \cdot ts) \\ & \quad \square (\text{EndReject!}ts \rightarrow \text{NoReject} \cdot ts) \rangle \end{aligned}$$



**definition**  $Tel:: \langle Phones \Rightarrow channels\ process \rangle$   
**where**  $\langle Tel\ p \equiv (T \cdot p \llbracket \{StartReject\ p, EndReject\ p\} \rrbracket NoReject \cdot p) \setminus \{StartReject\ p, EndReject\ p\} \rangle$

## 11.4 A Connector with the Network

**fixrec**  $Call \quad :: \langle Phones \rightarrow channels\ process \rangle$   
**and**  $BUSY \quad :: \langle Phones \rightarrow Phones \rightarrow channels\ process \rangle$   
**and**  $Connected \quad :: \langle Phones \rightarrow Phones \rightarrow channels\ process \rangle$   
**where**  
 $Call\text{-}rec \ [simp\ del]: \langle Call \cdot os = (tcO! (os, Osetup) \rightarrow tcOdial?(x,ts)|(x=os) \rightarrow (BUSY \cdot os \cdot ts)) ; Call \cdot os \rangle$   
 $| \ BUSY\text{-}rec \ [simp\ del]: \langle BUSY \cdot os \cdot ts = (if\ ts = os \ then\ ctO!(os, Obusy) \rightarrow tcO!(os, Odiscon-o) \rightarrow SKIP \ else\ ctT!(ts, Tsetup, os) \rightarrow ( (tcT!(ts, Tbusy, os) \rightarrow ctO!(os, Obusy) \rightarrow tcO!(os, Odiscon-o) \rightarrow SKIP) \square (tcT!(ts, Talert, os) \rightarrow ctO!(os, Oalert) \rightarrow tcT!(ts, Tconnect, os) \rightarrow ctO!(os, Oconnect) \rightarrow Connected \cdot os \cdot ts) \square (tcT!(ts, Tconnect, os) \rightarrow ctO!(os, Oconnect) \rightarrow Connected \cdot os \cdot ts))) \rangle$   
 $| \ Connected\text{-}rec \ [simp\ del]: \langle Connected \cdot os \cdot ts = (tcO!(os, Odiscon-o) \rightarrow (( (ctT!(ts, Tdiscon-o, os) \rightarrow tcT!(ts, Tdiscon-t, os) \rightarrow SKIP) \square (tcT!(ts, Tdiscon-t, os) \rightarrow ctT!(ts, Tdiscon-o, os) \rightarrow SKIP) ) ; (ctO!(os, Odiscon-t) \rightarrow SKIP))) \square (tcT!(ts, Tdiscon-t, os) \rightarrow ( (ctO!(os, Odiscon-t) \rightarrow ctT!(ts, Tdiscon-o, os) \rightarrow tcO!(os, Odiscon-o) \rightarrow SKIP) ) \square (tcO!(os, Odiscon-o) \rightarrow ctT!(ts, Tdiscon-o, os) \rightarrow ctO!(os, Odiscon-t) \rightarrow SKIP) ) \rangle$

## 11.5 Combining NETWORK and TELEPHONES to a SYSTEM

**definition**  $NETWORK$   $:: \langle channels\ process \rangle$   
**where**  $\langle NETWORK \equiv (\parallel os \in \# (mset\text{-}set\ phones). Call \cdot os) \rangle$

**definition**  $TELEPHONES$   $:: \langle channels\ process \rangle$   
**where**  $\langle TELEPHONES \equiv (\parallel ts \in \# (mset\text{-}set\ phones). Tel\ ts) \rangle$

**definition**  $SYSTEM$   $:: \langle channels\ process \rangle$   
**where**  $\langle SYSTEM \equiv NETWORK \llbracket VisibleEvents \rrbracket TELEPHONES \rangle$

We underline here the usefulness of the architectural operators, especially *MultiSync* but also *MultiNdet* which appears in *Aside* recursive definition.

## 11.6 Simple Model of a User

**fixrec**  $User$   $:: \langle Phones \rightarrow channels\ process \rangle$   
**and**  $UserSCon$   $:: \langle Phones \rightarrow channels\ process \rangle$

**where**

$User\text{-}rec[simp\ del] : \langle User \cdot u = (off\text{-}hook!u \rightarrow$   
 $(tone\text{-}dial!u \rightarrow$   
 $(\sqcap p \in phones. digits!(u,p) \rightarrow tone\text{-}quiet!u \rightarrow$   
 $( (tone\text{-}ring!u \rightarrow connected!u \rightarrow UserSCon \cdot u)$   
 $\sqcap (connected!u \rightarrow UserSCon \cdot u)$   
 $\sqcap (tone\text{-}busy!u \rightarrow on\text{-}hook!u \rightarrow User \cdot u)$   
 $)$   
 $)$   
 $)$   
 $\sqcap (connected!u \rightarrow UserSCon \cdot u)$   
 $)$   
 $\sqcap (tone\text{-}ring!u \rightarrow off\text{-}hook!u \rightarrow connected!u \rightarrow UserSCon \cdot u) \rangle$   
 $| UserSCon\text{-}rec[simp\ del] : \langle UserSCon \cdot u = (tone\text{-}busy!u \rightarrow on\text{-}hook!u \rightarrow User \cdot u) \rangle$   
 $\triangleright (on\text{-}hook!u \rightarrow User \cdot u) \rangle$

**fixrec**  $User\text{-}Ndet$   $:: \langle Phones \rightarrow channels\ process \rangle$   
**and**  $UserSCon\text{-}Ndet$   $:: \langle Phones \rightarrow channels\ process \rangle$

**where**

$User\text{-}Ndet\text{-}rec[simp\ del] : \langle User\text{-}Ndet \cdot u = (off\text{-}hook!u \rightarrow$   
 $(tone\text{-}dial!u \rightarrow$   
 $(\sqcap p \in phones. digits!(u,p) \rightarrow tone\text{-}quiet!u \rightarrow$   
 $( (tone\text{-}ring!u \rightarrow connected!u \rightarrow UserSCon\text{-}Ndet \cdot u)$   
 $\sqcap (connected!u \rightarrow UserSCon\text{-}Ndet \cdot u)$   
 $\sqcap (tone\text{-}busy!u \rightarrow on\text{-}hook!u \rightarrow User\text{-}Ndet \cdot u)$   
 $)$   
 $)$   
 $)$

$$\begin{aligned}
& \rangle \\
& \sqcap (\text{connected!}u \rightarrow \text{UserSCon-Ndet}\cdot u) \\
& \rangle \\
& \sqcap (\text{tone-ring!}u \rightarrow \text{off-hook!}u \rightarrow \text{connected!}u \rightarrow \text{UserSCon-Ndet}\cdot u) \rangle \\
& | \text{UserSCon-Ndet-rec}[\text{simp del}]: \langle \text{UserSCon-Ndet}\cdot u = (\text{tone-busy!}u \rightarrow \text{on-hook!}u \\
& \rightarrow \text{User-Ndet}\cdot u) \sqcap (\text{on-hook!}u \rightarrow \text{User-Ndet}\cdot u) \rangle
\end{aligned}$$

**definition**  $\text{Implement}T \quad :: \langle \text{Phones} \Rightarrow \text{channels process} \rangle$   
**where**  $\langle \text{Implement}T \text{ ts} \equiv ((\text{Tel ts}) \llbracket \text{EventsIPhone ts} \cup \text{EventsUser ts} \rrbracket (\text{User}\cdot \text{ts}))$   
 $\quad \setminus (\text{EventsIPhone ts} \cup \text{EventsUser ts}) \rangle$

## 11.7 Toplevel Proof-Goals

This has been proven in an ancient FDR model for  $\text{max-phones} = 5\dots$

**lemma**  $\langle \forall p \in \text{phones. deadlock-free} (\text{Tel } p) \rangle \langle \text{proof} \rangle$   
**lemma**  $\langle \forall p \in \text{phones. deadlock-free-v2} (\text{Call}\cdot p) \rangle \langle \text{proof} \rangle$   
**lemma**  $\langle \text{deadlock-free-v2 NETWORK} \rangle \langle \text{proof} \rangle$   
**lemma**  $\langle \text{deadlock-free-v2 SYSTEM} \rangle \langle \text{proof} \rangle$   
**lemma**  $\langle \text{lifelock-free SYSTEM} \rangle \langle \text{proof} \rangle$   
**lemma**  $\langle \forall p \in \text{phones. lifelock-free} (\text{Implement}T \text{ } p) \rangle \langle \text{proof} \rangle$   
**lemma**  $\langle \forall p \in \text{phones. Tel } p \sqsubseteq_{FD} \text{Implement}T \text{ } p \rangle \langle \text{proof} \rangle$

**lemma**  $\langle \forall p \in \text{phones. Tel}\cdot p \sqsubseteq_F \text{RUN UNIV} \rangle \langle \text{proof} \rangle$

this should represent "deterministic" in process-algebraic terms. . .

**end**

**end**



# Chapter 12

## Results on *events-of*

```
theory EventsProperties
  imports CSPM
begin
```

### 12.1 With Operators of HOL-CSP

```
lemma events-of-def-tickFree:
  ⟨events-of P = (⋃ t∈{t ∈ T P. tickFree t}. {a. ev a ∈ set t})⟩
⟨proof⟩
```

```
lemma events-BOT: ⟨events-of ⊥ = UNIV⟩
and events-SKIP: ⟨events-of SKIP = {}⟩
and events-STOP: ⟨events-of STOP = {}⟩
⟨proof⟩
```

```
lemma anti-mono-events-T: ⟨P ⊆T Q ⟹ events-of Q ⊆ events-of P⟩
⟨proof⟩
```

```
lemma anti-mono-events-F: ⟨P ⊆F Q ⟹ events-of Q ⊆ events-of P⟩
⟨proof⟩
```

```
lemma anti-mono-events-FD: ⟨P ⊆FD Q ⟹ events-of Q ⊆ events-of P⟩
⟨proof⟩
```

```
lemmas events-fix-prefix =
  events-DF[of ⟨{a}⟩, simplified DF-def Mndetprefix-unit] for a
```

```
lemma events-Mndetprefix:
  ⟨events-of (Mndetprefix A P) = A ∪ (⋃ a∈A. events-of (P a))⟩
⟨proof⟩
```

**lemma** *events-Mprefix*:

$\langle \text{events-of } (M\text{prefix } A P) = A \cup (\bigcup_{a \in A} \text{events-of } (P a)) \rangle$   
*<proof>*

**lemma** *events-prefix*:  $\langle \text{events-of } (a \rightarrow P) = \text{insert } a (\text{events-of } P) \rangle$

*<proof>*

**lemma** *events-Ndet*:  $\langle \text{events-of } (P \sqcap Q) = \text{events-of } P \cup \text{events-of } Q \rangle$

*<proof>*

**lemma** *events-Det*:  $\langle \text{events-of } (P \square Q) = \text{events-of } P \cup \text{events-of } Q \rangle$

*<proof>*

**lemma** *events-Renaming*:

$\langle \text{events-of } (\text{Renaming } P f) = (\text{if } \mathcal{D} P = \{\} \text{ then } f \text{ ' events-of } P \text{ else } UNIV) \rangle$   
*<proof>*

**lemma** *events-Seq*:

$\langle \text{events-of } (P ; Q) =$   
*(if non-terminating } P \text{ then events-of } P \text{ else events-of } P \cup \text{events-of } Q) \rangle  
*<proof>**

**lemma** *events-Sync*:  $\langle \text{events-of } (P \llbracket S \rrbracket Q) \subseteq \text{events-of } P \cup \text{events-of } Q \rangle$

*<proof>*

**lemma** *events-Inter*:

$\langle \text{events-of } ((P :: \text{'}\alpha \text{ process}) \parallel Q) = \text{events-of } P \cup \text{events-of } Q \rangle$   
*<proof>*

**lemma** *empty-div-hide-events-Hiding*:  $\langle \text{events-of } (P \setminus B) \subseteq \text{events-of } P - B \rangle$

**if**  $\langle \text{div-hide } P B = \{\} \rangle$

*<proof>*

**lemma** *not-empty-div-hide-events-Hiding*:

$\langle \text{div-hide } P B \neq \{\} \implies \text{events-of } (P \setminus B) = UNIV \rangle$

*⟨proof⟩*

*events-of and deadlock-free*

**lemma** *nonempty-events-if-deadlock-free*:  $\langle \text{deadlock-free } P \implies \text{events-of } P \neq \{\} \rangle$   
*⟨proof⟩*

**lemma** *events-in-DF*:  $\langle DF\ A \sqsubseteq_{FD}\ P \implies \text{events-of } P \subseteq A \rangle$   
*⟨proof⟩*

**lemma** *nonempty-events-if-deadlock-free<sub>SKIP</sub>*:  
 $\langle \text{deadlock-free}_{SKIP}\ P \implies [tick] \in \mathcal{T}\ P \vee \text{events-of } P \neq \{\} \rangle$   
*⟨proof⟩*

**lemma** *events-in-DF<sub>SKIP</sub>*:  $\langle DF_{SKIP}\ A \sqsubseteq_{FD}\ P \implies \text{events-of } P \subseteq A \rangle$   
*⟨proof⟩*

**lemma**  $\langle \neg \text{events-of } P \subseteq A \implies \neg DF\ A \sqsubseteq_{FD}\ P \rangle$   
**and**  $\langle \neg \text{events-of } P \subseteq A \implies \neg DF_{SKIP}\ A \sqsubseteq_{FD}\ P \rangle$   
*⟨proof⟩*

## 12.2 With Architectural Operators of HOL-CSPM

**lemma** *events-MultiNdet*:  
 $\langle \text{finite } A \implies \text{events-of } (\text{MultiNdet } A\ P) = (\bigcup_{a \in A}. \text{events-of } (P\ a)) \rangle$   
*⟨proof⟩*

**lemma** *events-MultiDet*:  
 $\langle \text{finite } A \implies \text{events-of } (\text{MultiDet } A\ P) = (\bigcup_{a \in A}. \text{events-of } (P\ a)) \rangle$   
*⟨proof⟩*

**lemma** *events-MultiSeq*:  
 $\langle \text{events-of } (SEQ\ a \in@ L. P\ a) =$   
 $(\bigcup_{a \in \text{set } (take\ (Suc\ (first\ elem\ (\lambda a. \text{non-terminating } (P\ a))\ L))\ L}. \text{events-of } (P\ a)) \rangle$   
*⟨proof⟩*

**lemma** *events-MultiSeq-subset*:  
 $\langle \text{events-of } (SEQ\ a \in@ L. P\ a) \subseteq (\bigcup_{a \in \text{set } L. \text{events-of } (P\ a)) \rangle$   
*⟨proof⟩*

**lemma** *events-MultiSync*:  
 $\langle \text{events-of } ([S] a \in\# M. P\ a) \subseteq (\bigcup_{a \in \text{set-mset } M. \text{events-of } (P\ a)) \rangle$   
*⟨proof⟩*

**lemma** *events-MultiInter*:  
 $\langle \text{events-of } (\bigvee a \in \# M. P a) = (\bigcup a \in \text{set-mset } M. \text{events-of } (P a)) \rangle$   
*<proof>*

**end**



# Chapter 13

## Deadlock Results

```
theory DeadlockResults
  imports CSPM
begin
```

When working with the interleaving  $P|||Q$ , we intuitively expect it to be *deadlock-free* when both  $P$  and  $Q$  are.

This chapter contains several results about deadlock notion, and concludes with a proof of the theorem we just mentioned.

### 13.1 Unfolding Lemmas for the Projections of $DF$ and $DF_{SKIP}$

$DF$  and  $DF_{SKIP}$  naturally appear when we work around *deadlock-free* and *deadlock-free<sub>SKIP</sub>* notions (because

*deadlock-free*  $P \equiv DF\ UNIV \sqsubseteq_{FD}\ P$

*deadlock-free<sub>SKIP</sub>*  $P \equiv DF_{SKIP}\ UNIV \sqsubseteq_F\ P$ ).

It is therefore convenient to have the following rules for unfolding the projections.

**lemma** *F-DF*:

```
 $\langle \mathcal{F}\ (DF\ A) =$ 
  (if  $A = \{\}$  then  $\{(s, X). s = []\}$ 
  else  $(\bigcup_{x \in A}. \{[]\} \times \{ref.\ ev\ x \notin ref\}) \cup$ 
     $\{(tr, ref). tr \neq [] \wedge hd\ tr = ev\ x \wedge (tl\ tr, ref) \in \mathcal{F}\ (DF\ A)\}) \rangle$ 
```

**and** *F-DF<sub>SKIP</sub>*:

```
 $\langle \mathcal{F}\ (DF_{SKIP}\ A) =$ 
  (if  $A = \{\}$  then  $\{(s, X). s = [] \vee s = [tick]\}$ 
  else  $\{(s, X) \mid s \in X. s = [] \wedge tick \notin X \vee s = [tick]\} \cup$ 
     $(\bigcup_{x \in A}. \{[]\} \times \{ref.\ ev\ x \notin ref\}) \cup$ 
     $\{(tr, ref). tr \neq [] \wedge hd\ tr = ev\ x \wedge (tl\ tr, ref) \in \mathcal{F}\ (DF_{SKIP}\ A)\}) \rangle$ 
```

*<proof>*

**corollary** *Cons-F-DF*:

$$\langle (x \# t, X) \in \mathcal{F} (DF A) \implies (t, X) \in \mathcal{F} (DF A) \rangle$$

**and** *Cons-F-DF<sub>SKIP</sub>*:

$$\langle x \neq tick \implies (x \# t, X) \in \mathcal{F} (DF_{SKIP} A) \implies (t, X) \in \mathcal{F} (DF_{SKIP} A) \rangle$$

*<proof>*

**lemma** *D-DF*:  $\langle \mathcal{D} (DF A) = (if A = \{\} then \{\}$

$$else \{s. s \neq [] \wedge (\exists x \in A. hd s = ev x \wedge tl s \in \mathcal{D} (DF A))\} \rangle$$

**and** *D-DF<sub>SKIP</sub>*:  $\langle \mathcal{D} (DF_{SKIP} A) = (if A = \{\} then \{\}$

$$else \{s. s \neq [] \wedge (\exists x \in A. hd s = ev x \wedge tl s \in \mathcal{D} (DF_{SKIP} A))\} \rangle$$

*<proof>*

**lemma** *T-DF*:

$$\langle \mathcal{T} (DF A) =$$

$$(if A = \{\} then \{[]\}$$

$$else \{s. s = [] \vee s \neq [] \wedge (\exists x \in A. hd s = ev x \wedge tl s \in \mathcal{T} (DF A))\} \rangle$$

**and** *T-DF<sub>SKIP</sub>*:

$$\langle \mathcal{T} (DF_{SKIP} A) =$$

$$(if A = \{\} then \{[], [tick]\}$$

$$else \{s. s = [] \vee s = [tick] \vee$$

$$s \neq [] \wedge (\exists x \in A. hd s = ev x \wedge tl s \in \mathcal{T} (DF_{SKIP} A))\} \rangle$$

*<proof>*

## 13.2 Characterizations for *deadlock-free*, *deadlock-free<sub>SKIP</sub>*

We want more results like  $\llbracket \text{deadlock-free } P; \text{ deadlock-free } Q \rrbracket \implies \text{deadlock-free } (P \sqcap Q)$ , and we want to add the reciprocal when possible.

The first thing we notice is that we only have to care about the failures

**lemma**  $\langle \text{deadlock-free}_{SKIP} P \equiv DF_{SKIP} UNIV \sqsubseteq_F P \rangle$

*<proof>*

**lemma** *deadlock-free-F*:  $\langle \text{deadlock-free } P \longleftrightarrow DF UNIV \sqsubseteq_F P \rangle$

*<proof>*

**lemma** *deadlock-free-Mprefix-iff*:  $\langle \text{deadlock-free } (\square a \in A \rightarrow P a) \longleftrightarrow$

$$A \neq \{\} \wedge (\forall a \in A. \text{ deadlock-free } (P a)) \rangle$$

**and** *deadlock-free<sub>SKIP</sub>-Mprefix-iff*:  $\langle \text{deadlock-free}_{SKIP} (Mprefix A P) \longleftrightarrow$

$$A \neq \{\} \wedge (\forall a \in A. \text{ deadlock-free}_{SKIP} (P a)) \rangle$$

*<proof>*

**lemmas** *deadlock-free-prefix-iff* =

$deadlock-free-Mprefix-iff$  [of  $\langle \{a\} \rangle \langle \lambda a. P \rangle$ , folded write0-def, simplified]  
**and**  $deadlock-free_{SKIP}-prefix-iff =$   
 $deadlock-free_{SKIP}-Mprefix-iff$  [of  $\langle \{a\} \rangle \langle \lambda a. P \rangle$ , folded write0-def, simplified]  
**for**  $a P$

**lemma**  $deadlock-free-Mndetprefix-iff$ :  $\langle deadlock-free (\sqcap a \in A \rightarrow P a) \longleftrightarrow$   
 $A \neq \{\} \wedge (\forall a \in A. deadlock-free (P a)) \rangle$   
**and**  $deadlock-free_{SKIP}-Mndetprefix-iff$ :  $\langle deadlock-free_{SKIP} (\sqcap a \in A \rightarrow P a)$   
 $\longleftrightarrow$   
 $A \neq \{\} \wedge (\forall a \in A. deadlock-free_{SKIP} (P a)) \rangle$   
 $\langle proof \rangle$

**lemma**  $deadlock-free-Ndet-iff$ :  $\langle deadlock-free (P \sqcap Q) \longleftrightarrow$   
 $deadlock-free P \wedge deadlock-free Q \rangle$   
**and**  $deadlock-free_{SKIP}-Ndet-iff$ :  $\langle deadlock-free_{SKIP} (P \sqcap Q) \longleftrightarrow$   
 $deadlock-free_{SKIP} P \wedge deadlock-free_{SKIP} Q \rangle$   
 $\langle proof \rangle$

**lemma**  $deadlock-free-GlobalNdet-iff$ :  $\langle deadlock-free (\sqcap a \in A. P a) \longleftrightarrow$   
 $A \neq \{\} \wedge (\forall a \in A. deadlock-free (P a)) \rangle$   
**and**  $deadlock-free_{SKIP}-GlobalNdet-iff$ :  $\langle deadlock-free_{SKIP} (\sqcap a \in A. P a) \longleftrightarrow$   
 $A \neq \{\} \wedge (\forall a \in A. deadlock-free_{SKIP} (P a)) \rangle$   
 $\langle proof \rangle$

**lemma**  $deadlock-free-MultiNdet-iff$ :  $\langle deadlock-free (\prod a \in A. P a) \longleftrightarrow$   
 $A \neq \{\} \wedge (\forall a \in A. deadlock-free (P a)) \rangle$   
**and**  $deadlock-free_{SKIP}-MultiNdet-iff$ :  $\langle deadlock-free_{SKIP} (\prod a \in A. P a) \longleftrightarrow$   
 $A \neq \{\} \wedge (\forall a \in A. deadlock-free_{SKIP} (P a)) \rangle$   
**if**  $fin$ :  $\langle finite A \rangle$   
 $\langle proof \rangle$

**lemma**  $deadlock-free-MultiDet$ :  
 $\langle \llbracket A \neq \{\}; finite A; \forall a \in A. deadlock-free (P a) \rrbracket \implies deadlock-free (\sqcap a \in A. P a) \rangle$   
**and**  $deadlock-free_{SKIP}-MultiDet$ :  
 $\langle \llbracket A \neq \{\}; finite A; \forall a \in A. deadlock-free_{SKIP} (P a) \rrbracket \implies deadlock-free_{SKIP} (\sqcap a \in A. P a) \rangle$   
 $\langle proof \rangle$

**lemma** *deadlock-free-Det*:

$\langle \text{deadlock-free } P \implies \text{deadlock-free } Q \implies \text{deadlock-free } (P \square Q) \rangle$

**and** *deadlock-free<sub>SKIP</sub>-Det*:

$\langle \text{deadlock-free}_{\text{SKIP}} P \implies \text{deadlock-free}_{\text{SKIP}} Q \implies \text{deadlock-free}_{\text{SKIP}} (P \square Q) \rangle$

$\langle \text{proof} \rangle$

For  $P \square Q$ , we can not expect more:

**lemma**

$\langle \exists P Q. \text{deadlock-free } P \wedge \neg \text{deadlock-free } Q \wedge$   
 $\text{deadlock-free } (P \square Q) \rangle$

$\langle \exists P Q. \text{deadlock-free}_{\text{SKIP}} P \wedge \neg \text{deadlock-free}_{\text{SKIP}} Q \wedge$   
 $\text{deadlock-free}_{\text{SKIP}} (P \square Q) \rangle$

$\langle \text{proof} \rangle$

**lemma** *FD-Mndetprefix-iff*:

$\langle A \neq \{\} \implies P \sqsubseteq_{\text{FD}} \sqcap a \in A \rightarrow Q \iff (\forall a \in A. P \sqsubseteq_{\text{FD}} (a \rightarrow Q)) \rangle$

$\langle \text{proof} \rangle$

**lemma** *Mndetprefix-FD*:  $\langle (\exists a \in A. (a \rightarrow Q) \sqsubseteq_{\text{FD}} P) \implies \sqcap a \in A \rightarrow Q \sqsubseteq_{\text{FD}} P \rangle$

$\langle \text{proof} \rangle$

*Mprefix, Sync and deadlock-free*

**lemma** *Mprefix-Sync-deadlock-free*:

**assumes** *not-all-empty*:  $\langle A \neq \{\} \vee B \neq \{\} \vee A' \cap B' \neq \{\} \rangle$

**and**  $\langle A \cap S = \{\} \rangle$  **and**  $\langle A' \subseteq S \rangle$  **and**  $\langle B \cap S = \{\} \rangle$  **and**  $\langle B' \subseteq S \rangle$

**and**  $\langle \forall x \in A. \text{deadlock-free } (P x \llbracket S \rrbracket \text{Mprefix } (B \cup B') Q) \rangle$

**and**  $\langle \forall y \in B. \text{deadlock-free } (\text{Mprefix } (A \cup A') P \llbracket S \rrbracket Q y) \rangle$

**and**  $\langle \forall x \in A' \cap B'. \text{deadlock-free } ((P x \llbracket S \rrbracket Q x)) \rangle$

**shows**  $\langle \text{deadlock-free } (\text{Mprefix } (A \cup A') P \llbracket S \rrbracket \text{Mprefix } (B \cup B') Q) \rangle$

$\langle \text{proof} \rangle$

**lemmas** *Mprefix-Sync-subset-deadlock-free = Mprefix-Sync-deadlock-free*

**[where**  $A = \{\}$  **and**  $B = \{\}$  **], simplified]**

**and** *Mprefix-Sync-indep-deadlock-free = Mprefix-Sync-deadlock-free*

**[where**  $A' = \{\}$  **and**  $B' = \{\}$  **], simplified]**

**and** *Mprefix-Sync-right-deadlock-free = Mprefix-Sync-deadlock-free*

**[where**  $A = \{\}$  **and**  $B' = \{\}$  **], simplified]**

**and** *Mprefix-Sync-left-deadlock-free = Mprefix-Sync-deadlock-free*

**[where**  $A' = \{\}$  **and**  $B = \{\}$  **], simplified]**

## 13.3 Results on Renaming

The *Renaming* operator is new (release of 2023), so here are its properties on reference processes from *HOL-CSP.Assertions*, and deadlock notion.

### 13.3.1 Behaviour with References Processes

For  $DF$

**lemma** *DF-FD-Renaming-DF*:  $\langle DF (f \text{ ' } A) \sqsubseteq_{FD} Renaming (DF A) f \rangle$   
*<proof>*

**lemma** *Renaming-DF-FD-DF*:  $\langle Renaming (DF A) f \sqsubseteq_{FD} DF (f \text{ ' } A) \rangle$   
**if** *finitary*:  $\langle finitary f \rangle$   
*<proof>*

For  $DF_{SKIP}$

**lemma** *DF\_{SKIP}-FD-Renaming-DF\_{SKIP}*:  
 $\langle DF_{SKIP} (f \text{ ' } A) \sqsubseteq_{FD} Renaming (DF_{SKIP} A) f \rangle$   
*<proof>*

**lemma** *Renaming-DF\_{SKIP}-FD-DF\_{SKIP}*:  
 $\langle Renaming (DF_{SKIP} A) f \sqsubseteq_{FD} DF_{SKIP} (f \text{ ' } A) \rangle$   
**if** *finitary*:  $\langle finitary f \rangle$   
*<proof>*

For  $RUN$

**lemma** *RUN-FD-Renaming-RUN*:  $\langle RUN (f \text{ ' } A) \sqsubseteq_{FD} Renaming (RUN A) f \rangle$   
*<proof>*

**lemma** *Renaming-RUN-FD-RUN*:  $\langle Renaming (RUN A) f \sqsubseteq_{FD} RUN (f \text{ ' } A) \rangle$   
**if** *finitary*:  $\langle finitary f \rangle$   
*<proof>*

For  $CHAOS$

**lemma** *CHAOS-FD-Renaming-CHAOS*:  
 $\langle CHAOS (f \text{ ' } A) \sqsubseteq_{FD} Renaming (CHAOS A) f \rangle$   
*<proof>*

**lemma** *Renaming-CHAOS-FD-CHAOS*:  
 $\langle Renaming (CHAOS A) f \sqsubseteq_{FD} CHAOS (f \text{ ' } A) \rangle$   
**if** *finitary*:  $\langle finitary f \rangle$   
*<proof>*

For  $CHAOS_{SKIP}$

**lemma** *CHAOS\_{SKIP}-FD-Renaming-CHAOS\_{SKIP}*:  
 $\langle CHAOS_{SKIP} (f \text{ ' } A) \sqsubseteq_{FD} Renaming (CHAOS_{SKIP} A) f \rangle$   
*<proof>*

**lemma** *Renaming-CHAOS<sub>SKIP</sub>-FD-CHAOS<sub>SKIP</sub>*:  
 $\langle \text{Renaming } (CHAOS_{SKIP} A) f \sqsubseteq_{FD} CHAOS_{SKIP} (f ' A) \rangle$   
**if** *finitary*:  $\langle \text{finitary } f \rangle$   
 $\langle \text{proof} \rangle$

### 13.3.2 Corollaries on *deadlock-free* and *deadlock-free<sub>SKIP</sub>*

**lemmas** *Renaming-DF* =  
 $FD\text{-antisym}[OF \text{ Renaming-DF-FD-DF } DF\text{-FD-Renaming-DF}]$   
**and** *Renaming-DF<sub>SKIP</sub>* =  
 $FD\text{-antisym}[OF \text{ Renaming-DF}_{SKIP}\text{-FD-DF}_{SKIP} \text{ DF}_{SKIP}\text{-FD-Renaming-DF}_{SKIP}]$   
**and** *Renaming-RUN* =  
 $FD\text{-antisym}[OF \text{ Renaming-RUN-FD-RUN } RUN\text{-FD-Renaming-RUN}]$   
**and** *Renaming-CHAOS* =  
 $FD\text{-antisym}[OF \text{ Renaming-CHAOS-FD-CHAOS } CHAOS\text{-FD-Renaming-CHAOS}]$   
**and** *Renaming-CHAOS<sub>SKIP</sub>* =  
 $FD\text{-antisym}[OF \text{ Renaming-CHAOS}_{SKIP}\text{-FD-CHAOS}_{SKIP} \text{ CHAOS}_{SKIP}\text{-FD-Renaming-CHAOS}_{SKIP}]$

**lemma** *deadlock-free-imp-deadlock-free-Renaming*:  $\langle \text{deadlock-free } (Renaming P f) \rangle$   
**if**  $\langle \text{deadlock-free } P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deadlock-free-Renaming-imp-deadlock-free*:  $\langle \text{deadlock-free } P \rangle$   
**if**  $\langle \text{inj } f \rangle$  **and**  $\langle \text{deadlock-free } (Renaming P f) \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *deadlock-free-Renaming-iff*:  
 $\langle \text{inj } f \implies \text{deadlock-free } (Renaming P f) \iff \text{deadlock-free } P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deadlock-free<sub>SKIP</sub>-imp-deadlock-free<sub>SKIP</sub>-Renaming*:  
 $\langle \text{deadlock-free}_{SKIP} (Renaming P f) \rangle$  **if**  $\langle \text{deadlock-free}_{SKIP} P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *deadlock-free<sub>SKIP</sub>-Renaming-imp-deadlock-free<sub>SKIP</sub>*:  
 $\langle \text{deadlock-free}_{SKIP} P \rangle$  **if**  $\langle \text{inj } f \rangle$  **and**  $\langle \text{deadlock-free}_{SKIP} (Renaming P f) \rangle$   
 $\langle \text{proof} \rangle$

**corollary** *deadlock-free<sub>SKIP</sub>-Renaming-iff*:  
 $\langle \text{inj } f \implies \text{deadlock-free}_{SKIP} (Renaming P f) \iff \text{deadlock-free}_{SKIP} P \rangle$   
 $\langle \text{proof} \rangle$

## 13.4 Big Results

### 13.4.1 Interesting Equivalence

**lemma** *deadlock-free-of-Sync-iff-DF-FD-DF-Sync-DF*:

$$\langle (\forall P Q. \text{deadlock-free } (P::'\alpha \text{ process}) \longrightarrow \text{deadlock-free } Q \longrightarrow \\ \text{deadlock-free } (P \llbracket S \rrbracket Q)) \\ \longleftrightarrow (DF (UNIV::'\alpha \text{ set}) \sqsubseteq_{FD} (DF UNIV \llbracket S \rrbracket DF UNIV)) \rangle \text{ (is } \langle ?lhs \longleftrightarrow ?rhs \rangle) \\ \langle \text{proof} \rangle$$

From this general equivalence on *Sync*, we immediately obtain the equivalence on  $A \parallel B$ :  $(\forall P Q. \text{deadlock-free } P \longrightarrow \text{deadlock-free } Q \longrightarrow \text{deadlock-free } (P \parallel Q)) = DF UNIV \sqsubseteq_{FD} DF UNIV \parallel DF UNIV$ .

### 13.4.2 STOP and SKIP Synchronized with DF A

**lemma** *DF-FD-DF-Sync-STOP-or-SKIP-iff*:

$$\langle (DF A \sqsubseteq_{FD} DF A \llbracket S \rrbracket P) \longleftrightarrow A \cap S = \{\} \rangle \\ \text{if } P\text{-disj}: \langle P = STOP \vee P = SKIP \rangle \\ \langle \text{proof} \rangle$$

**lemma** *DF-Sync-STOP-or-SKIP-FD-DF*:  $\langle DF A \llbracket S \rrbracket P \sqsubseteq_{FD} DF A \rangle$   
 if *P-disj*:  $\langle P = STOP \vee P = SKIP \rangle$  **and** *empty-inter*:  $\langle A \cap S = \{\} \rangle$   
 $\langle \text{proof} \rangle$

**lemmas** *DF-FD-DF-Sync-STOP-iff* =  
 $DF-FD-DF-Sync-STOP-or-SKIP-iff$  [of *STOP*, *simplified*]  
**and** *DF-FD-DF-Sync-SKIP-iff* =  
 $DF-FD-DF-Sync-STOP-or-SKIP-iff$  [of *SKIP*, *simplified*]  
**and** *DF-Sync-STOP-FD-DF* =  
 $DF-Sync-STOP-or-SKIP-FD-DF$  [of *STOP*, *simplified*]  
**and** *DF-Sync-SKIP-FD-DF* =  
 $DF-Sync-STOP-or-SKIP-FD-DF$  [of *SKIP*, *simplified*]

### 13.4.3 Finally, deadlock-free (P ||| Q)

**theorem** *DF-F-DF-Sync-DF*:  $\langle DF (A \cup B::'\alpha \text{ set}) \sqsubseteq_F DF A \llbracket S \rrbracket DF B \rangle$   
 if *nonempty*:  $\langle A \neq \{\} \wedge B \neq \{\} \rangle$   
**and** *intersect-hyp*:  $\langle B \cap S = \{\} \vee (\exists y. B \cap S = \{y\} \wedge A \cap S \subseteq \{y\}) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *DF-FD-DF-Sync-DF*:

$$\langle A \neq \{\} \wedge B \neq \{\} \implies B \cap S = \{\} \vee (\exists y. B \cap S = \{y\} \wedge A \cap S \subseteq \{y\}) \implies \\ DF (A \cup B) \sqsubseteq_{FD} DF A \llbracket S \rrbracket DF B \rangle \\ \langle \text{proof} \rangle$$

**theorem** *DF-FD-DF-Sync-DF-iff*:

$$\begin{aligned} &\langle DF (A \cup B) \sqsubseteq_{FD} DF A \llbracket S \rrbracket DF B \longleftrightarrow \\ & \quad ( \text{if } A = \{\} \text{ then } B \cap S = \{\} \\ & \quad \text{else if } B = \{\} \text{ then } A \cap S = \{\} \\ & \quad \text{else } A \cap S = \{\} \vee (\exists a. A \cap S = \{a\} \wedge B \cap S \subseteq \{a\}) \vee \\ & \quad \quad B \cap S = \{\} \vee (\exists b. B \cap S = \{b\} \wedge A \cap S \subseteq \{b\}) \rangle \\ & \text{(is } \langle ?FD-ref \longleftrightarrow ( \text{if } A = \{\} \text{ then } B \cap S = \{\} \\ & \quad \text{else if } B = \{\} \text{ then } A \cap S = \{\} \\ & \quad \text{else } ?cases \rangle \rangle) \end{aligned}$$

$\langle proof \rangle$

**lemma**

$$\begin{aligned} &\langle (\forall a \in A. X a \cap S = \{\} \vee (\forall b \in A. \exists y. X a \cap S = \{y\} \wedge X b \cap S \subseteq \{y\})) \\ & \longleftrightarrow (\forall a \in A. \forall b \in A. \exists y. (X a \cup X b) \cap S \subseteq \{y\}) \rangle \\ & \text{— this is the reason we write ugly\_hyp this way} \\ & \langle proof \rangle \end{aligned}$$

**lemma** *DF-FD-DF-MultiSync-DF*:

$$\begin{aligned} &\langle DF (\bigcup x \in (\text{insert } a \ A). X x) \sqsubseteq_{FD} \llbracket S \rrbracket x \in \# \text{ mset-set } (\text{insert } a \ A). DF (X x) \rangle \\ & \text{if fin: } \langle \text{finite } A \rangle \text{ and nonempty: } \langle X a \neq \{\} \rangle \langle \forall b \in A. X b \neq \{\} \rangle \\ & \text{and ugly-hyp: } \langle \forall b \in A. X b \cap S = \{\} \vee (\exists y. X b \cap S = \{y\} \wedge X a \cap S \subseteq \{y\}) \rangle \\ & \quad \langle \forall b \in A. \forall c \in A. \exists y. (X b \cup X c) \cap S \subseteq \{y\} \rangle \end{aligned}$$

$\langle proof \rangle$

**lemma** *DF-FD-DF-MultiSync-DF'*:

$$\begin{aligned} &\langle \llbracket \text{finite } A; \forall a \in A. X a \neq \{\}; \forall a \in A. \forall b \in A. \exists y. (X a \cup X b) \cap S \subseteq \{y\} \rrbracket \\ & \implies DF (\bigcup a \in A. X a) \sqsubseteq_{FD} \llbracket S \rrbracket a \in \# \text{ mset-set } A. DF (X a) \rangle \\ & \langle proof \rangle \end{aligned}$$

**lemmas** *DF-FD-DF-MultiInter-DF =*

$$DF-FD-DF-MultiSync-DF' [\text{where } S = \langle \{\} \rangle, \text{ simplified}]$$

**and** *DF-FD-DF-MultiPar-DF =*

$$DF-FD-DF-MultiSync-DF' [\text{where } S = UNIV, \text{ simplified}]$$

**and** *DF-FD-DF-MultiPar-DF' =*

$$DF-FD-DF-MultiSync-DF' [\text{where } S = UNIV, \text{ simplified}]$$

**lemma**  $\langle DF \{a\} = DF \{a\} \llbracket S \rrbracket STOP \longleftrightarrow a \notin S \rangle$



*<proof>*

**lemma**  $\langle DF \{a\} \llbracket S \rrbracket STOP = STOP \longleftrightarrow a \in S \rangle$   
*<proof>*

**corollary** *DF-FD-DF-Inter-DF*:  $\langle DF (A::'\alpha \text{ set}) \sqsubseteq_{FD} DF A \parallel DF A \rangle$   
*<proof>*

**corollary** *DF-UNIV-FD-DF-UNIV-Inter-DF-UNIV*:  
 $\langle DF UNIV \sqsubseteq_{FD} DF UNIV \parallel DF UNIV \rangle$   
*<proof>*

**corollary** *Inter-deadlock-free*:  
 $\langle deadlock\text{-free } P \implies deadlock\text{-free } Q \implies deadlock\text{-free } (P \parallel Q) \rangle$   
*<proof>*

**theorem** *MultiInter-deadlock-free*:  
 $\langle M \neq \{\#\} \implies \forall p \in \# M. deadlock\text{-free } (P p) \implies$   
 $deadlock\text{-free } (\parallel p \in \# M. P p) \rangle$   
*<proof>*

**end**



# Chapter 14

## Conclusion

In this session, we defined five architectural operators: *MultiDet*, *MultiNdet* and *GlobalNdet*, *MultiSync*, and *MultiSeq* as respective generalizations of  $P \sqcap Q$ ,  $P \sqcap Q$ ,  $P \llbracket S \rrbracket Q$ , and  $P ; Q$ .

We did this in a fully-abstract way, that is:

- $(\sqcap)$  is commutative, idempotent and admits *STOP* as a neutral element so we defined *MultiDet* on a *finite 'α set*  $A$  by making it equal to *STOP* when  $A = \emptyset$ .
- $(\sqcap)$  is also commutative and idempotent so we defined *MultiNdet* on a *finite 'α set*  $A$  by making it equal to *STOP* when  $A = \emptyset$ . Beware of the fact that *STOP* is not the neutral element for  $(\sqcap)$  (this operator does not admit a neutral element) so we **do not have** the equality

$$\sqcap_{p \in \{a\}} P p = P a \sqcap (\sqcap_{p \in \emptyset} P p)$$

while this holds for  $(\sqcap)$  and *MultiDet*.

As its failures and divergences can easily be generalized to the infinite case, we also defined *GlobalNdet* verifying

$$\text{finite } A \implies \sqcap_{p \in A} P p = \sqcap_{p \in A} P p$$

- *Sync* is commutative but is not idempotent so we defined *MultiSync* on a *'α multiset*  $M$  to keep the multiplicity of the processes. We made it equal to *STOP* when  $M = \{\#\}$  but like  $(\sqcap)$ , *Sync* does not admit a neutral element so beware of the fact that in general

$$\llbracket S \rrbracket p \in \#\{a\#\}. P p \neq P a \llbracket S \rrbracket (\llbracket S \rrbracket p \in \#\{a\#\}. P p)$$

- $(;)$  is neither commutative nor idempotent, so we defined *MultiSeq* on a *'α list*  $L$  to keep the multiplicity and the order of the processes. Since *SKIP* is the neutral element for  $(;)$ , we have

$$SEQ\ p \in @ [a].\ P\ p = (SEQ\ p \in @ [].\ P\ p) ; P\ a$$

$$SEQ\ p \in @ [a].\ P\ p = P\ a ; (SEQ\ p \in @ [].\ P\ p)$$

On our architectural operators we proved continuity (under weakest liberal assumptions), wrote refinements rules and obtained results about the behaviour with other operators inherited from the binary rules.

We presented two examples: Dining Philosophers, and POTS.

In both, we underlined the usefulness of the architectural operators for modeling complex systems.

Finally we provided powerful results on *events-of* and *deadlock-free* among which the most important is undoubtedly :

$$\llbracket M \neq \{\#\}; \forall p \in \#M. \text{deadlock-free} (P\ p) \rrbracket \implies \text{deadlock-free} (\lll p \in \#M. P\ p \rrl)$$

This theorem allows, for example, to establish:

$$0 < n \implies \text{deadlock-free} (\lll i \in \#mset [0..<n]. P\ i \rrl)$$

under the assumption that a family of processes parameterized by  $i :: nat$  verifies  $\forall i < n. \text{deadlock-free} (P\ i)$ .

# Bibliography

- [1] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 281–305, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
- [2] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.
- [3] S. Taha, L. Ye, and B. Wolff. Hol-csp version 2.0. *Archive of Formal Proofs*, April 2019. <https://isa-afp.org/entries/HOL-CSP.html>, Formal proof development.