

# Possibilistic Noninterference\*

Andrei Popescu

Johannes Hölzl

December 14, 2021

## Abstract

We formalize a wide variety of Volpano/Smith-style noninterference notions for a while language with parallel composition. We systematize and classify these notions according to compositionality w.r.t. the language constructs. Compositionality yields sound syntactic criteria (a.k.a. type systems) in a uniform way.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Bisimilarity, abstractly</b>	<b>4</b>
<b>3</b>	<b>The programming language and its semantics</b>	<b>7</b>
3.1	Syntax and operational semantics . . . . .	7
3.2	Sublanguages . . . . .	15
<b>4</b>	<b>During-execution security</b>	<b>21</b>
4.1	Basic setting . . . . .	21
4.2	Compatibility and discreteness . . . . .	22
4.3	Terminating-interactive discreteness . . . . .	24
4.4	Self-isomorphism . . . . .	26
4.5	MustT-interactive self-isomorphism . . . . .	27
4.6	Notions of bisimilarity . . . . .	28
<b>5</b>	<b>Compositionality of the during-execution security notions</b>	<b>44</b>
5.1	Discreteness versus language constructs: . . . . .	44
5.2	Discreteness versus language constructs: . . . . .	46
5.3	Self-Isomorphism versus language constructs: . . . . .	49
5.4	Self-Isomorphism versus language constructs: . . . . .	51
5.5	Strong bisimilarity versus language constructs . . . . .	53

---

\*Supported by the DFG project Ni 491/13-1 (part of the DFG priority program RS3) and the DFG RTG 1480.

5.5.1	01T-bisimilarity versus language constructs . . . . .	59
5.5.2	01-bisimilarity versus language constructs . . . . .	65
5.5.3	WT-bisimilarity versus language constructs . . . . .	78
5.5.4	T-bisimilarity versus language constructs . . . . .	84
5.5.5	W-bisimilarity versus language constructs . . . . .	91
<b>6</b>	<b>After-execution security</b>	<b>107</b>
6.1	Setup for bisimilarities . . . . .	108
6.2	Execution traces . . . . .	116
6.3	Relationship between during-execution and after-execution security . . . . .	117
<b>7</b>	<b>Concrete setting</b>	<b>121</b>
7.1	Programs from EXAMPLE 1 . . . . .	124

## 1 Introduction

This is a formalization of the mathematical development presented in the paper [1]:

- a uniform framework where a wide range of language-based noninterference variants from the literature are expressed and compared w.r.t. their *contracts*: the strength of the security properties they ensure weighed against the harshness of the syntactic conditions they enforce;
- syntactic criteria for proving that a program has a specific noninterference property, using only compositionality, which captures uniformly several security type-system results from the literature and suggests a further improved type system.

There are two auxiliary theories:

- MyTactics, introducing a few customized tactics;
- Bisim, describing an abstract notion of bisimilarity relation, namely, the greatest symmetric relation that is a fixpoint of a monotonic operator—this shall be instantiated to several concrete bisimilarity later.

The main theories of the development (shown in Fig. 1) are organized similarly to the sectionwise structure of [1]:

Language\_Semantics corresponds to §2 in [1]. It introduces and customizes the syntax and small-step operational semantics of a while language with parallel composition, using notations very similar to the paper.

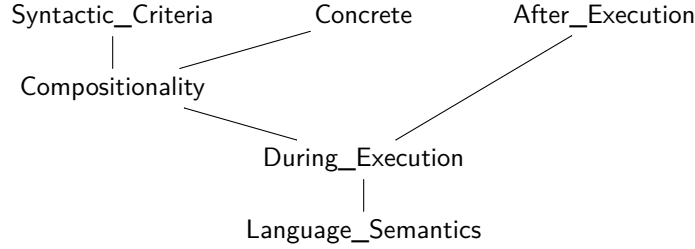


Figure 1: Main Theory Structure

`During_Execution`<sup>1</sup> mainly corresponds to §3 in [1], defining the various coinductive notions from there: self isomorphism, discreteness, variations of strong, weak and 01-bisimilarity. Prop. 1 from the paper, stating implications between these notions, is proved as the theorems `bis_imp` and `siso_bis`.<sup>2</sup> The bisimilarity inclusions stated in `bis_imp` are slightly more general than those in Prop. 1, in that they employ the binary version of the relation, e.g.,  $c \approx_s d \implies c \approx_{\text{WT}} d$  instead of  $c \approx_s c \implies c \approx_{\text{WT}} c$ .

`Compositionality` mainly corresponds to the homonymous §4 in [1]. The paper’s compositionality result, Prop. 2, is scattered through the theory as theorems with self-explanatory names, indicating the compositionality relationship between notions of noninterference and language constructs, e.g., `While_WbisT` (while versus termination-sensitive weak bisimilarity), `Par_ZO_bis` (parallel composition versus 01-bisimilarity).

Theories `During_Execution` and `Compositionality` also include the novel notion of noninterference  $\approx_{\tau}$  introduced in §5 of [1], based on the “must terminate” condition, which is given the same treatment as the other notions: `bis_imp` in `During_Execution` states the implication relationship between  $\approx_{\tau}$  and the other bisimilarities (Prop. 3.(1) from [1]), while various intuitively named theorems from `Language_Semantics` state the compositionality properties of  $\approx_{\tau}$  (Prop. 3.(2) from [1]).

`Syntactic_Criteria` corresponds to the homonymous §6 in [1]. The syntactic analogues of the semantics notions, indicated in the paper by overlining, e.g., `discr`, are in the scripts prefixed by “SC” (from “syntactic criterion”), e.g., `SC_discr`, `SC_WbisT`. Props. 4 and 5 from the paper (stating the relationship between the syntactic and the semantic notions and the implications between the syntactic notions, respectively) are again scattered through the theory under self-explanatory names.

`Concrete` contains an instantiation of the indistinguishability relation  $\sim$  from [1] to the standard two-level security setting described in the paper’s Exam-

<sup>1</sup>“During-execution” (bisimilarity-based) noninterference should be contrasted with “after-execution” (trace-based) noninterference according to the distinction made in [1] at the beginning of §7.

<sup>2</sup>To help the reader distinguish the main results from the auxiliary lemmas, the former are marked in the scripts with the keyword “theorem”.

ple 2.

Finally, `After_Execution` corresponds to §7 in [1], dealing with the after-execution guarantees of the during-execution notions of security. Prop. 6 in the paper is stated in the scripts as theorems `Sbis_trace`, `ZObisT_trace` and `WbisT_trace`, Prop. 7 as theorems `ZObis_trace` and `Wbis_trace`, and Prop. 8 as theorem `BisT_trace`.

## 2 Bisimilarity, abstractly

```
theory Bisim
imports Interface
begin
```

```
type-synonym 'a rel = ('a * 'a) set
type-synonym ('cmd,'state)config = 'cmd * 'state
```

```
definition mono where
mono Retr  $\equiv$ 
 $\forall$  theta theta'. theta  $\leq$  theta'  $\longrightarrow$  Retr theta  $\leq$  Retr theta'
```

```
definition simul where
simul Retr theta  $\equiv$  theta  $\leq$  Retr theta
```

```
definition bisim where
bisim Retr theta  $\equiv$  sym theta  $\wedge$  simul Retr theta
```

```
lemma mono-Union:
assumes mono Retr
shows Union (Retr ' Theta)  $\leq$  Retr (Union Theta)
proof -
  have  $\forall$  theta'  $\in$  Retr ' Theta. theta'  $\subseteq$  Retr (Union Theta)
  using assms unfolding mono-def by blast
  thus ?thesis by blast
qed
```

```
lemma mono-Un:
assumes mono Retr
shows Retr theta Un Retr theta'  $\subseteq$  Retr (theta Un theta')
using assms unfolding mono-def
by (metis Un-least Un-upper1 Un-upper2)
```

```
lemma sym-Union:
assumes  $\bigwedge$ theta. theta  $\in$  Theta  $\implies$  sym theta
shows sym (Union Theta)
using assms unfolding sym-def by blast
```

```
lemma sym-Un:
assumes sym theta1 and sym theta2
```

**shows**  $\text{sym } (\text{theta1 Un theta2})$   
**using**  $\text{assms sym-Union[of \{theta1,theta2\}]}$  **by auto**

**lemma** *simul-Union*:  
**assumes**  $\text{mono Retr}$   
**and**  $\bigwedge \text{theta. theta} \in \text{Theta} \implies \text{simul Retr theta}$   
**shows**  $\text{simul Retr } (\text{Union Theta})$   
**proof** –  
  **have**  $\forall \text{theta} \in \text{Theta. theta} \subseteq \text{Retr theta}$   
  **using**  $\text{assms unfolding simul-def}$  **by blast**  
  **hence**  $\text{Union Theta} \subseteq \text{Union } (\text{Retr ` Theta})$  **by blast**  
  **also have**  $\dots \subseteq \text{Retr } (\text{Union Theta})$  **using**  $\text{mono-Union assms unfolding mono-def}$   
**by auto**  
  **finally have**  $\text{Union Theta} \subseteq \text{Retr } (\text{Union Theta})$  .  
  **thus**  $?thesis$  **unfolding simul-def** **by simp**  
**qed**

**lemma** *simul-Un*:  
**assumes**  $\text{mono Retr}$  **and**  $\text{simul Retr theta1}$  **and**  $\text{simul Retr theta2}$   
**shows**  $\text{simul Retr } (\text{theta1 Un theta2})$   
**using**  $\text{assms simul-Union[of Retr \{theta1,theta2\}]}$  **by auto**

**lemma** *bisim-Union*:  
**assumes**  $\text{mono Retr}$  **and**  $\bigwedge \text{theta. theta} \in \text{Theta} \implies \text{bisim Retr theta}$   
**shows**  $\text{bisim Retr } (\text{Union Theta})$   
**using**  $\text{assms unfolding bisim-def}$   
**using**  $\text{sym-Union simul-Union}$  **by blast**

**lemma** *bisim-Un*:  
**assumes**  $\text{mono Retr}$  **and**  $\text{bisim Retr theta1}$  **and**  $\text{bisim Retr theta2}$   
**shows**  $\text{bisim Retr } (\text{theta1 Un theta2})$   
**using**  $\text{assms bisim-Union[of Retr \{theta1,theta2\}]}$  **by auto**

**definition** *bis where*  
 $\text{bis Retr} \equiv \text{Union } \{\text{theta. bisim Retr theta}\}$

**lemma** *bisim-bis[simp]*:  
**assumes**  $\text{mono Retr}$   
**shows**  $\text{bisim Retr } (\text{bis Retr})$   
**using**  $\text{assms unfolding mono-def}$   
**by** ( $\text{metis CollectD assms bis-def bisim-Union}$ )

**corollary** *sym-bis[simp]*:  $\text{mono Retr} \implies \text{sym } (\text{bis Retr})$   
**and** *simul-bis[simp]*:  $\text{mono Retr} \implies \text{simul Retr } (\text{bis Retr})$   
**using** *bisim-bis* **unfolding bisim-def** **by auto**

**lemma** *bis-raw-coind*:  
**assumes**  $\text{mono Retr}$  **and**  $\text{sym theta}$  **and**  $\text{theta} \subseteq \text{Retr theta}$   
**shows**  $\text{theta} \subseteq \text{bis Retr}$

using *assms unfolding mono-def bis-def bisim-def simul-def* by *blast*

**lemma** *bis-prefix[simp]*:  
assumes *mono Retr*  
shows  $\text{bis Retr} \subseteq \text{Retr} (\text{bis Retr})$   
by (*metis assms bisim-bis bisim-def simul-def*)

**lemma** *bis-coind*:  
assumes \*: *mono Retr* and *sym theta* and \*\*:  $\text{theta} \subseteq \text{Retr} (\text{theta Un} (\text{bis Retr}))$   
shows  $\text{theta} \subseteq \text{bis Retr}$   
**proof** –  
  let *?theta'* =  $\text{theta Un} (\text{bis Retr})$   
  have *sym ?theta'* by (*metis Bisim.sym-Un sym-bis assms*)  
  moreover have  $\text{?theta}' \subseteq \text{Retr} \text{?theta}'$   
  by (*metis assms mono-Un Un-least bis-prefix le-supI2 subset-trans*)  
  ultimately show *?thesis* using \* *bis-raw-coind* by *blast*  
**qed**

**lemma** *bis-coind2*:  
assumes \*: *mono Retr* and  
\*\*:  $\text{theta} \subseteq \text{Retr} (\text{theta Un} (\text{bis Retr}))$  and  
\*\*\*:  $\text{theta}^{-1} \subseteq \text{Retr} ((\text{theta}^{-1}) \text{Un} (\text{bis Retr}))$   
shows  $\text{theta} \subseteq \text{bis Retr}$   
**proof** –  
  let *?th* =  $\text{theta Un} \text{theta}^{-1}$   
  have *sym ?th* by (*metis sym-Un-converse*)  
  moreover  
  {have  $\text{?th} \subseteq \text{Retr} (\text{theta Un} (\text{bis Retr})) \text{Un} \text{Retr} (\text{theta}^{-1} \text{Un} (\text{bis Retr}))$   
    using \*\* \*\*\* *Un-mono* by *blast*  
    also have  $\dots \subseteq \text{Retr} ((\text{theta Un} (\text{bis Retr})) \text{Un} (\text{theta}^{-1} \text{Un} (\text{bis Retr})))$   
    using \* *mono-Un* by *blast*  
    also have  $\dots = \text{Retr} (\text{?th Un} (\text{bis Retr}))$  by (*metis Un-assoc Un-commute Un-left-absorb*)  
    finally have  $\text{?th} \subseteq \text{Retr} (\text{?th Un} (\text{bis Retr}))$ .  
  }  
  ultimately have  $\text{?th} \subseteq \text{bis Retr}$  using *assms bis-coind* by *blast*  
  thus *?thesis* by *blast*  
**qed**

**lemma** *bis-raw-coind2*:  
assumes \*: *mono Retr* and  
\*\*:  $\text{theta} \subseteq \text{Retr} \text{theta}$  and  
\*\*\*:  $\text{theta}^{-1} \subseteq \text{Retr} (\text{theta}^{-1})$   
shows  $\text{theta} \subseteq \text{bis Retr}$   
**proof** –  
  have  $\text{theta} \subseteq \text{Retr} (\text{theta Un} (\text{bis Retr}))$  and  
   $\text{theta}^{-1} \subseteq \text{Retr} ((\text{theta}^{-1}) \text{Un} (\text{bis Retr}))$   
  using *assms* by (*metis mono-Un le-supI1 subset-trans*) +  
  thus *?thesis* using \* *bis-coind2* by *blast*

qed

**lemma** *mono-bis*:

**assumes** *mono Retr1 and mono Retr2*

**and**  $\bigwedge$  *theta. Retr1 theta  $\subseteq$  Retr2 theta*

**shows** *bis Retr1  $\subseteq$  bis Retr2*

**by** (*metis assms bis-prefix bis-raw-coind subset-trans sym-bis*)

end

### 3 The programming language and its semantics

**theory** *Language-Semantics* **imports** *Interface* **begin**

#### 3.1 Syntax and operational semantics

**datatype** (*'test, 'atom*) *com* =  
  *Atm 'atom* |  
  *Seq ('test, 'atom) com ('test, 'atom) com*  
  (*- ;; - [60, 61] 60*) |  
  *If 'test ('test, 'atom) com ('test, 'atom) com*  
  (*((if -/ then -/ else -) [0, 0, 61] 61)*) |  
  *While 'test ('test, 'atom) com*  
  (*((while -/ do -) [0, 61] 61)*) |  
  *Par ('test, 'atom) com ('test, 'atom) com*  
  (*- | - [60, 61] 60*)

**locale** *PL* =

**fixes**

*tval :: 'test  $\Rightarrow$  'state  $\Rightarrow$  bool and*

*aval :: 'atom  $\Rightarrow$  'state  $\Rightarrow$  'state*

**context** *PL*

**begin**

Conventions and notations: – suffixes: “C” for “Continuation”, “T” for “termination” – prefix: “M” for multistep – *tst*, *tst'* are tests – *atm*, *atm'* are atoms (atomic commands) – *s*, *s'*, *t*, *t'* are states – *c*, *c'*, *d*, *d'* are commands – *cf*, *cf'* are configurations, i.e., pairs command-state

**inductive** *transT* ::

(*('test, 'atom) com \* 'state  $\Rightarrow$  'state  $\Rightarrow$  bool*

(**infix**  $\rightarrow$  *t 55*)

**where**

*Atm[simp]:*

(*Atm atm, s*)  $\rightarrow$  *t aval atm s*

| *WhileFalse[simp]:*

$\sim$  *tval tst s  $\Longrightarrow$  (While tst c, s)  $\rightarrow$  t s*

**lemmas** *trans-Atm* = *Atm*  
**lemmas** *trans-WhileFalse* = *WhileFalse*

**inductive** *transC* ::  
 (('test,'atom)com \* 'state) ⇒ (('test,'atom)com \* 'state) ⇒ bool  
 (**infix** →*c* 55)  
**and** *MtransC* ::  
 (('test,'atom)com \* 'state) ⇒ (('test,'atom)com \* 'state) ⇒ bool  
 (**infix** →\**c* 55)  
**where**

*SeqC*[*simp*]:  
 (c1, s) →*c* (c1', s') ⇒ (c1 ;; c2, s) →*c* (c1' ;; c2, s')  
 | *SeqT*[*simp*]:  
 (c1, s) →*t* s' ⇒ (c1 ;; c2, s) →*c* (c2, s')  
 | *IfTrue*[*simp*]:  
 tval *tst* s ⇒ (If *tst* c1 c2, s) →*c* (c1, s)  
 | *IfFalse*[*simp*]:  
 ~ tval *tst* s ⇒ (If *tst* c1 c2, s) →*c* (c2, s)  
 | *WhileTrue*[*simp*]:  
 tval *tst* s ⇒ (While *tst* c, s) →*c* (c ;; (While *tst* c), s)  
  
 | *ParCL*[*simp*]:  
 (c1, s) →*c* (c1', s') ⇒ (Par c1 c2, s) →*c* (Par c1' c2, s')  
 | *ParCR*[*simp*]:  
 (c2, s) →*c* (c2', s') ⇒ (Par c1 c2, s) →*c* (Par c1 c2', s')  
 | *ParTL*[*simp*]:  
 (c1, s) →*t* s' ⇒ (Par c1 c2, s) →*c* (c2, s')  
 | *ParTR*[*simp*]:  
 (c2, s) →*t* s' ⇒ (Par c1 c2, s) →*c* (c1, s')  
 | *Refl*:  
 (c,s) →\**c* (c,s)  
 | *Step*:  
 [(c,s) →\**c* (c',s'); (c',s') →*c* (c'',s'')] ⇒ (c,s) →\**c* (c'',s')

**lemmas** *trans-SeqC* = *SeqC* **lemmas** *trans-SeqT* = *SeqT*  
**lemmas** *trans-IfTrue* = *IfTrue* **lemmas** *trans-IfFalse* = *IfFalse*  
**lemmas** *trans-WhileTrue* = *WhileTrue*  
**lemmas** *trans-ParCL* = *ParCL* **lemmas** *trans-ParCR* = *ParCR*  
**lemmas** *trans-ParTL* = *ParTL* **lemmas** *trans-ParTR* = *ParTR*  
**lemmas** *trans-Refl* = *Refl* **lemmas** *trans-Step* = *Step*

**lemma** *MtransC-Refl*[*simp*]: *cf* →\**c* *cf*  
**using** *trans-Refl* **by**(*cases cf*, *simp*)

**lemmas** *transC-induct* = *transC-MtransC.inducts*(1)  
 [*split-format*(*complete*),  
**where** ?*P2.0* = λ c s c' s'. *True*]



**lemmas** *MtransC-induct-temp* = *transC-MtransC.inducts(2)*[*split-format(complete)*]

**inductive** *MtransT* ::  
((*'test,'atom*)*com* \* *'state*)  $\Rightarrow$  *'state*  $\Rightarrow$  *bool*  
(**infix**  $\rightarrow^*t$  55)

**where**

*StepT*:  
[[*cf*  $\rightarrow^*c$  *cf'*; *cf'*  $\rightarrow t$  *s'*]]  $\Longrightarrow$  *cf*  $\rightarrow^*t$  *s''*

**lemma** *MtransC-rtranclp-transC*:

*MtransC* = *transC*  $\hat{**}$

**proof** –

{**fix** *c s c' s'*  
  **have** (*c,s*)  $\rightarrow^*c$  (*c',s'*)  $\Longrightarrow$  *transC*  $\hat{**}$  (*c,s*) (*c',s'*)  
  **apply**(*rule MtransC-induct-temp*[*of - - c s c' s' \lambda c s c' s'. True*]) **by auto**  
}  
**moreover**  
{**fix** *c s c' s'*  
  **have** *transC*  $\hat{**}$  (*c,s*) (*c',s'*)  $\Longrightarrow$  (*c,s*)  $\rightarrow^*c$  (*c',s'*)  
  **apply**(*erule rtranclp.induct*) **using trans-Step** **by auto**  
}  
**ultimately show** *?thesis*  
**apply** – **apply**(*rule ext, rule ext*) **by auto**  
**qed**

**lemma** *transC-MtransC[simp]*:

**assumes** *cf*  $\rightarrow c$  *cf'*

**shows** *cf*  $\rightarrow^*c$  *cf'*

**using** *assms unfolding MtransC-rtranclp-transC* **by blast**

**lemma** *MtransC-Trans*:

**assumes** *cf*  $\rightarrow^*c$  *cf'* **and** *cf'*  $\rightarrow^*c$  *cf''*

**shows** *cf*  $\rightarrow^*c$  *cf''*

**using** *assms rtranclp-trans*[*of transC cf cf' cf''*]

**unfolding** *MtransC-rtranclp-transC* **by blast**

**lemma** *MtransC-StepC*:

**assumes** \*: *cf*  $\rightarrow^*c$  *cf'* **and** \*\*: *cf'*  $\rightarrow c$  *cf''*

**shows** *cf*  $\rightarrow^*c$  *cf''*

**proof** –

**have** *cf'*  $\rightarrow^*c$  *cf''* **using** \*\* **by simp**

**thus** *?thesis* **using** \* *MtransC-Trans* **by blast**

**qed**

**lemma** *MtransC-induct*[*consumes 1, case-names Refl Trans*]:

**assumes** *cf*  $\rightarrow^*c$  *cf'*

**and**  $\bigwedge cf. \textit{phi}$  *cf* *cf'*

**and**

$\bigwedge cf$  *cf'* *cf''*.

$\llbracket cf \rightarrow^* c cf'; \text{phi } cf cf'; cf' \rightarrow c cf'' \rrbracket$   
 $\implies \text{phi } cf cf''$   
**shows**  $\text{phi } cf cf'$   
**using** *assms unfolding MtransC-rtranclp-transC*  
**using** *rtranclp.induct[of transC cf cf'] by blast*

**lemma** *MtransC-induct2*[*consumes 1, case-names Refl Trans, induct pred: MtransC*]:  
**assumes**  $(c,s) \rightarrow^* c (c',s')$   
**and**  $\bigwedge c s. \text{phi } c s c s$   
**and**  
 $\bigwedge c s c' s' c'' s''.$   
 $\llbracket (c,s) \rightarrow^* c (c',s'); \text{phi } c s c' s'; (c',s') \rightarrow c (c'',s'') \rrbracket$   
 $\implies \text{phi } c s c'' s''$   
**shows**  $\text{phi } c s c' s'$   
**using** *assms*  
*MtransC-induct*[*of (c,s) (c',s')  $\lambda(c,s) (c',s'). \text{phi } c s c' s'$  by blast*

**lemma** *transT-MtransT*[*simp*]:  
**assumes**  $cf \rightarrow t s'$   
**shows**  $cf \rightarrow^* t s'$   
**by** (*metis PL.MtransC-Refl PL.MtransT.intros assms*)

**lemma** *MtransC-MtransT*:  
**assumes**  $cf \rightarrow^* c cf'$  **and**  $cf' \rightarrow^* t cf''$   
**shows**  $cf \rightarrow^* t cf''$   
**by** (*metis MtransT.cases PL.MtransC-Trans PL.MtransT.intros assms*)

**lemma** *transC-MtransT*[*simp*]:  
**assumes**  $cf \rightarrow c cf'$  **and**  $cf' \rightarrow^* t s''$   
**shows**  $cf \rightarrow^* t s''$   
**by** (*metis PL.MtransC-MtransT assms(1) assms(2) transC-MtransC*)

Inversion rules, nchotomies and such:

**lemma** *Atm-transC-simp*[*simp*]:  
 $\sim (Atm \text{ atm}, s) \rightarrow c cf$   
**apply** *clarify apply*(*erule transC.cases*) **by** *auto*

**lemma** *Atm-transC-invert*[*elim!*]:  
**assumes**  $(Atm \text{ atm}, s) \rightarrow c cf$   
**shows**  $\text{phi}$   
**using** *assms by simp*

**lemma** *Atm-transT-invert*[*elim!*]:  
**assumes**  $(Atm \text{ atm}, s) \rightarrow t s'$   
**and**  $s' = \text{aval atm } s \implies \text{phi}$   
**shows**  $\text{phi}$   
**using** *assms apply - apply*(*erule transT.cases*) **by** *auto*

**lemma** *Seq-transC-invert*[*elim!*]:

**assumes**  $(c1 ;; c2, s) \rightarrow c (c', s')$   
**and**  $\bigwedge c1'. \llbracket (c1, s) \rightarrow c (c1', s'); c' = c1' ;; c2 \rrbracket \Longrightarrow phi$   
**and**  $\llbracket (c1, s) \rightarrow t s'; c' = c2 \rrbracket \Longrightarrow phi$   
**shows**  $phi$   
**using** *assms* **apply** – **apply**(*erule transC.cases*) **by** *auto*

**lemma** *Seq-transT-invert[simp]*:  
 $\sim (c1 ;; c2, s) \rightarrow t s'$   
**apply** *clarify* **apply**(*erule transT.cases*) **by** *auto*

**lemma** *If-transC-invert[elim!]*:  
**assumes**  $(If\ tst\ c1\ c2, s) \rightarrow c (c', s')$   
**and**  $\llbracket tval\ tst\ s; c' = c1; s' = s \rrbracket \Longrightarrow phi$   
**and**  $\llbracket \sim\ tval\ tst\ s; c' = c2; s' = s \rrbracket \Longrightarrow phi$   
**shows**  $phi$   
**using** *assms* **apply** – **apply**(*erule transC.cases*) **by** *auto*

**lemma** *If-transT-simp[simp]*:  
 $\sim (If\ b\ c1\ c2, s) \rightarrow t s'$   
**apply** *clarify* **apply**(*erule transT.cases*) **by** *auto*

**lemma** *If-transT-invert[elim!]*:  
**assumes**  $(If\ b\ c1\ c2, s) \rightarrow t s'$   
**shows**  $phi$   
**using** *assms* **by** *simp*

**lemma** *While-transC-invert[elim]*:  
**assumes**  $(While\ tst\ c1, s) \rightarrow c (c', s')$   
**and**  $\llbracket tval\ tst\ s; c' = c1 ;; (While\ tst\ c1); s' = s \rrbracket \Longrightarrow phi$   
**shows**  $phi$   
**using** *assms* **apply** – **apply**(*erule transC.cases*) **by** *auto*

**lemma** *While-transT-invert[elim!]*:  
**assumes**  $(While\ tst\ c1, s) \rightarrow t s'$   
**and**  $\llbracket \sim\ tval\ tst\ s; s' = s \rrbracket \Longrightarrow phi$   
**shows**  $phi$   
**using** *assms* **apply** – **apply**(*erule transT.cases*) **by** *blast+*

**lemma** *Par-transC-invert[elim]*:  
**assumes**  $(Par\ c1\ c2, s) \rightarrow c (c', s')$   
**and**  $\bigwedge c1'. \llbracket (c1, s) \rightarrow c (c1', s'); c' = Par\ c1'\ c2 \rrbracket \Longrightarrow phi$   
**and**  $\llbracket (c1, s) \rightarrow t s'; c' = c2 \rrbracket \Longrightarrow phi$   
**and**  $\bigwedge c2'. \llbracket (c2, s) \rightarrow c (c2', s'); c' = Par\ c1\ c2' \rrbracket \Longrightarrow phi$   
**and**  $\llbracket (c2, s) \rightarrow t s'; c' = c1 \rrbracket \Longrightarrow phi$   
**shows**  $phi$   
**using** *assms* **apply** – **apply**(*erule transC.cases*) **by** *auto*

**lemma** *Par-transT-simp[simp]*:  
 $\sim (Par\ c1\ c2, s) \rightarrow t s'$

**apply** *clarify* **apply**(*erule transT.cases*) **by** *auto*

**lemma** *Par-transT-invert[elim!]*:  
**assumes** (*Par c1 c2, s*)  $\rightarrow t s'$   
**shows** *phi*  
**using** *assms* **by** *simp*

**lemma** *trans-nchotomy*:  
 $(\exists c' s'. (c, s) \rightarrow c (c', s')) \vee$   
 $(\exists s'. (c, s) \rightarrow t s')$

**proof** –

**let** *?phiC* =  $\lambda c. \exists c' s'. (c, s) \rightarrow c (c', s')$   
**let** *?phiT* =  $\lambda c. \exists s'. (c, s) \rightarrow t s'$   
**let** *?phi* =  $\lambda c. ?phiC c \vee ?phiT c$   
**show** *?phi c*  
**apply**(*induct c*)  
**by**(*metis Atm, metis SeqC SeqT, metis IfFalse IfTrue,*  
*metis WhileFalse WhileTrue,*  
*metis ParCL ParCR ParTL ParTR*)

**qed**

**corollary** *trans-invert*:  
**assumes**  
 $\bigwedge c' s'. (c, s) \rightarrow c (c', s') \implies phi$   
**and**  $\bigwedge s'. (c, s) \rightarrow t s' \implies phi$   
**shows** *phi*  
**using** *assms trans-nchotomy* **by** *blast*

**lemma** *not-transC-transT*:  
 $\llbracket cf \rightarrow c cf'; cf \rightarrow t s' \rrbracket \implies phi$   
**apply**(*erule transC.cases*) **by** *auto*

**lemmas** *MtransT-invert* = *MtransT.cases*

**lemma** *MtransT-invert2*:  
**assumes** (*c, s*)  $\rightarrow * t s''$   
**and**  $\bigwedge c' s'. \llbracket (c, s) \rightarrow * c (c', s'); (c', s') \rightarrow t s'' \rrbracket \implies phi$   
**shows** *phi*  
**using** *assms* **apply** – **apply**(*erule MtransT.cases*) **by** *auto*

**lemma** *Seq-MtransC-invert[elim!]*:  
**assumes** (*c1 ;; c2, s*)  $\rightarrow * c (d', t')$   
**and**  $\bigwedge c1'. \llbracket (c1, s) \rightarrow * c (c1', t'); d' = c1' ;; c2 \rrbracket \implies phi$   
**and**  $\bigwedge s'. \llbracket (c1, s) \rightarrow * t s'; (c2, s') \rightarrow * c (d', t') \rrbracket \implies phi$   
**shows** *phi*  
**proof** –  
  **{fix** *c*  
    **have** (*c, s*)  $\rightarrow * c (d', t') \implies$   
     $\forall c1 c2.$

```

c = c1 ;; c2 →
(∃ c1'. (c1, s) →*c (c1', t') ∧ d' = c1' ;; c2) ∨
(∃ s'. (c1, s) →*t s' ∧ (c2, s') →*c (d', t'))
apply(erule MtransC-induct2) proof(tactic ‹mauto-no-simp-tac @{context}›)
fix c s d' t' d'' t'' c1 c2
assume
∀ c1 c2. c = c1 ;; c2 →
  (∃ c1'. (c1, s) →*c (c1', t') ∧ d' = c1' ;; c2) ∨
  (∃ s'. (c1, s) →*t s' ∧ (c2, s') →*c (d', t'))
and 1: (d', t') →c (d'', t'') and c = c1 ;; c2
hence IH:
(∃ c1'. (c1, s) →*c (c1', t') ∧ d' = c1' ;; c2) ∨
(∃ s'. (c1, s) →*t s' ∧ (c2, s') →*c (d', t')) by simp
show (∃ c1''. (c1, s) →*c (c1'', t'') ∧ d'' = c1'' ;; c2) ∨
  (∃ s''. (c1, s) →*t s'' ∧ (c2, s'') →*c (d'', t''))
proof –
  {fix c1' assume 2: (c1, s) →*c (c1', t') and d': d' = c1' ;; c2
  have ?thesis
  using 1 unfolding d' apply – proof(erule Seq-transC-invert)
    fix c1'' assume (c1', t') →c (c1'', t'') and d'': d'' = c1'' ;; c2
    hence (c1, s) →*c (c1'', t'') using 2 MtransC-StepC by blast
    thus ?thesis using d'' by blast
  next
  assume (c1', t') →t t'' and d'': d'' = c2
  hence (c1, s) →*t t'' using 2 MtransT.StepT by blast
  thus ?thesis unfolding d'' by auto
  qed
  }
moreover
{fix s' assume 2: (c1, s) →*t s' and (c2, s') →*c (d', t')
hence (c2, s') →*c (d'', t'') using 1 MtransC-StepC by blast
hence ?thesis using 2 by blast
}
ultimately show ?thesis using IH by blast
qed
qed (metis PL.MtransC-Refl)
}
thus ?thesis using assms by blast
qed

lemma Seq-MtransT-invert[elim!]:
assumes *: (c1 ;; c2, s) →*t s''
and **: ∧ s'. [(c1, s) →*t s'; (c2, s') →*t s''] ⇒ phi
shows phi
proof –
obtain d' t' where 1: (c1 ;; c2, s) →*c (d', t') and 2: (d', t') →t s''
using * apply – apply(erule MtransT-invert2) by auto
show ?thesis
using 1 apply – proof(erule Seq-MtransC-invert)

```

```

    fix c1' assume d' = c1' ;; c2
    hence False using 2 by simp
    thus ?thesis by simp
  next
    fix s' assume 3: (c1, s) →*t s' and (c2, s') →*c (d', t')
    hence (c2, s') →*t s'' using 2 MtransT.StepT by blast
    thus ?thesis using 3 ** by blast
  qed
qed

```

Direct rules for the multi-step relations

```

lemma Seq-MtransC[simp]:
  assumes (c1, s) →*c (c1', s')
  shows (c1 ;; c2, s) →*c (c1' ;; c2, s')
  using assms apply – apply(erule MtransC-induct2)
  apply simp by (metis MtransC-StepC SeqC)

```

```

lemma Seq-MtransT-MtransC[simp]:
  assumes (c1, s) →*t s'
  shows (c1 ;; c2, s) →*c (c2, s')
  using assms apply – apply(erule MtransT-invert)
  by (metis MtransC-StepC MtransT-invert2 PL.SeqT PL.Seq-MtransC assms)

```

```

lemma ParCL-MtransC[simp]:
  assumes (c1, s) →*c (c1', s')
  shows (Par c1 c2, s) →*c (Par c1' c2, s')
  using assms apply – apply(erule MtransC-induct2)
  apply simp by (metis MtransC-StepC ParCL)

```

```

lemma ParCR-MtransC[simp]:
  assumes (c2, s) →*c (c2', s')
  shows (Par c1 c2, s) →*c (Par c1 c2', s')
  using assms apply – apply(erule MtransC-induct2)
  apply simp by (metis MtransC-StepC ParCR)

```

```

lemma ParTL-MtransC[simp]:
  assumes (c1, s) →*t s'
  shows (Par c1 c2, s) →*c (c2, s')
  using assms apply – apply(erule MtransT-invert)
  by (metis MtransC-StepC MtransT-invert2 PL.ParTL ParCL-MtransC assms)

```

```

lemma ParTR-MtransC[simp]:
  assumes (c2, s) →*t s'
  shows (Par c1 c2, s) →*c (c1, s')
  using assms apply – apply(erule MtransT-invert)
  by (metis MtransC-StepC MtransT-invert2 PL.ParTR ParCR-MtransC assms)

```

## 3.2 Sublanguages

**fun** *noWhile* **where**

*noWhile* (*Atm atm*) = *True*  
| *noWhile* (*c1 ;; c2*) = (*noWhile c1*  $\wedge$  *noWhile c2*)  
| *noWhile* (*If b c1 c2*) = (*noWhile c1*  $\wedge$  *noWhile c2*)  
| *noWhile* (*While b c*) = *False*  
| *noWhile* (*Par c1 c2*) = (*noWhile c1*  $\wedge$  *noWhile c2*)

**fun** *seq* **where**

*seq* (*Atm atm*) = *True*  
| *seq* (*c1 ;; c2*) = (*seq c1*  $\wedge$  *seq c2*)  
| *seq* (*If b c1 c2*) = (*seq c1*  $\wedge$  *seq c2*)  
| *seq* (*While b c*) = *seq c*  
| *seq* (*Par c1 c2*) = *False*

**lemma** *noWhile-transC*:

**assumes** *noWhile c* **and**  $(c,s) \rightarrow c (c',s')$

**shows** *noWhile c'*

**proof** –

**have**  $(c,s) \rightarrow c (c',s') \implies \text{noWhile } c \longrightarrow \text{noWhile } c'$   
**by**(*erule transC-induct, auto*)  
**thus** *?thesis* **using** *assms* **by** *simp*

**qed**

**lemma** *seq-transC*:

**assumes** *seq c* **and**  $(c,s) \rightarrow c (c',s')$

**shows** *seq c'*

**proof** –

**have**  $(c,s) \rightarrow c (c',s') \implies \text{seq } c \longrightarrow \text{seq } c'$   
**by**(*erule transC-induct, auto*)  
**thus** *?thesis* **using** *assms* **by** *simp*

**qed**

**abbreviation** *wfP-on* **where**

*wfP-on phi A*  $\equiv$  *wfP* ( $\lambda a b. a \in A \wedge b \in A \wedge \text{phi } a b$ )

**fun** *numSt* **where**

*numSt* (*Atm atm*) = *Suc 0*  
| *numSt* (*c1 ;; c2*) = *numSt c1* + *numSt c2*  
| *numSt* (*If b c1 c2*) = *1* + *max* (*numSt c1*) (*numSt c2*)  
| *numSt* (*Par c1 c2*) = *numSt c1* + *numSt c2*

**lemma** *numSt-gt-0[simp]*:

*noWhile c*  $\implies$  *numSt c* > 0

**by**(*induct c, auto*)

**lemma** *numSt-transC*:

**assumes** *noWhile* *c* **and**  $(c,s) \rightarrow c (c',s')$   
**shows**  $\text{numSt } c' < \text{numSt } c$   
**using** *assms* **apply** – **apply**(*induct* *c* *arbitrary*: *c'*) **by** *auto*

**corollary** *wfP-tranC-noWhile*:

*wfP*  $(\lambda (c',s') (c,s). \text{noWhile } c \wedge (c,s) \rightarrow c (c',s'))$

**proof** –

**let**  $?K = \{((c',s'),(c,s)). \text{noWhile } c \wedge (c,s) \rightarrow c (c',s')\}$   
**have**  $?K \leq \text{inv-image } \{(m,n). m < n\} (\lambda(c,s). \text{numSt } c)$  **by**(*auto simp add*:  
*numSt-transC*)  
**hence** *wf*  $?K$  **using** *wf-less wf-subset*[of -  $?K$ ] **by** *blast*  
**thus** *?thesis* **unfolding** *wfP-def*  
**by** (*metis CollectD Collect-mem-eq Compl-eq Compl-iff double-complement*)  
**qed**

**lemma** *noWhile-MtransT*:

**assumes** *noWhile* *c*

**shows**  $\exists s'. (c,s) \rightarrow^* t s'$

**proof** –

**have**  $\text{noWhile } c \longrightarrow (\forall s. \exists s'. (c,s) \rightarrow^* t s')$   
**apply**(*rule measure-induct*[of *numSt*]) **proof** *clarify*  
**fix** *c* :: (*'test,atom*) *com* **and** *s*  
**assume** *IH*:  $\forall c'. \text{numSt } c' < \text{numSt } c \longrightarrow \text{noWhile } c' \longrightarrow$   
 $(\forall s'. \exists s''. (c', s') \rightarrow^* t s'')$  **and** *c*: *noWhile* *c*  
**show**  $\exists s''. (c, s) \rightarrow^* t s''$   
**proof**(*rule trans-invert*[of *c s*])  
**fix** *c' s'* **assume** *cs*:  $(c, s) \rightarrow c (c', s')$   
**hence**  $\text{numSt } c' < \text{numSt } c$  **and** *noWhile* *c'*  
**using** *numSt-transC noWhile-transC* *c* **by** *blast+*  
**then obtain** *s''* **where**  $(c', s') \rightarrow^* t s''$  **using** *IH* **by** *blast*  
**hence**  $(c, s) \rightarrow^* t s''$  **using** *cs* **by** *simp*  
**thus** *?thesis* **by** *blast*  
**next**  
**fix** *s'* **assume**  $(c, s) \rightarrow t s'$   
**hence**  $(c, s) \rightarrow^* t s'$  **by** *simp*  
**thus** *?thesis* **by** *blast*  
**qed**  
**qed**  
**thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**coinductive** *mayDiverge* **where**

*intro*:

$\llbracket (c,s) \rightarrow c (c',s') \wedge \text{mayDiverge } c' s' \rrbracket$   
 $\implies \text{mayDiverge } c s$

Coinduction for may-diverge :



**lemma** *mayDiverge-coind*[*consumes 1, case-names Hyp, induct pred: mayDiverge*]:  
**assumes** \*: *phi c s* **and**  
 \*\*:  $\bigwedge c s. \text{phi } c s \implies$   
        $\exists c' s'. (c,s) \rightarrow c (c',s') \wedge (\text{phi } c' s' \vee \text{mayDiverge } c' s')$   
**shows** *mayDiverge c s*  
**using** \* **apply**(*elim mayDiverge.coinduct*) **using** \*\* **by** *auto*

**lemma** *mayDiverge-raw-coind*[*consumes 1, case-names Hyp*]:  
**assumes** \*: *phi c s* **and**  
 \*\*:  $\bigwedge c s. \text{phi } c s \implies$   
        $\exists c' s'. (c,s) \rightarrow c (c',s') \wedge \text{phi } c' s'$   
**shows** *mayDiverge c s*  
**using** \* **apply** *induct* **using** \*\* **by** *blast*

May-diverge versus transition:

**lemma** *mayDiverge-transC*:  
**assumes** *mayDiverge c s*  
**shows**  $\exists c' s'. (c,s) \rightarrow c (c',s') \wedge \text{mayDiverge } c' s'$   
**using** *assms* **by** (*elim mayDiverge.cases*) *blast*

**lemma** *transC-mayDiverge*:  
**assumes**  $(c,s) \rightarrow c (c',s')$  **and** *mayDiverge c' s'*  
**shows** *mayDiverge c s*  
**using** *assms* **by** (*metis mayDiverge.intro*)

**lemma** *mayDiverge-not-transT*:  
**assumes** *mayDiverge c s*  
**shows**  $\neg (c,s) \rightarrow t s'$   
**by** (*metis assms mayDiverge-transC not-transC-transT*)

**lemma** *MtransC-mayDiverge*:  
**assumes**  $(c,s) \rightarrow * c (c',s')$  **and** *mayDiverge c' s'*  
**shows** *mayDiverge c s*  
**using** *assms transC-mayDiverge* **by** (*induct*) *auto*

**lemma** *not-MtransT-mayDiverge*:  
**assumes**  $\bigwedge s'. \neg (c,s) \rightarrow * t s'$   
**shows** *mayDiverge c s*  
**proof** –  
**have**  $\forall s'. \neg (c,s) \rightarrow * t s' \implies ?thesis$   
**proof** (*induct rule: mayDiverge-raw-coind*)  
**case** (*Hyp c s*)  
**hence**  $\forall s''. \neg (c, s) \rightarrow t s''$  **by** (*metis transT-MtransT*)  
**then obtain** *c' s'* **where**  $1: (c,s) \rightarrow c (c',s')$  **by** (*metis trans-invert*)  
**hence**  $\forall s''. \neg (c', s') \rightarrow * t s''$  **using** *Hyp 1* **by** (*metis transC-MtransT*)  
**thus** *?case* **using** *1* **by** *blast*  
**qed**  
**thus** *?thesis* **using** *assms* **by** *simp*  
**qed**

**lemma** *not-mayDiverge-Atm[simp]*:  
 $\neg \text{mayDiverge } (\text{Atm } \text{atm}) \ s$   
**by** (*metis Atm-transC-invert mayDiverge.simps*)

**lemma** *mayDiverge-Seq-L*:  
**assumes**  $\text{mayDiverge } c1 \ s$   
**shows**  $\text{mayDiverge } (c1 \ ;\ ;\ c2) \ s$   
**proof** –  
 {**fix**  $c$   
**assume**  $\exists \ c1 \ c2. \ c = c1 \ ;\ ;\ c2 \wedge \text{mayDiverge } c1 \ s$   
**hence**  $\text{mayDiverge } c \ s$   
**proof** (*induct rule: mayDiverge-raw-coind*)  
**case** (*Hyp*  $c \ s$ )  
**then obtain**  $c1 \ c2$  **where**  $c = c1 \ ;\ ;\ c2$   
**and**  $\text{mayDiverge } c1 \ s$  **by** *blast*  
**then obtain**  $c1' \ s'$  **where**  $(c1, s) \rightarrow_c (c1', s')$   
**and**  $\text{mayDiverge } c1' \ s'$  **by** (*metis mayDiverge-transC*)  
**thus** *?case* **using**  $c \ \text{SeqC}$  **by** *metis*  
**qed**  
 }  
**thus** *?thesis* **using** *assms* **by** *auto*  
**qed**

**lemma** *mayDiverge-Seq-R*:  
**assumes**  $c1: (c1, s) \rightarrow^* t \ s'$  **and**  $c2: \text{mayDiverge } c2 \ s'$   
**shows**  $\text{mayDiverge } (c1 \ ;\ ;\ c2) \ s$   
**proof** –  
**have**  $(c1 \ ;\ ;\ c2, s) \rightarrow^* c (c2, s')$   
**using**  $c1$  **by** (*metis Seq-MtransT-MtransC*)  
**thus** *?thesis* **by** (*metis MtransC-mayDiverge c2*)  
**qed**

**lemma** *mayDiverge-If-L*:  
**assumes**  $\text{tval } \text{tst } s$  **and**  $\text{mayDiverge } c1 \ s$   
**shows**  $\text{mayDiverge } (\text{If } \text{tst } c1 \ c2) \ s$   
**using** *assms IfTrue transC-mayDiverge* **by** *metis*

**lemma** *mayDiverge-If-R*:  
**assumes**  $\neg \text{tval } \text{tst } s$  **and**  $\text{mayDiverge } c2 \ s$   
**shows**  $\text{mayDiverge } (\text{If } \text{tst } c1 \ c2) \ s$   
**using** *assms IfFalse transC-mayDiverge* **by** *metis*

**lemma** *mayDiverge-If*:  
**assumes**  $\text{mayDiverge } c1 \ s$  **and**  $\text{mayDiverge } c2 \ s$   
**shows**  $\text{mayDiverge } (\text{If } \text{tst } c1 \ c2) \ s$   
**using** *assms mayDiverge-If-L mayDiverge-If-R*  
**by** (*cases tval tst s*) *auto*

**lemma** *mayDiverge-Par-L*:  
**assumes** *mayDiverge c1 s*  
**shows** *mayDiverge (Par c1 c2) s*  
**proof** –  
    {fix *c*  
    **assume**  $\exists c1\ c2. c = \text{Par } c1\ c2 \wedge \text{mayDiverge } c1\ s$   
    **hence** *mayDiverge c s*  
    **proof** (*induct rule: mayDiverge-raw-coind*)  
    **case** (*Hyp c s*)  
    **then obtain** *c1 c2* **where**  $c: c = \text{Par } c1\ c2$   
    **and** *mayDiverge c1 s* **by** *blast*  
    **then obtain** *c1' s'* **where**  $(c1, s) \rightarrow_c (c1', s')$   
    **and** *mayDiverge c1' s'* **by** (*metis mayDiverge-transC*)  
    **thus** ?*case using c ParCL by metis*  
    **qed**  
    }  
    **thus** ?*thesis using assms by auto*  
**qed**

**lemma** *mayDiverge-Par-R*:  
**assumes** *mayDiverge c2 s*  
**shows** *mayDiverge (Par c1 c2) s*  
**proof** –  
    {fix *c*  
    **assume**  $\exists c1\ c2. c = \text{Par } c1\ c2 \wedge \text{mayDiverge } c2\ s$   
    **hence** *mayDiverge c s*  
    **proof** (*induct rule: mayDiverge-raw-coind*)  
    **case** (*Hyp c s*)  
    **then obtain** *c1 c2* **where**  $c: c = \text{Par } c1\ c2$   
    **and** *mayDiverge c2 s* **by** *blast*  
    **then obtain** *c2' s'* **where**  $(c2, s) \rightarrow_c (c2', s')$   
    **and** *mayDiverge c2' s'* **by** (*metis mayDiverge-transC*)  
    **thus** ?*case using c ParCR by metis*  
    **qed**  
    }  
    **thus** ?*thesis using assms by auto*  
**qed**

**definition** *mustT* **where**  
*mustT c s*  $\equiv \neg \text{mayDiverge } c\ s$

**lemma** *mustT-transC*:  
**assumes** *mustT c s* **and**  $(c, s) \rightarrow_c (c', s')$   
**shows** *mustT c' s'*  
**using** *assms intro unfolding mustT-def by blast*

**lemma** *transT-not-mustT*:  
**assumes**  $(c, s) \rightarrow_t s'$

**shows**  $mustT\ c\ s$   
**by** (*metis* *assms* *mayDiverge-not-transT* *mustT-def*)

**lemma** *mustT-MtransC*:  
**assumes**  $mustT\ c\ s$  **and**  $(c,s) \rightarrow^*c\ (c',s')$   
**shows**  $mustT\ c'\ s'$   
**proof** –  
  **have**  $(c,s) \rightarrow^*c\ (c',s') \implies mustT\ c\ s \longrightarrow mustT\ c'\ s'$   
  **apply**(*erule* *MtransC-induct2*) **using** *mustT-transC* **by** *blast+*  
  **thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**lemma** *mustT-MtransT*:  
**assumes**  $mustT\ c\ s$   
**shows**  $\exists\ s'.\ (c,s) \rightarrow^*t\ s'$   
**using** *assms* *not-MtransT-mayDiverge* **unfolding** *mustT-def* **by** *blast*

**lemma** *mustT-Atm[simp]*:  
 $mustT\ (Atm\ atm)\ s$   
**by** (*metis* *not-mayDiverge-Atm* *mustT-def*)

**lemma** *mustT-Seq-L*:  
**assumes**  $mustT\ (c1\ ;;\ c2)\ s$   
**shows**  $mustT\ c1\ s$   
**by** (*metis* *PL.mayDiverge-Seq-L* *assms* *mustT-def*)

**lemma** *mustT-Seq-R*:  
**assumes**  $mustT\ (c1\ ;;\ c2)\ s$  **and**  $(c1,\ s) \rightarrow^*t\ s'$   
**shows**  $mustT\ c2\ s'$   
**by** (*metis* *Seq-MtransT-MtransC* *mustT-MtransC* *assms*)

**lemma** *mustT-If-L*:  
**assumes**  $tval\ tst\ s$  **and**  $mustT\ (If\ tst\ c1\ c2)\ s$   
**shows**  $mustT\ c1\ s$   
**by** (*metis* *assms* *trans-IfTrue* *mustT-transC*)

**lemma** *mustT-If-R*:  
**assumes**  $\neg\ tval\ tst\ s$  **and**  $mustT\ (If\ tst\ c1\ c2)\ s$   
**shows**  $mustT\ c2\ s$   
**by** (*metis* *assms* *trans-IfFalse* *mustT-transC*)

**lemma** *mustT-If*:  
**assumes**  $mustT\ (If\ tst\ c1\ c2)\ s$   
**shows**  $mustT\ c1\ s \vee mustT\ c2\ s$   
**by** (*metis* *assms* *mustT-If-L* *mustT-If-R*)

**lemma** *mustT-Par-L*:  
**assumes**  $mustT\ (Par\ c1\ c2)\ s$   
**shows**  $mustT\ c1\ s$

by (metis assms mayDiverge-Par-L mustT-def)

**lemma** *mustT-Par-R*:

**assumes** *mustT* (Par *c1 c2*) *s*

**shows** *mustT c2 s*

by (metis assms mayDiverge-Par-R mustT-def)

**definition** *determOn* **where**

*determOn phi r*  $\equiv$

$\forall a b b'. phi a \wedge r a b \wedge r a b' \longrightarrow b = b'$

**lemma** *determOn-seq-transT*:

*determOn* ( $\lambda(c,s). seq c$ ) *transT*

**proof** –

{**fix** *c s s1' s2'*

**have**  $seq c \wedge (c,s) \rightarrow t s1' \wedge (c,s) \rightarrow t s2' \longrightarrow s1' = s2'$

**apply**(*induct c arbitrary: s1' s2'*) **by** *auto*

}

**thus** *?thesis* **unfolding** *determOn-def* **by** *fastforce*

**qed**

**end**

**end**

## 4 During-execution security

**theory** *During-Execution*

**imports** *Bisim Language-Semantics* **begin**

### 4.1 Basic setting

**locale** *PL-Indis = PL tval aval*

**for**

*tval* :: *'test*  $\Rightarrow$  *'state*  $\Rightarrow$  *bool* **and**

*aval* :: *'atom*  $\Rightarrow$  *'state*  $\Rightarrow$  *'state*

+

**fixes**

*indis* :: *'state* *rel*

**assumes**

*equiv-indis: equiv UNIV indis*

**context** *PL-Indis*

**begin**

**abbreviation** *indisAbbrev* (**infix**  $\approx$  50)

**where**  $s1 \approx s2 \equiv (s1, s2) \in indis$

**definition** *indisE* (**infix**  $\approx_e$  50) **where**

$se1 \approx_e se2 \equiv$

*case* (*se1, se2*) *of*

(*Inl s1, Inl s2*)  $\Rightarrow s1 \approx s2$

| (*Inr err1, Inr err2*)  $\Rightarrow err1 = err2$

**lemma** *refl-indis*: *refl indis*

**and** *trans-indis*: *trans indis*

**and** *sym-indis*: *sym indis*

**using** *equiv-indis* **unfolding** *equiv-def* **by** *auto*

**lemma** *indis-refl[intro]*:  $s \approx s$

**using** *refl-indis* **unfolding** *refl-on-def* **by** *simp*

**lemma** *indis-trans*:  $\llbracket s \approx s'; s' \approx s'' \rrbracket \Longrightarrow s \approx s''$

**using** *trans-indis* **unfolding** *trans-def* **by** *blast*

**lemma** *indis-sym*:  $s \approx s' \Longrightarrow s' \approx s$

**using** *sym-indis* **unfolding** *sym-def* **by** *blast*

## 4.2 Compatibility and discreteness

**definition** *compatTst* **where**

*compatTst tst*  $\equiv$

$\forall s t. s \approx t \longrightarrow tval\ tst\ s = tval\ tst\ t$

**definition** *compatAtm* **where**

*compatAtm atm*  $\equiv$

$\forall s t. s \approx t \longrightarrow aval\ atm\ s \approx aval\ atm\ t$

**definition** *presAtm* **where**

*presAtm atm*  $\equiv$

$\forall s. s \approx aval\ atm\ s$

**coinductive** *discr* **where**

*intro*:

$\llbracket \bigwedge s c' s'. (c, s) \rightarrow c (c', s') \Longrightarrow s \approx s' \wedge discr\ c';$

$\bigwedge s s'. (c, s) \rightarrow t s' \Longrightarrow s \approx s' \rrbracket$

$\Longrightarrow discr\ c$

**lemma** *presAtm-compatAtm[simp]*:

**assumes** *presAtm atm*

**shows** *compatAtm atm*

**using** *assms* **unfolding** *compatAtm-def*

by (*metis presAtm-def indis-sym indis-trans*)

Coinduction for discreteness:

**lemma** *discr-coind*:

**assumes** \*: *phi c* and

\*\* :  $\bigwedge c s c' s'. \llbracket \text{phi } c; (c,s) \rightarrow c (c',s') \rrbracket \implies s \approx s' \wedge (\text{phi } c' \vee \text{discr } c')$  and

\*\*\* :  $\bigwedge c s s'. \llbracket \text{phi } c; (c,s) \rightarrow t s \rrbracket \implies s \approx s'$

**shows** *discr c*

**using** \* **apply** – **apply**(*erule discr.coinduct*) **using** \*\* \*\*\* **by** *auto*

**lemma** *discr-raw-coind*:

**assumes** \*: *phi c* and

\*\* :  $\bigwedge c s c' s'. \llbracket \text{phi } c; (c,s) \rightarrow c (c',s') \rrbracket \implies s \approx s' \wedge \text{phi } c'$  and

\*\*\* :  $\bigwedge c s s'. \llbracket \text{phi } c; (c,s) \rightarrow t s \rrbracket \implies s \approx s'$

**shows** *discr c*

**using** \* **apply** – **apply**(*erule discr-coind*) **using** \*\* \*\*\* **by** *blast+*

Discreteness versus transition:

**lemma** *discr-transC*:

**assumes** \*: *discr c* and \*\*:  $(c,s) \rightarrow c (c',s')$

**shows** *discr c'*

**using** \* **apply** – **apply**(*erule discr.cases*) **using** \*\* **by** *blast*

**lemma** *discr-MtransC*:

**assumes** *discr c* and  $(c,s) \rightarrow *c (c',s')$

**shows** *discr c'*

**proof** –

**have**  $(c,s) \rightarrow *c (c',s') \implies \text{discr } c \longrightarrow \text{discr } c'$

**apply**(*erule MtransC-induct2*) **using** *discr-transC* **by** *blast+*

**thus** ?thesis **using** *assms* **by** *blast*

**qed**

**lemma** *discr-transC-indis*:

**assumes** \*: *discr c* and \*\*:  $(c,s) \rightarrow c (c',s')$

**shows**  $s \approx s'$

**using** \* **apply** – **apply**(*erule discr.cases*) **using** \*\* **by** *blast*

**lemma** *discr-MtransC-indis*:

**assumes** *discr c* and  $(c,s) \rightarrow *c (c',s')$

**shows**  $s \approx s'$

**proof** –

**have**  $(c,s) \rightarrow *c (c',s') \implies \text{discr } c \longrightarrow s \approx s'$

**apply**(*erule MtransC-induct2*)

**apply** (*metis indis-refl*)

**by** (*metis discr.cases discr-MtransC indis-trans*)

**thus** ?thesis **using** *assms* **by** *blast*

**qed**

**lemma** *discr-transT*:

**assumes** \*: *discr* *c* **and** \*\*:  $(c,s) \rightarrow t s'$   
**shows**  $s \approx s'$   
**using** \* **apply** – **apply**(*erule* *discr.cases*) **using** \*\* **by** *blast*

**lemma** *discr-MtransT*:

**assumes** \*: *discr* *c* **and** \*\*:  $(c,s) \rightarrow *t s'$   
**shows**  $s \approx s'$

**proof** –

**obtain**  $d' t'$  **where**

*cs*:  $(c,s) \rightarrow *c (d',t')$  **and**  $d't'$ :  $(d',t') \rightarrow t s'$

**using** \*\* **by**(*rule* *MtransT-invert2*)

**hence**  $s \approx t'$  **using** \* *discr-MtransC-indis* **by** *blast*

**moreover**

{**have** *discr*  $d'$  **using** *cs* \* *discr-MtransC* **by** *blast*

**hence**  $t' \approx s'$  **using**  $d't'$  *discr-transT* **by** *blast*

}

**ultimately show** *?thesis* **using** *indis-trans* **by** *blast*

**qed**

### 4.3 Terminating-interactive discreteness

**coinductive** *discr0* **where**

*intro*:

$\llbracket \bigwedge s c' s'. \llbracket \text{mustT } c s; (c,s) \rightarrow c (c',s') \rrbracket \implies s \approx s' \wedge \text{discr0 } c' \rrbracket$   
 $\bigwedge s s'. \llbracket \text{mustT } c s; (c,s) \rightarrow t s' \rrbracket \implies s \approx s' \rrbracket$   
 $\implies \text{discr0 } c$

Coinduction for 0-discreteness:

**lemma** *discr0-coind*[*consumes 1, case-names Cont Term, induct pred: discr0*]:

**assumes** \*: *phi* *c* **and**

\*\* :  $\bigwedge c s c' s'$ .

$\llbracket \text{mustT } c s; \text{phi } c; (c,s) \rightarrow c (c',s') \rrbracket \implies$

$s \approx s' \wedge (\text{phi } c' \vee \text{discr0 } c')$  **and**

\*\*\* :  $\bigwedge c s s'. \llbracket \text{mustT } c s; \text{phi } c; (c,s) \rightarrow t s' \rrbracket \implies s \approx s'$

**shows** *discr0* *c*

**using** \* **apply** – **apply**(*erule* *discr0.coinduct*) **using** \*\* \*\*\* **by** *auto*

**lemma** *discr0-raw-coind*[*consumes 1, case-names Cont Term*]:

**assumes** \*: *phi* *c* **and**

\*\* :  $\bigwedge c s c' s'. \llbracket \text{mustT } c s; \text{phi } c; (c,s) \rightarrow c (c',s') \rrbracket \implies s \approx s' \wedge \text{phi } c'$  **and**

\*\*\* :  $\bigwedge c s s'. \llbracket \text{mustT } c s; \text{phi } c; (c,s) \rightarrow t s' \rrbracket \implies s \approx s'$

**shows** *discr0* *c*

**using** \* **apply** – **apply**(*erule* *discr0-coind*) **using** \*\* \*\*\* **by** *blast+*

0-Discreteness versus transition:

**lemma** *discr0-transC*:

**assumes** \*: *discr0* *c* **and** \*\*:  $\text{mustT } c s (c,s) \rightarrow c (c',s')$

**shows** *discr0*  $c'$

**using** \* **apply** – **apply**(*erule* *discr0.cases*) **using** \*\* **by** *blast*



**lemma** *discr0-MtransC*:  
**assumes** *discr0 c* **and** *mustT c s (c,s) →\*c (c',s')*  
**shows** *discr0 c'*  
**proof** –  
    **have**  $(c,s) \rightarrow^*c (c',s') \implies \text{mustT } c \ s \ \wedge \ \text{discr0 } c \longrightarrow \text{discr0 } c'$   
    **apply**(*erule MtransC-induct2*) **using** *discr0-transC mustT-MtransC*  
    **by** *blast+*  
    **thus** *?thesis using assms by blast*  
**qed**

**lemma** *discr0-transC-indis*:  
**assumes** \*: *discr0 c* **and** \*\*: *mustT c s (c,s) →c (c',s')*  
**shows**  $s \approx s'$   
**using** \* **apply** – **apply**(*erule discr0.cases*) **using** \*\* **by** *blast*

**lemma** *discr0-MtransC-indis*:  
**assumes** *discr0 c* **and** *mustT c s (c,s) →\*c (c',s')*  
**shows**  $s \approx s'$   
**proof** –  
    **have**  $(c,s) \rightarrow^*c (c',s') \implies \text{mustT } c \ s \ \wedge \ \text{discr0 } c \longrightarrow s \approx s'$   
    **apply**(*erule MtransC-induct2*)  
    **apply** (*metis indis-refl*)  
    **by** (*metis discr0-MtransC discr0-transC-indis indis-trans mustT-MtransC*)  
    **thus** *?thesis using assms by blast*  
**qed**

**lemma** *discr0-transT*:  
**assumes** \*: *discr0 c* **and** \*\*: *mustT c s (c,s) →t s'*  
**shows**  $s \approx s'$   
**using** \* **apply** – **apply**(*erule discr0.cases*) **using** \*\* **by** *blast*

**lemma** *discr0-MtransT*:  
**assumes** \*: *discr0 c* **and** \*\*\*: *mustT c s* **and** \*\*:  $(c,s) \rightarrow^*t s'$   
**shows**  $s \approx s'$   
**proof** –  
    **obtain**  $d' \ t'$  **where**  
    *cs: (c,s) →\*c (d',t')* **and**  $d't': (d',t') \rightarrow t \ s'$   
    **using** \*\* **by**(*rule MtransT-invert2*)  
    **hence**  $s \approx t'$  **using** \* *discr0-MtransC-indis* \*\*\* **by** *blast*  
    **moreover**  
    {**have** *discr0 d'* **using** *cs* \* *discr0-MtransC* \*\*\* **by** *blast*  
    **hence**  $t' \approx s'$   
    **using** \*\*\* **by** (*metis mustT-MtransC cs d't' discr0-transT*)  
    }  
    **ultimately show** *?thesis using indis-trans by blast*  
**qed**

**lemma** *discr-discr0[simp]*:  $\text{discr } c \implies \text{discr0 } c$

by (induct rule: *discr0-coind*)  
 (metis *discr-transC discr-transC-indis discr-transT*)+

#### 4.4 Self-isomorphism

**coinductive** *siso* where

*intro*:

$\llbracket \bigwedge s c' s'. (c,s) \rightarrow c (c',s') \implies \textit{siso } c' \rrbracket$ ;  
 $\bigwedge s t c' s'. \llbracket s \approx t; (c,s) \rightarrow c (c',s') \rrbracket \implies \exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$ ;  
 $\bigwedge s t s'. \llbracket s \approx t; (c,s) \rightarrow t s' \rrbracket \implies \exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$   
 $\implies \textit{siso } c$

Coinduction for self-isomorphism:

**lemma** *siso-coind*:

**assumes** \*: *phi c* and

\*\* :  $\bigwedge c s c' s'. \llbracket \textit{phi } c; (c,s) \rightarrow c (c',s') \rrbracket \implies \textit{phi } c' \vee \textit{siso } c'$  and

\*\*\* :  $\bigwedge c s t c' s'. \llbracket \textit{phi } c; s \approx t; (c,s) \rightarrow c (c',s') \rrbracket \implies \exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$

and

\*\*\*\* :  $\bigwedge c s t s'. \llbracket \textit{phi } c; s \approx t; (c,s) \rightarrow t s' \rrbracket \implies \exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

**shows** *siso c*

**using** \* **apply** – **apply**(*erule siso.coinduct*) **using** \*\* \*\*\* \*\*\*\* **by** *auto*

**lemma** *siso-raw-coind*:

**assumes** \*: *phi c* and

\*\* :  $\bigwedge c s c' s'. \llbracket \textit{phi } c; (c,s) \rightarrow c (c',s') \rrbracket \implies \textit{phi } c'$  and

\*\*\* :  $\bigwedge c s t c' s'. \llbracket \textit{phi } c; s \approx t; (c,s) \rightarrow c (c',s') \rrbracket \implies \exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$

and

\*\*\*\* :  $\bigwedge c s t s'. \llbracket \textit{phi } c; s \approx t; (c,s) \rightarrow t s' \rrbracket \implies \exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

**shows** *siso c*

**using** \* **apply** – **apply**(*erule siso-coind*) **using** \*\* \*\*\* \*\*\*\* **by** *blast+*

Self-Isomorphism versus transition:

**lemma** *siso-transC*:

**assumes** \*: *siso c* and \*\*:  $(c,s) \rightarrow c (c',s')$

**shows** *siso c'*

**using** \* **apply** – **apply**(*erule siso.cases*) **using** \*\* **by** *blast*

**lemma** *siso-MtransC*:

**assumes** *siso c* and  $(c,s) \rightarrow *c (c',s')$

**shows** *siso c'*

**proof**–

have  $(c,s) \rightarrow *c (c',s') \implies \textit{siso } c \longrightarrow \textit{siso } c'$

**apply**(*erule MtransC-induct2*) **using** *siso-transC* **by** *blast+*

**thus** *?thesis* **using** *assms* **by** *blast*

**qed**

**lemma** *siso-transC-indis*:

**assumes** \*: *siso c* and \*\*:  $(c,s) \rightarrow c (c',s')$  and \*\*\*:  $s \approx t$

**shows**  $\exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$

using \* apply – apply(erule *siso.cases*) using \*\* \*\*\* by blast

lemma *siso-transT*:

assumes \*: *siso c* and \*\*:  $(c,s) \rightarrow t s'$  and \*\*\*:  $s \approx t$

shows  $\exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

using \* apply – apply(erule *siso.cases*) using \*\* \*\*\* by blast

## 4.5 MustT-interactive self-isomorphism

coinductive *siso0* where

intro:

$\llbracket \bigwedge s c' s'. \llbracket \text{mustT } c s; (c,s) \rightarrow c (c',s') \rrbracket \implies \text{siso0 } c' \rrbracket$

$\bigwedge s t c' s'.$

$\llbracket \text{mustT } c s; \text{mustT } c t; s \approx t; (c,s) \rightarrow c (c',s') \rrbracket \implies$

$\exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$ ;

$\bigwedge s t s'.$

$\llbracket \text{mustT } c s; \text{mustT } c t; s \approx t; (c,s) \rightarrow t s' \rrbracket \implies$

$\exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

$\implies \text{siso0 } c$

Coinduction for self-isomorphism:

lemma *siso0-coind*[*consumes 1, case-names Indef Cont Term, induct pred: discr0*]:

assumes \*: *phi c* and

\*\* :  $\bigwedge c s c' s'. \llbracket \text{phi } c; \text{mustT } c s; (c,s) \rightarrow c (c',s') \rrbracket \implies \text{phi } c' \vee \text{siso0 } c'$  and

\*\*\* :  $\bigwedge c s t c' s'.$

$\llbracket \text{phi } c; \text{mustT } c s; \text{mustT } c t; s \approx t; (c,s) \rightarrow c (c',s') \rrbracket \implies$

$\exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$  and

\*\*\*\* :  $\bigwedge c s t s'.$

$\llbracket \text{mustT } c s; \text{mustT } c t; \text{phi } c; s \approx t; (c,s) \rightarrow t s' \rrbracket \implies$

$\exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

shows *siso0 c*

using \* apply – apply(erule *siso0.coinduct*) using \*\* \*\*\* \*\*\*\* by auto

lemma *siso0-raw-coind*[*consumes 1, case-names Indef Cont Term*]:

assumes \*: *phi c* and

\*\* :  $\bigwedge c s c' s'. \llbracket \text{phi } c; \text{mustT } c s; (c,s) \rightarrow c (c',s') \rrbracket \implies \text{phi } c'$  and

\*\*\* :  $\bigwedge c s t c' s'.$

$\llbracket \text{phi } c; \text{mustT } c s; \text{mustT } c t; s \approx t; (c,s) \rightarrow c (c',s') \rrbracket \implies$

$\exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$  and

\*\*\*\* :  $\bigwedge c s t s'.$

$\llbracket \text{phi } c; \text{mustT } c s; \text{mustT } c t; s \approx t; (c,s) \rightarrow t s' \rrbracket \implies$

$\exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

shows *siso0 c*

using \* apply – apply(erule *siso0-coind*) using \*\* \*\*\* \*\*\*\* by blast+

Self-Isomorphism versus transition:

lemma *siso0-transC*:

assumes \*: *siso0 c* and \*\*:  $\text{mustT } c s (c,s) \rightarrow c (c',s')$

shows *siso0 c'*

using \* apply – apply(*erule* *siso0.cases*) using \*\* by blast

**lemma** *siso0-MtransC*:

assumes *siso0* *c* and *mustT* *c* *s* and  $(c,s) \rightarrow^* c (c',s')$

shows *siso0* *c'*

**proof** –

have  $(c,s) \rightarrow^* c (c',s') \implies \text{mustT } c \ s \ \wedge \ \text{siso0 } c \longrightarrow \text{siso0 } c'$

apply(*erule* *MtransC-induct2*) using *siso0-transC* *mustT-MtransC* *siso0-transC*

by *blast+*

thus *?thesis* using *assms* by *blast*

qed

**lemma** *siso0-transC-indis*:

assumes \*: *siso0* *c*

and \*\*: *mustT* *c* *s* *mustT* *c* *t*  $(c,s) \rightarrow c (c',s')$

and \*\*\*:  $s \approx t$

shows  $\exists t'. s' \approx t' \wedge (c,t) \rightarrow c (c',t')$

using \* apply – apply(*erule* *siso0.cases*) using \*\* \*\*\* by *blast*

**lemma** *siso0-transT*:

assumes \*: *siso0* *c*

and \*\*: *mustT* *c* *s* *mustT* *c* *t*  $(c,s) \rightarrow t s'$

and \*\*\*:  $s \approx t$

shows  $\exists t'. s' \approx t' \wedge (c,t) \rightarrow t t'$

using \* apply – apply(*erule* *siso0.cases*) using \*\* \*\*\* by *blast*

## 4.6 Notions of bisimilarity

Matchers:

**definition** *matchC-C* where

*matchC-C* *theta* *c* *d*  $\equiv$

$\forall s \ t \ c' \ s'.$

$s \approx t \wedge (c,s) \rightarrow c (c',s')$

$\longrightarrow$

$(\exists d' \ t'. (d,t) \rightarrow c (d',t') \wedge s' \approx t' \wedge (c',d') \in \text{theta})$

**definition** *matchC-ZOC* where

*matchC-ZOC* *theta* *c* *d*  $\equiv$

$\forall s \ t \ c' \ s'.$

$s \approx t \wedge (c,s) \rightarrow c (c',s')$

$\longrightarrow$

$(s' \approx t \wedge (c',d) \in \text{theta})$

$\vee$

$(\exists d' \ t'. (d,t) \rightarrow c (d',t') \wedge s' \approx t' \wedge (c',d') \in \text{theta})$

**definition** *matchC-ZO* where

*matchC-ZO* *theta* *c* *d*  $\equiv$

$\forall s \ t \ c' \ s'.$

$s \approx t \wedge (c,s) \rightarrow c (c',s')$

$$\begin{aligned}
&\longrightarrow \\
&(s' \approx t \wedge (c', d) \in \text{theta}) \\
&\vee \\
&(\exists d' t'. (d, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{theta}) \\
&\vee \\
&(\exists t'. (d, t) \rightarrow t t' \wedge s' \approx t' \wedge \text{discr } c')
\end{aligned}$$

**definition** *matchT-T* where

$$\begin{aligned}
\text{matchT-T } c \ d &\equiv \\
\forall s \ t \ s'. & \\
s \approx t \wedge (c, s) &\rightarrow t \ s' \\
\longrightarrow & \\
(\exists t'. (d, t) \rightarrow t \ t' \wedge s' \approx t') &
\end{aligned}$$

**definition** *matchT-ZO* where

$$\begin{aligned}
\text{matchT-ZO } c \ d &\equiv \\
\forall s \ t \ s'. & \\
s \approx t \wedge (c, s) &\rightarrow t \ s' \\
\longrightarrow & \\
(s' \approx t \wedge \text{discr } d) & \\
\vee & \\
(\exists d' t'. (d, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge \text{discr } d') & \\
\vee & \\
(\exists t'. (d, t) \rightarrow t \ t' \wedge s' \approx t') &
\end{aligned}$$

**definition** *matchC-MC* where

$$\begin{aligned}
\text{matchC-MC } \text{theta} \ c \ d &\equiv \\
\forall s \ t \ c' \ s'. & \\
s \approx t \wedge (c, s) &\rightarrow c (c', s') \\
\longrightarrow & \\
(\exists d' t'. (d, t) \rightarrow *c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{theta}) &
\end{aligned}$$

**definition** *matchC-TMC* where

$$\begin{aligned}
\text{matchC-TMC } \text{theta} \ c \ d &\equiv \\
\forall s \ t \ c' \ s'. & \\
\text{mustT } c \ s \wedge \text{mustT } d \ t \wedge s \approx t \wedge (c, s) &\rightarrow c (c', s') \\
\longrightarrow & \\
(\exists d' t'. (d, t) \rightarrow *c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{theta}) &
\end{aligned}$$

**definition** *matchC-M* where

$$\begin{aligned}
\text{matchC-M } \text{theta} \ c \ d &\equiv \\
\forall s \ t \ c' \ s'. & \\
s \approx t \wedge (c, s) &\rightarrow c (c', s') \\
\longrightarrow & \\
(\exists d' t'. (d, t) \rightarrow *c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{theta}) & \\
\vee & \\
(\exists t'. (d, t) \rightarrow *t \ t' \wedge s' \approx t' \wedge \text{discr } c') &
\end{aligned}$$

**definition** *matchT-MT* where

$$\begin{aligned} \text{matchT-MT } c \ d &\equiv \\ \forall \ s \ t \ s'. & \\ s \approx t \wedge (c,s) \rightarrow t \ s' & \\ \longrightarrow & \\ (\exists \ t'. (d,t) \rightarrow *t \ t' \wedge s' \approx t') & \end{aligned}$$

**definition** *matchT-TMT* where

$$\begin{aligned} \text{matchT-TMT } c \ d &\equiv \\ \forall \ s \ t \ s'. & \\ \text{mustT } c \ s \wedge \text{mustT } d \ t \wedge s \approx t \wedge (c,s) \rightarrow t \ s' & \\ \longrightarrow & \\ (\exists \ t'. (d,t) \rightarrow *t \ t' \wedge s' \approx t') & \end{aligned}$$

**definition** *matchT-M* where

$$\begin{aligned} \text{matchT-M } c \ d &\equiv \\ \forall \ s \ t \ s'. & \\ s \approx t \wedge (c,s) \rightarrow t \ s' & \\ \longrightarrow & \\ (\exists \ d' \ t'. (d,t) \rightarrow *c \ (d',t') \wedge s' \approx t' \wedge \text{discr } d') & \\ \vee & \\ (\exists \ t'. (d,t) \rightarrow *t \ t' \wedge s' \approx t') & \end{aligned}$$

**lemmas** *match-defs* =

*matchC-C-def*  
*matchC-ZOC-def matchC-ZO-def*  
*matchT-T-def matchT-ZO-def*  
*matchC-MC-def matchC-M-def*  
*matchT-MT-def matchT-M-def*  
*matchC-TMC-def matchT-TMT-def*

**lemma** *matchC-C-def2*:

$$\begin{aligned} \text{matchC-C } \text{theta} \ d \ c &= \\ (\forall \ s \ t \ d' \ t'. & \\ s \approx t \wedge (d,t) \rightarrow c \ (d',t') & \\ \longrightarrow & \\ (\exists \ c' \ s'. (c,s) \rightarrow c \ (c',s') \wedge s' \approx t' \wedge (d',c') \in \text{theta})) & \end{aligned}$$

**unfolding** *matchC-C-def* using *indis-sym* by *blast*

**lemma** *matchC-ZOC-def2*:

$$\begin{aligned} \text{matchC-ZOC } \text{theta} \ d \ c &= \\ (\forall \ s \ t \ d' \ t'. & \\ s \approx t \wedge (d,t) \rightarrow c \ (d',t') & \\ \longrightarrow & \\ (s \approx t' \wedge (d',c) \in \text{theta}) & \\ \vee & \end{aligned}$$

$(\exists c' s'. (c,s) \rightarrow c (c',s') \wedge s' \approx t' \wedge (d',c') \in \text{theta}))$   
**unfolding** *matchC-ZOC-def* **using** *indis-sym* **by** *blast*

**lemma** *matchC-ZO-def2*:

*matchC-ZO* *theta* *d* *c* =

$(\forall s t d' t'.$

$s \approx t \wedge (d,t) \rightarrow c (d',t')$

$\rightarrow$

$(s \approx t' \wedge (d',c) \in \text{theta})$

$\vee$

$(\exists c' s'. (c,s) \rightarrow c (c',s') \wedge s' \approx t' \wedge (d',c') \in \text{theta})$

$\vee$

$(\exists s'. (c,s) \rightarrow t s' \wedge s' \approx t' \wedge \text{discr } d')$ )

**unfolding** *matchC-ZO-def* **using** *indis-sym* **by** *blast*

**lemma** *matchT-T-def2*:

*matchT-T* *d* *c* =

$(\forall s t t'.$

$s \approx t \wedge (d,t) \rightarrow t t'$

$\rightarrow$

$(\exists s'. (c,s) \rightarrow t s' \wedge s' \approx t')$ )

**unfolding** *matchT-T-def* **using** *indis-sym* **by** *blast*

**lemma** *matchT-ZO-def2*:

*matchT-ZO* *d* *c* =

$(\forall s t t'.$

$s \approx t \wedge (d,t) \rightarrow t t'$

$\rightarrow$

$(s \approx t' \wedge \text{discr } c)$

$\vee$

$(\exists c' s'. (c,s) \rightarrow c (c',s') \wedge s' \approx t' \wedge \text{discr } c')$

$\vee$

$(\exists s'. (c,s) \rightarrow t s' \wedge s' \approx t')$ )

**unfolding** *matchT-ZO-def* **using** *indis-sym* **by** *blast*

**lemma** *matchC-MC-def2*:

*matchC-MC* *theta* *d* *c* =

$(\forall s t d' t'.$

$s \approx t \wedge (d,t) \rightarrow c (d',t')$

$\rightarrow$

$(\exists c' s'. (c,s) \rightarrow *c (c',s') \wedge s' \approx t' \wedge (d',c') \in \text{theta}))$

**unfolding** *matchC-MC-def* **using** *indis-sym* **by** *blast*

**lemma** *matchC-TMC-def2*:

*matchC-TMC* *theta* *d* *c* =

$(\forall s t d' t'.$

$\text{mustT } c \ s \wedge \text{mustT } d \ t \wedge s \approx t \wedge (d,t) \rightarrow c (d',t')$

$\longrightarrow$   
 $(\exists c' s'. (c,s) \rightarrow *c (c',s') \wedge s' \approx t' \wedge (d',c') \in \text{theta}))$   
**unfolding** *matchC-TMC-def* **using** *indis-sym* **by** *blast*

**lemma** *matchC-M-def2*:  
 $\text{matchC-M } \text{theta } d \ c =$   
 $(\forall s \ t \ d' \ t'.$   
 $s \approx t \wedge (d,t) \rightarrow c (d',t')$   
 $\longrightarrow$   
 $(\exists c' s'. (c,s) \rightarrow *c (c',s') \wedge s' \approx t' \wedge (d',c') \in \text{theta}))$   
 $\vee$   
 $(\exists s'. (c,s) \rightarrow *t s' \wedge s' \approx t' \wedge \text{discr } d')$   
**unfolding** *matchC-M-def* **using** *indis-sym* **by** *blast*

**lemma** *matchT-MT-def2*:  
 $\text{matchT-MT } d \ c =$   
 $(\forall s \ t \ t'.$   
 $s \approx t \wedge (d,t) \rightarrow t \ t'$   
 $\longrightarrow$   
 $(\exists s'. (c,s) \rightarrow *t s' \wedge s' \approx t'))$   
**unfolding** *matchT-MT-def* **using** *indis-sym* **by** *blast*

**lemma** *matchT-TMT-def2*:  
 $\text{matchT-TMT } d \ c =$   
 $(\forall s \ t \ t'.$   
 $\text{mustT } c \ s \wedge \text{mustT } d \ t \wedge s \approx t \wedge (d,t) \rightarrow t \ t'$   
 $\longrightarrow$   
 $(\exists s'. (c,s) \rightarrow *t s' \wedge s' \approx t'))$   
**unfolding** *matchT-TMT-def* **using** *indis-sym* **by** *blast*

**lemma** *matchT-M-def2*:  
 $\text{matchT-M } d \ c =$   
 $(\forall s \ t \ t'.$   
 $s \approx t \wedge (d,t) \rightarrow t \ t'$   
 $\longrightarrow$   
 $(\exists c' s'. (c,s) \rightarrow *c (c',s') \wedge s' \approx t' \wedge \text{discr } c')$   
 $\vee$   
 $(\exists s'. (c,s) \rightarrow *t s' \wedge s' \approx t'))$   
**unfolding** *matchT-M-def* **using** *indis-sym* **by** *blast*

Retracts:

**definition** *Sretr* **where**  
 $\text{Sretr } \text{theta} \equiv$   
 $\{(c,d).$   
 $\text{matchC-C } \text{theta } c \ d \wedge$   
 $\text{matchT-T } c \ d\}$

**definition** *ZOretr* **where**



$ZOretr\ theta \equiv$   
 $\{(c,d).$   
 $\quad matchC-ZO\ theta\ c\ d \wedge$   
 $\quad matchT-ZO\ c\ d\}$

**definition**  $ZOretrT$  **where**  
 $ZOretrT\ theta \equiv$   
 $\{(c,d).$   
 $\quad matchC-ZOC\ theta\ c\ d \wedge$   
 $\quad matchT-T\ c\ d\}$

**definition**  $Wretr$  **where**  
 $Wretr\ theta \equiv$   
 $\{(c,d).$   
 $\quad matchC-M\ theta\ c\ d \wedge$   
 $\quad matchT-M\ c\ d\}$

**definition**  $WretrT$  **where**  
 $WretrT\ theta \equiv$   
 $\{(c,d).$   
 $\quad matchC-MC\ theta\ c\ d \wedge$   
 $\quad matchT-MT\ c\ d\}$

**definition**  $RetrT$  **where**  
 $RetrT\ theta \equiv$   
 $\{(c,d).$   
 $\quad matchC-TMC\ theta\ c\ d \wedge$   
 $\quad matchT-TMT\ c\ d\}$

**lemmas**  $Retr-defs =$   
 $Sretr-def$   
 $ZOretr-def\ ZOretrT-def$   
 $Wretr-def\ WretrT-def$   
 $RetrT-def$

The associated bisimilarity relations:

**definition**  $Sbis$  **where**  $Sbis \equiv bis\ Sretr$   
**definition**  $ZObis$  **where**  $ZObis \equiv bis\ ZOretr$   
**definition**  $ZObisT$  **where**  $ZObisT \equiv bis\ ZOretrT$   
**definition**  $Wbis$  **where**  $Wbis \equiv bis\ Wretr$   
**definition**  $WbisT$  **where**  $WbisT \equiv bis\ WretrT$   
**definition**  $BisT$  **where**  $BisT \equiv bis\ RetrT$

**lemmas**  $bis-defs =$   
 $Sbis-def$

*ZObis-def ZObisT-def*  
*Wbis-def WbisT-def*  
*BisT-def*

**abbreviation** *Sbis-abbrev* (**infix**  $\approx_s$  55) **where**  $c1 \approx_s c2 \equiv (c1, c2) \in Sbis$   
**abbreviation** *ZObis-abbrev* (**infix**  $\approx_{01}$  55) **where**  $c1 \approx_{01} c2 \equiv (c1, c2) \in ZObis$   
**abbreviation** *ZObisT-abbrev* (**infix**  $\approx_{01T}$  55) **where**  $c1 \approx_{01T} c2 \equiv (c1, c2) \in ZObisT$   
**abbreviation** *Wbis-abbrev* (**infix**  $\approx_w$  55) **where**  $c1 \approx_w c2 \equiv (c1, c2) \in Wbis$   
**abbreviation** *WbisT-abbrev* (**infix**  $\approx_{wT}$  55) **where**  $c1 \approx_{wT} c2 \equiv (c1, c2) \in WbisT$   
**abbreviation** *BisT-abbrev* (**infix**  $\approx_T$  55) **where**  $c1 \approx_T c2 \equiv (c1, c2) \in BisT$

**lemma** *mono-Retr*:  
*mono Sretr*  
*mono ZOretr mono ZOretrT*  
*mono Wretr mono WretrT*  
*mono RetrT*  
**unfolding** *mono-def Retr-defs match-defs* **by** *blast+*

**lemma** *Sbis-prefix*:  
*Sbis*  $\subseteq$  *Sretr Sbis*  
**unfolding** *Sbis-def* **using** *mono-Retr bis-prefix* **by** *blast*

**lemma** *Sbis-sym*: *sym Sbis*  
**unfolding** *Sbis-def* **using** *mono-Retr sym-bis* **by** *blast*

**lemma** *Sbis-Sym*:  $c \approx_s d \implies d \approx_s c$   
**using** *Sbis-sym* **unfolding** *sym-def* **by** *blast*

**lemma** *Sbis-converse*:  
 $((c, d) \in \text{theta}^{-1} \cup Sbis) = ((d, c) \in \text{theta} \cup Sbis)$   
**by** (*metis Sbis-sym converseI converse-Un converse-converse sym-conv-converse-eq*)

**lemma**  
*Sbis-matchC-C*:  $\bigwedge s t. c \approx_s d \implies \text{matchC-C } Sbis \ c \ d$   
**and**  
*Sbis-matchT-T*:  $\bigwedge c d. c \approx_s d \implies \text{matchT-T } c \ d$   
**using** *Sbis-prefix* **unfolding** *Sretr-def* **by** *auto*

**lemmas** *Sbis-step* = *Sbis-matchC-C Sbis-matchT-T*

**lemma**  
*Sbis-matchC-C-rev*:  $\bigwedge s t. s \approx_s t \implies \text{matchC-C } Sbis \ t \ s$   
**and**  
*Sbis-matchT-T-rev*:  $\bigwedge s t. s \approx_s t \implies \text{matchT-T } t \ s$   
**using** *Sbis-step Sbis-sym* **unfolding** *sym-def* **by** *blast+*

**lemmas** *Sbis-step-rev* = *Sbis-matchC-C-rev* *Sbis-matchT-T-rev*

**lemma** *Sbis-coind*:  
assumes *sym theta* and  $theta \subseteq Sretr (theta \cup Sbis)$   
shows  $theta \subseteq Sbis$   
using *assms mono-Retr bis-coind*  
unfolding *Sbis-def* by *blast*

**lemma** *Sbis-raw-coind*:  
assumes *sym theta* and  $theta \subseteq Sretr theta$   
shows  $theta \subseteq Sbis$   
using *assms mono-Retr bis-raw-coind*  
unfolding *Sbis-def* by *blast*

**lemma** *Sbis-coind2*:  
assumes  $theta \subseteq Sretr (theta \cup Sbis)$  and  
 $theta \hat{-}1 \subseteq Sretr ((theta \hat{-}1) \cup Sbis)$   
shows  $theta \subseteq Sbis$   
using *assms mono-Retr bis-coind2*  
unfolding *Sbis-def* by *blast*

**lemma** *Sbis-raw-coind2*:  
assumes  $theta \subseteq Sretr theta$  and  
 $theta \hat{-}1 \subseteq Sretr (theta \hat{-}1)$   
shows  $theta \subseteq Sbis$   
using *assms mono-Retr bis-raw-coind2*  
unfolding *Sbis-def* by *blast*

**lemma** *ZObis-prefix*:  
 $ZObis \subseteq ZOretr ZObis$   
unfolding *ZObis-def* using *mono-Retr bis-prefix* by *blast*

**lemma** *ZObis-sym*: *sym ZObis*  
unfolding *ZObis-def* using *mono-Retr sym-bis* by *blast*

**lemma** *ZObis-converse*:  
 $((c,d) \in theta \hat{-}1 \cup ZObis) = ((d,c) \in theta \cup ZObis)$   
by (*metis ZObis-sym converseI converse-Un converse-converse sym-conv-converse-eq*)

**lemma** *ZObis-Sym*:  $s \approx 01 t \implies t \approx 01 s$   
using *ZObis-sym* unfolding *sym-def* by *blast*

**lemma**  
*ZObis-matchC-ZO*:  $\bigwedge s t. s \approx 01 t \implies matchC-ZO ZObis s t$   
and  
*ZObis-matchT-ZO*:  $\bigwedge s t. s \approx 01 t \implies matchT-ZO s t$   
using *ZObis-prefix* unfolding *ZOretr-def* by *auto*

**lemmas**  $ZObis\text{-}step = ZObis\text{-}matchC\text{-}ZO\ ZObis\text{-}matchT\text{-}ZO$

**lemma**

$ZObis\text{-}matchC\text{-}ZO\text{-}rev: \bigwedge s\ t. s \approx_{01} t \implies matchC\text{-}ZO\ ZObis\ t\ s$

**and**

$ZObis\text{-}matchT\text{-}ZO\text{-}rev: \bigwedge s\ t. s \approx_{01} t \implies matchT\text{-}ZO\ t\ s$

**using**  $ZObis\text{-}step\ ZObis\text{-}sym$  **unfolding**  $sym\text{-}def$  **by**  $blast+$

**lemmas**  $ZObis\text{-}step\text{-}rev = ZObis\text{-}matchC\text{-}ZO\text{-}rev\ ZObis\text{-}matchT\text{-}ZO\text{-}rev$

**lemma**  $ZObis\text{-}coind$ :

**assumes**  $sym\ \theta$  **and**  $\theta \subseteq ZOrtr\ (\theta \cup ZObis)$

**shows**  $\theta \subseteq ZObis$

**using**  $assms\ mono\text{-}Retr\ bis\text{-}coind$

**unfolding**  $ZObis\text{-}def$  **by**  $blast$

**lemma**  $ZObis\text{-}raw\text{-}coind$ :

**assumes**  $sym\ \theta$  **and**  $\theta \subseteq ZOrtr\ \theta$

**shows**  $\theta \subseteq ZObis$

**using**  $assms\ mono\text{-}Retr\ bis\text{-}raw\text{-}coind$

**unfolding**  $ZObis\text{-}def$  **by**  $blast$

**lemma**  $ZObis\text{-}coind2$ :

**assumes**  $\theta \subseteq ZOrtr\ (\theta \cup ZObis)$  **and**

$\theta^{-1} \subseteq ZOrtr\ ((\theta^{-1}) \cup ZObis)$

**shows**  $\theta \subseteq ZObis$

**using**  $assms\ mono\text{-}Retr\ bis\text{-}coind2$

**unfolding**  $ZObis\text{-}def$  **by**  $blast$

**lemma**  $ZObis\text{-}raw\text{-}coind2$ :

**assumes**  $\theta \subseteq ZOrtr\ \theta$  **and**

$\theta^{-1} \subseteq ZOrtr\ (\theta^{-1})$

**shows**  $\theta \subseteq ZObis$

**using**  $assms\ mono\text{-}Retr\ bis\text{-}raw\text{-}coind2$

**unfolding**  $ZObis\text{-}def$  **by**  $blast$

**lemma**  $ZObisT\text{-}prefix$ :

$ZObisT \subseteq ZOrtrT\ ZObisT$

**unfolding**  $ZObisT\text{-}def$  **using**  $mono\text{-}Retr\ bis\text{-}prefix$  **by**  $blast$

**lemma**  $ZObisT\text{-}sym$ :  $sym\ ZObisT$

**unfolding**  $ZObisT\text{-}def$  **using**  $mono\text{-}Retr\ sym\text{-}bis$  **by**  $blast$

**lemma**  $ZObisT\text{-}Sym$ :  $s \approx_{01T} t \implies t \approx_{01T} s$

**using**  $ZObisT\text{-}sym$  **unfolding**  $sym\text{-}def$  **by**  $blast$

**lemma**  $ZObisT\text{-}converse$ :

$((c,d) \in \text{theta}^{\wedge-1} \cup \text{ZObisT}) = ((d,c) \in \text{theta} \cup \text{ZObisT})$   
**by** (*metis ZObisT-sym converseI converse-Un converse-converse sym-conv-converse-eq*)

**lemma**

*ZObisT-matchC-ZOC*:  $\bigwedge s t. s \approx 01T t \implies \text{matchC-ZOC } \text{ZObisT } s t$

**and**

*ZObisT-matchT-T*:  $\bigwedge s t. s \approx 01T t \implies \text{matchT-T } s t$

**using** *ZObisT-prefix unfolding ZOretrT-def* **by** *auto*

**lemmas** *ZObisT-step* = *ZObisT-matchC-ZOC ZObisT-matchT-T*

**lemma**

*ZObisT-matchC-ZOC-rev*:  $\bigwedge s t. s \approx 01T t \implies \text{matchC-ZOC } \text{ZObisT } t s$

**and**

*ZObisT-matchT-T-rev*:  $\bigwedge s t. s \approx 01T t \implies \text{matchT-T } t s$

**using** *ZObisT-step ZObisT-sym unfolding sym-def* **by** *blast+*

**lemmas** *ZObisT-step-rev* = *ZObisT-matchC-ZOC-rev ZObisT-matchT-T-rev*

**lemma** *ZObisT-coind*:

**assumes** *sym theta* **and**  $\text{theta} \subseteq \text{ZOretrT } (\text{theta} \cup \text{ZObisT})$

**shows**  $\text{theta} \subseteq \text{ZObisT}$

**using** *assms mono-Retr bis-coind*

**unfolding** *ZObisT-def* **by** *blast*

**lemma** *ZObisT-raw-coind*:

**assumes** *sym theta* **and**  $\text{theta} \subseteq \text{ZOretrT } \text{theta}$

**shows**  $\text{theta} \subseteq \text{ZObisT}$

**using** *assms mono-Retr bis-raw-coind*

**unfolding** *ZObisT-def* **by** *blast*

**lemma** *ZObisT-coind2*:

**assumes**  $\text{theta} \subseteq \text{ZOretrT } (\text{theta} \cup \text{ZObisT})$  **and**

$\text{theta}^{\wedge-1} \subseteq \text{ZOretrT } ((\text{theta}^{\wedge-1}) \cup \text{ZObisT})$

**shows**  $\text{theta} \subseteq \text{ZObisT}$

**using** *assms mono-Retr bis-coind2*

**unfolding** *ZObisT-def* **by** *blast*

**lemma** *ZObisT-raw-coind2*:

**assumes**  $\text{theta} \subseteq \text{ZOretrT } \text{theta}$  **and**

$\text{theta}^{\wedge-1} \subseteq \text{ZOretrT } (\text{theta}^{\wedge-1})$

**shows**  $\text{theta} \subseteq \text{ZObisT}$

**using** *assms mono-Retr bis-raw-coind2*

**unfolding** *ZObisT-def* **by** *blast*

**lemma** *Wbis-prefix*:

$\text{Wbis} \subseteq \text{Wretr } \text{Wbis}$

**unfolding** *Wbis-def* **using** *mono-Retr bis-prefix* **by** *blast*

**lemma** *Wbis-sym: sym Wbis*  
**unfolding** *Wbis-def* **using** *mono-Retr sym-bis* **by** *blast*

**lemma** *Wbis-converse:*  
 $((c,d) \in \text{theta}^{-1} \cup \text{Wbis}) = ((d,c) \in \text{theta} \cup \text{Wbis})$   
**by** (*metis Wbis-sym converseI converse-Un converse-converse sym-conv-converse-eq*)

**lemma** *Wbis-Sym:  $c \approx_w d \implies d \approx_w c$*   
**using** *Wbis-sym* **unfolding** *sym-def* **by** *blast*

**lemma**  
*Wbis-matchC-M:  $\bigwedge c d. c \approx_w d \implies \text{matchC-M } \text{Wbis } c d$*   
**and**  
*Wbis-matchT-M:  $\bigwedge c d. c \approx_w d \implies \text{matchT-M } c d$*   
**using** *Wbis-prefix* **unfolding** *Wretr-def* **by** *auto*

**lemmas** *Wbis-step = Wbis-matchC-M Wbis-matchT-M*

**lemma**  
*Wbis-matchC-M-rev:  $\bigwedge s t. s \approx_w t \implies \text{matchC-M } \text{Wbis } t s$*   
**and**  
*Wbis-matchT-M-rev:  $\bigwedge s t. s \approx_w t \implies \text{matchT-M } t s$*   
**using** *Wbis-step Wbis-sym* **unfolding** *sym-def* **by** *blast+*

**lemmas** *Wbis-step-rev = Wbis-matchC-M-rev Wbis-matchT-M-rev*

**lemma** *Wbis-coind:*  
**assumes** *sym theta* **and**  $\text{theta} \subseteq \text{Wretr } (\text{theta} \cup \text{Wbis})$   
**shows**  $\text{theta} \subseteq \text{Wbis}$   
**using** *assms mono-Retr bis-coind*  
**unfolding** *Wbis-def* **by** *blast*

**lemma** *Wbis-raw-coind:*  
**assumes** *sym theta* **and**  $\text{theta} \subseteq \text{Wretr } \text{theta}$   
**shows**  $\text{theta} \subseteq \text{Wbis}$   
**using** *assms mono-Retr bis-raw-coind*  
**unfolding** *Wbis-def* **by** *blast*

**lemma** *Wbis-coind2:*  
**assumes**  $\text{theta} \subseteq \text{Wretr } (\text{theta} \cup \text{Wbis})$  **and**  
 $\text{theta}^{-1} \subseteq \text{Wretr } ((\text{theta}^{-1}) \cup \text{Wbis})$   
**shows**  $\text{theta} \subseteq \text{Wbis}$   
**using** *assms mono-Retr bis-coind2*  
**unfolding** *Wbis-def* **by** *blast*

**lemma** *Wbis-raw-coind2:*  
**assumes**  $\text{theta} \subseteq \text{Wretr } \text{theta}$  **and**  
 $\text{theta}^{-1} \subseteq \text{Wretr } (\text{theta}^{-1})$

**shows**  $\theta \subseteq Wbis$   
**using** *assms mono-Retr bis-raw-coind2*  
**unfolding** *Wbis-def* **by** *blast*

**lemma** *WbisT-prefix*:  
 $WbisT \subseteq WretrT\ WbisT$   
**unfolding** *WbisT-def* **using** *mono-Retr bis-prefix* **by** *blast*

**lemma** *WbisT-sym*:  $\text{sym } WbisT$   
**unfolding** *WbisT-def* **using** *mono-Retr sym-bis* **by** *blast*

**lemma** *WbisT-Sym*:  $c \approx_{wT} d \implies d \approx_{wT} c$   
**using** *WbisT-sym* **unfolding** *sym-def* **by** *blast*

**lemma** *WbisT-converse*:  
 $((c,d) \in \theta^{-1} \cup WbisT) = ((d,c) \in \theta \cup WbisT)$   
**by** (*metis WbisT-sym converseI converse-Un converse-converse sym-conv-converse-eq*)

**lemma**  
 $WbisT\text{-matchC-MC}: \bigwedge c\ d. c \approx_{wT} d \implies \text{matchC-MC } WbisT\ c\ d$   
**and**  
 $WbisT\text{-matchT-MT}: \bigwedge c\ d. c \approx_{wT} d \implies \text{matchT-MT } c\ d$   
**using** *WbisT-prefix* **unfolding** *WretrT-def* **by** *auto*

**lemmas**  $WbisT\text{-step} = WbisT\text{-matchC-MC } WbisT\text{-matchT-MT}$

**lemma**  
 $WbisT\text{-matchC-MC-rev}: \bigwedge s\ t. s \approx_{wT} t \implies \text{matchC-MC } WbisT\ t\ s$   
**and**  
 $WbisT\text{-matchT-MT-rev}: \bigwedge s\ t. s \approx_{wT} t \implies \text{matchT-MT } t\ s$   
**using** *WbisT-step* *WbisT-sym* **unfolding** *sym-def* **by** *blast+*

**lemmas**  $WbisT\text{-step-rev} = WbisT\text{-matchC-MC-rev } WbisT\text{-matchT-MT-rev}$

**lemma** *WbisT-coind*:  
**assumes** *sym theta* **and**  $\theta \subseteq WretrT\ (\theta \cup WbisT)$   
**shows**  $\theta \subseteq WbisT$   
**using** *assms mono-Retr bis-coind*  
**unfolding** *WbisT-def* **by** *blast*

**lemma** *WbisT-raw-coind*:  
**assumes** *sym theta* **and**  $\theta \subseteq WretrT\ \theta$   
**shows**  $\theta \subseteq WbisT$   
**using** *assms mono-Retr bis-raw-coind*  
**unfolding** *WbisT-def* **by** *blast*

**lemma** *WbisT-coind2*:  
**assumes**  $\theta \subseteq WretrT\ (\theta \cup WbisT)$  **and**

$theta \hat{-} 1 \subseteq WretrT ((theta \hat{-} 1) \cup WbisT)$   
**shows**  $theta \subseteq WbisT$   
**using** *assms mono-Retr bis-coind2*  
**unfolding** *WbisT-def* **by** *blast*

**lemma** *WbisT-raw-coind2*:  
**assumes**  $theta \subseteq WretrT theta$  **and**  
 $theta \hat{-} 1 \subseteq WretrT (theta \hat{-} 1)$   
**shows**  $theta \subseteq WbisT$   
**using** *assms mono-Retr bis-raw-coind2*  
**unfolding** *WbisT-def* **by** *blast*

**lemma** *WbisT-coinduct[consumes 1, case-names sym cont termi]*:  
**assumes**  $\varphi: \varphi c d$   
**assumes**  $S: \bigwedge c d. \varphi c d \implies \varphi d c$   
**assumes**  $C: \bigwedge c s d t c' s'. \llbracket \varphi c d ; s \approx t ; (c, s) \rightarrow c (c', s') \rrbracket \implies \exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge (\varphi c' d' \vee c' \approx wT d')$   
**assumes**  $T: \bigwedge c s d t s'. \llbracket \varphi c d ; s \approx t ; (c, s) \rightarrow t s' \rrbracket \implies \exists t'. (d, t) \rightarrow^* t' \wedge s' \approx t'$   
**shows**  $c \approx wT d$

**proof** –  
**let**  $?v = \{(c, d). \varphi c d\}$   
**have** *sym ?v* **by** (*auto intro! symI S*)  
**moreover**  
**have**  $?v \subseteq WretrT (?v \cup WbisT)$   
**using**  $C T$  **by** (*auto simp: WretrT-def matchC-MC-def matchT-MT-def*)  
**ultimately have**  $?v \subseteq WbisT$   
**using** *WbisT-coind* **by** *auto*  
**with**  $\varphi$  **show** *?thesis*  
**by** *auto*  
**qed**

**lemma** *BisT-prefix*:  
 $BisT \subseteq RetrT BisT$   
**unfolding** *BisT-def* **using** *mono-Retr bis-prefix* **by** *blast*

**lemma** *BisT-sym: sym BisT*  
**unfolding** *BisT-def* **using** *mono-Retr sym-bis* **by** *blast*

**lemma** *BisT-Sym: c ≈ T d ⟹ d ≈ T c*  
**using** *BisT-sym* **unfolding** *sym-def* **by** *blast*

**lemma** *BisT-converse*:  
 $((c, d) \in theta \hat{-} 1 \cup BisT) = ((d, c) \in theta \cup BisT)$   
**by** (*metis BisT-sym converseI converse-Un converse-converse sym-conv-converse-eq*)

**lemma**



*BisT-matchC-TMC*:  $\bigwedge c d. c \approx T d \implies \text{matchC-TMC } \text{BisT } c d$   
**and**

*BisT-matchT-TMT*:  $\bigwedge c d. c \approx T d \implies \text{matchT-TMT } c d$   
**using** *BisT-prefix* **unfolding** *RetrT-def* **by** *auto*

**lemmas** *BisT-step* = *BisT-matchC-TMC* *BisT-matchT-TMT*

**lemma**

*BisT-matchC-TMC-rev*:  $\bigwedge c d. c \approx T d \implies \text{matchC-TMC } \text{BisT } d c$   
**and**

*BisT-matchT-TMT-rev*:  $\bigwedge c d. c \approx T d \implies \text{matchT-TMT } d c$   
**using** *BisT-step* *BisT-sym* **unfolding** *sym-def* **by** *blast+*

**lemmas** *BisT-step-rev* = *BisT-matchC-TMC-rev* *BisT-matchT-TMT-rev*

**lemma** *BisT-coind*:

**assumes** *sym* *theta* **and**  $\text{theta} \subseteq \text{RetrT } (\text{theta} \cup \text{BisT})$   
**shows**  $\text{theta} \subseteq \text{BisT}$   
**using** *assms mono-Retr bis-coind*  
**unfolding** *BisT-def* **by** *blast*

**lemma** *BisT-raw-coind*:

**assumes** *sym* *theta* **and**  $\text{theta} \subseteq \text{RetrT } \text{theta}$   
**shows**  $\text{theta} \subseteq \text{BisT}$   
**using** *assms mono-Retr bis-raw-coind*  
**unfolding** *BisT-def* **by** *blast*

**lemma** *BisT-coind2*:

**assumes**  $\text{theta} \subseteq \text{RetrT } (\text{theta} \cup \text{BisT})$  **and**  
 $\text{theta} \hat{-} 1 \subseteq \text{RetrT } ((\text{theta} \hat{-} 1) \cup \text{BisT})$   
**shows**  $\text{theta} \subseteq \text{BisT}$   
**using** *assms mono-Retr bis-coind2*  
**unfolding** *BisT-def* **by** *blast*

**lemma** *BisT-raw-coind2*:

**assumes**  $\text{theta} \subseteq \text{RetrT } \text{theta}$  **and**  
 $\text{theta} \hat{-} 1 \subseteq \text{RetrT } (\text{theta} \hat{-} 1)$   
**shows**  $\text{theta} \subseteq \text{BisT}$   
**using** *assms mono-Retr bis-raw-coind2*  
**unfolding** *BisT-def* **by** *blast*

Inclusions between bisimilarities:

**lemma** *match-imp[simp]*:

$\bigwedge \text{theta } c1 c2. \text{matchC-C } \text{theta } c1 c2 \implies \text{matchC-ZOC } \text{theta } c1 c2$

$\bigwedge \text{theta } c1 c2. \text{matchC-ZOC } \text{theta } c1 c2 \implies \text{matchC-ZO } \text{theta } c1 c2$

$\bigwedge \text{theta } c1 c2. \text{matchC-ZOC } \text{theta } c1 c2 \implies \text{matchC-MC } \text{theta } c1 c2$

$\bigwedge \theta c1 c2. \text{matchC-ZO } \theta c1 c2 \implies \text{matchC-M } \theta c1 c2$

$\bigwedge \theta c1 c2. \text{matchC-MC } \theta c1 c2 \implies \text{matchC-M } \theta c1 c2$

$\bigwedge c1 c2. \text{matchT-T } c1 c2 \implies \text{matchT-ZO } c1 c2$

$\bigwedge c1 c2. \text{matchT-T } c1 c2 \implies \text{matchT-MT } c1 c2$

$\bigwedge c1 c2. \text{matchT-ZO } c1 c2 \implies \text{matchT-M } c1 c2$

$\bigwedge c1 c2. \text{matchT-MT } c1 c2 \implies \text{matchT-M } c1 c2$

$\bigwedge \theta c1 c2. \text{matchC-MC } \theta c1 c2 \implies \text{matchC-TMC } \theta c1 c2$

$\bigwedge \theta c1 c2. \text{matchT-MT } c1 c2 \implies \text{matchT-TMT } c1 c2$

**unfolding** *match-defs* **apply**(*tactic*  $\langle \text{mauto-no-simp-tac } @\{\text{context}\} \rangle$ )

**apply** *fastforce* **apply** *fastforce*

**apply** (*metis* *MtransC-Refl* *transC-MtransC*)

**by** *force+*

**lemma** *Retr-incl*:

$\bigwedge \theta. \text{Sretr } \theta \subseteq \text{ZOretrT } \theta$

$\bigwedge \theta. \text{ZOretrT } \theta \subseteq \text{ZOretr } \theta$

$\bigwedge \theta. \text{ZOretrT } \theta \subseteq \text{WretrT } \theta$

$\bigwedge \theta. \text{ZOretr } \theta \subseteq \text{Wretr } \theta$

$\bigwedge \theta. \text{WretrT } \theta \subseteq \text{Wretr } \theta$

$\bigwedge \theta. \text{WretrT } \theta \subseteq \text{RetrT } \theta$

**unfolding** *Retr-defs* **by** *auto*

**lemma** *bis-incl*:

$\text{Sbis} \subseteq \text{ZObisT}$

$\text{ZObisT} \subseteq \text{ZObis}$

$\text{ZObisT} \subseteq \text{WbisT}$

$\text{ZObis} \subseteq \text{Wbis}$

$\text{WbisT} \subseteq \text{Wbis}$

$\text{WbisT} \subseteq \text{BisT}$

**unfolding** *bis-defs*

**using** *Retr-incl* *mono-bis* *mono-Retr* **by** *blast+*

```

lemma bis-imp[simp]:
 $\wedge c1\ c2. c1 \approx_s c2 \implies c1 \approx_{01T} c2$ 

 $\wedge c1\ c2. c1 \approx_{01T} c2 \implies c1 \approx_{01} c2$ 

 $\wedge c1\ c2. c1 \approx_{01T} c2 \implies c1 \approx_{wT} c2$ 

 $\wedge c1\ c2. c1 \approx_{01} c2 \implies c1 \approx_w c2$ 

 $\wedge c1\ c2. c1 \approx_{wT} c2 \implies c1 \approx_w c2$ 

 $\wedge c1\ c2. c1 \approx_{wT} c2 \implies c1 \approx_T c2$ 
using bis-incl rev-subsetD by auto

```

Self-isomorphism implies strong bisimilarity:

```

lemma siso-Sbis[simp]:
assumes siso c
shows  $c \approx_s c$ 
proof –
  let ?theta =  $\{(c,c) \mid c . \text{siso } c\}$ 
  have  $?theta \subseteq \text{Sbis}$ 
  proof (rule Sbis-raw-coind)
    show sym ?theta unfolding sym-def by blast
  next
    show  $?theta \subseteq \text{Sretr } ?theta$ 
  proof clarify
    fix c assume c: siso c
    show  $(c, c) \in \text{Sretr } ?theta$ 
    unfolding Sretr-def proof (clarify, intro conjI)
      show matchC-C ?theta c c
      unfolding matchC-C-def apply simp
      by (metis c siso-transC siso-transC-indis)
    next
      show matchT-T c c
      unfolding matchT-T-def
      by (metis c siso-transT)
    qed
  qed
  thus ?thesis using assms by blast
qed

```

0-Self-isomorphism implies weak T 0-bisimilarity:

```

lemma siso0-Sbis[simp]:
assumes siso0 c
shows  $c \approx_T c$ 
proof –
  let ?theta =  $\{(c,c) \mid c . \text{siso0 } c\}$ 

```

```

have ?theta  $\subseteq$  BisT
proof(rule BisT-raw-coind)
  show sym ?theta unfolding sym-def by blast
next
show ?theta  $\subseteq$  RetrT ?theta
proof clarify
  fix c assume c: siso0 c
  show (c, c)  $\in$  RetrT ?theta
  unfolding RetrT-def proof (clarify, intro conjI)
    show matchC-TMC ?theta c c
    unfolding matchC-TMC-def apply simp
    by (metis c siso0-transC siso0-transC-indis transC-MtransC)
  next
  show matchT-TMT c c
  unfolding matchC-TMC-def
  by (metis c matchT-TMT-def siso0-transT transT-MtransT)
qed
qed
qed
thus ?thesis using assms by blast
qed

```

end

end

## 5 Compositionality of the during-execution security notions

theory Compositionality imports During-Execution begin

```

context PL-Indis
begin

```

### 5.1 Discreetness versus language constructs:

```

theorem discr-Atm[simp]:
  discr (Atm atm) = presAtm atm
proof-
  {fix c
   have
     ( $\exists$  atm. c = Atm atm  $\wedge$  presAtm atm)
      $\implies$  discr c
  }

```

```

apply(erule discr-coind)
apply (metis Atm-transC-invert)
by (metis PL.Atm-transT-invert presAtm-def)
}
moreover have discr (Atm atm)  $\implies$  presAtm atm
  by (metis Atm presAtm-def discr-transT)
ultimately show ?thesis by blast
qed

```

```

theorem discr-If[simp]:
assumes discr c1 and discr c2
shows discr (If tst c1 c2)
proof –
  {fix c
  have
    ( $\exists$  tst c1 c2. c = If tst c1 c2  $\wedge$  discr c1  $\wedge$  discr c2)  $\implies$  discr c
    apply(erule discr-coind)
    apply (metis PL.If-transC-invert indis-refl)
    by (metis If-transT-invert)
  }
  thus ?thesis using assms by blast
qed

```

```

theorem discr-Seq[simp]:
assumes *: discr c1 and **: discr c2
shows discr (c1 ;; c2)
proof –
  {fix c
  have
    ( $\exists$  c1 c2. c = c1 ;; c2  $\wedge$  discr c1  $\wedge$  discr c2)
     $\implies$  discr c
    apply(erule discr-coind)
    proof(tactic<clarify-all-tac @{context}>)
      fix c s c' s' c1 c2
      assume c1: discr c1 and c2: discr c2
      assume (c1 ;; c2, s)  $\rightarrow$  c (c', s')
      thus s  $\approx$  s'  $\wedge$  (( $\exists$  c1 c2. c' = c1 ;; c2  $\wedge$  discr c1  $\wedge$  discr c2)  $\vee$  discr c')
      apply – apply(erule Seq-transC-invert)
      apply (metis c1 c2 discr-transC discr-transC-indis)
      by (metis c1 c2 discr.cases)
    qed (insert Seq-transT-invert, blast)
  }
  thus ?thesis using assms by blast
qed

```

```

theorem discr-While[simp]:
assumes discr c
shows discr (While tst c)
proof –

```

```

{fix c
  have
    ( $\exists$  tst d. c = While tst d  $\wedge$  discr d)  $\vee$ 
    ( $\exists$  tst d1 d. c = d1 ;; (While tst d)  $\wedge$  discr d1  $\wedge$  discr d)
     $\implies$  discr c
  apply(erule discr-coind)
  apply(tactic <mauto-no-simp-tac @{context}>)
  apply (metis While-transC-invert indis-refl)
  apply (metis Seq-transC-invert discr.cases)
  apply (metis While-transC-invert)
  apply (metis Seq-transC-invert discr.cases)
  apply (metis PL.While-transT-invert indis-refl)
  by (metis Seq-transT-invert)
}
thus ?thesis using assms by blast
qed

```

```

theorem discr-Par[simp]:
  assumes *: discr c1 and **: discr c2
  shows discr (Par c1 c2)
  proof -
    {fix c
      have
        ( $\exists$  c1 c2. c = Par c1 c2  $\wedge$  discr c1  $\wedge$  discr c2)
         $\implies$  discr c
      apply(erule discr-coind)
      proof(tactic <clarify-all-tac @{context}>)
        fix c s c' s' c1 c2
        assume c1: discr c1 and c2: discr c2
        assume (Par c1 c2, s)  $\rightarrow$  c (c', s')
        thus s  $\approx$  s'  $\wedge$  (( $\exists$  c1 c2. c' = Par c1 c2  $\wedge$  discr c1  $\wedge$  discr c2)  $\vee$  discr c')
        apply - apply(erule Par-transC-invert)
        by(metis c1 c2 discr.cases)
      qed
    }
  thus ?thesis using assms by blast
qed

```

## 5.2 Discreteness versus language constructs:

```

theorem discr0-Atm[simp]:
  discr0 (Atm atm) = presAtm atm
  proof -
    {fix c
      have
        ( $\exists$  atm. c = Atm atm  $\wedge$  presAtm atm)
         $\implies$  discr0 c
      apply(erule discr0-coind)
      apply (metis Atm-transC-invert)
    }

```

```

  by (metis discr-Atm discr-transT)
}
moreover have discr0 (Atm atm)  $\implies$  presAtm atm
by (metis Atm discr0-MtransT presAtm-def mustT-Atm transT-MtransT)
ultimately show ?thesis by blast
qed

```

```

theorem discr0-If[simp]:
assumes discr0 c1 and discr0 c2
shows discr0 (If tst c1 c2)
proof –
  {fix c
  have
    ( $\exists$  tst c1 c2. c = If tst c1 c2  $\wedge$  discr0 c1  $\wedge$  discr0 c2)  $\implies$  discr0 c
  apply(erule discr0-coind)
  apply (metis If-transC-invert indis-refl)
  by (metis If-transT-invert)
  }
thus ?thesis using assms by blast
qed

```

```

theorem discr0-Seq[simp]:
assumes *: discr0 c1 and **: discr0 c2
shows discr0 (c1 ;; c2)
proof –
  {fix c
  have
    ( $\exists$  c1 c2. c = c1 ;; c2  $\wedge$  discr0 c1  $\wedge$  discr0 c2)
     $\implies$  discr0 c
  apply(erule discr0-coind)
  proof(tactic<clarify-all-tac @{context}>)
    fix c s c' s' c1 c2
    assume mt: mustT (c1 ;; c2) s
    and c1: discr0 c1 and c2: discr0 c2
    assume (c1 ;; c2, s)  $\rightarrow$  c (c', s')
    thus s  $\approx$  s'  $\wedge$  (( $\exists$  c1 c2. c' = c1 ;; c2  $\wedge$  discr0 c1  $\wedge$  discr0 c2)  $\vee$  discr0 c')
    apply – apply(erule Seq-transC-invert)
    apply (metis mustT-Seq-L c1 c2 discr0-MtransC discr0-MtransC-indis mt
      transC-MtransC)
    by (metis c1 c2 discr0-transT mt mustT-Seq-L)
  qed (insert Seq-transT-invert, blast)
  }
thus ?thesis using assms by blast
qed

```

```

theorem discr0-While[simp]:
assumes discr0 c
shows discr0 (While tst c)
proof –

```

```

{fix c
  have
    ( $\exists$  tst d. c = While tst d  $\wedge$  discr0 d)  $\vee$ 
    ( $\exists$  tst d1 d. c = d1 ;; (While tst d)  $\wedge$  discr0 d1  $\wedge$  discr0 d)
     $\implies$  discr0 c
  proof (induct rule: discr0-coind)
    case (Term c s s')
      thus s  $\approx$  s'
      apply (elim exE disjE conjE)
      apply (metis While-transT-invert indis-refl)
      by (metis Seq-transT-invert)
    next
      case (Cont c s c' s')
        thus ?case
        apply (intro conjI)
        apply (elim exE disjE conjE)
        apply (metis While-transC-invert indis-refl)
        apply (metis Seq-transC-invert discr0-MtransC-indis discr0-transT
          mustT-Seq-L transC-MtransC)

        apply (elim exE disjE conjE)
        apply (metis While-transC-invert)
        by (metis Cont( $\exists$ ) Seq-transC-invert discr0-transC mustT-Seq-L)
      qed
    }
  thus ?thesis using assms by blast
qed

```

```

theorem discr0-Par[simp]:
  assumes *: discr0 c1 and **: discr0 c2
  shows discr0 (Par c1 c2)
  proof -
    {fix c
      have
        ( $\exists$  c1 c2. c = Par c1 c2  $\wedge$  discr0 c1  $\wedge$  discr0 c2)
         $\implies$  discr0 c
      apply (induct rule: discr0-coind)
      proof (tactic (clarify-all-tac @ {context}))
        fix c s c' s' c1 c2
        assume mt: mustT (Par c1 c2) s and c1: discr0 c1 and c2: discr0 c2
        assume (Par c1 c2, s)  $\rightarrow$ c (c', s')
        thus s  $\approx$  s'  $\wedge$  (( $\exists$  c1 c2. c' = Par c1 c2  $\wedge$  discr0 c1  $\wedge$  discr0 c2)  $\vee$  discr0 c')
        apply (elim Par-transC-invert)
        apply (metis c1 c2 discr0.simps mt mustT-Par-L)
        apply (metis c1 c2 discr0-transT mt mustT-Par-L)
        apply (metis c1 c2 discr0.simps indis-sym mt mustT-Par-R)
        by (metis PL.mustT-Par-R c1 c2 discr0-transT mt)
      qed
    }
  }

```



thus ?thesis using assms by blast  
qed

### 5.3 Self-Isomorphism versus language constructs:

**theorem** *siso-Atm[simp]*:  
*siso (Atm atm) = compatAtm atm*  
**proof** –  
 {**fix** *c*  
**have**  
 $(\exists \text{ atm. } c = \text{Atm atm} \wedge \text{compatAtm atm})$   
 $\implies \text{siso } c$   
**apply**(erule *siso-coind*)  
**apply** (*metis Atm-transC-invert*)  
**apply** (*metis PL.Atm-transC-invert*)  
**by** (*metis Atm-transT-invert PL.Atm compatAtm-def*)  
 }  
**moreover have** *siso (Atm atm)  $\implies$  compatAtm atm unfolding compatAtm-def*  
**by** (*metis Atm Atm-transT-invert siso-transT*)  
**ultimately show** ?thesis **by** blast  
 qed

**theorem** *siso-If[simp]*:  
**assumes** *compatTst tst and siso c1 and siso c2*  
**shows** *siso (If tst c1 c2)*  
**proof** –  
 {**fix** *c*  
**have**  
 $(\exists \text{ tst } c1 \text{ } c2. c = \text{If } \text{tst } c1 \text{ } c2 \wedge \text{compatTst } \text{tst} \wedge \text{siso } c1 \wedge \text{siso } c2) \implies \text{siso } c$   
**apply**(erule *siso-coind*)  
**apply** (*metis PL.If-transC-invert indis-refl*)  
**apply** (*metis IfTrue PL.IfFalse PL.If-transC-invert compatTst-def*)  
**by** (*metis If-transT-invert*)  
 }  
 thus ?thesis using assms **by** blast  
 qed

**theorem** *siso-Seq[simp]*:  
**assumes** \*: *siso c1* **and** \*\*: *siso c2*  
**shows** *siso (c1 ;; c2)*  
**proof** –  
 {**fix** *c*  
**have**  
 $(\exists \text{ } c1 \text{ } c2. c = c1 ;; c2 \wedge \text{siso } c1 \wedge \text{siso } c2)$   
 $\implies \text{siso } c$   
**apply**(erule *siso-coind*)  
**proof**(*tactic<clarify-all-tac @{context}>*)  
**fix** *c s t c' s' c1 c2*  
**assume** *s  $\approx$  t and (c1 ;; c2, s)  $\rightarrow$  c (c', s') and siso c1 and siso c2*  
 }  
 thus ?thesis **by** blast  
 qed

```

    thus  $\exists t'. s' \approx t' \wedge (c1 ;; c2, t) \rightarrow c (c', t')$ 
    apply - apply(erule Seq-transC-invert)
    apply (metis SeqC siso-transC-indis)
    by (metis PL.SeqT siso-transT)
  qed (insert Seq-transT-invert siso-transC, blast+)
}
thus ?thesis using assms by blast
qed

```

```

theorem siso-While[simp]:
assumes compatTst tst and siso c
shows siso (While tst c)
proof -
  {fix c
   have
    ( $\exists tst d. compatTst\ tst \wedge c = While\ tst\ d \wedge siso\ d$ )  $\vee$ 
    ( $\exists tst d1 d. compatTst\ tst \wedge c = d1 ;; (While\ tst\ d) \wedge siso\ d1 \wedge siso\ d$ )
     $\implies$  siso c
   apply(erule siso-coind)
   apply auto
   apply (metis PL.Seq-transC-invert siso-transC)
   apply (metis WhileTrue While-transC-invert compatTst-def)
   apply (metis PL.SeqC siso-transC-indis)
   apply (metis PL.SeqT siso-transT)
   by (metis WhileFalse compatTst-def)
  }
  thus ?thesis using assms by blast
qed

```

```

theorem siso-Par[simp]:
assumes *: siso c1 and **: siso c2
shows siso (Par c1 c2)
proof -
  {fix c
   have
    ( $\exists c1 c2. c = Par\ c1\ c2 \wedge siso\ c1 \wedge siso\ c2$ )
     $\implies$  siso c
   apply(erule siso-coind)
   proof(tactic<clarify-all-tac @{context}>)
     fix c s t c' s' c1 c2
     assume  $s \approx t$  and (Par c1 c2, s)  $\rightarrow c (c', s')$  and c1: siso c1 and c2: siso c2
     thus  $\exists t'. s' \approx t' \wedge (Par\ c1\ c2, t) \rightarrow c (c', t')$ 
     apply - apply(erule Par-transC-invert)
     by(metis ParCL ParTL ParCR ParTR c1 c2 siso-transT siso-transC-indis)+
   }
  qed (insert Par-transC-invert siso-transC Par-transT-invert, blast+)
}
thus ?thesis using assms by blast
qed

```

## 5.4 Self-Isomorphism versus language constructs:

**theorem** *siso0-Atm[simp]*:  
*siso0 (Atm atm) = compatAtm atm*  
**proof** –  
 {**fix** *c*  
**have**  
 ( $\exists$  *atm*. *c* = *Atm atm*  $\wedge$  *compatAtm atm*)  
 $\implies$  *siso0 c*  
**apply**(*erule siso0-coind*)  
**apply** (*metis Atm-transC-invert*)  
**apply** (*metis PL.Atm-transC-invert*)  
**by** (*metis Atm-transT-invert PL.Atm.compatAtm-def*)  
 }  
**moreover have** *siso0 (Atm atm)  $\implies$  compatAtm atm* **unfolding** *compatAtm-def*  
**by** (*metis Atm Atm-transT-invert siso0-transT mustT-Atm*)  
**ultimately show** *?thesis* **by** *blast*  
**qed**

**theorem** *siso0-If[simp]*:  
**assumes** *compatTst tst* **and** *siso0 c1* **and** *siso0 c2*  
**shows** *siso0 (If tst c1 c2)*  
**proof** –  
 {**fix** *c*  
**have**  
 ( $\exists$  *tst c1 c2*. *c* = *If tst c1 c2*  $\wedge$  *compatTst tst*  $\wedge$  *siso0 c1*  $\wedge$  *siso0 c2*)  $\implies$  *siso0*  
*c*  
**apply**(*erule siso0-coind*)  
**apply** (*metis PL.If-transC-invert indis-refl*)  
**apply** (*metis IfTrue PL.IfFalse PL.If-transC-invert compatTst-def*)  
**by** (*metis If-transT-invert*)  
 }  
**thus** *?thesis* **using** *assms* **by** *blast*  
**qed**

**theorem** *siso0-Seq[simp]*:  
**assumes** *\**: *siso0 c1* **and** *\*\**: *siso0 c2*  
**shows** *siso0 (c1 ;; c2)*  
**proof** –  
 {**fix** *c*  
**have**  
 ( $\exists$  *c1 c2*. *c* = *c1 ;; c2*  $\wedge$  *siso0 c1*  $\wedge$  *siso0 c2*)  
 $\implies$  *siso0 c*  
**proof** (*induct rule: siso0-coind*)  
**case** (*Indef c s c' s'*)  
**thus** *?case*  
**by** (*metis Seq-transC-invert mustT-Seq-L siso0-transC*)  
**next**  
**case** (*Cont c s t c' s'*)  
**then obtain** *c1 c2*

```

where  $c: c = c1 ;; c2$  and  $mt: mustT (c1 ;; c2) s mustT (c1 ;; c2) t$ 
and  $st: s \approx t$  and  $siso1: siso0 c1$  and  $siso2: siso0 c2$  by auto
hence  $mt1: mustT c1 s mustT c1 t$ 
by (metis mustT-Seq-L)+
have  $(c1 ;; c2, s) \rightarrow_c (c', s')$  using  $c$  Cont by auto
thus ?case
proof (elim Seq-transC-invert)
  fix  $c1'$  assume  $c1: (c1, s) \rightarrow_c (c1', s')$  and  $c': c' = c1' ;; c2$ 
  obtain  $t'$  where  $(c1, t) \rightarrow_c (c1', t')$  and  $s' \approx t'$ 
  using  $siso1 c1 st mt1$  by (metis siso0-transC-indis)
  thus ?thesis by (metis SeqC c c')
next
  assume  $(c1, s) \rightarrow_t s'$  and  $c' = c2$ 
  thus ?thesis by (metis c SeqT mt1 siso0-transT siso1 st)
qed
qed auto
}
thus ?thesis using assms by blast
qed

```

```

theorem siso0-While[simp]:
assumes compatTst tst and siso0 c
shows siso0 (While tst c)
proof –
  {fix  $c$ 
  have
     $(\exists tst d. compatTst tst \wedge c = While\ tst\ d \wedge siso0\ d) \vee$ 
     $(\exists tst d1 d. compatTst\ tst \wedge c = d1 ;; (While\ tst\ d) \wedge siso0\ d1 \wedge siso0\ d)$ 
     $\implies siso0\ c$ 
  apply(erule siso0-coind)
  apply auto
  apply (metis mustT-Seq-L siso0-transC)
  apply (metis WhileTrue While-transC-invert compatTst-def)
  apply (metis SeqC mustT-Seq-L siso0-transC-indis)
  apply (metis SeqT mustT-Seq-L siso0-transT)
  by (metis WhileFalse compatTst-def)
  }
  thus ?thesis using assms by blast
qed

```

```

theorem siso0-Par[simp]:
assumes  $*$ : siso0 c1 and  $**$ : siso0 c2
shows siso0 (Par c1 c2)
proof –
  {fix  $c$ 
  have
     $(\exists c1 c2. c = Par\ c1\ c2 \wedge siso0\ c1 \wedge siso0\ c2)$ 
     $\implies siso0\ c$ 
  proof (induct rule: siso0-coind)

```

```

case (Indef c s c' s')
then obtain c1 c2 where c: c = Par c1 c2
and c1: siso0 c1 and c2: siso0 c2 by auto
hence (Par c1 c2, s)  $\rightarrow$  c (c', s') using c Indef by auto
thus ?case
apply(elim Par-transC-invert)
by (metis Indef c c1 c2 mustT-Par-L mustT-Par-R isoo0-transC)+
next
case (Cont c s t c' s')
then obtain c1 c2 where c: c = Par c1 c2
and c1: siso0 c1 and c2: siso0 c2 by auto
hence mt: mustT c1 s mustT c1 t mustT c2 s mustT c2 t
by (metis Cont mustT-Par-L mustT-Par-R)+
have (Par c1 c2, s)  $\rightarrow$  c (c', s') using c Cont by auto
thus ?case
apply(elim Par-transC-invert)
apply (metis Cont ParCL c c1 mt isoo0-transC-indis)
apply (metis Cont ParTL c c1 mt isoo0-transT)
apply (metis Cont ParCR c c2 mt isoo0-transC-indis)
by (metis Cont ParTR c c2 mt isoo0-transT)
qed auto
}
thus ?thesis using assms by blast
qed

```

## 5.5 Strong bisimilarity versus language constructs

Atomic commands:

**definition** *thetaAtm* **where**  
*thetaAtm atm*  $\equiv \{(Atm\ atm, Atm\ atm)\}$

**lemma** *thetaAtm-sym*:  
*sym (thetaAtm atm)*  
**unfolding** *thetaAtm-def sym-def* **by** *blast*

**lemma** *thetaAtm-Sretr*:  
**assumes** *compatAtm atm*  
**shows** *thetaAtm atm*  $\subseteq$  *Sretr (thetaAtm atm)*  
**using** *assms*  
**unfolding** *compatAtm-def Sretr-def matchC-C-def matchT-T-def thetaAtm-def*  
**apply** *simp* **by** (*metis Atm-transT-invert Atm*)

**lemma** *thetaAtm-Sbis*:  
**assumes** *compatAtm atm*  
**shows** *thetaAtm atm*  $\subseteq$  *Sbis*  
**apply**(*rule Sbis-raw-coind*)  
**using** *assms thetaAtm-sym thetaAtm-Sretr* **by** *auto*

**theorem** *Atm-Sbis[simp]*:

**assumes** *compatAtm atm*  
**shows** *Atm atm  $\approx_s$  Atm atm*  
**using** *assms thetaAtm-Sbis unfolding thetaAtm-def by auto*

Sequential composition:

**definition** *thetaSeq* where  
*thetaSeq*  $\equiv$   
 $\{(c1 ;; c2, d1 ;; d2) \mid c1\ c2\ d1\ d2. c1 \approx_s d1 \wedge c2 \approx_s d2\}$

**lemma** *thetaSeq-sym*:  
*sym thetaSeq*  
**unfolding** *thetaSeq-def sym-def using Sbis-Sym by blast*

**lemma** *thetaSeq-Sretr*:  
*thetaSeq  $\subseteq$  Sretr (thetaSeq Un Sbis)*

**proof** –  
 {**fix** *c1 c2 d1 d2*  
   **assume** *c1d1: c1  $\approx_s$  d1 and c2d2: c2  $\approx_s$  d2*  
   **hence** *matchC-C1: matchC-C Sbis c1 d1 and matchC-C2: matchC-C Sbis c2 d2*  
   **and** *matchT-T1: matchT-T c1 d1 and matchT-T2: matchT-T c2 d2*  
   **using** *Sbis-matchC-C Sbis-matchT-T by auto*  
   **have** *(c1 ;; c2, d1 ;; d2)  $\in$  Sretr (thetaSeq Un Sbis)*  
   **unfolding** *Sretr-def proof (clarify, intro conjI)*  
   **show** *matchC-C (thetaSeq Un Sbis) (c1 ;; c2) (d1 ;; d2)*  
   **unfolding** *matchC-C-def proof (tactic <mauto-no-simp-tac @{context}>)*  
   **fix** *s t c' s'*  
   **assume** *st: s  $\approx$  t assume (c1 ;; c2, s)  $\rightarrow_c$  (c', s')*  
   **thus**  $\exists d' t'. (d1 ;; d2, t) \rightarrow_c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaSeq Un Sbis}$   
   **apply** – **proof**(*erule Seq-transC-invert*)  
   **fix** *c1' assume c1s: (c1, s)  $\rightarrow_c$  (c1', s') and c': c' = c1' ;; c2*  
   **hence**  $\exists d1' t'. (d1, t) \rightarrow_c (d1', t') \wedge s' \approx t' \wedge c1' \approx_s d1'$   
   **using** *st matchC-C1 unfolding matchC-C-def by blast*  
   **thus** *?thesis unfolding c' thetaSeq-def*  
   **apply** *simp by (metis SeqC c2d2)*  
   **next**  
   **assume** *(c1, s)  $\rightarrow_t$  s' and c': c' = c2*  
   **hence**  $\exists t'. (d1, t) \rightarrow_t t' \wedge s' \approx t'$   
   **using** *st matchT-T1 unfolding matchT-T-def by auto*  
   **thus** *?thesis*  
   **unfolding** *c' thetaSeq-def*  
   **apply** *simp by (metis PL.SeqT c2d2)*  
   **qed**  
   **qed**  
   **qed** (*unfold matchT-T-def, auto*)  
 }  
**thus** *?thesis unfolding thetaSeq-def by auto*  
**qed**

**lemma** *thetaSeq-Sbis*:  
*thetaSeq*  $\subseteq$  *Sbis*  
**apply**(rule *Sbis-coind*)  
**using** *thetaSeq-sym thetaSeq-Sretr* **by** *auto*

**theorem** *Seq-Sbis[simp]*:  
**assumes** *c1*  $\approx_s$  *d1* **and** *c2*  $\approx_s$  *d2*  
**shows** *c1* ;; *c2*  $\approx_s$  *d1* ;; *d2*  
**using** *assms thetaSeq-Sbis* **unfolding** *thetaSeq-def* **by** *blast*

Conditional:

**definition** *thetaIf* **where**  
*thetaIf*  $\equiv$   
 $\{(If\ tst\ c1\ c2,\ If\ tst\ d1\ d2) \mid\ tst\ c1\ c2\ d1\ d2.\ compaTst\ tst \wedge c1 \approx_s d1 \wedge c2 \approx_s d2\}$

**lemma** *thetaIf-sym*:  
*sym thetaIf*  
**unfolding** *thetaIf-def sym-def* **using** *Sbis-Sym* **by** *blast*

**lemma** *thetaIf-Sretr*:  
*thetaIf*  $\subseteq$  *Sretr* (*thetaIf Un Sbis*)

**proof** –  
**{fix** *tst c1 c2 d1 d2*  
**assume** *tst: compaTst tst* **and** *c1d1: c1*  $\approx_s$  *d1* **and** *c2d2: c2*  $\approx_s$  *d2*  
**hence** *matchC-C1: matchC-C Sbis c1 d1* **and** *matchC-C2: matchC-C Sbis c2*  
*d2*  
**and** *matchT-T1: matchT-T c1 d1* **and** *matchT-T2: matchT-T c2 d2*  
**using** *Sbis-matchC-C Sbis-matchT-T* **by** *auto*  
**have** (*If tst c1 c2, If tst d1 d2*)  $\in$  *Sretr* (*thetaIf Un Sbis*)  
**unfolding** *Sretr-def* **proof** (*clarify, intro conjI*)  
**show** *matchC-C* (*thetaIf Un Sbis*) (*If tst c1 c2*) (*If tst d1 d2*)  
**unfolding** *matchC-C-def* **proof** (*tactic*  $\langle$ *mauto-no-simp-tac*  $\@$  $\{context\}$  $\rangle$ )  
**fix** *s t c' s'*  
**assume** *st: s*  $\approx$  *t* **assume** (*If tst c1 c2, s*)  $\rightarrow_c$  (*c', s'*)  
**thus**  $\exists d' t'. (If\ tst\ d1\ d2,\ t) \rightarrow_c (d', t') \wedge s' \approx t' \wedge (c', d') \in\ thetaIf\ Un\ Sbis$   
**apply** – **apply**(erule *If-transC-invert*)  
**unfolding** *thetaIf-def*  
**apply** *simp* **apply** (*metis IfTrue c1d1 compaTst-def st tst*)  
**apply** *simp* **by** (*metis IfFalse c2d2 compaTst-def st tst*)  
**qed**  
**qed** (*unfold matchT-T-def, auto*)  
**}**  
**thus** *?thesis* **unfolding** *thetaIf-def* **by** *auto*  
**qed**

**lemma** *thetaIf-Sbis*:  
*thetaIf*  $\subseteq$  *Sbis*  
**apply**(rule *Sbis-coind*)

**using** *thetaIf-sym thetaIf-Sretr* **by** *auto*

**theorem** *If-Sbis[simp]*:

**assumes** *compatTst tst* **and**  $c1 \approx_s d1$  **and**  $c2 \approx_s d2$

**shows** *If tst c1 c2  $\approx_s$  If tst d1 d2*

**using** *assms thetaIf-Sbis* **unfolding** *thetaIf-def* **by** *blast*

While loop:

**definition** *thetaWhile* **where**

*thetaWhile*  $\equiv$

$\{(While\ tst\ c,\ While\ tst\ d) \mid\ tst\ c\ d.\ compatTst\ tst \wedge c \approx_s d\}\ Un$

$\{(c1\ ;;\ (While\ tst\ c),\ d1\ ;;\ (While\ tst\ d)) \mid\ tst\ c1\ d1\ c\ d.\ compatTst\ tst \wedge c1 \approx_s d1 \wedge c \approx_s d\}$

**lemma** *thetaWhile-sym*:

*sym thetaWhile*

**unfolding** *thetaWhile-def sym-def* **using** *Sbis-Sym* **by** *blast*

**lemma** *thetaWhile-Sretr*:

*thetaWhile*  $\subseteq$  *Sretr (thetaWhile Un Sbis)*

**proof** –

**{fix** *tst c d*

**assume** *tst: compatTst tst* **and** *c-d: c  $\approx_s$  d*

**hence** *matchC-C: matchC-C Sbis c d*

**and** *matchT-T: matchT-T c d*

**using** *Sbis-matchC-C Sbis-matchT-T* **by** *auto*

**have**  $(While\ tst\ c,\ While\ tst\ d) \in Sretr\ (thetaWhile\ Un\ Sbis)$

**unfolding** *Sretr-def* **proof** (*clarify, intro conjI*)

**show** *matchC-C (thetaWhile  $\cup$  Sbis) (While tst c) (While tst d)*

**unfolding** *matchC-C-def* **proof** (*tactic <mauto-no-simp-tac @{\context}>*)

**fix** *s t c' s'*

**assume** *st: s  $\approx$  t* **assume**  $(While\ tst\ c,\ s) \rightarrow c\ (c',\ s')$

**thus**  $\exists d'\ t'. (While\ tst\ d,\ t) \rightarrow c\ (d',\ t') \wedge s' \approx t' \wedge (c',\ d') \in thetaWhile \cup$

*Sbis*

**apply** – **apply**(*erule While-transC-invert*)

**unfolding** *thetaWhile-def* **apply** *simp*

**by** (*metis WhileTrue c-d compatTst-def st tst*)

**qed**

**next**

**show** *matchT-T (While tst c) (While tst d)*

**unfolding** *matchT-T-def* **proof** (*tactic <mauto-no-simp-tac @{\context}>*)

**fix** *s t s'* **assume** *st: s  $\approx$  t* **assume**  $(While\ tst\ c,\ s) \rightarrow t\ s'$

**thus**  $\exists t'. (While\ tst\ d,\ t) \rightarrow t\ t' \wedge s' \approx t'$

**apply** – **apply**(*erule While-transT-invert*)

**unfolding** *thetaWhile-def* **apply** *simp*

**by** (*metis PL.WhileFalse compatTst-def st tst*)

**qed**

**qed**

**}**



```

moreover
{fix tst c1 d1 c d
  assume tst: compatTst tst and c1d1: c1  $\approx_s$  d1 and c-d: c  $\approx_s$  d
  hence matchC-C1: matchC-C Sbis c1 d1 and matchC-C: matchC-C Sbis c d
    and matchT-T1: matchT-T c1 d1 and matchT-T: matchT-T c d
  using Sbis-matchC-C Sbis-matchT-T by auto
  have (c1 ;; (While tst c), d1 ;; (While tst d))  $\in$  Sretr (thetaWhile Un Sbis)
  unfolding Sretr-def proof (clarify, intro conjI)
  show matchC-C (thetaWhile  $\cup$  Sbis) (c1 ;; (While tst c)) (d1 ;; (While tst d))
  unfolding matchC-C-def proof (tactic  $\langle$ mauto-no-simp-tac  $\@$ {context} $\rangle$ )
    fix s t c' s'
    assume st: s  $\approx$  t assume (c1 ;; (While tst c), s)  $\rightarrow_c$  (c', s')
    thus  $\exists$  d' t'. (d1 ;; (While tst d), t)  $\rightarrow_c$  (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  (c', d')  $\in$ 
thetaWhile  $\cup$  Sbis
    apply – proof(erule Seq-transC-invert)
    fix c1' assume (c1, s)  $\rightarrow_c$  (c1', s') and c': c' = c1' ;; (While tst c)
    hence  $\exists$  d' t'. (d1, t)  $\rightarrow_c$  (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  c1'  $\approx_s$  d'
    using st matchC-C1 unfolding matchC-C-def by blast
    thus ?thesis
    unfolding c' thetaWhile-def
    apply simp by (metis SeqC c-d tst)
  next
    assume (c1, s)  $\rightarrow_t$  s' and c': c' = While tst c
    hence  $\exists$  t'. (d1, t)  $\rightarrow_t$  t'  $\wedge$  s'  $\approx$  t'
    using st matchT-T1 unfolding matchT-T-def by auto
    thus ?thesis
    unfolding c' thetaWhile-def
    apply simp by (metis PL.SeqT c-d tst)
  qed
qed
qed (unfold matchT-T-def, auto)
}
ultimately show ?thesis unfolding thetaWhile-def by auto
qed

```

```

lemma thetaWhile-Sbis:
thetaWhile  $\subseteq$  Sbis
apply(rule Sbis-coind)
using thetaWhile-sym thetaWhile-Sretr by auto

```

```

theorem While-Sbis[simp]:
assumes compatTst tst and c  $\approx_s$  d
shows While tst c  $\approx_s$  While tst d
using assms thetaWhile-Sbis unfolding thetaWhile-def by auto

```

Parallel composition:

```

definition thetaPar where
thetaPar  $\equiv$ 
{(Par c1 c2, Par d1 d2) | c1 c2 d1 d2. c1  $\approx_s$  d1  $\wedge$  c2  $\approx_s$  d2}

```

```

lemma thetaPar-sym:
  sym thetaPar
  unfolding thetaPar-def sym-def using Sbis-Sym by blast

lemma thetaPar-Sretr:
  thetaPar  $\subseteq$  Sretr (thetaPar Un Sbis)
proof –
  {fix c1 c2 d1 d2
   assume c1d1: c1  $\approx_s$  d1 and c2d2: c2  $\approx_s$  d2
   hence matchC-C1: matchC-C Sbis c1 d1 and matchC-C2: matchC-C Sbis c2
  d2
   and matchT-T1: matchT-T c1 d1 and matchT-T2: matchT-T c2 d2
   using Sbis-matchC-C Sbis-matchT-T by auto
   have (Par c1 c2, Par d1 d2)  $\in$  Sretr (thetaPar Un Sbis)
   unfolding Sretr-def proof (clarify, intro conjI)
   show matchC-C (thetaPar  $\cup$  Sbis) (Par c1 c2) (Par d1 d2)
   unfolding matchC-C-def proof (tactic  $\langle$ mauto-no-simp-tac  $\@$ {context} $\rangle$ )
     fix s t c' s'
     assume st: s  $\approx$  t assume (Par c1 c2, s)  $\rightarrow_c$  (c', s')
     thus  $\exists d' t'$ . (Par d1 d2, t)  $\rightarrow_c$  (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  (c', d')  $\in$  thetaPar  $\cup$  Sbis
     apply – proof(erule Par-transC-invert)
       fix c1' assume c1s: (c1, s)  $\rightarrow_c$  (c1', s') and c': c' = Par c1' c2
       hence  $\exists d' t'$ . (d1, t)  $\rightarrow_c$  (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  c1'  $\approx_s$  d'
       using st matchC-C1 unfolding matchC-C-def by blast
       thus ?thesis unfolding c' thetaPar-def
       apply simp by(metis ParCL c2d2)
     next
     assume (c1, s)  $\rightarrow_t$  s' and c': c' = c2
     hence  $\exists t'$ . (d1, t)  $\rightarrow_t$  t'  $\wedge$  s'  $\approx$  t'
     using st matchT-T1 unfolding matchT-T-def by auto
     thus ?thesis
     unfolding c' thetaPar-def
     apply simp by (metis PL.ParTL c2d2)
     next
     fix c2' assume (c2, s)  $\rightarrow_c$  (c2', s') and c': c' = Par c1 c2'
     hence  $\exists d' t'$ . (d2, t)  $\rightarrow_c$  (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  c2'  $\approx_s$  d'
     using st matchC-C2 unfolding matchC-C-def by blast
     thus ?thesis
     unfolding c' thetaPar-def
     apply simp by (metis ParCR c1d1)
     next
     assume (c2, s)  $\rightarrow_t$  s' and c': c' = c1
     hence  $\exists t'$ . (d2, t)  $\rightarrow_t$  t'  $\wedge$  s'  $\approx$  t'
     using st matchT-T2 unfolding matchT-T-def by auto
     thus ?thesis
     unfolding c' thetaPar-def
     apply simp by (metis PL.ParTR c1d1)
  }
qed

```

```

    qed
  qed (unfold matchT-T-def, auto)
}
thus ?thesis unfolding thetaPar-def by auto
qed

```

```

lemma thetaPar-Sbis:
  thetaPar  $\subseteq$  Sbis
  apply (rule Sbis-coind)
  using thetaPar-sym thetaPar-Sretr by auto

```

```

theorem Par-Sbis[simp]:
  assumes  $c1 \approx_s d1$  and  $c2 \approx_s d2$ 
  shows  $Par\ c1\ c2 \approx_s Par\ d1\ d2$ 
  using assms thetaPar-Sbis unfolding thetaPar-def by blast

```

### 5.5.1 01T-bisimilarity versus language constructs

Atomic commands:

```

theorem Atm-ZObisT:
  assumes compatAtm atm
  shows  $Atm\ atm \approx_{01T}\ Atm\ atm$ 
  by (metis Atm-Sbis assms bis-imp)

```

Sequential composition:

```

definition thetaSeqZOT where
  thetaSeqZOT  $\equiv$ 
  {( $c1 \ ;\ ;\ c2, d1 \ ;\ ;\ d2$ ) |  $c1\ c2\ d1\ d2. c1 \approx_{01T}\ d1 \wedge c2 \approx_{01T}\ d2$ }

```

```

lemma thetaSeqZOT-sym:
  sym thetaSeqZOT
  unfolding thetaSeqZOT-def sym-def using ZObisT-Sym by blast

```

```

lemma thetaSeqZOT-ZOretrT:
  thetaSeqZOT  $\subseteq$  ZOretrT (thetaSeqZOT Un ZObisT)

```

```

proof -
  {fix  $c1\ c2\ d1\ d2$ 
   assume  $c1d1: c1 \approx_{01T}\ d1$  and  $c2d2: c2 \approx_{01T}\ d2$ 
   hence matchC-ZOC1: matchC-ZOC ZObisT  $c1\ d1$  and matchC-ZOC2: matchC-ZOC
     ZObisT  $c2\ d2$ 
   and matchT-T1: matchT-T  $c1\ d1$  and matchT-T2: matchT-T  $c2\ d2$ 
   using ZObisT-matchC-ZOC ZObisT-matchT-T by auto
   have ( $c1 \ ;\ ;\ c2, d1 \ ;\ ;\ d2$ )  $\in$  ZOretrT (thetaSeqZOT Un ZObisT)
   unfolding ZOretrT-def proof (clarify, intro conjI)
     show matchC-ZOC (thetaSeqZOT Un ZObisT) ( $c1 \ ;\ ;\ c2$ ) ( $d1 \ ;\ ;\ d2$ )
     unfolding matchC-ZOC-def proof (tactic <mauto-no-simp-tac @{context}>)
       fix  $s\ t\ c'\ s'$ 
       assume  $st: s \approx t$  assume ( $c1 \ ;\ ;\ c2, s$ )  $\rightarrow_c (c', s')$ 
       thus

```

```

      (s' ≈ t ∧ (c', d1 ;; d2) ∈ thetaSeqZOT Un ZObisT) ∨
      (∃ d' t'. (d1 ;; d2, t) →c (d', t') ∧ s' ≈ t' ∧ (c', d') ∈ thetaSeqZOT Un
ZObisT)
apply – proof(erule Seq-transC-invert)
  fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = c1' ;; c2
  hence
    (s' ≈ t ∧ c1' ≈01T d1) ∨
    (∃ d1' t'. (d1, t) →c (d1', t') ∧ s' ≈ t' ∧ c1' ≈01T d1')
  using st matchC-ZOC1 unfolding matchC-ZOC-def by auto
  thus ?thesis unfolding c' thetaSeqZOT-def
  apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
  apply simp apply (metis c2d2)
  apply simp by (metis SeqC c2d2 )
next
  assume (c1, s) →t s' and c': c' = c2
  hence ∃ t'. (d1, t) →t t' ∧ s' ≈ t'
  using st matchT-T1 unfolding matchT-T-def by auto
  thus ?thesis
  unfolding c' thetaSeqZOT-def
  apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
  apply simp by (metis PL.SeqT c2d2)
  qed
qed
qed (unfold matchT-T-def, auto)
}
thus ?thesis unfolding thetaSeqZOT-def by auto
qed

```

```

lemma thetaSeqZOT-ZObisT:
  thetaSeqZOT ⊆ ZObisT
apply(rule ZObisT-coind)
using thetaSeqZOT-sym thetaSeqZOT-ZOretrT by auto

```

```

theorem Seq-ZObisT[simp]:
assumes c1 ≈01T d1 and c2 ≈01T d2
shows c1 ;; c2 ≈01T d1 ;; d2
using assms thetaSeqZOT-ZObisT unfolding thetaSeqZOT-def by blast

```

Conditional:

```

definition thetaIfZOT where
  thetaIfZOT ≡
  {(If tst c1 c2, If tst d1 d2) | tst c1 c2 d1 d2. compatTst tst ∧ c1 ≈01T d1 ∧ c2
≈01T d2}

```

```

lemma thetaIfZOT-sym:
  sym thetaIfZOT
unfolding thetaIfZOT-def sym-def using ZObisT-Sym by blast

```

```

lemma thetaIfZOT-ZOretrT:

```

$\text{thetaIfZOT} \subseteq \text{ZOretrT} (\text{thetaIfZOT Un ZObisT})$   
**proof** –  
 {**fix**  $tst\ c1\ c2\ d1\ d2$   
   **assume**  $tst: \text{compatTst}\ tst$  **and**  $c1d1: c1 \approx_{01T}\ d1$  **and**  $c2d2: c2 \approx_{01T}\ d2$   
   **hence**  $\text{matchC-ZOC1}: \text{matchC-ZOC}\ \text{ZObisT}\ c1\ d1$  **and**  $\text{matchC-ZOC2}: \text{matchC-ZOC}\ \text{ZObisT}\ c2\ d2$   
   **and**  $\text{matchT-T1}: \text{matchT-T}\ c1\ d1$  **and**  $\text{matchT-T2}: \text{matchT-T}\ c2\ d2$   
   **using**  $\text{ZObisT-matchC-ZOC}\ \text{ZObisT-matchT-T}$  **by**  $\text{auto}$   
   **have**  $(\text{If}\ tst\ c1\ c2, \text{If}\ tst\ d1\ d2) \in \text{ZOretrT} (\text{thetaIfZOT Un ZObisT})$   
   **unfolding**  $\text{ZOretrT-def}$  **proof** ( $\text{clarify}, \text{intro}\ \text{conjI}$ )  
   **show**  $\text{matchC-ZOC} (\text{thetaIfZOT Un ZObisT}) (\text{If}\ tst\ c1\ c2) (\text{If}\ tst\ d1\ d2)$   
   **unfolding**  $\text{matchC-ZOC-def}$  **proof** ( $\text{tactic}\ \langle \text{mauto-no-simp-tac}\ @\{\text{context}\} \rangle$ )  
     **fix**  $s\ t\ c'\ s'$   
     **assume**  $st: s \approx t$  **assume**  $(\text{If}\ tst\ c1\ c2, s) \rightarrow c (c', s')$   
     **thus**  
        $(s' \approx t \wedge (c', \text{If}\ tst\ d1\ d2) \in \text{thetaIfZOT Un ZObisT}) \vee$   
        $(\exists d'\ t'. (\text{If}\ tst\ d1\ d2, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaIfZOT Un ZObisT})$   
     **apply** – **apply** ( $\text{erule}\ \text{If-transC-invert}$ )  
     **unfolding**  $\text{thetaIfZOT-def}$   
     **apply**  $\text{simp}$  **apply** ( $\text{metis}\ \text{IfTrue}\ c1d1\ \text{compatTst-def}\ st\ tst$ )  
     **apply**  $\text{simp}$  **by** ( $\text{metis}\ \text{IfFalse}\ c2d2\ \text{compatTst-def}\ st\ tst$ )  
     **qed**  
   **qed** ( $\text{unfold}\ \text{matchT-T-def}, \text{auto}$ )  
 }  
**thus**  $?thesis$  **unfolding**  $\text{thetaIfZOT-def}$  **by**  $\text{auto}$   
**qed**

**lemma**  $\text{thetaIfZOT-ZObisT}$ :  
 $\text{thetaIfZOT} \subseteq \text{ZObisT}$   
**apply** ( $\text{rule}\ \text{ZObisT-coind}$ )  
**using**  $\text{thetaIfZOT-sym}\ \text{thetaIfZOT-ZOretrT}$  **by**  $\text{auto}$

**theorem**  $\text{If-ZObisT}[\text{simp}]$ :  
**assumes**  $\text{compatTst}\ tst$  **and**  $c1 \approx_{01T}\ d1$  **and**  $c2 \approx_{01T}\ d2$   
**shows**  $\text{If}\ tst\ c1\ c2 \approx_{01T}\ \text{If}\ tst\ d1\ d2$   
**using**  $\text{assms}\ \text{thetaIfZOT-ZObisT}$  **unfolding**  $\text{thetaIfZOT-def}$  **by**  $\text{blast}$

While loop:

**definition**  $\text{thetaWhileZOT}$  **where**  
 $\text{thetaWhileZOT} \equiv$   
 $\{(\text{While}\ tst\ c, \text{While}\ tst\ d) \mid \text{tst}\ c\ d.\ \text{compatTst}\ tst \wedge c \approx_{01T}\ d\} \text{ Un}$   
 $\{(c1 ;; (\text{While}\ tst\ c), d1 ;; (\text{While}\ tst\ d)) \mid \text{tst}\ c1\ d1\ c\ d.\ \text{compatTst}\ tst \wedge c1 \approx_{01T}\ d1 \wedge c \approx_{01T}\ d\}$

**lemma**  $\text{thetaWhileZOT-sym}$ :  
 $\text{sym}\ \text{thetaWhileZOT}$   
**unfolding**  $\text{thetaWhileZOT-def}\ \text{sym-def}$  **using**  $\text{ZObisT-Sym}$  **by**  $\text{blast}$

```

lemma thetaWhileZOT-ZOretrT:
thetaWhileZOT  $\subseteq$  ZOretrT (thetaWhileZOT Un ZObisT)
proof –
  {fix tst c d
    assume tst: compatTst tst and c-d: c  $\approx$ 01T d
    hence matchC-ZOC: matchC-ZOC ZObisT c d
    and matchT-T: matchT-T c d
    using ZObisT-matchC-ZOC ZObisT-matchT-T by auto
    have (While tst c, While tst d)  $\in$  ZOretrT (thetaWhileZOT Un ZObisT)
    unfolding ZOretrT-def proof (clarify, intro conjI)
    show matchC-ZOC (thetaWhileZOT  $\cup$  ZObisT) (While tst c) (While tst d)
    unfolding matchC-ZOC-def proof (tactic  $\langle$ mauto-no-simp-tac @{\context}\rangle)
    fix s t c' s'
    assume st: s  $\approx$  t assume (While tst c, s)  $\rightarrow$  c (c', s')
    thus
    (s'  $\approx$  t  $\wedge$  (c', While tst d)  $\in$  thetaWhileZOT  $\cup$  ZObisT)  $\vee$ 
    ( $\exists$  d' t'. (While tst d, t)  $\rightarrow$  c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  (c', d')  $\in$  thetaWhileZOT  $\cup$ 
ZObisT)
    apply – apply(erule While-transC-invert)
    unfolding thetaWhileZOT-def apply simp
    by (metis WhileTrue c-d compatTst-def st st)
    qed
  next
    show matchT-T (While tst c) (While tst d)
    unfolding matchT-T-def proof (tactic  $\langle$ mauto-no-simp-tac @{\context}\rangle)
    fix s t s' assume st: s  $\approx$  t assume (While tst c, s)  $\rightarrow$  t s'
    thus  $\exists$  t'. (While tst d, t)  $\rightarrow$  t t'  $\wedge$  s'  $\approx$  t'
    apply – apply(erule While-transT-invert)
    unfolding thetaWhileZOT-def apply simp
    by (metis PL.WhileFalse compatTst-def st st)
    qed
  }
  moreover
  {fix tst c1 d1 c d
    assume tst: compatTst tst and c1d1: c1  $\approx$ 01T d1 and c-d: c  $\approx$ 01T d
    hence matchC-ZOC1: matchC-ZOC ZObisT c1 d1 and matchC-ZOC: matchC-ZOC
ZObisT c d
    and matchT-T1: matchT-T c1 d1 and matchT-T: matchT-T c d
    using ZObisT-matchC-ZOC ZObisT-matchT-T by auto
    have (c1 ;; (While tst c), d1 ;; (While tst d))  $\in$  ZOretrT (thetaWhileZOT Un
ZObisT)
    unfolding ZOretrT-def proof (clarify, intro conjI)
    show matchC-ZOC (thetaWhileZOT  $\cup$  ZObisT) (c1 ;; (While tst c)) (d1 ;;
(While tst d))
    unfolding matchC-ZOC-def proof (tactic  $\langle$ mauto-no-simp-tac @{\context}\rangle)
    fix s t c' s'
    assume st: s  $\approx$  t assume (c1 ;; (While tst c), s)  $\rightarrow$  c (c', s')
    thus

```

```

      (s' ≈ t ∧ (c', d1 ;; (While tst d)) ∈ thetaWhileZOT ∪ ZObisT) ∨
      (∃ d' t'. (d1 ;; (While tst d), t) → c (d', t') ∧ s' ≈ t' ∧ (c', d') ∈ thetaWhileZOT
      ∪ ZObisT)
    apply – proof(erule Seq-transC-invert)
      fix c1' assume (c1, s) → c (c1', s') and c': c' = c1' ;; (While tst c)
      hence
        (s' ≈ t ∧ c1' ≈01T d1) ∨
        (∃ d' t'. (d1, t) → c (d', t') ∧ s' ≈ t' ∧ c1' ≈01T d')
      using st matchC-ZOC1 unfolding matchC-ZOC-def by auto
      thus ?thesis
      unfolding c' thetaWhileZOT-def
      apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
      apply simp apply (metis c-d tst)
      apply simp by (metis SeqC c-d tst)
    next
      assume (c1, s) → t s' and c': c' = While tst c
      hence ∃ t'. (d1, t) → t t' ∧ s' ≈ t'
      using st matchT-T1 unfolding matchT-T-def by auto
      thus ?thesis
      unfolding c' thetaWhileZOT-def
      apply simp by (metis PL.SeqT c-d tst)
    qed
  qed
  qed (unfold matchT-T-def, auto)
}
ultimately show ?thesis unfolding thetaWhileZOT-def by auto
qed

```

**lemma** *thetaWhileZOT-ZObisT*:  
*thetaWhileZOT* ⊆ *ZObisT*  
 apply(rule *ZObisT-coind*)  
 using *thetaWhileZOT-sym thetaWhileZOT-ZOretrT* by auto

**theorem** *While-ZObisT[simp]*:  
 assumes *compatTst tst* and *c ≈01T d*  
 shows *While tst c ≈01T While tst d*  
 using *assms thetaWhileZOT-ZObisT* unfolding *thetaWhileZOT-def* by auto

Parallel composition:

**definition** *thetaParZOT* where  
*thetaParZOT* ≡  
 {(Par c1 c2, Par d1 d2) | c1 c2 d1 d2. c1 ≈01T d1 ∧ c2 ≈01T d2}

**lemma** *thetaParZOT-sym*:  
*sym thetaParZOT*  
 unfolding *thetaParZOT-def sym-def* using *ZObisT-Sym* by blast

**lemma** *thetaParZOT-ZOretrT*:  
*thetaParZOT* ⊆ *ZOretrT (thetaParZOT Un ZObisT)*

**proof** –

```

{fix c1 c2 d1 d2
  assume c1d1: c1 ≈01T d1 and c2d2: c2 ≈01T d2
  hence matchC-ZOC1: matchC-ZOC ZObisT c1 d1 and matchC-ZOC2: matchC-ZOC
ZObisT c2 d2
  and matchT-T1: matchT-T c1 d1 and matchT-T2: matchT-T c2 d2
  using ZObisT-matchC-ZOC ZObisT-matchT-T by auto
  have (Par c1 c2, Par d1 d2) ∈ ZOrerT (thetaParZOT Un ZObisT)
  unfolding ZOrerT-def proof (clarify, intro conjI)
  show matchC-ZOC (thetaParZOT ∪ ZObisT) (Par c1 c2) (Par d1 d2)
  unfolding matchC-ZOC-def proof (tactic ⟨mauto-no-simp-tac @{context}⟩)
  fix s t c' s'
  assume st: s ≈ t assume (Par c1 c2, s) →c (c', s')
  thus
  (s' ≈ t ∧ (c', Par d1 d2) ∈ thetaParZOT ∪ ZObisT) ∨
  (∃ d' t'. (Par d1 d2, t) →c (d', t') ∧ s' ≈ t' ∧ (c', d') ∈ thetaParZOT ∪
ZObisT)
  apply – proof(erule Par-transC-invert)
  fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = Par c1' c2
  hence
  (s' ≈ t ∧ c1' ≈01T d1) ∨
  (∃ d' t'. (d1, t) →c (d', t') ∧ s' ≈ t' ∧ c1' ≈01T d')
  using st matchC-ZOC1 unfolding matchC-ZOC-def by auto
  thus ?thesis unfolding c' thetaParZOT-def
  apply – apply(tactic ⟨mauto-no-simp-tac @{context}⟩)
  apply simp apply (metis c2d2)
  apply simp by(metis ParCL c2d2)
next
  assume (c1, s) →t s' and c': c' = c2
  hence ∃ t'. (d1, t) →t t' ∧ s' ≈ t'
  using st matchT-T1 unfolding matchT-T-def by auto
  thus ?thesis
  unfolding c' thetaParZOT-def
  apply simp by (metis PL.ParTL c2d2)
next
  fix c2' assume (c2, s) →c (c2', s') and c': c' = Par c1 c2'
  hence
  (s' ≈ t ∧ c2' ≈01T d2) ∨
  (∃ d' t'. (d2, t) →c (d', t') ∧ s' ≈ t' ∧ c2' ≈01T d')
  using st matchC-ZOC2 unfolding matchC-ZOC-def by auto
  thus ?thesis
  unfolding c' thetaParZOT-def
  apply – apply(tactic ⟨mauto-no-simp-tac @{context}⟩)
  apply simp apply (metis c1d1)
  apply simp by (metis ParCR c1d1)
next
  assume (c2, s) →t s' and c': c' = c1
  hence ∃ t'. (d2, t) →t t' ∧ s' ≈ t'
  using st matchT-T2 unfolding matchT-T-def by auto

```



```

    thus ?thesis
    unfolding c' thetaParZOT-def
    apply simp by (metis PL.ParTR c1d1)
  qed
  qed
  qed (unfold matchT-T-def, auto)
}
thus ?thesis unfolding thetaParZOT-def by auto
qed

```

```

lemma thetaParZOT-ZObisT:
  thetaParZOT  $\subseteq$  ZObisT
  apply (rule ZObisT-coind)
  using thetaParZOT-sym thetaParZOT-ZOretrT by auto

```

```

theorem Par-ZObisT[simp]:
  assumes c1  $\approx$ 01T d1 and c2  $\approx$ 01T d2
  shows Par c1 c2  $\approx$ 01T Par d1 d2
  using assms thetaParZOT-ZObisT unfolding thetaParZOT-def by blast

```

## 5.5.2 01-bisimilarity versus language constructs

Discreetness:

```

theorem discr-ZObis[simp]:
  assumes *: discr c and **: discr d
  shows c  $\approx$ 01 d
  proof -
    let ?theta = {(c,d) | c d. discr c  $\wedge$  discr d}
    have ?theta  $\subseteq$  ZObis
    proof (rule ZObis-raw-coind)
      show sym ?theta unfolding sym-def by blast
    next
      show ?theta  $\subseteq$  ZOretr ?theta
    proof clarify
      fix c d assume c: discr c and d: discr d
      show (c, d)  $\in$  ZOretr ?theta
      unfolding ZOretr-def proof (clarify, intro conjI)
        show matchC-ZO ?theta c d
        unfolding matchC-ZO-def proof (tactic <mauto-no-simp-tac @ {context}>)
          fix s t c' s'
          assume st: s  $\approx$  t and cs: (c, s)  $\rightarrow$  c (c', s')
          show
            (s'  $\approx$  t  $\wedge$  (c', d)  $\in$  ?theta)  $\vee$ 
            ( $\exists$  d' t'. (d, t)  $\rightarrow$  c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  (c', d')  $\in$  ?theta)  $\vee$ 
            ( $\exists$  t'. (d, t)  $\rightarrow$  t'  $\wedge$  s'  $\approx$  t'  $\wedge$  discr c')
        proof -
          have s  $\approx$  s' using c cs discr-transC-indis by blast
          hence s't: s'  $\approx$  t using st indis-trans indis-sym by blast
          have discr c' using c cs discr-transC by blast
        qed
      qed
    qed
  qed

```

```

      hence (c',d) ∈ ?theta using d by blast
      thus ?thesis using s't by blast
    qed
  qed
next
show matchT-ZO c d
unfolding matchT-ZO-def proof (tactic ‹mauto-no-simp-tac @ {context}›)
  fix s t s'
  assume st: s ≈ t and cs: (c, s) → t s'
  show
    (s' ≈ t ∧ discr d) ∨
    (∃ d' t'. (d, t) → c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
    (∃ t'. (d, t) → t t' ∧ s' ≈ t')
  proof -
    have s ≈ s' using c cs discr-transT by blast
    hence s't: s' ≈ t using st indis-trans indis-sym by blast
    thus ?thesis using d by blast
  qed
qed
qed
qed
qed
thus ?thesis using assms by blast
qed

```

Atomic commands:

```

theorem Atm-ZObis[simp]:
assumes compatAtm atm
shows Atm atm ≈01 Atm atm
by (metis Atm-Sbis assms bis-imp)

```

Sequential composition:

```

definition thetaSeqZO where
thetaSeqZO ≡
  {(c1 ;; c2, d1 ;; d2) | c1 c2 d1 d2. c1 ≈01T d1 ∧ c2 ≈01 d2}

```

```

lemma thetaSeqZO-sym:
sym thetaSeqZO
unfolding thetaSeqZO-def sym-def using ZObisT-Sym ZObis-Sym by blast

```

```

lemma thetaSeqZO-ZOretr:
thetaSeqZO ⊆ ZOretr (thetaSeqZO Un ZObis)
proof -
  {fix c1 c2 d1 d2
   assume c1d1: c1 ≈01T d1 and c2d2: c2 ≈01 d2
   hence matchC-ZOC1: matchC-ZOC ZObisT c1 d1 and matchC-ZO2: matchC-ZO
ZObis c2 d2
   and matchT-T1: matchT-T c1 d1 and matchT-ZO2: matchT-ZO c2 d2
   using ZObisT-matchC-ZOC ZObisT-matchT-T ZObis-matchC-ZO ZObis-matchT-ZO

```

```

by auto
have (c1 ;; c2, d1 ;; d2) ∈ ZOretr (thetaSeqZO Un ZObis)
unfolding ZOretr-def proof (clarify, intro conjI)
show matchC-ZO (thetaSeqZO Un ZObis) (c1 ;; c2) (d1 ;; d2)
unfolding matchC-ZO-def proof (tactic ‹mauto-no-simp-tac @{context}›)
  fix s t c' s'
  assume st: s ≈ t assume (c1 ;; c2, s) →c (c', s')
  thus
    (s' ≈ t ∧ (c', d1 ;; d2) ∈ thetaSeqZO Un ZObis) ∨
    (∃ d' t'. (d1 ;; d2, t) →c (d', t') ∧ s' ≈ t' ∧ (c', d') ∈ thetaSeqZO Un ZObis)
∨
  (∃ t'. (d1 ;; d2, t) →t t' ∧ s' ≈ t' ∧ discr c')
apply – proof (erule Seq-transC-invert)
  fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = c1' ;; c2
  hence
    (s' ≈ t ∧ c1' ≈01T d1) ∨
    (∃ d1' t'. (d1, t) →c (d1', t') ∧ s' ≈ t' ∧ c1' ≈01T d1')
  using st matchC-ZOC1 unfolding matchC-ZOC-def by auto
  thus ?thesis unfolding c' thetaSeqZO-def
  apply – apply (tactic ‹mauto-no-simp-tac @{context}›)
  apply simp apply (metis c2d2)
  apply simp by (metis SeqC c2d2 )
next
  assume (c1, s) →t s' and c': c' = c2
  hence ∃ t'. (d1, t) →t t' ∧ s' ≈ t'
  using st matchT-T1 unfolding matchT-T-def by auto
  thus ?thesis
  unfolding c' thetaSeqZO-def
  apply – apply (tactic ‹mauto-no-simp-tac @{context}›)
  apply simp by (metis PL.SeqT c2d2)
qed
qed
qed (unfold matchT-ZO-def, auto)
}
thus ?thesis unfolding thetaSeqZO-def by auto
qed

```

```

lemma thetaSeqZO-ZObis:
thetaSeqZO ⊆ ZObis
apply (rule ZObis-coind)
using thetaSeqZO-sym thetaSeqZO-ZOretr by auto

```

```

theorem Seq-ZObisT-ZObis[simp]:
assumes c1 ≈01T d1 and c2 ≈01 d2
shows c1 ;; c2 ≈01 d1 ;; d2
using assms thetaSeqZO-ZObis unfolding thetaSeqZO-def by blast

```

```

theorem Seq-iso-ZObis[simp]:
assumes iso e and c2 ≈01 d2

```

**shows**  $e ;; c2 \approx 01 e ;; d2$   
**using** *assms* **by** *auto*

**definition** *thetaSeqZOD* **where**

*thetaSeqZOD*  $\equiv$   
 $\{(c1 ;; c2, d1 ;; d2) \mid c1 \ c2 \ d1 \ d2. \ c1 \approx 01 \ d1 \ \wedge \ \text{discr} \ c2 \ \wedge \ \text{discr} \ d2\}$

**lemma** *thetaSeqZOD-sym*:

*sym thetaSeqZOD*

**unfolding** *thetaSeqZOD-def sym-def* **using** *ZObis-Sym* **by** *blast*

**lemma** *thetaSeqZOD-ZOretr*:

*thetaSeqZOD*  $\subseteq$  *ZOretr* (*thetaSeqZOD Un ZObis*)

**proof** –

{**fix** *c1 c2 d1 d2*  
**assume** *c1d1*:  $c1 \approx 01 \ d1$  **and** *c2*:  $\text{discr} \ c2$  **and** *d2*:  $\text{discr} \ d2$   
**hence** *matchC-ZO*: *matchC-ZO ZObis c1 d1*  
**and** *matchT-ZO*: *matchT-ZO c1 d1*  
**using** *ZObis-matchC-ZO ZObis-matchT-ZO* **by** *auto*  
**have**  $(c1 ;; c2, d1 ;; d2) \in \text{ZOretr} \ (\text{thetaSeqZOD} \ \text{Un} \ \text{ZObis})$   
**unfolding** *ZOretr-def* **proof** (*clarify, intro conjI*)  
**show** *matchC-ZO* (*thetaSeqZOD Un ZObis*) (*c1 ;; c2*) (*d1 ;; d2*)  
**unfolding** *matchC-ZO-def* **proof** (*tactic*  $\langle \text{mauto-no-simp-tac} \ @\{\text{context}\} \rangle$ )  
**fix** *s t c' s'*  
**assume** *st*:  $s \approx t$  **assume**  $(c1 ;; c2, s) \rightarrow c \ (c', s')$   
**thus**  
 $(s' \approx t \ \wedge \ (c', d1 ;; d2) \in \text{thetaSeqZOD} \ \text{Un} \ \text{ZObis}) \vee$   
 $(\exists d' t'. \ (d1 ;; d2, t) \rightarrow c \ (d', t') \ \wedge \ s' \approx t' \ \wedge \ (c', d') \in \text{thetaSeqZOD} \ \text{Un} \ \text{ZObis}) \vee$   
 $(\exists t'. \ (d1 ;; d2, t) \rightarrow t \ \wedge \ s' \approx t' \ \wedge \ \text{discr} \ c')$   
**apply** – **proof**(*erule Seq-transC-invert*)  
**fix** *c1'* **assume** *c1s*:  $(c1, s) \rightarrow c \ (c1', s')$  **and** *c'*:  $c' = c1' ;; c2$   
**hence**  
 $(s' \approx t \ \wedge \ c1' \approx 01 \ d1) \vee$   
 $(\exists d' t'. \ (d1, t) \rightarrow c \ (d', t') \ \wedge \ s' \approx t' \ \wedge \ c1' \approx 01 \ d') \vee$   
 $(\exists t'. \ (d1, t) \rightarrow t \ \wedge \ s' \approx t' \ \wedge \ \text{discr} \ c1')$   
**using** *st matchC-ZO* **unfolding** *matchC-ZO-def* **by** *auto*  
**thus** *?thesis* **unfolding** *c' thetaSeqZOD-def*  
**apply** – **apply**(*tactic*  $\langle \text{mauto-no-simp-tac} \ @\{\text{context}\} \rangle$ )  
**apply** *simp apply* (*metis c2 d2*)  
**apply** *simp apply* (*metis SeqC c2 d2*)  
**apply** *simp by* (*metis SeqT c2 d2 discr-Seq discr-ZObis*)  
**next**  
**assume**  $(c1, s) \rightarrow t \ s'$  **and** *c'*:  $c' = c2$   
**hence**  
 $(s' \approx t \ \wedge \ \text{discr} \ d1) \vee$   
 $(\exists d' t'. \ (d1, t) \rightarrow c \ (d', t') \ \wedge \ s' \approx t' \ \wedge \ \text{discr} \ d') \vee$

```

      (∃ t'. (d1, t) →t t' ∧ s' ≈ t')
    using st matchT-ZO unfolding matchT-ZO-def by auto
    thus ?thesis
    unfolding c' thetaSeqZOD-def
    apply – apply (tactic ‹mauto-no-simp-tac @ {context}›)
    apply simp apply (metis c2 d2 discr-Seq discr-ZObis)
    apply simp apply (metis SeqC c2 d2 discr-Seq discr-ZObis)
    apply simp by (metis SeqT c2 d2 discr-ZObis)
  qed
  qed
  qed (unfold matchT-ZO-def, auto)
}
thus ?thesis unfolding thetaSeqZOD-def by auto
qed

```

**lemma** *thetaSeqZOD-ZObis*:  
*thetaSeqZOD* ⊆ *ZObis*  
**apply** (rule *ZObis-coind*)  
**using** *thetaSeqZOD-sym thetaSeqZOD-ZOretr* **by** *auto*

**theorem** *Seq-ZObis-discr[simp]*:  
**assumes** *c1 ≈01 d1 and discr c2 and discr d2*  
**shows** *c1 ;; c2 ≈01 d1 ;; d2*  
**using** *assms thetaSeqZOD-ZObis* **unfolding** *thetaSeqZOD-def* **by** *blast*

Conditional:

**definition** *thetaIfZO* **where**  
*thetaIfZO* ≡  
 {(If *tst c1 c2*, If *tst d1 d2*) | *tst c1 c2 d1 d2. compatTst tst* ∧ *c1 ≈01 d1* ∧ *c2 ≈01 d2*}

**lemma** *thetaIfZO-sym*:  
*sym thetaIfZO*  
**unfolding** *thetaIfZO-def sym-def* **using** *ZObis-Sym* **by** *blast*

**lemma** *thetaIfZO-ZOretr*:  
*thetaIfZO* ⊆ *ZOretr (thetaIfZO Un ZObis)*  
**proof** –  
 {**fix** *tst c1 c2 d1 d2*  
   **assume** *tst: compatTst tst and c1d1: c1 ≈01 d1 and c2d2: c2 ≈01 d2*  
   **hence** *matchC-ZO1: matchC-ZO ZObis c1 d1 and matchC-ZO2: matchC-ZO ZObis c2 d2*  
   **and** *matchT-ZO1: matchT-ZO c1 d1 and matchT-ZO2: matchT-ZO c2 d2*  
   **using** *ZObis-matchC-ZO ZObis-matchT-ZO* **by** *auto*  
   **have** (If *tst c1 c2*, If *tst d1 d2*) ∈ *ZOretr (thetaIfZO Un ZObis)*  
   **unfolding** *ZOretr-def* **proof** (clarify, intro conjI)  
   **show** *matchC-ZO (thetaIfZO Un ZObis) (If tst c1 c2) (If tst d1 d2)*  
   **unfolding** *matchC-ZO-def* **proof** (tactic ‹mauto-no-simp-tac @ {context}›)  
   **fix** *s t c' s'*

```

assume st:  $s \approx t$  assume (If tst c1 c2, s)  $\rightarrow c$  (c', s')
thus
( $s' \approx t \wedge (c', \text{If } \textit{tst} \textit{d1} \textit{d2}) \in \textit{thetaIfZO} \textit{Un} \textit{ZObis}$ )  $\vee$ 
( $\exists d' t'. (\text{If } \textit{tst} \textit{d1} \textit{d2}, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge (c', d') \in \textit{thetaIfZO} \textit{Un}$ 
ZObis)  $\vee$ 
( $\exists t'. (\text{If } \textit{tst} \textit{d1} \textit{d2}, t) \rightarrow t t' \wedge s' \approx t' \wedge \textit{discr} c'$ )
apply – apply (erule If-transC-invert)
unfolding thetaIfZO-def
apply simp apply (metis IfTrue c1d1 compatTst-def st tst)
apply simp by (metis IfFalse c2d2 compatTst-def st tst)
qed
qed (unfold matchT-ZO-def, auto)
}
thus ?thesis unfolding thetaIfZO-def by auto
qed

```

```

lemma thetaIfZO-ZObis:
thetaIfZO  $\subseteq$  ZObis
apply (rule ZObis-coind)
using thetaIfZO-sym thetaIfZO-ZOretr by auto

```

```

theorem If-ZObis[simp]:
assumes compatTst tst and  $c1 \approx_{01} d1$  and  $c2 \approx_{01} d2$ 
shows If tst  $c1$   $c2 \approx_{01}$  If tst  $d1$   $d2$ 
using assms thetaIfZO-ZObis unfolding thetaIfZO-def by blast

```

While loop:

01-bisimilarity does not interact with / preserve the While construct in any interesting way.

Parallel composition:

```

definition thetaParZOL1 where
thetaParZOL1  $\equiv$ 
{(Par  $c1$   $c2$ , d) |  $c1$   $c2$  d.  $c1 \approx_{01} d \wedge \textit{discr} c2$ }

```

```

lemma thetaParZOL1-ZOretr:
thetaParZOL1  $\subseteq$  ZOretr (thetaParZOL1 Un ZObis)

```

```

proof –
{fix c1 c2 d
assume c1d:  $c1 \approx_{01} d$  and c2: discr c2
hence matchC-ZO: matchC-ZO ZObis  $c1$  d
and matchT-ZO: matchT-ZO  $c1$  d
using ZObis-matchC-ZO ZObis-matchT-ZO by auto
have (Par  $c1$   $c2$ , d)  $\in$  ZOretr (thetaParZOL1 Un ZObis)
unfolding ZOretr-def proof (clarify, intro conjI)
show matchC-ZO (thetaParZOL1  $\cup$  ZObis) (Par  $c1$   $c2$ ) d
unfolding matchC-ZO-def proof (tactic  $\langle$ mauto-no-simp-tac  $\@$ {context} $\rangle$ )
fix  $s$   $t$   $c'$   $s'$ 
assume st:  $s \approx t$  assume (Par  $c1$   $c2$ , s)  $\rightarrow c$  (c', s')

```

**thus**  
 $(s' \approx t \wedge (c', d) \in \text{thetaParZOL1} \cup \text{ZObis}) \vee$   
 $(\exists d' t'. (d, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaParZOL1} \cup \text{ZObis}) \vee$   
 $(\exists t'. (d, t) \rightarrow t t' \wedge s' \approx t' \wedge \text{discr } c')$   
**apply – proof**(erule Par-transC-invert)  
**fix**  $c1'$  **assume**  $(c1, s) \rightarrow c (c1', s')$  **and**  $c': c' = \text{Par } c1' c2$   
**hence**  
 $(s' \approx t \wedge c1' \approx 01 d) \vee$   
 $(\exists d' t'. (d, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge c1' \approx 01 d') \vee$   
 $(\exists t'. (d, t) \rightarrow t t' \wedge s' \approx t' \wedge \text{discr } c1')$   
**using** *st matchC-ZO unfolding matchC-ZO-def* **by** *blast*  
**thus** *?thesis unfolding thetaParZOL1-def*  
**apply – apply**(elim disjE exE conjE)  
**apply simp apply** (metis c2 c')  
**apply simp apply** (metis c2 c')  
**apply simp by** (metis c' c2 discr-Par)  
**next**  
**assume**  $(c1, s) \rightarrow t s'$  **and**  $c': c' = c2$   
**hence**  
 $(s' \approx t \wedge \text{discr } d) \vee$   
 $(\exists d' t'. (d, t) \rightarrow c (d', t') \wedge s' \approx t' \wedge \text{discr } d') \vee$   
 $(\exists t'. (d, t) \rightarrow t t' \wedge s' \approx t')$   
**using** *st matchT-ZO unfolding matchT-ZO-def* **by** *blast*  
**thus** *?thesis unfolding thetaParZOL1-def*  
**apply – apply**(elim disjE exE conjE)  
**apply simp apply** (metis c' c2 discr-ZObis)  
**apply simp apply** (metis c' c2 discr-ZObis)  
**apply simp by** (metis c' c2)  
**next**  
**fix**  $c2'$  **assume**  $c2s: (c2, s) \rightarrow c (c2', s')$  **and**  $c': c' = \text{Par } c1 c2'$   
**hence**  $s \approx s'$  **using** *c2 discr-transC-indis* **by** *blast*  
**hence**  $s't: s' \approx t$  **using** *st indis-sym indis-trans* **by** *blast*  
**have** *discr c2'* **using** *c2 c2s discr-transC* **by** *blast*  
**thus** *?thesis using s't c1d unfolding thetaParZOL1-def c' by simp*  
**next**  
**assume**  $(c2, s) \rightarrow t s'$  **and**  $c': c' = c1$   
**hence**  $s \approx s'$  **using** *c2 discr-transT* **by** *blast*  
**hence**  $s't: s' \approx t$  **using** *st indis-sym indis-trans* **by** *blast*  
**thus** *?thesis using c1d unfolding thetaParZOL1-def c' by simp*  
**qed**  
**qed**  
**qed** (*unfold matchT-ZO-def, auto*)  
**}**  
**thus** *?thesis unfolding thetaParZOL1-def by blast*  
**qed**

**lemma** *thetaParZOL1-converse-ZOretr*:  
 $\text{thetaParZOL1} \hat{\sim}^{-1} \subseteq \text{ZOretr } (\text{thetaParZOL1} \hat{\sim}^{-1} \text{ Un } \text{ZObis})$   
**proof –**

```

{fix c1 c2 d
  assume c1d: c1 ≈01 d and c2: discr c2
  hence matchC-ZO: matchC-ZO ZObis d c1
    and matchT-ZO: matchT-ZO d c1
  using ZObis-matchC-ZO-rev ZObis-matchT-ZO-rev by auto
  have (d, Par c1 c2) ∈ ZOretr (thetaParZOL1-1 ∪ ZObis)
  unfolding ZOretr-def proof (clarify, intro conjI)
    show matchC-ZO (thetaParZOL1-1 ∪ ZObis) d (Par c1 c2)
    unfolding matchC-ZO-def2 ZObis-converse proof (tactic ‹mauto-no-simp-tac
@{context}›)
      fix s t d' t'
      assume s ≈ t and (d, t) →c (d', t')
      hence
        (s ≈ t' ∧ d' ≈01 c1) ∨
        (∃ c' s'. (c1, s) →c (c', s') ∧ s' ≈ t' ∧ d' ≈01 c') ∨
        (∃ s'. (c1, s) →t s' ∧ s' ≈ t' ∧ discr d')
      using matchC-ZO unfolding matchC-ZO-def2 by auto
      thus
        (s ≈ t' ∧ (Par c1 c2, d') ∈ thetaParZOL1 ∪ ZObis) ∨
        (∃ c' s'. (Par c1 c2, s) →c (c', s') ∧ s' ≈ t' ∧ (c', d') ∈ thetaParZOL1 ∪
ZObis) ∨
        (∃ s'. (Par c1 c2, s) →t s' ∧ s' ≈ t' ∧ discr d')
      unfolding thetaParZOL1-def
      apply – apply (tactic ‹mauto-no-simp-tac @{context}›)
      apply simp apply (metis ZObis-Sym c2)
      apply simp apply (metis ParCL ZObis-sym c2 sym-def)
      apply simp by (metis ParTL c2 discr-ZObis)
    qed
  next
  show matchT-ZO d (Par c1 c2)
  unfolding matchT-ZO-def2 ZObis-converse proof (tactic ‹mauto-no-simp-tac
@{context}›)
    fix s t t'
    assume s ≈ t and (d, t) →t t'
    hence
      (s ≈ t' ∧ discr c1) ∨
      (∃ c' s'. (c1, s) →c (c', s') ∧ s' ≈ t' ∧ discr c') ∨
      (∃ s'. (c1, s) →t s' ∧ s' ≈ t')
    using matchT-ZO unfolding matchT-ZO-def2 by auto
    thus
      (s ≈ t' ∧ discr (Par c1 c2)) ∨
      (∃ c' s'. (Par c1 c2, s) →c (c', s') ∧ s' ≈ t' ∧ discr c') ∨
      (∃ s'. (Par c1 c2, s) →t s' ∧ s' ≈ t')
    apply – apply (tactic ‹mauto-no-simp-tac @{context}›)
    apply simp apply (metis c2 discr-Par)
    apply simp apply (metis ParCL c2 discr-Par)
    apply simp by (metis ParTL c2)
  qed
qed

```



```

}
thus ?thesis unfolding thetaParZOL1-def by blast
qed

```

```

lemma thetaParZOL1-ZObis:
thetaParZOL1  $\subseteq$  ZObis
apply(rule ZObis-coind2)
using thetaParZOL1-ZOretr thetaParZOL1-converse-ZOretr by auto

```

```

theorem Par-ZObis-discrL1[simp]:
assumes c1  $\approx$ 01 d and discr c2
shows Par c1 c2  $\approx$ 01 d
using assms thetaParZOL1-ZObis unfolding thetaParZOL1-def by blast

```

```

theorem Par-ZObis-discrR1[simp]:
assumes c  $\approx$ 01 d1 and discr d2
shows c  $\approx$ 01 Par d1 d2
using assms Par-ZObis-discrL1 ZObis-Sym by blast

```

```

definition thetaParZOL2 where
thetaParZOL2  $\equiv$ 
{(Par c1 c2, d) | c1 c2 d. discr c1  $\wedge$  c2  $\approx$ 01 d}

```

```

lemma thetaParZOL2-ZOretr:
thetaParZOL2  $\subseteq$  ZOretr (thetaParZOL2 Un ZObis)
proof –

```

```

{fix c1 c2 d
assume c2d: c2  $\approx$ 01 d and c1: discr c1
hence matchC-ZO: matchC-ZO ZObis c2 d
and matchT-ZO: matchT-ZO c2 d
using ZObis-matchC-ZO ZObis-matchT-ZO by auto
have (Par c1 c2, d)  $\in$  ZOretr (thetaParZOL2 Un ZObis)
unfolding ZOretr-def proof (clarify, intro conjI)
show matchC-ZO (thetaParZOL2  $\cup$  ZObis) (Par c1 c2) d
unfolding matchC-ZO-def proof (tactic <mauto-no-simp-tac @{context}>)}
fix s t c' s'
assume st: s  $\approx$  t assume (Par c1 c2, s)  $\rightarrow$  c (c', s')
thus
(s'  $\approx$  t  $\wedge$  (c', d)  $\in$  thetaParZOL2  $\cup$  ZObis)  $\vee$ 
( $\exists$  d' t'. (d, t)  $\rightarrow$  c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  (c', d')  $\in$  thetaParZOL2  $\cup$  ZObis)  $\vee$ 
( $\exists$  t'. (d, t)  $\rightarrow$  t t'  $\wedge$  s'  $\approx$  t'  $\wedge$  discr c')
apply – proof(erule Par-transC-invert)
fix c1' assume c1s: (c1, s)  $\rightarrow$  c (c1', s') and c': c' = Par c1' c2
hence s  $\approx$  s' using c1 discr-transC-indis by blast
hence s't: s'  $\approx$  t using st indis-sym indis-trans by blast
have discr c1' using c1 c1s discr-transC by blast
thus ?thesis using s't c2d unfolding thetaParZOL2-def c' by simp

```

```

next
  assume (c1, s) →t s' and c': c' = c2
  hence s ≈ s' using c1 discr-transT by blast
  hence s't: s' ≈ t using st indis-sym indis-trans by blast
  thus ?thesis using c2d unfolding thetaParZOL2-def c' by simp
next
  fix c2' assume (c2, s) →c (c2', s') and c': c' = Par c1 c2'
  hence
    (s' ≈ t ∧ c2' ≈01 d) ∨
    (∃ d' t'. (d, t) →c (d', t') ∧ s' ≈ t' ∧ c2' ≈01 d') ∨
    (∃ t'. (d, t) →t t' ∧ s' ≈ t' ∧ discr c2')
  using st matchC-ZO unfolding matchC-ZO-def by blast
  thus ?thesis unfolding thetaParZOL2-def
  apply – apply (elim disjE exE conjE)
  apply simp apply (metis c1 c')
  apply simp apply (metis c1 c')
  apply simp by (metis c' c1 discr-Par)
next
  assume (c2, s) →t s' and c': c' = c1
  hence
    (s' ≈ t ∧ discr d) ∨
    (∃ d' t'. (d, t) →c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
    (∃ t'. (d, t) →t t' ∧ s' ≈ t')
  using st matchT-ZO unfolding matchT-ZO-def by blast
  thus ?thesis unfolding thetaParZOL2-def
  apply – apply (elim disjE exE conjE)
  apply simp apply (metis c' c1 discr-ZObis)
  apply simp apply (metis c' c1 discr-ZObis)
  apply simp by (metis c' c1)
qed
qed
qed (unfold matchT-ZO-def, auto)
}
thus ?thesis unfolding thetaParZOL2-def by blast
qed

lemma thetaParZOL2-converse-ZOretr:
  thetaParZOL2 ^-1 ⊆ ZOretr (thetaParZOL2 ^-1 Un ZObis)
proof –
  {fix c1 c2 d
  assume c2d: c2 ≈01 d and c1: discr c1
  hence matchC-ZO: matchC-ZO ZObis d c2
  and matchT-ZO: matchT-ZO d c2
  using ZObis-matchC-ZO-rev ZObis-matchT-ZO-rev by auto
  have (d, Par c1 c2) ∈ ZOretr (thetaParZOL2^-1 ∪ ZObis)
  unfolding ZOretr-def proof (clarify, intro conjI)
  show matchC-ZO (thetaParZOL2^-1 ∪ ZObis) d (Par c1 c2)
  unfolding matchC-ZO-def2 ZObis-converse proof (tactic ‹mauto-no-simp-tac
  @{context}›)

```

```

fix  $s\ t\ d'\ t'$ 
assume  $s \approx t$  and  $(d, t) \rightarrow c\ (d', t')$ 
hence
 $(s \approx t' \wedge d' \approx_{01} c2) \vee$ 
 $(\exists c'\ s'. (c2, s) \rightarrow c\ (c', s') \wedge s' \approx t' \wedge d' \approx_{01} c') \vee$ 
 $(\exists s'. (c2, s) \rightarrow t\ s' \wedge s' \approx t' \wedge \text{discr}\ d')$ 
using matchC-ZO unfolding matchC-ZO-def2 by auto
thus
 $(s \approx t' \wedge (\text{Par}\ c1\ c2, d') \in \text{thetaParZOL2} \cup \text{ZObis}) \vee$ 
 $(\exists c'\ s'. (\text{Par}\ c1\ c2, s) \rightarrow c\ (c', s') \wedge s' \approx t' \wedge (c', d') \in \text{thetaParZOL2} \cup$ 
ZObis)  $\vee$ 
 $(\exists s'. (\text{Par}\ c1\ c2, s) \rightarrow t\ s' \wedge s' \approx t' \wedge \text{discr}\ d')$ 
unfolding thetaParZOL2-def
apply – apply(tactic  $\langle \text{mauto-no-simp-tac } @\{\text{context}\} \rangle$ )
apply simp apply (metis ZObis-Sym  $c1$ )
apply simp apply (metis ParCR ZObis-sym  $c1$  sym-def)
apply simp by (metis ParTR  $c1$  discr-ZObis)
qed
next
show matchT-ZO  $d\ (\text{Par}\ c1\ c2)$ 
unfolding matchT-ZO-def2 ZObis-converse proof (tactic  $\langle \text{mauto-no-simp-tac } @\{\text{context}\} \rangle$ )
fix  $s\ t\ t'$ 
assume  $s \approx t$  and  $(d, t) \rightarrow t\ t'$ 
hence
 $(s \approx t' \wedge \text{discr}\ c2) \vee$ 
 $(\exists c'\ s'. (c2, s) \rightarrow c\ (c', s') \wedge s' \approx t' \wedge \text{discr}\ c') \vee$ 
 $(\exists s'. (c2, s) \rightarrow t\ s' \wedge s' \approx t')$ 
using matchT-ZO unfolding matchT-ZO-def2 by auto
thus
 $(s \approx t' \wedge \text{discr}\ (\text{Par}\ c1\ c2)) \vee$ 
 $(\exists c'\ s'. (\text{Par}\ c1\ c2, s) \rightarrow c\ (c', s') \wedge s' \approx t' \wedge \text{discr}\ c') \vee$ 
 $(\exists s'. (\text{Par}\ c1\ c2, s) \rightarrow t\ s' \wedge s' \approx t')$ 
apply – apply(tactic  $\langle \text{mauto-no-simp-tac } @\{\text{context}\} \rangle$ )
apply simp apply (metis  $c1$  discr-Par)
apply simp apply (metis ParCR  $c1$  discr-Par)
apply simp by (metis ParTR  $c1$ )
qed
qed
qed
thus ?thesis unfolding thetaParZOL2-def by blast
qed

```

**lemma** *thetaParZOL2-ZObis*:

*thetaParZOL2*  $\subseteq$  *ZObis*

**apply**(*rule* *ZObis-coind2*)

**using** *thetaParZOL2-ZOretr thetaParZOL2-converse-ZOretr* **by** *auto*

**theorem** *Par-ZObis-discrL2[simp]*:

**assumes**  $c2 \approx 01 d$  **and**  $discr\ c1$   
**shows**  $Par\ c1\ c2 \approx 01 d$   
**using** *assms thetaParZOL2-ZObis* **unfolding** *thetaParZOL2-def* **by** *blast*

**theorem** *Par-ZObis-discrR2[simp]*:  
**assumes**  $c \approx 01 d2$  **and**  $discr\ d1$   
**shows**  $c \approx 01 Par\ d1\ d2$   
**using** *assms Par-ZObis-discrL2 ZObis-Sym* **by** *blast*

**definition** *thetaParZO* **where**  
 $thetaParZO \equiv$   
 $\{(Par\ c1\ c2, Par\ d1\ d2) \mid c1\ c2\ d1\ d2. c1 \approx 01 d1 \wedge c2 \approx 01 d2\}$

**lemma** *thetaParZO-sym*:  
*sym thetaParZO*  
**unfolding** *thetaParZO-def sym-def* **using** *ZObis-Sym* **by** *blast*

**lemma** *thetaParZO-ZOretr*:  
 $thetaParZO \subseteq ZOretr\ (thetaParZO\ Un\ ZObis)$

**proof** –  
**{fix**  $c1\ c2\ d1\ d2$   
**assume**  $c1d1: c1 \approx 01 d1$  **and**  $c2d2: c2 \approx 01 d2$   
**hence**  $matchC-ZO1: matchC-ZO\ ZObis\ c1\ d1$  **and**  $matchC-ZO2: matchC-ZO\ ZObis\ c2\ d2$   
**and**  $matchT-ZO1: matchT-ZO\ c1\ d1$  **and**  $matchT-ZO2: matchT-ZO\ c2\ d2$   
**using** *ZObis-matchC-ZO ZObis-matchT-ZO* **by** *auto*  
**have**  $(Par\ c1\ c2, Par\ d1\ d2) \in ZOretr\ (thetaParZO\ Un\ ZObis)$   
**unfolding** *ZOretr-def* **proof** (*clarify, intro conjI*)  
**show**  $matchC-ZO\ (thetaParZO \cup ZObis)\ (Par\ c1\ c2)\ (Par\ d1\ d2)$   
**unfolding** *matchC-ZO-def* **proof** (*tactic <mauto-no-simp-tac @{context}>*)  
**fix**  $s\ t\ c'\ s'$   
**assume**  $st: s \approx t$  **assume**  $(Par\ c1\ c2, s) \rightarrow c\ (c', s')$   
**thus**  
 $(s' \approx t \wedge (c', Par\ d1\ d2) \in thetaParZO \cup ZObis) \vee$   
 $(\exists d'\ t'. (Par\ d1\ d2, t) \rightarrow c\ (d', t') \wedge s' \approx t' \wedge (c', d') \in thetaParZO \cup$   
 $ZObis) \vee$   
 $(\exists t'. (Par\ d1\ d2, t) \rightarrow t\ t' \wedge s' \approx t' \wedge discr\ c')$   
**apply** – **proof**(*erule Par-transC-invert*)  
**fix**  $c1'$  **assume**  $c1s: (c1, s) \rightarrow c\ (c1', s')$  **and**  $c': c' = Par\ c1'\ c2$   
**hence**  
 $(s' \approx t \wedge c1' \approx 01 d1) \vee$   
 $(\exists d'\ t'. (d1, t) \rightarrow c\ (d', t') \wedge s' \approx t' \wedge c1' \approx 01 d1) \vee$   
 $(\exists t'. (d1, t) \rightarrow t\ t' \wedge s' \approx t' \wedge discr\ c1')$   
**using** *st matchC-ZO1* **unfolding** *matchC-ZO-def* **by** *auto*  
**thus** *?thesis* **unfolding**  $c'\ thetaParZO-def$   
**apply** – **apply**(*tactic <mauto-no-simp-tac @{context}>*)  
**apply** *simp* **apply** (*metis c2d2*)

```

apply simp apply (metis ParCL c2d2)
apply simp by (metis ParTL Par-ZObis-discrL2 c2d2)
next
assume (c1, s) →t s' and c': c' = c2
hence
(s' ≈ t ∧ discr d1) ∨
(∃ d' t'. (d1, t) →c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
(∃ t'. (d1, t) →t t' ∧ s' ≈ t')
using st matchT-ZO1 unfolding matchT-ZO-def by auto
thus ?thesis
unfolding c' thetaParZO-def
apply – apply(tactic ⟨mauto-no-simp-tac @ {context}⟩)
apply simp apply (metis Par-ZObis-discrR2 c2d2)
apply simp apply (metis PL.ParCL Par-ZObis-discrR2 c2d2)
apply simp by (metis PL.ParTL c2d2)
next
fix c2' assume (c2, s) →c (c2', s') and c': c' = Par c1 c2'
hence
(s' ≈ t ∧ c2' ≈01 d2) ∨
(∃ d' t'. (d2, t) →c (d', t') ∧ s' ≈ t' ∧ c2' ≈01 d') ∨
(∃ t'. (d2, t) →t t' ∧ s' ≈ t' ∧ discr c2')
using st matchC-ZO2 unfolding matchC-ZO-def by auto
thus ?thesis
unfolding c' thetaParZO-def
apply – apply(tactic ⟨mauto-no-simp-tac @ {context}⟩)
apply simp apply (metis c1d1)
apply simp apply (metis PL.ParCR c1d1)
apply simp by (metis PL.ParTR Par-ZObis-discrL1 c1d1)
next
assume (c2, s) →t s' and c': c' = c1
hence
(s' ≈ t ∧ discr d2) ∨
(∃ d' t'. (d2, t) →c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
(∃ t'. (d2, t) →t t' ∧ s' ≈ t')
using st matchT-ZO2 unfolding matchT-ZO-def by auto
thus ?thesis
unfolding c' thetaParZO-def
apply – apply(tactic ⟨mauto-no-simp-tac @ {context}⟩)
apply simp apply (metis Par-ZObis-discrR1 c1d1)
apply simp apply (metis PL.ParCR Par-ZObis-discrR1 c1d1)
apply simp by (metis PL.ParTR c1d1)
qed
qed
qed (unfold matchT-ZO-def, auto)
}
thus ?thesis unfolding thetaParZO-def by auto
qed

```

**lemma** thetaParZO-ZObis:

*thetaParZO*  $\subseteq$  *ZObis*  
**apply**(rule *ZObis-coind*)  
**using** *thetaParZO-sym thetaParZO-ZOretr* **by** *auto*

**theorem** *Par-ZObis[simp]*:  
**assumes** *c1*  $\approx_{01}$  *d1* **and** *c2*  $\approx_{01}$  *d2*  
**shows** *Par c1 c2*  $\approx_{01}$  *Par d1 d2*  
**using** *assms thetaParZO-ZObis unfolding thetaParZO-def* **by** *blast*

### 5.5.3 WT-bisimilarity versus language constructs

Discreetness:

**theorem** *noWhile-discr-WbisT[simp]*:  
**assumes** *noWhile c1* **and** *noWhile c2*  
**and** *discr c1* **and** *discr c2*  
**shows** *c1*  $\approx_{wT}$  *c2*  
**proof** –  
**from** *assms* **have** *noWhile c1*  $\wedge$  *noWhile c2*  $\wedge$  *discr c1*  $\wedge$  *discr c2* **by** *auto*  
**then show** *?thesis*  
**proof** (*induct rule: WbisT-coinduct*)  
**case** *cont* **then show** *?case*  
**by** (*metis MtransC-Refl noWhile-transC discr-transC discr-transC-indis indis-sym indis-trans*)  
**next**  
**case** *termi* **then show** *?case*  
**by** (*metis discr-MtransT indis-sym indis-trans noWhile-MtransT transT-MtransT*)  
**qed** *simp*  
**qed**

Atomic commands:

**theorem** *Atm-WbisT*:  
**assumes** *compatAtm atm*  
**shows** *Atm atm*  $\approx_{wT}$  *Atm atm*  
**by** (*metis Atm-Sbis assms bis-imp*)

Sequential composition:

**definition** *thetaSeqWT* **where**  
*thetaSeqWT*  $\equiv$   
 $\{(c1 ;; c2, d1 ;; d2) \mid c1\ c2\ d1\ d2. c1 \approx_{wT} d1 \wedge c2 \approx_{wT} d2\}$

**lemma** *thetaSeqWT-sym*:  
*sym thetaSeqWT*  
**unfolding** *thetaSeqWT-def sym-def* **using** *WbisT-Sym* **by** *blast*

**lemma** *thetaSeqWT-WretrT*:  
*thetaSeqWT*  $\subseteq$  *WretrT (thetaSeqWT Un WbisT)*  
**proof** –  
**{fix** *c1 c2 d1 d2*

```

assume  $c1d1: c1 \approx_{wT} d1$  and  $c2d2: c2 \approx_{wT} d2$ 
hence  $matchC-MC1: matchC-MC \text{ WbisT } c1 \ d1$  and  $matchC-MC2: matchC-MC$ 
 $\text{WbisT } c2 \ d2$ 
and  $matchT-MT1: matchT-MT \ c1 \ d1$  and  $matchT-T2: matchT-MT \ c2 \ d2$ 
using  $\text{WbisT-matchC-MC}$   $\text{WbisT-matchT-MT}$  by auto
have  $(c1 ;; c2, d1 ;; d2) \in \text{WretrT } (\text{thetaSeqWT } \text{Un } \text{WbisT})$ 
unfolding  $\text{WretrT-def}$  proof (clarify, intro conjI)
show  $matchC-MC (\text{thetaSeqWT } \text{Un } \text{WbisT}) (c1 ;; c2) (d1 ;; d2)$ 
unfolding  $matchC-MC-def$  proof (tactic <mauto-no-simp-tac @\{context\})
fix  $s \ t \ c' \ s'$ 
assume  $st: s \approx t$  assume  $(c1 ;; c2, s) \rightarrow c (c', s')$ 
thus  $(\exists d' \ t'. (d1 ;; d2, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaSeqWT}$ 
 $\text{Un } \text{WbisT})$ 
apply – proof(erule Seq-transC-invert)
fix  $c1' \ \text{assume}$   $c1s: (c1, s) \rightarrow c (c1', s')$  and  $c': c' = c1' ;; c2$ 
hence  $\exists d1' \ t'. (d1, t) \rightarrow^* c (d1', t') \wedge s' \approx t' \wedge c1' \approx_{wT} d1'$ 
using  $st \ matchC-MC1$  unfolding  $matchC-MC-def$  by blast
thus ?thesis unfolding  $c' \ \text{thetaSeqWT-def}$ 
apply simp by (metis PL.Seq-MtransC c2d2)
next
assume  $(c1, s) \rightarrow t \ s'$  and  $c': c' = c2$ 
hence  $\exists t'. (d1, t) \rightarrow^* t' \wedge s' \approx t'$ 
using  $st \ matchT-MT1$  unfolding  $matchT-MT-def$  by auto
thus ?thesis
unfolding  $c' \ \text{thetaSeqWT-def}$ 
apply – apply(tactic <mauto-no-simp-tac @\{context\})
apply simp by (metis Seq-MtransT-MtransC c2d2)
qed
qed
qed (unfold matchT-MT-def, auto)
}
thus ?thesis unfolding  $\text{thetaSeqWT-def}$  by auto
qed

```

```

lemma  $\text{thetaSeqWT-WbisT}$ :
 $\text{thetaSeqWT} \subseteq \text{WbisT}$ 
apply(rule WbisT-coind)
using  $\text{thetaSeqWT-sym}$   $\text{thetaSeqWT-WretrT}$  by auto

```

```

theorem  $\text{Seq-WbisT}[simp]$ :
assumes  $c1 \approx_{wT} d1$  and  $c2 \approx_{wT} d2$ 
shows  $c1 ;; c2 \approx_{wT} d1 ;; d2$ 
using  $assms \ \text{thetaSeqWT-WbisT}$  unfolding  $\text{thetaSeqWT-def}$  by blast

```

Conditional:

**definition**  $\text{thetaIfWT}$  **where**

```

 $\text{thetaIfWT} \equiv$ 
 $\{(If \ \text{tst } c1 \ c2, \ If \ \text{tst } d1 \ d2) \mid \ \text{tst } c1 \ c2 \ d1 \ d2. \ \text{compatTst } \text{tst} \wedge c1 \approx_{wT} d1 \wedge c2$ 
 $\approx_{wT} d2\}$ 

```

**lemma** *thetaIfWT-sym*:  
*sym thetaIfWT*  
**unfolding** *thetaIfWT-def sym-def* **using** *WbisT-Sym* **by** *blast*

**lemma** *thetaIfWT-WretrT*:  
*thetaIfWT*  $\subseteq$  *WretrT* (*thetaIfWT Un WbisT*)  
**proof** –  
  {**fix** *tst c1 c2 d1 d2*  
    **assume** *tst: compatTst tst* **and** *c1d1: c1  $\approx_{wT}$  d1* **and** *c2d2: c2  $\approx_{wT}$  d2*  
    **hence** *matchC-MC1: matchC-MC WbisT c1 d1* **and** *matchC-MC2: matchC-MC WbisT c2 d2*  
    **and** *matchT-MT1: matchT-MT c1 d1* **and** *matchT-MT2: matchT-MT c2 d2*  
    **using** *WbisT-matchC-MC WbisT-matchT-MT* **by** *auto*  
    **have** (*If tst c1 c2, If tst d1 d2*)  $\in$  *WretrT* (*thetaIfWT Un WbisT*)  
    **unfolding** *WretrT-def* **proof** (*clarify, intro conjI*)  
    **show** *matchC-MC* (*thetaIfWT Un WbisT*) (*If tst c1 c2*) (*If tst d1 d2*)  
    **unfolding** *matchC-MC-def* **proof** (*tactic <mauto-no-simp-tac @{context}>*)  
    **fix** *s t c' s'*  
    **assume** *st: s  $\approx$  t* **assume** (*If tst c1 c2, s*)  $\rightarrow$  *c (c', s')*  
    **thus**  $\exists d' t'. (If\ tst\ d1\ d2, t) \rightarrow *c (d', t') \wedge s' \approx t' \wedge (c', d') \in \theta_{IfWT}$   
*Un WbisT*  
    **apply** – **apply**(*erule If-transC-invert*)  
    **unfolding** *thetaIfWT-def*  
    **apply** *simp* **apply** (*metis IfTrue c1d1 compatTst-def st transC-MtransC tst*)  
    **apply** *simp* **by** (*metis IfFalse c2d2 compatTst-def st transC-MtransC tst*)  
    **qed**  
    **qed** (*unfold matchT-MT-def, auto*)  
  }  
  **thus** *?thesis* **unfolding** *thetaIfWT-def* **by** *auto*  
**qed**

**lemma** *thetaIfWT-WbisT*:  
*thetaIfWT*  $\subseteq$  *WbisT*  
**apply**(*rule WbisT-coind*)  
**using** *thetaIfWT-sym thetaIfWT-WretrT* **by** *auto*

**theorem** *If-WbisT[simp]*:  
**assumes** *compatTst tst* **and** *c1  $\approx_{wT}$  d1* **and** *c2  $\approx_{wT}$  d2*  
**shows** *If tst c1 c2  $\approx_{wT}$  If tst d1 d2*  
**using** *assms thetaIfWT-WbisT* **unfolding** *thetaIfWT-def* **by** *blast*

While loop:

**definition** *thetaWhileW* **where**  
*thetaWhileW*  $\equiv$   
  {(*While tst c, While tst d*) | *tst c d. compatTst tst  $\wedge$  c  $\approx_{wT}$  d*} *Un*  
  {(*c1 ;; (While tst c), d1 ;; (While tst d)*) | *tst c1 d1 c d.*  
   *compatTst tst  $\wedge$  c1  $\approx_{wT}$  d1  $\wedge$  c  $\approx_{wT}$  d*}



```

lemma thetaWhileW-sym:
  sym thetaWhileW
  unfolding thetaWhileW-def sym-def using WbisT-Sym by blast

lemma thetaWhileW-WretrT:
  thetaWhileW  $\subseteq$  WretrT (thetaWhileW Un WbisT)
proof –
  {fix tst c d
    assume tst: compatTst tst and c-d: c  $\approx$  wT d
    hence matchC-MC: matchC-MC WbisT c d
      and matchT-MT: matchT-MT c d
    using WbisT-matchC-MC WbisT-matchT-MT by auto
    have (While tst c, While tst d)  $\in$  WretrT (thetaWhileW Un WbisT)
    unfolding WretrT-def proof (clarify, intro conjI)
      show matchC-MC (thetaWhileW  $\cup$  WbisT) (While tst c) (While tst d)
      unfolding matchC-MC-def proof (tactic  $\langle$ mauto-no-simp-tac  $\@$ {context} $\rangle$ )
        fix s t c' s'
        assume st: s  $\approx$  t assume (While tst c, s)  $\rightarrow$  c (c', s')
        thus  $\exists$  d' t'. (While tst d, t)  $\rightarrow$  *c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$ 
          (c', d')  $\in$  thetaWhileW  $\cup$  WbisT
        apply – apply(erule While-transC-invert)
        unfolding thetaWhileW-def apply simp
        by (metis PL.WhileTrue PL.transC-MtransC c-d compatTst-def st tst)
      qed
    next
      show matchT-MT (While tst c) (While tst d)
      unfolding matchT-MT-def proof (tactic  $\langle$ mauto-no-simp-tac  $\@$ {context} $\rangle$ )
        fix s t s' assume st: s  $\approx$  t assume (While tst c, s)  $\rightarrow$  t s'
        thus  $\exists$  t'. (While tst d, t)  $\rightarrow$  *t t'  $\wedge$  s'  $\approx$  t'
        apply – apply(erule While-transT-invert)
        unfolding thetaWhileW-def apply simp
        by (metis WhileFalse compatTst-def st transT-MtransT tst)
      qed
    }
  }
moreover
  {fix tst c1 d1 c d
    assume tst: compatTst tst and c1d1: c1  $\approx$  wT d1 and c-d: c  $\approx$  wT d
    hence matchC-MC1: matchC-MC WbisT c1 d1 and matchC-MC: matchC-MC
WbisT c d
      and matchT-MT1: matchT-MT c1 d1 and matchT-MT: matchT-MT c d
    using WbisT-matchC-MC WbisT-matchT-MT by auto
    have (c1 ;; (While tst c), d1 ;; (While tst d))  $\in$  WretrT (thetaWhileW Un
WbisT)
    unfolding WretrT-def proof (clarify, intro conjI)
      show matchC-MC (thetaWhileW  $\cup$  WbisT) (c1 ;; (While tst c)) (d1 ;; (While
tst d))
      unfolding matchC-MC-def proof (tactic  $\langle$ mauto-no-simp-tac  $\@$ {context} $\rangle$ )
        fix s t c' s'

```

```

assume  $st: s \approx t$  assume  $(c1 ;; (While\ tst\ c), s) \rightarrow c (c', s')$ 
thus  $\exists d' t'. (d1 ;; (While\ tst\ d), t) \rightarrow *c (d', t') \wedge$ 
 $s' \approx t' \wedge (c', d') \in \theta_{WhileW} \cup W_{bisT}$ 
apply – proof(erule Seq-transC-invert)
  fix  $c1'$  assume  $(c1, s) \rightarrow c (c1', s')$  and  $c': c' = c1' ;; (While\ tst\ c)$ 
  hence  $\exists d' t'. (d1, t) \rightarrow *c (d', t') \wedge s' \approx t' \wedge c1' \approx_{wT} d'$ 
  using st matchC-MC1 unfolding matchC-MC-def by blast
  thus ?thesis
  unfolding  $c'$  thetaWhileW-def
  apply simp by (metis PL.Seq-MtransC c-d tst)
next
  assume  $(c1, s) \rightarrow t s'$  and  $c': c' = While\ tst\ c$ 
  hence  $\exists t'. (d1, t) \rightarrow *t t' \wedge s' \approx t'$ 
  using st matchT-MT1 unfolding matchT-MT-def by auto
  thus ?thesis
  unfolding  $c'$  thetaWhileW-def
  apply simp by (metis PL.Seq-MtransT-MtransC c-d tst)
qed
qed
qed (unfold matchT-MT-def, auto)
}
ultimately show ?thesis unfolding thetaWhileW-def by auto
qed

```

```

lemma thetaWhileW-WbisT:
 $\theta_{WhileW} \subseteq W_{bisT}$ 
apply(rule WbisT-coind)
using thetaWhileW-sym thetaWhileW-WretrT by auto

```

```

theorem While-WbisT[simp]:
assumes compatTst tst and  $c \approx_{wT} d$ 
shows  $While\ tst\ c \approx_{wT} While\ tst\ d$ 
using assms thetaWhileW-WbisT unfolding thetaWhileW-def by auto

```

Parallel composition:

```

definition thetaParWT where
 $\theta_{ParWT} \equiv$ 
 $\{(Par\ c1\ c2, Par\ d1\ d2) \mid c1\ c2\ d1\ d2. c1 \approx_{wT} d1 \wedge c2 \approx_{wT} d2\}$ 

```

```

lemma thetaParWT-sym:
sym thetaParWT
unfolding thetaParWT-def sym-def using WbisT-Sym by blast

```

```

lemma thetaParWT-WretrT:
 $\theta_{ParWT} \subseteq W_{retrT} (\theta_{ParWT} \cup_{n} W_{bisT})$ 
proof –
  {fix  $c1\ c2\ d1\ d2$ 
  assume  $c1d1: c1 \approx_{wT} d1$  and  $c2d2: c2 \approx_{wT} d2$ 
  hence matchC-MC1: matchC-MC WbisT c1 d1 and matchC-MC2: matchC-MC

```

```

WbisT c2 d2
  and matchT-MT1: matchT-MT c1 d1 and matchT-MT2: matchT-MT c2 d2
  using WbisT-matchC-MC WbisT-matchT-MT by auto
  have (Par c1 c2, Par d1 d2) ∈ WretrT (thetaParWT Un WbisT)
  unfolding WretrT-def proof (clarify, intro conjI)
  show matchC-MC (thetaParWT ∪ WbisT) (Par c1 c2) (Par d1 d2)
  unfolding matchC-MC-def proof (tactic ‹mauto-no-simp-tac @_{context}›)
  fix s t c' s'
  assume st: s ≈ t assume (Par c1 c2, s) →c (c', s')
  thus ∃ d' t'. (Par d1 d2, t) →*c (d', t') ∧ s' ≈ t' ∧
    (c', d') ∈ thetaParWT ∪ WbisT
  apply – proof(erule Par-transC-invert)
  fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = Par c1' c2
  hence ∃ d' t'. (d1, t) →*c (d', t') ∧ s' ≈ t' ∧ c1' ≈wT d'
  using st matchC-MC1 unfolding matchC-MC-def by blast
  thus ?thesis unfolding c' thetaParWT-def
  apply simp by (metis PL.ParCL-MtransC c2d2)
next
  assume (c1, s) →t s' and c': c' = c2
  hence ∃ t'. (d1, t) →*t t' ∧ s' ≈ t'
  using st matchT-MT1 unfolding matchT-MT-def by blast
  thus ?thesis
  unfolding c' thetaParWT-def
  apply simp by (metis PL.ParTL-MtransC c2d2)
next
  fix c2' assume (c2, s) →c (c2', s') and c': c' = Par c1 c2'
  hence ∃ d' t'. (d2, t) →*c (d', t') ∧ s' ≈ t' ∧ c2' ≈wT d'
  using st matchC-MC2 unfolding matchC-MC-def by blast
  thus ?thesis
  unfolding c' thetaParWT-def
  apply simp by (metis PL.ParCR-MtransC c1d1)
next
  assume (c2, s) →t s' and c': c' = c1
  hence ∃ t'. (d2, t) →*t t' ∧ s' ≈ t'
  using st matchT-MT2 unfolding matchT-MT-def by blast
  thus ?thesis
  unfolding c' thetaParWT-def
  apply simp by (metis PL.ParTR-MtransC c1d1)
qed
qed
qed (unfold matchT-MT-def, auto)
}
thus ?thesis unfolding thetaParWT-def by auto
qed

lemma thetaParWT-WbisT:
  thetaParWT ⊆ WbisT
  apply(rule WbisT-coind)
  using thetaParWT-sym thetaParWT-WretrT by auto

```

**theorem** *Par-WbisT*[*simp*]:  
**assumes**  $c1 \approx_{wT} d1$  **and**  $c2 \approx_{wT} d2$   
**shows**  $\text{Par } c1 \ c2 \approx_{wT} \text{Par } d1 \ d2$   
**using** *assms thetaParWT-WbisT* **unfolding** *thetaParWT-def* **by** *blast*

#### 5.5.4 T-bisimilarity versus language constructs

T-Discreetness:

**definition** *thetaFDW0* **where**  
*thetaFDW0*  $\equiv$   
 $\{(c1, c2). \text{discr0 } c1 \wedge \text{discr0 } c2\}$

**lemma** *thetaFDW0-sym*:  
*sym thetaFDW0*  
**unfolding** *thetaFDW0-def sym-def* **using** *Sbis-Sym* **by** *blast*

**lemma** *thetaFDW0-RetrT*:  
 $\text{thetaFDW0} \subseteq \text{RetrT } \text{thetaFDW0}$

**proof** –

{**fix**  $c \ d$   
**assume**  $c: \text{discr0 } c$  **and**  $d: \text{discr0 } d$   
**have**  $(c, d) \in \text{RetrT } \text{thetaFDW0}$   
**unfolding** *RetrT-def* **proof** (*clarify, intro conjI*)  
**show** *matchC-TMC thetaFDW0 c d*  
**unfolding** *matchC-TMC-def* **proof** (*tactic*  $\langle \text{mauto-no-simp-tac } @\{\text{context}\} \rangle$ )  
**fix**  $s \ t \ c' \ s'$  **assume**  $\text{mustT } c \ s \ \text{mustT } d \ t$   
 $s \approx t$  **and**  $(c, s) \rightarrow_c (c', s')$   
**thus**  $\exists d' \ t'. (d, t) \rightarrow_{*c} (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaFDW0}$   
**unfolding** *thetaFDW0-def* **apply** *simp*  
**by** (*metis MtransC-Refl noWhile-transC c d discr0-transC discr0-transC-indis*  
*indis-sym indis-trans*)  
**qed**  
**next**  
**show** *matchT-TMT c d*  
**unfolding** *matchT-TMT-def* **proof** (*tactic*  $\langle \text{mauto-no-simp-tac } @\{\text{context}\} \rangle$ )  
**fix**  $s \ t \ s'$  **assume**  $mt: \text{mustT } c \ s \ \text{mustT } d \ t$   
**and**  $st: s \approx t$  **and**  $cs: (c, s) \rightarrow_t s'$   
**obtain**  $t'$  **where**  $dt: (d, t) \rightarrow_{*t} t'$  **by** (*metis mt mustT-MtransT*)  
**hence**  $t \approx t'$  **and**  $s \approx s'$  **using**  $mt \ cs \ c \ d \ \text{discr0-transT } \text{discr0-MtransT}$  **by**  
*blast*+  
**hence**  $s' \approx t'$  **using**  $st \ \text{indis-trans } \text{indis-sym}$  **by** *blast*  
**thus**  $\exists t'. (d, t) \rightarrow_{*t} t' \wedge s' \approx t'$  **using**  $dt$  **by** *blast*  
**qed**  
**qed**  
**}**  
**thus** *?thesis* **unfolding** *thetaFDW0-def* **by** *blast*  
**qed**

**lemma** *thetaFDW0-BisT*:  
*thetaFDW0*  $\subseteq$  *BisT*  
**apply**(rule *BisT-raw-coind*)  
**using** *thetaFDW0-sym thetaFDW0-RetrT* **by** *auto*

**theorem** *discr0-BisT[simp]*:  
**assumes** *discr0 c1* **and** *discr0 c2*  
**shows** *c1*  $\approx^T$  *c2*  
**using** *assms thetaFDW0-BisT* **unfolding** *thetaFDW0-def* **by** *blast*

Atomic commands:

**theorem** *Atm-BisT*:  
**assumes** *compatAtm atm*  
**shows** *Atm atm*  $\approx^T$  *Atm atm*  
**by** (*metis assms siso0-Atm siso0-Sbis*)

Sequential composition:

**definition** *thetaSeqTT* **where**  
*thetaSeqTT*  $\equiv$   
 $\{(c1 ;; c2, d1 ;; d2) \mid c1 \ c2 \ d1 \ d2. \ c1 \approx^T \ d1 \ \wedge \ c2 \approx^T \ d2\}$

**lemma** *thetaSeqTT-sym*:  
*sym thetaSeqTT*  
**unfolding** *thetaSeqTT-def sym-def* **using** *BisT-Sym* **by** *blast*

**lemma** *thetaSeqTT-RetrT*:  
*thetaSeqTT*  $\subseteq$  *RetrT* (*thetaSeqTT*  $\cup$  *BisT*)

**proof** –  
**{fix** *c1 c2 d1 d2*  
**assume** *c1d1: c1*  $\approx^T$  *d1* **and** *c2d2: c2*  $\approx^T$  *d2*  
**hence** *matchC-TMC1: matchC-TMC BisT c1 d1* **and** *matchC-TMC2: matchC-TMC BisT c2 d2*  
**and** *matchT-TMT1: matchT-TMT c1 d1* **and** *matchT-T2: matchT-TMT c2 d2*  
**using** *BisT-matchC-TMC BisT-matchT-TMT* **by** *auto*  
**have** (*c1 ;; c2, d1 ;; d2*)  $\in$  *RetrT* (*thetaSeqTT*  $\cup$  *BisT*)  
**unfolding** *RetrT-def* **proof** (*clarify, intro conjI*)  
**show** *matchC-TMC (thetaSeqTT*  $\cup$  *BisT) (c1 ;; c2) (d1 ;; d2)*  
**unfolding** *matchC-TMC-def* **proof** (*tactic*  $\langle$ *mauto-no-simp-tac*  $\text{@}\{context\}$  $\rangle$ )  
**fix** *s t c' s'*  
**assume** *mt: mustT (c1 ;; c2) s mustT (d1 ;; d2) t*  
**and** *st: s*  $\approx$  *t*  
**hence** *mt1: mustT c1 s mustT d1 t*  
**by** (*metis mustT-Seq-L mustT-Seq-R*)  
**assume** *0: (c1 ;; c2, s)  $\rightarrow$  c (c', s')*  
**thus** ( $\exists d' t'. (d1 ;; d2, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaSeqTT} \cup \text{BisT}$ )  
**proof**(*elim Seq-transC-invert*)

```

fix  $c1'$  assume  $c1s: (c1, s) \rightarrow c (c1', s')$  and  $c': c' = c1'$  ;;  $c2$ 
hence  $\exists d1' t'. (d1, t) \rightarrow *c (d1', t') \wedge s' \approx t' \wedge c1' \approx T d1'$ 
using  $mt1\ st\ matchC\ -TMC1$  unfolding  $matchC\ -TMC\ -def$  by  $blast$ 
thus  $?thesis$  unfolding  $c'\ thetaSeqTT\ -def$ 
apply  $simp$  by  $(metis\ Seq\ -MtransC\ c2d2)$ 
next
assume  $c1: (c1, s) \rightarrow t\ s'$  and  $c': c' = c2$ 
then obtain  $t'$  where  $d1: (d1, t) \rightarrow *t\ t'$  and  $s't': s' \approx t'$ 
using  $mt1\ st\ matchT\ -TMT1$  unfolding  $matchT\ -TMT\ -def$  by  $blast$ 
hence  $mt1: mustT\ c2\ s'\ mustT\ d2\ t'$ 
apply  $(metis\ 0\ c'\ mt\ mustT\ -transC)$ 
by  $(metis\ mustT\ -Seq\ -R\ d1\ mt(2))$ 
thus  $?thesis$ 
unfolding  $c'\ thetaSeqTT\ -def$ 
apply  $-$  apply  $(tactic\ \langle mauto\ -no\ -simp\ -tac\ @\{context\} \rangle)$ 
apply  $simp$  by  $(metis\ Seq\ -MtransT\ -MtransC\ c2d2\ d1\ s't')$ 
qed
qed
qed  $(unfold\ matchT\ -TMT\ -def,\ auto)$ 
}
thus  $?thesis$  unfolding  $thetaSeqTT\ -def$  by  $auto$ 
qed

```

```

lemma  $thetaSeqTT\ -BisT$ :
 $thetaSeqTT \subseteq BisT$ 
apply  $(rule\ BisT\ -coind)$ 
using  $thetaSeqTT\ -sym\ thetaSeqTT\ -RetrT$  by  $auto$ 

```

```

theorem  $Seq\ -BisT[simp]$ :
assumes  $c1 \approx T d1$  and  $c2 \approx T d2$ 
shows  $c1 ;; c2 \approx T d1 ;; d2$ 
using  $assms\ thetaSeqTT\ -BisT$  unfolding  $thetaSeqTT\ -def$  by  $blast$ 

```

Conditional:

```

definition  $thetaIfTT$  where
 $thetaIfTT \equiv$ 
 $\{(If\ tst\ c1\ c2,\ If\ tst\ d1\ d2) \mid tst\ c1\ c2\ d1\ d2.\ compatTst\ tst \wedge c1 \approx T d1 \wedge c2 \approx T d2\}$ 

```

```

lemma  $thetaIfTT\ -sym$ :
 $sym\ thetaIfTT$ 
unfolding  $thetaIfTT\ -def\ sym\ -def$  using  $BisT\ -Sym$  by  $blast$ 

```

```

lemma  $thetaIfTT\ -RetrT$ :
 $thetaIfTT \subseteq RetrT (thetaIfTT \cup BisT)$ 
proof  $-$ 
{fix  $tst\ c1\ c2\ d1\ d2$ 
assume  $tst: compatTst\ tst$  and  $c1d1: c1 \approx T d1$  and  $c2d2: c2 \approx T d2$ 
hence  $matchC\ -TMC1: matchC\ -TMC\ BisT\ c1\ d1$  and  $matchC\ -TMC2: matchC\ -TMC$ 

```

```

BisT c2 d2
  and matchT-TMT1: matchT-TMT c1 d1 and matchT-TMT2: matchT-TMT
c2 d2
  using BisT-matchC-TMC BisT-matchT-TMT by auto
  have (If tst c1 c2, If tst d1 d2) ∈ RetrT (thetaIfTT ∪ BisT)
  unfolding RetrT-def proof (clarify, intro conjI)
  show matchC-TMC (thetaIfTT ∪ BisT) (If tst c1 c2) (If tst d1 d2)
  unfolding matchC-TMC-def proof (tactic ‹mauto-no-simp-tac @ {context}›)
  fix s t c' s'
  assume st: s ≈ t assume (If tst c1 c2, s) → c (c', s')
  thus ∃ d' t'. (If tst d1 d2, t) →* c (d', t') ∧ s' ≈ t' ∧
    (c', d') ∈ thetaIfTT ∪ BisT
  apply – apply (erule If-transC-invert)
  unfolding thetaIfTT-def
  apply simp apply (metis IfTrue c1d1 compatTst-def st transC-MtransC tst)
  apply simp by (metis IfFalse c2d2 compatTst-def st transC-MtransC tst)
  qed
  qed (unfold matchT-TMT-def, auto)
}
thus ?thesis unfolding thetaIfTT-def by auto
qed

```

```

lemma thetaIfTT-BisT:
  thetaIfTT ⊆ BisT
  apply (rule BisT-coind)
  using thetaIfTT-sym thetaIfTT-RetrT by auto

```

```

theorem If-BisT[simp]:
  assumes compatTst tst and c1 ≈T d1 and c2 ≈T d2
  shows If tst c1 c2 ≈T If tst d1 d2
  using assms thetaIfTT-BisT unfolding thetaIfTT-def by blast

```

While loop:

```

definition thetaWhileW0 where
  thetaWhileW0 ≡
  {(While tst c, While tst d) | tst c d. compatTst tst ∧ c ≈T d} ∪
  {(c1 ;; (While tst c), d1 ;; (While tst d)) | tst c1 d1 c d.
    compatTst tst ∧ c1 ≈T d1 ∧ c ≈T d}

```

```

lemma thetaWhileW0-sym:
  sym thetaWhileW0
  unfolding thetaWhileW0-def sym-def using BisT-Sym by blast

```

```

lemma thetaWhileW0-RetrT:
  thetaWhileW0 ⊆ RetrT (thetaWhileW0 ∪ BisT)
  proof –
  {fix tst c d
  assume tst: compatTst tst and c-d: c ≈T d
  hence matchC-TMC: matchC-TMC BisT c d

```

```

    and matchT-TMT: matchT-TMT c d
  using BisT-matchC-TMC BisT-matchT-TMT by auto
  have (While tst c, While tst d) ∈ RetrT (thetaWhileW0 ∪ BisT)
  unfolding RetrT-def proof (clarify, intro conjI)
  show matchC-TMC (thetaWhileW0 ∪ BisT) (While tst c) (While tst d)
  unfolding matchC-TMC-def proof (tactic ⟨mauto-no-simp-tac @{context}⟩)
    fix s t c' s'
    assume st: s ≈ t assume (While tst c, s) → c (c', s')
    thus ∃ d' t'. (While tst d, t) →* c (d', t') ∧ s' ≈ t' ∧
      (c', d') ∈ thetaWhileW0 ∪ BisT
    apply – apply(erule While-transC-invert)
    unfolding thetaWhileW0-def apply simp
    by (metis WhileTrue transC-MtransC c-d compatTst-def st tst)
  qed
next
  show matchT-TMT (While tst c) (While tst d)
  unfolding matchT-TMT-def proof (tactic ⟨mauto-no-simp-tac @{context}⟩)
    fix s t s' assume st: s ≈ t assume (While tst c, s) → t s'
    thus ∃ t'. (While tst d, t) →* t t' ∧ s' ≈ t'
    apply – apply(erule While-transT-invert)
    unfolding thetaWhileW0-def apply simp
    by (metis WhileFalse compatTst-def st transT-MtransT tst)
  qed
qed
}
moreover
{fix tst c1 d1 c d
  assume tst: compatTst tst and c1d1: c1 ≈T d1 and c-d: c ≈T d
  hence matchC-TMC1: matchC-TMC BisT c1 d1 and matchC-TMC: matchC-TMC
BisT c d
  and matchT-TMT1: matchT-TMT c1 d1 and matchT-TMT: matchT-TMT
c d
  using BisT-matchC-TMC BisT-matchT-TMT by auto
  have (c1 ;; (While tst c), d1 ;; (While tst d)) ∈ RetrT (thetaWhileW0 ∪ BisT)
  unfolding RetrT-def proof (clarify, intro conjI)
  show matchC-TMC (thetaWhileW0 ∪ BisT) (c1 ;; (While tst c)) (d1 ;; (While
tst d))
  unfolding matchC-TMC-def proof (tactic ⟨mauto-no-simp-tac @{context}⟩)
    fix s t c' s'
    assume mt: mustT (c1 ;; While tst c) s mustT (d1 ;; While tst d) t
    and st: s ≈ t
    hence mt1: mustT c1 s mustT d1 t
    by (metis mustT-Seq-L mustT-Seq-R)
    assume 0: (c1 ;; (While tst c), s) → c (c', s')
    thus ∃ d' t'. (d1 ;; (While tst d), t) →* c (d', t') ∧
      s' ≈ t' ∧ (c', d') ∈ thetaWhileW0 ∪ BisT
    apply – proof(erule Seq-transC-invert)
    fix c1' assume (c1, s) → c (c1', s') and c': c' = c1' ;; (While tst c)
    hence ∃ d' t'. (d1, t) →* c (d', t') ∧ s' ≈ t' ∧ c1' ≈T d'

```



```

    using mt1 st matchC-TMC1 unfolding matchC-TMC-def by blast
    thus ?thesis
    unfolding c' thetaWhileW0-def
    apply simp by (metis Seq-MtransC c-d tst)
  next
    assume (c1, s) →t s' and c': c' = While tst c
    then obtain t' where (d1, t) →*t t' ∧ s' ≈ t'
    using mt1 st matchT-TMT1 unfolding matchT-TMT-def by metis
    thus ?thesis
    unfolding c' thetaWhileW0-def
    apply simp by (metis Seq-MtransT-MtransC c-d tst)
  qed
qed
qed (unfold matchT-TMT-def, auto)
}
ultimately show ?thesis unfolding thetaWhileW0-def by auto
qed

```

```

lemma thetaWhileW0-BisT:
  thetaWhileW0 ⊆ BisT
  apply(rule BisT-coind)
  using thetaWhileW0-sym thetaWhileW0-RetrT by auto

```

```

theorem While-BisT[simp]:
  assumes compatTst tst and c ≈T d
  shows While tst c ≈T While tst d
  using assms thetaWhileW0-BisT unfolding thetaWhileW0-def by auto

```

Parallel composition:

```

definition thetaParTT where
  thetaParTT ≡
  {(Par c1 c2, Par d1 d2) | c1 c2 d1 d2. c1 ≈T d1 ∧ c2 ≈T d2}

```

```

lemma thetaParTT-sym:
  sym thetaParTT
  unfolding thetaParTT-def sym-def using BisT-Sym by blast

```

```

lemma thetaParTT-RetrT:
  thetaParTT ⊆ RetrT (thetaParTT ∪ BisT)
  proof –
    {fix c1 c2 d1 d2
     assume c1d1: c1 ≈T d1 and c2d2: c2 ≈T d2
     hence matchC-TMC1: matchC-TMC BisT c1 d1 and matchC-TMC2: matchC-TMC
     BisT c2 d2
     and matchT-TMT1: matchT-TMT c1 d1 and matchT-TMT2: matchT-TMT
     c2 d2
     using BisT-matchC-TMC BisT-matchT-TMT by auto
     have (Par c1 c2, Par d1 d2) ∈ RetrT (thetaParTT ∪ BisT)
     unfolding RetrT-def proof (clarify, intro conjI)

```

```

show matchC-TMC (thetaParTT  $\cup$  BisT) (Par c1 c2) (Par d1 d2)
unfolding matchC-TMC-def proof (tactic ‹mauto-no-simp-tac @{context}›)
  fix s t c' s'
  assume mustT (Par c1 c2) s and mustT (Par d1 d2) t
  and st: s  $\approx$  t
  hence mt: mustT c1 s mustT c2 s
    mustT d1 t mustT d2 t
  by (metis mustT-Par-L mustT-Par-R)+
  assume (Par c1 c2, s)  $\rightarrow$ c (c', s')
  thus  $\exists$  d' t'. (Par d1 d2, t)  $\rightarrow$ *c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$ 
    (c', d')  $\in$  thetaParTT  $\cup$  BisT
  proof(elim Par-transC-invert)
    fix c1' assume c1s: (c1, s)  $\rightarrow$ c (c1', s') and c': c' = Par c1' c2
    hence  $\exists$  d' t'. (d1, t)  $\rightarrow$ *c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  c1'  $\approx$ T d'
    using mt st matchC-TMC1 unfolding matchC-TMC-def by blast
    thus ?thesis unfolding c' thetaParTT-def
    apply simp by (metis ParCL-MtransC c2d2)
  next
    assume (c1, s)  $\rightarrow$ t s' and c': c' = c2
    hence  $\exists$  t'. (d1, t)  $\rightarrow$ *t t'  $\wedge$  s'  $\approx$  t'
    using mt st matchT-TMT1 unfolding matchT-TMT-def by blast
    thus ?thesis
    unfolding c' thetaParTT-def
    apply simp by (metis PL.ParTL-MtransC c2d2)
  next
    fix c2' assume (c2, s)  $\rightarrow$ c (c2', s') and c': c' = Par c1 c2'
    hence  $\exists$  d' t'. (d2, t)  $\rightarrow$ *c (d', t')  $\wedge$  s'  $\approx$  t'  $\wedge$  c2'  $\approx$ T d'
    using mt st matchC-TMC2 unfolding matchC-TMC-def by blast
    thus ?thesis
    unfolding c' thetaParTT-def
    apply simp by (metis PL.ParCR-MtransC c1d1)
  next
    assume (c2, s)  $\rightarrow$ t s' and c': c' = c1
    hence  $\exists$  t'. (d2, t)  $\rightarrow$ *t t'  $\wedge$  s'  $\approx$  t'
    using mt st matchT-TMT2 unfolding matchT-TMT-def by blast
    thus ?thesis
    unfolding c' thetaParTT-def
    apply simp by (metis PL.ParTR-MtransC c1d1)
  qed
qed
qed (unfold matchT-TMT-def, auto)
}
thus ?thesis unfolding thetaParTT-def by auto
qed

```

```

lemma thetaParTT-BisT:
  thetaParTT  $\subseteq$  BisT
apply(rule BisT-coind)
using thetaParTT-sym thetaParTT-RetrT by auto

```

**theorem** *Par-BisT[simp]*:  
**assumes**  $c1 \approx T d1$  **and**  $c2 \approx T d2$   
**shows**  $Par\ c1\ c2 \approx T Par\ d1\ d2$   
**using** *assms thetaParTT-BisT unfolding thetaParTT-def by blast*

### 5.5.5 W-bisimilarity versus language constructs

Atomic commands:

**theorem** *Atm-Wbis[simp]*:  
**assumes** *compatAtm atm*  
**shows**  $Atm\ atm \approx w\ Atm\ atm$   
**by** (*metis Atm-Sbis assms bis-imp*)

Discreetness:

**theorem** *discr-Wbis[simp]*:  
**assumes** \*: *discr c* **and** \*\*: *discr d*  
**shows**  $c \approx w d$   
**by** (*metis \* \*\* bis-imp(4) discr-ZObis*)

Sequential composition:

**definition** *thetaSeqW* **where**  
*thetaSeqW*  $\equiv$   
 $\{(c1 ;; c2, d1 ;; d2) \mid c1\ c2\ d1\ d2. c1 \approx wT d1 \wedge c2 \approx w d2\}$

**lemma** *thetaSeqW-sym*:  
*sym thetaSeqW*  
**unfolding** *thetaSeqW-def sym-def* **using** *WbisT-Sym Wbis-Sym* **by** *blast*

**lemma** *thetaSeqW-Wretr*:  
*thetaSeqW*  $\subseteq$  *Wretr (thetaSeqW  $\cup$  Wbis)*  
**proof** –  
  **{fix**  $c1\ c2\ d1\ d2$   
  **assume**  $c1d1: c1 \approx wT d1$  **and**  $c2d2: c2 \approx w d2$   
  **hence** *matchC-MC1: matchC-MC WbisT c1 d1* **and** *matchC-W2: matchC-M*  
  *Wbis c2 d2*  
  **and** *matchT-MT1: matchT-MT c1 d1* **and** *matchT-M2: matchT-M c2 d2*  
  **using** *WbisT-matchC-MC WbisT-matchT-MT Wbis-matchC-M Wbis-matchT-M*  
  **by** *auto*  
  **have**  $(c1 ;; c2, d1 ;; d2) \in Wretr\ (thetaSeqW \cup Wbis)$   
  **unfolding** *Wretr-def* **proof** (*clarify, intro conjI*)  
  **show** *matchC-M (thetaSeqW  $\cup$  Wbis) (c1 ;; c2) (d1 ;; d2)*  
  **unfolding** *matchC-M-def* **proof** (*tactic <mauto-no-simp-tac @{context}>*)  
  **fix**  $s\ t\ c'\ s'$   
  **assume**  $st: s \approx t$  **assume**  $(c1 ;; c2, s) \rightarrow c (c', s')$   
  **thus**  
   $(\exists d'\ t'. (d1 ;; d2, t) \rightarrow *c (d', t') \wedge s' \approx t' \wedge (c', d') \in thetaSeqW \cup Wbis)$

$\vee$

```

  (∃ t'. (d1 ;; d2, t) →*t t' ∧ s' ≈ t' ∧ discr c')
apply – proof(erule Seq-transC-invert)
  fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = c1' ;; c2
  hence ∃ d1' t'. (d1, t) →*c (d1', t') ∧ s' ≈ t' ∧ c1' ≈wT d1'
  using st matchC-MC1 unfolding matchC-MC-def by blast
  thus ?thesis unfolding c' thetaSeqW-def
  apply simp by (metis PL.Seq-MtransC c2d2)
next
  assume (c1, s) →t s' and c': c' = c2
  hence ∃ t'. (d1, t) →*t t' ∧ s' ≈ t'
  using st matchT-MT1 unfolding matchT-MT-def by auto
  thus ?thesis
  unfolding c' thetaSeqW-def
  apply simp by (metis PL.Seq-MtransT-MtransC c2d2)
qed
qed
qed (unfold matchT-M-def, auto)
}
thus ?thesis unfolding thetaSeqW-def by auto
qed

```

**lemma** *thetaSeqW-Wbis*:  
*thetaSeqW* ⊆ *Wbis*  
**apply**(rule *Wbis-coind*)  
**using** *thetaSeqW-sym thetaSeqW-Wretr* **by** auto

**theorem** *Seq-WbisT-Wbis[simp]*:  
**assumes** *c1* ≈wT *d1* **and** *c2* ≈w *d2*  
**shows** *c1* ;; *c2* ≈w *d1* ;; *d2*  
**using** *assms thetaSeqW-Wbis* **unfolding** *thetaSeqW-def* **by** blast

**theorem** *Seq-iso-Wbis[simp]*:  
**assumes** *siso* *e* **and** *c2* ≈w *d2*  
**shows** *e* ;; *c2* ≈w *e* ;; *d2*  
**using** *assms* **by** auto

**definition** *thetaSeqWD* **where**  
*thetaSeqWD* ≡  
 {(*c1* ;; *c2*, *d1* ;; *d2*) | *c1* *c2* *d1* *d2*. *c1* ≈w *d1* ∧ *discr* *c2* ∧ *discr* *d2*}

**lemma** *thetaSeqWD-sym*:  
*sym thetaSeqWD*  
**unfolding** *thetaSeqWD-def sym-def* **using** *Wbis-Sym* **by** blast

**lemma** *thetaSeqWD-Wretr*:  
*thetaSeqWD* ⊆ *Wretr* (*thetaSeqWD* ∪ *Wbis*)  
**proof** –

```

{fix c1 c2 d1 d2
  assume c1d1: c1 ≈w d1 and c2: discr c2 and d2: discr d2
  hence matchC-M: matchC-M Wbis c1 d1
    and matchT-M: matchT-M c1 d1
  using Wbis-matchC-M Wbis-matchT-M by auto
  have (c1 ;; c2, d1 ;; d2) ∈ Wretr (thetaSeqWD ∪ Wbis)
  unfolding Wretr-def proof (clarify, intro conjI)
  show matchC-M (thetaSeqWD ∪ Wbis) (c1 ;; c2) (d1 ;; d2)
  unfolding matchC-M-def proof (tactic ‹mauto-no-simp-tac @{context}›)
    fix s t c' s'
    assume st: s ≈ t assume (c1 ;; c2, s) →c (c', s')
    thus
      (∃ d' t'. (d1 ;; d2, t) →*c (d', t') ∧ s' ≈ t' ∧ (c', d') ∈ thetaSeqWD ∪ Wbis)
∨
      (∃ t'. (d1 ;; d2, t) →*t t' ∧ s' ≈ t' ∧ discr c')
  apply – proof(erule Seq-transC-invert)
  fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = c1' ;; c2
  hence
    (∃ d' t'. (d1, t) →*c (d', t') ∧ s' ≈ t' ∧ c1' ≈w d') ∨
    (∃ t'. (d1, t) →*t t' ∧ s' ≈ t' ∧ discr c1')
  using st matchC-M unfolding matchC-M-def by blast
  thus ?thesis unfolding c' thetaSeqWD-def
  apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
  apply simp apply (metis PL.Seq-MtransC c2 d2)
  apply simp by (metis PL.Seq-MtransT-MtransC c2 d2 discr-Seq discr-Wbis)
next
  assume (c1, s) →t s' and c': c' = c2
  hence
    (∃ d' t'. (d1, t) →*c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
    (∃ t'. (d1, t) →*t t' ∧ s' ≈ t')
  using st matchT-M unfolding matchT-M-def by blast
  thus ?thesis
  unfolding c' thetaSeqWD-def
  apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
  apply simp apply (metis PL.Seq-MtransC c2 d2 discr-Seq discr-Wbis)
  apply simp by (metis PL.Seq-MtransT-MtransC c2 d2 discr-Wbis)
qed
qed
qed (unfold matchT-M-def, auto)
}
thus ?thesis unfolding thetaSeqWD-def by auto
qed

```

```

lemma thetaSeqWD-Wbis:
  thetaSeqWD ⊆ Wbis
  apply(rule Wbis-coind)
  using thetaSeqWD-sym thetaSeqWD-Wretr by auto

```

```

theorem Seq-Wbis-discr[simp]:

```

**assumes**  $c1 \approx_w d1$  **and**  $discr\ c2$  **and**  $discr\ d2$   
**shows**  $c1 ;; c2 \approx_w d1 ;; d2$   
**using** *assms* *thetaSeqWD-Wbis* **unfolding** *thetaSeqWD-def* **by** *blast*

Conditional:

**definition** *thetaIfW* **where**

$thetaIfW \equiv$   
 $\{(If\ tst\ c1\ c2,\ If\ tst\ d1\ d2) \mid tst\ c1\ c2\ d1\ d2.\ compatTst\ tst \wedge c1 \approx_w d1 \wedge c2 \approx_w d2\}$

**lemma** *thetaIfW-sym*:

*sym* *thetaIfW*

**unfolding** *thetaIfW-def* *sym-def* **using** *Wbis-Sym* **by** *blast*

**lemma** *thetaIfW-Wretr*:

$thetaIfW \subseteq Wretr\ (thetaIfW \cup Wbis)$

**proof** –

{**fix**  $tst\ c1\ c2\ d1\ d2$   
**assume**  $tst: compatTst\ tst$  **and**  $c1d1: c1 \approx_w d1$  **and**  $c2d2: c2 \approx_w d2$   
**hence**  $matchC-M1: matchC-M\ Wbis\ c1\ d1$  **and**  $matchC-M2: matchC-M\ Wbis\ c2\ d2$   
**and**  $matchT-M1: matchT-M\ c1\ d1$  **and**  $matchT-M2: matchT-M\ c2\ d2$   
**using** *Wbis-matchC-M* *Wbis-matchT-M* **by** *auto*  
**have**  $(If\ tst\ c1\ c2,\ If\ tst\ d1\ d2) \in Wretr\ (thetaIfW \cup Wbis)$   
**unfolding** *Wretr-def* **proof** (*clarify*, *intro* *conjI*)  
**show**  $matchC-M\ (thetaIfW \cup Wbis)\ (If\ tst\ c1\ c2)\ (If\ tst\ d1\ d2)$   
**unfolding** *matchC-M-def* **proof** (*tactic*  $\langle mauto-no-simp-tac\ @\{context\} \rangle$ )  
**fix**  $s\ t\ c'\ s'$   
**assume**  $st: s \approx t$  **assume**  $(If\ tst\ c1\ c2,\ s) \rightarrow c\ (c',\ s')$   
**thus**  
 $(\exists d'\ t'. (If\ tst\ d1\ d2,\ t) \rightarrow *c\ (d',\ t') \wedge s' \approx t' \wedge (c',\ d') \in thetaIfW \cup Wbis)$   
 $\vee$   
 $(\exists t'. (If\ tst\ d1\ d2,\ t) \rightarrow *t\ t' \wedge s' \approx t' \wedge discr\ c')$   
**apply** – **apply**(*erule* *If-transC-invert*)  
**unfolding** *thetaIfW-def*  
**apply** *simp* **apply** (*metis* *IfTrue* *c1d1* *compatTst-def* *st* *transC-MtransC* *tst*)  
**apply** *simp* **by** (*metis* *IfFalse* *c2d2* *compatTst-def* *st* *transC-MtransC* *tst*)  
**qed**  
**qed** (*unfold* *matchT-M-def*, *auto*)  
**}**  
**thus** *?thesis* **unfolding** *thetaIfW-def* **by** *auto*  
**qed**

**lemma** *thetaIfW-Wbis*:

$thetaIfW \subseteq Wbis$

**apply**(*rule* *Wbis-coind*)

**using** *thetaIfW-sym* *thetaIfW-Wretr* **by** *auto*

**theorem** *If-Wbis[simp]*:

**assumes** *compatTst tst and c1 ≈<sub>w</sub> d1 and c2 ≈<sub>w</sub> d2*  
**shows** *If tst c1 c2 ≈<sub>w</sub> If tst d1 d2*  
**using** *assms thetaIfW-Wbis unfolding thetaIfW-def by blast*

While loop:

Again, w-bisimilarity does not interact with / preserve the While construct in any interesting way.

Parallel composition:

**definition** *thetaParWL1 where*

*thetaParWL1*  $\equiv$   
 $\{(Par\ c1\ c2,\ d) \mid c1\ c2\ d.\ c1\ \approx_w\ d\ \wedge\ discr\ c2\}$

**lemma** *thetaParWL1-Wretr:*

*thetaParWL1*  $\subseteq$  *Wretr (thetaParWL1  $\cup$  Wbis)*

**proof** –

**{fix** *c1 c2 d*  
**assume** *c1d: c1 ≈<sub>w</sub> d and c2: discr c2*  
**hence** *matchC-M: matchC-M Wbis c1 d*  
**and** *matchT-M: matchT-M c1 d*  
**using** *Wbis-matchC-M Wbis-matchT-M by auto*  
**have** *(Par c1 c2, d) ∈ Wretr (thetaParWL1  $\cup$  Wbis)*  
**unfolding** *Wretr-def proof (clarify, intro conjI)*  
**show** *matchC-M (thetaParWL1  $\cup$  Wbis) (Par c1 c2) d*  
**unfolding** *matchC-M-def proof (tactic ⟨mauto-no-simp-tac @{context}⟩)*  
**fix** *s t c' s'*  
**assume** *st: s ≈ t assume (Par c1 c2, s) →<sub>c</sub> (c', s')*  
**thus**  
 $(\exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge (c', d') \in \text{thetaParWL1} \cup \text{Wbis}) \vee$   
 $(\exists t'. (d, t) \rightarrow^* t' \wedge s' \approx t' \wedge \text{discr } c')$   
**apply** – **proof**(*erule Par-transC-invert*)  
**fix** *c1' assume (c1, s) →<sub>c</sub> (c1', s') and c': c' = Par c1' c2*  
**hence**  
 $(\exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge c1' \approx_w d') \vee$   
 $(\exists t'. (d, t) \rightarrow^* t' \wedge s' \approx t' \wedge \text{discr } c1')$   
**using** *st matchC-M unfolding matchC-M-def by blast*  
**thus** *?thesis unfolding thetaParWL1-def*  
**apply** – **apply**(*elim disjE exE conjE*)  
**apply** *simp apply (metis c2 c')*  
**apply** *simp by (metis c' c2 discr-Par)*  
**next**  
**assume** *(c1, s) →<sub>t</sub> s' and c': c' = c2*  
**hence**  
 $(\exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge \text{discr } d') \vee$   
 $(\exists t'. (d, t) \rightarrow^* t' \wedge s' \approx t')$   
**using** *st matchT-M unfolding matchT-M-def by blast*  
**thus** *?thesis unfolding thetaParWL1-def*  
**apply** – **apply**(*elim disjE exE conjE*)  
**apply** *simp apply (metis c' c2 discr-Wbis)*

```

    apply simp by (metis c' c2)
  next
    fix c2' assume c2s: (c2, s) →c (c2', s') and c': c' = Par c1 c2'
    hence s ≈ s' using c2 discr-transC-indis by blast
    hence s't: s' ≈ t using st indis-sym indis-trans by blast
    have discr c2' using c2 c2s discr-transC by blast
    thus ?thesis using s't c1d unfolding thetaParWL1-def c' by auto
  next
    assume (c2, s) →t s' and c': c' = c1
    hence s ≈ s' using c2 discr-transT by blast
    hence s't: s' ≈ t using st indis-sym indis-trans by blast
    thus ?thesis using c1d unfolding thetaParWL1-def c' by auto
  qed
  qed
  qed (unfold matchT-M-def, auto)
}
thus ?thesis unfolding thetaParWL1-def by blast
qed

lemma thetaParWL1-converse-Wretr:
  thetaParWL1  $\hat{-}1 \subseteq$  Wretr (thetaParWL1  $\hat{-}1 \cup$  Wbis)
proof -
  {fix c1 c2 d
   assume c1d: c1 ≈w d and c2: discr c2
   hence matchC-M: matchC-M Wbis d c1
     and matchT-M: matchT-M d c1
   using Wbis-matchC-M-rev Wbis-matchT-M-rev by auto
   have (d, Par c1 c2) ∈ Wretr (thetaParWL1-1 ∪ Wbis)
   unfolding Wretr-def proof (clarify, intro conjI)
     show matchC-M (thetaParWL1-1 ∪ Wbis) d (Par c1 c2)
     unfolding matchC-M-def2 Wbis-converse proof (tactic ‹mauto-no-simp-tac
@{context}›)
       fix s t d' t'
       assume s ≈ t and (d, t) →c (d', t')
       hence
         (∃ c' s'. (c1, s) →*c (c', s') ∧ s' ≈ t' ∧ d' ≈w c') ∨
         (∃ s'. (c1, s) →*t s' ∧ s' ≈ t' ∧ discr d')
       using matchC-M unfolding matchC-M-def2 by blast
       thus
         (∃ c' s'. (Par c1 c2, s) →*c (c', s') ∧ s' ≈ t' ∧ (c', d') ∈ thetaParWL1 ∪
Wbis) ∨
         (∃ s'. (Par c1 c2, s) →*t s' ∧ s' ≈ t' ∧ discr d')
       unfolding thetaParWL1-def
       apply - apply (tactic ‹mauto-no-simp-tac @{context}›)
       apply simp apply (metis PL.ParCL-MtransC Wbis-Sym c2)
       apply simp by (metis PL.ParTL-MtransC c2 discr-Wbis)
     qed
   next
     show matchT-M d (Par c1 c2)

```



```

unfolding matchT-M-def2 Wbis-converse proof (tactic ‹mauto-no-simp-tac
@{context}›)
  fix s t t'
  assume s ≈ t and (d, t) →t t'
  hence
  ( $\exists c' s'. (c1, s) \rightarrow*c (c', s') \wedge s' \approx t' \wedge \text{discr } c'$ )  $\vee$ 
  ( $\exists s'. (c1, s) \rightarrow*t s' \wedge s' \approx t'$ )
  using matchT-M unfolding matchT-M-def2 by blast
  thus
  ( $\exists c' s'. (\text{Par } c1\ c2, s) \rightarrow*c (c', s') \wedge s' \approx t' \wedge \text{discr } c'$ )  $\vee$ 
  ( $\exists s'. (\text{Par } c1\ c2, s) \rightarrow*t s' \wedge s' \approx t'$ )
  apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
  apply (metis PL.ParCL-MtransC c2 discr-Par)
  by (metis PL.ParTL-MtransC c2)
  qed
qed
}
thus ?thesis unfolding thetaParWL1-def by blast
qed

```

```

lemma thetaParWL1-Wbis:
thetaParWL1  $\subseteq$  Wbis
apply(rule Wbis-coind2)
using thetaParWL1-Wretr thetaParWL1-converse-Wretr by auto

```

```

theorem Par-Wbis-discrL1[simp]:
assumes c1 ≈w d and discr c2
shows Par c1 c2 ≈w d
using assms thetaParWL1-Wbis unfolding thetaParWL1-def by blast

```

```

theorem Par-Wbis-discrR1[simp]:
assumes c ≈w d1 and discr d2
shows c ≈w Par d1 d2
using assms Par-Wbis-discrL1 Wbis-Sym by blast

```

```

definition thetaParWL2 where
thetaParWL2  $\equiv$ 
{(Par c1 c2, d) | c1 c2 d. discr c1  $\wedge$  c2 ≈w d}

```

```

lemma thetaParWL2-Wretr:
thetaParWL2  $\subseteq$  Wretr (thetaParWL2  $\cup$  Wbis)
proof –
  {fix c1 c2 d
  assume c2d: c2 ≈w d and c1: discr c1
  hence matchC-M: matchC-M Wbis c2 d
  and matchT-M: matchT-M c2 d
  using Wbis-matchC-M Wbis-matchT-M by auto

```

```

have (Par c1 c2, d) ∈ Wretr (thetaParWL2 ∪ Wbis)
unfolding Wretr-def proof (clarify, intro conjI)
show matchC-M (thetaParWL2 ∪ Wbis) (Par c1 c2) d
unfolding matchC-M-def proof (tactic ⟨mauto-no-simp-tac @{context}⟩)
  fix s t c' s'
  assume st: s ≈ t assume (Par c1 c2, s) →c (c', s')
  thus
    (∃ d' t'. (d, t) →*c (d', t') ∧ s' ≈ t' ∧ (c', d') ∈ thetaParWL2 ∪ Wbis) ∨
    (∃ t'. (d, t) →*t t' ∧ s' ≈ t' ∧ discr c')
  apply – proof(erule Par-transC-invert)
    fix c1' assume c1s: (c1, s) →c (c1', s') and c': c' = Par c1' c2
    hence s ≈ s' using c1 discr-transC-indis by blast
    hence s't: s' ≈ t using st indis-sym indis-trans by blast
    have discr c1' using c1 c1s discr-transC by blast
    thus ?thesis using s't c2d unfolding thetaParWL2-def c' by auto
  next
    assume (c1, s) →t s' and c': c' = c2
    hence s ≈ s' using c1 discr-transT by blast
    hence s't: s' ≈ t using st indis-sym indis-trans by blast
    thus ?thesis using c2d unfolding thetaParWL2-def c' by auto
  next
    fix c2' assume (c2, s) →c (c2', s') and c': c' = Par c1 c2'
    hence
      (∃ d' t'. (d, t) →*c (d', t') ∧ s' ≈ t' ∧ c2' ≈w d') ∨
      (∃ t'. (d, t) →*t t' ∧ s' ≈ t' ∧ discr c2')
    using st matchC-M unfolding matchC-M-def by blast
    thus ?thesis unfolding thetaParWL2-def
    apply – apply(elim disjE exE conjE)
    apply simp apply (metis c1 c')
    apply simp by (metis c' c1 discr-Par)
  next
    assume (c2, s) →t s' and c': c' = c1
    hence
      (∃ d' t'. (d, t) →*c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
      (∃ t'. (d, t) →*t t' ∧ s' ≈ t')
    using st matchT-M unfolding matchT-M-def by blast
    thus ?thesis unfolding thetaParWL2-def
    apply – apply(elim disjE exE conjE)
    apply simp apply (metis c' c1 discr-Wbis)
    apply simp by (metis c' c1)
  qed
qed
qed (unfold matchT-M-def, auto)
}
thus ?thesis unfolding thetaParWL2-def by blast
qed

```

**lemma** thetaParWL2-converse-Wretr:  
thetaParWL2  $\hat{\sim}$ -1  $\subseteq$  Wretr (thetaParWL2  $\hat{\sim}$ -1  $\cup$  Wbis)

```

proof –
  {fix  $c1\ c2\ d$ 
    assume  $c2d: c2 \approx_w d$  and  $c1: discr\ c1$ 
    hence  $matchC-M: matchC-M\ Wbis\ d\ c2$ 
    and  $matchT-M: matchT-M\ d\ c2$ 
    using  $Wbis-matchC-M-rev\ Wbis-matchT-M-rev$  by  $auto$ 
    have  $(d, Par\ c1\ c2) \in Wretr\ (thetaParWL2^{-1} \cup Wbis)$ 
    unfolding  $Wretr-def$  proof ( $clarify, intro\ conjI$ )
    show  $matchC-M\ (thetaParWL2^{-1} \cup Wbis)\ d\ (Par\ c1\ c2)$ 
    unfolding  $matchC-M-def2\ Wbis-converse$  proof ( $tactic\ \langle mauto-no-simp-tac$ 
@{ $context$ } $\rangle$ )
    fix  $s\ t\ d'\ t'$ 
    assume  $s \approx t$  and  $(d, t) \rightarrow c\ (d', t')$ 
    hence
    ( $\exists\ c'\ s'. (c2, s) \rightarrow *c\ (c', s') \wedge s' \approx t' \wedge d' \approx_w c'$ )  $\vee$ 
    ( $\exists\ s'. (c2, s) \rightarrow *t\ s' \wedge s' \approx t' \wedge discr\ d'$ )
    using  $matchC-M$  unfolding  $matchC-M-def2$  by  $blast$ 
    thus
    ( $\exists\ c'\ s'. (Par\ c1\ c2, s) \rightarrow *c\ (c', s') \wedge s' \approx t' \wedge (c', d') \in thetaParWL2 \cup$ 
 $Wbis)$   $\vee$ 
    ( $\exists\ s'. (Par\ c1\ c2, s) \rightarrow *t\ s' \wedge s' \approx t' \wedge discr\ d'$ )
    unfolding  $thetaParWL2-def$ 
    apply – apply( $tactic\ \langle mauto-no-simp-tac\ @\{context\}\rangle$ )
    apply  $simp$  apply ( $metis\ PL.ParCR-MtransC\ Wbis-Sym\ c1$ )
    apply  $simp$  by ( $metis\ PL.ParTR-MtransC\ c1\ discr-Wbis$ )
    qed
  next
    show  $matchT-M\ d\ (Par\ c1\ c2)$ 
    unfolding  $matchT-M-def2\ Wbis-converse$  proof ( $tactic\ \langle mauto-no-simp-tac$ 
@{ $context$ } $\rangle$ )
    fix  $s\ t\ t'$ 
    assume  $s \approx t$  and  $(d, t) \rightarrow t\ t'$ 
    hence
    ( $\exists\ c'\ s'. (c2, s) \rightarrow *c\ (c', s') \wedge s' \approx t' \wedge discr\ c'$ )  $\vee$ 
    ( $\exists\ s'. (c2, s) \rightarrow *t\ s' \wedge s' \approx t'$ )
    using  $matchT-M$  unfolding  $matchT-M-def2$  by  $blast$ 
    thus
    ( $\exists\ c'\ s'. (Par\ c1\ c2, s) \rightarrow *c\ (c', s') \wedge s' \approx t' \wedge discr\ c'$ )  $\vee$ 
    ( $\exists\ s'. (Par\ c1\ c2, s) \rightarrow *t\ s' \wedge s' \approx t'$ )
    apply – apply( $tactic\ \langle mauto-no-simp-tac\ @\{context\}\rangle$ )
    apply ( $metis\ PL.ParCR-MtransC\ c1\ discr-Par$ )
    by ( $metis\ PL.ParTR-MtransC\ c1$ )
    qed
  qed
}
thus  $?thesis$  unfolding  $thetaParWL2-def$  by  $blast$ 
qed

```

**lemma**  $thetaParWL2-Wbis$ :

*thetaParWL2*  $\subseteq$  *Wbis*  
**apply**(rule *Wbis-coind2*)  
**using** *thetaParWL2-Wretr thetaParWL2-converse-Wretr* **by** *auto*

**theorem** *Par-Wbis-discrL2[simp]*:  
**assumes** *c2*  $\approx_w$  *d* **and** *discr c1*  
**shows** *Par c1 c2*  $\approx_w$  *d*  
**using** *assms thetaParWL2-Wbis unfolding thetaParWL2-def* **by** *blast*

**theorem** *Par-Wbis-discrR2[simp]*:  
**assumes** *c*  $\approx_w$  *d2* **and** *discr d1*  
**shows** *c*  $\approx_w$  *Par d1 d2*  
**using** *assms Par-Wbis-discrL2 Wbis-Sym* **by** *blast*

**definition** *thetaParW* **where**  
*thetaParW*  $\equiv$   
 $\{(Par\ c1\ c2,\ Par\ d1\ d2) \mid c1\ c2\ d1\ d2.\ c1\ \approx_w\ d1\ \wedge\ c2\ \approx_w\ d2\}$

**lemma** *thetaParW-sym*:  
*sym thetaParW*  
**unfolding** *thetaParW-def sym-def* **using** *Wbis-Sym* **by** *blast*

**lemma** *thetaParW-Wretr*:  
*thetaParW*  $\subseteq$  *Wretr (thetaParW  $\cup$  Wbis)*

**proof** –  
**{fix** *c1 c2 d1 d2*  
**assume** *c1d1*: *c1*  $\approx_w$  *d1* **and** *c2d2*: *c2*  $\approx_w$  *d2*  
**hence** *matchC-M1*: *matchC-M Wbis c1 d1* **and** *matchC-M2*: *matchC-M Wbis c2 d2*  
**and** *matchT-M1*: *matchT-M c1 d1* **and** *matchT-M2*: *matchT-M c2 d2*  
**using** *Wbis-matchC-M Wbis-matchT-M* **by** *auto*  
**have**  $(Par\ c1\ c2,\ Par\ d1\ d2) \in Wretr\ (thetaParW\ \cup\ Wbis)$   
**unfolding** *Wretr-def* **proof** (*clarify, intro conjI*)  
**show** *matchC-M (thetaParW  $\cup$  Wbis) (Par c1 c2) (Par d1 d2)*  
**unfolding** *matchC-M-def* **proof** (*tactic <mauto-no-simp-tac @ {context}>*)  
**fix** *s t c' s'*  
**assume** *st*: *s*  $\approx$  *t* **assume**  $(Par\ c1\ c2,\ s) \rightarrow c\ (c',\ s')$   
**thus**  
 $(\exists\ d'\ t'.\ (Par\ d1\ d2,\ t) \rightarrow *c\ (d',\ t') \wedge s' \approx t' \wedge (c',\ d') \in thetaParW\ \cup\ Wbis)$   
 $\vee$   
 $(\exists\ t'.\ (Par\ d1\ d2,\ t) \rightarrow *t\ t' \wedge s' \approx t' \wedge discr\ c')$   
**apply** – **proof**(*erule Par-transC-invert*)  
**fix** *c1'* **assume** *c1s*:  $(c1,\ s) \rightarrow c\ (c1',\ s')$  **and** *c'*: *c' = Par c1' c2*  
**hence**  
 $(\exists\ d'\ t'.\ (d1,\ t) \rightarrow *c\ (d',\ t') \wedge s' \approx t' \wedge c1' \approx_w d')$   $\vee$   
 $(\exists\ t'.\ (d1,\ t) \rightarrow *t\ t' \wedge s' \approx t' \wedge discr\ c1')$   
**using** *st matchC-M1 unfolding matchC-M-def* **by** *blast*

```

thus ?thesis unfolding c' thetaParW-def
apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
apply simp apply (metis PL.ParCL-MtransC c2d2)
apply simp by (metis PL.ParTL-MtransC Par-Wbis-discrL2 c2d2)
next
assume (c1, s) →t s' and c': c' = c2
hence
(∃ d' t'. (d1, t) →*c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
(∃ t'. (d1, t) →*t t' ∧ s' ≈ t')
using st matchT-M1 unfolding matchT-M-def by blast
thus ?thesis
unfolding c' thetaParW-def
apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
apply simp apply (metis PL.ParCL-MtransC Par-Wbis-discrR2 c2d2)
apply simp by (metis PL.ParTL-MtransC c2d2)
next
fix c2' assume (c2, s) →c (c2', s') and c': c' = Par c1 c2'
hence
(∃ d' t'. (d2, t) →*c (d', t') ∧ s' ≈ t' ∧ c2' ≈w d') ∨
(∃ t'. (d2, t) →*t t' ∧ s' ≈ t' ∧ discr c2')
using st matchC-M2 unfolding matchC-M-def by blast
thus ?thesis
unfolding c' thetaParW-def
apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
apply simp apply (metis PL.ParCR-MtransC c1d1)
apply simp by (metis PL.ParTR-MtransC Par-Wbis-discrL1 c1d1)
next
assume (c2, s) →t s' and c': c' = c1
hence
(∃ d' t'. (d2, t) →*c (d', t') ∧ s' ≈ t' ∧ discr d') ∨
(∃ t'. (d2, t) →*t t' ∧ s' ≈ t')
using st matchT-M2 unfolding matchT-M-def by blast
thus ?thesis
unfolding c' thetaParW-def
apply – apply(tactic ‹mauto-no-simp-tac @{context}›)
apply simp apply (metis PL.ParCR-MtransC Par-Wbis-discrR1 c1d1)
apply simp by (metis PL.ParTR-MtransC c1d1)
qed
qed
qed (unfold matchT-M-def, auto)
}
thus ?thesis unfolding thetaParW-def by auto
qed

```

```

lemma thetaParW-Wbis:
thetaParW ⊆ Wbis
apply(rule Wbis-coind)
using thetaParW-sym thetaParW-Wretr by auto

```

**theorem** *Par-Wbis[simp]*:  
**assumes**  $c1 \approx_w d1$  **and**  $c2 \approx_w d2$   
**shows**  $Par\ c1\ c2 \approx_w Par\ d1\ d2$   
**using** *assms thetaParW-Wbis* **unfolding** *thetaParW-def* **by** *blast*

**end**

**end**  
**theory** *Syntactic-Criteria*  
**imports** *Compositionality*  
**begin**

**context** *PL-Indis*  
**begin**

**lemma** *noWhile[intro]*:  
 $noWhile\ (Atm\ atm)$   
 $noWhile\ c1 \implies noWhile\ c2 \implies noWhile\ (Seq\ c1\ c2)$   
 $noWhile\ c1 \implies noWhile\ c2 \implies noWhile\ (If\ tst\ c1\ c2)$   
 $noWhile\ c1 \implies noWhile\ c2 \implies noWhile\ (Par\ c1\ c2)$   
**by** *auto*

**lemma** *discr[intro]*:  
 $presAtm\ atm \implies discr\ (Atm\ atm)$   
 $discr\ c1 \implies discr\ c2 \implies discr\ (Seq\ c1\ c2)$   
 $discr\ c1 \implies discr\ c2 \implies discr\ (If\ tst\ c1\ c2)$   
 $discr\ c \implies discr\ (While\ tst\ c)$   
 $discr\ c1 \implies discr\ c2 \implies discr\ (Par\ c1\ c2)$   
**by** *auto*

**lemma** *siso[intro]*:  
 $compatAtm\ atm \implies siso\ (Atm\ atm)$   
 $siso\ c1 \implies siso\ c2 \implies siso\ (Seq\ c1\ c2)$   
 $compatTst\ tst \implies siso\ c1 \implies siso\ c2 \implies siso\ (If\ tst\ c1\ c2)$   
 $compatTst\ tst \implies siso\ c \implies siso\ (While\ tst\ c)$   
 $siso\ c1 \implies siso\ c2 \implies siso\ (Par\ c1\ c2)$   
**by** *auto*

**lemma** *Sbis[intro]*:  
 $compatAtm\ atm \implies Atm\ atm \approx_s Atm\ atm$   
 $c1 \approx_s c1 \implies c2 \approx_s c2 \implies Seq\ c1\ c2 \approx_s Seq\ c1\ c2$   
 $compatTst\ tst \implies c1 \approx_s c1 \implies c2 \approx_s c2 \implies If\ tst\ c1\ c2 \approx_s If\ tst\ c1\ c2$   
 $compatTst\ tst \implies c \approx_s c \implies While\ tst\ c \approx_s While\ tst\ c$   
 $c1 \approx_s c1 \implies c2 \approx_s c2 \implies Par\ c1\ c2 \approx_s Par\ c1\ c2$   
**by** *auto*

**lemma** *ZObisT*[intro]:

$compatAtm\ atm \implies Atm\ atm \approx_{01T}\ Atm\ atm$   
 $c1 \approx_{01T}\ c1 \implies c2 \approx_{01T}\ c2 \implies Seq\ c1\ c2 \approx_{01T}\ Seq\ c1\ c2$   
 $compatTst\ tst \implies c1 \approx_{01T}\ c1 \implies c2 \approx_{01T}\ c2 \implies If\ tst\ c1\ c2 \approx_{01T}\ If\ tst\ c1\ c2$   
 $compatTst\ tst \implies c \approx_{01T}\ c \implies While\ tst\ c \approx_{01T}\ While\ tst\ c$   
 $c1 \approx_{01T}\ c1 \implies c2 \approx_{01T}\ c2 \implies Par\ c1\ c2 \approx_{01T}\ Par\ c1\ c2$   
**by** *auto*

**lemma** *BisT*[intro]:

$compatAtm\ atm \implies Atm\ atm \approx_T\ Atm\ atm$   
 $c1 \approx_T\ c1 \implies c2 \approx_T\ c2 \implies Seq\ c1\ c2 \approx_T\ Seq\ c1\ c2$   
 $compatTst\ tst \implies c1 \approx_T\ c1 \implies c2 \approx_T\ c2 \implies If\ tst\ c1\ c2 \approx_T\ If\ tst\ c1\ c2$   
 $compatTst\ tst \implies c \approx_T\ c \implies While\ tst\ c \approx_T\ While\ tst\ c$   
 $c1 \approx_T\ c1 \implies c2 \approx_T\ c2 \implies Par\ c1\ c2 \approx_T\ Par\ c1\ c2$   
**by** *auto*

**lemma** *WbisT*[intro]:

$compatAtm\ atm \implies Atm\ atm \approx_{wT}\ Atm\ atm$   
 $c1 \approx_{wT}\ c1 \implies c2 \approx_{wT}\ c2 \implies Seq\ c1\ c2 \approx_{wT}\ Seq\ c1\ c2$   
 $compatTst\ tst \implies c1 \approx_{wT}\ c1 \implies c2 \approx_{wT}\ c2 \implies If\ tst\ c1\ c2 \approx_{wT}\ If\ tst\ c1\ c2$   
 $compatTst\ tst \implies c \approx_{wT}\ c \implies While\ tst\ c \approx_{wT}\ While\ tst\ c$   
 $c1 \approx_{wT}\ c1 \implies c2 \approx_{wT}\ c2 \implies Par\ c1\ c2 \approx_{wT}\ Par\ c1\ c2$   
**by** *auto*

**lemma** *ZObis*[intro]:

$compatAtm\ atm \implies Atm\ atm \approx_{01}\ Atm\ atm$   
 $c1 \approx_{01T}\ c1 \implies c2 \approx_{01}\ c2 \implies Seq\ c1\ c2 \approx_{01}\ Seq\ c1\ c2$   
 $c1 \approx_{01}\ c1 \implies discr\ c2 \implies Seq\ c1\ c2 \approx_{01}\ Seq\ c1\ c2$   
 $compatTst\ tst \implies c1 \approx_{01}\ c1 \implies c2 \approx_{01}\ c2 \implies If\ tst\ c1\ c2 \approx_{01}\ If\ tst\ c1\ c2$   
 $c1 \approx_{01}\ c1 \implies c2 \approx_{01}\ c2 \implies Par\ c1\ c2 \approx_{01}\ Par\ c1\ c2$   
**by** *auto*

**lemma** *Wbis*[intro]:

$compatAtm\ atm \implies Atm\ atm \approx_w\ Atm\ atm$   
 $c1 \approx_{wT}\ c1 \implies c2 \approx_w\ c2 \implies Seq\ c1\ c2 \approx_w\ Seq\ c1\ c2$   
 $c1 \approx_w\ c1 \implies discr\ c2 \implies Seq\ c1\ c2 \approx_w\ Seq\ c1\ c2$   
 $compatTst\ tst \implies c1 \approx_w\ c1 \implies c2 \approx_w\ c2 \implies If\ tst\ c1\ c2 \approx_w\ If\ tst\ c1\ c2$   
 $c1 \approx_w\ c1 \implies c2 \approx_w\ c2 \implies Par\ c1\ c2 \approx_w\ Par\ c1\ c2$   
**by** *auto*

**lemma** *discr-noWhile-WbisT*[intro]:  $discr\ c \implies noWhile\ c \implies c \approx_{wT}\ c$

**by** *auto*

**lemma** *siso-ZObis*[intro]:  $siso\ c \implies c \approx_{01}\ c$

by *auto*

**lemma** *WbisT-Wbis[intro]*:  $c \approx_w T c \implies c \approx_w c$   
by *auto*

**lemma** *ZObis-Wbis[intro]*:  $c \approx_{01} c \implies c \approx_w c$   
by *auto*

**lemma** *discr-BisT[intro]*:  $\text{discr } c \implies c \approx T c$   
by *auto*

**lemma** *WbisT-BisT[intro]*:  $c \approx_w T c \implies c \approx T c$   
using *bis-incl* by *auto*

**lemma** *ZObisT-ZObis[intro]*:  $c \approx_{01T} c \implies c \approx_{01} c$   
by *auto*

**lemma** *siso-ZObisT[intro]*:  $\text{siso } c \implies c \approx_{01T} c$   
by *auto*

**primrec** *SC-discr* where

$SC\text{-discr } (Atm \ atm) \quad \longleftrightarrow \text{presAtm } atm$   
|  $SC\text{-discr } (Seq \ c1 \ c2) \quad \longleftrightarrow SC\text{-discr } c1 \wedge SC\text{-discr } c2$   
|  $SC\text{-discr } (If \ tst \ c1 \ c2) \quad \longleftrightarrow SC\text{-discr } c1 \wedge SC\text{-discr } c2$   
|  $SC\text{-discr } (While \ tst \ c) \quad \longleftrightarrow SC\text{-discr } c$   
|  $SC\text{-discr } (Par \ c1 \ c2) \quad \longleftrightarrow SC\text{-discr } c1 \wedge SC\text{-discr } c2$

**primrec** *SC-siso* where

$SC\text{-siso } (Atm \ atm) \quad \longleftrightarrow \text{compatAtm } atm$   
|  $SC\text{-siso } (Seq \ c1 \ c2) \quad \longleftrightarrow SC\text{-siso } c1 \wedge SC\text{-siso } c2$   
|  $SC\text{-siso } (If \ tst \ c1 \ c2) \quad \longleftrightarrow \text{compatTst } tst \wedge SC\text{-siso } c1 \wedge SC\text{-siso } c2$   
|  $SC\text{-siso } (While \ tst \ c) \quad \longleftrightarrow \text{compatTst } tst \wedge SC\text{-siso } c$   
|  $SC\text{-siso } (Par \ c1 \ c2) \quad \longleftrightarrow SC\text{-siso } c1 \wedge SC\text{-siso } c2$

**primrec** *SC-WbisT* where

$SC\text{-WbisT } (Atm \ atm) \quad \longleftrightarrow \text{compatAtm } atm$   
|  $SC\text{-WbisT } (Seq \ c1 \ c2) \quad \longleftrightarrow (SC\text{-WbisT } c1 \wedge SC\text{-WbisT } c2) \vee$   
 $\quad \quad \quad (\text{noWhile } (Seq \ c1 \ c2) \wedge SC\text{-discr } (Seq \ c1 \ c2)) \vee$   
 $\quad \quad \quad SC\text{-siso } (Seq \ c1 \ c2)$   
|  $SC\text{-WbisT } (If \ tst \ c1 \ c2) \quad \longleftrightarrow (\text{if } \text{compatTst } \text{tst}$   
 $\quad \quad \quad \text{then } (SC\text{-WbisT } c1 \wedge SC\text{-WbisT } c2)$   
 $\quad \quad \quad \text{else } ((\text{noWhile } (If \ tst \ c1 \ c2) \wedge SC\text{-discr } (If \ tst \ c1 \ c2)) \vee$   
 $\quad \quad \quad SC\text{-siso } (If \ tst \ c1 \ c2)))$   
|  $SC\text{-WbisT } (While \ tst \ c) \quad \longleftrightarrow (\text{if } \text{compatTst } \text{tst}$   
 $\quad \quad \quad \text{then } SC\text{-WbisT } c$   
 $\quad \quad \quad \text{else } ((\text{noWhile } (While \ tst \ c) \wedge SC\text{-discr } (While \ tst \ c)) \vee$   
 $\quad \quad \quad SC\text{-siso } (While \ tst \ c)))$   
|  $SC\text{-WbisT } (Par \ c1 \ c2) \quad \longleftrightarrow (SC\text{-WbisT } c1 \wedge SC\text{-WbisT } c2) \vee$   
 $\quad \quad \quad (\text{noWhile } (Par \ c1 \ c2) \wedge SC\text{-discr } (Par \ c1 \ c2)) \vee$



*SC-iso (Par c1 c2)*

**primrec SC-ZObis where**

*SC-ZObis (Atm atm)*  $\longleftrightarrow$  *compatAtm atm*  
| *SC-ZObis (Seq c1 c2)*  $\longleftrightarrow$  (*SC-iso c1*  $\wedge$  *SC-ZObis c2*)  $\vee$   
*(SC-ZObis c1*  $\wedge$  *SC-discr c2)*  $\vee$   
*SC-discr (Seq c1 c2)*  $\vee$   
*SC-iso (Seq c1 c2)*  
| *SC-ZObis (If tst c1 c2)*  $\longleftrightarrow$  (*if compatTst tst*  
*then (SC-ZObis c1*  $\wedge$  *SC-ZObis c2)*  
*else (SC-discr (If tst c1 c2)*  $\vee$   
*SC-iso (If tst c1 c2))*)  
| *SC-ZObis (While tst c)*  $\longleftrightarrow$  *SC-discr (While tst c)*  $\vee$   
*SC-iso (While tst c)*  
| *SC-ZObis (Par c1 c2)*  $\longleftrightarrow$  (*SC-ZObis c1*  $\wedge$  *SC-ZObis c2*)  $\vee$   
*SC-discr (Par c1 c2)*  $\vee$   
*SC-iso (Par c1 c2)*

**primrec SC-Wbis where**

*SC-Wbis (Atm atm)*  $\longleftrightarrow$  *compatAtm atm*  
| *SC-Wbis (Seq c1 c2)*  $\longleftrightarrow$  (*SC-WbisT c1*  $\wedge$  *SC-Wbis c2*)  $\vee$   
*(SC-Wbis c1*  $\wedge$  *SC-discr c2)*  $\vee$   
*SC-ZObis (Seq c1 c2)*  $\vee$   
*SC-WbisT (Seq c1 c2)*  
| *SC-Wbis (If tst c1 c2)*  $\longleftrightarrow$  (*if compatTst tst*  
*then (SC-Wbis c1*  $\wedge$  *SC-Wbis c2)*  
*else (SC-ZObis (If tst c1 c2)*  $\vee$   
*SC-WbisT (If tst c1 c2))*)  
| *SC-Wbis (While tst c)*  $\longleftrightarrow$  *SC-ZObis (While tst c)*  $\vee$   
*SC-WbisT (While tst c)*  
| *SC-Wbis (Par c1 c2)*  $\longleftrightarrow$  (*SC-Wbis c1*  $\wedge$  *SC-Wbis c2*)  $\vee$   
*SC-ZObis (Par c1 c2)*  $\vee$   
*SC-WbisT (Par c1 c2)*

**primrec SC-BisT where**

*SC-BisT (Atm atm)*  $\longleftrightarrow$  *compatAtm atm*  
| *SC-BisT (Seq c1 c2)*  $\longleftrightarrow$  (*SC-BisT c1*  $\wedge$  *SC-BisT c2*)  $\vee$   
*SC-discr (Seq c1 c2)*  $\vee$   
*SC-WbisT (Seq c1 c2)*  
| *SC-BisT (If tst c1 c2)*  $\longleftrightarrow$  (*if compatTst tst*  
*then (SC-BisT c1*  $\wedge$  *SC-BisT c2)*  
*else (SC-discr (If tst c1 c2)*  $\vee$   
*SC-WbisT (If tst c1 c2))*)  
| *SC-BisT (While tst c)*  $\longleftrightarrow$  (*if compatTst tst*  
*then SC-BisT c*  
*else (SC-discr (While tst c)*  $\vee$   
*SC-WbisT (While tst c))*)  
| *SC-BisT (Par c1 c2)*  $\longleftrightarrow$  (*SC-BisT c1*  $\wedge$  *SC-BisT c2*)  $\vee$   
*SC-discr (Par c1 c2)*  $\vee$

*SC-WbisT (Par c1 c2)*

**theorem** *SC-discr[intro]: SC-discr c  $\implies$  discr c*  
**by** (*induct c auto*)

**theorem** *SC-siso[intro]: SC-siso c  $\implies$  siso c*  
**by** (*induct c auto*)

**theorem** *SC-siso-imp-SC-WbisT[intro]: SC-siso c  $\implies$  SC-WbisT c*  
**by** (*induct c auto*)

**theorem** *SC-discr-imp-SC-WbisT[intro]: noWhile c  $\implies$  SC-discr c  $\implies$  SC-WbisT c*  
**by** (*induct c (auto simp: presAtm-compatAtm)*)

**theorem** *SC-WbisT[intro]: SC-WbisT c  $\implies$  c  $\approx_{wT}$  c*  
**by** (*induct c (auto split: if-split-asm)*)

**theorem** *SC-discr-imp-SC-ZObis[intro]: SC-discr c  $\implies$  SC-ZObis c*  
**by** (*induct c (auto simp: presAtm-compatAtm)*)

**theorem** *SC-siso-imp-SC-ZObis[intro]: SC-siso c  $\implies$  SC-ZObis c*  
**by** (*induct c auto*)

**theorem** *SC-ZObis[intro]: SC-ZObis c  $\implies$  c  $\approx_{01}$  c*  
**by** (*induct c (auto split: if-split-asm intro: discr-ZObis)*)

**theorem** *SC-ZObis-imp-SC-Wbis[intro]: SC-ZObis c  $\implies$  SC-Wbis c*  
**by** (*induct c auto*)

**theorem** *SC-WbisT-imp-SC-Wbis[intro]: SC-WbisT c  $\implies$  SC-Wbis c*  
**by** (*induct c auto*)

**theorem** *SC-Wbis[intro]: SC-Wbis c  $\implies$  c  $\approx_w$  c*  
**by** (*induct c (auto split: if-split-asm intro: discr-ZObis)*)

**theorem** *SC-discr-imp-SC-BisT[intro]: SC-discr c  $\implies$  SC-BisT c*  
**by** (*induct c (auto simp: presAtm-compatAtm)*)

**theorem** *SC-WbisT-imp-SC-BisT[intro]: SC-WbisT c  $\implies$  SC-BisT c*  
**by** (*induct c auto*)

**theorem** *SC-ZObis-imp-SC-BisT[intro]: SC-ZObis c  $\implies$  SC-BisT c*  
**by** (*induct c auto*)

**theorem** *SC-Wbis-imp-SC-BisT[intro]: SC-Wbis c  $\implies$  SC-BisT c*  
**by** (*induct c (auto split: if-split-asm)*)

**theorem** *SC-BisT[intro]: SC-BisT c  $\implies$  c  $\approx_T$  c*

**by** (*induct c*) (*auto split: if-split-asm*)

**theorem** *SC-WbisT-While*:  $SC\text{-}WbisT\ (While\ tst\ c) \longleftrightarrow SC\text{-}WbisT\ c \wedge compatTst\ tst$

**by** *simp*

**theorem** *SC-ZObis-While*:  $SC\text{-}ZObis\ (While\ tst\ c) \longleftrightarrow (compatTst\ tst \wedge SC\text{-}siso\ c) \vee SC\text{-}discr\ c$

**by** *auto*

**theorem** *SC-ZObis-If*:  $SC\text{-}ZObis\ (If\ tst\ c1\ c2) \longleftrightarrow (if\ compatTst\ tst\ then\ SC\text{-}ZObis\ c1 \wedge SC\text{-}ZObis\ c2\ else\ SC\text{-}discr\ c1 \wedge SC\text{-}discr\ c2)$

**by** *simp*

**theorem** *SC-WbisT-Seq*:  $SC\text{-}WbisT\ (Seq\ c1\ c2) \longleftrightarrow (SC\text{-}WbisT\ c1 \wedge SC\text{-}WbisT\ c2)$

**by** *auto*

**theorem** *SC-ZObis-Seq*:  $SC\text{-}ZObis\ (Seq\ c1\ c2) \longleftrightarrow (SC\text{-}siso\ c1 \wedge SC\text{-}ZObis\ c2) \vee$

$(SC\text{-}ZObis\ c1 \wedge SC\text{-}discr\ c2)$

**by** *auto*

**theorem** *SC-Wbis-Seq*:  $SC\text{-}Wbis\ (Seq\ c1\ c2) \longleftrightarrow (SC\text{-}WbisT\ c1 \wedge SC\text{-}Wbis\ c2) \vee (SC\text{-}Wbis\ c1 \wedge SC\text{-}discr\ c2)$

**by** *auto*

**theorem** *SC-BisT-Par*:

$SC\text{-}BisT\ (Par\ c1\ c2) \longleftrightarrow (SC\text{-}BisT\ c1 \wedge SC\text{-}BisT\ c2)$

**by** *auto*

**end**

**end**

## 6 After-execution security

**theory** *After-Execution*

**imports** *During-Execution*

**begin**

**context** *PL-Indis*

**begin**

## 6.1 Setup for bisimilarities

**lemma** *Sbis-transC*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx s \ d$  **and**  $s \approx t$  **and**  $(c,s) \rightarrow c \ (c',s')$   
**obtains**  $d' \ t'$  **where**  
 $(d,t) \rightarrow c \ (d',t')$  **and**  $s' \approx t'$  **and**  $c' \approx s \ d'$   
**using** *assms Sbis-matchC-C*[OF 0]  
**unfolding** *matchC-C-def* **by** *blast*

**lemma** *Sbis-transT*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx s \ d$  **and**  $s \approx t$  **and**  $(c,s) \rightarrow t \ s'$   
**obtains**  $t'$  **where**  $(d,t) \rightarrow t \ t'$  **and**  $s' \approx t'$   
**using** *assms Sbis-matchT-T*[OF 0]  
**unfolding** *matchT-T-def* **by** *blast*

**lemma** *Sbis-transC2*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx s \ d$  **and**  $s \approx t$  **and**  $(d,t) \rightarrow c \ (d',t')$   
**obtains**  $c' \ s'$  **where**  
 $(c,s) \rightarrow c \ (c',s')$  **and**  $s' \approx t'$  **and**  $c' \approx s \ d'$   
**using** *assms Sbis-matchC-C-rev*[OF 0] *Sbis-Sym*  
**unfolding** *matchC-C-def2* **by** *blast*

**lemma** *Sbis-transT2*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx s \ d$  **and**  $s \approx t$  **and**  $(d,t) \rightarrow t \ t'$   
**obtains**  $s'$  **where**  $(c,s) \rightarrow t \ s'$  **and**  $s' \approx t'$   
**using** *assms Sbis-matchT-T-rev*[OF 0] *Sbis-Sym*  
**unfolding** *matchT-T-def2* **by** *blast*

**lemma** *ZObisT-transC*[consumes 3, case-names Match MatchS]:  
**assumes**  $0: c \approx 01T \ d$  **and**  $s \approx t$  **and**  $(c,s) \rightarrow c \ (c',s')$   
**and**  $\bigwedge d' \ t'. \llbracket (d,t) \rightarrow c \ (d',t'); \ s' \approx t'; \ c' \approx 01T \ d \rrbracket \implies \textit{thesis}$   
**and**  $\llbracket s' \approx t; \ c' \approx 01T \ d \rrbracket \implies \textit{thesis}$   
**shows** *thesis*  
**using** *assms ZObisT-matchC-ZOC*[OF 0]  
**unfolding** *matchC-ZOC-def* **by** *blast*

**lemma** *ZObisT-transT*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx 01T \ d$  **and**  $s \approx t$  **and**  $(c,s) \rightarrow t \ s'$   
**obtains**  $t'$  **where**  $(d,t) \rightarrow t \ t'$  **and**  $s' \approx t'$   
**using** *assms ZObisT-matchT-T*[OF 0]  
**unfolding** *matchT-T-def* **by** *blast*

**lemma** *ZObisT-transC2*[consumes 3, case-names Match MatchS]:  
**assumes**  $0: c \approx 01T \ d$  **and**  $2: s \approx t$  **and**  $3: (d,t) \rightarrow c \ (d',t')$   
**and**  $4: \bigwedge c' \ s'. \llbracket (c,s) \rightarrow c \ (c',s'); \ s' \approx t'; \ c' \approx 01T \ d \rrbracket \implies \textit{thesis}$   
**and**  $5: \llbracket s \approx t'; \ c \approx 01T \ d \rrbracket \implies \textit{thesis}$   
**shows** *thesis*  
**using** *assms ZObisT-matchC-ZOC-rev*[OF 0] *ZObisT-Sym*  
**unfolding** *matchC-ZOC-def2* **by** *blast*

**lemma** *ZObisT-transT2*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx_{01T} d$  **and**  $s \approx t$  **and**  $(d, t) \rightarrow t'$   
**obtains**  $s'$  **where**  $(c, s) \rightarrow t s'$  **and**  $s' \approx t'$   
**using** *assms ZObisT-matchT-T-rev*[OF 0] *ZObisT-Sym*  
**unfolding** *matchT-T-def2* **by** *blast*

**lemma** *WbisT-transC*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx_{wT} d$  **and**  $s \approx t$  **and**  $(c, s) \rightarrow c (c', s')$   
**obtains**  $d' t'$  **where**  
 $(d, t) \rightarrow^* c (d', t')$  **and**  $s' \approx t'$  **and**  $c' \approx_{wT} d'$   
**using** *assms WbisT-matchC-MC*[OF 0]  
**unfolding** *matchC-MC-def* **by** *blast*

**lemma** *WbisT-transT*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx_{wT} d$  **and**  $s \approx t$  **and**  $(c, s) \rightarrow t s'$   
**obtains**  $t'$  **where**  $(d, t) \rightarrow^* t t'$  **and**  $s' \approx t'$   
**using** *assms WbisT-matchT-MT*[OF 0]  
**unfolding** *matchT-MT-def* **by** *blast*

**lemma** *WbisT-transC2*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx_{wT} d$  **and**  $s \approx t$  **and**  $(d, t) \rightarrow c (d', t')$   
**obtains**  $c' s'$  **where**  
 $(c, s) \rightarrow^* c (c', s')$  **and**  $s' \approx t'$  **and**  $c' \approx_{wT} d'$   
**using** *assms WbisT-matchC-MC-rev*[OF 0] *WbisT-Sym*  
**unfolding** *matchC-MC-def2* **by** *blast*

**lemma** *WbisT-transT2*[consumes 3, case-names Match]:  
**assumes**  $0: c \approx_{wT} d$  **and**  $s \approx t$  **and**  $(d, t) \rightarrow t t'$   
**obtains**  $s'$  **where**  $(c, s) \rightarrow^* t s'$  **and**  $s' \approx t'$   
**using** *assms WbisT-matchT-MT-rev*[OF 0] *WbisT-Sym*  
**unfolding** *matchT-MT-def2* **by** *blast*

**lemma** *WbisT-MtransC*[consumes 3, case-names Match]:  
**assumes**  $1: c \approx_{wT} d$  **and**  $2: s \approx t$  **and**  $3: (c, s) \rightarrow^* c (c', s')$   
**obtains**  $d' t'$  **where**  
 $(d, t) \rightarrow^* c (d', t')$  **and**  $s' \approx t'$  **and**  $c' \approx_{wT} d'$   
**proof** –  
**have**  $(c, s) \rightarrow^* c (c', s') \implies$   
 $c \approx_{wT} d \implies s \approx t \implies$   
 $(\exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge c' \approx_{wT} d')$   
**proof** (*induct rule: MtransC-induct2*)  
**case** (*Trans c s c' s' c'' s''*)  
**then obtain**  $d' t'$  **where**  $d: (d, t) \rightarrow^* c (d', t')$   
**and**  $s' \approx t'$  **and**  $c' \approx_{wT} d'$   
**and**  $(c', s') \rightarrow c (c'', s'')$  **by** *auto*  
**then obtain**  $d'' t''$  **where**  $s'' \approx t''$  **and**  $c'' \approx_{wT} d''$

**and**  $(d', t') \rightarrow^* c (d'', t')$  **by** (*metis WbisT-transC*)  
**thus**  $?case$  **using**  $d$  **by** (*metis MtransC-Trans*)  
**qed** (*metis MtransC-Refl*)  
**thus** *thesis* **using** *that assms* **by** *auto*  
**qed**

**lemma** *WbisT-MtransT[consumes 3, case-names Match]*:

**assumes**  $1: c \approx_{wT} d$  **and**  $2: s \approx t$  **and**  $3: (c, s) \rightarrow^* t s'$   
**obtains**  $t'$  **where**  $(d, t) \rightarrow^* t t'$  **and**  $s' \approx t'$

**proof** –

**obtain**  $c'' s''$  **where**  $4: (c, s) \rightarrow^* c (c'', s'')$   
**and**  $5: (c'', s'') \rightarrow t s'$  **using**  $3$  **by** (*metis MtransT-invert2*)  
**then obtain**  $d'' t''$  **where**  $d: (d, t) \rightarrow^* c (d'', t'')$   
**and**  $s'' \approx t''$  **and**  $c'' \approx_{wT} d''$  **using**  $1\ 2\ 4$  *WbisT-MtransC* **by** *blast*  
**then obtain**  $t'$  **where**  $s' \approx t'$  **and**  $(d'', t'') \rightarrow^* t t'$   
**by** (*metis 5 WbisT-transT*)  
**thus** *thesis* **using**  $d$  **that by** (*metis MtransC-MtransT*)  
**qed**

**lemma** *WbisT-MtransC2[consumes 3, case-names Match]*:

**assumes**  $c \approx_{wT} d$  **and**  $s \approx t$  **and**  $1: (d, t) \rightarrow^* c (d', t')$

**obtains**  $c' s'$  **where**

$(c, s) \rightarrow^* c (c', s')$  **and**  $s' \approx t'$  **and**  $c' \approx_{wT} d'$

**proof** –

**have**  $d \approx_{wT} c$  **and**  $t \approx s$   
**using** *assms* **by** (*metis WbisT-Sym indis-sym*)  
**then obtain**  $c' s'$  **where**  
 $(c, s) \rightarrow^* c (c', s')$  **and**  $t' \approx s'$  **and**  $d' \approx_{wT} c'$   
**by** (*metis 1 WbisT-MtransC*)  
**thus**  $?thesis$  **using** *that by* (*metis WbisT-Sym indis-sym*)  
**qed**

**lemma** *WbisT-MtransT2[consumes 3, case-names Match]*:

**assumes**  $c \approx_{wT} d$  **and**  $s \approx t$  **and**  $(d, t) \rightarrow^* t t'$

**obtains**  $s'$  **where**  $(c, s) \rightarrow^* t s'$  **and**  $s' \approx t'$

**by** (*metis WbisT-MtransT WbisT-Sym assms indis-sym*)

**lemma** *ZObis-transC[consumes 3, case-names Match MatchO MatchS]*:

**assumes**  $0: c \approx_{01} d$  **and**  $s \approx t$  **and**  $(c, s) \rightarrow c (c', s')$

**and**  $\bigwedge d' t'. \llbracket (d, t) \rightarrow c (d', t'); s' \approx t'; c' \approx_{01} d' \rrbracket \implies thesis$

**and**  $\bigwedge t'. \llbracket (d, t) \rightarrow t t'; s' \approx t'; discr c \rrbracket \implies thesis$

**and**  $\llbracket s' \approx t; c' \approx_{01} d \rrbracket \implies thesis$

**shows** *thesis*

**using** *assms* *ZObis-matchC-ZO[OF 0]*

**unfolding** *matchC-ZO-def* **by** *blast*

**lemma** *ZObis-transT[consumes 3, case-names Match MatchO MatchS]*:

**assumes**  $0: c \approx_{01} d$  **and**  $s \approx t$  **and**  $(c, s) \rightarrow t s'$

**and**  $\bigwedge t'. \llbracket (d,t) \rightarrow t t'; s' \approx t' \rrbracket \Longrightarrow thesis$   
**and**  $\bigwedge d' t'. \llbracket (d,t) \rightarrow c (d',t'); s' \approx t'; discr d' \rrbracket \Longrightarrow thesis$   
**and**  $\llbracket s' \approx t; discr d \rrbracket \Longrightarrow thesis$   
**shows** *thesis*  
**using** *assms ZObis-matchT-ZO[OF 0]*  
**unfolding** *matchT-ZO-def* **by** *blast*

**lemma** *ZObis-transC2[consumes 3, case-names Match MatchO MatchS]:*  
**assumes**  $0: c \approx 01 d$  **and**  $s \approx t$  **and**  $(d,t) \rightarrow c (d',t')$   
**and**  $\bigwedge c' s'. \llbracket (c,s) \rightarrow c (c',s'); s' \approx t'; c' \approx 01 d' \rrbracket \Longrightarrow thesis$   
**and**  $\bigwedge s'. \llbracket (c,s) \rightarrow t s'; s' \approx t'; discr d' \rrbracket \Longrightarrow thesis$   
**and**  $\llbracket s \approx t'; c \approx 01 d' \rrbracket \Longrightarrow thesis$   
**shows** *thesis*  
**using** *assms ZObis-matchC-ZO-rev[OF 0] ZObis-Sym*  
**unfolding** *matchC-ZO-def2* **by** *blast*

**lemma** *ZObis-transT2[consumes 3, case-names Match MatchO MatchS]:*  
**assumes**  $0: c \approx 01 d$  **and**  $s \approx t$  **and**  $(d,t) \rightarrow t t'$   
**and**  $\bigwedge s'. \llbracket (c,s) \rightarrow t s'; s' \approx t' \rrbracket \Longrightarrow thesis$   
**and**  $\bigwedge c' s'. \llbracket (c,s) \rightarrow c (c',s'); s' \approx t'; discr c' \rrbracket \Longrightarrow thesis$   
**and**  $\llbracket s \approx t'; discr c \rrbracket \Longrightarrow thesis$   
**shows** *thesis*  
**using** *assms ZObis-matchT-ZO-rev[OF 0] ZObis-Sym*  
**unfolding** *matchT-ZO-def2* **by** *blast*

**lemma** *Wbis-transC[consumes 3, case-names Match MatchO]:*  
**assumes**  $0: c \approx w d$  **and**  $s \approx t$  **and**  $(c,s) \rightarrow c (c',s')$   
**and**  $\bigwedge d' t'. \llbracket (d,t) \rightarrow *c (d',t'); s' \approx t'; c' \approx w d' \rrbracket \Longrightarrow thesis$   
**and**  $\bigwedge t'. \llbracket (d,t) \rightarrow *t t'; s' \approx t'; discr c' \rrbracket \Longrightarrow thesis$   
**shows** *thesis*  
**using** *assms Wbis-matchC-M[OF 0]*  
**unfolding** *matchC-M-def* **by** *blast*

**lemma** *Wbis-transT[consumes 3, case-names Match MatchO]:*  
**assumes**  $0: c \approx w d$  **and**  $s \approx t$  **and**  $(c,s) \rightarrow t s'$   
**and**  $\bigwedge t'. \llbracket (d,t) \rightarrow *t t'; s' \approx t' \rrbracket \Longrightarrow thesis$   
**and**  $\bigwedge d' t'. \llbracket (d,t) \rightarrow *c (d',t'); s' \approx t'; discr d' \rrbracket \Longrightarrow thesis$   
**shows** *thesis*  
**using** *assms Wbis-matchT-M[OF 0]*  
**unfolding** *matchT-M-def* **by** *blast*

**lemma** *Wbis-transC2[consumes 3, case-names Match MatchO]:*  
**assumes**  $0: c \approx w d$  **and**  $s \approx t$  **and**  $(d,t) \rightarrow c (d',t')$   
**and**  $\bigwedge c' s'. \llbracket (c,s) \rightarrow *c (c',s'); s' \approx t'; c' \approx w d' \rrbracket \Longrightarrow thesis$   
**and**  $\bigwedge s'. \llbracket (c,s) \rightarrow *t s'; s' \approx t'; discr d' \rrbracket \Longrightarrow thesis$   
**shows** *thesis*  
**using** *assms Wbis-matchC-M-rev[OF 0] Wbis-Sym*  
**unfolding** *matchC-M-def2* **by** *blast*

**lemma** *Wbis-transT2*[consumes 3, case-names Match MatchO]:  
**assumes**  $0: c \approx_w d$  **and**  $s \approx t$  **and**  $(d, t) \rightarrow t'$   
**and**  $\bigwedge s'. \llbracket (c, s) \rightarrow^* t s'; s' \approx t' \rrbracket \implies \text{thesis}$   
**and**  $\bigwedge c' s'. \llbracket (c, s) \rightarrow^* c (c', s'); s' \approx t'; \text{discr } c \rrbracket \implies \text{thesis}$   
**shows** *thesis*  
**using** *assms Wbis-matchT-M-rev*[OF 0] *Wbis-Sym*  
**unfolding** *matchT-M-def2* **by** *blast*

**lemma** *Wbis-MtransC*[consumes 3, case-names Match MatchO]:  
**assumes**  $c \approx_w d$  **and**  $s \approx t$  **and**  $(c, s) \rightarrow^* c (c', s')$   
**and**  $\bigwedge d' t'. \llbracket (d, t) \rightarrow^* c (d', t'); s' \approx t'; c' \approx_w d' \rrbracket \implies \text{thesis}$   
**and**  $\bigwedge t'. \llbracket (d, t) \rightarrow^* t t'; s' \approx t'; \text{discr } c \rrbracket \implies \text{thesis}$   
**shows** *thesis*

**proof** –

**have**  $(c, s) \rightarrow^* c (c', s') \implies$   
 $c \approx_w d \implies s \approx t \implies$   
 $(\exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge c' \approx_w d') \vee$   
 $(\exists t'. (d, t) \rightarrow^* t t' \wedge s' \approx t' \wedge \text{discr } c')$

**proof** (*induct rule: MtransC-induct2*)

**case** (*Trans*  $c$   $s$   $c'$   $s'$   $c''$   $s''$ )

**hence**  $c's': (c', s') \rightarrow c (c'', s'')$

**and**

$(\exists d' t'. (d, t) \rightarrow^* c (d', t') \wedge s' \approx t' \wedge c' \approx_w d') \vee$   
 $(\exists t'. (d, t) \rightarrow^* t t' \wedge s' \approx t' \wedge \text{discr } c')$  **by** *auto*

**thus** *?case* (**is**  $?A \vee ?B$ )

**proof** (*elim disjE exE conjE*)

**fix**  $d' t'$

**assume**  $c'd': c' \approx_w d'$  **and**  $s't': s' \approx t'$

**and**  $dt: (d, t) \rightarrow^* c (d', t')$

**from**  $c'd' s't' c's'$  **show** *?case*

**apply** (*cases rule: Wbis-transC*)

**by** (*metis dt MtransC-Trans MtransC-MtransT*)**+**

**next**

**fix**  $t'$

**assume**  $s't': s' \approx t'$  **and**  $c': \text{discr } c'$

**and**  $dt: (d, t) \rightarrow^* t t'$

**from**  $c' s't' c's'$  **show** *?case*

**by** (*metis discr.simps dt indis-sym indis-trans*)

**qed**

**qed** *auto*

**thus** *thesis* **using** *assms* **by** *auto*

**qed**

**lemma** *Wbis-MtransT*[consumes 3, case-names Match MatchO]:  
**assumes**  $c-d: c \approx_w d$  **and**  $st: s \approx t$  **and**  $cs: (c, s) \rightarrow^* t s'$   
**and**  $1: \bigwedge t'. \llbracket (d, t) \rightarrow^* t t'; s' \approx t' \rrbracket \implies \text{thesis}$   
**and**  $2: \bigwedge d' t'. \llbracket (d, t) \rightarrow^* c (d', t'); s' \approx t'; \text{discr } d' \rrbracket \implies \text{thesis}$



**shows** *thesis*  
**using** *cs* **proof**(*elim MtransT-invert2*)  
**fix** *c'' s''* **assume** *cs*:  $(c, s) \rightarrow^* c (c'', s'')$   
**and** *c''s''*:  $(c'', s'') \rightarrow t s'$   
**from** *c-d st cs* **show** *thesis*  
**proof** (*cases rule: Wbis-MtransC*)  
**fix** *d'' t''*  
**assume** *dt*:  $(d, t) \rightarrow^* c (d'', t'')$   
**and** *s''t''*:  $s'' \approx t''$  **and** *c''d''*:  $c'' \approx_w d''$   
**from** *c''d'' s''t'' c''s''* **show** *thesis*  
**apply** (*cases rule: Wbis-transT*)  
**by** (*metis 1 2 dt MtransC-MtransT MtransC-Trans*)+  
**next**  
**case** (*MatchO t'*)  
**thus** *?thesis* **using** *1 c''s''*  
**by** (*metis discr-MtransT indis-sym indis-trans transT-MtransT*)  
**qed**  
**qed**

**lemma** *Wbis-MtransC2*[*consumes 3, case-names Match MatchO*]:  
**assumes**  $c \approx_w d$  **and**  $s \approx t$  **and** *dt*:  $(d, t) \rightarrow^* c (d', t')$   
**and** *1*:  $\bigwedge c' s'. \llbracket (c, s) \rightarrow^* c (c', s'); s' \approx t'; c' \approx_w d' \rrbracket \implies \textit{thesis}$   
**and** *2*:  $\bigwedge s'. \llbracket (c, s) \rightarrow^* t s'; s' \approx t'; \textit{discr } d' \rrbracket \implies \textit{thesis}$   
**shows** *thesis*  
**proof**–  
**have** *dc*:  $d \approx_w c$  **and** *ts*:  $t \approx s$   
**by** (*metis assms Wbis-Sym indis-sym*)+  
**from** *dc ts dt* **show** *thesis*  
**apply**(*cases rule: Wbis-MtransC*)  
**by** (*metis 1 2 Wbis-Sym indis-sym*)+  
**qed**

**lemma** *Wbis-MtransT2*[*consumes 3, case-names Match MatchO*]:  
**assumes**  $c \approx_w d$  **and**  $s \approx t$  **and** *dt*:  $(d, t) \rightarrow^* t t'$   
**and** *1*:  $\bigwedge s'. \llbracket (c, s) \rightarrow^* t s'; s' \approx t' \rrbracket \implies \textit{thesis}$   
**and** *2*:  $\bigwedge c' s'. \llbracket (c, s) \rightarrow^* c (c', s'); s' \approx t'; \textit{discr } c' \rrbracket \implies \textit{thesis}$   
**shows** *thesis*  
**proof**–  
**have** *dc*:  $d \approx_w c$  **and** *ts*:  $t \approx s$   
**by** (*metis assms Wbis-Sym indis-sym*)+  
**from** *dc ts dt* **show** *thesis*  
**apply**(*cases rule: Wbis-MtransT*)  
**by** (*metis 1 2 Wbis-Sym indis-sym*)+  
**qed**

**lemma** *BisT-transC*[*consumes 5, case-names Match*]:  
**assumes** *0*:  $c \approx_T d$   
**and** *mustT c s* **and** *mustT d t*

**and**  $s \approx t$  **and**  $(c,s) \rightarrow c (c',s')$   
**obtains**  $d' t'$  **where**  
 $(d,t) \rightarrow *c (d',t')$  **and**  $s' \approx t'$  **and**  $c' \approx T d'$   
**using** *assms* *BisT-matchC-TMC*[OF 0]  
**unfolding** *matchC-TMC-def* **by** *blast*

**lemma** *BisT-transT*[*consumes* 5, *case-names* *Match*]:  
**assumes**  $0: c \approx T d$   
**and** *mustT*  $c s$  **and** *mustT*  $d t$   
**and**  $s \approx t$  **and**  $(c,s) \rightarrow t s'$   
**obtains**  $t'$  **where**  $(d,t) \rightarrow *t t'$  **and**  $s' \approx t'$   
**using** *assms* *BisT-matchT-TMT*[OF 0]  
**unfolding** *matchT-TMT-def* **by** *blast*

**lemma** *BisT-transC2*[*consumes* 5, *case-names* *Match*]:  
**assumes**  $0: c \approx T d$   
**and** *mustT*  $c s$  **and** *mustT*  $d t$   
**and**  $s \approx t$  **and**  $(d,t) \rightarrow c (d',t')$   
**obtains**  $c' s'$  **where**  
 $(c,s) \rightarrow *c (c',s')$  **and**  $s' \approx t'$  **and**  $c' \approx T d'$   
**using** *assms* *BisT-matchC-TMC-rev*[OF 0] *BisT-Sym*  
**unfolding** *matchC-TMC-def2* **by** *blast*

**lemma** *BisT-transT2*[*consumes* 5, *case-names* *Match*]:  
**assumes**  $0: c \approx T d$   
**and** *mustT*  $c s$  **and** *mustT*  $d t$   
**and**  $s \approx t$  **and**  $(d,t) \rightarrow t t'$   
**obtains**  $s'$  **where**  $(c,s) \rightarrow *t s'$  **and**  $s' \approx t'$   
**using** *assms* *BisT-matchT-TMT-rev*[OF 0] *BisT-Sym*  
**unfolding** *matchT-TMT-def2* **by** *blast*

**lemma** *BisT-MtransC*[*consumes* 5, *case-names* *Match*]:  
**assumes**  $c \approx T d$   
**and** *mustT*  $c s$  *mustT*  $d t$   
**and**  $s \approx t$  **and**  $(c,s) \rightarrow *c (c',s')$   
**obtains**  $d' t'$  **where**  
 $(d,t) \rightarrow *c (d',t')$  **and**  $s' \approx t'$  **and**  $c' \approx T d'$   
**proof** –  
**have**  $(c,s) \rightarrow *c (c',s') \implies$   
 $mustT c s \implies mustT d t \implies$   
 $c \approx T d \implies s \approx t \implies$   
 $(\exists d' t'. (d,t) \rightarrow *c (d',t') \wedge s' \approx t' \wedge c' \approx T d')$   
**proof** (*induct rule: MtransC-induct2*)  
**case** (*Trans*  $c s c' s' c'' s''$ )  
**then obtain**  $d' t'$  **where**  $d: (d, t) \rightarrow *c (d', t')$   
**and**  $s' \approx t'$  **and**  $c' \approx T d'$   
**and**  $c's': (c', s') \rightarrow c (c'', s'')$  **by** *auto*  
**moreover have** *mustT*  $c' s'$  *mustT*  $d' t'$

by (*metis Trans mustT-MtransC d*)+  
 ultimately obtain  $d'' t''$  where  $s'' \approx t''$  and  $c'' \approx T d''$   
 and  $(d', t') \rightarrow^* c (d'', t')$  by (*metis BisT-transC*)  
 thus ?*case using d* by (*metis MtransC-Trans*)  
 qed (*metis MtransC-Refl*)  
 thus *thesis using that assms by auto*  
 qed

**lemma** *BisT-MtransT*[*consumes 5, case-names Match*]:  
 assumes 1:  $c \approx T d$   
 and *ter*:  $mustT c s mustT d t$   
 and 2:  $s \approx t$  and 3:  $(c, s) \rightarrow^* t s'$   
 obtains  $t'$  where  $(d, t) \rightarrow^* t' t'$  and  $s' \approx t'$   
 proof –  
 obtain  $c'' s''$  where 4:  $(c, s) \rightarrow^* c (c'', s'')$   
 and 5:  $(c'', s'') \rightarrow t s'$  using 3 by (*metis MtransT-invert2*)  
 then obtain  $d'' t''$  where  $d: (d, t) \rightarrow^* c (d'', t'')$   
 and  $s'' \approx t''$  and  $c'' \approx T d''$  using 1 2 *ter* 4 *BisT-MtransC* by *blast*  
 moreover have  $mustT c'' s'' mustT d'' t''$   
 by (*metis d 4 assms mustT-MtransC*)+  
 ultimately obtain  $t'$  where  $s' \approx t'$  and  $(d'', t'') \rightarrow^* t t'$   
 by (*metis 5 ter BisT-transT*)  
 thus *thesis using d that by (metis MtransC-MtransT)*  
 qed

**lemma** *BisT-MtransC2*[*consumes 3, case-names Match*]:  
 assumes  $c \approx T d$   
 and *ter*:  $mustT c s mustT d t$   
 and  $s \approx t$  and 1:  $(d, t) \rightarrow^* c (d', t')$   
 obtains  $c' s'$  where  
 $(c, s) \rightarrow^* c (c', s')$  and  $s' \approx t'$  and  $c' \approx T d'$   
 proof –  
 have  $d \approx T c$  and  $t \approx s$   
 using *assms* by (*metis BisT-Sym indis-sym*)+  
 then obtain  $c' s'$  where  
 $(c, s) \rightarrow^* c (c', s')$  and  $t' \approx s'$  and  $d' \approx T c'$   
 by (*metis 1 ter BisT-MtransC*)  
 thus ?*thesis using that by (metis BisT-Sym indis-sym)*  
 qed

**lemma** *BisT-MtransT02*[*consumes 3, case-names Match*]:  
 assumes  $c \approx T d$   
 and *ter*:  $mustT c s mustT d t$   
 and  $s \approx t$  and  $(d, t) \rightarrow^* t t'$   
 obtains  $s'$  where  $(c, s) \rightarrow^* t s'$  and  $s' \approx t'$   
 by (*metis BisT-MtransT BisT-Sym assms indis-sym*)

## 6.2 Execution traces

**primrec** *parTrace* **where**

*parTrace* []  $\longleftrightarrow$  *False* |

*parTrace* (cf#cfl)  $\longleftrightarrow$  (cfl  $\neq$  []  $\longrightarrow$  *parTrace* cfl  $\wedge$  cf  $\rightarrow$ c hd cfl)

**lemma** *trans-Step2*:

cf  $\rightarrow^*c$  cf'  $\implies$  cf'  $\rightarrow$ c cf''  $\implies$  cf  $\rightarrow^*c$  cf''

**using** *trans-Step*[of fst cf snd cf fst cf' snd cf' fst cf'' snd cf'']

**by** *simp*

**lemma** *parTrace-not-empty*[*simp*]: *parTrace* cfl  $\implies$  cfl  $\neq$  []

**by** (*cases* cfl = []) *simp*

**lemma** *parTrace-snoc*[*simp*]:

*parTrace* (cfl@[cf])  $\longleftrightarrow$  (cfl  $\neq$  []  $\longrightarrow$  *parTrace* cfl  $\wedge$  last cfl  $\rightarrow$ c cf)

**by** (*induct* cfl) *auto*

**lemma** *MtransC-Ex-parTrace*:

**assumes** cf  $\rightarrow^*c$  cf' **shows**  $\exists$  cfl. *parTrace* cfl  $\wedge$  hd cfl = cf  $\wedge$  last cfl = cf'

**using** *assms*

**proof** (*induct* rule: *MtransC-induct*)

**case** (*Refl* cf) **then show** ?*case*

**by** (*auto* *intro!*: *exI*[of - [cf]])

**next**

**case** (*Trans* cf cf' cf'')

**then obtain** cfl **where** *parTrace* cfl hd cfl = cf last cfl = cf' **by** *auto*

**with**  $\langle$ cf'  $\rightarrow$ c cf'' $\rangle$  **show** ?*case*

**by** (*auto* *intro!*: *exI*[of - cfl @ [cf'']])

**qed**

**lemma** *parTrace-imp-MtransC*:

**assumes** *pT*: *parTrace* cfl

**shows** (hd cfl)  $\rightarrow^*c$  (last cfl)

**using** *pT* **proof** (*induct* cfl rule: *rev-induct*)

**case** (*snoc* cf cfl)

**with** *trans-Step2*[of hd cfl last cfl cf]

**show** ?*case*

**by** *auto*

**qed** *simp*

**fun** *finTrace* **where**

*finTrace* (cfl,s)  $\longleftrightarrow$

*parTrace* cfl  $\wedge$  last cfl  $\rightarrow$ t s

**declare** *finTrace.simps*[*simp* *del*]

**definition** *lengthFT* tr  $\equiv$  *Suc* (*length* (fst tr))

**definition**  $fstate\ tr \equiv snd\ tr$

**definition**  $iconfig\ tr \equiv hd\ (fst\ tr)$

**lemma**  $MtransT\text{-}Ex\text{-}finTrace$ :

**assumes**  $cf \rightarrow^* t\ s$  **shows**  $\exists tr. finTrace\ tr \wedge iconfig\ tr = cf \wedge fstate\ tr = s$

**proof** –

**from**  $\langle cf \rightarrow^* t\ s \rangle$  **obtain**  $cf'\ cfl$  **where**  $parTrace\ cfl\ hd\ cfl = cf\ last\ cfl \rightarrow^* t\ s$

**by**  $(auto\ simp: MtransT.simps\ dest!: MtransC\text{-}Ex\text{-}parTrace)$

**then show**  $?thesis$

**by**  $(auto\ simp: finTrace.simps\ iconfig\text{-}def\ fstate\text{-}def$

$intro!: exI[of\ -\ cfl]\ exI[of\ -\ s])$

**qed**

**lemma**  $finTrace\text{-}imp\text{-}MtransT$ :

$finTrace\ tr \implies iconfig\ tr \rightarrow^* t\ fstate\ tr$

**using**  $parTrace\text{-}imp\text{-}MtransC[of\ fst\ tr]$

**by**  $(cases\ tr)$

$(auto\ simp\ add: iconfig\text{-}def\ fstate\text{-}def\ finTrace.simps\ MtransT.simps$

$simp\ del: split\text{-}paired\text{-}Ex)$

### 6.3 Relationship between during-execution and after-execution security

**lemma**  $WbisT\text{-}trace2$ :

**assumes**  $bis: c \approx_w T\ d\ s \approx t$

**and**  $tr: finTrace\ tr\ iconfig\ tr = (c, s)$

**shows**  $\exists tr'. finTrace\ tr' \wedge iconfig\ tr' = (d, t) \wedge fstate\ tr \approx fstate\ tr'$

**proof** –

**from**  $tr\ finTrace\text{-}imp\text{-}MtransT[of\ tr]$

**have**  $(c, s) \rightarrow^* t\ fstate\ tr$

**by**  $auto$

**from**  $WbisT\text{-}MtransT[OF\ bis\ this]$

**obtain**  $t'$  **where**  $(d, t) \rightarrow^* t'\ fstate\ tr \approx t'$

**by**  $auto$

**from**  $MtransT\text{-}Ex\text{-}finTrace[OF\ this(1)]\ this(2)$

**show**  $?thesis$  **by**  $auto$

**qed**

**theorem**  $WbisT\text{-}trace$ :

**assumes**  $c \approx_w T\ c$  **and**  $s \approx t$

**and**  $finTrace\ tr$  **and**  $iconfig\ tr = (c, s)$

**shows**  $\exists tr'. finTrace\ tr' \wedge iconfig\ tr' = (c, t) \wedge fstate\ tr \approx fstate\ tr'$

**using**  $WbisT\text{-}trace2[OF\ assms]$  .

**theorem**  $ZObisT\text{-}trace2$ :

**assumes**  $bis: c \approx 01T d s \approx t$   
**and**  $tr: finTrace\ tr\ iconfig\ tr = (c,s)$   
**shows**  $\exists tr'. finTrace\ tr' \wedge iconfig\ tr' = (d,t) \wedge$   
 $fstate\ tr \approx fstate\ tr' \wedge lengthFT\ tr' \leq lengthFT\ tr$   
**proof** –  
**obtain**  $s' cfl$  **where**  $tr\text{-}eq: tr = (cfl, s')$  **by**  $(cases\ tr)\ auto$   
**with**  $tr$  **have**  $cfl: cfl \neq []\ parTrace\ cfl\ last\ cfl \rightarrow t\ s'\ hd\ cfl = (c,s)$   
**by**  $(auto\ simp\ add: finTrace.simps\ iconfig\text{-}def)$   
**from**  $this\ bis$   
**show**  $?thesis\ unfolding\ tr\text{-}eq\ fstate\text{-}def\ snd\text{-}conv$   
**proof**  $(induct\ cfl\ arbitrary: c\ d\ s\ t\ rule: list\text{-}nonempty\text{-}induct)$   
**case**  $(single\ cf)$   
**with**  $ZObisT\text{-}transT[of\ c\ d\ s\ t\ s']$   
**obtain**  $t'$  **where**  $(d,t) \rightarrow t\ t'\ s' \approx t'\ cf = (c,s)$   
**by**  $auto$   
**then show**  $?case$   
**by**  $(intro\ exI[of\ -\ ((d,t), t')])$   
 $(simp\ add: finTrace.simps\ parTrace\text{-}def\ iconfig\text{-}def\ fstate\text{-}def\ lengthFT\text{-}def)$   
**next**  
**case**  $(cons\ cf\ cfl)$   
**then have**  $cfl: parTrace\ cfl\ last\ cfl \rightarrow t\ s'$   
**by**  $auto$   
  
**from**  $cons$  **have**  $(c,s) \rightarrow c\ (fst\ (hd\ cfl), snd\ (hd\ cfl))$   
**unfolding**  $parTrace\text{-}def$  **by**  $(auto\ simp\ add: hd\text{-}conv\text{-}nth)$   
**with**  $\langle c \approx 01T d \rangle \langle s \approx t \rangle$  **show**  $?case$   
**proof**  $(cases\ rule: ZObisT\text{-}transC)$   
**case**  $MatchS$   
**from**  $cons(2)[OF\ cfl - this(2,1)]$   
**show**  $?thesis$   
**by**  $(auto\ simp: lengthFT\text{-}def\ le\text{-}Suc\text{-}eq)$   
**next**  
**case**  $(Match\ d'\ t')$   
**from**  $cons(2)[OF\ cfl - Match(3,2)]$   
**obtain**  $cfl'\ s$  **where**  $finTrace\ (cfl', s)\ hd\ cfl' = (d', t')\ s' \approx s\ length\ cfl' \leq$   
 $length\ cfl$   
**by**  $(auto\ simp: iconfig\text{-}def\ lengthFT\text{-}def)$   
**with**  $Match(1)$  **show**  $?thesis$   
**by**  $(intro\ exI[of\ -\ ((d,t)\#cfl', s)])$   
 $(auto\ simp: iconfig\text{-}def\ lengthFT\text{-}def\ finTrace.simps)$   
**qed**  
**qed**  
**qed**

**theorem**  $ZObisT\text{-}trace:$

**assumes**  $c \approx 01T c s \approx t$   
**and**  $finTrace\ tr\ iconfig\ tr = (c,s)$   
**shows**  $\exists tr'. finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge$   
 $fstate\ tr \approx fstate\ tr' \wedge lengthFT\ tr' \leq lengthFT\ tr$

using *ZObisT-trace2*[*OF assms*] .

**theorem** *Sbis-trace*:

assumes *bis*:  $c \approx s \ d \ s \approx t$

and *tr*: *finTrace* *tr* *iconfig* *tr* = (*c*,*s*)

shows  $\exists tr'. \text{finTrace } tr' \wedge \text{iconfig } tr' = (d,t) \wedge \text{fstate } tr \approx \text{fstate } tr' \wedge$   
 $\text{lengthFT } tr' = \text{lengthFT } tr$

**proof** –

**obtain** *s'* *cfl* **where** *tr-eq*: *tr* = (*cfl*, *s'*) **by** (*cases tr*) *auto*

**with** *tr* **have** *cfl*: *cfl*  $\neq []$  *parTrace* *cfl* *last cfl*  $\rightarrow t \ s' \ \text{hd } cfl = (c,s)$

**by** (*auto simp add: finTrace.simps iconfig-def*)

**from** *this bis*

**show** *?thesis unfolding tr-eq fstate-def snd-conv*

**proof** (*induct cfl arbitrary: c d s t rule: list-nonempty-induct*)

**case** (*single cf*)

**with** *Sbis-transT*[*of c d s t s'*]

**obtain** *t'* **where** (*d,t*)  $\rightarrow t \ t' \ s' \approx t' \ cf = (c,s)$

**by** *auto*

**with** *single show ?case*

**by** (*intro exI[of - ((d,t), t')]*)

(*simp add: lengthFT-def iconfig-def indis-refl finTrace.simps*)

**next**

**case** (*cons cf cfl*)

**with** *Sbis-transC*[*of c d s t fst (hd cfl) snd (hd cfl)*]

**obtain** *d'* *t'* **where** \*: (*d,t*)  $\rightarrow c \ (d',t') \ \text{snd} \ (\text{hd } cfl) \approx t' \ \text{fst} \ (\text{hd } cfl) \approx s \ d'$

**by** *auto*

**moreover**

**with** *cons(2)*[*of fst (hd cfl) snd (hd cfl) d' t'*] *cons(1,3,4)*

**obtain** *cfl'* *s* **where** *finTrace* (*cfl'*, *s*) *hd cfl'* = (*d'*, *t'*) *s'*  $\approx s \ \text{length } cfl' =$

*length cfl*

**by** (*auto simp: iconfig-def lengthFT-def*)

**ultimately show** *?case*

**by** (*intro exI[of - ((d,t)#cfl', s)]*)

(*auto simp: finTrace.simps lengthFT-def iconfig-def*)

**qed**

**qed**

**corollary** *siso-trace*:

assumes *siso* *c* **and**  $s \approx t$

**and** *finTrace* *tr* **and** *iconfig* *tr* = (*c*,*s*)

**shows**

$\exists tr'. \text{finTrace } tr' \wedge \text{iconfig } tr' = (c,t) \wedge \text{fstate } tr \approx \text{fstate } tr'$   
 $\wedge \text{lengthFT } tr' = \text{lengthFT } tr$

**apply**(*rule Sbis-trace*)

**using** *assms* **by** *auto*

**theorem** *Wbis-trace*:

assumes *T*:  $\bigwedge s. \text{mustT } c \ s$

**and**  $bis: c \approx_w c \ s \approx t$   
**and**  $tr: \text{finTrace } tr \ \text{iconfig } tr = (c, s)$   
**shows**  $\exists tr'. \text{finTrace } tr' \wedge \text{iconfig } tr' = (c, t) \wedge \text{fstate } tr \approx \text{fstate } tr'$   
**proof** –  
**from**  $tr \ \text{finTrace-imp-MtransT}[of \ tr]$   
**have**  $(c, s) \rightarrow^* t \ \text{fstate } tr$   
**by** *auto*  
**from**  $bis \ \text{this}$   
**show** *?thesis*  
**proof** (*cases rule: Wbis-MtransT*)  
**case** (*Match t'*)  
**from**  $MtransT\text{-Ex-finTrace}[OF \ \text{this}(1)] \ \text{this}(2)$   
**show** *?thesis* **by** *auto*  
**next**  
**case** (*MatchO d' t'*)  
**from**  $T[THEN \ \text{mustT-MtransC}, \ OF \ \text{MatchO}(1)] \ \text{have} \ \text{mustT } d' \ t'$   
**from**  $\text{this}[THEN \ \text{mustT-MtransT}] \ \text{obtain } s' \ \text{where } (d', t') \rightarrow^* t \ s' \ ..$   
**from**  $\text{MatchO}(1) \ \langle d', t' \rangle \rightarrow^* t \ s' \ \rangle \ \text{have } (c, t) \rightarrow^* t \ s' \ \text{by } (\text{rule } MtransC\text{-MtransT})$   
**note**  $MtransT\text{-Ex-finTrace}[OF \ \text{this}]$   
**moreover**  
**from**  $\langle \text{discr } d' \rangle \ \langle d', t' \rangle \rightarrow^* t \ s' \ \rangle \ \text{have } t' \approx s' \ \text{by } (\text{rule } \text{discr-MtransT})$   
**with**  $\langle \text{fstate } tr \approx t' \rangle \ \text{have } \text{fstate } tr \approx s' \ \text{by } (\text{rule } \text{indis-trans})$   
**ultimately show** *?thesis*  
**by** *auto*  
**qed**  
**qed**

**corollary** *ZObis-trace:*

**assumes**  $T: \bigwedge s. \text{mustT } c \ s$   
**and**  $ZObis: c \approx_{01} c \ \text{and} \ \text{indis}: s \approx t$   
**and**  $tr: \text{finTrace } tr \ \text{iconfig } tr = (c, s)$   
**shows**  $\exists tr'. \text{finTrace } tr' \wedge \text{iconfig } tr' = (c, t) \wedge \text{fstate } tr \approx \text{fstate } tr'$   
**by** (*rule Wbis-trace[OF T bis-imp(4)][OF ZObis] indis tr*)

**theorem** *BisT-trace:*

**assumes**  $bis: c \approx_T c \ s \approx t$   
**and**  $T: \text{mustT } c \ s \ \text{mustT } c \ t$   
**and**  $tr: \text{finTrace } tr \ \text{iconfig } tr = (c, s)$   
**shows**  $\exists tr'. \text{finTrace } tr' \wedge \text{iconfig } tr' = (c, t) \wedge \text{fstate } tr \approx \text{fstate } tr'$   
**proof** –  
**from**  $tr \ \text{finTrace-imp-MtransT}[of \ tr]$   
**have**  $(c, s) \rightarrow^* t \ \text{fstate } tr$   
**by** *auto*  
**from**  $BisT\text{-MtransT}[OF \ \text{bis}(1) \ T \ \text{bis}(2) \ \text{this}]$   
**obtain**  $t' \ \text{where } (c, t) \rightarrow^* t \ t' \ \text{fstate } tr \approx t'$   
**from**  $MtransT\text{-Ex-finTrace}[OF \ \text{this}(1)] \ \text{this}(2)$   
**show** *?thesis*



by *auto*  
**qed**  
**end**

**end**

## 7 Concrete setting

**theory** *Concrete*  
**imports** *Syntactic-Criteria After-Execution*  
**begin**

**lemma** (in *PL-Indis*) *WbisT-If-cross*:

**assumes**  $c1 \approx_{wT} c2$   $c1 \approx_{wT} c1$   $c2 \approx_{wT} c2$

**shows**  $(\text{If } \text{tst } c1 \ c2) \approx_{wT} (\text{If } \text{tst } c1 \ c2)$

**proof** –

**define**  $\varphi$

**where**  $\varphi \ c \ d \longleftrightarrow (\exists c1' \ c2'. \ c = \text{If } \text{tst } c1' \ c2' \wedge d = \text{If } \text{tst } c1' \ c2' \wedge c1' \approx_{wT} c2' \wedge c1' \approx_{wT} c1' \wedge c2' \approx_{wT} c2')$

**for**  $c \ d$

**with** *assms* **have**  $\varphi$   $(\text{If } \text{tst } c1 \ c2)$   $(\text{If } \text{tst } c1 \ c2)$  **by** *auto*

**then show** *?thesis*

**proof** (*induct rule: WbisT-coinduct[where  $\varphi=\varphi$ ]*)

**case**  $(\text{cont } c \ s \ d \ t \ c' \ s')$

**note**  $\text{cont}(2,3)$

**moreover from** *cont* **obtain**  $c1 \ c2$

**where**  $\varphi: c = \text{If } \text{tst } c1 \ c2 \ d = \text{If } \text{tst } c1 \ c2 \ c1 \approx_{wT} c2 \ c1 \approx_{wT} c1 \ c2 \approx_{wT} c2$

**by** (*auto simp:  $\varphi$ -def*)

**moreover then have**  $c2 \approx_{wT} c1$

**using** *WbisT-sym unfolding sym-def* **by** *blast*

**ultimately have**  $(d, t) \rightarrow^* c \ (\text{if } \text{tval } \text{tst } t \ \text{then } c1 \ \text{else } c2, t) \wedge s' \approx t \wedge$

$(\varphi \ c' \ (\text{if } \text{tval } \text{tst } t \ \text{then } c1 \ \text{else } c2) \vee c' \approx_{wT} (\text{if } \text{tval } \text{tst } t \ \text{then } c1 \ \text{else } c2))$

**by** (*auto simp:  $\varphi$ -def*)

**then show** *?case* **by** *auto*

**qed** (*auto simp:  $\varphi$ -def*)

**qed**

We instantiate the following notions, kept generic so far:

- On the language syntax:
  - atoms, tests and states just like at the possibilistic case;
  - choices, to either if-choices (based on tests) or binary fixed-probability choices;
  - the schedulers, to the uniform one

- On the security semantics, the lattice of levels and the indis relation, again, just like at the possibilistic case.

**datatype** *level* = *Lo* | *Hi*

**lemma** [*simp*]:  $\bigwedge l. l \neq Hi \longleftrightarrow l = Lo$  **and**

[*simp*]:  $\bigwedge l. Hi \neq l \longleftrightarrow Lo = l$  **and**

[*simp*]:  $\bigwedge l. l \neq Lo \longleftrightarrow l = Hi$  **and**

[*simp*]:  $\bigwedge l. Lo \neq l \longleftrightarrow Hi = l$

**by** (*metis level.exhaust level.simps(2)*)<sup>+</sup>

**lemma** [*dest*]:  $\bigwedge l A. [l \in A; Lo \notin A] \Longrightarrow l = Hi$  **and**

[*dest*]:  $\bigwedge l A. [l \in A; Hi \notin A] \Longrightarrow l = Lo$

**by** (*metis level.exhaust*)<sup>+</sup>

**declare** *level.split*[*split*]

**instantiation** *level* :: *complete-lattice*

**begin**

**definition** *top-level*: *top*  $\equiv Hi$

**definition** *bot-level*: *bot*  $\equiv Lo$

**definition** *inf-level*: *inf* *l1* *l2*  $\equiv$  if *Lo*  $\in$  {*l1*,*l2*} then *Lo* else *Hi*

**definition** *sup-level*: *sup* *l1* *l2*  $\equiv$  if *Hi*  $\in$  {*l1*,*l2*} then *Hi* else *Lo*

**definition** *less-eq-level*: *less-eq* *l1* *l2*  $\equiv$  (*l1* = *Lo*  $\vee$  *l2* = *Hi*)

**definition** *less-level*: *less* *l1* *l2*  $\equiv$  *l1* = *Lo*  $\wedge$  *l2* = *Hi*

**definition** *Inf-level*: *Inf* *L*  $\equiv$  if *Lo*  $\in$  *L* then *Lo* else *Hi*

**definition** *Sup-level*: *Sup* *L*  $\equiv$  if *Hi*  $\in$  *L* then *Hi* else *Lo*

**instance**

**proof** **qed** (*auto simp: top-level bot-level inf-level sup-level less-eq-level less-level Inf-level Sup-level*)

**end**

**lemma** *sup-eq-Lo*[*simp*]: *sup* *a* *b* = *Lo*  $\longleftrightarrow$  *a* = *Lo*  $\wedge$  *b* = *Lo*

**by** (*auto simp: sup-level*)

**datatype** *var* = *h* | *h'* | *l* | *l'*

**datatype** *exp* = *Ct nat* | *Var var* | *Plus exp exp* | *Minus exp exp*

**datatype** *test* = *Tr* | *Eq exp exp* | *Gt exp exp* | *Non test*

**datatype** *atom* = *Assign var exp*

**type-synonym** *state* = *var*  $\Rightarrow$  *nat*

**syntax**

-*assign* :: '*a*  $\Rightarrow$  '*a*  $\Rightarrow$  '*a* (- ::= - [1000, 61] 61)

**translations**

*x* ::= *expr* == *CONST Atm* (*CONST Assign x expr*)

**primrec** *sec* **where**

*sec h* = *Hi*

| *sec*  $h' = Hi$   
 | *sec*  $l = Lo$   
 | *sec*  $l' = Lo$

**fun** *eval* **where**

*eval* (*Ct*  $n$ )  $s = n$   
 | *eval* (*Var*  $x$ )  $s = s\ x$   
 | *eval* (*Plus*  $e1\ e2$ )  $s = \text{eval } e1\ s + \text{eval } e2\ s$   
 | *eval* (*Minus*  $e1\ e2$ )  $s = \text{eval } e1\ s - \text{eval } e2\ s$

**fun** *tval* **where**

*tval* *Tr*  $s = True$   
 | *tval* (*Eq*  $e1\ e2$ )  $s = (\text{eval } e1\ s = \text{eval } e2\ s)$   
 | *tval* (*Gt*  $e1\ e2$ )  $s = (\text{eval } e1\ s > \text{eval } e2\ s)$   
 | *tval* (*Non*  $e$ )  $s = (\neg \text{tval } e\ s)$

**fun** *aval* **where**

*aval* (*Assign*  $x\ e$ )  $s = (s\ (x := \text{eval } e\ s))$

**definition** *indis* :: (*state* \* *state*) **setwhere**

*indis*  $\equiv \{(s, t). \text{ALL } x. \text{sec } x = Lo \longrightarrow s\ x = t\ x\}$

**interpretation** *Example-PL: PL-Indis* *tval* *aval* *indis*

**proof**

**show** *equiv UNIV indis*

**unfolding** *refl-on-def sym-def trans-def equiv-def indis-def* **by** *auto*

**qed**

**fun** *exprSec* **where**

*exprSec* (*Ct*  $n$ ) = *bot*  
 | *exprSec* (*Var*  $x$ ) = *sec*  $x$   
 | *exprSec* (*Plus*  $e1\ e2$ ) = *sup* (*exprSec*  $e1$ ) (*exprSec*  $e2$ )  
 | *exprSec* (*Minus*  $e1\ e2$ ) = *sup* (*exprSec*  $e1$ ) (*exprSec*  $e2$ )

**fun** *tstSec* **where**

*tstSec* *Tr* = *bot*  
 | *tstSec* (*Eq*  $e1\ e2$ ) = *sup* (*exprSec*  $e1$ ) (*exprSec*  $e2$ )  
 | *tstSec* (*Gt*  $e1\ e2$ ) = *sup* (*exprSec*  $e1$ ) (*exprSec*  $e2$ )  
 | *tstSec* (*Non*  $e$ ) = *tstSec*  $e$

**lemma** *exprSec-Lo-eval-eq*: *exprSec* *expr* = *Lo*  $\implies (s, t) \in \text{indis} \implies \text{eval } \text{expr } s = \text{eval } \text{expr } t$

**by** (*induct expr*) (*auto simp: indis-def*)

**lemma** *compatAtmSyntactic[simp]*: *exprSec* *expr* = *Lo*  $\vee$  *sec*  $v = Hi \implies \text{Example-PL.compatAtm (Assign } v\ \text{expr)}$

**unfolding** *Example-PL.compatAtm-def*

**by** (*induct expr*)

(*auto simp: indis-def intro!: arg-cong2[where f=(+)] arg-cong2[where f=(-)]*)

*exprSec-Lo-eval-eq*)

**lemma** *presAtmSyntactic[simp]*:  $sec\ v = Hi \implies Example-PL.presAtm$  (*Assign v expr*)

**unfolding** *Example-PL.presAtm-def* **by** (*simp add: indis-def*)

**lemma** *compatTstSyntactic[simp]*:  $tstSec\ tst = Lo \implies Example-PL.compatTst\ tst$

**unfolding** *Example-PL.compatTst-def*

**by** (*induct tst*)

(*simp-all, safe del: iffI*

*intro!*: *arg-cong2[where f=(=)] arg-cong2[where f=(<) :: nat  $\Rightarrow$  nat  $\Rightarrow$  bool]* *exprSec-Lo-eval-eq*)

**lemma** *Example-PL.SC-discr* ( $h ::= Ct\ 0$ )

**by** (*simp add: Example-PL.SC-discr.simps*)

**abbreviation** *siso*  $c \equiv Example-PL.siso\ c$

**abbreviation** *siso0*  $c \equiv Example-PL.siso0\ c$

**abbreviation** *discr*  $c \equiv Example-PL.discr\ c$

**abbreviation** *discr0*  $c \equiv Example-PL.discr0\ c$

**abbreviation** *Sbis-abbrev* (**infix**  $\approx_s$  55) **where**  $c1 \approx_s c2 \equiv (c1, c2) \in Example-PL.Sbis$

**abbreviation** *ZObis-abbrev* (**infix**  $\approx_{01}$  55) **where**  $c1 \approx_{01} c2 \equiv (c1, c2) \in Example-PL.ZObis$

**abbreviation** *ZObisT-abbrev* (**infix**  $\approx_{01T}$  55) **where**  $c1 \approx_{01T} c2 \equiv (c1, c2) \in Example-PL.ZObisT$

**abbreviation** *Wbis-abbrev* (**infix**  $\approx_w$  55) **where**  $c1 \approx_w c2 \equiv (c1, c2) \in Example-PL.Wbis$

**abbreviation** *WbisT-abbrev* (**infix**  $\approx_{wT}$  55) **where**  $c1 \approx_{wT} c2 \equiv (c1, c2) \in Example-PL.WbisT$

**abbreviation** *BisT-abbrev* (**infix**  $\approx_T$  55) **where**  $c1 \approx_T c2 \equiv (c1, c2) \in Example-PL.BisT$

## 7.1 Programs from EXAMPLE 1

**definition** [*simp*]:  $c0 = (h ::= Ct\ 0)$

**definition** [*simp*]:  $c1 = (if\ Eq\ (Var\ l)\ (Ct\ 0)\ then\ h ::= Ct\ 1\ else\ l ::= Ct\ 2)$

**definition** [*simp*]:  $c2 = (if\ Eq\ (Var\ h)\ (Ct\ 0)\ then\ h ::= Ct\ 1\ else\ h ::= Ct\ 2)$

**definition** [*simp*]:  $c3 = (if\ Eq\ (Var\ h)\ (Ct\ 0)\ then\ h ::= Ct\ 1\ ;;\ h ::= Ct\ 2\ else\ h ::= Ct\ 3)$

**definition** [*simp*]:  $c4 = l ::= Ct\ 4\ ;;\ c3$

**definition** [*simp*]:  $c5 = c3\ ;;\ l ::= Ct\ 4$

**definition** [*simp*]:  $c6 = l ::= Var\ h$

**definition** [simp]:  $c7 = l ::= \text{Var } h ;; l ::= \text{Ct } 0$

**definition** [simp]:  $c8 = h' ::= \text{Var } h ;;$   
 $\text{while } Gt (\text{Var } h) (\text{Ct } 0) \text{ do } (h ::= \text{Minus } (\text{Var } h) (\text{Ct } 1) ;; h' ::= \text{Plus } (\text{Var } h')$   
 $(\text{Ct } 1)) ;;$   
 $l ::= \text{Ct } 4$

**definition** [simp]:  $c9 = c7 \mid l' ::= \text{Var } l$

**definition** [simp]:  $c10 = c5 \mid l ::= \text{Ct } 5$

**definition** [simp]:  $c11 = c8 \mid l ::= \text{Ct } 5$

**declare** *bot-level*[iff]

**theorem** *c0*: *siso c0 discr c0*  
**by** *auto*

**theorem** *c1*: *siso c1 c1  $\approx_s$  c1*  
**by** *auto*

**theorem** *c2*: *discr c2*  
**by** *auto*

**theorem** *Sbis-c2*: *c2  $\approx_s$  c2*  
**oops**

**theorem** *c3*: *discr c3*  
**by** *auto*

**theorem** *c4*: *c4  $\approx_{01}$  c4*  
**by** *auto*

**theorem** *c5*: *c5  $\approx_w$  c5*  
**by** *auto*

Example 4 from the paper

**theorem** *c3  $\approx_{wT}$  c3* **by** *auto*

**theorem** *c5  $\approx_{wT}$  c5* **by** *auto*

**corollary** *discr (while Eq (Var h) (Ct 0) do h ::= Ct 0)*  
**by** *auto*

Example 5 from the paper

**definition** [simp]:  $c12 \equiv h ::= \text{Ct } 4 ;;$   
 $\text{while } Gt (\text{Var } h) (\text{Ct } 0)$   
 $\text{do } (h ::= \text{Minus } (\text{Var } h) (\text{Ct } 1) ;; h' ::= \text{Plus } (\text{Var } h') (\text{Ct } 1)) ;;$

```

l ::= Ct 1

corollary (c12 | l ::= Ct 2) ≈T (c12 | l ::= Ct 2)
  by auto

definition [simp]: c13 =
  (if Eq (Var h) (Ct 0) then h ::= Ct 1 ;; l ::= Ct 2 else l ::= Ct 2) ;; l' ::= Ct 4

lemma c13-inner:
  (h ::= Ct 1 ;; l ::= Ct 2) ≈wT (l ::= Ct 2)
proof –
  define φ where φ =
    (λ(c :: (test, atom) com) (d :: (test, atom) com).
      c = h ::= Ct 1 ;; l ::= Ct 2 ∧ d = l ::= Ct 2 ∨
      d = h ::= Ct 1 ;; l ::= Ct 2 ∧ c = l ::= Ct 2)
  then have φ (h ::= Ct 1 ;; l ::= Ct 2) (l ::= Ct 2)
    by auto
  then show ?thesis
proof (induct rule: Example-PL.WbisT-coinduct[where φ=φ])
  case sym then show ?case by (auto simp add: φ-def)
next
  case (cont c s d t c' s') then show ?case
    by (auto simp add: φ-def intro!: exI[of - l ::= Ct 2] exI[of - t])
      (auto simp: indis-def)
next
  have exec:
    ∧t. Example-PL.MtransT (h ::= Ct 1 ;; l ::= Ct 2, t) (aval (Assign l (Ct
2)) (aval (Assign h (Ct 1)) t))
    by (simp del: aval.simps)
      (blast intro: Example-PL.transC-MtransT Example-PL.transC-MtransC.SeqT
Example-PL.transT.Atm Example-PL.transT-MtransT)
  case (termi c s d t s') with exec show ?case
    by (auto simp add: φ-def intro!: exI[of - t (h := 1, l := 2)])
      (auto simp: indis-def)
  qed
qed

theorem c13 ≈wT c13
  using c13-inner
  by (auto intro!: Example-PL.Seq-WbisT Example-PL.WbisT-If-cross)

end

```

## References

- [1] A. Popescu, J. Hölzl, and T. Nipkow. Proving possibilistic, probabilistic noninterference. In *Certified Programs and Proofs (CPP) '12*, 2012.