Possibilistic Noninterference*

Andrei Popescu Johannes Hölzl

August 16, 2018

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

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

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.

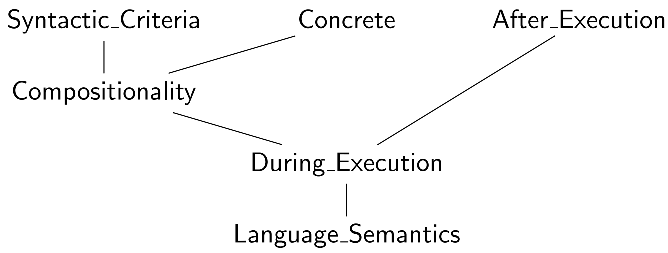

Figure 1: Main Theory Structure

During_Execution[1] mainly corresponds to §3 in [1], defining the various coinductive notions from there: self isomorphism, discreetness, 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.[2] 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., $c \approx_{\mathsf{S}} d \Longrightarrow c \approx_{\mathsf{WT}} d$ instead of $c \approx_{\mathsf{S}} c \Longrightarrow c \approx_{\mathsf{WT}} c$.

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, indication the compositionality relationship between notions of noninterference and language constructs, e.g., While_WbisT (while versus termination-sensitive weak bisimilarity), Par_ZObis (parallel composition versus 01-bisimilarity).

Theories During_Execution and Compositionality also include the novel notion of noninterference $\approx_{\mathsf{T}}$ 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 $\approx_{\mathsf{T}}$ and the other bisimilarities (Prop. 3.(1) from [1]), while various intuitively named theorems from Language_Semantics state the compositionality properties of $\approx_{\mathsf{T}}$ (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., $\overline{\mathsf{discr}}$, 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-

[1] "During-execution" (bisimilarity-based) noninterference should be contrasted with "after-execution" (trace-based) noninterference according to the distinction made in [1] at the begining of §7.

[2] 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 $Union\ (Retr\ `\ Theta) \leq Retr\ (Union\ Theta)$
$\langle proof \rangle$

lemma *mono-Un*:
assumes *mono Retr*
shows $Retr\ theta\ Un\ Retr\ theta' \subseteq Retr\ (theta\ Un\ theta')$
$\langle proof \rangle$

lemma *sym-Union*:
assumes $\bigwedge theta.\ theta \in Theta \Longrightarrow sym\ theta$
shows $sym\ (Union\ Theta)$
$\langle proof \rangle$

lemma *sym-Un*:
assumes *sym theta1* **and** *sym theta2*
shows $sym\ (theta1\ Un\ theta2)$
$\langle proof \rangle$

lemma *simul-Union*:
assumes *mono Retr*

and $\bigwedge$*theta. theta* $\in$ *Theta* $\Longrightarrow$ *simul Retr theta*
shows *simul Retr* (*Union Theta*)
⟨*proof*⟩

lemma *simul-Un*:
assumes *mono Retr* **and** *simul Retr theta1* **and** *simul Retr theta2*
shows *simul Retr* (*theta1 Un theta2*)
⟨*proof*⟩

lemma *bisim-Union*:
assumes *mono Retr* **and** $\bigwedge$*theta. theta* $\in$ *Theta* $\Longrightarrow$ *bisim Retr theta*
shows *bisim Retr* (*Union Theta*)
⟨*proof*⟩

lemma *bisim-Un*:
assumes *mono Retr* **and** *bisim Retr theta1* **and** *bisim Retr theta2*
shows *bisim Retr* (*theta1 Un theta2*)
⟨*proof*⟩

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

lemma *bisim-bis*[*simp*]:
assumes *mono Retr*
shows *bisim Retr* (*bis Retr*)
⟨*proof*⟩

corollary *sym-bis*[*simp*]: *mono Retr* $\Longrightarrow$ *sym* (*bis Retr*)
and *simul-bis*[*simp*]: *mono Retr* $\Longrightarrow$ *simul Retr* (*bis Retr*)
⟨*proof*⟩

lemma *bis-raw-coind*:
assumes *mono Retr* **and** *sym theta* **and** *theta* $\subseteq$ *Retr theta*
shows *theta* $\subseteq$ *bis Retr*
⟨*proof*⟩

lemma *bis-prefix*[*simp*]:
assumes *mono Retr*
shows *bis Retr* $\subseteq$ *Retr* (*bis Retr*)
⟨*proof*⟩

lemma *bis-coind*:
assumes $*$: *mono Retr* **and** *sym theta* **and** $**$: *theta* $\subseteq$ *Retr* (*theta Un* (*bis Retr*))
shows *theta* $\subseteq$ *bis Retr*
⟨*proof*⟩

lemma *bis-coind2*:
assumes $*$: *mono Retr* **and**
$**$: *theta* $\subseteq$ *Retr* (*theta Un* (*bis Retr*)) **and**

$***$: *theta* $\hat{}{-1}$ $\subseteq$ *Retr* ((*theta* $\hat{}{-1}$) *Un* (*bis Retr*))
shows *theta* $\subseteq$ *bis Retr*
$\langle$*proof*$\rangle$

lemma *bis-raw-coind2*:
assumes $*$: *mono Retr* **and**
$**$: *theta* $\subseteq$ *Retr theta* **and**
$***$: *theta* $\hat{}{-1}$ $\subseteq$ *Retr* (*theta* $\hat{}{-1}$)
shows *theta* $\subseteq$ *bis Retr*
$\langle$*proof*$\rangle$

lemma *mono-bis*:
assumes *mono Retr1* **and** *mono Retr2*
and $\bigwedge$ *theta. Retr1 theta* $\subseteq$ *Retr2 theta*
shows *bis Retr1* $\subseteq$ *bis Retr2*
$\langle$*proof*$\rangle$

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) ⇒ 'state ⇒ bool
(**infix** →t 55)
where
 Atm[*simp*]:
(Atm atm, s) →t aval atm s
| *WhileFalse*[*simp*]:
~ tval tst s ⟹ (While tst c, s) →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*
⟨*proof*⟩

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*) ⇒ ′*state* ⇒ *bool*
(**infix** →∗*t* *55*)
where
StepT:
⟦*cf* →∗*c* *cf′*; *cf′* →*t* *s″*⟧ ⟹ *cf* →∗*t* *s″*

lemma *MtransC-rtranclp-transC*:
MtransC = *transC* ˆ∗∗
⟨*proof*⟩

lemma *transC-MtransC*[*simp*]:
assumes *cf* →*c* *cf′*
shows *cf* →∗*c* *cf′*
⟨*proof*⟩

lemma *MtransC-Trans*:
assumes *cf* →∗*c* *cf′* **and** *cf′* →∗*c* *cf″*
shows *cf* →∗*c* *cf″*
⟨*proof*⟩

lemma *MtransC-StepC*:
assumes ∗: *cf* →∗*c* *cf′* **and** ∗∗: *cf′* →*c* *cf″*
shows *cf* →∗*c* *cf″*
⟨*proof*⟩

lemma *MtransC-induct*[*consumes 1*, *case-names Refl Trans*]:
assumes *cf* →∗*c* *cf′*
and ⋀*cf*. *phi cf cf*
and
⋀ *cf cf′ cf″*.
⟦*cf* →∗*c* *cf′*; *phi cf cf′*; *cf′* →*c* *cf″*⟧
⟹ *phi cf cf″*
shows *phi cf cf′*
⟨*proof*⟩

lemma *MtransC-induct2*[*consumes 1*, *case-names Refl Trans*, *induct pred*: *MtransC*]:
assumes (*c*,*s*) →∗*c* (*c′*,*s′*)
and ⋀*c s*. *phi c s c s*

and
$\bigwedge$ *c s c′ s′ c″ s″.*
$[\![(c,s) \rightarrow\!*c\ (c',s'); phi\ c\ s\ c'\ s'; (c',s') \rightarrow c\ (c'',s'')]\!]$
$\Longrightarrow phi\ c\ s\ c''\ s''$
shows *phi c s c′ s′*
⟨*proof*⟩

lemma *transT-MtransT*[*simp*]:
assumes $cf \rightarrow t\ s'$
shows $cf \rightarrow\!*t\ s'$
⟨*proof*⟩

lemma *MtransC-MtransT*:
assumes $cf \rightarrow\!*c\ cf'$ **and** $cf' \rightarrow\!*t\ cf''$
shows $cf \rightarrow\!*t\ cf''$
⟨*proof*⟩

lemma *transC-MtransT*[*simp*]:
assumes $cf \rightarrow c\ cf'$ **and** $cf' \rightarrow\!*t\ s''$
shows $cf \rightarrow\!*t\ s''$
⟨*proof*⟩

Inversion rules, nchotomies and such:

lemma *Atm-transC-simp*[*simp*]:
~ $(Atm\ atm,\ s) \rightarrow c\ cf$
⟨*proof*⟩

lemma *Atm-transC-invert*[*elim*!]:
assumes $(Atm\ atm,\ s) \rightarrow c\ cf$
shows *phi*
⟨*proof*⟩

lemma *Atm-transT-invert*[*elim*!]:
assumes $(Atm\ atm,\ s) \rightarrow t\ s'$
and $s' = aval\ atm\ s \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *Seq-transC-invert*[*elim*!]:
assumes $(c1\ ;;\ c2,\ s) \rightarrow c\ (c',\ s')$
and $\bigwedge c1'.\ [\![(c1,\ s) \rightarrow c\ (c1',s');\ c' = c1'\ ;;\ c2]\!] \Longrightarrow phi$
and $[\![(c1,\ s) \rightarrow t\ s';\ c' = c2]\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *Seq-transT-invert*[*simp*]:
~ $(c1\ ;;\ c2,\ s) \rightarrow t\ s'$
⟨*proof*⟩

lemma *If-transC-invert*[*elim*!]:
assumes (*If tst c1 c2*, s) $\rightarrow c$ (c', s')
and $[\![$*tval tst s*; $c' = c1$; $s' = s]\!] \Longrightarrow$ *phi*
and $[\![$~ *tval tst s*; $c' = c2$; $s' = s]\!] \Longrightarrow$ *phi*
shows *phi*
⟨*proof*⟩

lemma *If-transT-simp*[*simp*]:
~ (*If b c1 c2*, s) $\rightarrow t$ s'
⟨*proof*⟩

lemma *If-transT-invert*[*elim*!]:
assumes (*If b c1 c2*, s) $\rightarrow t$ s'
shows *phi*
⟨*proof*⟩

lemma *While-transC-invert*[*elim*]:
assumes (*While tst c1*, s) $\rightarrow c$ (c', s')
and $[\![$*tval tst s*; $c' = c1$;; (*While tst c1*); $s' = s]\!] \Longrightarrow$ *phi*
shows *phi*
⟨*proof*⟩

lemma *While-transT-invert*[*elim*!]:
assumes (*While tst c1*, s) $\rightarrow t$ s'
and $[\![$~ *tval tst s*; $s' = s]\!] \Longrightarrow$ *phi*
shows *phi*
⟨*proof*⟩

lemma *Par-transC-invert*[*elim*!]:
assumes (*Par c1 c2*, s) $\rightarrow c$ (c', s')
and $\bigwedge$ $c1'$. $[\![$($c1$, s) $\rightarrow c$ ($c1'$,s'); $c' =$ *Par* $c1'$ $c2]\!] \Longrightarrow$ *phi*
and $[\![$($c1$, s) $\rightarrow t$ s'; $c' = c2]\!] \Longrightarrow$ *phi*
and $\bigwedge$ $c2'$. $[\![$($c2$, s) $\rightarrow c$ ($c2'$,s'); $c' =$ *Par* $c1$ $c2']\!] \Longrightarrow$ *phi*
and $[\![$($c2$, s) $\rightarrow t$ s'; $c' = c1]\!] \Longrightarrow$ *phi*
shows *phi*
⟨*proof*⟩

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

lemma *Par-transT-invert*[*elim*!]:
assumes (*Par c1 c2*, s) $\rightarrow t$ s'
shows *phi*
⟨*proof*⟩

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

⟨*proof*⟩

corollary *trans-invert*:
assumes
$\bigwedge c' s'. (c,s) \to c\ (c',s') \Longrightarrow phi$
and $\bigwedge s'. (c,s) \to t\ s' \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *not-transC-transT*:
$[\![cf \to c\ cf'; cf \to t\ s']\!] \Longrightarrow phi$
⟨*proof*⟩

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

lemma *MtransT-invert2*:
assumes $(c, s) \to{*}t\ s''$
and $\bigwedge c' s'. [\![(c,s) \to{*}c\ (c',s'); (c', s') \to t\ s'']\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *Seq-MtransC-invert*[*elim*!]:
assumes $(c1 ;; c2, s) \to{*}c\ (d', t')$
and $\bigwedge c1'. [\![(c1, s) \to{*}c\ (c1',t'); d' = c1' ;; c2]\!] \Longrightarrow phi$
and $\bigwedge s'. [\![(c1, s) \to{*}t\ s'; (c2, s') \to{*}c\ (d',t')]\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

lemma *Seq-MtransT-invert*[*elim*!]:
assumes $*$: $(c1 ;; c2, s) \to{*}t\ s''$
and $**$: $\bigwedge s'. [\![(c1, s) \to{*}t\ s'; (c2, s') \to{*}t\ s'']\!] \Longrightarrow phi$
shows *phi*
⟨*proof*⟩

Direct rules for the multi-step relations

lemma *Seq-MtransC*[*simp*]:
assumes $(c1, s) \to{*}c\ (c1', s')$
shows $(c1 ;; c2, s) \to{*}c\ (c1' ;; c2, s')$
⟨*proof*⟩

lemma *Seq-MtransT-MtransC*[*simp*]:
assumes $(c1, s) \to{*}t\ s'$
shows $(c1 ;; c2, s) \to{*}c\ (c2, s')$
⟨*proof*⟩

lemma *ParCL-MtransC*[*simp*]:
assumes $(c1, s) \to{*}c\ (c1', s')$
shows $(Par\ c1\ c2, s) \to{*}c\ (Par\ c1'\ c2, s')$
⟨*proof*⟩

lemma *ParCR-MtransC*[*simp*]:
assumes $(c2,\ s) \rightarrow{*}c\ (c2',\ s')$
shows $(Par\ c1\ c2,\ s) \rightarrow{*}c\ (Par\ c1\ c2',\ s')$
⟨*proof*⟩

lemma *ParTL-MtransC*[*simp*]:
assumes $(c1,\ s) \rightarrow{*}t\ s'$
shows $(Par\ c1\ c2,\ s) \rightarrow{*}c\ (c2,\ s')$
⟨*proof*⟩

lemma *ParTR-MtransC*[*simp*]:
assumes $(c2,\ s) \rightarrow{*}t\ s'$
shows $(Par\ c1\ c2,\ s) \rightarrow{*}c\ (c1,\ s')$
⟨*proof*⟩

3.2 Sublanguages

fun *noWhile* **where**
noWhile (*Atm atm*) = *True*
|*noWhile* (*c1* ;; *c2*) = (*noWhile c1* ∧ *noWhile c2*)
|*noWhile* (*If b c1 c2*) = (*noWhile c1* ∧ *noWhile c2*)
|*noWhile* (*While b c*) = *False*
|*noWhile* (*Par c1 c2*) = (*noWhile c1* ∧ *noWhile c2*)

fun *seq* **where**
seq (*Atm atm*) = *True*
|*seq* (*c1* ;; *c2*) = (*seq c1* ∧ *seq c2*)
|*seq* (*If b c1 c2*) = (*seq c1* ∧ *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*⟩

lemma *seq-transC*:
assumes *seq c* **and** $(c,s) \rightarrow c\ (c',s')$
shows *seq* c'
⟨*proof*⟩

abbreviation *wfP-on* **where**
wfP-on phi A ≡ *wfP* ($\lambda a\ b.\ a \in A \wedge b \in A \wedge phi\ a\ b$)

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*]:
$noWhile\ c \Longrightarrow numSt\ c > 0$
⟨*proof*⟩

lemma *numSt-transC*:
assumes *noWhile c* **and** $(c,s) \to c\ (c',s')$
shows $numSt\ c' < numSt\ c$
⟨*proof*⟩

corollary *wfP-tranC-noWhile*:
$wfP\ (\lambda\ (c',s')\ (c,s).\ noWhile\ c \wedge (c,s) \to c\ (c',s'))$
⟨*proof*⟩

lemma *noWhile-MtransT*:
assumes *noWhile c*
shows $\exists\ s'.\ (c,s) \to *t\ s'$
⟨*proof*⟩

coinductive *mayDiverge* **where**
intro:
$[\![(c,s) \to c\ (c',s') \wedge mayDiverge\ c'\ s']\!]$
$\Longrightarrow mayDiverge\ c\ s$

Coinduction for may-diverge :

lemma *mayDiverge-coind*[*consumes 1*, *case-names Hyp*, *induct pred*: *mayDiverge*]:
assumes $*$: *phi c s* **and**
$**$: $\bigwedge\ c\ s.\ phi\ c\ s \Longrightarrow$
$\exists\ c'\ s'.\ (c,s) \to c\ (c',s') \wedge (phi\ c'\ s' \vee mayDiverge\ c'\ s')$
shows *mayDiverge c s*
⟨*proof*⟩

lemma *mayDiverge-raw-coind*[*consumes 1*, *case-names Hyp*]:
assumes $*$: *phi c s* **and**
$**$: $\bigwedge\ c\ s.\ phi\ c\ s \Longrightarrow$
$\exists\ c'\ s'.\ (c,s) \to c\ (c',s') \wedge phi\ c'\ s'$
shows *mayDiverge c s*
⟨*proof*⟩

May-diverge versus transition:

lemma *mayDiverge-transC*:
assumes *mayDiverge c s*
shows $\exists\ c'\ s'.\ (c,s) \to c\ (c',s') \wedge mayDiverge\ c'\ s'$
⟨*proof*⟩

lemma *transC-mayDiverge*:
assumes $(c,s) \to c\ (c',s')$ **and** *mayDiverge* $c'\ s'$
shows *mayDiverge c s*
⟨*proof*⟩

lemma *mayDiverge-not-transT*:
assumes *mayDiverge c s*
shows $\neg\ (c,s) \to t\ s'$
⟨*proof*⟩

lemma *MtransC-mayDiverge*:
assumes $(c,s) \to * c\ (c',s')$ **and** *mayDiverge* $c'\ s'$
shows *mayDiverge c s*
⟨*proof*⟩

lemma *not-MtransT-mayDiverge*:
assumes $\bigwedge s'.\ \neg\ (c,s) \to * t\ s'$
shows *mayDiverge c s*
⟨*proof*⟩

lemma *not-mayDiverge-Atm*[*simp*]:
¬ *mayDiverge* (*Atm atm*) *s*
⟨*proof*⟩

lemma *mayDiverge-Seq-L*:
assumes *mayDiverge c1 s*
shows *mayDiverge* (*c1* ;; *c2*) *s*
⟨*proof*⟩

lemma *mayDiverge-Seq-R*:
assumes *c1*: $(c1,\ s) \to * t\ s'$ **and** *c2*: *mayDiverge c2* s'
shows *mayDiverge* (*c1* ;; *c2*) *s*
⟨*proof*⟩

lemma *mayDiverge-If-L*:
assumes *tval tst s* **and** *mayDiverge c1 s*
shows *mayDiverge* (*If tst c1 c2*) *s*
⟨*proof*⟩

lemma *mayDiverge-If-R*:
assumes ¬ *tval tst s* **and** *mayDiverge c2 s*
shows *mayDiverge* (*If tst c1 c2*) *s*
⟨*proof*⟩

lemma *mayDiverge-If*:
assumes *mayDiverge c1 s* **and** *mayDiverge c2 s*
shows *mayDiverge* (*If tst c1 c2*) *s*
⟨*proof*⟩

lemma *mayDiverge-Par-L*:
assumes *mayDiverge c1 s*
shows *mayDiverge* (*Par c1 c2*) *s*
⟨*proof*⟩

lemma *mayDiverge-Par-R*:
assumes *mayDiverge c2 s*
shows *mayDiverge* (*Par c1 c2*) *s*
⟨*proof*⟩

definition *mustT* **where**
mustT c s ≡ ¬ *mayDiverge c s*

lemma *mustT-transC*:
assumes *mustT c s* **and** (*c*,*s*) →*c* (*c'*,*s'*)
shows *mustT c' s'*
⟨*proof*⟩

lemma *transT-not-mustT*:
assumes (*c*,*s*) →*t s'*
shows *mustT c s*
⟨*proof*⟩

lemma *mustT-MtransC*:
assumes *mustT c s* **and** (*c*,*s*) →∗*c* (*c'*,*s'*)
shows *mustT c' s'*
⟨*proof*⟩

lemma *mustT-MtransT*:
assumes *mustT c s*
shows ∃ *s'*. (*c*,*s*) →∗*t s'*
⟨*proof*⟩

lemma *mustT-Atm*[*simp*]:
mustT (*Atm atm*) *s*
⟨*proof*⟩

lemma *mustT-Seq-L*:
assumes *mustT* (*c1* ;; *c2*) *s*
shows *mustT c1 s*
⟨*proof*⟩

lemma *mustT-Seq-R*:
assumes *mustT* (*c1* ;; *c2*) *s* **and** (*c1*, *s*) →∗*t s'*
shows *mustT c2 s'*
⟨*proof*⟩

lemma *mustT-If-L*:
assumes *tval tst s* **and** *mustT* (*If tst c1 c2*) *s*
shows *mustT c1 s*
⟨*proof*⟩

lemma *mustT-If-R*:
assumes ¬ *tval tst s* **and** *mustT* (*If tst c1 c2*) *s*
shows *mustT c2 s*
⟨*proof*⟩

lemma *mustT-If*:
assumes *mustT* (*If tst c1 c2*) *s*
shows *mustT c1 s* ∨ *mustT c2 s*
⟨*proof*⟩

lemma *mustT-Par-L*:
assumes *mustT* (*Par c1 c2*) *s*
shows *mustT c1 s*
⟨*proof*⟩

lemma *mustT-Par-R*:
assumes *mustT* (*Par c1 c2*) *s*
shows *mustT c2 s*
⟨*proof*⟩

definition *determOn* **where**
determOn phi r ≡
∀ *a b b'*. *phi a* ∧ *r a b* ∧ *r a b'* ⟶ *b* = *b'*

lemma *determOn-seq-transT*:
determOn (λ(*c*,*s*). *seq c*) *transT*
⟨*proof*⟩

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*
⟨*proof*⟩

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

lemma *indis-trans*: $[\![s \approx s';\ s' \approx s'']\!] \Longrightarrow s \approx s''$
⟨*proof*⟩

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

4.2 Compatibility and discreetness

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:
$[\![\bigwedge\ s\ c'\ s'.\ (c,s) \rightarrow c\ (c',s') \Longrightarrow s \approx s' \wedge discr\ c';$
$\bigwedge\ s\ s'.\ (c,s) \rightarrow t\ s' \Longrightarrow s \approx s']\!]$
$\Longrightarrow discr\ c$

lemma *presAtm-compatAtm*[*simp*]:
assumes *presAtm atm*
shows *compatAtm atm*
⟨*proof*⟩

Coinduction for discreetness:

lemma *discr-coind*:
assumes $*$: *phi c* **and**
$**$: $\bigwedge\ c\ s\ c'\ s'.\ [\![phi\ c;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow s \approx s' \wedge (phi\ c' \vee discr\ c')$ **and**
$***$: $\bigwedge\ c\ s\ s'.\ [\![phi\ c;\ (c,s) \rightarrow t\ s']\!] \Longrightarrow s \approx s'$
shows *discr c*
⟨*proof*⟩

lemma *discr-raw-coind*:
assumes $*$: *phi c* **and**
$**$: $\bigwedge\ c\ s\ c'\ s'.\ [\![phi\ c;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow s \approx s' \wedge phi\ c'$ **and**
$***$: $\bigwedge\ c\ s\ s'.\ [\![phi\ c;\ (c,s) \rightarrow t\ s']\!] \Longrightarrow s \approx s'$
shows *discr c*
⟨*proof*⟩

Discreetness versus transition:

lemma *discr-transC*:
assumes $*$: *discr c* **and** $**$: $(c,s) \rightarrow c\ (c',s')$
shows *discr* c'
⟨*proof*⟩

lemma *discr-MtransC*:
assumes *discr c* **and** $(c,s) \rightarrow *c\ (c',s')$
shows *discr* c'
⟨*proof*⟩

lemma *discr-transC-indis*:
assumes $*$: *discr c* **and** $**$: $(c,s) \rightarrow c\ (c',s')$
shows $s \approx s'$
⟨*proof*⟩

lemma *discr-MtransC-indis*:
assumes *discr c* **and** $(c,s) \rightarrow *c\ (c',s')$
shows $s \approx s'$
⟨*proof*⟩

lemma *discr-transT*:
assumes $*$: *discr c* **and** $**$: $(c,s) \to t\ s'$
shows $s \approx s'$
$\langle proof \rangle$

lemma *discr-MtransT*:
assumes $*$: *discr c* **and** $**$: $(c,s) \to *t\ s'$
shows $s \approx s'$
$\langle proof \rangle$

4.3 Terminating-interctive discreetness

coinductive *discr0* **where**
intro:
$[\![\bigwedge s\ c'\ s'.\ [\![mustT\ c\ s;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow s \approx s' \wedge discr0\ c';$
$\bigwedge s\ s'.\ [\![mustT\ c\ s;\ (c,s) \to t\ s']\!] \Longrightarrow s \approx s']\!]$
$\Longrightarrow discr0\ c$

Coinduction for 0-discreetness:

lemma *discr0-coind*[*consumes 1*, *case-names Cont Term*, *induct pred*: *discr0*]:
assumes $*$: *phi c* **and**
$**$: $\bigwedge c\ s\ c'\ s'.$
$[\![mustT\ c\ s;\ phi\ c;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow$
$s \approx s' \wedge (phi\ c' \vee discr0\ c')$ **and**
$***$: $\bigwedge c\ s\ s'.\ [\![mustT\ c\ s;\ phi\ c;\ (c,s) \to t\ s']\!] \Longrightarrow s \approx s'$
shows *discr0 c*
$\langle proof \rangle$

lemma *discr0-raw-coind*[*consumes 1*, *case-names Cont Term*]:
assumes $*$: *phi c* **and**
$**$: $\bigwedge c\ s\ c'\ s'.\ [\![mustT\ c\ s;\ phi\ c;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow s \approx s' \wedge phi\ c'$ **and**
$***$: $\bigwedge c\ s\ s'.\ [\![mustT\ c\ s;\ phi\ c;\ (c,s) \to t\ s']\!] \Longrightarrow s \approx s'$
shows *discr0 c*
$\langle proof \rangle$

0-Discreetness versus transition:

lemma *discr0-transC*:
assumes $*$: *discr0 c* **and** $**$: $mustT\ c\ s\ (c,s) \to c\ (c',s')$
shows $discr0\ c'$
$\langle proof \rangle$

lemma *discr0-MtransC*:
assumes *discr0 c* **and** $mustT\ c\ s\ (c,s) \to *c\ (c',s')$
shows $discr0\ c'$
$\langle proof \rangle$

lemma *discr0-transC-indis*:
assumes $*$: *discr0 c* **and** $**$: $mustT\ c\ s\ (c,s) \to c\ (c',s')$
shows $s \approx s'$

$\langle proof \rangle$

lemma *discr0-MtransC-indis*:
assumes *discr0 c* **and** *mustT c s* $(c,s) \rightarrow\!*c\ (c',s')$
shows $s \approx s'$
$\langle proof \rangle$

lemma *discr0-transT*:
assumes $*$: *discr0 c* **and** $**$: *mustT c s* $(c,s) \rightarrow t\ s'$
shows $s \approx s'$
$\langle proof \rangle$

lemma *discr0-MtransT*:
assumes $*$: *discr0 c* **and** $***$: *mustT c s* **and** $**$: $(c,s) \rightarrow\!*t\ s'$
shows $s \approx s'$
$\langle proof \rangle$

lemma *discr-discr0*[*simp*]: *discr c* $\Longrightarrow$ *discr0 c*
$\langle proof \rangle$

4.4 Self-isomorphism

coinductive *siso* **where**
intro:
$[\![\bigwedge s\ c'\ s'.\ (c,s) \rightarrow c\ (c',s') \Longrightarrow siso\ c';$
$\bigwedge s\ t\ c'\ s'.\ [\![s \approx t;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow \exists\ t'.\ s' \approx t' \wedge (c,t) \rightarrow c\ (c',t');$
$\bigwedge s\ t\ s'.\ [\![s \approx t;\ (c,s) \rightarrow t\ s']\!] \Longrightarrow \exists\ t'.\ s' \approx t' \wedge (c,t) \rightarrow t\ t']\!]$
$\Longrightarrow siso\ c$

Coinduction for self-isomorphism:

lemma *siso-coind*:
assumes $*$: *phi c* **and**
$**$: $\bigwedge c\ s\ c'\ s'.\ [\![phi\ c;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow phi\ c' \vee siso\ c'$ **and**
$***$: $\bigwedge c\ s\ t\ c'\ s'.\ [\![phi\ c;\ s \approx t;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow \exists\ t'.\ s' \approx t' \wedge (c,t) \rightarrow c\ (c',t')$ **and**
$****$: $\bigwedge c\ s\ t\ s'.\ [\![phi\ c;\ s \approx t;\ (c,s) \rightarrow t\ s']\!] \Longrightarrow \exists\ t'.\ s' \approx t' \wedge (c,t) \rightarrow t\ t'$
shows *siso c*
$\langle proof \rangle$

lemma *siso-raw-coind*:
assumes $*$: *phi c* **and**
$**$: $\bigwedge c\ s\ c'\ s'.\ [\![phi\ c;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow phi\ c'$ **and**
$***$: $\bigwedge c\ s\ t\ c'\ s'.\ [\![phi\ c;\ s \approx t;\ (c,s) \rightarrow c\ (c',s')]\!] \Longrightarrow \exists\ t'.\ s' \approx t' \wedge (c,t) \rightarrow c\ (c',t')$ **and**
$****$: $\bigwedge c\ s\ t\ s'.\ [\![phi\ c;\ s \approx t;\ (c,s) \rightarrow t\ s']\!] \Longrightarrow \exists\ t'.\ s' \approx t' \wedge (c,t) \rightarrow t\ t'$
shows *siso c*
$\langle proof \rangle$

Self-Isomorphism versus transition:

lemma *siso-transC*:
assumes $*$: *siso c* **and** $**$: $(c,s) \to c\ (c',s')$
shows *siso* c'
$\langle proof \rangle$

lemma *siso-MtransC*:
assumes *siso c* **and** $(c,s) \to{*}c\ (c',s')$
shows *siso* c'
$\langle proof \rangle$

lemma *siso-transC-indis*:
assumes $*$: *siso c* **and** $**$: $(c,s) \to c\ (c',s')$ **and** $***$: $s \approx t$
shows $\exists\ t'.\ s' \approx t' \wedge (c,t) \to c\ (c',t')$
$\langle proof \rangle$

lemma *siso-transT*:
assumes $*$: *siso c* **and** $**$: $(c,s) \to t\ s'$ **and** $***$: $s \approx t$
shows $\exists\ t'.\ s' \approx t' \wedge (c,t) \to t\ t'$
$\langle proof \rangle$

4.5 MustT-interactive self-isomorphism

coinductive *siso0* **where**
intro:
$[\![\bigwedge\ s\ c'\ s'.\ [\![mustT\ c\ s;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow siso0\ c';$
$\bigwedge\ s\ t\ c'\ s'.$
$[\![mustT\ c\ s;\ mustT\ c\ t;\ s \approx t;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow$
$\exists\ t'.\ s' \approx t' \wedge (c,t) \to c\ (c',t');$
$\bigwedge\ s\ t\ s'.$
$[\![mustT\ c\ s;\ mustT\ c\ t;\ s \approx t;\ (c,s) \to t\ s']\!] \Longrightarrow$
$\exists\ t'.\ s' \approx t' \wedge (c,t) \to t\ t']\!]$
$\Longrightarrow siso0\ c$

Coinduction for self-isomorphism:

lemma *siso0-coind*[*consumes 1*, *case-names Indef Cont Term*, *induct pred*: *discr0*]:
assumes $*$: *phi c* **and**
$**$: $\bigwedge\ c\ s\ c'\ s'.\ [\![phi\ c;\ mustT\ c\ s;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow phi\ c' \vee siso0\ c'$ **and**
$***$: $\bigwedge\ c\ s\ t\ c'\ s'.$
$[\![phi\ c;\ mustT\ c\ s;\ mustT\ c\ t;\ s \approx t;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow$
$\exists\ t'.\ s' \approx t' \wedge (c,t) \to c\ (c',t')$ **and**
$****$: $\bigwedge\ c\ s\ t\ s'.$
$[\![mustT\ c\ s;\ mustT\ c\ t;\ phi\ c;\ s \approx t;\ (c,s) \to t\ s']\!] \Longrightarrow$
$\exists\ t'.\ s' \approx t' \wedge (c,t) \to t\ t'$
shows *siso0 c*
$\langle proof \rangle$

lemma *siso0-raw-coind*[*consumes 1*, *case-names Indef Cont Term*]:
assumes $*$: *phi c* **and**
$**$: $\bigwedge\ c\ s\ c'\ s'.\ [\![phi\ c;\ mustT\ c\ s;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow phi\ c'$ **and**

$***$: $\bigwedge$ c s t c' s'.
$[\![phi\ c;\ mustT\ c\ s;\ mustT\ c\ t;\ s \approx t;\ (c,s) \to c\ (c',s')]\!] \Longrightarrow$
$\exists\ t'.\ s' \approx t' \wedge (c,t) \to c\ (c',t')$ **and**
$****$: $\bigwedge$ c s t s'.
$[\![phi\ c;\ mustT\ c\ s;\ mustT\ c\ t;\ s \approx t;\ (c,s) \to t\ s']\!] \Longrightarrow$
$\exists\ t'.\ s' \approx t' \wedge (c,t) \to t\ t'$
shows *siso0 c*
$\langle proof \rangle$

Self-Isomorphism versus transition:

lemma *siso0-transC*:
assumes $*$: *siso0 c* **and** $**$: $mustT\ c\ s\ (c,s) \to c\ (c',s')$
shows *siso0* c'
$\langle proof \rangle$

lemma *siso0-MtransC*:
assumes *siso0 c* **and** *mustT c s* **and** $(c,s) \to *c\ (c',s')$
shows *siso0* c'
$\langle proof \rangle$

lemma *siso0-transC-indis*:
assumes $*$: *siso0 c*
and $**$: $mustT\ c\ s\ mustT\ c\ t\ (c,s) \to c\ (c',s')$
and $***$: $s \approx t$
shows $\exists\ t'.\ s' \approx t' \wedge (c,t) \to c\ (c',t')$
$\langle proof \rangle$

lemma *siso0-transT*:
assumes $*$: *siso0 c*
and $**$: $mustT\ c\ s\ mustT\ c\ t\ (c,s) \to t\ s'$
and $***$: $s \approx t$
shows $\exists\ t'.\ s' \approx t' \wedge (c,t) \to t\ t'$
$\langle proof \rangle$

4.6 Notions of bisimilarity

Matchers:

definition *matchC-C* **where**
$matchC\text{-}C\ theta\ c\ d \equiv$
$\forall\ s\ t\ c'\ s'.$
$s \approx t \wedge (c,s) \to c\ (c',s')$
$\longrightarrow$
$(\exists\ d'\ t'.\ (d,t) \to c\ (d',t') \wedge s' \approx t' \wedge (c',d') \in theta)$

definition *matchC-ZOC* **where**
$matchC\text{-}ZOC\ theta\ c\ d \equiv$
$\forall\ s\ t\ c'\ s'.$
$s \approx t \wedge (c,s) \to c\ (c',s')$
$\longrightarrow$

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

definition *matchC-ZO* **where**
matchC-ZO theta c d $\equiv$
$\forall\ s\ t\ c'\ s'.$
$s \approx t \wedge (c,s) \rightarrow c\ (c',s')$
$\longrightarrow$
$(s' \approx t \wedge (c',d) \in \mathit{theta})$
$\vee$
$(\exists\ d'\ t'.\ (d,t) \rightarrow c\ (d',t') \wedge s' \approx t' \wedge (c',d') \in \mathit{theta})$
$\vee$
$(\exists\ t'.\ (d,t) \rightarrow t\ t' \wedge s' \approx t' \wedge \mathit{discr}\ c')$

definition *matchT-T* **where**
matchT-T c d $\equiv$
$\forall\ s\ t\ s'.$
$s \approx t \wedge (c,s) \rightarrow t\ s'$
$\longrightarrow$
$(\exists\ t'.\ (d,t) \rightarrow t\ t' \wedge s' \approx t')$

definition *matchT-ZO* **where**
matchT-ZO c d $\equiv$
$\forall\ s\ t\ s'.$
$s \approx t \wedge (c,s) \rightarrow t\ s'$
$\longrightarrow$
$(s' \approx t \wedge \mathit{discr}\ d)$
$\vee$
$(\exists\ d'\ t'.\ (d,t) \rightarrow c\ (d',t') \wedge s' \approx t' \wedge \mathit{discr}\ d')$
$\vee$
$(\exists\ t'.\ (d,t) \rightarrow t\ t' \wedge s' \approx t')$

definition *matchC-MC* **where**
matchC-MC theta c d $\equiv$
$\forall\ s\ t\ c'\ s'.$
$s \approx t \wedge (c,s) \rightarrow c\ (c',s')$
$\longrightarrow$
$(\exists\ d'\ t'.\ (d,t) \rightarrow{*}c\ (d',t') \wedge s' \approx t' \wedge (c',d') \in \mathit{theta})$

definition *matchC-TMC* **where**
matchC-TMC theta c d $\equiv$
$\forall\ s\ t\ c'\ s'.$
$\mathit{mustT}\ c\ s \wedge \mathit{mustT}\ d\ t \wedge s \approx t \wedge (c,s) \rightarrow c\ (c',s')$
$\longrightarrow$
$(\exists\ d'\ t'.\ (d,t) \rightarrow{*}c\ (d',t') \wedge s' \approx t' \wedge (c',d') \in \mathit{theta})$

definition *matchC-M* **where**
matchC-M theta c d ≡
$\forall$ *s t c′ s′*.
$s \approx t \wedge (c,s) \rightarrow c\ (c',s')$
$\longrightarrow$
$(\exists\ d'\ t'.\ (d,t) \rightarrow{*}c\ (d',t') \wedge s' \approx t' \wedge (c',d') \in theta)$
$\vee$
$(\exists\ t'.\ (d,t) \rightarrow{*}t\ t' \wedge s' \approx t' \wedge discr\ c')$

definition *matchT-MT* **where**
matchT-MT c d ≡
$\forall$ *s t s′*.
$s \approx t \wedge (c,s) \rightarrow t\ s'$
$\longrightarrow$
$(\exists\ t'.\ (d,t) \rightarrow{*}t\ t' \wedge s' \approx t')$

definition *matchT-TMT* **where**
matchT-TMT c d ≡
$\forall$ *s t s′*.
$mustT\ c\ s \wedge mustT\ d\ t \wedge s \approx t \wedge (c,s) \rightarrow t\ s'$
$\longrightarrow$
$(\exists\ t'.\ (d,t) \rightarrow{*}t\ t' \wedge s' \approx t')$

definition *matchT-M* **where**
matchT-M c d ≡
$\forall$ *s t s′*.
$s \approx t \wedge (c,s) \rightarrow t\ s'$
$\longrightarrow$
$(\exists\ d'\ t'.\ (d,t) \rightarrow{*}c\ (d',t') \wedge s' \approx t' \wedge discr\ d')$
$\vee$
$(\exists\ t'.\ (d,t) \rightarrow{*}t\ t' \wedge s' \approx t')$

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*:
matchC-C theta d c =
($\forall$ *s t d′ t′*.
$s \approx t \wedge (d,t) \rightarrow c\ (d',t')$
$\longrightarrow$
$(\exists\ c'\ s'.\ (c,s) \rightarrow c\ (c',s') \wedge s' \approx t' \wedge (d',c') \in theta)$)
⟨*proof*⟩

lemma *matchC-ZOC-def2*:
matchC-ZOC theta d c=
$(\forall\ s\ t\ d'\ t'.$
$s \approx t \wedge (d,t) \rightarrow c\ (d',t')$
$\longrightarrow$
$(s \approx t' \wedge (d',c) \in theta)$
$\vee$
$(\exists\ c'\ s'.\ (c,s) \rightarrow c\ (c',s') \wedge s' \approx t' \wedge (d',c') \in theta))$
$\langle proof \rangle$

lemma *matchC-ZO-def2*:
matchC-ZO theta d c =
$(\forall\ s\ t\ d'\ t'.$
$s \approx t \wedge (d,t) \rightarrow c\ (d',t')$
$\longrightarrow$
$(s \approx t' \wedge (d',c) \in theta)$
$\vee$
$(\exists\ c'\ s'.\ (c,s) \rightarrow c\ (c',s') \wedge s' \approx t' \wedge (d',c') \in theta)$
$\vee$
$(\exists\ s'.\ (c,s) \rightarrow t\ s' \wedge s' \approx t' \wedge discr\ d'))$
$\langle proof \rangle$

lemma *matchT-T-def2*:
matchT-T d c =
$(\forall\ s\ t\ t'.$
$s \approx t \wedge (d,t) \rightarrow t\ t'$
$\longrightarrow$
$(\exists\ s'.\ (c,s) \rightarrow t\ s' \wedge s' \approx t'))$
$\langle proof \rangle$

lemma *matchT-ZO-def2*:
matchT-ZO d c =
$(\forall\ s\ t\ t'.$
$s \approx t \wedge (d,t) \rightarrow t\ t'$
$\longrightarrow$
$(s \approx t' \wedge discr\ c)$
$\vee$
$(\exists\ c'\ s'.\ (c,s) \rightarrow c\ (c',s') \wedge s' \approx t' \wedge discr\ c')$
$\vee$
$(\exists\ s'.\ (c,s) \rightarrow t\ s' \wedge s' \approx t'))$
$\langle proof \rangle$

lemma *matchC-MC-def2*:
matchC-MC theta d c=
$(\forall\ s\ t\ d'\ t'.$
$s \approx t \wedge (d,t) \rightarrow c\ (d',t')$

$\longrightarrow$
$(\exists\ c'\ s'.\ (c,s) \to{*}c\ (c',s') \wedge s' \approx t' \wedge (d',c') \in \mathit{theta}))$
⟨*proof*⟩

lemma *matchC-TMC-def2*:
matchC-TMC theta d c=
$(\forall\ s\ t\ d'\ t'.$
$\mathit{mustT}\ c\ s \wedge \mathit{mustT}\ d\ t \wedge s \approx t \wedge (d,t) \to c\ (d',t')$
$\longrightarrow$
$(\exists\ c'\ s'.\ (c,s) \to{*}c\ (c',s') \wedge s' \approx t' \wedge (d',c') \in \mathit{theta}))$
⟨*proof*⟩

lemma *matchC-M-def2*:
matchC-M theta d c =
$(\forall\ s\ t\ d'\ t'.$
$s \approx t \wedge (d,t) \to c\ (d',t')$
$\longrightarrow$
$(\exists\ c'\ s'.\ (c,s) \to{*}c\ (c',s') \wedge s' \approx t' \wedge (d',c') \in \mathit{theta})$
$\vee$
$(\exists\ s'.\ (c,s) \to{*}t\ s' \wedge s' \approx t' \wedge \mathit{discr}\ d'))$
⟨*proof*⟩

lemma *matchT-MT-def2*:
matchT-MT d c =
$(\forall\ s\ t\ t'.$
$s \approx t \wedge (d,t) \to t\ t'$
$\longrightarrow$
$(\exists\ s'.\ (c,s) \to{*}t\ s' \wedge s' \approx t'))$
⟨*proof*⟩

lemma *matchT-TMT-def2*:
matchT-TMT d c =
$(\forall\ s\ t\ t'.$
$\mathit{mustT}\ c\ s \wedge \mathit{mustT}\ d\ t \wedge s \approx t \wedge (d,t) \to t\ t'$
$\longrightarrow$
$(\exists\ s'.\ (c,s) \to{*}t\ s' \wedge s' \approx t'))$
⟨*proof*⟩

lemma *matchT-M-def2*:
matchT-M d c =
$(\forall\ s\ t\ t'.$
$s \approx t \wedge (d,t) \to t\ t'$
$\longrightarrow$
$(\exists\ c'\ s'.\ (c,s) \to{*}c\ (c',s') \wedge s' \approx t' \wedge \mathit{discr}\ c')$
$\vee$
$(\exists\ s'.\ (c,s) \to{*}t\ s' \wedge s' \approx t'))$
⟨*proof*⟩

Retracts:

definition *Sretr* **where**
Sretr theta $\equiv$
$\{(c,d).$
$\quad$ *matchC-C theta c d* $\wedge$
$\quad$ *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** ≈ 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
⟨*proof*⟩

lemma *Sbis-prefix*:
$Sbis \subseteq Sretr\ Sbis$
⟨*proof*⟩

lemma *Sbis-sym*: *sym Sbis*
⟨*proof*⟩

lemma *Sbis-Sym*: $c \approx s\ d \Longrightarrow d \approx s\ c$
⟨*proof*⟩

lemma *Sbis-converse*:
$((c,d) \in theta\hat{\ }{-1} \cup Sbis) = ((d,c) \in theta \cup Sbis)$
⟨*proof*⟩

lemma
Sbis-matchC-C: $\bigwedge s\ t.\ c \approx s\ d \Longrightarrow matchC\text{-}C\ Sbis\ c\ d$
and
Sbis-matchT-T: $\bigwedge c\ d.\ c \approx s\ d \Longrightarrow matchT\text{-}T\ c\ d$
⟨*proof*⟩

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

lemma
Sbis-matchC-C-rev: $\bigwedge$ *s t*. *s* $\approx$*s t* $\Longrightarrow$ *matchC-C Sbis t s*
and
Sbis-matchT-T-rev: $\bigwedge$ *s t*. *s* $\approx$*s t* $\Longrightarrow$ *matchT-T t s*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *Sbis-raw-coind*:
assumes *sym theta* **and** *theta* $\subseteq$ *Sretr theta*
shows *theta* $\subseteq$ *Sbis*
⟨*proof*⟩

lemma *Sbis-coind2*:
assumes *theta* $\subseteq$ *Sretr* (*theta* $\cup$ *Sbis*) **and**
theta ^−*1* $\subseteq$ *Sretr* ((*theta* ^−*1*) $\cup$ *Sbis*)
shows *theta* $\subseteq$ *Sbis*
⟨*proof*⟩

lemma *Sbis-raw-coind2*:
assumes *theta* $\subseteq$ *Sretr theta* **and**
theta ^−*1* $\subseteq$ *Sretr* (*theta* ^−*1*)
shows *theta* $\subseteq$ *Sbis*
⟨*proof*⟩

lemma *ZObis-prefix*:
ZObis $\subseteq$ *ZOretr ZObis*
⟨*proof*⟩

lemma *ZObis-sym*: *sym ZObis*
⟨*proof*⟩

lemma *ZObis-converse*:
((*c*,*d*) $\in$ *theta*^−*1* $\cup$ *ZObis*) = ((*d*,*c*) $\in$ *theta* $\cup$ *ZObis*)
⟨*proof*⟩

lemma *ZObis-Sym*: *s* $\approx$*01 t* $\Longrightarrow$ *t* $\approx$*01 s*
⟨*proof*⟩

lemma

ZObis-matchC-ZO: $\bigwedge$ *s t. s ≈01 t* $\Longrightarrow$ *matchC-ZO ZObis s t*
and
ZObis-matchT-ZO: $\bigwedge$ *s t. s ≈01 t* $\Longrightarrow$ *matchT-ZO s t*
⟨*proof*⟩

lemmas *ZObis-step* = *ZObis-matchC-ZO ZObis-matchT-ZO*

lemma
ZObis-matchC-ZO-rev: $\bigwedge$ *s t. s ≈01 t* $\Longrightarrow$ *matchC-ZO ZObis t s*
and
ZObis-matchT-ZO-rev: $\bigwedge$ *s t. s ≈01 t* $\Longrightarrow$ *matchT-ZO t s*
⟨*proof*⟩

lemmas *ZObis-step-rev* = *ZObis-matchC-ZO-rev ZObis-matchT-ZO-rev*

lemma *ZObis-coind*:
assumes *sym theta* **and** *theta* ⊆ *ZOretr* (*theta* ∪ *ZObis*)
shows *theta* ⊆ *ZObis*
⟨*proof*⟩

lemma *ZObis-raw-coind*:
assumes *sym theta* **and** *theta* ⊆ *ZOretr theta*
shows *theta* ⊆ *ZObis*
⟨*proof*⟩

lemma *ZObis-coind2*:
assumes *theta* ⊆ *ZOretr* (*theta* ∪ *ZObis*) **and**
theta ^−*1* ⊆ *ZOretr* ((*theta* ^−*1*) ∪ *ZObis*)
shows *theta* ⊆ *ZObis*
⟨*proof*⟩

lemma *ZObis-raw-coind2*:
assumes *theta* ⊆ *ZOretr theta* **and**
theta ^−*1* ⊆ *ZOretr* (*theta* ^−*1*)
shows *theta* ⊆ *ZObis*
⟨*proof*⟩

lemma *ZObisT-prefix*:
ZObisT ⊆ *ZOretrT ZObisT*
⟨*proof*⟩

lemma *ZObisT-sym*: *sym ZObisT*
⟨*proof*⟩

lemma *ZObisT-Sym*: *s ≈01T t* $\Longrightarrow$ *t ≈01T s*
⟨*proof*⟩

lemma *ZObisT-converse*:

$((c,d) \in theta\hat{}{-1} \cup ZObisT) = ((d,c) \in theta \cup ZObisT)$
⟨proof⟩

lemma
ZObisT-matchC-ZOC: $\bigwedge$ *s t*. $s \approx 01T\ t \Longrightarrow$ *matchC-ZOC ZObisT s t*
and
ZObisT-matchT-T: $\bigwedge$ *s t*. $s \approx 01T\ t \Longrightarrow$ *matchT-T s t*
⟨proof⟩

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

lemma
ZObisT-matchC-ZOC-rev: $\bigwedge$ *s t*. $s \approx 01T\ t \Longrightarrow$ *matchC-ZOC ZObisT t s*
and
ZObisT-matchT-T-rev: $\bigwedge$ *s t*. $s \approx 01T\ t \Longrightarrow$ *matchT-T t s*
⟨proof⟩

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

lemma *ZObisT-coind*:
assumes *sym theta* **and** $theta \subseteq ZOretrT\ (theta \cup ZObisT)$
shows $theta \subseteq ZObisT$
⟨proof⟩

lemma *ZObisT-raw-coind*:
assumes *sym theta* **and** $theta \subseteq ZOretrT\ theta$
shows $theta \subseteq ZObisT$
⟨proof⟩

lemma *ZObisT-coind2*:
assumes $theta \subseteq ZOretrT\ (theta \cup ZObisT)$ **and**
$theta\ \hat{}{-1} \subseteq ZOretrT\ ((theta\ \hat{}{-1}) \cup ZObisT)$
shows $theta \subseteq ZObisT$
⟨proof⟩

lemma *ZObisT-raw-coind2*:
assumes $theta \subseteq ZOretrT\ theta$ **and**
$theta\ \hat{}{-1} \subseteq ZOretrT\ (theta\ \hat{}{-1})$
shows $theta \subseteq ZObisT$
⟨proof⟩

lemma *Wbis-prefix*:
$Wbis \subseteq Wretr\ Wbis$
⟨proof⟩

lemma *Wbis-sym*: *sym Wbis*
⟨proof⟩

lemma *Wbis-converse*:
$((c,d) \in theta\hat{\ }{-1} \cup Wbis) = ((d,c) \in theta \cup Wbis)$
$\langle proof \rangle$

lemma *Wbis-Sym*: $c \approx w\ d \Longrightarrow d \approx w\ c$
$\langle proof \rangle$

lemma
Wbis-matchC-M: $\bigwedge c\ d.\ c \approx w\ d \Longrightarrow matchC\text{-}M\ Wbis\ c\ d$
and
Wbis-matchT-M: $\bigwedge c\ d.\ c \approx w\ d \Longrightarrow matchT\text{-}M\ c\ d$
$\langle proof \rangle$

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

lemma
Wbis-matchC-M-rev: $\bigwedge s\ t.\ s \approx w\ t \Longrightarrow matchC\text{-}M\ Wbis\ t\ s$
and
Wbis-matchT-M-rev: $\bigwedge s\ t.\ s \approx w\ t \Longrightarrow matchT\text{-}M\ t\ s$
$\langle proof \rangle$

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

lemma *Wbis-coind*:
assumes *sym theta* **and** $theta \subseteq Wretr\ (theta \cup Wbis)$
shows $theta \subseteq Wbis$
$\langle proof \rangle$

lemma *Wbis-raw-coind*:
assumes *sym theta* **and** $theta \subseteq Wretr\ theta$
shows $theta \subseteq Wbis$
$\langle proof \rangle$

lemma *Wbis-coind2*:
assumes $theta \subseteq Wretr\ (theta \cup Wbis)$ **and**
$theta\ \hat{\ }{-1} \subseteq Wretr\ ((theta\ \hat{\ }{-1}) \cup Wbis)$
shows $theta \subseteq Wbis$
$\langle proof \rangle$

lemma *Wbis-raw-coind2*:
assumes $theta \subseteq Wretr\ theta$ **and**
$theta\ \hat{\ }{-1} \subseteq Wretr\ (theta\ \hat{\ }{-1})$
shows $theta \subseteq Wbis$
$\langle proof \rangle$

lemma *WbisT-prefix*:
$WbisT \subseteq WretrT\ WbisT$
$\langle proof \rangle$

lemma *WbisT-sym*: *sym WbisT*
⟨*proof*⟩

lemma *WbisT-Sym*: $c \approx wT\ d \Longrightarrow d \approx wT\ c$
⟨*proof*⟩

lemma *WbisT-converse*:
$((c,d) \in theta\hat{\ }{-1} \cup WbisT) = ((d,c) \in theta \cup WbisT)$
⟨*proof*⟩

lemma
WbisT-matchC-MC: $\bigwedge\ c\ d.\ c \approx wT\ d \Longrightarrow$ *matchC-MC WbisT c d*
and
WbisT-matchT-MT: $\bigwedge\ c\ d.\ c \approx wT\ d \Longrightarrow$ *matchT-MT c d*
⟨*proof*⟩

lemmas *WbisT-step* = *WbisT-matchC-MC WbisT-matchT-MT*

lemma
WbisT-matchC-MC-rev: $\bigwedge\ s\ t.\ s \approx wT\ t \Longrightarrow$ *matchC-MC WbisT t s*
and
WbisT-matchT-MT-rev: $\bigwedge\ s\ t.\ s \approx wT\ t \Longrightarrow$ *matchT-MT t s*
⟨*proof*⟩

lemmas *WbisT-step-rev* = *WbisT-matchC-MC-rev WbisT-matchT-MT-rev*

lemma *WbisT-coind*:
assumes *sym theta* **and** $theta \subseteq WretrT\ (theta \cup WbisT)$
shows $theta \subseteq WbisT$
⟨*proof*⟩

lemma *WbisT-raw-coind*:
assumes *sym theta* **and** $theta \subseteq WretrT\ theta$
shows $theta \subseteq WbisT$
⟨*proof*⟩

lemma *WbisT-coind2*:
assumes $theta \subseteq WretrT\ (theta \cup WbisT)$ **and**
$theta\ \hat{\ }{-1} \subseteq WretrT\ ((theta\ \hat{\ }{-1}) \cup WbisT)$
shows $theta \subseteq WbisT$
⟨*proof*⟩

lemma *WbisT-raw-coind2*:
assumes $theta \subseteq WretrT\ theta$ **and**
$theta\ \hat{\ }{-1} \subseteq WretrT\ (theta\ \hat{\ }{-1})$
shows $theta \subseteq WbisT$
⟨*proof*⟩

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

lemma *BisT-prefix*:
BisT ⊆ *RetrT BisT*
⟨*proof*⟩

lemma *BisT-sym*: *sym BisT*
⟨*proof*⟩

lemma *BisT-Sym*: $c \approx T\ d \Longrightarrow d \approx T\ c$
⟨*proof*⟩

lemma *BisT-converse*:
$((c,d) \in theta\hat{}{-1} \cup BisT) = ((d,c) \in theta \cup BisT)$
⟨*proof*⟩

lemma
BisT-matchC-TMC: $\bigwedge\ c\ d.\ c \approx T\ d \Longrightarrow$ *matchC-TMC BisT c d*
and
BisT-matchT-TMT: $\bigwedge\ c\ d.\ c \approx T\ d \Longrightarrow$ *matchT-TMT c d*
⟨*proof*⟩

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

lemma
BisT-matchC-TMC-rev: $\bigwedge\ c\ d.\ c \approx T\ d \Longrightarrow$ *matchC-TMC BisT d c*
and
BisT-matchT-TMT-rev: $\bigwedge\ c\ d.\ c \approx T\ d \Longrightarrow$ *matchT-TMT d c*
⟨*proof*⟩

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

lemma *BisT-coind*:
assumes *sym theta* **and** *theta* ⊆ *RetrT* (*theta* ∪ *BisT*)
shows *theta* ⊆ *BisT*
⟨*proof*⟩

lemma *BisT-raw-coind*:
assumes *sym theta* **and** *theta* ⊆ *RetrT theta*

shows *theta* ⊆ *BisT*
⟨*proof*⟩

lemma *BisT-coind2*:
assumes *theta* ⊆ *RetrT* (*theta* ∪ *BisT*) **and**
theta ˆ−*1* ⊆ *RetrT* ((*theta* ˆ−*1*) ∪ *BisT*)
shows *theta* ⊆ *BisT*
⟨*proof*⟩

lemma *BisT-raw-coind2*:
assumes *theta* ⊆ *RetrT theta* **and**
theta ˆ−*1* ⊆ *RetrT* (*theta* ˆ−*1*)
shows *theta* ⊆ *BisT*
⟨*proof*⟩

Inclusions between bisimilarities:

lemma *match-imp*[*simp*]:
⋀ *theta c1 c2*. *matchC-C theta c1 c2* ⟹ *matchC-ZOC theta c1 c2*

⋀ *theta c1 c2*. *matchC-ZOC theta c1 c2* ⟹ *matchC-ZO theta c1 c2*

⋀ *theta c1 c2*. *matchC-ZOC theta c1 c2* ⟹ *matchC-MC theta c1 c2*

⋀ *theta c1 c2*. *matchC-ZO theta c1 c2* ⟹ *matchC-M theta c1 c2*

⋀ *theta c1 c2*. *matchC-MC theta c1 c2* ⟹ *matchC-M theta c1 c2*

⋀ *c1 c2*. *matchT-T c1 c2* ⟹ *matchT-ZO c1 c2*

⋀ *c1 c2*. *matchT-T c1 c2* ⟹ *matchT-MT c1 c2*

⋀ *c1 c2*. *matchT-ZO c1 c2* ⟹ *matchT-M c1 c2*

⋀ *c1 c2*. *matchT-MT c1 c2* ⟹ *matchT-M c1 c2*

⋀ *theta c1 c2*. *matchC-MC theta c1 c2* ⟹ *matchC-TMC theta c1 c2*

⋀ *theta c1 c2*. *matchT-MT c1 c2* ⟹ *matchT-TMT c1 c2*
⟨*proof*⟩

lemma *Retr-incl*:
⋀*theta*. *Sretr theta* ⊆ *ZOretrT theta*

⋀*theta*. *ZOretrT theta* ⊆ *ZOretr theta*

⋀*theta*. *ZOretrT theta* ⊆ *WretrT theta*

⋀*theta*. *ZOretr theta* ⊆ *Wretr theta*

$\bigwedge$*theta. WretrT theta* $\subseteq$ *Wretr theta*

$\bigwedge$*theta. WretrT theta* $\subseteq$ *RetrT theta*
⟨*proof*⟩

lemma *bis-incl*:
Sbis $\subseteq$ *ZObisT*

ZObisT $\subseteq$ *ZObis*

ZObisT $\subseteq$ *WbisT*

ZObis $\subseteq$ *Wbis*

WbisT $\subseteq$ *Wbis*

WbisT $\subseteq$ *BisT*
⟨*proof*⟩

lemma *bis-imp*[*simp*]:
$\bigwedge$ *c1 c2. c1* $\approx$*s c2* $\Longrightarrow$ *c1* $\approx$*01T c2*

$\bigwedge$ *c1 c2. c1* $\approx$*01T c2* $\Longrightarrow$ *c1* $\approx$*01 c2*

$\bigwedge$ *c1 c2. c1* $\approx$*01T c2* $\Longrightarrow$ *c1* $\approx$*wT c2*

$\bigwedge$ *c1 c2. c1* $\approx$*01 c2* $\Longrightarrow$ *c1* $\approx$*w c2*

$\bigwedge$ *c1 c2. c1* $\approx$*wT c2* $\Longrightarrow$ *c1* $\approx$*w c2*

$\bigwedge$ *c1 c2. c1* $\approx$*wT c2* $\Longrightarrow$ *c1* $\approx$*T c2*
⟨*proof*⟩

Self-isomorphism implies strong bisimilarity:

lemma *siso-Sbis*[*simp*]:
assumes *siso c*
shows *c* $\approx$*s c*
⟨*proof*⟩

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

lemma *siso0-Sbis*[*simp*]:
assumes *siso0 c*
shows *c* $\approx$*T c*
⟨*proof*⟩

end

end

5 Compasitionality 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*⟩

theorem *discr-If* [*simp*]:
assumes *discr c1* **and** *discr c2*
shows *discr* (*If tst c1 c2*)
⟨*proof*⟩

theorem *discr-Seq*[*simp*]:
assumes ∗: *discr c1* **and** ∗∗: *discr c2*
shows *discr* (*c1* ;; *c2*)
⟨*proof*⟩

theorem *discr-While*[*simp*]:
assumes *discr c*
shows *discr* (*While tst c*)
⟨*proof*⟩

theorem *discr-Par*[*simp*]:
assumes ∗: *discr c1* **and** ∗∗: *discr c2*
shows *discr* (*Par c1 c2*)
⟨*proof*⟩

5.2 Discreetness versus language constructs:

theorem *discr0-Atm*[*simp*]:
discr0 (*Atm atm*) = *presAtm atm*
⟨*proof*⟩

theorem *discr0-If* [*simp*]:
assumes *discr0 c1* **and** *discr0 c2*
shows *discr0* (*If tst c1 c2*)

⟨*proof*⟩

theorem *discr0-Seq*[*simp*]:
assumes ∗: *discr0 c1* **and** ∗∗: *discr0 c2*
shows *discr0* (*c1* ;; *c2*)
⟨*proof*⟩

theorem *discr0-While*[*simp*]:
assumes *discr0 c*
shows *discr0* (*While tst c*)
⟨*proof*⟩

theorem *discr0-Par*[*simp*]:
assumes ∗: *discr0 c1* **and** ∗∗: *discr0 c2*
shows *discr0* (*Par c1 c2*)
⟨*proof*⟩

5.3 Self-Isomorphism versus language constructs:

theorem *siso-Atm*[*simp*]:
siso (*Atm atm*) = *compatAtm atm*
⟨*proof*⟩

theorem *siso-If*[*simp*]:
assumes *compatTst tst* **and** *siso c1* **and** *siso c2*
shows *siso* (*If tst c1 c2*)
⟨*proof*⟩

theorem *siso-Seq*[*simp*]:
assumes ∗: *siso c1* **and** ∗∗: *siso c2*
shows *siso* (*c1* ;; *c2*)
⟨*proof*⟩

theorem *siso-While*[*simp*]:
assumes *compatTst tst* **and** *siso c*
shows *siso* (*While tst c*)
⟨*proof*⟩

theorem *siso-Par*[*simp*]:
assumes ∗: *siso c1* **and** ∗∗: *siso c2*
shows *siso* (*Par c1 c2*)
⟨*proof*⟩

5.4 Self-Isomorphism versus language constructs:

theorem *siso0-Atm*[*simp*]:
siso0 (*Atm atm*) = *compatAtm atm*
⟨*proof*⟩

theorem *siso0-If*[*simp*]:

assumes *compatTst tst* **and** *siso0 c1* **and** *siso0 c2*
shows *siso0* (*If tst c1 c2*)
⟨*proof*⟩

theorem *siso0-Seq*[*simp*]:
assumes ∗: *siso0 c1* **and** ∗∗: *siso0 c2*
shows *siso0* (*c1* ;; *c2*)
⟨*proof*⟩

theorem *siso0-While*[*simp*]:
assumes *compatTst tst* **and** *siso0 c*
shows *siso0* (*While tst c*)
⟨*proof*⟩

theorem *siso0-Par*[*simp*]:
assumes ∗: *siso0 c1* **and** ∗∗: *siso0 c2*
shows *siso0* (*Par c1 c2*)
⟨*proof*⟩

5.5 Strong bisimilarity versus language constructs

Atomic commands:

definition *thetaAtm* **where**
thetaAtm atm ≡ {(*Atm atm*, *Atm atm*)}

lemma *thetaAtm-sym*:
sym (*thetaAtm atm*)
⟨*proof*⟩

lemma *thetaAtm-Sretr*:
assumes *compatAtm atm*
shows *thetaAtm atm* ⊆ *Sretr* (*thetaAtm atm*)
⟨*proof*⟩

lemma *thetaAtm-Sbis*:
assumes *compatAtm atm*
shows *thetaAtm atm* ⊆ *Sbis*
⟨*proof*⟩

theorem *Atm-Sbis*[*simp*]:
assumes *compatAtm atm*
shows *Atm atm* ≈*s Atm atm*
⟨*proof*⟩

Sequential composition:

definition *thetaSeq* **where**
thetaSeq ≡
{(*c1* ;; *c2*, *d1* ;; *d2*) | *c1 c2 d1 d2*. *c1* ≈*s d1* ∧ *c2* ≈*s d2*}

lemma *thetaSeq-sym*:
sym thetaSeq
⟨*proof*⟩

lemma *thetaSeq-Sretr*:
thetaSeq ⊆ *Sretr* (*thetaSeq Un Sbis*)
⟨*proof*⟩

lemma *thetaSeq-Sbis*:
thetaSeq ⊆ *Sbis*
⟨*proof*⟩

theorem *Seq-Sbis*[*simp*]:
assumes *c1* ≈*s d1* **and** *c2* ≈*s d2*
shows *c1* ;; *c2* ≈*s d1* ;; *d2*
⟨*proof*⟩

Conditional:

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

lemma *thetaIf-sym*:
sym thetaIf
⟨*proof*⟩

lemma *thetaIf-Sretr*:
thetaIf ⊆ *Sretr* (*thetaIf Un Sbis*)
⟨*proof*⟩

lemma *thetaIf-Sbis*:
thetaIf ⊆ *Sbis*
⟨*proof*⟩

theorem *If-Sbis*[*simp*]:
assumes *compatTst tst* **and** *c1* ≈*s d1* **and** *c2* ≈*s d2*
shows *If tst c1 c2* ≈*s If tst d1 d2*
⟨*proof*⟩

While loop:

definition *thetaWhile* **where**
thetaWhile ≡
{(*While tst c*, *While tst d*) | *tst c d*. *compatTst tst* ∧ *c* ≈*s d*} *Un*
{(*c1* ;; (*While tst c*), *d1* ;; (*While tst d*)) | *tst c1 d1 c d*. *compatTst tst* ∧ *c1* ≈*s d1* ∧ *c* ≈*s d*}

lemma *thetaWhile-sym*:
sym thetaWhile

⟨*proof*⟩

lemma *thetaWhile-Sretr*:
thetaWhile ⊆ *Sretr* (*thetaWhile Un Sbis*)
⟨*proof*⟩

lemma *thetaWhile-Sbis*:
thetaWhile ⊆ *Sbis*
⟨*proof*⟩

theorem *While-Sbis*[*simp*]:
assumes *compatTst tst* **and** *c* ≈*s d*
shows *While tst c* ≈*s While tst d*
⟨*proof*⟩

Parallel composition:

definition *thetaPar* **where**
thetaPar ≡
{(*Par c1 c2*, *Par d1 d2*) | *c1 c2 d1 d2*. *c1* ≈*s d1* ∧ *c2* ≈*s d2*}

lemma *thetaPar-sym*:
sym thetaPar
⟨*proof*⟩

lemma *thetaPar-Sretr*:
thetaPar ⊆ *Sretr* (*thetaPar Un Sbis*)
⟨*proof*⟩

lemma *thetaPar-Sbis*:
thetaPar ⊆ *Sbis*
⟨*proof*⟩

theorem *Par-Sbis*[*simp*]:
assumes *c1* ≈*s d1* **and** *c2* ≈*s d2*
shows *Par c1 c2* ≈*s Par d1 d2*
⟨*proof*⟩

5.5.1 01T-bisimilarity versus language constructs

Atomic commands:

theorem *Atm-ZObisT*:
assumes *compatAtm atm*
shows *Atm atm* ≈*01T Atm atm*
⟨*proof*⟩

Sequential composition:

definition *thetaSeqZOT* **where**
thetaSeqZOT ≡

$\{(c1 ;; c2, d1 ;; d2) \mid c1\ c2\ d1\ d2.\ c1 \approx 01T\ d1 \wedge c2 \approx 01T\ d2\}$

lemma *thetaSeqZOT-sym*:
sym thetaSeqZOT
⟨*proof*⟩

lemma *thetaSeqZOT-ZOretrT*:
$thetaSeqZOT \subseteq ZOretrT\ (thetaSeqZOT\ Un\ ZObisT)$
⟨*proof*⟩

lemma *thetaSeqZOT-ZObisT*:
$thetaSeqZOT \subseteq ZObisT$
⟨*proof*⟩

theorem *Seq-ZObisT*[*simp*]:
assumes $c1 \approx 01T\ d1$ **and** $c2 \approx 01T\ d2$
shows $c1 ;; c2 \approx 01T\ d1 ;; d2$
⟨*proof*⟩

Conditional:

definition *thetaIfZOT* **where**
$thetaIfZOT \equiv$
$\{(If\ tst\ c1\ c2, If\ tst\ d1\ d2) \mid tst\ c1\ c2\ d1\ d2.\ compatTst\ tst \wedge c1 \approx 01T\ d1 \wedge c2 \approx 01T\ d2\}$

lemma *thetaIfZOT-sym*:
sym thetaIfZOT
⟨*proof*⟩

lemma *thetaIfZOT-ZOretrT*:
$thetaIfZOT \subseteq ZOretrT\ (thetaIfZOT\ Un\ ZObisT)$
⟨*proof*⟩

lemma *thetaIfZOT-ZObisT*:
$thetaIfZOT \subseteq ZObisT$
⟨*proof*⟩

theorem *If-ZObisT*[*simp*]:
assumes *compatTst tst* **and** $c1 \approx 01T\ d1$ **and** $c2 \approx 01T\ d2$
shows $If\ tst\ c1\ c2 \approx 01T\ If\ tst\ d1\ d2$
⟨*proof*⟩

While loop:

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

lemma *thetaWhileZOT-sym*:
sym thetaWhileZOT
⟨*proof*⟩

lemma *thetaWhileZOT-ZOretrT*:
thetaWhileZOT ⊆ *ZOretrT* (*thetaWhileZOT Un ZObisT*)
⟨*proof*⟩

lemma *thetaWhileZOT-ZObisT*:
thetaWhileZOT ⊆ *ZObisT*
⟨*proof*⟩

theorem *While-ZObisT*[*simp*]:
assumes *compatTst tst* **and** $c \approx$*01T* d
shows *While tst c* ≈*01T* *While tst d*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *thetaParZOT-ZOretrT*:
thetaParZOT ⊆ *ZOretrT* (*thetaParZOT Un ZObisT*)
⟨*proof*⟩

lemma *thetaParZOT-ZObisT*:
thetaParZOT ⊆ *ZObisT*
⟨*proof*⟩

theorem *Par-ZObisT*[*simp*]:
assumes *c1* ≈*01T* *d1* **and** *c2* ≈*01T* *d2*
shows *Par c1 c2* ≈*01T* *Par d1 d2*
⟨*proof*⟩

5.5.2 01-bisimilarity versus language constructs

Discreetness:

theorem *discr-ZObis*[*simp*]:
assumes ∗: *discr c* **and** ∗∗: *discr d*
shows *c* ≈*01* *d*
⟨*proof*⟩

Atomic commands:

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

assumes *compatAtm atm*
shows *Atm atm* $\approx$*01 Atm atm*
⟨*proof*⟩

Sequential composition:

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

lemma *thetaSeqZO-sym*:
sym thetaSeqZO
⟨*proof*⟩

lemma *thetaSeqZO-ZOretr*:
thetaSeqZO $\subseteq$ *ZOretr* (*thetaSeqZO Un ZObis*)
⟨*proof*⟩

lemma *thetaSeqZO-ZObis*:
thetaSeqZO $\subseteq$ *ZObis*
⟨*proof*⟩

theorem *Seq-ZObisT-ZObis*[*simp*]:
assumes *c1* $\approx$*01T d1* **and** *c2* $\approx$*01 d2*
shows *c1* ;; *c2* $\approx$*01 d1* ;; *d2*
⟨*proof*⟩

theorem *Seq-siso-ZObis*[*simp*]:
assumes *siso e* **and** *c2* $\approx$*01 d2*
shows *e* ;; *c2* $\approx$*01 e* ;; *d2*
⟨*proof*⟩

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

lemma *thetaSeqZOD-sym*:
sym thetaSeqZOD
⟨*proof*⟩

lemma *thetaSeqZOD-ZOretr*:
thetaSeqZOD $\subseteq$ *ZOretr* (*thetaSeqZOD Un ZObis*)
⟨*proof*⟩

lemma *thetaSeqZOD-ZObis*:
thetaSeqZOD $\subseteq$ *ZObis*
⟨*proof*⟩

theorem *Seq-ZObis-discr*[*simp*]:
assumes $c1 \approx 01\ d1$ **and** *discr c2* **and** *discr d2*
shows $c1\ ;;\ c2 \approx 01\ d1\ ;;\ d2$
⟨*proof*⟩

Conditional:

definition *thetaIfZO* **where**
thetaIfZO $\equiv$
$\{(If\ tst\ c1\ c2,\ If\ tst\ d1\ d2) \mid tst\ c1\ c2\ d1\ d2.\ compatTst\ tst \wedge c1 \approx 01\ d1 \wedge c2 \approx 01\ d2\}$

lemma *thetaIfZO-sym*:
sym thetaIfZO
⟨*proof*⟩

lemma *thetaIfZO-ZOretr*:
$thetaIfZO \subseteq ZOretr\ (thetaIfZO\ Un\ ZObis)$
⟨*proof*⟩

lemma *thetaIfZO-ZObis*:
$thetaIfZO \subseteq ZObis$
⟨*proof*⟩

theorem *If-ZObis*[*simp*]:
assumes *compatTst tst* **and** $c1 \approx 01\ d1$ **and** $c2 \approx 01\ d2$
shows $If\ tst\ c1\ c2 \approx 01\ If\ tst\ d1\ d2$
⟨*proof*⟩

While loop:

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

Parallel composition:

definition *thetaParZOL1* **where**
thetaParZOL1 $\equiv$
$\{(Par\ c1\ c2,\ d) \mid c1\ c2\ d.\ c1 \approx 01\ d \wedge discr\ c2\}$

lemma *thetaParZOL1-ZOretr*:
$thetaParZOL1 \subseteq ZOretr\ (thetaParZOL1\ Un\ ZObis)$
⟨*proof*⟩

lemma *thetaParZOL1-converse-ZOretr*:
$thetaParZOL1\ \hat{}{-1} \subseteq ZOretr\ (thetaParZOL1\ \hat{}{-1}\ Un\ ZObis)$
⟨*proof*⟩

lemma *thetaParZOL1-ZObis*:
$thetaParZOL1 \subseteq ZObis$
⟨*proof*⟩

theorem *Par-ZObis-discrL1* [*simp*]:
assumes $c1 \approx 01\ d$ **and** *discr c2*
shows $Par\ c1\ c2 \approx 01\ d$
⟨*proof*⟩

theorem *Par-ZObis-discrR1* [*simp*]:
assumes $c \approx 01\ d1$ **and** *discr d2*
shows $c \approx 01\ Par\ d1\ d2$
⟨*proof*⟩

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

lemma *thetaParZOL2-ZOretr*:
$thetaParZOL2 \subseteq ZOretr\ (thetaParZOL2\ Un\ ZObis)$
⟨*proof*⟩

lemma *thetaParZOL2-converse-ZOretr*:
$thetaParZOL2\ \hat{}-1 \subseteq ZOretr\ (thetaParZOL2\ \hat{}-1\ Un\ ZObis)$
⟨*proof*⟩

lemma *thetaParZOL2-ZObis*:
$thetaParZOL2 \subseteq ZObis$
⟨*proof*⟩

theorem *Par-ZObis-discrL2* [*simp*]:
assumes $c2 \approx 01\ d$ **and** *discr c1*
shows $Par\ c1\ c2 \approx 01\ d$
⟨*proof*⟩

theorem *Par-ZObis-discrR2* [*simp*]:
assumes $c \approx 01\ d2$ **and** *discr d1*
shows $c \approx 01\ Par\ d1\ d2$
⟨*proof*⟩

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
⟨*proof*⟩

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

lemma *thetaParZO-ZObis*:
thetaParZO $\subseteq$ *ZObis*
⟨*proof*⟩

theorem *Par-ZObis*[*simp*]:
assumes *c1* $\approx$*01 d1* **and** *c2* $\approx$*01 d2*
shows *Par c1 c2* $\approx$*01 Par d1 d2*
⟨*proof*⟩

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*⟩

Atomic commands:

theorem *Atm-WbisT*:
assumes *compatAtm atm*
shows *Atm atm* $\approx$*wT Atm atm*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *thetaSeqWT-WretrT*:
thetaSeqWT $\subseteq$ *WretrT* (*thetaSeqWT Un WbisT*)
⟨*proof*⟩

lemma *thetaSeqWT-WbisT*:
thetaSeqWT $\subseteq$ *WbisT*
⟨*proof*⟩

theorem *Seq-WbisT*[*simp*]:
assumes *c1* $\approx$*wT d1* **and** *c2* $\approx$*wT d2*
shows *c1* ;; *c2* $\approx$*wT d1* ;; *d2*

$\langle proof \rangle$

Conditional:

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

lemma *thetaIfWT-sym*:
sym thetaIfWT
$\langle proof \rangle$

lemma *thetaIfWT-WretrT*:
$thetaIfWT \subseteq WretrT\ (thetaIfWT\ Un\ WbisT)$
$\langle proof \rangle$

lemma *thetaIfWT-WbisT*:
$thetaIfWT \subseteq WbisT$
$\langle proof \rangle$

theorem *If-WbisT*[*simp*]:
assumes *compatTst tst* **and** $c1 \approx wT\ d1$ **and** $c2 \approx wT\ d2$
shows $If\ tst\ c1\ c2 \approx wT\ If\ tst\ d1\ d2$
$\langle proof \rangle$

While loop:

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

lemma *thetaWhileW-sym*:
sym thetaWhileW
$\langle proof \rangle$

lemma *thetaWhileW-WretrT*:
$thetaWhileW \subseteq WretrT\ (thetaWhileW\ Un\ WbisT)$
$\langle proof \rangle$

lemma *thetaWhileW-WbisT*:
$thetaWhileW \subseteq WbisT$
$\langle proof \rangle$

theorem *While-WbisT*[*simp*]:
assumes *compatTst tst* **and** $c \approx wT\ d$
shows $While\ tst\ c \approx wT\ While\ tst\ d$
$\langle proof \rangle$

Parallel composition:

definition *thetaParWT* **where**
$thetaParWT \equiv$
$\{(Par\ c1\ c2,\ Par\ d1\ d2) \mid c1\ c2\ d1\ d2.\ c1 \approx wT\ d1 \land c2 \approx wT\ d2\}$

lemma *thetaParWT-sym*:
sym thetaParWT
⟨*proof*⟩

lemma *thetaParWT-WretrT*:
$thetaParWT \subseteq WretrT\ (thetaParWT\ Un\ WbisT)$
⟨*proof*⟩

lemma *thetaParWT-WbisT*:
$thetaParWT \subseteq WbisT$
⟨*proof*⟩

theorem *Par-WbisT*[*simp*]:
assumes $c1 \approx wT\ d1$ **and** $c2 \approx wT\ d2$
shows $Par\ c1\ c2 \approx wT\ Par\ d1\ d2$
⟨*proof*⟩

5.5.4 T-bisimilarity versus language constructs

T-Discreetness:

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

lemma *thetaFDW0-sym*:
sym thetaFDW0
⟨*proof*⟩

lemma *thetaFDW0-RetrT*:
$thetaFDW0 \subseteq RetrT\ thetaFDW0$
⟨*proof*⟩

lemma *thetaFDW0-BisT*:
$thetaFDW0 \subseteq BisT$
⟨*proof*⟩

theorem *discr0-BisT*[*simp*]:
assumes *discr0 c1* **and** *discr0 c2*
shows $c1 \approx T\ c2$
⟨*proof*⟩

Atomic commands:

theorem *Atm-BisT*:
assumes *compatAtm atm*
shows $Atm\ atm \approx T\ Atm\ atm$

⟨*proof*⟩

Sequential composition:

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

lemma *thetaSeqTT-sym*:
sym thetaSeqTT
⟨*proof*⟩

lemma *thetaSeqTT-RetrT*:
thetaSeqTT ⊆ *RetrT* (*thetaSeqTT* ∪ *BisT*)
⟨*proof*⟩

lemma *thetaSeqTT-BisT*:
thetaSeqTT ⊆ *BisT*
⟨*proof*⟩

theorem *Seq-BisT*[*simp*]:
assumes *c1* ≈*T d1* **and** *c2* ≈*T d2*
shows *c1* ;; *c2* ≈*T d1* ;; *d2*
⟨*proof*⟩

Conditional:

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

lemma *thetaIfTT-sym*:
sym thetaIfTT
⟨*proof*⟩

lemma *thetaIfTT-RetrT*:
thetaIfTT ⊆ *RetrT* (*thetaIfTT* ∪ *BisT*)
⟨*proof*⟩

lemma *thetaIfTT-BisT*:
thetaIfTT ⊆ *BisT*
⟨*proof*⟩

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*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *thetaWhileW0-RetrT*:
thetaWhileW0 ⊆ *RetrT* (*thetaWhileW0* ∪ *BisT*)
⟨*proof*⟩

lemma *thetaWhileW0-BisT*:
thetaWhileW0 ⊆ *BisT*
⟨*proof*⟩

theorem *While-BisT*[*simp*]:
assumes *compatTst tst* **and** *c* ≈*T d*
shows *While tst c* ≈*T While tst d*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *thetaParTT-RetrT*:
thetaParTT ⊆ *RetrT* (*thetaParTT* ∪ *BisT*)
⟨*proof*⟩

lemma *thetaParTT-BisT*:
thetaParTT ⊆ *BisT*
⟨*proof*⟩

theorem *Par-BisT*[*simp*]:
assumes *c1* ≈*T d1* **and** *c2* ≈*T d2*
shows *Par c1 c2* ≈*T Par d1 d2*
⟨*proof*⟩

5.5.5 W-bisimilarity versus language constructs

Atomic commands:

theorem *Atm-Wbis*[*simp*]:
assumes *compatAtm atm*

shows *Atm atm* $\approx$*w Atm atm*
⟨*proof*⟩

Discreetness:

theorem *discr-Wbis*[*simp*]:
assumes $*$: *discr c* **and** $**$: *discr d*
shows *c* $\approx$*w d*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *thetaSeqW-Wretr*:
thetaSeqW $\subseteq$ *Wretr* (*thetaSeqW* $\cup$ *Wbis*)
⟨*proof*⟩

lemma *thetaSeqW-Wbis*:
thetaSeqW $\subseteq$ *Wbis*
⟨*proof*⟩

theorem *Seq-WbisT-Wbis*[*simp*]:
assumes *c1* $\approx$*wT d1* **and** *c2* $\approx$*w d2*
shows *c1* ;; *c2* $\approx$*w d1* ;; *d2*
⟨*proof*⟩

theorem *Seq-siso-Wbis*[*simp*]:
assumes *siso e* **and** *c2* $\approx$*w d2*
shows *e* ;; *c2* $\approx$*w e* ;; *d2*
⟨*proof*⟩

definition *thetaSeqWD* **where**
thetaSeqWD $\equiv$
$\{(c1\ ;;\ c2,\ d1\ ;;\ d2) \mid c1\ c2\ d1\ d2.\ c1 \approx w\ d1 \wedge discr\ c2 \wedge discr\ d2\}$

lemma *thetaSeqWD-sym*:
sym thetaSeqWD
⟨*proof*⟩

lemma *thetaSeqWD-Wretr*:
thetaSeqWD $\subseteq$ *Wretr* (*thetaSeqWD* $\cup$ *Wbis*)
⟨*proof*⟩

lemma *thetaSeqWD-Wbis*:
thetaSeqWD $\subseteq$ *Wbis*
$\langle$*proof*$\rangle$

theorem *Seq-Wbis-discr*[*simp*]:
assumes *c1* $\approx$*w d1* **and** *discr c2* **and** *discr d2*
shows *c1* ;; *c2* $\approx$*w d1* ;; *d2*
$\langle$*proof*$\rangle$

Conditional:

definition *thetaIfW* **where**
thetaIfW $\equiv$
$\{$(*If tst c1 c2*, *If tst d1 d2*) | *tst c1 c2 d1 d2*. *compatTst tst* $\wedge$ *c1* $\approx$*w d1* $\wedge$ *c2* $\approx$*w d2*$\}$

lemma *thetaIfW-sym*:
sym thetaIfW
$\langle$*proof*$\rangle$

lemma *thetaIfW-Wretr*:
thetaIfW $\subseteq$ *Wretr* (*thetaIfW* $\cup$ *Wbis*)
$\langle$*proof*$\rangle$

lemma *thetaIfW-Wbis*:
thetaIfW $\subseteq$ *Wbis*
$\langle$*proof*$\rangle$

theorem *If-Wbis*[*simp*]:
assumes *compatTst tst* **and** *c1* $\approx$*w d1* **and** *c2* $\approx$*w d2*
shows *If tst c1 c2* $\approx$*w If tst d1 d2*
$\langle$*proof*$\rangle$

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*) | *c1 c2 d*. *c1* $\approx$*w d* $\wedge$ *discr c2*$\}$

lemma *thetaParWL1-Wretr*:
thetaParWL1 $\subseteq$ *Wretr* (*thetaParWL1* $\cup$ *Wbis*)
$\langle$*proof*$\rangle$

lemma *thetaParWL1-converse-Wretr*:
thetaParWL1 ^$-$*1* $\subseteq$ *Wretr* (*thetaParWL1* ^$-$*1* $\cup$ *Wbis*)

⟨*proof*⟩

lemma *thetaParWL1-Wbis*:
thetaParWL1 $\subseteq$ *Wbis*
⟨*proof*⟩

theorem *Par-Wbis-discrL1*[*simp*]:
assumes *c1* $\approx$*w d* **and** *discr c2*
shows *Par c1 c2* $\approx$*w d*
⟨*proof*⟩

theorem *Par-Wbis-discrR1*[*simp*]:
assumes *c* $\approx$*w d1* **and** *discr d2*
shows *c* $\approx$*w Par d1 d2*
⟨*proof*⟩

definition *thetaParWL2* **where**
thetaParWL2 $\equiv$
$\{$(*Par c1 c2*, *d*) | *c1 c2 d*. *discr c1* $\wedge$ *c2* $\approx$*w d*$\}$

lemma *thetaParWL2-Wretr*:
thetaParWL2 $\subseteq$ *Wretr* (*thetaParWL2* $\cup$ *Wbis*)
⟨*proof*⟩

lemma *thetaParWL2-converse-Wretr*:
thetaParWL2 ^−*1* $\subseteq$ *Wretr* (*thetaParWL2* ^−*1* $\cup$ *Wbis*)
⟨*proof*⟩

lemma *thetaParWL2-Wbis*:
thetaParWL2 $\subseteq$ *Wbis*
⟨*proof*⟩

theorem *Par-Wbis-discrL2*[*simp*]:
assumes *c2* $\approx$*w d* **and** *discr c1*
shows *Par c1 c2* $\approx$*w d*
⟨*proof*⟩

theorem *Par-Wbis-discrR2*[*simp*]:
assumes *c* $\approx$*w d2* **and** *discr d1*
shows *c* $\approx$*w Par d1 d2*
⟨*proof*⟩

definition *thetaParW* **where**
thetaParW $\equiv$
$\{$(*Par c1 c2*, *Par d1 d2*) | *c1 c2 d1 d2*. *c1* $\approx$*w d1* $\wedge$ *c2* $\approx$*w d2*$\}$

lemma *thetaParW-sym*:
sym thetaParW
⟨*proof*⟩

lemma *thetaParW-Wretr*:
thetaParW ⊆ *Wretr* (*thetaParW* ∪ *Wbis*)
⟨*proof*⟩

lemma *thetaParW-Wbis*:
thetaParW ⊆ *Wbis*
⟨*proof*⟩

theorem *Par-Wbis*[*simp*]:
assumes *c1* ≈*w d1* **and** *c2* ≈*w d2*
shows *Par c1 c2* ≈*w Par d1 d2*
⟨*proof*⟩

end

end
theory *Syntactic-Criteria*
  imports *Compositionality*
begin

context *PL-Indis*
begin

lemma *noWhile*[*intro*]:
  noWhile (*Atm atm*)
  noWhile c1 ⟹ *noWhile c2* ⟹ *noWhile* (*Seq c1 c2*)
  noWhile c1 ⟹ *noWhile c2* ⟹ *noWhile* (*If tst c1 c2*)
  noWhile c1 ⟹ *noWhile c2* ⟹ *noWhile* (*Par c1 c2*)
  ⟨*proof*⟩

lemma *discr*[*intro*]:
  presAtm atm ⟹ *discr* (*Atm atm*)
  discr c1 ⟹ *discr c2* ⟹ *discr* (*Seq c1 c2*)
  discr c1 ⟹ *discr c2* ⟹ *discr* (*If tst c1 c2*)
  discr c ⟹ *discr* (*While tst c*)
  discr c1 ⟹ *discr c2* ⟹ *discr* (*Par c1 c2*)
  ⟨*proof*⟩

lemma *siso*[*intro*]:
  compatAtm atm ⟹ *siso* (*Atm atm*)

siso c1 $\Longrightarrow$ *siso c2* $\Longrightarrow$ *siso* (*Seq c1 c2*)
compatTst tst $\Longrightarrow$ *siso c1* $\Longrightarrow$ *siso c2* $\Longrightarrow$ *siso* (*If tst c1 c2*)
compatTst tst $\Longrightarrow$ *siso c* $\Longrightarrow$ *siso* (*While tst c*)
siso c1 $\Longrightarrow$ *siso c2* $\Longrightarrow$ *siso* (*Par c1 c2*)
⟨*proof*⟩

lemma *Sbis*[*intro*]:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*s Atm atm*
c1 $\approx$*s c1* $\Longrightarrow$ *c2* $\approx$*s c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*s Seq c1 c2*
compatTst tst $\Longrightarrow$ *c1* $\approx$*s c1* $\Longrightarrow$ *c2* $\approx$*s c2* $\Longrightarrow$ *If tst c1 c2* $\approx$*s If tst c1 c2*
compatTst tst $\Longrightarrow$ *c* $\approx$*s c* $\Longrightarrow$ *While tst c* $\approx$*s While tst c*
c1 $\approx$*s c1* $\Longrightarrow$ *c2* $\approx$*s c2* $\Longrightarrow$ *Par c1 c2* $\approx$*s Par c1 c2*
⟨*proof*⟩

lemma *ZObisT*[*intro*]:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*01T Atm atm*
c1 $\approx$*01T c1* $\Longrightarrow$ *c2* $\approx$*01T c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*01T Seq c1 c2*
compatTst tst $\Longrightarrow$ *c1* $\approx$*01T c1* $\Longrightarrow$ *c2* $\approx$*01T c2* $\Longrightarrow$ *If tst c1 c2* $\approx$*01T If tst c1 c2*
compatTst tst $\Longrightarrow$ *c* $\approx$*01T c* $\Longrightarrow$ *While tst c* $\approx$*01T While tst c*
c1 $\approx$*01T c1* $\Longrightarrow$ *c2* $\approx$*01T c2* $\Longrightarrow$ *Par c1 c2* $\approx$*01T Par c1 c2*
⟨*proof*⟩

lemma *BisT*[*intro*]:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*T Atm atm*
c1 $\approx$*T c1* $\Longrightarrow$ *c2* $\approx$*T c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*T Seq c1 c2*
compatTst tst $\Longrightarrow$ *c1* $\approx$*T c1* $\Longrightarrow$ *c2* $\approx$*T c2* $\Longrightarrow$ *If tst c1 c2* $\approx$*T If tst c1 c2*
compatTst tst $\Longrightarrow$ *c* $\approx$*T c* $\Longrightarrow$ *While tst c* $\approx$*T While tst c*
c1 $\approx$*T c1* $\Longrightarrow$ *c2* $\approx$*T c2* $\Longrightarrow$ *Par c1 c2* $\approx$*T Par c1 c2*
⟨*proof*⟩

lemma *WbisT*[*intro*]:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*wT Atm atm*
c1 $\approx$*wT c1* $\Longrightarrow$ *c2* $\approx$*wT c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*wT Seq c1 c2*
compatTst tst $\Longrightarrow$ *c1* $\approx$*wT c1* $\Longrightarrow$ *c2* $\approx$*wT c2* $\Longrightarrow$ *If tst c1 c2* $\approx$*wT If tst c1 c2*
compatTst tst $\Longrightarrow$ *c* $\approx$*wT c* $\Longrightarrow$ *While tst c* $\approx$*wT While tst c*
c1 $\approx$*wT c1* $\Longrightarrow$ *c2* $\approx$*wT c2* $\Longrightarrow$ *Par c1 c2* $\approx$*wT Par c1 c2*
⟨*proof*⟩

lemma *ZObis*[*intro*]:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*01 Atm atm*
c1 $\approx$*01T c1* $\Longrightarrow$ *c2* $\approx$*01 c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*01 Seq c1 c2*
c1 $\approx$*01 c1* $\Longrightarrow$ *discr c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*01 Seq c1 c2*
compatTst tst $\Longrightarrow$ *c1* $\approx$*01 c1* $\Longrightarrow$ *c2* $\approx$*01 c2* $\Longrightarrow$ *If tst c1 c2* $\approx$*01 If tst c1 c2*
c1 $\approx$*01 c1* $\Longrightarrow$ *c2* $\approx$*01 c2* $\Longrightarrow$ *Par c1 c2* $\approx$*01 Par c1 c2*
⟨*proof*⟩

lemma *Wbis*[*intro*]:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*w Atm atm*

$c1 \approx wT\ c1 \Longrightarrow c2 \approx w\ c2 \Longrightarrow Seq\ c1\ c2 \approx w\ Seq\ c1\ c2$
$c1 \approx w\ c1 \Longrightarrow discr\ c2 \Longrightarrow Seq\ c1\ c2 \approx w\ Seq\ c1\ c2$
$compatTst\ tst \Longrightarrow c1 \approx w\ c1 \Longrightarrow c2 \approx w\ c2 \Longrightarrow If\ tst\ c1\ c2 \approx w\ If\ tst\ c1\ c2$
$c1 \approx w\ c1 \Longrightarrow c2 \approx w\ c2 \Longrightarrow Par\ c1\ c2 \approx w\ Par\ c1\ c2$
⟨*proof*⟩

lemma *discr-noWhile-WbisT*[*intro*]: $discr\ c \Longrightarrow noWhile\ c \Longrightarrow c \approx wT\ c$
⟨*proof*⟩

lemma *siso-ZObis*[*intro*]: $siso\ c \Longrightarrow c \approx 01\ c$
⟨*proof*⟩

lemma *WbisT-Wbis*[*intro*]: $c \approx wT\ c \Longrightarrow c \approx w\ c$
⟨*proof*⟩

lemma *ZObis-Wbis*[*intro*]: $c \approx 01\ c \Longrightarrow c \approx w\ c$
⟨*proof*⟩

lemma *discr-BisT*[*intro*]: $discr\ c \Longrightarrow c \approx T\ c$
⟨*proof*⟩

lemma *WbisT-BisT*[*intro*]: $c \approx wT\ c \Longrightarrow c \approx T\ c$
⟨*proof*⟩

lemma *ZObisT-ZObis*[*intro*]: $c \approx 01T\ c \Longrightarrow c \approx 01\ c$
⟨*proof*⟩

lemma *siso-ZObisT*[*intro*]: $siso\ c \Longrightarrow c \approx 01T\ c$
⟨*proof*⟩

primrec *SC-discr* **where**
SC-discr (*Atm atm*) $\longleftrightarrow$ *presAtm atm*
| *SC-discr* (*Seq c1 c2*) $\longleftrightarrow$ *SC-discr c1* $\wedge$ *SC-discr c2*
| *SC-discr* (*If tst c1 c2*) $\longleftrightarrow$ *SC-discr c1* $\wedge$ *SC-discr c2*
| *SC-discr* (*While tst c*) $\longleftrightarrow$ *SC-discr c*
| *SC-discr* (*Par c1 c2*) $\longleftrightarrow$ *SC-discr c1* $\wedge$ *SC-discr c2*

primrec *SC-siso* **where**
SC-siso (*Atm atm*) $\longleftrightarrow$ *compatAtm atm*
| *SC-siso* (*Seq c1 c2*) $\longleftrightarrow$ *SC-siso c1* $\wedge$ *SC-siso c2*
| *SC-siso* (*If tst c1 c2*) $\longleftrightarrow$ *compatTst tst* $\wedge$ *SC-siso c1* $\wedge$ *SC-siso c2*
| *SC-siso* (*While tst c*) $\longleftrightarrow$ *compatTst tst* $\wedge$ *SC-siso c*
| *SC-siso* (*Par c1 c2*) $\longleftrightarrow$ *SC-siso c1* $\wedge$ *SC-siso c2*

primrec *SC-WbisT* **where**
SC-WbisT (*Atm atm*) $\longleftrightarrow$ *compatAtm atm*

| *SC-WbisT* (*Seq c1 c2*) ⟷ (*SC-WbisT c1* ∧ *SC-WbisT c2*) ∨
(*noWhile* (*Seq c1 c2*) ∧ *SC-discr* (*Seq c1 c2*)) ∨
SC-siso (*Seq c1 c2*)
| *SC-WbisT* (*If tst c1 c2*) ⟷ (*if compatTst tst*
then (*SC-WbisT c1* ∧ *SC-WbisT c2*)
else ((*noWhile* (*If tst c1 c2*) ∧ *SC-discr* (*If tst c1 c2*)) ∨
SC-siso (*If tst c1 c2*)))
| *SC-WbisT* (*While tst c*) ⟷ (*if compatTst tst*
then SC-WbisT c
else ((*noWhile* (*While tst c*) ∧ *SC-discr* (*While tst c*)) ∨
SC-siso (*While tst c*)))
| *SC-WbisT* (*Par c1 c2*) ⟷ (*SC-WbisT c1* ∧ *SC-WbisT c2*) ∨
(*noWhile* (*Par c1 c2*) ∧ *SC-discr* (*Par c1 c2*)) ∨
SC-siso (*Par c1 c2*)

primrec *SC-ZObis* **where**
SC-ZObis (*Atm atm*) ⟷ *compatAtm atm*
| *SC-ZObis* (*Seq c1 c2*) ⟷ (*SC-siso c1* ∧ *SC-ZObis c2*) ∨
(*SC-ZObis c1* ∧ *SC-discr c2*) ∨
SC-discr (*Seq c1 c2*) ∨
SC-siso (*Seq c1 c2*)
| *SC-ZObis* (*If tst c1 c2*) ⟷ (*if compatTst tst*
then (*SC-ZObis c1* ∧ *SC-ZObis c2*)
else (*SC-discr* (*If tst c1 c2*) ∨
SC-siso (*If tst c1 c2*)))
| *SC-ZObis* (*While tst c*) ⟷ *SC-discr* (*While tst c*) ∨
SC-siso (*While tst c*)
| *SC-ZObis* (*Par c1 c2*) ⟷ (*SC-ZObis c1* ∧ *SC-ZObis c2*) ∨
SC-discr (*Par c1 c2*) ∨
SC-siso (*Par c1 c2*)

primrec *SC-Wbis* **where**
SC-Wbis (*Atm atm*) ⟷ *compatAtm atm*
| *SC-Wbis* (*Seq c1 c2*) ⟷ (*SC-WbisT c1* ∧ *SC-Wbis c2*) ∨
(*SC-Wbis c1* ∧ *SC-discr c2*) ∨
SC-ZObis (*Seq c1 c2*) ∨
SC-WbisT (*Seq c1 c2*)
| *SC-Wbis* (*If tst c1 c2*) ⟷ (*if compatTst tst*
then (*SC-Wbis c1* ∧ *SC-Wbis c2*)
else (*SC-ZObis* (*If tst c1 c2*) ∨
SC-WbisT (*If tst c1 c2*)))
| *SC-Wbis* (*While tst c*) ⟷ *SC-ZObis* (*While tst c*) ∨
SC-WbisT (*While tst c*)
| *SC-Wbis* (*Par c1 c2*) ⟷ (*SC-Wbis c1* ∧ *SC-Wbis c2*) ∨
SC-ZObis (*Par c1 c2*) ∨
SC-WbisT (*Par c1 c2*)

primrec *SC-BisT* **where**
SC-BisT (*Atm atm*) ⟷ *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* $\Longrightarrow$ *discr c*
⟨*proof*⟩

theorem *SC-siso*[*intro*]: *SC-siso c* $\Longrightarrow$ *siso c*
⟨*proof*⟩

theorem *SC-siso-imp-SC-WbisT*[*intro*]: *SC-siso c* $\Longrightarrow$ *SC-WbisT c*
⟨*proof*⟩

theorem *SC-discr-imp-SC-WbisT*[*intro*]: *noWhile c* $\Longrightarrow$ *SC-discr c* $\Longrightarrow$ *SC-WbisT c*
⟨*proof*⟩

theorem *SC-WbisT*[*intro*]: *SC-WbisT c* $\Longrightarrow$ *c* $\approx$*wT c*
⟨*proof*⟩

theorem *SC-discr-imp-SC-ZObis*[*intro*]: *SC-discr c* $\Longrightarrow$ *SC-ZObis c*
⟨*proof*⟩

theorem *SC-siso-imp-SC-ZObis*[*intro*]: *SC-siso c* $\Longrightarrow$ *SC-ZObis c*
⟨*proof*⟩

theorem *SC-ZObis*[*intro*]: *SC-ZObis c* $\Longrightarrow$ *c* $\approx$*01 c*
⟨*proof*⟩

theorem *SC-ZObis-imp-SC-Wbis*[*intro*]: *SC-ZObis c* $\Longrightarrow$ *SC-Wbis c*
⟨*proof*⟩

theorem *SC-WbisT-imp-SC-Wbis*[*intro*]: *SC-WbisT c* $\Longrightarrow$ *SC-Wbis c*
⟨*proof*⟩

theorem *SC-Wbis*[*intro*]: *SC-Wbis c* $\Longrightarrow$ *c* $\approx$*w c*
⟨*proof*⟩

theorem *SC-discr-imp-SC-BisT*[*intro*]: *SC-discr c* $\Longrightarrow$ *SC-BisT c*
⟨*proof*⟩

theorem *SC-WbisT-imp-SC-BisT*[*intro*]: *SC-WbisT c* $\Longrightarrow$ *SC-BisT c*
⟨*proof*⟩

theorem *SC-ZObis-imp-SC-BisT*[*intro*]: *SC-ZObis c* $\Longrightarrow$ *SC-BisT c*
⟨*proof*⟩

theorem *SC-Wbis-imp-SC-BisT*[*intro*]: *SC-Wbis c* $\Longrightarrow$ *SC-BisT c*
⟨*proof*⟩

theorem *SC-BisT*[*intro*]: *SC-BisT c* $\Longrightarrow$ *c* $\approx$*T c*
⟨*proof*⟩

theorem *SC-WbisT-While*: *SC-WbisT* (*While tst c*) $\longleftrightarrow$ *SC-WbisT c* $\wedge$ *compatTst tst*
⟨*proof*⟩

theorem *SC-ZObis-While*: *SC-ZObis* (*While tst c*) $\longleftrightarrow$ (*compatTst tst* $\wedge$ *SC-siso c*) $\vee$ *SC-discr c*
⟨*proof*⟩

theorem *SC-ZObis-If*: *SC-ZObis* (*If tst c1 c2*) $\longleftrightarrow$ (*if compatTst tst then SC-ZObis c1* $\wedge$ *SC-ZObis c2 else SC-discr c1* $\wedge$ *SC-discr c2*)
⟨*proof*⟩

theorem *SC-WbisT-Seq*: *SC-WbisT* (*Seq c1 c2*) $\longleftrightarrow$ (*SC-WbisT c1* $\wedge$ *SC-WbisT c2*)
⟨*proof*⟩

theorem *SC-ZObis-Seq*: *SC-ZObis* (*Seq c1 c2*) $\longleftrightarrow$ (*SC-siso c1* $\wedge$ *SC-ZObis c2*) $\vee$
(*SC-ZObis c1* $\wedge$ *SC-discr c2*)
⟨*proof*⟩

theorem *SC-Wbis-Seq*: *SC-Wbis* (*Seq c1 c2*) $\longleftrightarrow$ (*SC-WbisT c1* $\wedge$ *SC-Wbis c2*) $\vee$ (*SC-Wbis c1* $\wedge$ *SC-discr c2*)
⟨*proof*⟩

theorem *SC-BisT-Par*:
SC-BisT (*Par c1 c2*) $\longleftrightarrow$ (*SC-BisT c1* $\wedge$ *SC-BisT c2*)
⟨*proof*⟩

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** $(c,s) \to c\ (c',s')$
obtains $d'\ t'$ **where**
$(d,t) \to c\ (d',t')$ **and** $s' \approx t'$ **and** $c' \approx s\ d'$
⟨*proof*⟩

lemma *Sbis-transT*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx s\ d$ **and** $s \approx t$ **and** $(c,s) \to t\ s'$
obtains t' **where** $(d,t) \to t\ t'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *Sbis-transC2*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx s\ d$ **and** $s \approx t$ **and** $(d,t) \to c\ (d',t')$
obtains $c'\ s'$ **where**
$(c,s) \to c\ (c',s')$ **and** $s' \approx t'$ **and** $c' \approx s\ d'$
⟨*proof*⟩

lemma *Sbis-transT2*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx s\ d$ **and** $s \approx t$ **and** $(d,t) \to t\ t'$
obtains s' **where** $(c,s) \to t\ s'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *ZObisT-transC*[*consumes 3*, *case-names Match MatchS*]:
assumes *0*: $c \approx 01T\ d$ **and** $s \approx t$ **and** $(c,s) \to c\ (c',s')$
and $\bigwedge d'\ t'.\ [\![(d,t) \to c\ (d',t');\ s' \approx t';\ c' \approx 01T\ d']\!] \Longrightarrow thesis$
and $[\![s' \approx t;\ c' \approx 01T\ d]\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *ZObisT-transT*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx 01T\ d$ **and** $s \approx t$ **and** $(c,s) \to t\ s'$
obtains t' **where** $(d,t) \to t\ t'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *ZObisT-transC2*[*consumes 3*, *case-names Match MatchS*]:

assumes *0*: $c \approx 01T\ d$ **and** *2*: $s \approx t$ **and** *3*: $(d,t) \to c\ (d',t')$
and *4*: $\bigwedge c'\ s'.\ [\![(c,s) \to c\ (c',s');\ s' \approx t';\ c' \approx 01T\ d']\!] \Longrightarrow thesis$
and *5*: $[\![s \approx t';\ c \approx 01T\ d']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *ZObisT-transT2*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx 01T\ d$ **and** $s \approx t$ **and** $(d,t) \to t\ t'$
obtains s' **where** $(c,s) \to t\ s'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *WbisT-transC*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx wT\ d$ **and** $s \approx t$ **and** $(c,s) \to c\ (c',s')$
obtains $d'\ t'$ **where**
$(d,t) \to *c\ (d',t')$ **and** $s' \approx t'$ **and** $c' \approx wT\ d'$
⟨*proof*⟩

lemma *WbisT-transT*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx wT\ d$ **and** $s \approx t$ **and** $(c,s) \to t\ s'$
obtains t' **where** $(d,t) \to *t\ t'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *WbisT-transC2*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx wT\ d$ **and** $s \approx t$ **and** $(d,t) \to c\ (d',t')$
obtains $c'\ s'$ **where**
$(c,s) \to *c\ (c',s')$ **and** $s' \approx t'$ **and** $c' \approx wT\ d'$
⟨*proof*⟩

lemma *WbisT-transT2*[*consumes 3*, *case-names Match*]:
assumes *0*: $c \approx wT\ d$ **and** $s \approx t$ **and** $(d,t) \to t\ t'$
obtains s' **where** $(c,s) \to *t\ s'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *WbisT-MtransC*[*consumes 3*, *case-names Match*]:
assumes *1*: $c \approx wT\ d$ **and** *2*: $s \approx t$ **and** *3*: $(c,s) \to *c\ (c',s')$
obtains $d'\ t'$ **where**
$(d,t) \to *c\ (d',t')$ **and** $s' \approx t'$ **and** $c' \approx wT\ d'$
⟨*proof*⟩

lemma *WbisT-MtransT*[*consumes 3*, *case-names Match*]:
assumes *1*: $c \approx wT\ d$ **and** *2*: $s \approx t$ **and** *3*: $(c,s) \to *t\ s'$
obtains t' **where** $(d,t) \to *t\ t'$ **and** $s' \approx t'$
⟨*proof*⟩

lemma *WbisT-MtransC2*[*consumes 3*, *case-names Match*]:
assumes $c \approx wT\ d$ **and** $s \approx t$ **and** *1*: $(d,t) \to *c\ (d',t')$
obtains $c'\ s'$ **where**

$(c,s) \rightarrow *c\ (c',s')$ **and** $s' \approx t'$ **and** $c' \approx wT\ d'$
⟨*proof*⟩

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** $s' \approx t'$
⟨*proof*⟩

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 $\bigwedge d'\ t'.\ [\![(d,t) \rightarrow c\ (d',t');\ s' \approx t';\ c' \approx 01\ d']\!] \Longrightarrow thesis$
and $\bigwedge t'.\ [\![(d,t) \rightarrow t\ t';\ s' \approx t';\ discr\ c']\!] \Longrightarrow thesis$
and $[\![s' \approx t;\ c' \approx 01\ d]\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

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 $\bigwedge t'.\ [\![(d,t) \rightarrow t\ t';\ s' \approx t']\!] \Longrightarrow thesis$
and $\bigwedge d'\ t'.\ [\![(d,t) \rightarrow c\ (d',t');\ s' \approx t';\ discr\ d']\!] \Longrightarrow thesis$
and $[\![s' \approx t;\ discr\ d]\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

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 $\bigwedge c'\ s'.\ [\![(c,s) \rightarrow c\ (c',s');\ s' \approx t';\ c' \approx 01\ d']\!] \Longrightarrow thesis$
and $\bigwedge s'.\ [\![(c,s) \rightarrow t\ s';\ s' \approx t';\ discr\ d']\!] \Longrightarrow thesis$
and $[\![s \approx t';\ c \approx 01\ d']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

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 $\bigwedge s'.\ [\![(c,s) \rightarrow t\ s';\ s' \approx t']\!] \Longrightarrow thesis$
and $\bigwedge c'\ s'.\ [\![(c,s) \rightarrow c\ (c',s');\ s' \approx t';\ discr\ c']\!] \Longrightarrow thesis$
and $[\![s \approx t';\ discr\ c]\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

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 $\bigwedge d'\ t'.\ [\![(d,t) \rightarrow *c\ (d',t');\ s' \approx t';\ c' \approx w\ d']\!] \Longrightarrow thesis$
and $\bigwedge t'.\ [\![(d,t) \rightarrow *t\ t';\ s' \approx t';\ discr\ c']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-transT*[*consumes 3*, *case-names Match MatchO*]:
assumes *0*: $c \approx w\ d$ **and** $s \approx t$ **and** $(c,s) \to t\ s'$
and $\bigwedge t'.\ [\![(d,t) \to{*}t\ t';\ s' \approx t']\!] \Longrightarrow thesis$
and $\bigwedge d'\ t'.\ [\![(d,t) \to{*}c\ (d',t');\ s' \approx t';\ discr\ d']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-transC2*[*consumes 3*, *case-names Match MatchO*]:
assumes *0*: $c \approx w\ d$ **and** $s \approx t$ **and** $(d,t) \to c\ (d',t')$
and $\bigwedge c'\ s'.\ [\![(c,s) \to{*}c\ (c',s');\ s' \approx t';\ c' \approx w\ d']\!] \Longrightarrow thesis$
and $\bigwedge s'.\ [\![(c,s) \to{*}t\ s';\ s' \approx t';\ discr\ d']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-transT2*[*consumes 3*, *case-names Match MatchO*]:
assumes *0*: $c \approx w\ d$ **and** $s \approx t$ **and** $(d,t) \to t\ t'$
and $\bigwedge s'.\ [\![(c,s) \to{*}t\ s';\ s' \approx t']\!] \Longrightarrow thesis$
and $\bigwedge c'\ s'.\ [\![(c,s) \to{*}c\ (c',s');\ s' \approx t';\ discr\ c']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-MtransC*[*consumes 3*, *case-names Match MatchO*]:
assumes $c \approx w\ d$ **and** $s \approx t$ **and** $(c,s) \to{*}c\ (c',s')$
and $\bigwedge d'\ t'.\ [\![(d,t) \to{*}c\ (d',t');\ s' \approx t';\ c' \approx w\ d']\!] \Longrightarrow thesis$
and $\bigwedge t'.\ [\![(d,t) \to{*}t\ t';\ s' \approx t';\ discr\ c']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-MtransT*[*consumes 3*, *case-names Match MatchO*]:
assumes *c-d*: $c \approx w\ d$ **and** *st*: $s \approx t$ **and** *cs*: $(c,s) \to{*}t\ s'$
and *1*: $\bigwedge t'.\ [\![(d,t) \to{*}t\ t';\ s' \approx t']\!] \Longrightarrow thesis$
and *2*: $\bigwedge d'\ t'.\ [\![(d,t) \to{*}c\ (d',t');\ s' \approx t';\ discr\ d']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-MtransC2*[*consumes 3*, *case-names Match MatchO*]:
assumes $c \approx w\ d$ **and** $s \approx t$ **and** *dt*: $(d,t) \to{*}c\ (d',t')$
and *1*: $\bigwedge c'\ s'.\ [\![(c,s) \to{*}c\ (c',s');\ s' \approx t';\ c' \approx w\ d']\!] \Longrightarrow thesis$
and *2*: $\bigwedge s'.\ [\![(c,s) \to{*}t\ s';\ s' \approx t';\ discr\ d']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma *Wbis-MtransT2*[*consumes 3*, *case-names Match MatchO*]:
assumes $c \approx w\ d$ **and** $s \approx t$ **and** *dt*: $(d,t) \to{*}t\ t'$
and *1*: $\bigwedge s'.\ [\![(c,s) \to{*}t\ s';\ s' \approx t']\!] \Longrightarrow thesis$
and *2*: $\bigwedge c'\ s'.\ [\![(c,s) \to{*}c\ (c',s');\ s' \approx t';\ discr\ c']\!] \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

lemma $BisT\text{-}transC[consumes\ 5,\ case\text{-}names\ Match]$:
assumes 0: $c \approx T\ d$
and $mustT\ c\ s$ **and** $mustT\ d\ t$
and $s \approx t$ **and** $(c,s) \to c\ (c',s')$
obtains $d'\ t'$ **where**
$(d,t) \to *c\ (d',t')$ **and** $s' \approx t'$ **and** $c' \approx T\ d'$
$\langle proof \rangle$

lemma $BisT\text{-}transT[consumes\ 5,\ case\text{-}names\ Match]$:
assumes 0: $c \approx T\ d$
and $mustT\ c\ s$ **and** $mustT\ d\ t$
and $s \approx t$ **and** $(c,s) \to t\ s'$
obtains t' **where** $(d,t) \to *t\ t'$ **and** $s' \approx t'$
$\langle proof \rangle$

lemma $BisT\text{-}transC2[consumes\ 5,\ case\text{-}names\ Match]$:
assumes 0: $c \approx T\ d$
and $mustT\ c\ s$ **and** $mustT\ d\ t$
and $s \approx t$ **and** $(d,t) \to c\ (d',t')$
obtains $c'\ s'$ **where**
$(c,s) \to *c\ (c',s')$ **and** $s' \approx t'$ **and** $c' \approx T\ d'$
$\langle proof \rangle$

lemma $BisT\text{-}transT2[consumes\ 5,\ case\text{-}names\ Match]$:
assumes 0: $c \approx T\ d$
and $mustT\ c\ s$ **and** $mustT\ d\ t$
and $s \approx t$ **and** $(d,t) \to t\ t'$
obtains s' **where** $(c,s) \to *t\ s'$ **and** $s' \approx t'$
$\langle proof \rangle$

lemma $BisT\text{-}MtransC[consumes\ 5,\ case\text{-}names\ Match]$:
assumes $c \approx T\ d$
and $mustT\ c\ s\ mustT\ d\ t$
and $s \approx t$ **and** $(c,s) \to *c\ (c',s')$
obtains $d'\ t'$ **where**
$(d,t) \to *c\ (d',t')$ **and** $s' \approx t'$ **and** $c' \approx T\ d'$
$\langle proof \rangle$

lemma $BisT\text{-}MtransT[consumes\ 5,\ case\text{-}names\ Match]$:
assumes 1: $c \approx T\ d$
and ter: $mustT\ c\ s\ mustT\ d\ t$
and 2: $s \approx t$ **and** 3: $(c,s) \to *t\ s'$
obtains t' **where** $(d,t) \to *t\ t'$ **and** $s' \approx t'$
$\langle proof \rangle$

lemma $BisT\text{-}MtransC2[consumes\ 3,\ case\text{-}names\ Match]$:

assumes $c \approx T\ d$
and *ter*: *mustT c s mustT d t*
and $s \approx t$ **and** *1*: $(d,t) \to *c\ (d',t')$
obtains $c'\ s'$ **where**
$(c,s) \to *c\ (c',s')$ **and** $s' \approx t'$ **and** $c' \approx T\ d'$
⟨*proof*⟩

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** $(d,t) \to *t\ t'$
obtains s' **where** $(c,s) \to *t\ s'$ **and** $s' \approx t'$
⟨*proof*⟩

6.2 Execution traces

primrec *parTrace* **where**
parTrace [] $\longleftrightarrow$ *False* |
parTrace $(cf\#cfl) \longleftrightarrow (cfl \neq [] \longrightarrow parTrace\ cfl \wedge cf \to c\ hd\ cfl)$

lemma *trans-Step2*:
$cf \to *c\ cf' \Longrightarrow cf' \to c\ cf'' \Longrightarrow cf \to *c\ cf''$
⟨*proof*⟩

lemma *parTrace-not-empty*[*simp*]: $parTrace\ cfl \Longrightarrow cfl \neq []$
⟨*proof*⟩

lemma *parTrace-snoc*[*simp*]:
$parTrace\ (cfl@[cf]) \longleftrightarrow (cfl \neq [] \longrightarrow parTrace\ cfl \wedge last\ cfl \to c\ cf)$
⟨*proof*⟩

lemma *MtransC-Ex-parTrace*:
assumes $cf \to *c\ cf'$ **shows** $\exists\, cfl.\ parTrace\ cfl \wedge hd\ cfl = cf \wedge last\ cfl = cf'$
⟨*proof*⟩

lemma *parTrace-imp-MtransC*:
assumes *pT*: *parTrace cfl*
shows $(hd\ cfl) \to *c\ (last\ cfl)$
⟨*proof*⟩

fun *finTrace* **where**
finTrace $(cfl,s) \longleftrightarrow$
$parTrace\ cfl \wedge last\ cfl \to t\ s$

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-Ex-finTrace*:
assumes $cf \rightarrow *t\ s$ **shows** $\exists\, tr.\ finTrace\ tr \wedge iconfig\ tr = cf \wedge fstate\ tr = s$
⟨*proof*⟩

lemma *finTrace-imp-MtransT*:
$finTrace\ tr \Longrightarrow iconfig\ tr \rightarrow *t\ fstate\ tr$
⟨*proof*⟩

6.3 Relationship between during-execution and after-execution security

lemma *WbisT-trace2*:
assumes *bis*: $c \approx wT\ d\ s \approx t$
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*⟩

theorem *WbisT-trace*:
assumes $c \approx wT\ c$ **and** $s \approx t$
and $finTrace\ tr$ **and** $iconfig\ tr = (c,s)$
shows $\exists\, tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge fstate\ tr \approx fstate\ tr'$
⟨*proof*⟩

theorem *ZObisT-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*⟩

theorem *ZObisT-trace*:
assumes $c \approx 01T\ c\ s \approx t$
and $finTrace\ tr\ iconfig\ tr = (c,s)$
shows $\exists\, tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge$
$fstate\ tr \approx fstate\ tr' \wedge lengthFT\ tr' \leq lengthFT\ tr$
⟨*proof*⟩

theorem *Sbis-trace*:
assumes *bis*: $c \approx s\ d\ s \approx t$
and *tr*: $finTrace\ tr\ iconfig\ tr = (c,s)$
shows $\exists\, tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (d,t) \wedge fstate\ tr \approx fstate\ tr' \wedge$
$lengthFT\ tr' = lengthFT\ tr$

⟨*proof*⟩

corollary *siso-trace*:
assumes *siso c* **and** $s \approx t$
and *finTrace tr* **and** *iconfig tr* = (*c*,*s*)
shows
$\exists\ tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge fstate\ tr \approx fstate\ tr'$
$\quad \wedge\ lengthFT\ tr' = lengthFT\ tr$
⟨*proof*⟩

theorem *Wbis-trace*:
assumes *T*: $\bigwedge s.\ mustT\ c\ s$
and *bis*: $c \approx w\ c\ s \approx t$
and *tr*: *finTrace tr iconfig tr* = (*c*,*s*)
shows $\exists\, tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge fstate\ tr \approx fstate\ tr'$
⟨*proof*⟩

corollary *ZObis-trace*:
assumes *T*: $\bigwedge s.\ mustT\ c\ s$
and *ZObis*: $c \approx 01\ c$ **and** *indis*: $s \approx t$
and *tr*: *finTrace tr iconfig tr* = (*c*,*s*)
shows $\exists\, tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge fstate\ tr \approx fstate\ tr'$
⟨*proof*⟩

theorem *BisT-trace*:
assumes *bis*: $c \approx T\ c\ s \approx t$
and *T*: *mustT c s mustT c t*
and *tr*: *finTrace tr iconfig tr* = (*c*,*s*)
shows $\exists\, tr'.\ finTrace\ tr' \wedge iconfig\ tr' = (c,t) \wedge fstate\ tr \approx fstate\ tr'$
⟨*proof*⟩

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 (*If tst c1 c2*) $\approx wT$ (*If tst c1 c2*)

$\langle proof \rangle$

We instatiate 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*. $Lo \neq l \longleftrightarrow Hi = l$
$\langle proof \rangle$

lemma [*dest*]: $\bigwedge$ *l A*. $[\![l \in A;\ Lo \notin A]\!] \Longrightarrow l = Hi$ **and**
    [*dest*]: $\bigwedge$ *l A*. $[\![l \in A;\ Hi \notin A]\!] \Longrightarrow l = Lo$
$\langle proof \rangle$

declare *level.split*[*split*]

instantiation *level* :: *complete-lattice*
begin
  definition *top-level*: *top* ≡ *Hi*
  definition *bot-level*: *bot* ≡ *Lo*
  definition *inf-level*: *inf l1 l2* ≡ *if Lo* ∈ {*l1*,*l2*} *then Lo else Hi*
  definition *sup-level*: *sup l1 l2* ≡ *if Hi* ∈ {*l1*,*l2*} *then Hi else Lo*
  definition *less-eq-level*: *less-eq l1 l2* ≡ (*l1* = *Lo* ∨ *l2* = *Hi*)
  definition *less-level*: *less l1 l2* ≡ *l1* = *Lo* ∧ *l2* = *Hi*
  definition *Inf-level*: *Inf L* ≡ *if Lo* ∈ *L then Lo else Hi*
  definition *Sup-level*: *Sup L* ≡ *if Hi* ∈ *L then Hi else Lo*
instance
  $\langle proof \rangle$
end

lemma *sup-eq-Lo*[*simp*]: *sup a b* = *Lo* ⟷ *a* = *Lo* ∧ *b* = *Lo*
  $\langle proof \rangle$

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* ⇒ *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* = *eval e1 s* + *eval e2 s*
|*eval* (*Minus e1 e2*) *s* = *eval e1 s* − *eval e2 s*

fun *tval* **where**
tval Tr s = *True*
|*tval* (*Eq e1 e2*) *s* = (*eval e1 s* = *eval e2 s*)
|*tval* (*Gt e1 e2*) *s* = (*eval e1 s* > *eval e2 s*)
|*tval* (*Non e*) *s* = (¬ *tval e s*)

fun *aval* **where**
aval (*Assign x e*) *s* = (*s* (*x* := *eval e s*))

definition *indis* :: (*state* ∗ *state*) *set***where**
indis ≡ {(*s*,*t*). *ALL x*. *sec x* = *Lo* ⟶ *s x* = *t x*}

interpretation *Example-PL*: *PL-Indis tval aval indis*
⟨*proof*⟩

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* ⟹ (*s*, *t*) ∈ *indis* ⟹ *eval expr s*

$= eval\ expr\ t$
⟨proof⟩

lemma *compatAtmSyntactic*[*simp*]: $exprSec\ expr = Lo \lor sec\ v = Hi \Longrightarrow Example\text{-}PL.compatAtm\ (Assign\ v\ expr)$
⟨proof⟩

lemma *presAtmSyntactic*[*simp*]: $sec\ v = Hi \Longrightarrow Example\text{-}PL.presAtm\ (Assign\ v\ expr)$
⟨proof⟩

lemma *compatTstSyntactic*[*simp*]: $tstSec\ tst = Lo \Longrightarrow Example\text{-}PL.compatTst\ tst$
⟨proof⟩

lemma $Example\text{-}PL.SC\text{-}discr\ (h ::= Ct\ 0)$
⟨proof⟩

abbreviation $siso\ c \equiv Example\text{-}PL.siso\ c$
abbreviation $siso0\ c \equiv Example\text{-}PL.siso0\ c$
abbreviation $discr\ c \equiv Example\text{-}PL.discr\ c$
abbreviation $discr0\ c \equiv Example\text{-}PL.discr0\ c$
abbreviation *Sbis-abbrev* (**infix** $\approx s$ *55*) **where** $c1 \approx s\ c2 \equiv (c1,c2) \in Example\text{-}PL.Sbis$
abbreviation *ZObis-abbrev* (**infix** ≈ 01 *55*) **where** $c1 \approx 01\ c2 \equiv (c1,c2) \in Example\text{-}PL.ZObis$
abbreviation *ZObisT-abbrev* (**infix** $\approx 01T$ *55*) **where** $c1 \approx 01T\ c2 \equiv (c1,c2) \in Example\text{-}PL.ZObisT$
abbreviation *Wbis-abbrev* (**infix** $\approx w$ *55*) **where** $c1 \approx w\ c2 \equiv (c1,c2) \in Example\text{-}PL.Wbis$
abbreviation *WbisT-abbrev* (**infix** $\approx wT$ *55*) **where** $c1 \approx wT\ c2 \equiv (c1,c2) \in Example\text{-}PL.WbisT$
abbreviation *BisT-abbrev* (**infix** $\approx T$ *55*) **where** $c1 \approx T\ c2 \equiv (c1,c2) \in Example\text{-}PL.BisT$

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*]: $c7 = l ::= Var\ h\ ;;\ l ::= Ct\ 0$

definition [*simp*]: $c8 = h' ::= Var\ h$;;
  while Gt (*Var h*) (*Ct 0*) *do* ($h ::= Minus\ (Var\ h)\ (Ct\ 1)$;; $h' ::= Plus\ (Var\ h')$ (*Ct 1*)) ;;
  $l ::= Ct\ 4$

definition [*simp*]: $c9 = c7 \mid l' ::= Var\ l$

definition [*simp*]: $c10 = c5 \mid l ::= Ct\ 5$

definition [*simp*]: $c11 = c8 \mid l ::= Ct\ 5$

declare *bot-level*[*iff*]

theorem *c0*: *siso c0 discr c0*
  ⟨*proof*⟩

theorem *c1*: *siso c1* $c1 \approx s\ c1$
  ⟨*proof*⟩

theorem *c2*: *discr c2*
  ⟨*proof*⟩

theorem *Sbis-c2*: $c2 \approx s\ c2$
  ⟨*proof*⟩

theorem *c3*: *discr c3*
  ⟨*proof*⟩

theorem *c4*: $c4 \approx 01\ c4$
  ⟨*proof*⟩

theorem *c5*: $c5 \approx w\ c5$
  ⟨*proof*⟩

Example 4 from the paper

theorem $c3 \approx wT\ c3$ ⟨*proof*⟩

theorem $c5 \approx wT\ c5$ ⟨*proof*⟩

corollary *discr* (*while Eq* (*Var h*) (*Ct 0*) *do* $h ::= Ct\ 0$)
  ⟨*proof*⟩

Example 5 from the paper

definition [*simp*]: $c12 \equiv h ::= Ct\ 4$;;
  while Gt (*Var h*) (*Ct 0*)
    do ($h ::= Minus\ (Var\ h)\ (Ct\ 1)$;; $h' ::= Plus\ (Var\ h')\ (Ct\ 1)$) ;;
  $l ::= Ct\ 1$

corollary $(c12 \mid l ::= Ct\ 2) \approx T\ (c12 \mid l ::= Ct\ 2)$

$\langle proof \rangle$

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*)
$\langle proof \rangle$

theorem *c13* ≈*wT* *c13*
$\langle proof \rangle$

end

References

[1] A. Popescu, J. Hölzl, and T. Nipkow. Proving possibilistic, probabilistic noninterference. In *Certified Programs and Proofs (CPP) '12*, 2012.