

Probabilistic Noninterference

Andrei Popescu

Johannes Hölzl

August 16, 2018

Abstract

We formalize a probabilistic noninterference for a multi-threaded language with uniform scheduling, where probabilistic behaviour comes from both the scheduler and the individual threads. We define notions probabilistic noninterference in two variants: resumption-based and trace-based. For the resumption-based notions, we prove compositionality w.r.t. the language constructs and establish sound type-system-like syntactic criteria.

This is a formalization of the mathematical development presented in the papers [1, 2]. It is the probabilistic variant of the [Possibilistic Noninterference AFP](#) entry.

Contents

1	Simple While Language with probabilistic choice and parallel execution	2
1.1	Preliminaries	2
1.2	Syntax	7
1.2.1	Operational Small-Step Semantics	11
1.2.2	Operations on configurations	21
2	Resumption-Based Noninterference	21
2.1	Preliminaries	21
2.2	Infrastructure for partitions	22
2.3	Basic setting for bisimilarity	28
2.4	Discreetness	29
2.5	Self-isomorphism	30
2.6	Notions of bisimilarity	31
2.7	List versions of the bisimilarities	37
2.8	Discreetness for configurations	39
3	Trace-Based Noninterference	40
3.1	Preliminaries	40
3.2	Quasi strong termination traces	43

3.3	Terminating configurations	44
3.4	Final Theorems	45
4	Compositionality of Resumption-Based Noninterference	45
4.1	Compatibility and discreteness of atoms, tests and choices . .	46
4.2	Compositionality of self-isomorphism	46
4.3	Discreteness versus language constructs:	48
4.4	Strong bisimilarity versus language constructs	49
4.5	01-bisimilarity versus language constructs	59
5	Syntactic Criteria	69
6	Concrete setting	72
6.1	The secure programs from the paper's Example 3	75

1 Simple While Language with probabilistic choice and parallel execution

```
theory Language-Semantics
imports Interface
begin
```

1.1 Preliminaries

Trivia

```
declare zero-le-mult-iff[simp]
declare split-mult-pos-le[simp]
declare zero-le-divide-iff[simp]
```

```
lemma in-minus-Un[simp]:
assumes  $i \in I$ 
shows  $I - \{i\} \cup \{i\} = I$  and  $\{i\} \cup (I - \{i\}) = I$ 
⟨proof⟩
```

```
lemma less-plus-cases[case-names Left Right]:
assumes
*:  $(i::nat) < n1 \implies phi$  and
**:  $\bigwedge i2. i = n1 + i2 \implies phi$ 
shows  $phi$ 
⟨proof⟩
```

```
lemma less-plus-elim[elim!, consumes 1, case-names Left Right]:
assumes  $i: (i::nat) < n1 + n2$  and
*:  $i < n1 \implies phi$  and
**:  $\bigwedge i2. [i2 < n2; i = n1 + i2] \implies phi$ 
shows  $phi$ 
⟨proof⟩
```

lemma *nth-append-singl*[simp]:
 $i < \text{length } al \implies (al @ [a]) ! i = al ! i$
(proof)

lemma *take-append-singl*[simp]:
assumes $n < \text{length } al$ **shows** $\text{take } n (al @ [a]) = \text{take } n al$
(proof)

lemma *length-unique-prefix*:
 $al1 \leq al \implies al2 \leq al \implies \text{length } al1 = \text{length } al2 \implies al1 = al2$
(proof)

take:

lemma *take-length*[simp]:
 $\text{take } (\text{length } al) al = al$
(proof)

lemma *take-le*:
assumes $n < \text{length } al$
shows $\text{take } n al @ [al ! n] \leq al$
(proof)

lemma *list-less-iff-prefix*: $a < b \iff \text{strict-prefix } a b$
(proof)

lemma *take-lt*:
 $n < \text{length } al \implies \text{take } n al < al$
(proof)

lemma *le-take*:
assumes $n1 \leq n2$
shows $\text{take } n1 al \leq \text{take } n2 al$
(proof)

lemma *inj-take*:
assumes $n1 \leq \text{length } al$ **and** $n2 \leq \text{length } al$
shows $\text{take } n1 al = \text{take } n2 al \iff n1 = n2$
(proof)

lemma *lt-take*: $n1 < n2 \implies n2 \leq \text{length } al \implies \text{take } n1 al < \text{take } n2 al$
(proof)

lsum:

definition *lsum* :: $(a \Rightarrow \text{nat}) \Rightarrow 'a \text{ list} \Rightarrow \text{nat}$ **where**
 $lsum f al \equiv \text{sum-list } (\text{map } f al)$

lemma *lsum-simps*[simp]:
 $lsum f [] = 0$

$lsum\ f\ (al\ @\ [a]) = lsum\ f\ al + f\ a$
<proof>

lemma *lsum-append*:

$lsum\ f\ (al\ @\ bl) = lsum\ f\ al + lsum\ f\ bl$
<proof>

lemma *lsum-cong[fundef-cong]*:

assumes $\bigwedge a. a \in set\ al \implies f\ a = g\ a$

shows $lsum\ f\ al = lsum\ g\ al$

<proof>

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

assumes $al \neq []$ **and** $\bigwedge a. a \in set\ al \implies 0 < f\ a$

shows $0 < lsum\ f\ al$

<proof>

lemma *lsum-mono[simp]*:

assumes $al \leq bl$

shows $lsum\ f\ al \leq lsum\ f\ bl$

<proof>

lemma *lsum-mono2[simp]*:

assumes $f: \bigwedge b. b \in set\ bl \implies f\ b > 0$ **and** $le: al < bl$

shows $lsum\ f\ al < lsum\ f\ bl$

<proof>

lemma *lsum-take[simp]*:

$lsum\ f\ (take\ n\ al) \leq lsum\ f\ al$

<proof>

lemma *less-lsum-nchotomy*:

assumes $f: \bigwedge a. a \in set\ al \implies 0 < f\ a$

and $i: (i::nat) < lsum\ f\ al$

shows $\exists n\ j. n < length\ al \wedge j < f\ (al\ !\ n) \wedge i = lsum\ f\ (take\ n\ al) + j$

<proof>

lemma *less-lsum-unique*:

assumes $\bigwedge a. a \in set\ al \implies (0::nat) < f\ a$

and $n1 < length\ al \wedge j1 < f\ (al\ !\ n1)$

and $n2 < length\ al \wedge j2 < f\ (al\ !\ n2)$

and $lsum\ f\ (take\ n1\ al) + j1 = lsum\ f\ (take\ n2\ al) + j2$

shows $n1 = n2 \wedge j1 = j2$

<proof>

definition *locate-pred* **where**

$locate\ pred\ f\ al\ (i::nat)\ n\ j \equiv$

$fst\ n\ j < length\ al \wedge$

$snd\ n\ j < f\ (al\ !\ (fst\ n\ j)) \wedge$

$i = lsum\ f\ (take\ (fst\ n-j)\ al) + (snd\ n-j)$

definition *locate* **where**

$locate\ f\ al\ i \equiv SOME\ n-j.\ locate-pred\ f\ al\ i\ n-j$

definition *locate1* **where** $locate1\ f\ al\ i \equiv fst\ (locate\ f\ al\ i)$

definition *locate2* **where** $locate2\ f\ al\ i \equiv snd\ (locate\ f\ al\ i)$

lemma *locate-pred-ex*:

assumes $\bigwedge a.\ a \in set\ al \implies 0 < f\ a$

and $i < lsum\ f\ al$

shows $\exists\ n-j.\ locate-pred\ f\ al\ i\ n-j$

<proof>

lemma *locate-pred-unique*:

assumes $\bigwedge a.\ a \in set\ al \implies 0 < f\ a$

and $locate-pred\ f\ al\ i\ n1-j1\ locate-pred\ f\ al\ i\ n2-j2$

shows $n1-j1 = n2-j2$

<proof>

lemma *locate-locate-pred*:

assumes $\bigwedge a.\ a \in set\ al \implies (0::nat) < f\ a$

and $i < lsum\ f\ al$

shows $locate-pred\ f\ al\ i\ (locate\ f\ al\ i)$

<proof>

lemma *locate-locate-pred-unique*:

assumes $\bigwedge a.\ a \in set\ al \implies (0::nat) < f\ a$

and $locate-pred\ f\ al\ i\ n-j$

shows $n-j = locate\ f\ al\ i$

<proof>

lemma *locate*:

assumes $\bigwedge a.\ a \in set\ al \implies 0 < f\ a$

and $i < lsum\ f\ al$

shows $locate1\ f\ al\ i < length\ al \wedge$

$locate2\ f\ al\ i < f\ (al\ !\ (locate1\ f\ al\ i)) \wedge$

$i = lsum\ f\ (take\ (locate1\ f\ al\ i)\ al) + (locate2\ f\ al\ i)$

<proof>

lemma *locate-unique*:

assumes $\bigwedge a.\ a \in set\ al \implies 0 < f\ a$

and $n < length\ al$ **and** $j < f\ (al\ !\ n)$ **and** $i = lsum\ f\ (take\ n\ al) + j$

shows $n = locate1\ f\ al\ i \wedge j = locate2\ f\ al\ i$

<proof>

sum:

lemma *sum-2[simp]*:

$sum\ f\ \{..< 2\} = f\ 0 + f\ (Suc\ 0)$

<proof>

lemma *inj-Plus[simp]*:
inj ((+) (a::nat))
<proof>

lemma *inj-on-Plus[simp]*:
inj-on ((+) (a::nat)) A
<proof>

lemma *Plus-int[simp]*:
fixes a :: nat
shows (+) b ‘ {..*a*} = {b ..< b + a}
<proof>

lemma *sum-minus[simp]*:
fixes a :: nat
shows *sum f* {a ..< a + b} = *sum* (%x. *f* (a + x)) {..*b*}
<proof>

lemma *sum-Un-introL*:
assumes A1 = B1 Un C1 **and** x = x1 + x2
finite A1 **and**
B1 Int C1 = {} **and**
sum f1 B1 = x1 **and** *sum f1* C1 = x2
shows *sum f1* A1 = x
<proof>

lemma *sum-Un-intro*:
assumes A1 = B1 Un C1 **and** A2 = B2 Un C2 **and**
finite A1 **and** *finite* A2 **and**
B1 Int C1 = {} **and** B2 Int C2 = {} **and**
sum f1 B1 = *sum f2* B2 **and** *sum f1* C1 = *sum f2* C2
shows *sum f1* A1 = *sum f2* A2
<proof>

lemma *sum-UN-introL*:
assumes A1: A1 = (UN n : N. B1 n) **and** a2: a2 = *sum b2* N **and**
fin: *finite* N \wedge n. n \in N \implies *finite* (B1 n) **and**
int: \bigwedge m n. {m, n} \subseteq N \wedge m \neq n \implies B1 m \cap B1 n = {} **and**
ss: \bigwedge n. n \in N \implies *sum f1* (B1 n) = b2 n
shows *sum f1* A1 = a2 (**is** ?L = a2)
<proof>

lemma *sum-UN-intro*:
assumes A1: A1 = (UN n : N. B1 n) **and** A2: A2 = (UN n : N. B2 n) **and**
fin: *finite* N \wedge n. n \in N \implies *finite* (B1 n) \wedge *finite* (B2 n) **and**
int: \bigwedge m n. {m, n} \subseteq N \wedge m \neq n \implies B1 m \cap B1 n = {} \wedge m n. {m, n} \subseteq N
 \implies B2 m \cap B2 n = {} **and**

ss: $\bigwedge n. n \in N \implies \text{sum } f1 (B1\ n) = \text{sum } f2 (B2\ n)$
shows $\text{sum } f1\ A1 = \text{sum } f2\ A2$ (**is** ?L = ?R)
 <proof>

lemma *sum-Minus-intro*:
fixes $f1 :: 'a1 \Rightarrow \text{real}$ **and** $f2 :: 'a2 \Rightarrow \text{real}$
assumes $B1 = A1 - \{a1\}$ **and** $B2 = A2 - \{a2\}$ **and**
 $a1 : A1$ **and** $a2 : A2$ **and** *finite* $A1$ **and** *finite* $A2$
 $\text{sum } f1\ A1 = \text{sum } f2\ A2$ **and** $f1\ a1 = f2\ a2$
shows $\text{sum } f1\ B1 = \text{sum } f2\ B2$
 <proof>

lemma *sum-singl-intro*:
assumes $b = f\ a$
and *finite* A **and** $a \in A$
and $\bigwedge a'. \llbracket a' \in A; a' \neq a \rrbracket \implies f\ a' = 0$
shows $\text{sum } f\ A = b$
 <proof>

lemma *sum-all0-intro*:
assumes $b = 0$
and $\bigwedge a. a \in A \implies f\ a = 0$
shows $\text{sum } f\ A = b$
 <proof>

lemma *sum-1*:
assumes I : *finite* I **and** *ss*: $\text{sum } f\ I = 1$ **and** $i: i \in I - I1$ **and** $I1: I1 \subseteq I$
and $f: \bigwedge i. i \in I \implies (0::\text{real}) \leq f\ i$
shows $f\ i \leq 1 - \text{sum } f\ I1$
 <proof>

1.2 Syntax

datatype (*'test, 'atom, 'choice*) *cmd* =
 Done
 | *Atm 'atom*
 | *Seq ('test, 'atom, 'choice) cmd ('test, 'atom, 'choice) cmd (- ;; - [60, 61] 60)*
 | *While 'test ('test, 'atom, 'choice) cmd*
 | *Ch 'choice ('test, 'atom, 'choice) cmd ('test, 'atom, 'choice) cmd*
 | *Par ('test, 'atom, 'choice) cmd list*
 | *ParT ('test, 'atom, 'choice) cmd list*

fun *noWhile where*
noWhile Done $\longleftrightarrow \text{True}$
 | *noWhile (Atm atm)* $\longleftrightarrow \text{True}$
 | *noWhile (c1 ;; c2)* $\longleftrightarrow \text{noWhile } c1 \wedge \text{noWhile } c2$
 | *noWhile (While tst c)* $\longleftrightarrow \text{False}$
 | *noWhile (Ch ch c1 c2)* $\longleftrightarrow \text{noWhile } c1 \wedge \text{noWhile } c2$

| $noWhile (Par\ cl) \longleftrightarrow (\forall\ c \in\ set\ cl.\ noWhile\ c)$
| $noWhile (ParT\ cl) \longleftrightarrow (\forall\ c \in\ set\ cl.\ noWhile\ c)$

fun *finished* **where**

finished Done $\longleftrightarrow True$
| *finished (Atm atm)* $\longleftrightarrow False$
| *finished (c1 ;; c2)* $\longleftrightarrow False$
| *finished (While tst c)* $\longleftrightarrow False$
| *finished (Ch ch c1 c2)* $\longleftrightarrow False$
| *finished (Par cl)* $\longleftrightarrow (\forall\ c \in\ set\ cl.\ finished\ c)$
| *finished (ParT cl)* $\longleftrightarrow (\forall\ c \in\ set\ cl.\ finished\ c)$

definition *noWhileL* **where**

noWhileL cl $\equiv \forall\ c \in\ set\ cl.\ noWhile\ c$

lemma *fin-Par-noWhileL[simp]*:

noWhile (Par cl) $\longleftrightarrow noWhileL\ cl$
<proof>

lemma *fin-ParT-noWhileL[simp]*:

noWhile (ParT cl) $\longleftrightarrow noWhileL\ cl$
<proof>

declare *noWhile.simps(6)* [*simp del*]

declare *noWhile.simps(7)* [*simp del*]

lemma *noWhileL-intro[intro]*:

assumes $\bigwedge\ c.\ c \in\ set\ cl \implies noWhile\ c$
shows *noWhileL cl*
<proof>

lemma *noWhileL-fin[simp]*:

assumes *noWhileL cl* **and** $c \in\ set\ cl$
shows *noWhile c*
<proof>

lemma *noWhileL-update[simp]*:

assumes *cl: noWhileL cl* **and** *c': noWhile c'*
shows *noWhileL (cl[n := c'])*
<proof>

definition *finishedL* **where**

finishedL cl $\equiv \forall\ c \in\ set\ cl.\ finished\ c$

lemma *finished-Par-finishedL[simp]*:

finished (Par cl) $\longleftrightarrow finishedL\ cl$
<proof>

lemma *finished-ParT-finishedL*[simp]:
finished (ParT cl) \longleftrightarrow *finishedL* cl
 ⟨proof⟩

declare *finished.simps*(6) [simp del]
declare *finished.simps*(7) [simp del]

lemma *finishedL-intro*[intro]:
assumes $\bigwedge c. c \in \text{set } cl \implies \text{finished } c$
shows *finishedL* cl
 ⟨proof⟩

lemma *finishedL-finished*[simp]:
assumes *finishedL* cl **and** $c \in \text{set } cl$
shows *finished* c
 ⟨proof⟩

lemma *finishedL-update*[simp]:
assumes cl: *finishedL* cl **and** c': *finished* c'
shows *finishedL* (cl[n := c'])
 ⟨proof⟩

lemma *finished-fin*[simp]:
finished c \implies *noWhile* c
 ⟨proof⟩

lemma *finishedL-noWhileL*[simp]:
finishedL cl \implies *noWhileL* cl
 ⟨proof⟩

locale PL =
fixes
 aval :: 'atom \Rightarrow 'state \Rightarrow 'state **and**
 tval :: 'test \Rightarrow 'state \Rightarrow bool **and**
 cval :: 'choice \Rightarrow 'state \Rightarrow real
assumes
 properCh: $\bigwedge ch\ s. 0 \leq \text{cval } ch\ s \wedge \text{cval } ch\ s \leq 1$
begin

lemma [simp]: $(n::nat) < N \implies 0 \leq 1 / N$ ⟨proof⟩

lemma [simp]: $(n::nat) < N \implies 1 / N \leq 1$ ⟨proof⟩

lemma [simp]: $(n::nat) < N \implies 0 \leq 1 - 1 / N$ ⟨proof⟩

lemma *sum-equal*: $0 < (N::nat) \implies \text{sum } (\lambda n. 1/N) \{.. < N\} = 1$
 ⟨proof⟩

fun *proper* **where**

$proper\ Done \longleftrightarrow True$
 $|\ proper\ (Atm\ x) \longleftrightarrow True$
 $| \ proper\ (Seq\ c1\ c2) \longleftrightarrow proper\ c1 \wedge proper\ c2$
 $| \ proper\ (While\ tst\ c) \longleftrightarrow proper\ c$
 $| \ proper\ (Ch\ ch\ c1\ c2) \longleftrightarrow proper\ c1 \wedge proper\ c2$
 $| \ proper\ (Par\ cl) \longleftrightarrow cl \neq [] \wedge (\forall\ c \in set\ cl.\ proper\ c)$
 $| \ proper\ (ParT\ cl) \longleftrightarrow cl \neq [] \wedge (\forall\ c \in set\ cl.\ proper\ c)$

definition *properL where*

$properL\ cl \equiv cl \neq [] \wedge (\forall\ c \in set\ cl.\ proper\ c)$

lemma *proper-Par-properL[simp]:*

$proper\ (Par\ cl) \longleftrightarrow properL\ cl$
 $\langle proof \rangle$

lemma *proper-ParT-properL[simp]:*

$proper\ (ParT\ cl) \longleftrightarrow properL\ cl$
 $\langle proof \rangle$

declare *proper.simps(6) [simp del]*

declare *proper.simps(7) [simp del]*

lemma *properL-intro[intro]:*

$\llbracket cl \neq []; \bigwedge c. c \in set\ cl \implies proper\ c \rrbracket \implies properL\ cl$
 $\langle proof \rangle$

lemma *properL-notEmp[simp]:* $properL\ cl \implies cl \neq []$

$\langle proof \rangle$

lemma *properL-proper[simp]:*

$\llbracket properL\ cl; c \in set\ cl \rrbracket \implies proper\ c$
 $\langle proof \rangle$

lemma *properL-update[simp]:*

assumes *cl: properL cl and c': proper c'*

shows $properL\ (cl[n := c'])$

$\langle proof \rangle$

lemma *proper-induct[consumes 1, case-names Done Atm Seq While Ch Par ParT]:*

assumes **: proper c*

and *Done: phi Done*

and *Atm: $\bigwedge atm. phi\ (Atm\ atm)$*

and *Seq: $\bigwedge c1\ c2. \llbracket phi\ c1; phi\ c2 \rrbracket \implies phi\ (c1\ ;;\ c2)$*

and *While: $\bigwedge tst\ c. phi\ c \implies phi\ (While\ tst\ c)$*

and *Ch: $\bigwedge ch\ c1\ c2. \llbracket phi\ c1; phi\ c2 \rrbracket \implies phi\ (Ch\ ch\ c1\ c2)$*

and *Par: $\bigwedge cl. \llbracket properL\ cl; \bigwedge c. c \in set\ cl \implies phi\ c \rrbracket \implies phi\ (Par\ cl)$*

and *ParT: $\bigwedge cl. \llbracket properL\ cl; \bigwedge c. c \in set\ cl \implies phi\ c \rrbracket \implies phi\ (ParT\ cl)$*

shows $phi\ c$

$\langle proof \rangle$

1.2.1 Operational Small-Step Semantics

definition $theFT\ cl \equiv \{n. n < length\ cl \wedge finished\ (cl!n)\}$

definition $theNFT\ cl \equiv \{n. n < length\ cl \wedge \neg finished\ (cl!n)\}$

lemma $finite-theFT[simp]: finite\ (theFT\ cl)$
 $\langle proof \rangle$

lemma $theFT-length[simp]: n \in theFT\ cl \implies n < length\ cl$
 $\langle proof \rangle$

lemma $theFT-finished[simp]: n \in theFT\ cl \implies finished\ (cl!n)$
 $\langle proof \rangle$

lemma $finite-theNFT[simp]: finite\ (theNFT\ cl)$
 $\langle proof \rangle$

lemma $theNFT-length[simp]: n \in theNFT\ cl \implies n < length\ cl$
 $\langle proof \rangle$

lemma $theNFT-notFinished[simp]: n \in theNFT\ cl \implies \neg finished\ (cl!n)$
 $\langle proof \rangle$

lemma $theFT-Int-theNFT[simp]:$
 $theFT\ cl\ Int\ theNFT\ cl = \{\}$ **and** $theNFT\ cl\ Int\ theFT\ cl = \{\}$
 $\langle proof \rangle$

lemma $theFT-Un-theNFT[simp]:$
 $theFT\ cl\ Un\ theNFT\ cl = \{.. < length\ cl\}$ **and**
 $theNFT\ cl\ Un\ theFT\ cl = \{.. < length\ cl\}$
 $\langle proof \rangle$

lemma $in-theFT-theNFT[simp]:$
assumes $n1 \in theFT\ cl$ **and** $n2 \in theNFT\ cl$
shows $n1 \neq n2$ **and** $n2 \neq n1$
 $\langle proof \rangle$

definition $WtFT\ cl \equiv sum\ (\lambda\ (n::nat). 1/(length\ cl))\ (theFT\ cl)$

definition $WtNFT\ cl \equiv sum\ (\lambda\ (n::nat). 1/(length\ cl))\ (theNFT\ cl)$

lemma $WtFT-WtNFT[simp]:$
assumes $0 < length\ cl$
shows $WtFT\ cl + WtNFT\ cl = 1$ (**is** $?A = 1$)
 $\langle proof \rangle$

lemma *WtNFT-1-WtFT*: $0 < \text{length } cl \implies \text{WtNFT } cl = 1 - \text{WtFT } cl$
<proof>

lemma *WtNFT-WtFT-1[simp]*:
assumes $0 < \text{length } cl$ **and** $\text{WtFT } cl \neq 1$
shows $\text{WtNFT } cl / (1 - \text{WtFT } cl) = 1$ (**is** $?A / ?B = 1$)
<proof>

lemma *WtFT-ge-0[simp]*: $\text{WtFT } cl \geq 0$
<proof>

lemma *WtFT-le-1[simp]*: $\text{WtFT } cl \leq 1$ (**is** $?L \leq 1$)
<proof>

lemma *le-1-WtFT[simp]*: $0 \leq 1 - \text{WtFT } cl$ (**is** $0 \leq ?R$)
<proof>

lemma *WtFT-lt-1[simp]*: $\text{WtFT } cl \neq 1 \implies \text{WtFT } cl < 1$
<proof>

lemma *lt-1-WtFT[simp]*: $\text{WtFT } cl \neq 1 \implies 0 < 1 - \text{WtFT } cl$
<proof>

lemma *notFinished-WtFT[simp]*:
assumes $n < \text{length } cl$ **and** $\neg \text{finished } (cl ! n)$
shows $1 / \text{length } cl \leq 1 - \text{WtFT } cl$
<proof>

fun *brn* :: ('test, 'atom, 'choice) cmd \Rightarrow nat **where**
 brn Done = 1
 | *brn (Atm atm)* = 1
 | *brn (c1 ;; c2)* = *brn c1*
 | *brn (While tst c)* = 1
 | *brn (Ch ch c1 c2)* = 2
 | *brn (Par cl)* = *lsum brn cl*
 | *brn (ParT cl)* = *lsum brn cl*

lemma *brn-gt-0*: *proper c* $\implies 0 < \text{brn } c$
<proof>

lemma *brn-gt-0-L*: $\llbracket \text{properL } cl; c \in \text{set } cl \rrbracket \implies 0 < \text{brn } c$
<proof>

definition *locateT* \equiv *locate1 brn* **definition** *locateI* \equiv *locate2 brn*

definition *brnL cl n* \equiv *lsum brn (take n cl)*

lemma *brnL-lsum*: $\text{brnL } cl \ (\text{length } cl) = \text{lsum } \text{brn } cl$
<proof>

lemma *brnL-unique*:

assumes *properL* cl **and** $n1 < \text{length } cl \wedge j1 < \text{brn } (cl ! n1)$
and $n2 < \text{length } cl \wedge j2 < \text{brn } (cl ! n2)$ **and** $\text{brnL } cl \ n1 + j1 = \text{brnL } cl \ n2 + j2$
shows $n1 = n2 \wedge j1 = j2$
<proof>

lemma *brn-Par-simp[simp]*: $\text{brn } (\text{Par } cl) = \text{brnL } cl \ (\text{length } cl)$
<proof>

lemma *brn-ParT-simp[simp]*: $\text{brn } (\text{ParT } cl) = \text{brnL } cl \ (\text{length } cl)$
<proof>

declare *brn.simps(6)[simp del]* **declare** *brn.simps(7)[simp del]*

lemma *brnL-0[simp]*: $\text{brnL } cl \ 0 = 0$
<proof>

lemma *brnL-Suc[simp]*: $n < \text{length } cl \implies \text{brnL } cl \ (\text{Suc } n) = \text{brnL } cl \ n + \text{brn } (cl ! n)$
<proof>

lemma *brnL-mono[simp]*: $n1 \leq n2 \implies \text{brnL } cl \ n1 \leq \text{brnL } cl \ n2$
<proof>

lemma *brnL-mono2[simp]*:
assumes p : *properL* cl **and** n : $n1 < n2$ **and** l : $n2 \leq \text{length } cl$
shows $\text{brnL } cl \ n1 < \text{brnL } cl \ n2$ (**is** $?L < ?R$)
<proof>

lemma *brn-index[simp]*:
assumes n : $n < \text{length } cl$ **and** i : $i < \text{brn } (cl ! n)$
shows $\text{brnL } cl \ n + i < \text{brnL } cl \ (\text{length } cl)$ (**is** $?L < ?R$)
<proof>

lemma *brnL-gt-0[simp]*: $\llbracket \text{properL } cl; 0 < n \rrbracket \implies 0 < \text{brnL } cl \ n$
<proof>

lemma *locateTI*:

assumes *properL* cl **and** $ii < \text{brn } (\text{Par } cl)$
shows
 $\text{locateT } cl \ ii < \text{length } cl \wedge$
 $\text{locateI } cl \ ii < \text{brn } (cl ! (\text{locateT } cl \ ii)) \wedge$
 $ii = \text{brnL } cl \ (\text{locateT } cl \ ii) + \text{locateI } cl \ ii$
<proof>

lemma *locateTI-unique*:

assumes $\text{properL } cl$ **and** $n < \text{length } cl$
and $i < \text{brn } (cl ! n)$ **and** $ii = \text{brnL } cl \ n + i$
shows $n = \text{locateT } cl \ ii \wedge i = \text{locateI } cl \ ii$
 $\langle \text{proof} \rangle$

definition pickFT-pred **where** $\text{pickFT-pred } cl \ n \equiv n < \text{length } cl \wedge \text{finished } (cl ! n)$
definition pickFT **where** $\text{pickFT } cl \equiv \text{SOME } n. \text{pickFT-pred } cl \ n$

lemma pickFT-pred :
assumes $\text{WtFT } cl = 1$ **shows** $\exists n. \text{pickFT-pred } cl \ n$
 $\langle \text{proof} \rangle$

lemma $\text{pickFT-pred-pickFT}$: $\text{WtFT } cl = 1 \implies \text{pickFT-pred } cl \ (\text{pickFT } cl)$
 $\langle \text{proof} \rangle$

lemma $\text{pickFT-length[simp]}$: $\text{WtFT } cl = 1 \implies \text{pickFT } cl < \text{length } cl$
 $\langle \text{proof} \rangle$

lemma $\text{pickFT-finished[simp]}$: $\text{WtFT } cl = 1 \implies \text{finished } (cl ! (\text{pickFT } cl))$
 $\langle \text{proof} \rangle$

lemma $\text{pickFT-theFT[simp]}$: $\text{WtFT } cl = 1 \implies \text{pickFT } cl \in \text{theFT } cl$
 $\langle \text{proof} \rangle$

fun wt-cont-eff **where**
 $\text{wt-cont-eff } \text{Done } s \ i = (1, \text{Done}, s)$
 $|$
 $\text{wt-cont-eff } (\text{Atm } atm) \ s \ i = (1, \text{Done}, \text{aval } atm \ s)$
 $|$
 $\text{wt-cont-eff } (c1 \ ;\ ; \ c2) \ s \ i =$
 $(\text{case } \text{wt-cont-eff } c1 \ s \ i \ \text{of}$
 $(x, c1', s') \Rightarrow$
 $\text{if } \text{finished } c1' \ \text{then } (x, c2, s') \ \text{else } (x, c1' \ ;\ ; \ c2, s'))$
 $|$
 $\text{wt-cont-eff } (\text{While } \text{tst } c) \ s \ i =$
 $(\text{if } \text{tval } \text{tst } s$
 $\text{then } (1, c \ ;\ ; \ (\text{While } \text{tst } c), s)$
 $\text{else } (1, \text{Done}, s))$
 $|$
 $\text{wt-cont-eff } (\text{Ch } ch \ c1 \ c2) \ s \ i =$
 $(\text{if } i = 0 \ \text{then } \text{cval } ch \ s \ \text{else } 1 - \text{cval } ch \ s,$
 $\text{if } i = 0 \ \text{then } c1 \ \text{else } c2,$
 $s)$
 $|$
 $\text{wt-cont-eff } (\text{Par } cl) \ s \ ii =$
 $(\text{if } cl ! (\text{locateT } cl \ ii) \in \text{set } cl \ \text{then}$
 $(\text{case } \text{wt-cont-eff}$

```

      (cl ! (locateT cl ii))
      s
      (locateI cl ii) of
(w, c', s') ⇒
  ((1 / (length cl)) * w,
   Par (cl [(locateT cl ii) := c']),
   s')
else undefined)
|
wt-cont-eff (ParT cl) s ii =
  (if cl ! (locateT cl ii) ∈ set cl
   then
     (case wt-cont-eff
        (cl ! (locateT cl ii))
        s
        (locateI cl ii) of
      (w, c', s') ⇒
        (if WtFT cl = 1
         then (if locateT cl ii = pickFT cl ∧ locateI cl ii = 0
              then 1
              else 0)
         else if finished (cl ! (locateT cl ii))
              then 0
              else (1 / (length cl))
                   / (1 - WtFT cl)
                   * w,
                 ParT (cl [(locateT cl ii) := c']),
                 s'))
     else undefined)

```

definition *wt where* $wt\ c\ s\ i = fst\ (wt\text{-cont-eff}\ c\ s\ i)$

definition *cont where* $cont\ c\ s\ i = fst\ (snd\ (wt\text{-cont-eff}\ c\ s\ i))$

definition *eff where* $eff\ c\ s\ i = snd\ (snd\ (wt\text{-cont-eff}\ c\ s\ i))$

lemma *wt-Done[simp]*: $wt\ Done\ s\ i = 1$
 ⟨proof⟩

lemma *wt-Atm[simp]*: $wt\ (Atm\ atm)\ s\ i = 1$
 ⟨proof⟩

lemma *wt-Seq[simp]*:
 $wt\ (c1\ ;;\ c2)\ s = wt\ c1\ s$
 ⟨proof⟩

lemma *wt-While[simp]*: $wt\ (While\ tst\ c)\ s\ i = 1$
 ⟨proof⟩

lemma *wt-Ch-L[simp]*: $wt (Ch\ ch\ c1\ c2)\ s\ 0 = cval\ ch\ s$
<proof>

lemma *wt-Ch-R[simp]*: $wt (Ch\ ch\ c1\ c2)\ s\ (Suc\ n) = 1 - cval\ ch\ s$
<proof>

lemma *cont-Done[simp]*: $cont\ Done\ s\ i = Done$
<proof>

lemma *cont-Atm[simp]*: $cont\ (Atm\ atm)\ s\ i = Done$
<proof>

lemma *cont-Seq-finished[simp]*: $finished\ (cont\ c1\ s\ i) \implies cont\ (c1\ ;;\ c2)\ s\ i = c2$
<proof>

lemma *cont-Seq-notFinished[simp]*:
assumes $\neg\ finished\ (cont\ c1\ s\ i)$
shows $cont\ (c1\ ;;\ c2)\ s\ i = (cont\ c1\ s\ i) ;;\ c2$
<proof>

lemma *cont-Seq-not-eq-finished[simp]*: $\neg\ finished\ c2 \implies \neg\ finished\ (cont\ (Seq\ c1\ c2)\ s\ i)$
<proof>

lemma *cont-While-False[simp]*: $tval\ tst\ s = False \implies cont\ (While\ tst\ c)\ s\ i = Done$
<proof>

lemma *cont-While-True[simp]*: $tval\ tst\ s = True \implies cont\ (While\ tst\ c)\ s\ i = c ;;\ (While\ tst\ c)$
<proof>

lemma *cont-Ch-L[simp]*: $cont\ (Ch\ ch\ c1\ c2)\ s\ 0 = c1$
<proof>

lemma *cont-Ch-R[simp]*: $cont\ (Ch\ ch\ c1\ c2)\ s\ (Suc\ n) = c2$
<proof>

lemma *eff-Done[simp]*: $eff\ Done\ s\ i = s$
<proof>

lemma *eff-Atm[simp]*: $eff\ (Atm\ atm)\ s\ i = aval\ atm\ s$
<proof>

lemma *eff-Seq[simp]*: $eff\ (c1\ ;;\ c2)\ s = eff\ c1\ s$
<proof>

lemma *eff-While[simp]*: $\text{eff } (\text{While } \text{tst } c) s i = s$
 $\langle \text{proof} \rangle$

lemma *eff-Ch[simp]*: $\text{eff } (\text{Ch } ch c1 c2) s i = s$
 $\langle \text{proof} \rangle$

lemma *brnL-nchotomy*:

assumes *properL cl and ii < brnL cl (length cl)*

shows $\exists n i. n < \text{length } cl \wedge i < \text{brn } (cl ! n) \wedge ii = \text{brnL } cl n + i$
 $\langle \text{proof} \rangle$

corollary *brnL-cases[consumes 2, case-names Local, elim]*:

assumes *properL cl and ii < brnL cl (length cl) and*

$\bigwedge n i. \llbracket n < \text{length } cl; i < \text{brn } (cl ! n); ii = \text{brnL } cl n + i \rrbracket \implies phi$

shows *phi*

$\langle \text{proof} \rangle$

lemma *wt-cont-eff-Par[simp]*:

assumes *p: properL cl*

and *n: n < length cl and i: i < brn (cl ! n)*

shows

$\text{wt } (\text{Par } cl) s (\text{brnL } cl n + i) =$
 $1 / (\text{length } cl) * \text{wt } (cl ! n) s i$
 $(\text{is } ?wL = ?wR)$

$\text{cont } (\text{Par } cl) s (\text{brnL } cl n + i) =$
 $\text{Par } (cl [n := \text{cont } (cl ! n) s i])$
 $(\text{is } ?mL = ?mR)$

$\text{eff } (\text{Par } cl) s (\text{brnL } cl n + i) =$
 $\text{eff } (cl ! n) s i$
 $(\text{is } ?eL = ?eR)$
 $\langle \text{proof} \rangle$

lemma *cont-eff-ParT[simp]*:

assumes *p: properL cl*

and *n: n < length cl and i: i < brn (cl ! n)*

shows

$\text{cont } (\text{ParT } cl) s (\text{brnL } cl n + i) =$
 $\text{ParT } (cl [n := \text{cont } (cl ! n) s i])$
 $(\text{is } ?mL = ?mR)$

$\text{eff } (\text{ParT } cl) s (\text{brnL } cl n + i) =$
 $\text{eff } (cl ! n) s i$
 $(\text{is } ?eL = ?eR)$
 $\langle \text{proof} \rangle$

lemma *wt-ParT-WtFT-pickFT-0[simp]*:
assumes p : *properL cl* **and** *WtFT: WtFT cl = 1*
shows $wt (ParT cl) s (brnL cl (pickFT cl)) = 1$
(is $?wL = 1$ **)**
 $\langle proof \rangle$

lemma *wt-ParT-WtFT-notPickFT-0[simp]*:
assumes p : *properL cl* **and** n : $n < length\ cl$ **and** i : $i < brn (cl\ !\ n)$
and *WtFT: WtFT cl = 1* **and** ni : $n = pickFT\ cl \longrightarrow i \neq 0$
shows $wt (ParT cl) s (brnL cl\ n + i) = 0$ **(is** $?wL = 0$ **)**
 $\langle proof \rangle$

lemma *wt-ParT-notWtFT-finished[simp]*:
assumes p : *properL cl* **and** n : $n < length\ cl$ **and** i : $i < brn (cl\ !\ n)$
and *WtFT: WtFT cl \neq 1* **and** f : *finished (cl ! n)*
shows $wt (ParT cl) s (brnL cl\ n + i) = 0$ **(is** $?wL = 0$ **)**
 $\langle proof \rangle$

lemma *wt-cont-eff-ParT-notWtFT-notFinished[simp]*:
assumes p : *properL cl* **and** n : $n < length\ cl$ **and** i : $i < brn (cl\ !\ n)$
and *WtFT: WtFT cl \neq 1* **and** nf : $\neg finished (cl\ !\ n)$
shows $wt (ParT cl) s (brnL cl\ n + i) =$
 $(1 / (length\ cl)) / (1 - WtFT\ cl) * wt (cl\ !\ n) s\ i$ **(is** $?wL = ?wR$ **)**
 $\langle proof \rangle$

lemma *wt-ge-0[simp]*:
assumes *proper c* **and** $i < brn\ c$
shows $0 \leq wt\ c\ s\ i$
 $\langle proof \rangle$

lemma *wt-le-1[simp]*:
assumes *proper c* **and** $i < brn\ c$
shows $wt\ c\ s\ i \leq 1$
 $\langle proof \rangle$

abbreviation *fromPlus* $((1\ \{..\ < +-\})$) **where**
 $\{a\ ..\ < +\ b\} \equiv \{a\ ..\ < a + b\}$

lemma *brnL-UN*:
assumes *properL cl*
shows $\{..\ < brnL\ cl\ (length\ cl)\} = (\bigcup\ n < length\ cl. \{brnL\ cl\ n\ ..\ < +\ brn\ (cl!\ n)\})$
(is $?L = (\bigcup\ n < length\ cl. ?R\ n)$ **)**
 $\langle proof \rangle$

lemma *brnL-Int-lt*:
assumes $n12$: $n1 < n2$ **and** $n2$: $n2 < length\ cl$
shows
 $\{brnL\ cl\ n1\ ..\ < +\ brn\ (cl!\ n1)\} \cap \{brnL\ cl\ n2\ ..\ < +\ brn\ (cl!\ n2)\} = \{\}$

$\langle proof \rangle$

lemma *brnL-Int*:

assumes $n1 \neq n2$ **and** $n1 < length\ cl$ **and** $n2 < length\ cl$

shows $\{brnL\ cl\ n1\ ..<+ brn\ (cl!n1)\} \cap \{brnL\ cl\ n2\ ..<+ brn\ (cl!n2)\} = \{\}$

$\langle proof \rangle$

lemma *sum-wt-Par-sub[simp]*:

assumes $cl: properL\ cl$ **and** $n: n < length\ cl$ **and** $I: I \subseteq \{..< brn\ (cl!\ n)\}$

shows $sum\ (wt\ (Par\ cl)\ s)\ ((+)\ (brnL\ cl\ n)\ 'I) =$

$$1 / (length\ cl) * sum\ (wt\ (cl!\ n)\ s)\ I\ (\mathbf{is}\ ?L = ?wSch * ?R)$$

$\langle proof \rangle$

lemma *sum-wt-Par[simp]*:

assumes $cl: properL\ cl$ **and** $n: n < length\ cl$

shows $sum\ (wt\ (Par\ cl)\ s)\ \{brnL\ cl\ n\ ..<+ brn\ (cl!n)\} =$

$$1 / (length\ cl) * sum\ (wt\ (cl!\ n)\ s)\ \{..< brn\ (cl!\ n)\}\ (\mathbf{is}\ ?L = ?W * ?R)$$

$\langle proof \rangle$

lemma *sum-wt-ParT-sub-WtFT-pickFT-0[simp]*:

assumes $cl: properL\ cl$ **and** $nf: WtFT\ cl = 1$

and $I: I \subseteq \{..< brn\ (cl!\ (pickFT\ cl))\}\ 0 \in I$

shows $sum\ (wt\ (ParT\ cl)\ s)\ ((+)\ (brnL\ cl\ (pickFT\ cl))\ 'I) = 1\ (\mathbf{is}\ ?L = 1)$

$\langle proof \rangle$

lemma *sum-wt-ParT-sub-WtFT-pickFT-0-2[simp]*:

assumes $cl: properL\ cl$ **and** $nf: WtFT\ cl = 1$

and $II: II \subseteq \{..< brnL\ cl\ (length\ cl)\}\ brnL\ cl\ (pickFT\ cl) \in II$

shows $sum\ (wt\ (ParT\ cl)\ s)\ II = 1\ (\mathbf{is}\ ?L = 1)$

$\langle proof \rangle$

lemma *sum-wt-ParT-sub-WtFT-notPickFT-0[simp]*:

assumes $cl: properL\ cl$ **and** $nf: WtFT\ cl = 1$ **and** $n: n < length\ cl$

and $I: I \subseteq \{..< brn\ (cl!\ n)\}$ **and** $nI: n = pickFT\ cl \longrightarrow 0 \notin I$

shows $sum\ (wt\ (ParT\ cl)\ s)\ ((+)\ (brnL\ cl\ n)\ 'I) = 0\ (\mathbf{is}\ ?L = 0)$

$\langle proof \rangle$

lemma *sum-wt-ParT-sub-notWtFT-finished[simp]*:

assumes $cl: properL\ cl$ **and** $nf: WtFT\ cl \neq 1$

and $n: n < length\ cl$ **and** $cln: finished\ (cl!n)$ **and** $I: I \subseteq \{..< brn\ (cl!\ n)\}$

shows $sum\ (wt\ (ParT\ cl)\ s)\ ((+)\ (brnL\ cl\ n)\ 'I) = 0\ (\mathbf{is}\ ?L = 0)$

$\langle proof \rangle$

lemma *sum-wt-ParT-sub-notWtFT-notFinished[simp]*:

assumes $cl: properL\ cl$ **and** $nf: WtFT\ cl \neq 1$ **and** $n: n < length\ cl$

and $cln: \neg finished\ (cl!n)$ **and** $I: I \subseteq \{..< brn\ (cl!\ n)\}$

shows

$sum\ (wt\ (ParT\ cl)\ s)\ ((+)\ (brnL\ cl\ n)\ 'I) =$

$$(1 / (length\ cl)) / (1 - WtFT\ cl) * sum\ (wt\ (cl!\ n)\ s)\ I$$

(is ?L = ?w / (1 - ?wF) * ?R)
 <proof>

lemma *sum-wt-ParT-WtFT-pickFT-0*[simp]:
assumes *cl: properL cl and nf: WtFT cl = 1*
shows $\text{sum } (wt \ (ParT \ cl) \ s) \ \{brnL \ cl \ (pickFT \ cl) \ ..<+ \ brn \ (cl \ ! \ (pickFT \ cl))\} = 1$
 <proof>

lemma *sum-wt-ParT-WtFT-notPickFT-0*[simp]:
assumes *cl: properL cl and nf: WtFT cl = 1 and n: n < length cl n ≠ pickFT cl*
shows $\text{sum } (wt \ (ParT \ cl) \ s) \ \{brnL \ cl \ n \ ..<+ \ brn \ (cl!n)\} = 0$
 <proof>

lemma *sum-wt-ParT-notWtFT-finished*[simp]:
assumes *cl: properL cl and WtFT cl ≠ 1*
and *n: n < length cl and cln: finished (cl!n)*
shows $\text{sum } (wt \ (ParT \ cl) \ s) \ \{brnL \ cl \ n \ ..<+ \ brn \ (cl!n)\} = 0$
 <proof>

lemma *sum-wt-ParT-notWtFT-notFinished*[simp]:
assumes *cl: properL cl and nf: WtFT cl ≠ 1*
and *n: n < length cl and cln: ¬ finished (cl!n)*
shows
 $\text{sum } (wt \ (ParT \ cl) \ s) \ \{brnL \ cl \ n \ ..<+ \ brn \ (cl!n)\} =$
 $(1 / (\text{length } cl)) / (1 - WtFT \ cl) * \text{sum } (wt \ (cl \ ! \ n) \ s) \ \{..< \ brn \ (cl \ ! \ n)\}$
 <proof>

lemma *sum-wt*[simp]:
assumes *proper c*
shows $\text{sum } (wt \ c \ s) \ \{..< \ brn \ c\} = 1$
 <proof>

lemma *proper-cont*[simp]:
assumes *proper c and i < brn c*
shows *proper (cont c s i)*
 <proof>

lemma *sum-subset-le-1*[simp]:
assumes **: proper c and **: I ⊆ {..< brn c}*
shows $\text{sum } (wt \ c \ s) \ I \leq 1$
 <proof>

lemma *sum-le-1*[simp]:
assumes **: proper c and **: i < brn c*
shows $\text{sum } (wt \ c \ s) \ \{..i\} \leq 1$
 <proof>

1.2.2 Operations on configurations

definition $cont\text{-}eff\ cf\ b = snd\ (wt\text{-}cont\text{-}eff\ (fst\ cf)\ (snd\ cf)\ b)$

lemma $cont\text{-}eff$: $cont\text{-}eff\ cf\ b = (cont\ (fst\ cf)\ (snd\ cf)\ b, eff\ (fst\ cf)\ (snd\ cf)\ b)$
 $\langle proof \rangle$

end

end

2 Resumption-Based Noninterference

theory *Resumption-Based*
imports *Language-Semantics*
begin

type-synonym $'a\ rel = ('a \times 'a)\ set$

2.1 Preliminaries

lemma $int\text{-}emp[simp]$:
assumes $i > 0$
shows $\{..<i\} \neq \{\}$
 $\langle proof \rangle$

lemma $inj\text{-}on\text{-}inv\text{-}into[simp]$:
assumes $inj\text{-}on\ F\ P$
shows $inv\text{-}into\ P\ F\ ' (F\ ' P) = P$
 $\langle proof \rangle$

lemma $inj\text{-}on\text{-}inv\text{-}into2[simp]$:
 $inj\text{-}on\ (inv\text{-}into\ P\ F)\ (F\ ' P)$
 $\langle proof \rangle$

lemma $refl\text{-}gfp$:
assumes $1: mono\ Retr$ **and** $2: \bigwedge\ theta. refl\ theta \implies refl\ (Retr\ theta)$
shows $refl\ (gfp\ Retr)$
 $\langle proof \rangle$

lemma $sym\text{-}gfp$:
assumes $1: mono\ Retr$ **and** $2: \bigwedge\ theta. sym\ theta \implies sym\ (Retr\ theta)$
shows $sym\ (gfp\ Retr)$
 $\langle proof \rangle$

lemma $trancl\text{-}trans[simp]$:

assumes *trans R*
shows $P \hat{+} \subseteq R \longleftrightarrow P \subseteq R$
 $\langle proof \rangle$

lemma *trans-gfp*:
assumes 1: *mono Retr* **and** 2: $\bigwedge theta. trans\ theta \implies trans\ (Retr\ theta)$
shows *trans (gfp Retr)*
 $\langle proof \rangle$

lemma *O-subset-trans*:
assumes $r\ O\ r \subseteq r$
shows *trans r*
 $\langle proof \rangle$

lemma *trancl-imp-trans*:
assumes $r \hat{+} \subseteq r$
shows *trans r*
 $\langle proof \rangle$

lemma *sym-trans-gfp*:
assumes 1: *mono Retr* **and** 2: $\bigwedge theta. sym\ theta \wedge trans\ theta \implies sym\ (Retr\ theta) \wedge trans\ (Retr\ theta)$
shows $sym\ (gfp\ Retr) \wedge trans\ (gfp\ Retr)$
 $\langle proof \rangle$

2.2 Infrastructure for partitions

definition *part where*
 $part\ J\ P \equiv$
 $Union\ P = J \wedge$
 $(\forall J1\ J2. J1 \in P \wedge J2 \in P \wedge J1 \neq J2 \longrightarrow J1 \cap J2 = \{\})$

inductive-set *gen*
for $P :: 'a\ set\ set$ **and** $I :: 'a\ set$ **where**
 $incl[simp]: i \in I \implies i \in gen\ P\ I$
 $ext[simp]: \llbracket J \in P; j0 \in J; j0 \in gen\ P\ I; j \in J \rrbracket \implies j \in gen\ P\ I$

definition *partGen where*
 $partGen\ P \equiv \{gen\ P\ I \mid I. I \in P\}$

definition *finer where*
 $finer\ P\ Q \equiv$
 $(\forall J \in Q. J = Union\ \{I \in P. I \subseteq J\}) \wedge$
 $(P \neq \{\} \longrightarrow Q \neq \{\})$

definition *partJoin* **where**
partJoin $P Q \equiv \text{partGen } (P \cup Q)$

definition *compat* **where**
compat $I \text{ theta } f \equiv \forall i j. \{i, j\} \subseteq I \wedge i \neq j \longrightarrow (f i, f j) \in \text{theta}$

definition *partCompat* **where**
partCompat $P \text{ theta } f \equiv$
 $\forall I \in P. \text{compat } I \text{ theta } f$

definition *lift* **where**
lift $P F II \equiv \text{Union } \{F I \mid I . I \in P \wedge I \subseteq II\}$

part:

lemma *part-emp[simp]*:
part $J (\text{insert } \{ \} P) = \text{part } J P$
 $\langle \text{proof} \rangle$

lemma *finite-part[simp]*:
assumes *finite* I **and** *part* $I P$
shows *finite* P
 $\langle \text{proof} \rangle$

lemma *part-sum*:
assumes $P: \text{part } \{..<n::\text{nat}\} P$
shows $(\sum i<n. f i) = (\sum p \in P. \sum i \in p. f i)$
 $\langle \text{proof} \rangle$

lemma *part-Un[simp]*:
assumes *part* $I1 P1$ **and** *part* $I2 P2$ **and** $I1 \text{ Int } I2 = \{ \}$
shows *part* $(I1 \text{ Un } I2) (P1 \text{ Un } P2)$
 $\langle \text{proof} \rangle$

lemma *part-Un-singl[simp]*:
assumes *part* $K P$ **and** $\bigwedge I. I \in P \implies I0 \text{ Int } I = \{ \}$
shows *part* $(I0 \text{ Un } K) (\{I0\} \text{ Un } P)$
 $\langle \text{proof} \rangle$

lemma *part-Un-singl2*:
assumes $K01 = I0 \text{ Un } K1$
and *part* $K1 P$ **and** $\bigwedge I. I \in P \implies I0 \text{ Int } I = \{ \}$
shows *part* $K01 (\{I0\} \text{ Un } P)$
 $\langle \text{proof} \rangle$

lemma *part-UN*:
assumes $\bigwedge n. n \in N \implies \text{part } (I n) (P n)$

and $\bigwedge n1\ n2. \{n1,n2\} \subseteq N \wedge n1 \neq n2 \implies I\ n1 \cap I\ n2 = \{\}$
shows $part\ (UN\ n : N. I\ n)\ (UN\ n : N. P\ n)$
 $\langle proof \rangle$

gen:

lemma *incl-gen[simp]*:
 $I \subseteq gen\ P\ I$
 $\langle proof \rangle$

lemma *gen-incl-Un*:
 $gen\ P\ I \subseteq I \cup (Union\ P)$
 $\langle proof \rangle$

lemma *gen-incl*:
assumes $I \in P$
shows $gen\ P\ I \subseteq Union\ P$
 $\langle proof \rangle$

lemma *finite-gen*:
assumes *finite* P **and** $\bigwedge J. J \in P \implies finite\ J$ **and** *finite* I
shows *finite* $(gen\ P\ I)$
 $\langle proof \rangle$

lemma *subset-gen[simp]*:
assumes $J \in P$ **and** $gen\ P\ I \cap J \neq \{\}$
shows $J \subseteq gen\ P\ I$
 $\langle proof \rangle$

lemma *gen-subset-gen[simp]*:
assumes $J \in P$ **and** $gen\ P\ I \cap J \neq \{\}$
shows $gen\ P\ J \subseteq gen\ P\ I$
 $\langle proof \rangle$

lemma *gen-mono[simp]*:
assumes $I \subseteq J$
shows $gen\ P\ I \subseteq gen\ P\ J$
 $\langle proof \rangle$

lemma *gen-idem[simp]*:
 $gen\ P\ (gen\ P\ I) = gen\ P\ I$
 $\langle proof \rangle$

lemma *gen-nchotomy*:
assumes $J \in P$
shows $J \subseteq gen\ P\ I \vee gen\ P\ I \cap J = \{\}$
 $\langle proof \rangle$

lemma *gen-Union*:
assumes $I \in P$

shows $gen\ P\ I = Union\ \{J \in P . J \subseteq gen\ P\ I\}$
 $\langle proof \rangle$

lemma *subset-gen2*:
assumes $*$: $\{I, J\} \subseteq P$ **and** $**$: $gen\ P\ I \cap gen\ P\ J \neq \{\}$
shows $I \subseteq gen\ P\ J$
 $\langle proof \rangle$

lemma *gen-subset-gen2[simp]*:
assumes $\{I, J\} \subseteq P$ **and** $gen\ P\ I \cap gen\ P\ J \neq \{\}$
shows $gen\ P\ I \subseteq gen\ P\ J$
 $\langle proof \rangle$

lemma *gen-eq-gen*:
assumes $\{I, J\} \subseteq P$ **and** $gen\ P\ I \cap gen\ P\ J \neq \{\}$
shows $gen\ P\ I = gen\ P\ J$
 $\langle proof \rangle$

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

lemma *gen-empty2[simp]*:
 $gen\ \{\} I = I$
 $\langle proof \rangle$

lemma *emp-gen[simp]*:
assumes $gen\ P\ I = \{\}$
shows $I = \{\}$
 $\langle proof \rangle$

partGen:

lemma *partGen-ex*:
assumes $I \in P$
shows $\exists J \in partGen\ P. I \subseteq J$
 $\langle proof \rangle$

lemma *ex-partGen*:
assumes $J \in partGen\ P$ **and** $j: j \in J$
shows $\exists I \in P. j \in I$
 $\langle proof \rangle$

lemma *Union-partGen*: $\bigcup partGen\ P = \bigcup P$
 $\langle proof \rangle$

lemma *Int-partGen*:
assumes $*$: $\{I, J\} \subseteq partGen\ P$ **and** $**$: $I \cap J \neq \{\}$
shows $I = J$
 $\langle proof \rangle$

lemma *part-partGen*:
part (*Union P*) (*partGen P*)
<*proof*>

lemma *finite-partGen[simp]*:
assumes *finite P*
shows *finite* (*partGen P*)
<*proof*>

lemma *emp-partGen[simp]*:
assumes $\{\} \notin P$
shows $\{\} \notin \text{partGen } P$
<*proof*>

finer:

lemma *finer-partGen*:
finer P (*partGen P*)
<*proof*>

lemma *finer-nchotomy*:
assumes *P: part I0 P and Q: part I0 Q and PQ: finer P Q*
and *I: I ∈ P and II: II ∈ Q*
shows $I \subseteq II \vee (I \cap II = \{\})$
<*proof*>

lemma *finer-ex*:
assumes *P: part I0 P and Q: part I0 Q and PQ: finer P Q*
and *I: I ∈ P*
shows $\exists II. II \in Q \wedge I \subseteq II$
<*proof*>

partJoin:

lemma *partJoin-commute*:
partJoin P Q = partJoin Q P
<*proof*>

lemma *Union-partJoin-L*:
 $Union P \subseteq Union (\text{partJoin } P Q)$
<*proof*>

lemma *Union-partJoin-R*:
 $Union Q \subseteq Union (\text{partJoin } P Q)$
<*proof*>

lemma *part-partJoin[simp]*:
assumes *part I P and part I Q*
shows *part I* (*partJoin P Q*)
<*proof*>

lemma *finer-partJoin-L[simp]*:
assumes *: *part I P* **and** **: *part I Q*
shows *finer P (partJoin P Q)*
 \langle *proof* \rangle

lemma *finer-partJoin-R[simp]*:
assumes *: *part I P* **and** **: *part I Q*
shows *finer Q (partJoin P Q)*
 \langle *proof* \rangle

lemma *finer-emp[simp]*:
assumes *finer {} Q*
shows $Q \subseteq \{ \{ \} \}$
 \langle *proof* \rangle

compat:

lemma *part-emp-R[simp]*:
part I {} \longleftrightarrow I = {}
 \langle *proof* \rangle

lemma *part-emp-L[simp]*:
part {} P \implies P \subseteq { {} }
 \langle *proof* \rangle

lemma *finite-partJoin[simp]*:
assumes *finite P* **and** *finite Q*
shows *finite (partJoin P Q)*
 \langle *proof* \rangle

lemma *emp-partJoin[simp]*:
assumes $\{ \} \notin P$ **and** $\{ \} \notin Q$
shows $\{ \} \notin \text{partJoin } P \ Q$
 \langle *proof* \rangle

partCompat:

lemma *partCompat-Un[simp]*:
partCompat (P Un Q) theta f \longleftrightarrow
partCompat P theta f \wedge partCompat Q theta f
 \langle *proof* \rangle

lemma *partCompat-gen-aux*:
assumes *theta: sym theta trans theta*
and *fP: partCompat P theta f* **and** *I: I \in P*
and *i: i \in I* **and** *j: j \in gen P I* **and** *ij: i \neq j*
shows $(f \ i, f \ j) \in \text{theta}$
 \langle *proof* \rangle

lemma *partCompat-gen*:

assumes θ : *sym* θ *trans* θ
and fP : *partCompat* P θ f **and** I : $I \in P$
shows *compat* (*gen* P I) θ f
 \langle *proof* \rangle

lemma *partCompat-partGen*:
assumes *sym* θ **and** *trans* θ
and *partCompat* P θ f
shows *partCompat* (*partGen* P) θ f
 \langle *proof* \rangle

lemma *partCompat-partJoin*[*simp*]:
assumes *sym* θ **and** *trans* θ
and *partCompat* P θ f **and** *partCompat* Q θ f
shows *partCompat* (*partJoin* P Q) θ f
 \langle *proof* \rangle

lift:

lemma *inj-on-lift*:
assumes P : *part IO* P **and** Q : *part IO* Q **and** PQ : *finer* P Q
and F : *inj-on* F P **and** FP : *part JO* ($F \text{ ' } P$) **and** emp : $\{\} \notin F \text{ ' } P$
shows *inj-on* (*lift* P F) Q
 \langle *proof* \rangle

lemma *part-lift*:
assumes P : *part IO* P **and** Q : *part IO* Q **and** PQ : *finer* P Q
and F : *inj-on* F P **and** FP : *part JO* ($F \text{ ' } P$) **and** emp : $\{\} \notin P$ $\{\} \notin F \text{ ' } P$
shows *part JO* (*lift* P $F \text{ ' } Q$)
 \langle *proof* \rangle

lemma *finer-lift*:
assumes *finer* P Q
shows *finer* ($F \text{ ' } P$) (*lift* P $F \text{ ' } Q$)
 \langle *proof* \rangle

2.3 Basic setting for bisimilarity

locale *PL-Indis* =
PL aval tval cval
for $\text{aval} :: \text{'atom} \Rightarrow \text{'state} \Rightarrow \text{'state}$ **and**
 $\text{tval} :: \text{'test} \Rightarrow \text{'state} \Rightarrow \text{bool}$ **and**
 $\text{cval} :: \text{'choice} \Rightarrow \text{'state} \Rightarrow \text{real} +$
fixes
 $\text{indis} :: \text{'state} \text{ rel}$
assumes
equiv-indis: *equiv UNIV indis*

context *PL-Indis*
begin

abbreviation *indisAbbrev* (**infix** \approx 50)
where $s1 \approx s2 \equiv (s1, s2) \in indis$

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*[*trans*]: $\llbracket s \approx s'; s' \approx s'' \rrbracket \implies s \approx s''$
(*proof*)

lemma *indis-sym*[*sym*]: $s \approx s' \implies s' \approx s$
(*proof*)

2.4 Discreetness

coinductive *discr* **where**

intro:
 $(\bigwedge s i. i < brn\ c \longrightarrow s \approx eff\ c\ s\ i \wedge discr\ (cont\ c\ s\ i))$
 $\implies discr\ c$

definition *discrL* **where**

$discrL\ cl \equiv \forall\ c \in set\ cl. discr\ c$

lemma *discrL-intro*[*intro*]:
assumes $\bigwedge c. c \in set\ cl \implies discr\ c$
shows *discrL cl*
(*proof*)

lemma *discrL-discr*[*simp*]:
assumes *discrL cl* **and** $c \in set\ cl$
shows *discr c*
(*proof*)

lemma *discrL-update*[*simp*]:
assumes *cl: discrL cl* **and** $c': discr\ c'$
shows *discrL (cl[n := c'])*
(*proof*)

Coinduction for discreetness:

lemma *discr-coind*[*consumes 1, case-names Hyp, induct pred: discr*]:
assumes *: *phi c* **and**
**: $\bigwedge c\ s\ i. \llbracket phi\ c ; i < brn\ c \rrbracket$

$\implies s \approx \text{eff } c \ s \ i \wedge (\text{phi } (\text{cont } c \ s \ i) \vee \text{discr } (\text{cont } c \ s \ i))$
shows $\text{discr } c$
 $\langle \text{proof} \rangle$

lemma discr-raw-coind [$\text{consumes } 1, \text{case-names } \text{Hyp}$]:
assumes *: $\text{phi } c$ **and**
 **: $\bigwedge c \ s \ i. \llbracket i < \text{brn } c; \text{phi } c \rrbracket \implies s \approx \text{eff } c \ s \ i \wedge \text{phi } (\text{cont } c \ s \ i)$
shows $\text{discr } c$
 $\langle \text{proof} \rangle$

Discreetness versus transition:

lemma discr-cont [simp]:
assumes *: $\text{discr } c$ **and** **: $i < \text{brn } c$
shows $\text{discr } (\text{cont } c \ s \ i)$
 $\langle \text{proof} \rangle$

lemma discr-eff-indis [simp]:
assumes *: $\text{discr } c$ **and** **: $i < \text{brn } c$
shows $s \approx \text{eff } c \ s \ i$
 $\langle \text{proof} \rangle$

2.5 Self-isomorphism

coinductive siso **where**

intro :

$\llbracket \bigwedge s \ t \ i. i < \text{brn } c \implies \text{siso } (\text{cont } c \ s \ i);$
 $\bigwedge s \ t \ i.$
 $i < \text{brn } c \wedge s \approx t \implies$
 $\text{eff } c \ s \ i \approx \text{eff } c \ t \ i \wedge \text{wt } c \ s \ i = \text{wt } c \ t \ i \wedge \text{cont } c \ s \ i = \text{cont } c \ t \ i \rrbracket$
 $\implies \text{siso } c$

definition sisoL **where**

$\text{sisoL } cl \equiv \forall c \in \text{set } cl. \text{siso } c$

lemma sisoL-intro [intro]:
assumes $\bigwedge c. c \in \text{set } cl \implies \text{siso } c$
shows $\text{sisoL } cl$
 $\langle \text{proof} \rangle$

lemma sisoL-siso [simp]:
assumes $\text{sisoL } cl$ **and** $c \in \text{set } cl$
shows $\text{siso } c$
 $\langle \text{proof} \rangle$

lemma sisoL-update [simp]:
assumes $cl: \text{sisoL } cl$ **and** $c': \text{siso } c'$
shows $\text{sisoL } (cl[n := c'])$
 $\langle \text{proof} \rangle$

Coinduction for self-isomorphism:

lemma *siso-coind*[*consumes 1, case-names Obs Cont, induct pred: siso*]:
assumes *: *phi c and*
 **: $\bigwedge c s t i. \llbracket i < \text{brn } c; \text{phi } c; s \approx t \rrbracket \implies$
 $\text{eff } c s i \approx \text{eff } c t i \wedge \text{wt } c s i = \text{wt } c t i \wedge \text{cont } c s i = \text{cont } c t i$ **and**
 ***: $\bigwedge c s i. \llbracket i < \text{brn } c; \text{phi } c \rrbracket \implies \text{phi } (\text{cont } c s i) \vee \text{siso } (\text{cont } c s i)$
shows *siso c*
 ⟨*proof*⟩

lemma *siso-raw-coind*[*consumes 1, case-names Obs Cont*]:
assumes *: *phi c and*
 ***: $\bigwedge c s t i. \llbracket i < \text{brn } c; \text{phi } c; s \approx t \rrbracket \implies$
 $\text{eff } c s i \approx \text{eff } c t i \wedge \text{wt } c s i = \text{wt } c t i \wedge \text{cont } c s i = \text{cont } c t i$ **and**
 **: $\bigwedge c s i. \llbracket i < \text{brn } c; \text{phi } c \rrbracket \implies \text{phi } (\text{cont } c s i)$
shows *siso c*
 ⟨*proof*⟩

Self-Isomorphism versus transition:

lemma *siso-cont*[*simp*]:
assumes *: *siso c and* **: *i < brn c*
shows *siso (cont c s i)*
 ⟨*proof*⟩

lemma *siso-cont-indis*[*simp*]:
assumes *: *siso c and* **: *s ≈ t i < brn c*
shows *eff c s i ≈ eff c t i ∧ wt c s i = wt c t i ∧ cont c s i = cont c t i*
 ⟨*proof*⟩

2.6 Notions of bisimilarity

Matchers

definition *mC-C-part where*
mC-C-part c d P F \equiv
 $\{\} \notin P \wedge \{\} \notin F \text{ ' } P \wedge$
 $\text{part } \{.. < \text{brn } c\} P \wedge \text{part } \{.. < \text{brn } d\} (F \text{ ' } P)$

definition *mC-C-wt where*
mC-C-wt c d s t P F $\equiv \forall I \in P. \text{sum } (\text{wt } c s) I = \text{sum } (\text{wt } d t) (F I)$

definition *mC-C-eff-cont where*
mC-C-eff-cont theta c d s t P F \equiv
 $\forall I i j.$
 $I \in P \wedge i \in I \wedge j \in F I \implies$
 $\text{eff } c s i \approx \text{eff } d t j \wedge (\text{cont } c s i, \text{cont } d t j) \in \text{theta}$

definition *mC-C where*
mC-C theta c d s t P F \equiv
 $\text{mC-C-part } c d P F \wedge \text{inj-on } F P \wedge \text{mC-C-wt } c d s t P F \wedge \text{mC-C-eff-cont theta}$
 $c d s t P F$

definition *matchC-C* **where**

$$\text{matchC-C } \theta \text{ c d} \equiv \forall s t. s \approx t \longrightarrow (\exists P F. \text{mC-C } \theta \text{ c d s t P F})$$

definition *mC-ZOC-part* **where**

$$\begin{aligned} \text{mC-ZOC-part } \text{c d s t I0 P F} &\equiv \\ \{\} \notin P - \{\text{I0}\} \wedge \{\} \notin F' (P - \{\text{I0}\}) \wedge \text{I0} \in P \wedge \\ \text{part } \{..< \text{brn c}\} P \wedge \text{part } \{..< \text{brn d}\} (F' P) \end{aligned}$$

definition *mC-ZOC-wt* **where**

$$\begin{aligned} \text{mC-ZOC-wt } \text{c d s t I0 P F} &\equiv \\ \text{sum } (\text{wt c s}) \text{ I0} < 1 \wedge \text{sum } (\text{wt d t}) (F \text{ I0}) < 1 \longrightarrow \\ (\forall I \in P - \{\text{I0}\}. \\ \text{sum } (\text{wt c s}) I / (1 - \text{sum } (\text{wt c s}) \text{ I0}) = \\ \text{sum } (\text{wt d t}) (F I) / (1 - \text{sum } (\text{wt d t}) (F \text{ I0}))) \end{aligned}$$

definition *mC-ZOC-eff-cont0* **where**

$$\begin{aligned} \text{mC-ZOC-eff-cont0 } \theta \text{ c d s t I0 F} &\equiv \\ (\forall i \in \text{I0}. s \approx \text{eff c s i} \wedge (\text{cont c s i}, d) \in \theta) \wedge \\ (\forall j \in F \text{ I0}. t \approx \text{eff d t j} \wedge (c, \text{cont d t j}) \in \theta) \end{aligned}$$

definition *mC-ZOC-eff-cont* **where**

$$\begin{aligned} \text{mC-ZOC-eff-cont } \theta \text{ c d s t I0 P F} &\equiv \\ \forall I i j. \\ I \in P - \{\text{I0}\} \wedge i \in I \wedge j \in F I \longrightarrow \\ \text{eff c s i} \approx \text{eff d t j} \wedge \\ (\text{cont c s i}, \text{cont d t j}) \in \theta \end{aligned}$$

definition *mC-ZOC* **where**

$$\begin{aligned} \text{mC-ZOC } \theta \text{ c d s t I0 P F} &\equiv \\ \text{mC-ZOC-part } \text{c d s t I0 P F} \wedge \\ \text{inj-on } F P \wedge \\ \text{mC-ZOC-wt } \text{c d s t I0 P F} \wedge \\ \text{mC-ZOC-eff-cont0 } \theta \text{ c d s t I0 F} \wedge \\ \text{mC-ZOC-eff-cont } \theta \text{ c d s t I0 P F} \end{aligned}$$

definition *matchC-LC* **where**

$$\begin{aligned} \text{matchC-LC } \theta \text{ c d} &\equiv \\ \forall s t. s \approx t \longrightarrow (\exists \text{I0 P F}. \text{mC-ZOC } \theta \text{ c d s t I0 P F}) \end{aligned}$$

lemmas *m-defs* = *mC-C-def* *mC-ZOC-def*

lemmas *m-defsAll* =

$$\begin{aligned} \text{mC-C-def } \text{mC-C-part-def } \text{mC-C-wt-def } \text{mC-C-eff-cont-def} \\ \text{mC-ZOC-def } \text{mC-ZOC-part-def } \text{mC-ZOC-wt-def } \text{mC-ZOC-eff-cont0-def } \text{mC-ZOC-eff-cont-def} \end{aligned}$$

lemmas *match-defs* =

$$\text{matchC-C-def } \text{matchC-LC-def}$$

lemma *mC-C-mono*:
assumes *mC-C theta c d s t P F* **and** *theta ⊆ theta'*
shows *mC-C theta' c d s t P F*
⟨*proof*⟩

lemma *matchC-C-mono*:
assumes *matchC-C theta c d* **and** *theta ⊆ theta'*
shows *matchC-C theta' c d*
⟨*proof*⟩

lemma *mC-ZOC-mono*:
assumes *mC-ZOC theta c d s t I0 P F* **and** *theta ⊆ theta'*
shows *mC-ZOC theta' c d s t I0 P F*
⟨*proof*⟩

lemma *matchC-LC-mono*:
assumes *matchC-LC theta c d* **and** *theta ⊆ theta'*
shows *matchC-LC theta' c d*
⟨*proof*⟩

lemma *Int-not-in-eq-emp*:
 $P \cap \{I. I \notin P\} = \{\}$
⟨*proof*⟩

lemma *mC-C-mC-ZOC*:
assumes *mC-C theta c d s t P F*
shows *mC-ZOC theta c d s t {} (P Un { {} })* (%*I. if I ∈ P then F I else {}*)
(is *mC-ZOC theta c d s t ?I0 ?Q ?G*)
⟨*proof*⟩

lemma *matchC-C-matchC-LC*:
assumes *matchC-C theta c d*
shows *matchC-LC theta c d*
⟨*proof*⟩

Retracts:

definition *Sretr* **where**
Sretr theta ≡
 $\{(c, d). \text{matchC-C } \theta \text{ } c \text{ } d\}$

definition *ZOretr* **where**
ZOretr theta ≡
 $\{(c,d). \text{matchC-LC } \theta \text{ } c \text{ } d\}$

lemmas *Retr-defs* =
Sretr-def
ZOretr-def

lemma *mono-Retr*:

mono Sretr

mono ZOretr

<proof>

lemma *Retr-incl*:

Sretr theta \subseteq *ZOretr theta*

<proof>

The associated bisimilarity relations:

definition *Sbis* **where** *Sbis* \equiv *gfp Sretr*

definition *ZObis* **where** *ZObis* \equiv *gfp ZOretr*

abbreviation *Sbis-abbrev* (**infix** \approx_s 55) **where** $c \approx_s d \equiv (c, d) : Sbis$

abbreviation *ZObis-abbrev* (**infix** \approx_{01} 55) **where** $c \approx_{01} d \equiv (c, d) : ZObis$

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

lemma *bis-incl*:

Sbis \leq *ZObis*

<proof>

lemma *bis-imp[simp]*:

$\bigwedge c1\ c2. c1 \approx_s c2 \implies c1 \approx_{01} c2$

<proof>

lemma *Sbis-prefix*:

Sbis \leq *Sretr Sbis*

<proof>

lemma *Sbis-fix*:

Sretr Sbis = *Sbis*

<proof>

lemma *Sbis-mC-C*:

assumes $c \approx_s d$ **and** $s \approx t$

shows $\exists P\ F. mC-C\ Sbis\ c\ d\ s\ t\ P\ F$

<proof>

lemma *Sbis-coind*:

assumes $theta \leq Sretr$ (*theta Un Sbis*)

shows $theta \leq Sbis$

<proof>

lemma *Sbis-raw-coind*:

assumes $\theta \leq Sretr \theta$
shows $\theta \leq Sbis$
 $\langle proof \rangle$

lemma *mC-C-sym*:
assumes *mC-C* $\theta \ c \ d \ s \ t \ P \ F$
shows *mC-C* $(\theta^{-1}) \ d \ c \ t \ s \ (F \ ' \ P) \ (inv\text{-}into \ P \ F)$
 $\langle proof \rangle$

lemma *matchC-C-sym*:
assumes *matchC-C* $\theta \ c \ d$
shows *matchC-C* $(\theta^{-1}) \ d \ c$
 $\langle proof \rangle$

lemma *Sretr-sym*:
assumes *sym* θ
shows *sym* $(Sretr \ \theta)$
 $\langle proof \rangle$

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

lemma *Sbis-sym*: $c \approx s \ d \implies d \approx s \ c$
 $\langle proof \rangle$

lemma *mC-C-trans*:
assumes $*$: *mC-C* $\theta_1 \ c \ d \ s \ t \ P \ F$ **and** $**$: *mC-C* $\theta_2 \ d \ e \ t \ u \ (F \ ' \ P) \ G$
shows *mC-C* $(\theta_1 \ O \ \theta_2) \ c \ e \ s \ u \ P \ (G \ o \ F)$
 $\langle proof \rangle$

lemma *mC-C-finer*:
assumes $*$: *mC-C* $\theta \ c \ d \ s \ t \ P \ F$
and θ : *trans* θ
and Q : *finer* $P \ Q$ *finite* $Q \ \{\}$ $\notin Q \ part \ \{..<brn \ c\} \ Q$
and c : *partCompat* $Q \ indis \ (eff \ c \ s) \ partCompat \ Q \ \theta \ (cont \ c \ s)$
shows *mC-C* $\theta \ c \ d \ s \ t \ Q \ (lift \ P \ F)$
 $\langle proof \rangle$

lemma *mC-C-partCompat-eff*:
assumes $*$: *mC-C* $\theta \ c \ d \ s \ t \ P \ F$
shows *partCompat* $P \ indis \ (eff \ c \ s)$
 $\langle proof \rangle$

lemma *mC-C-partCompat-cont*:

assumes *: $mC-C$ θ c d s t P F
and θ : $\text{sym } \theta$ $\text{trans } \theta$
shows $\text{partCompat } P$ θ ($\text{cont } c$ s)
 $\langle \text{proof} \rangle$

lemma $\text{matchC-C-sym-trans}$:
assumes *: $\text{matchC-C } \theta$ $c1$ c **and** **: $\text{matchC-C } \theta$ c $c2$
and θ : $\text{sym } \theta$ $\text{trans } \theta$
shows $\text{matchC-C } \theta$ $c1$ $c2$
 $\langle \text{proof} \rangle$

lemma Sretr-sym-trans :
assumes $\text{sym } \theta \wedge \text{trans } \theta$
shows $\text{trans } (\text{Sretr } \theta)$
 $\langle \text{proof} \rangle$

lemma trans-Sbis : $\text{trans } \text{Sbis}$
 $\langle \text{proof} \rangle$

lemma Sbis-trans : $\llbracket c \approx_s d; d \approx_s e \rrbracket \implies c \approx_s e$
 $\langle \text{proof} \rangle$

lemma ZObis-prefix :
 $\text{ZObis} \leq \text{ZOretr } \text{ZObis}$
 $\langle \text{proof} \rangle$

lemma ZObis-fix :
 $\text{ZOretr } \text{ZObis} = \text{ZObis}$
 $\langle \text{proof} \rangle$

lemma ZObis-mC-ZOC :
assumes $c \approx_{01} d$ **and** $s \approx t$
shows $\exists I0$ P F . $mC-ZOC$ $\text{ZObis } c$ d s t $I0$ P F
 $\langle \text{proof} \rangle$

lemma ZObis-coind :
assumes $\theta \leq \text{ZOretr } (\theta \text{ Un } \text{ZObis})$
shows $\theta \leq \text{ZObis}$
 $\langle \text{proof} \rangle$

lemma ZObis-raw-coind :
assumes $\theta \leq \text{ZOretr } \theta$
shows $\theta \leq \text{ZObis}$
 $\langle \text{proof} \rangle$

lemma *mC-ZOC-sym*:
assumes *theta: sym theta* **and** **: mC-ZOC theta c d s t I0 P F*
shows *mC-ZOC theta d c t s (F I0) (F ' P) (inv-into P F)*
 \langle *proof* \rangle

lemma *matchC-LC-sym*:
assumes **: sym theta* **and** *matchC-LC theta c d*
shows *matchC-LC theta d c*
 \langle *proof* \rangle

lemma *ZOretr-sym*:
assumes *sym theta*
shows *sym (ZOretr theta)*
 \langle *proof* \rangle

lemma *sym-ZObis: sym ZObis*
 \langle *proof* \rangle

lemma *ZObis-sym: c \approx_{01} d \implies d \approx_{01} c*
 \langle *proof* \rangle

2.7 List versions of the bisimilarities

definition *SbisL* where
 $SbisL\ cl\ dl \equiv$
 $length\ cl = length\ dl \wedge (\forall\ n < length\ cl.\ cl\ !\ n \approx_s\ dl\ !\ n)$

lemma *SbisL-intro[intro]*:
assumes $length\ cl = length\ dl$ **and**
 $\bigwedge\ n.\ [n < length\ cl; n < length\ dl] \implies cl\ !\ n \approx_s\ dl\ !\ n$
shows $SbisL\ cl\ dl$
 \langle *proof* \rangle

lemma *SbisL-length[simp]*:
assumes $SbisL\ cl\ dl$
shows $length\ cl = length\ dl$
 \langle *proof* \rangle

lemma *SbisL-Sbis[simp]*:
assumes $SbisL\ cl\ dl$ **and** $n < length\ cl \vee n < length\ dl$
shows $cl\ !\ n \approx_s\ dl\ !\ n$
 \langle *proof* \rangle

lemma *SbisL-update[simp]*:
assumes $cl\ dl: SbisL\ cl\ dl$ **and** $c'\ d': c' \approx_s\ d'$
shows $SbisL\ (cl\ [n := c'])\ (dl\ [n := d'])$ (**is** $SbisL\ ?cl'\ ?dl'$)
 \langle *proof* \rangle

lemma *SbisL-update-L[simp]*:
assumes *SbisL cl dl* **and** $c' \approx_s dl!n$
shows *SbisL (cl[n := c']) dl*
 $\langle proof \rangle$

lemma *SbisL-update-R[simp]*:
assumes *SbisL cl dl* **and** $cl!n \approx_s d'$
shows *SbisL cl (dl[n := d'])*
 $\langle proof \rangle$

definition *ZObisL* **where**
 $ZObisL\ cl\ dl \equiv$
 $length\ cl = length\ dl \wedge (\forall\ n < length\ cl.\ cl!\ n \approx_{01}\ dl!\ n)$

lemma *ZObisL-intro[intro]*:
assumes $length\ cl = length\ dl$ **and**
 $\bigwedge n.\ [n < length\ cl; n < length\ dl] \implies cl!\ n \approx_{01}\ dl!\ n$
shows *ZObisL cl dl*
 $\langle proof \rangle$

lemma *ZObisL-length[simp]*:
assumes *ZObisL cl dl*
shows $length\ cl = length\ dl$
 $\langle proof \rangle$

lemma *ZObisL-ZObis[simp]*:
assumes *ZObisL cl dl* **and** $n < length\ cl \vee n < length\ dl$
shows $cl!\ n \approx_{01}\ dl!\ n$
 $\langle proof \rangle$

lemma *ZObisL-update[simp]*:
assumes *cldl: ZObisL cl dl* **and** $c'd': c' \approx_{01}\ d'$
shows *ZObisL (cl [n := c']) (dl [n := d'])* (**is** *ZObisL ?cl' ?dl'*)
 $\langle proof \rangle$

lemma *ZObisL-update-L[simp]*:
assumes *ZObisL cl dl* **and** $c' \approx_{01}\ dl!n$
shows *ZObisL (cl[n := c']) dl*
 $\langle proof \rangle$

lemma *ZObisL-update-R[simp]*:
assumes *ZObisL cl dl* **and** $cl!n \approx_{01}\ d'$
shows *ZObisL cl (dl[n := d'])*
 $\langle proof \rangle$

2.8 Discreetness for configurations

coinductive *discrCf* **where**

intro:

$$\begin{aligned} & (\bigwedge i. i < \text{brn} (\text{fst } cf) \longrightarrow \\ & \quad \text{snd } cf \approx \text{snd} (\text{cont-eff } cf \ i) \wedge \text{discrCf} (\text{cont-eff } cf \ i)) \\ & \implies \text{discrCf } cf \end{aligned}$$

Coinduction for discrness:

lemma *discrCf-coind*[*consumes 1, case-names Hyp, induct pred: discr*]:

assumes *: *phi cf* **and**

** : $\bigwedge cf \ i.$

$$\begin{aligned} & \llbracket i < \text{brn} (\text{fst } cf); \text{phi } cf \rrbracket \implies \\ & \quad \text{snd } cf \approx \text{snd} (\text{cont-eff } cf \ i) \wedge (\text{phi} (\text{cont-eff } cf \ i) \vee \text{discrCf} (\text{cont-eff } cf \ i)) \end{aligned}$$

shows *discrCf cf*

<proof>

lemma *discrCf-raw-coind*[*consumes 1, case-names Hyp*]:

assumes *: *phi cf* **and**

** : $\bigwedge cf \ i.$

$$\llbracket i < \text{brn} (\text{fst } cf); \text{phi } cf \rrbracket \implies \text{snd } cf \approx \text{snd} (\text{cont-eff } cf \ i) \wedge \text{phi} (\text{cont-eff } cf \ i)$$

shows *discrCf cf*

<proof>

Discreetness versus transition:

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

assumes *: *discrCf cf* **and** **: $i < \text{brn} (\text{fst } cf)$

shows *discrCf (cont-eff cf i)*

<proof>

lemma *discrCf-eff-indis*[*simp*]:

assumes *: *discrCf cf* **and** **: $i < \text{brn} (\text{fst } cf)$

shows $\text{snd } cf \approx \text{snd} (\text{cont-eff } cf \ i)$

<proof>

lemma *discr-discrCf*:

assumes *discr c*

shows *discrCf (c, s)*

<proof>

lemma *ZObis-pres-discrCfL*:

assumes $\text{fst } cf \approx_{01} \text{fst } df$ **and** $\text{snd } cf \approx \text{snd } df$ **and** *discrCf df*

shows *discrCf cf*

<proof>

corollary *ZObis-pres-discrCfR*:

assumes *discrCf cf* **and** $\text{fst } cf \approx_{01} \text{fst } df$ **and** $\text{snd } cf \approx \text{snd } df$

shows *discrCf df*

<proof>

end

end

3 Trace-Based Noninterference

theory *Trace-Based*
 imports *Resumption-Based*
begin

3.1 Preliminaries

lemma *dist-sum*:

fixes $f :: 'a \Rightarrow \text{real}$ and $g :: 'a \Rightarrow \text{real}$
 assumes $\bigwedge i. i \in I \Longrightarrow \text{dist } (f\ i) (g\ i) \leq e\ i$
 shows $\text{dist } (\sum_{i \in I}. f\ i) (\sum_{i \in I}. g\ i) \leq (\sum_{i \in I}. e\ i)$
(*proof*)

lemma *dist-mult[simp]*: $\text{dist } (x * y) (x * z) = |x| * \text{dist } y\ z$
(*proof*)

lemma *dist-divide[simp]*: $\text{dist } (y / r) (z / r) = \text{dist } y\ z / |r|$
(*proof*)

lemma *dist-weighted-sum*:

fixes $f :: 'a \Rightarrow \text{real}$ and $g :: 'b \Rightarrow \text{real}$
 assumes $\text{eps}: \bigwedge i\ j. i \in I \Longrightarrow j \in J \Longrightarrow w\ i \neq 0 \Longrightarrow v\ j \neq 0 \Longrightarrow \text{dist } (f\ i) (g\ j) \leq d\ i + e\ j$
 and $\text{pos}: \bigwedge i. i \in I \Longrightarrow 0 \leq w\ i \bigwedge j. j \in J \Longrightarrow 0 \leq v\ j$
 and $\text{sum}: (\sum_{i \in I}. w\ i) = 1 \ (\sum_{j \in J}. v\ j) = 1$
 shows $\text{dist } (\sum_{i \in I}. w\ i * f\ i) (\sum_{j \in J}. v\ j * g\ j) \leq (\sum_{i \in I}. w\ i * d\ i) + (\sum_{j \in J}. v\ j * e\ j)$
(*proof*)

lemma *field-abs-le-zero-epsilon*:

fixes $x :: 'a::\{\text{linordered-field}\}$
 assumes $\text{epsilon}: \bigwedge e. 0 < e \Longrightarrow |x| \leq e$
 shows $|x| = 0$
(*proof*)

lemma *nat-nat-induct[case-names less]*:

fixes $P :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$
 assumes $\text{less}: \bigwedge n\ m. (\bigwedge j\ k. j + k < n + m \Longrightarrow P\ j\ k) \Longrightarrow P\ n\ m$
 shows $P\ n\ m$
(*proof*)

lemma *part-insert*:

assumes *part A P* **assumes** $X \cap A = \{\}$
shows *part (A \cup X) (insert X P)*
<proof>

lemma *part-insert-subset*:

assumes $X: \text{part } (A - X) P X \subseteq A$
shows *part A (insert X P)*
<proof>

lemma *part-is-subset*:

part S P $\implies p \in P \implies p \subseteq S$
<proof>

lemma *dist-nonneg-bounded*:

fixes $l u x y :: \text{real}$
assumes $l \leq x x \leq u l \leq y y \leq u$
shows $\text{dist } x y \leq u - l$
<proof>

lemma *integrable-count-space-finite-support*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
shows $\text{finite } \{x \in X. f x \neq 0\} \implies \text{integrable } (\text{count-space } X) f$
<proof>

lemma *lebesgue-integral-point-measure*:

fixes $g :: - \Rightarrow \text{real}$
assumes $f: \text{finite } \{a \in A. 0 < f a \wedge g a \neq 0\}$
shows $\text{integral}^L (\text{point-measure } A f) g = (\sum a | a \in A \wedge 0 < f a \wedge g a \neq 0. f a * g a)$

<proof>

lemma (**in** *finite-measure*) *finite-measure-dist*:

assumes $AE: AE x \text{ in } M. x \notin C \longrightarrow (x \in A \longleftrightarrow x \in B)$
assumes *sets: A \in sets M B \in sets M C \in sets M*
shows $\text{dist } (\text{measure } M A) (\text{measure } M B) \leq \text{measure } M C$
<proof>

lemma (**in** *prob-space*) *prob-dist*:

assumes $AE: AE x \text{ in } M. \neg C x \longrightarrow (A x \longleftrightarrow B x)$
assumes *sets: Measurable.pred M A Measurable.pred M B Measurable.pred M C*
shows $\text{dist } \mathcal{P}(x \text{ in } M. A x) \mathcal{P}(x \text{ in } M. B x) \leq \mathcal{P}(x \text{ in } M. C x)$
<proof>

lemma *Least-eq-0-iff*: $(\exists i :: \text{nat}. P i) \implies (\text{LEAST } i. P i) = 0 \longleftrightarrow P 0$

<proof>

lemma *case-nat-comp-Suc[simp]*: $\text{case-nat } x f \circ \text{Suc} = f$

⟨proof⟩

lemma *sum-eq-0-iff*:

fixes $f :: - \Rightarrow 'a :: \{\text{comm-monoid-add, ordered-ab-group-add}\}$

shows $\text{finite } A \Longrightarrow (\bigwedge i. i \in A \Longrightarrow 0 \leq f i) \Longrightarrow (\sum_{i \in A}. f i) = 0 \longleftrightarrow (\forall i \in A. f i = 0)$

⟨proof⟩

lemma *sum-less-0-iff*:

fixes $f :: - \Rightarrow 'a :: \{\text{comm-monoid-add, ordered-ab-group-add}\}$

shows $\text{finite } A \Longrightarrow (\bigwedge i. i \in A \Longrightarrow 0 \leq f i) \Longrightarrow 0 < (\sum_{i \in A}. f i) \longleftrightarrow (\exists i \in A. 0 < f i)$

⟨proof⟩

context *PL-Indis*

begin

declare *emp-gen*[*simp del*]

interpretation *pmf-as-function* ⟨proof⟩

lift-definition *wt-pmf* :: $('test, 'atom, 'choice) \text{ cmd} \times 'state \Rightarrow \text{nat pmf}$ **is**

$\lambda(c, s) i. \text{if proper } c \text{ then if } i < \text{brn } c \text{ then wt } c \text{ s } i \text{ else } 0 \text{ else if } i = 0 \text{ then } 1 \text{ else } 0$

⟨proof⟩

definition *trans* :: $('test, 'atom, 'choice) \text{ cmd} \times 'state \Rightarrow (('test, 'atom, 'choice) \text{ cmd} \times 'state) \text{ pmf}$ **where**

$\text{trans } cf = \text{map-pmf } (\lambda i. \text{if proper } (fst \text{ cf}) \text{ then cont-eff } cf \text{ } i \text{ else } cf) (\text{wt-pmf } cf)$

sublocale *T?*: *MC-syntax trans* ⟨proof⟩

abbreviation $G \text{ cf} \equiv \text{set-pmf } (\text{trans } cf)$

lemma *set-pmf-map*: $\text{set-pmf } (\text{map-pmf } f \text{ } M) = f \text{ ' set-pmf } M$

⟨proof⟩

lemma *set-pmf-wt*:

$\text{set-pmf } (\text{wt-pmf } cf) = (\text{if proper } (fst \text{ cf}) \text{ then } \{i. i < \text{brn } (fst \text{ cf}) \wedge 0 < \text{wt } (fst \text{ cf}) (snd \text{ cf}) \text{ } i\} \text{ else } \{0\})$

⟨proof⟩

lemma *G-eq*:

$G \text{ cf} = (\text{if proper } (fst \text{ cf}) \text{ then } \{\text{cont-eff } cf \text{ } i \mid i. i < \text{brn } (fst \text{ cf}) \wedge 0 < \text{wt } (fst \text{ cf}) (snd \text{ cf}) \text{ } i\} \text{ else } \{cf\})$

⟨proof⟩

lemma *discrCf-G*: $\text{discrCf } cf \Longrightarrow cf' \in G \text{ cf} \Longrightarrow \text{discrCf } cf'$

⟨proof⟩

lemma *measurable-discrCf*[*measurable*]: *Measurable.pred (count-space UNIV) discrCf*
 ⟨*proof*⟩

lemma *measurable-indis*[*measurable*]: *Measurable.pred (count-space UNIV) (λx. snd x ≈ c)*
 ⟨*proof*⟩

lemma *integral-trans*:
proper (fst cf) ⇒
*(∫ x. f x ∂trans cf) = (∑ i < brn (fst cf). wt (fst cf) (snd cf) i * f (cont-eff cf i))*
 ⟨*proof*⟩

3.2 Quasi strong termination traces

abbreviation *qsend* ≡ *sfirst (holds discrCf)*

lemma *qsend-eq-0-iff*: *qsend cfT = 0 ⟷ discrCf (shd cfT)*
 ⟨*proof*⟩

lemma *qsend-eq-0[simp]*: *discrCf cf ⇒ qsend (cf ## ω) = 0*
 ⟨*proof*⟩

lemma *alw-discrCf*: *enabled cf ω ⇒ discrCf cf ⇒ alw (holds discrCf) ω*
 ⟨*proof*⟩

lemma *alw-discrCf-indis'*:
enabled cf ω ⇒ discrCf cf ⇒ snd cf ≈ t ⇒ alw (holds (λcf'. snd cf' ≈ t))
ω
 ⟨*proof*⟩

lemma *alw-discrCf-indis*:
enabled cf ω ⇒ discrCf cf ⇒ alw (holds (λcf'. snd cf' ≈ snd cf)) (cf ## ω)
 ⟨*proof*⟩

lemma *enabled-sdrop*: *enabled cf ω ⇒ enabled ((cf ## ω) !! n) (sdrop n ω)*
 ⟨*proof*⟩

lemma *sfirst-eq-eSuc*: *sfirst P ω = eSuc n ⟷ (¬ P ω) ∧ sfirst P (stl ω) = n*
 ⟨*proof*⟩

lemma *qsend-snth*: *qsend ω = enat n ⇒ discrCf (ω !! n)*
 ⟨*proof*⟩

lemma *indis-iff*: *a ≈ d ⇒ b ≈ d ⇒ a ≈ c ⟷ b ≈ c*
 ⟨*proof*⟩

lemma *enabled-qsend-indis*:

assumes $\text{enabled } cf \ \omega \ \text{qsend } (cf \ \#\# \ \omega) \leq n \ \text{qsend } (cf \ \#\# \ \omega) \leq m$
shows $\text{snd } ((cf \ \#\# \ \omega) \ \#\# \ n) \approx t \iff \text{snd } ((cf \ \#\# \ \omega) \ \#\# \ m) \approx t$
 ⟨proof⟩

lemma *enabled-imp-UNTIL-alw-discrCf*:
 $\text{enabled } (\text{shd } \omega) \ (\text{stl } \omega) \implies (\text{not } (\text{holds } \text{discrCf}) \ \text{until } (\text{alw } (\text{holds } \text{discrCf}))) \ \omega$
 ⟨proof⟩

lemma *less-qsend-iff-not-discrCf*:
 $\text{enabled } cf \ bT \implies n < \text{qsend } (cf \ \#\# \ bT) \iff \neg \text{discrCf } ((cf \ \#\# \ bT) \ \#\# \ n)$
 ⟨proof⟩

3.3 Terminating configurations

definition $\text{qstermCf } cf \iff (\forall cfT. \text{enabled } cf \ cfT \implies \text{qsend } (cf \ \#\# \ cfT) < \infty)$

lemma *qstermCf-E*:
 $\text{qstermCf } cf \implies cf' \in G \ cf \implies \text{qstermCf } cf'$
 ⟨proof⟩

abbreviation $\text{eff-at } cf \ bT \ n \equiv \text{snd } ((cf \ \#\# \ bT) \ \#\# \ n)$
abbreviation $\text{cont-at } cf \ bT \ n \equiv \text{fst } ((cf \ \#\# \ bT) \ \#\# \ n)$

definition $\text{amSec } c \iff (\forall s1 \ s2 \ n \ t. s1 \approx s2 \implies$
 $\mathcal{P}(bT \ \text{in } T.T \ (c, \ s1). \text{eff-at } (c, \ s1) \ bT \ n \approx t) =$
 $\mathcal{P}(bT \ \text{in } T.T \ (c, \ s2). \text{eff-at } (c, \ s2) \ bT \ n \approx t))$

definition $\text{eSec } c \iff (\forall s1 \ s2 \ t. s1 \approx s2 \implies$
 $\mathcal{P}(bT \ \text{in } T.T \ (c, \ s1). \exists n. \text{qsend } ((c, \ s1) \ \#\# \ bT) = n \wedge \text{eff-at } (c, \ s1) \ bT \ n \approx$
 $t) =$
 $\mathcal{P}(bT \ \text{in } T.T \ (c, \ s2). \exists n. \text{qsend } ((c, \ s2) \ \#\# \ bT) = n \wedge \text{eff-at } (c, \ s2) \ bT \ n \approx$
 $t))$

definition $\text{aeT } c \iff (\forall s. \text{AE } bT \ \text{in } T.T \ (c, s). \text{qsend } ((c, s) \ \#\# \ bT) < \infty)$

lemma *dist-Ps-upper-bound*:

fixes $cf1 \ cf2 :: ('test, 'atom, 'choice) \text{cmd} \times 'state \ \text{and } s :: 'state \ \text{and } S$
defines $S \ cf \ bT \equiv \exists n. \text{qsend } (cf \ \#\# \ bT) = n \wedge \text{eff-at } cf \ bT \ n \approx s$
defines $Ps \ cf \equiv \mathcal{P}(bT \ \text{in } T.T \ cf. S \ cf \ bT)$
defines $N \ cf \ n \ bT \equiv \neg \text{discrCf } ((cf \ \#\# \ bT) \ \#\# \ n)$
defines $Pn \ cf \ n \equiv \mathcal{P}(bT \ \text{in } T.T \ cf. N \ cf \ n \ bT)$
assumes *bisim: proper (fst cf1) proper (fst cf2) fst cf1 ≈01 fst cf2 snd cf1 ≈*
snd cf2
shows $\text{dist } (Ps \ cf1) \ (Ps \ cf2) \leq Pn \ cf1 \ n + Pn \ cf2 \ m$
 ⟨proof⟩

lemma *AE-T-max-qsend-time*:

fixes $cf \ \text{and } e :: \text{real} \ \text{assumes } \text{AE: } \text{AE } bT \ \text{in } T.T \ cf. \text{qsend } (cf \ \#\# \ bT) < \infty$
 $0 < e$

shows $\exists N. \mathcal{P}(bT \text{ in } T.T \text{ cf. } \neg \text{discrCf } ((\text{cf} \ \#\# \ bT) \ !! \ N)) < e$
 ⟨proof⟩

lemma *Ps-eq*:

fixes *cf1 cf2 s and S*

defines $S \text{ cf } bT \equiv \exists n. \text{qsend } (\text{cf} \ \#\# \ bT) = n \wedge \text{eff-at } \text{cf } bT \ n \approx s$

defines $Ps \ \text{cf} \equiv \mathcal{P}(bT \text{ in } T.T \ \text{cf. } S \ \text{cf } bT)$

assumes *qsterm1*: $AE \ bT \text{ in } T.T \ \text{cf1. } \text{qsend } (\text{cf1} \ \#\# \ bT) < \infty$

assumes *qsterm2*: $AE \ bT \text{ in } T.T \ \text{cf2. } \text{qsend } (\text{cf2} \ \#\# \ bT) < \infty$

and *bisim*: $\text{proper } (\text{fst } \text{cf1}) \ \text{proper } (\text{fst } \text{cf2}) \ \text{fst } \text{cf1} \approx_{01} \text{fst } \text{cf2} \ \text{snd } \text{cf1} \approx \text{snd } \text{cf2}$

shows $Ps \ \text{cf1} = Ps \ \text{cf2}$

⟨proof⟩

lemma *siso-trace*:

assumes *siso* $c \ s \approx t \ \text{enabled } (c, t) \ \text{cfT}$

shows *siso* $(\text{cont-at } (c, s) \ \text{cfT } n)$

and $\text{cont-at } (c, s) \ \text{cfT } n = \text{cont-at } (c, t) \ \text{cfT } n$

and $\text{eff-at } (c, s) \ \text{cfT } n \approx \text{eff-at } (c, t) \ \text{cfT } n$

⟨proof⟩

lemma *Sbis-trace*:

assumes *proper* $(\text{fst } \text{cf1}) \ \text{proper } (\text{fst } \text{cf2}) \ \text{fst } \text{cf1} \approx_s \text{fst } \text{cf2} \ \text{snd } \text{cf1} \approx \text{snd } \text{cf2}$

shows $\mathcal{P}(\text{cfT} \text{ in } T.T \ \text{cf1. } \text{eff-at } \text{cf1} \ \text{cfT } n \approx s') = \mathcal{P}(\text{cfT} \text{ in } T.T \ \text{cf2. } \text{eff-at } \text{cf2} \ \text{cfT } n \approx s')$

(**is** $?P \ \text{cf1} \ n = ?P \ \text{cf2} \ n$)

⟨proof⟩

3.4 Final Theorems

theorem *ZObis-eSec*: $\llbracket \text{proper } c; c \approx_{01} c; \text{aeT } c \rrbracket \implies eSec \ c$

⟨proof⟩

theorem *Sbis-amSec*: $\llbracket \text{proper } c; c \approx_s c \rrbracket \implies amSec \ c$

⟨proof⟩

theorem *amSec-eSec*:

assumes [*simp*]: $\text{proper } c$ **and** $\text{aeT } c \ \text{amSec } c$ **shows** $eSec \ c$

⟨proof⟩

end

end

4 Compositionality of Resumption-Based Noninterference

theory *Compositionality*

imports *Resumption-Based*

begin

context *PL-Indis*

begin

4.1 Compatibility and discreteness of atoms, tests and choices

definition *compatAtm* **where**

compatAtm atm \equiv

ALL s t. s \approx t \longrightarrow aval atm s \approx aval atm t

definition *presAtm* **where**

presAtm atm \equiv

ALL s. s \approx aval atm s

definition *compatTst* **where**

compatTst tst \equiv

ALL s t. s \approx t \longrightarrow tval tst s = tval tst t

lemma *discrAt-compatAt[simp]*:

assumes *presAtm atm*

shows *compatAtm atm*

\langle proof \rangle

definition *compatCh* **where**

compatCh ch $\equiv \forall s t. s \approx t \longrightarrow cval ch s = cval ch t$

lemma *compatCh-cval[simp]*:

assumes *compatCh ch* **and** *s \approx t*

shows *cval ch s = cval ch t*

\langle proof \rangle

4.2 Compositionality of self-isomorphism

Self-Isomorphism versus language constructs:

lemma *iso-Done[simp]*:

iso Done

\langle proof \rangle

lemma *iso-Atm[simp]*:

iso (Atm atm) = compatAtm atm

\langle proof \rangle

lemma *iso-Seq[simp]*:

assumes ***: *iso c1* **and** ****: *iso c2*

shows *iso (c1 ;; c2)*

\langle proof \rangle

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

lemma *siso-Ch*[simp]:
assumes *compatCh ch*
and *: *siso c1* **and** **: *siso c2*
shows *siso (Ch ch c1 c2)*
⟨*proof*⟩

lemma *siso-Par*[simp]:
assumes *properL cl* **and** *sisoL cl*
shows *siso (Par cl)*
⟨*proof*⟩

lemma *siso-ParT*[simp]:
assumes *properL cl* **and** *sisoL cl*
shows *siso (ParT cl)*
⟨*proof*⟩

Self-isomorphism implies strong bisimilarity:

lemma *bij-betw-emp*[simp]:
bij-betw f {} {}
⟨*proof*⟩

lemma *part-full*[simp]:
part I {I}
⟨*proof*⟩

definition *singlPart* **where**
singlPart I $\equiv \{\{i\} \mid i . i \in I\}$

lemma *part-singlPart*[simp]:
part I (singlPart I)
⟨*proof*⟩

lemma *singlPart-inj-on*[simp]:
inj-on (image f) (singlPart I) = inj-on f I
⟨*proof*⟩

lemma *singlPart-surj*[simp]:
(image f) ' (singlPart I) = (singlPart J) \longleftrightarrow f ' I = J
⟨*proof*⟩

lemma *singlPart-bij-betw*[simp]:
bij-betw (image f) (singlPart I) (singlPart J) = bij-betw f I J
⟨*proof*⟩

lemma *singlPart-finite1*:
assumes *finite (singlPart I)*
shows *finite (I::'a set)*
 \langle *proof* \rangle

lemma *singlPart-finite[simp]*:
finite (singlPart I) = finite I
 \langle *proof* \rangle

lemma *emp-notIn-singlPart[simp]*:
 $\{\} \notin \text{singlPart } I$
 \langle *proof* \rangle

lemma *Sbis-coinduct[consumes 1, case-names step, coinduct set]*:
 $R \ c \ d \implies$
 $(\bigwedge c \ d \ s \ t. R \ c \ d \implies s \approx t \implies$
 $\exists P \ F. \text{mC-C-part } c \ d \ P \ F \wedge \text{inj-on } F \ P \wedge \text{mC-C-wt } c \ d \ s \ t \ P \ F \wedge$
 $(\forall I \in P. \forall i \in I. \forall j \in F \ I.$
 $\text{eff } c \ s \ i \approx \text{eff } d \ t \ j \wedge (R \ (\text{cont } c \ s \ i) \ (\text{cont } d \ t \ j) \vee (\text{cont } c \ s \ i, \text{cont } d$
 $t \ j) \in \text{Sbis}))$
 $\implies (c, d) \in \text{Sbis}$
 \langle *proof* \rangle

lemma *siso-Sbis[simp]*: *siso c \implies c \approx_s c*
 \langle *proof* \rangle

4.3 Discreetness versus language constructs:

lemma *discr-Done[simp]*: *discr Done*
 \langle *proof* \rangle

lemma *discr-Atm-presAtm[simp]*: *discr (Atm atm) = presAtm atm*
 \langle *proof* \rangle

lemma *discr-Seq[simp]*:
discr c1 \implies discr c2 \implies discr (c1 ;; c2)
 \langle *proof* \rangle

lemma *discr-While[simp]*: **assumes** *discr c* **shows** *discr (While tst c)*
 \langle *proof* \rangle

lemma *discr-Ch[simp]*: *discr c1 \implies discr c2 \implies discr (Ch ch c1 c2)*
 \langle *proof* \rangle

lemma *discr-Par[simp]*: *properL cl \implies discrL cl \implies discr (Par cl)*
 \langle *proof* \rangle

lemma *discr-ParT[simp]*: *properL cl \implies discrL cl \implies discr (ParT cl)*

$\langle proof \rangle$

lemma *discr-finished[simp]*: $proper\ c \implies finished\ c \implies discr\ c$
 $\langle proof \rangle$

4.4 Strong bisimilarity versus language constructs

lemma *Sbis-pres-discr-L*:
 $c \approx_s d \implies discr\ d \implies discr\ c$
 $\langle proof \rangle$

lemma *Sbis-pres-discr-R*:
assumes $discr\ c$ **and** $c \approx_s d$
shows $discr\ d$
 $\langle proof \rangle$

lemma *Sbis-finished-discr-L*:
assumes $c \approx_s d$ **and** $proper\ d$ **and** $finished\ d$
shows $discr\ c$
 $\langle proof \rangle$

lemma *Sbis-finished-discr-R*:
assumes $proper\ c$ **and** $finished\ c$ **and** $c \approx_s d$
shows $discr\ d$
 $\langle proof \rangle$

definition *thetaSD* **where**
 $thetaSD \equiv \{(c, d) \mid c\ d .\ proper\ c \wedge proper\ d \wedge discr\ c \wedge discr\ d\}$

lemma *thetaSD-Sretr*:
 $thetaSD \subseteq Sretr\ thetaSD$
 $\langle proof \rangle$

lemma *thetaSD-Sbis*:
 $thetaSD \subseteq Sbis$
 $\langle proof \rangle$

theorem *discr-Sbis[simp]*:
assumes $proper\ c$ **and** $proper\ d$ **and** $discr\ c$ **and** $discr\ d$
shows $c \approx_s d$
 $\langle proof \rangle$

definition *thetaSDone* **where**
 $thetaSDone \equiv \{(Done, Done)\}$

lemma *thetaSDone-Sretr*:
 $thetaSDone \subseteq Sretr\ thetaSDone$
(proof)

lemma *thetaSDone-Sbis*:
 $thetaSDone \subseteq Sbis$
(proof)

theorem *Done-Sbis[simp]*:
 $Done \approx_s Done$
(proof)

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

lemma *thetaSAtm-Sretr*:
assumes *compatAtm atm*
shows $thetaSAtm\ atm \subseteq Sretr\ (thetaSAtm\ atm)$
(proof)

lemma *thetaSAtm-Sbis*:
assumes *compatAtm atm*
shows $thetaSAtm\ atm \subseteq Sbis$
(proof)

theorem *Atm-Sbis[simp]*:
assumes *compatAtm atm*
shows $Atm\ atm \approx_s Atm\ atm$
(proof)

definition *thetaSSeqI* where
 $thetaSSeqI \equiv$
 $\{(e ;; c, e ;; d) \mid e\ c\ d .\ siso\ e \wedge c \approx_s d\}$

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

lemma *thetaSSeqI-Sbis*:
 $thetaSSeqI \subseteq Sbis$
(proof)

theorem *Seq-siso-Sbis[simp]*:
assumes *siso e* and $c2 \approx_s d2$
shows $e ;; c2 \approx_s e ;; d2$
(proof)

definition *thetaSSeqD* where

$thetaSSeqD \equiv$
 $\{(c1 ;; c2, d1 ;; d2) \mid$
 $c1 \ c2 \ d1 \ d2.$
 $proper \ c1 \ \wedge \ proper \ d1 \ \wedge \ proper \ c2 \ \wedge \ proper \ d2 \ \wedge$
 $discr \ c2 \ \wedge \ discr \ d2 \ \wedge$
 $c1 \ \approx_s \ d1\}$

lemma *thetaSSeqD-Sretr*:

$thetaSSeqD \subseteq Sretr \ (thetaSSeqD \ Un \ Sbis)$
<proof>

lemma *thetaSSeqD-Sbis*:

$thetaSSeqD \subseteq Sbis$
<proof>

theorem *Seq-Sbis[simp]*:

assumes *proper c1 and proper d1 and proper c2 and proper d2*
and *c1 \approx_s d1 and discr c2 and discr d2*
shows *c1 ;; c2 \approx_s d1 ;; d2*
<proof>

definition *thetaSCh* where

$thetaSCh \ ch \ c1 \ c2 \ d1 \ d2 \equiv \{(Ch \ ch \ c1 \ c2, Ch \ ch \ d1 \ d2)\}$

lemma *thetaSCh-Sretr*:

assumes *compatCh ch and c1 \approx_s d1 and c2 \approx_s d2*
shows $thetaSCh \ ch \ c1 \ c2 \ d1 \ d2 \subseteq$
 $Sretr \ (thetaSCh \ ch \ c1 \ c2 \ d1 \ d2 \ \cup \ Sbis)$
(is $?th \subseteq Sretr \ (?th \ \cup \ Sbis)$
<proof>

lemma *thetaSCh-Sbis*:

assumes *compatCh ch and c1 \approx_s d1 and c2 \approx_s d2*
shows $thetaSCh \ ch \ c1 \ c2 \ d1 \ d2 \subseteq Sbis$
<proof>

theorem *Ch-iso-Sbis[simp]*:

assumes *compatCh ch and c1 \approx_s d1 and c2 \approx_s d2*
shows $Ch \ ch \ c1 \ c2 \ \approx_s \ Ch \ ch \ d1 \ d2$
<proof>

definition *shift* **where**

$shift\ cl\ n \equiv image\ (\%i.\ brnL\ cl\ n + i)$

definition *back* **where**

$back\ cl\ n \equiv image\ (\%ii.\ ii - brnL\ cl\ n)$

lemma *emp-shift*[*simp*]:

$shift\ cl\ n\ I = \{\} \longleftrightarrow I = \{\}$
(*proof*)

lemma *emp-shift-rev*[*simp*]:

$\{\} = shift\ cl\ n\ I \longleftrightarrow I = \{\}$
(*proof*)

lemma *emp-back*[*simp*]:

$back\ cl\ n\ II = \{\} \longleftrightarrow II = \{\}$
(*proof*)

lemma *emp-back-rev*[*simp*]:

$\{\} = back\ cl\ n\ II \longleftrightarrow II = \{\}$
(*proof*)

lemma *in-shift*[*simp*]:

$brnL\ cl\ n + i \in shift\ cl\ n\ I \longleftrightarrow i \in I$
(*proof*)

lemma *in-back*[*simp*]:

$ii \in II \implies ii - brnL\ cl\ n \in back\ cl\ n\ II$
(*proof*)

lemma *in-back2*[*simp*]:

assumes $ii > brnL\ cl\ n$ **and** $II \subseteq \{brnL\ cl\ n ..<+ brn\ (cl!n)\}$
shows $ii - brnL\ cl\ n \in back\ cl\ n\ II \longleftrightarrow ii \in II$ (**is** ?L \longleftrightarrow ?R)
(*proof*)

lemma *shift*[*simp*]:

assumes $I \subseteq \{..
shows $shift\ cl\ n\ I \subseteq \{brnL\ cl\ n ..<+ brn\ (cl!n)\}$
(*proof*)$

lemma *shift2*[*simp*]:

assumes $I \subseteq \{..
and $ii \in shift\ cl\ n\ I$
shows $brnL\ cl\ n \leq ii \wedge ii < brnL\ cl\ n + brn\ (cl!n)$
(*proof*)$

lemma *shift3*[*simp*]:

assumes $n: n < length\ cl$ **and** $I: I \subseteq \{..
and $ii: ii \in shift\ cl\ n\ I$$

shows $ii < brnL\ cl\ (length\ cl)$
<proof>

lemma *back[simp]*:
assumes $II \subseteq \{brnL\ cl\ n\ ..<+ \ brn\ (cl!n)\}$
shows $back\ cl\ n\ II \subseteq \{..< \ brn\ (cl!n)\}$
<proof>

lemma *back2[simp]*:
assumes $II \subseteq \{brnL\ cl\ n\ ..<+ \ brn\ (cl!n)\}$
and $i \in back\ cl\ n\ II$
shows $i < brn\ (cl!n)$
<proof>

lemma *shift-inj[simp]*:
 $shift\ cl\ n\ I1 = shift\ cl\ n\ I2 \longleftrightarrow I1 = I2$
<proof>

lemma *shift-mono[simp]*:
 $shift\ cl\ n\ I1 \subseteq shift\ cl\ n\ I2 \longleftrightarrow I1 \subseteq I2$
<proof>

lemma *shift-Int[simp]*:
 $shift\ cl\ n\ I1 \cap shift\ cl\ n\ I2 = \{\} \longleftrightarrow I1 \cap I2 = \{\}$
<proof>

lemma *inj-shift*: $inj\ (shift\ cl\ n)$
<proof>

lemma *inj-on-shift*: $inj-on\ (shift\ cl\ n)\ K$
<proof>

lemma *back-shift[simp]*:
 $back\ cl\ n\ (shift\ cl\ n\ I) = I$
<proof>

lemma *shift-back[simp]*:
assumes $II \subseteq \{brnL\ cl\ n\ ..<+ \ brn\ (cl!n)\}$
shows $shift\ cl\ n\ (back\ cl\ n\ II) = II$
<proof>

lemma *back-inj[simp]*:
assumes $II1: II1 \subseteq \{brnL\ cl\ n\ ..<+ \ brn\ (cl!n)\}$
and $II2: II2 \subseteq \{brnL\ cl\ n\ ..<+ \ brn\ (cl!n)\}$
shows $back\ cl\ n\ II1 = back\ cl\ n\ II2 \longleftrightarrow II1 = II2$ (**is** $?L = ?R \longleftrightarrow II1 = II2$)
<proof>

lemma *back-mono[simp]*:
assumes $II1 \subseteq \{brnL\ cl\ n\ ..<+ \ brn\ (cl!n)\}$

and $II2 \subseteq \{brnL\ cl\ n\ ..<\ brnL\ cl\ n\ +\ brn\ (cl!n)\}$
shows $back\ cl\ n\ II1 \subseteq back\ cl\ n\ II2 \longleftrightarrow II1 \subseteq II2$
(is $?L \subseteq ?R \longleftrightarrow II1 \subseteq II2)$
 $\langle proof \rangle$

lemma *back-Int[simp]*:
assumes $II1 \subseteq \{brnL\ cl\ n\ ..<+\ brn\ (cl!n)\}$
and $II2 \subseteq \{brnL\ cl\ n\ ..<\ brnL\ cl\ n\ +\ brn\ (cl!n)\}$
shows $back\ cl\ n\ II1 \cap back\ cl\ n\ II2 = \{\} \longleftrightarrow II1 \cap II2 = \{\}$
(is $?L \cap ?R = \{\} \longleftrightarrow II1 \cap II2 = \{\})$
 $\langle proof \rangle$

lemma *inj-on-back*:
 $inj\ on\ (back\ cl\ n)\ (Pow\ \{brnL\ cl\ n\ ..<+\ brn\ (cl!n)\})$
 $\langle proof \rangle$

lemma *shift-surj*:
assumes $II \subseteq \{brnL\ cl\ n\ ..<+\ brn\ (cl!n)\}$
shows $\exists I. I \subseteq \{..<\ brn\ (cl!n)\} \wedge shift\ cl\ n\ I = II$
 $\langle proof \rangle$

lemma *back-surj*:
assumes $I \subseteq \{..<\ brn\ (cl!n)\}$
shows $\exists II. II \subseteq \{brnL\ cl\ n\ ..<+\ brn\ (cl!n)\} \wedge back\ cl\ n\ II = I$
 $\langle proof \rangle$

lemma *shift-part[simp]*:
assumes $part\ \{..<\ brn\ (cl!n)\}\ P$
shows $part\ \{brnL\ cl\ n\ ..<+\ brn\ (cl!n)\}\ (shift\ cl\ n\ 'P)$
 $\langle proof \rangle$

lemma *part-brn-disj1*:
assumes $P: \bigwedge n. n < length\ cl \implies part\ \{..<\ brn\ (cl!n)\}\ (P\ n)$
and $n1: n1 < length\ cl$ **and** $n2: n2 < length\ cl$
and $II1: II1 \in shift\ cl\ n1\ ' (P\ n1)$ **and** $II2: II2 \in shift\ cl\ n2\ ' (P\ n2)$ **and** $d: n1 \neq n2$
shows $II1 \cap II2 = \{\}$
 $\langle proof \rangle$

lemma *part-brn-disj2*:
assumes $P: \bigwedge n. n < length\ cl \implies part\ \{..<\ brn\ (cl!n)\}\ (P\ n) \wedge \{\} \notin P\ n$
and $n1: n1 < length\ cl$ **and** $n2: n2 < length\ cl$ **and** $d: n1 \neq n2$
shows $shift\ cl\ n1\ ' (P\ n1) \cap shift\ cl\ n2\ ' (P\ n2) = \{\}$ **(is** $?L \cap ?R = \{\})$
 $\langle proof \rangle$

lemma *part-brn-disj3*:
assumes $P: \bigwedge n. n < length\ cl \implies part\ \{..<\ brn\ (cl!n)\}\ (P\ n)$
and $n1: n1 < length\ cl$ **and** $n2: n2 < length\ cl$
and $I1: I1 \in P\ n1$ **and** $I2: I2 \in P\ n2$ **and** $d: n1 \neq n2$

shows $\text{shift } cl \ n1 \ I1 \cap \text{shift } cl \ n2 \ I2 = \{\}$
 $\langle \text{proof} \rangle$

lemma $\text{sum-wt-Par-sub-shift[simp]}$:
assumes $cl: \text{properL } cl$ **and** $n: n < \text{length } cl$ **and**
 $I: I \subseteq \{.. < \text{brn } (cl \ ! \ n)\}$
shows
 $\text{sum } (wt \ (Par \ cl) \ s) \ (\text{shift } cl \ n \ I) =$
 $1 / (\text{length } cl) * \text{sum } (wt \ (cl \ ! \ n) \ s) \ I$
 $\langle \text{proof} \rangle$

lemma $\text{sum-wt-ParT-sub-WtFT-pickFT-0-shift[simp]}$:
assumes $cl: \text{properL } cl$ **and** $nf: WtFT \ cl = 1$
and $I: I \subseteq \{.. < \text{brn } (cl \ ! \ (\text{pickFT } cl))\}$ $0 \in I$
shows
 $\text{sum } (wt \ (ParT \ cl) \ s) \ (\text{shift } cl \ (\text{pickFT } cl) \ I) = 1$
 $\langle \text{proof} \rangle$

lemma $\text{sum-wt-ParT-sub-WtFT-notPickFT-0-shift[simp]}$:
assumes $cl: \text{properL } cl$ **and** $nf: WtFT \ cl = 1$ **and** $n: n < \text{length } cl$
and $I: I \subseteq \{.. < \text{brn } (cl \ ! \ n)\}$ **and** $nI: n = \text{pickFT } cl \longrightarrow 0 \notin I$
shows $\text{sum } (wt \ (ParT \ cl) \ s) \ (\text{shift } cl \ n \ I) = 0$
 $\langle \text{proof} \rangle$

lemma $\text{sum-wt-ParT-sub-notWtFT-finished-shift[simp]}$:
assumes $cl: \text{properL } cl$ **and** $nf: WtFT \ cl \neq 1$ **and** $n: n < \text{length } cl$ **and** $cln:$
 $\text{finished } (cl \ ! \ n)$
and $I: I \subseteq \{.. < \text{brn } (cl \ ! \ n)\}$
shows $\text{sum } (wt \ (ParT \ cl) \ s) \ (\text{shift } cl \ n \ I) = 0$
 $\langle \text{proof} \rangle$

lemma $\text{sum-wt-ParT-sub-notWtFT-notFinished-shift[simp]}$:
assumes $cl: \text{properL } cl$ **and** $nf: WtFT \ cl \neq 1$
and $n: n < \text{length } cl$ **and** $cln: \neg \text{finished } (cl \ ! \ n)$
and $I: I \subseteq \{.. < \text{brn } (cl \ ! \ n)\}$
shows
 $\text{sum } (wt \ (ParT \ cl) \ s) \ (\text{shift } cl \ n \ I) =$
 $(1 / (\text{length } cl)) / (1 - WtFT \ cl) * \text{sum } (wt \ (cl \ ! \ n) \ s) \ I$
 $\langle \text{proof} \rangle$

definition $UNpart$ **where**
 $UNpart \ cl \ P \equiv \bigcup_{n < \text{length } cl} \text{shift } cl \ n \ ' (P \ n)$

lemma $UNpart\text{-cases[elim, consumes 1, case-names Local]}$:
assumes $II \in UNpart \ cl \ P$ **and**
 $\bigwedge n \ I. \llbracket n < \text{length } cl; I \in P \ n; II = \text{shift } cl \ n \ I \rrbracket \implies \text{phi}$
shows phi

<proof>

lemma *emp-UNpart*:

assumes $\bigwedge n. n < \text{length } cl \implies \{\} \notin P n$

shows $\{\} \notin \text{UNpart } cl P$

<proof>

lemma *part-UNpart*:

assumes *cl*: *properL cl* **and**

P: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n)$

shows $\text{part } \{.. < \text{brnL } cl (\text{length } cl)\} (\text{UNpart } cl P)$

(**is part** ?J ?Q)

<proof>

definition *pickT-pred* **where**

pickT-pred *cl P II n* $\equiv n < \text{length } cl \wedge II \in \text{shift } cl n \text{ ' } (P n)$

definition *pickT* **where**

pickT *cl P II* $\equiv \text{SOME } n. \text{pickT-pred } cl P II n$

lemma *pickT-pred*:

assumes $II \in \text{UNpart } cl P$

shows $\exists n. \text{pickT-pred } cl P II n$

<proof>

lemma *pickT-pred-unique*:

assumes *P*: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

and *1*: *pickT-pred* *cl P II n1* **and** *2*: *pickT-pred* *cl P II n2*

shows $n1 = n2$

<proof>

lemma *pickT-pred-pickT*:

assumes $II \in \text{UNpart } cl P$

shows *pickT-pred* *cl P II* (*pickT* *cl P II*)

<proof>

lemma *pickT-pred-pickT-unique*:

assumes *P*: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

and *pickT-pred* *cl P II n*

shows $n = \text{pickT } cl P II$

<proof>

lemma *pickT-length[simp]*:

assumes $II \in \text{UNpart } cl P$

shows *pickT* *cl P II* $< \text{length } cl$

<proof>

lemma *pickT-shift[simp]*:
assumes $II \in UNpart\ cl\ P$
shows $II \in shift\ cl\ (pickT\ cl\ P\ II) \text{ ' } (P\ (pickT\ cl\ P\ II))$
 $\langle proof \rangle$

lemma *pickT-unique*:
assumes $P: \bigwedge n. n < length\ cl \implies part\ \{.. < brn\ (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$
and $n < length\ cl$ **and** $II \in shift\ cl\ n \text{ ' } (P\ n)$
shows $n = pickT\ cl\ P\ II$
 $\langle proof \rangle$

definition *UNlift where*
 $UNlift\ cl\ dl\ P\ F\ II \equiv$
 $shift\ dl\ (pickT\ cl\ P\ II)\ (F\ (pickT\ cl\ P\ II)\ (back\ cl\ (pickT\ cl\ P\ II)\ II))$

lemma *UNlift-shift[simp]*:
assumes $P: \bigwedge n. n < length\ cl \implies part\ \{.. < brn\ (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$
and $n: n < length\ cl$ **and** $I: I \in P\ n$
shows $UNlift\ cl\ dl\ P\ F\ (shift\ cl\ n\ I) = shift\ dl\ n\ (F\ n\ I)$
 $\langle proof \rangle$

lemma *UNlift-inj-on*:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{.. < brn\ (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$
and $FP: \bigwedge n. n < length\ dl \implies part\ \{.. < brn\ (dl!n)\} (F\ n \text{ ' } (P\ n)) \wedge \{\} \notin F\ n$
 $\text{ ' } (P\ n)$
and $F: \bigwedge n. n < length\ cl \implies inj\text{-on}\ (F\ n)\ (P\ n)$
shows $inj\text{-on}\ (UNlift\ cl\ dl\ P\ F)\ (UNpart\ cl\ P)\ (\text{is}\ inj\text{-on}\ ?G\ ?Q)$
 $\langle proof \rangle$

lemma *UNlift-UNpart*:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{.. < brn\ (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$
shows $(UNlift\ cl\ dl\ P\ F) \text{ ' } (UNpart\ cl\ P) = UNpart\ dl\ (\%n. F\ n \text{ ' } (P\ n))\ (\text{is}\ ?G$
 $\text{ ' } ?Q = ?R)$
 $\langle proof \rangle$

lemma *emp-UNlift-UNpart*:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{.. < brn\ (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$
and $FP: \bigwedge n. n < length\ dl \implies \{\} \notin F\ n \text{ ' } (P\ n)$
shows $\{\} \notin (UNlift\ cl\ dl\ P\ F) \text{ ' } (UNpart\ cl\ P)\ (\text{is}\ \{\} \notin ?R)$
 $\langle proof \rangle$

lemma *part-UNlift-UNpart*:
assumes $l: length\ cl = length\ dl$ **and** $dl: properL\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{.. < brn\ (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$
and $FP: \bigwedge n. n < length\ dl \implies part\ \{.. < brn\ (dl!n)\} (F\ n \text{ ' } (P\ n))$
shows $part\ \{.. < brnL\ dl\ (length\ dl)\} ((UNlift\ cl\ dl\ P\ F) \text{ ' } (UNpart\ cl\ P))\ (\text{is}\ part$

?C ?R)
 ⟨proof⟩

lemma *ss-wt-Par-UNlift*:

assumes *l*: $\text{length } cl = \text{length } dl$

and *cdl*: $\text{properL } cl \text{ properL } dl$ **and** *II*: $II \in \text{UNpart } cl \ P$

and *P*: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P \ n) \wedge \{\} \notin P \ n$

and *FP*: $\bigwedge n. n < \text{length } dl \implies \text{part } \{.. < \text{brn } (dl!n)\} (F \ n \ ' (P \ n))$

and *sw*:

$\bigwedge n \ I. \llbracket n < \text{length } cl; I \in P \ n \rrbracket \implies$

$\text{sum } (wt \ (cl \ ! \ n) \ s) \ I =$

$\text{sum } (wt \ (dl \ ! \ n) \ t) \ (F \ n \ I)$

and *st*: $s \approx t$

shows

$\text{sum } (wt \ (Par \ cl) \ s) \ II =$

$\text{sum } (wt \ (Par \ dl) \ t) \ (\text{UNlift } cl \ dl \ P \ F \ II) \ (\text{is } ?L = ?R)$

⟨proof⟩

definition *thetaSPar* **where**

thetaSPar \equiv

$\{(Par \ cl, \ Par \ dl) \mid$
 $\quad cl \ dl. \ \text{properL } cl \wedge \text{properL } dl \wedge \text{SbisL } cl \ dl\}$

lemma *cont-eff-Par-UNlift*:

assumes *l*: $\text{length } cl = \text{length } dl$

and *cdl*: $\text{properL } cl \ \text{properL } dl \ \text{SbisL } cl \ dl$

and *II*: $II \in \text{UNpart } cl \ P$ **and** *ii*: $ii \in II$ **and** *jj*: $jj \in \text{UNlift } cl \ dl \ P \ F \ II$

and *P*: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P \ n) \wedge \{\} \notin P \ n$

and *FP*: $\bigwedge n. n < \text{length } dl \implies \text{part } \{.. < \text{brn } (dl!n)\} (F \ n \ ' (P \ n))$

and *eff-cont*:

$\bigwedge n \ I \ i \ j. \llbracket n < \text{length } cl; I \in P \ n; i \in I; j \in F \ n \ I \rrbracket \implies$

$\text{eff } (cl!n) \ s \ i \approx \text{eff } (dl!n) \ t \ j \wedge$

$\text{cont } (cl!n) \ s \ i \approx \text{cont } (dl!n) \ t \ j$

and *st*: $s \approx t$

shows

$\text{eff } (Par \ cl) \ s \ ii \approx \text{eff } (Par \ dl) \ t \ jj \wedge$

$(\text{cont } (Par \ cl) \ s \ ii, \ \text{cont } (Par \ dl) \ t \ jj) \in \text{thetaSPar}$

(**is** ?eff \wedge ?cont)

⟨proof⟩

lemma *thetaSPar-Sretr*: $\text{thetaSPar} \subseteq \text{Sretr } (\text{thetaSPar})$

⟨proof⟩

lemma *thetaSPar-Sbis*: $\text{thetaSPar} \subseteq \text{Sbis}$

⟨proof⟩

theorem *Par-Sbis[simp]*:

assumes *properL cl and properL dl SbisL cl dl*
shows *Par cl \approx_s Par dl*
 ⟨*proof*⟩

4.5 01-bisimilarity versus language constructs

lemma *ZObis-pres-discr-L: $c \approx_{01} d \implies \text{discr } d \implies \text{discr } c$*
 ⟨*proof*⟩

theorem *ZObis-pres-discr-R:*
assumes *discr c and $c \approx_{01} d$*
shows *discr d*
 ⟨*proof*⟩

theorem *ZObis-finished-discr-L:*
assumes *$c \approx_{01} d$ and proper d and finished d*
shows *discr c*
 ⟨*proof*⟩

theorem *ZObis-finished-discr-R:*
assumes *proper c and finished c and $c \approx_{01} d$*
shows *discr d*
 ⟨*proof*⟩

theorem *discr-ZObis[simp]:*
assumes *proper c and proper d and discr c and discr d*
shows *$c \approx_{01} d$*
 ⟨*proof*⟩

theorem *Done-ZObis[simp]:*
Done \approx_{01} Done
 ⟨*proof*⟩

theorem *Atm-ZObis[simp]:*
assumes *compatAtm atm*
shows *Atm atm \approx_{01} Atm atm*
 ⟨*proof*⟩

definition *thetaZOSeqI where*
thetaZOSeqI \equiv
 $\{(e ;; c, e ;; d) \mid e c d . \text{siso } e \wedge c \approx_{01} d\}$

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

lemma *thetaZOSeqI-ZObis*:

thetaZOSeqI \subseteq *ZObis*

<proof>

theorem *Seq-iso-ZObis[simp]*:

assumes *iso e* **and** *c2 \approx 01 d2*

shows *e* ;; *c2 \approx 01 e* ;; *d2*

<proof>

definition *thetaZOSeqD* **where**

thetaZOSeqD \equiv

$\{(c1 \text{ ;; } c2, d1 \text{ ;; } d2) \mid$

$c1 \text{ } c2 \text{ } d1 \text{ } d2.$

$\text{proper } c1 \wedge \text{proper } d1 \wedge \text{proper } c2 \wedge \text{proper } d2 \wedge$

$\text{discr } c2 \wedge \text{discr } d2 \wedge$

$c1 \approx 01 d1\}$

lemma *thetaZOSeqD-ZOretr*:

thetaZOSeqD \subseteq *ZOretr* (*thetaZOSeqD Un ZObis*)

<proof>

lemma *thetaZOSeqD-ZObis*:

thetaZOSeqD \subseteq *ZObis*

<proof>

theorem *Seq-ZObis[simp]*:

assumes *proper c1* **and** *proper d1* **and** *proper c2* **and** *proper d2*

and *c1 \approx 01 d1* **and** *discr c2* **and** *discr d2*

shows *c1* ;; *c2 \approx 01 d1* ;; *d2*

<proof>

definition *thetaZOCh* **where**

thetaZOCh ch c1 c2 d1 d2 $\equiv \{(Ch \text{ } ch \text{ } c1 \text{ } c2, Ch \text{ } ch \text{ } d1 \text{ } d2)\}$

lemma *thetaZOCh-Sretr*:

assumes *compatCh ch* **and** *c1 \approx 01 d1* **and** *c2 \approx 01 d2*

shows *thetaZOCh ch c1 c2 d1 d2* \subseteq

Sretr (thetaZOCh ch c1 c2 d1 d2 \cup ZObis)

(**is** *?th* \subseteq *Sretr (?th \cup ZObis)*)

<proof>

lemma *thetaZOCh-ZOretr*:

assumes *compatCh ch* **and** *c1 \approx 01 d1* **and** *c2 \approx 01 d2*

shows *thetaZOCh ch c1 c2 d1 d2* \subseteq

ZOretr (thetaZOCh ch c1 c2 d1 d2 \cup ZObis)

<proof>

lemma *thetaZOCh-ZObis*:

assumes *compatCh ch and c1 ≈01 d1 and c2 ≈01 d2*

shows *thetaZOCh ch c1 c2 d1 d2 ⊆ ZObis*

<proof>

theorem *Ch-siso-ZObis[simp]*:

assumes *compatCh ch and c1 ≈01 d1 and c2 ≈01 d2*

shows *Ch ch c1 c2 ≈01 Ch ch d1 d2*

<proof>

definition *theFTOne where*

theFTOne cl dl ≡ theFT cl ∪ theFT dl

definition *theNFTBoth where*

theNFTBoth cl dl ≡ theNFT cl ∩ theNFT dl

lemma *theFTOne-sym*: *theFTOne cl dl = theFTOne dl cl*

<proof>

lemma *finite-theFTOne[simp]*:

finite (theFTOne cl dl)

<proof>

lemma *theFTOne-length-finished[simp]*:

assumes *n ∈ theFTOne cl dl*

shows *(n < length cl ∧ finished (cl!n)) ∨ (n < length dl ∧ finished (dl!n))*

<proof>

lemma *theFTOne-length[simp]*:

assumes *length cl = length dl and n ∈ theFTOne cl dl*

shows *n < length cl and n < length dl*

<proof>

lemma *theFTOne-intro[intro]*:

assumes $\bigwedge n. (n < \text{length } cl \wedge \text{finished } (cl!n)) \vee (n < \text{length } dl \wedge \text{finished } (dl!n))$

shows *n ∈ theFTOne cl dl*

<proof>

lemma *pickFT-theFTOne[simp]*:

assumes *WtFT cl = 1*

shows *pickFT cl ∈ theFTOne cl dl*

<proof>

lemma *finite-theNFTBoth[simp]*:

finite (theNFTBoth cl dl)

<proof>

lemma *theNFTBoth-sym*: $theNFTBoth\ cl\ dl = theNFTBoth\ dl\ cl$
 ⟨proof⟩

lemma *theNFTBoth-length-finished*[simp]:
assumes $n \in theNFTBoth\ cl\ dl$
shows $n < length\ cl$ **and** $\neg finished\ (cl!n)$
and $n < length\ dl$ **and** $\neg finished\ (dl!n)$
 ⟨proof⟩

lemma *theNFTBoth-intro*[intro]:
assumes $\bigwedge n. n < length\ cl \wedge \neg finished\ (cl!n) \wedge n < length\ dl \wedge \neg finished\ (dl!n)$
shows $n \in theNFTBoth\ cl\ dl$
 ⟨proof⟩

lemma *theFTOne-Int-theNFTBoth*[simp]:
 $theFTOne\ cl\ dl \cap theNFTBoth\ cl\ dl = \{\}$
and $theNFTBoth\ cl\ dl \cap theFTOne\ cl\ dl = \{\}$
 ⟨proof⟩

lemma *theFT-Un-theNFT-One-Both*[simp]:
assumes $length\ cl = length\ dl$
shows
 $theFTOne\ cl\ dl \cup theNFTBoth\ cl\ dl = \{.. < length\ cl\}$ **and**
 $theNFTBoth\ cl\ dl \cup theFTOne\ cl\ dl = \{.. < length\ cl\}$
 ⟨proof⟩

lemma *in-theFTOne-theNFTBoth*[simp]:
assumes $n1 \in theFTOne\ cl\ dl$ **and** $n2 \in theNFTBoth\ cl\ dl$
shows $n1 \neq n2$ **and** $n2 \neq n1$
 ⟨proof⟩

definition *BrnFT* **where**
 $BrnFT\ cl\ dl \equiv \bigcup n \in theFTOne\ cl\ dl. \{brnL\ cl\ n ..<+ brn\ (cl!n)\}$

definition *BrnNFT* **where**
 $BrnNFT\ cl\ dl \equiv \bigcup n \in theNFTBoth\ cl\ dl. \{brnL\ cl\ n ..<+ brn\ (cl!n)\}$

lemma *BrnFT-elim*[elim, consumes 1, case-names Local]:
assumes $ii \in BrnFT\ cl\ dl$
and $\bigwedge n\ i. \llbracket n \in theFTOne\ cl\ dl; i < brn\ (cl!n); ii = brnL\ cl\ n + i \rrbracket \implies phi$
shows phi
 ⟨proof⟩

lemma *finite-BrnFT*[simp]:

finite (*BrnFT cl dl*)
 ⟨*proof*⟩

lemma *BrnFT-incl-brnL[simp]*:
assumes *l*: *length cl = length dl* **and** *cl*: *properL cl*
shows *BrnFT cl dl* \subseteq $\{..
 \langle \textit{proof} \rangle$

lemma *BrnNFT-elim[elim, consumes 1, case-names Local]*:
assumes *ii* \in *BrnNFT cl dl*
and $\bigwedge n i. \llbracket n \in \textit{theNFTBoth cl dl}; i < \textit{brn} (cl!n); ii = \textit{brnL cl } n + i \rrbracket \implies \textit{phi}$
shows *phi*
 ⟨*proof*⟩

lemma *finite-BrnNFT[simp]*:
finite (*BrnNFT cl dl*)
 ⟨*proof*⟩

lemma *BrnNFT-incl-brnL[simp]*:
assumes *cl*: *properL cl*
shows *BrnNFT cl dl* \subseteq $\{..
 \langle \textit{proof} \rangle$

lemma *BrnFT-Int-BrnNFT[simp]*:
assumes *l*: *length cl = length dl*
shows
BrnFT cl dl \cap *BrnNFT cl dl* = $\{\}$ (**is** ?*L*)
and *BrnNFT cl dl* \cap *BrnFT cl dl* = $\{\}$ (**is** ?*R*)
 ⟨*proof*⟩

lemma *BrnFT-Un-BrnNFT[simp]*:
assumes *l*: *length cl = length dl* **and** *cl*: *properL cl*
shows *BrnFT cl dl* \cup *BrnNFT cl dl* = $\{..
 \langle \textit{proof} \rangle$

lemma *BrnFT-part*:
assumes *l*: *length cl = length dl*
and *P*: $\bigwedge n. n < \textit{length cl} \implies \textit{part} \{..
 \langle \textit{proof} \rangle$

lemma *brnL-pickFT-BrnFT[simp]*:
assumes *properL cl* **and** *WtFT cl = 1*
shows *brnL cl (pickFT cl)* \in *BrnFT cl dl*
 ⟨*proof*⟩

lemma *WtFT-ParT-BrnFT[simp]*:

assumes $\text{length } cl = \text{length } dl \text{ properL } cl \text{ and } \text{WtFT } cl = 1$
shows $\text{sum } (wt (ParT \text{ } cl) \text{ } s) (BrnFT \text{ } cl \text{ } dl) = 1$
 $\langle \text{proof} \rangle$

definition $UNpart1$ **where**
 $UNpart1 \text{ } cl \text{ } dl \text{ } P \equiv \bigcup n \in \text{theNFTBoth } cl \text{ } dl. \text{shift } cl \text{ } n \text{ ' } (P \text{ } n)$

definition $UNpart01$ **where**
 $UNpart01 \text{ } cl \text{ } dl \text{ } P \equiv \{BrnFT \text{ } cl \text{ } dl\} \cup UNpart1 \text{ } cl \text{ } dl \text{ } P$

lemma $BrnFT-UNpart01[simp]$:
 $BrnFT \text{ } cl \text{ } dl \in UNpart01 \text{ } cl \text{ } dl \text{ } P$
 $\langle \text{proof} \rangle$

lemma $UNpart1-cases[elim, consumes 1, case-names Local]$:
assumes $II \in UNpart1 \text{ } cl \text{ } dl \text{ } P$
 $\bigwedge n \text{ } I. \llbracket n \in \text{theNFTBoth } cl \text{ } dl; I \in P \text{ } n; II = \text{shift } cl \text{ } n \text{ } I \rrbracket \implies phi$
shows phi
 $\langle \text{proof} \rangle$

lemma $UNpart01-cases[elim, consumes 1, case-names Local0 Local]$:
assumes $II \in UNpart01 \text{ } cl \text{ } dl \text{ } P$ **and** $II = BrnFT \text{ } cl \text{ } dl \implies phi$
 $\bigwedge n \text{ } I. \llbracket n \in \text{theNFTBoth } cl \text{ } dl; I \in P \text{ } n; II = \text{shift } cl \text{ } n \text{ } I; II \in UNpart1 \text{ } cl \text{ } dl \text{ } P \rrbracket$
 $\implies phi$
shows phi
 $\langle \text{proof} \rangle$

lemma $emp-UNpart1$:
assumes $\bigwedge n. n < \text{length } cl \implies \{\} \notin P \text{ } n$
shows $\{\} \notin UNpart1 \text{ } cl \text{ } dl \text{ } P$
 $\langle \text{proof} \rangle$

lemma $emp-UNpart01$:
assumes $\bigwedge n. n < \text{length } cl \implies \{\} \notin P \text{ } n$
shows $\{\} \notin UNpart01 \text{ } cl \text{ } dl \text{ } P - \{BrnFT \text{ } cl \text{ } dl\}$
 $\langle \text{proof} \rangle$

lemma $BrnFT-Int-UNpart1[simp]$:
assumes $l: \text{length } cl = \text{length } dl$
and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P \text{ } n) \wedge \{\} \notin P \text{ } n$
and $II: II \in UNpart1 \text{ } cl \text{ } dl \text{ } P$
shows $BrnFT \text{ } cl \text{ } dl \cap II = \{\}$
 $\langle \text{proof} \rangle$

lemma $BrnFT-notIn-UNpart1$:
assumes $l: \text{length } cl = \text{length } dl$
and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P \text{ } n) \wedge \{\} \notin P \text{ } n$

shows $BrnFT\ cl\ dl \notin UNpart1\ cl\ dl\ P$
 ⟨proof⟩

lemma $UNpart1-UNpart01$:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{..< brn\ (cl!n)\}\ (P\ n) \wedge \{\}\ \notin P\ n$
shows $UNpart1\ cl\ dl\ P = UNpart01\ cl\ dl\ P - \{BrnFT\ cl\ dl\}$
 ⟨proof⟩

lemma $part-UNpart1[simp]$:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{..< brn\ (cl!n)\}\ (P\ n)$
shows $part\ (BrnNFT\ cl\ dl)\ (UNpart1\ cl\ dl\ P)$
 ⟨proof⟩

lemma $part-UNpart01$:
assumes $cl: properL\ cl$ **and** $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{..< brn\ (cl!n)\}\ (P\ n) \wedge \{\}\ \notin P\ n$
shows $part\ \{..< brnL\ cl\ (length\ cl)\}\ (UNpart01\ cl\ dl\ P)$
 ⟨proof⟩

definition $UNlift01$ **where**
 $UNlift01\ cl\ dl\ P\ F\ II \equiv$
 if $II = BrnFT\ cl\ dl$
 then $BrnFT\ dl\ cl$
 else $shift\ dl\ (pickT\ cl\ P\ II)\ (F\ (pickT\ cl\ P\ II)\ (back\ cl\ (pickT\ cl\ P\ II)\ II))$

lemma $UNlift01-BrnFT[simp]$:
 $UNlift01\ cl\ dl\ P\ F\ (BrnFT\ cl\ dl) = BrnFT\ dl\ cl$
 ⟨proof⟩

lemma $UNlift01-shift[simp]$:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{..< brn\ (cl!n)\}\ (P\ n) \wedge \{\}\ \notin P\ n$
and $n: n \in theNFTBoth\ cl\ dl$ **and** $I: I \in P\ n$
shows $UNlift01\ cl\ dl\ P\ F\ (shift\ cl\ n\ I) = shift\ dl\ n\ (F\ n\ I)$
 ⟨proof⟩

lemma $UNlift01-inj-on-UNpart1$:
assumes $l: length\ cl = length\ dl$
and $P: \bigwedge n. n < length\ cl \implies part\ \{..< brn\ (cl!n)\}\ (P\ n) \wedge \{\}\ \notin P\ n$
and $FP: \bigwedge n. n < length\ dl \implies part\ \{..< brn\ (dl!n)\}\ (F\ n\ ' (P\ n)) \wedge \{\}\ \notin F\ n$
 ' $(P\ n)$
and $F: \bigwedge n. n < length\ cl \implies inj-on\ (F\ n)\ (P\ n)$
shows $inj-on\ (UNlift01\ cl\ dl\ P\ F)\ (UNpart1\ cl\ dl\ P)\ (is\ inj-on\ ?G\ ?Q)$
 ⟨proof⟩

lemma *inj-on-singl*:

assumes *inj-on* $f A$ **and** $a0 \notin A$ **and** $\bigwedge a. a \in A \implies f a \neq f a0$

shows *inj-on* $f (\{a0\} \text{ Un } A)$

<proof>

lemma *UNlift01-inj-on*:

assumes $l: \text{length } cl = \text{length } dl$

and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

and $FP: \bigwedge n. n < \text{length } dl \implies \text{part } \{.. < \text{brn } (dl!n)\} (F n \text{ ' } (P n)) \wedge \{\} \notin F n \text{ ' } (P n)$

and $F: \bigwedge n. n < \text{length } cl \implies \text{inj-on } (F n) (P n)$

shows *inj-on* $(\text{UNlift01 } cl \ dl \ P \ F) (\text{UNpart01 } cl \ dl \ P)$

<proof>

lemma *UNlift01-UNpart1*:

assumes $l: \text{length } cl = \text{length } dl$

and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

shows $(\text{UNlift01 } cl \ dl \ P \ F) \text{ ' } (\text{UNpart1 } cl \ dl \ P) = \text{UNpart1 } dl \ cl \ (\%n. F n \text{ ' } (P n))$ **(is** $?G \text{ ' } ?Q = ?R$)

<proof>

lemma *UNlift01-UNpart01*:

assumes $l: \text{length } cl = \text{length } dl$

and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

shows $(\text{UNlift01 } cl \ dl \ P \ F) \text{ ' } (\text{UNpart01 } cl \ dl \ P) = \text{UNpart01 } dl \ cl \ (\%n. F n \text{ ' } (P n))$

<proof>

lemma *emp-UNlift01-UNpart1*:

assumes $l: \text{length } cl = \text{length } dl$

and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

and $FP: \bigwedge n. n < \text{length } dl \implies \{\} \notin F n \text{ ' } (P n)$

shows $\{\} \notin (\text{UNlift01 } cl \ dl \ P \ F) \text{ ' } (\text{UNpart1 } cl \ dl \ P)$ **(is** $\{\} \notin ?R$)

<proof>

lemma *emp-UNlift01-UNpart01*:

assumes $l: \text{length } cl = \text{length } dl$

and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

and $FP: \bigwedge n. n < \text{length } dl \implies \{\} \notin F n \text{ ' } (P n)$

shows $\{\} \notin (\text{UNlift01 } cl \ dl \ P \ F) \text{ ' } (\text{UNpart01 } cl \ dl \ P - \{\text{BrnFT } cl \ dl\})$

(is $\{\} \notin ?U \text{ ' } ?V$)

<proof>

lemma *part-UNlift01-UNpart1*:

assumes $l: \text{length } cl = \text{length } dl$ **and** $dl: \text{properL } dl$

and $P: \bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P n) \wedge \{\} \notin P n$

and $FP: \bigwedge n. n < \text{length } dl \implies \text{part } \{.. < \text{brn } (dl!n)\} (F n \text{ ' } (P n))$

shows $\text{part } (\text{BrnNFT } dl \ cl) ((\text{UNlift01 } cl \ dl \ P \ F) \text{ ' } (\text{UNpart1 } cl \ dl \ P))$ **(is** $\text{part } ?C \text{ ' } ?R$)

<proof>

lemma *part-UNlift01-UNpart01*:

assumes *l*: $\text{length } cl = \text{length } dl$ **and** *dl*: *properL dl*

and *P*: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$

and *FP*: $\bigwedge n. n < \text{length } dl \implies \text{part } \{.. < \text{brn } (dl!n)\} (F\ n\ ' (P\ n)) \wedge \{\} \notin (F\ n\ ' (P\ n))$

shows $\text{part } \{.. < \text{brnL } dl\ (\text{length } dl)\} ((UNlift01\ cl\ dl\ P\ F)\ ' (UNpart01\ cl\ dl\ P))$

(is part ?K ?R)

<proof>

lemma *diff-frac-eq-1*:

assumes *b* $\neq (0::\text{real})$

shows $1 - a / b = (b - a) / b$

<proof>

lemma *diff-frac-eq-2*:

assumes *b* $\neq (1::\text{real})$

shows $1 - (a - b) / (1 - b) = (1 - a) / (1 - b)$

(is ?L = ?R)

<proof>

lemma *triv-div-mult*:

assumes *vSF*: *vSF* $\neq (1::\text{real})$

and *L*: $L = (K - vSF) / (1 - vSF)$ **and** *Ln*: $L \neq 1$

shows $(VS / (1 - vSF) * V) / (1 - L) = (VS * V) / (1 - K)$

(is ?A = ?B)

<proof>

lemma *ss-wt-ParT-UNlift01*:

assumes *l*: $\text{length } cl = \text{length } dl$

and *cdl*: *properL cl properL dl* **and** *II*: $II \in UNpart01\ cl\ dl\ P - \{\text{BrnFT } cl\ dl\}$

and *P*: $\bigwedge n. n < \text{length } cl \implies \text{part } \{.. < \text{brn } (cl!n)\} (P\ n) \wedge \{\} \notin P\ n$

and *FP*: $\bigwedge n. n < \text{length } dl \implies \text{part } \{.. < \text{brn } (dl!n)\} (F\ n\ ' (P\ n))$

and *sw*:

$\bigwedge n\ I. \llbracket n < \text{length } cl; I \in P\ n \rrbracket \implies$

$\text{sum } (wt\ (cl\ !\ n)\ s)\ I =$

$\text{sum } (wt\ (dl\ !\ n)\ t)\ (F\ n\ I)$

and *st*: $s \approx t$

and *le1*: $\text{sum } (wt\ (\text{ParT } cl)\ s)\ (\text{BrnFT } cl\ dl) < 1$

$\text{sum } (wt\ (\text{ParT } dl)\ t)\ (\text{BrnFT } dl\ cl) < 1$

shows

$\text{sum } (wt\ (\text{ParT } cl)\ s)\ II /$

$(1 - \text{sum } (wt\ (\text{ParT } cl)\ s)\ (\text{BrnFT } cl\ dl)) =$

$\text{sum } (wt\ (\text{ParT } dl)\ t)\ (UNlift01\ cl\ dl\ P\ F\ II) /$

$(1 - \text{sum } (wt\ (\text{ParT } dl)\ t)\ (\text{BrnFT } dl\ cl))$

(is sum ?vP II / (1 - sum ?vP ?II-0) =

$sum \ ?wP \ ?JJ / (1 - sum \ ?wP \ ?JJ-0)$
 <proof>

definition *thetaZOParT* **where**

thetaZOParT \equiv
 $\{(ParT \ cl, \ ParT \ dl) \mid$
 $\quad cl \ dl.$
 $\quad properL \ cl \wedge \ properL \ dl \wedge \ SbisL \ cl \ dl\}$

lemma *cont-eff-ParT-BrnFT-L*:

assumes l : $length \ cl = length \ dl$
and $cldl$: $properL \ cl \ properL \ dl \ SbisL \ cl \ dl$
and ii : $ii \in BrnFT \ cl \ dl$
and *eff-cont*:

$\bigwedge n \ I \ i \ j. \llbracket n < length \ cl; \ I \in P \ n; \ i \in I; \ j \in F \ n \ I \rrbracket \implies$
 $\quad eff \ (cl!n) \ s \ i \approx eff \ (dl!n) \ t \ j \wedge$
 $\quad cont \ (cl!n) \ s \ i \approx_s cont \ (dl!n) \ t \ j$

shows

$s \approx eff \ (ParT \ cl) \ s \ ii \wedge$
 $\quad (cont \ (ParT \ cl) \ s \ ii, \ ParT \ dl) \in \ thetaZOParT$
 (is $?eff \wedge ?cont$)
 <proof>

lemma *cont-eff-ParT-BrnFT-R*:

assumes l : $length \ cl = length \ dl$
and $cldl$: $properL \ cl \ properL \ dl \ SbisL \ cl \ dl$
and jj : $jj \in BrnFT \ dl \ cl$
and *eff-cont*:

$\bigwedge n \ I \ i \ j. \llbracket n < length \ cl; \ I \in P \ n; \ i \in I; \ j \in F \ n \ I \rrbracket \implies$
 $\quad eff \ (cl!n) \ s \ i \approx eff \ (dl!n) \ t \ j \wedge cont \ (cl!n) \ s \ i \approx_s cont \ (dl!n) \ t \ j$

shows

$t \approx eff \ (ParT \ dl) \ t \ jj \wedge$
 $\quad (ParT \ cl, \ cont \ (ParT \ dl) \ t \ jj) \in \ thetaZOParT$
 (is $?eff \wedge ?cont$)
 <proof>

lemma *cont-eff-ParT-UNlift01*:

assumes l : $length \ cl = length \ dl$
and $cldl$: $properL \ cl \ properL \ dl \ SbisL \ cl \ dl$
and II : $II \in UNpart01 \ cl \ dl \ P - \{BrnFT \ cl \ dl\}$
and ii : $ii \in II$ **and** jj : $jj \in UNlift01 \ cl \ dl \ P \ F \ II$
and P : $\bigwedge n. \ n < length \ cl \implies part \ \{..< brn \ (cl!n)\} \ (P \ n) \wedge \{\} \notin P \ n$
and FP : $\bigwedge n. \ n < length \ dl \implies part \ \{..< brn \ (dl!n)\} \ (F \ n \ ' \ (P \ n))$
and *eff-cont*:

$\bigwedge n \ I \ i \ j. \llbracket n < length \ cl; \ I \in P \ n; \ i \in I; \ j \in F \ n \ I \rrbracket \implies$
 $\quad eff \ (cl!n) \ s \ i \approx$
 $\quad eff \ (dl!n) \ t \ j \wedge$

```

    cont (cl!n) s i ≈ s
    cont (dl!n) t j
and st: s ≈ t
shows
  eff (ParT cl) s ii ≈ eff (ParT dl) t jj ∧
  (cont (ParT cl) s ii, cont (ParT dl) t jj) ∈ thetaZOParT
(is ?eff ∧ ?cont)
⟨proof⟩

```

```

lemma thetaZOParT-ZOretr: thetaZOParT ⊆ ZOretr (thetaZOParT)
⟨proof⟩

```

```

lemma thetaZOParT-ZObis: thetaZOParT ⊆ ZObis
⟨proof⟩

```

```

theorem ParT-ZObis[simp]:
assumes properL cl and properL dl and SbisL cl dl
shows ParT cl ≈01 ParT dl
⟨proof⟩

```

end

end

5 Syntactic Criteria

```

theory Syntactic-Criteria
  imports Compositionality
begin

```

```

context PL-Indis
begin

```

```

lemma proper-intros[intro]:
  proper Done
  proper (Atm atm)
  proper c1 ⇒ proper c2 ⇒ proper (Seq c1 c2)
  proper c1 ⇒ proper c2 ⇒ proper (Ch ch c1 c2)
  proper c ⇒ proper (While tst c)
  properL cs ⇒ proper (Par cs)
  properL cs ⇒ proper (ParT cs)
  (∧c. c ∈ set cs ⇒ proper c) ⇒ cs ≠ [] ⇒ properL cs
⟨proof⟩

```

```

lemma discr:
  discr Done

```

$presAtm\ atm \implies discr\ (Atm\ atm)$
 $discr\ c1 \implies discr\ c2 \implies discr\ (Seq\ c1\ c2)$
 $discr\ c1 \implies discr\ c2 \implies discr\ (Ch\ ch\ c1\ c2)$
 $discr\ c \implies discr\ (While\ tst\ c)$
 $properL\ cs \implies (\bigwedge c. c \in set\ cs \implies discr\ c) \implies discr\ (Par\ cs)$
 $properL\ cs \implies (\bigwedge c. c \in set\ cs \implies discr\ c) \implies discr\ (ParT\ cs)$
 $\langle proof \rangle$

lemma *siso*:

$compatAtm\ atm \implies siso\ (Atm\ atm)$
 $siso\ c1 \implies siso\ c2 \implies siso\ (Seq\ c1\ c2)$
 $compatCh\ ch \implies siso\ c1 \implies siso\ c2 \implies siso\ (Ch\ ch\ c1\ c2)$
 $compatTst\ tst \implies siso\ c \implies siso\ (While\ tst\ c)$
 $properL\ cs \implies (\bigwedge c. c \in set\ cs \implies siso\ c) \implies siso\ (Par\ cs)$
 $properL\ cs \implies (\bigwedge c. c \in set\ cs \implies siso\ c) \implies siso\ (ParT\ cs)$
 $\langle proof \rangle$

lemma *Sbis*:

$compatAtm\ atm \implies Atm\ atm \approx_s Atm\ atm$
 $siso\ c1 \implies c2 \approx_s c2 \implies Seq\ c1\ c2 \approx_s Seq\ c1\ c2$
 $proper\ c1 \implies proper\ c2 \implies c1 \approx_s c1 \implies discr\ c2 \implies Seq\ c1\ c2 \approx_s Seq\ c1\ c2$
 $compatCh\ ch \implies c1 \approx_s c1 \implies c2 \approx_s c2 \implies Ch\ ch\ c1\ c2 \approx_s Ch\ ch\ c1\ c2$
 $properL\ cs \implies (\bigwedge c. c \in set\ cs \implies c \approx_s c) \implies Par\ cs \approx_s Par\ cs$
 $\langle proof \rangle$

lemma *ZObis*:

$compatAtm\ atm \implies Atm\ atm \approx_{01} Atm\ atm$
 $siso\ c1 \implies c2 \approx_{01} c2 \implies Seq\ c1\ c2 \approx_{01} Seq\ c1\ c2$
 $proper\ c1 \implies proper\ c2 \implies c1 \approx_{01} c1 \implies discr\ c2 \implies Seq\ c1\ c2 \approx_{01} Seq\ c1\ c2$
 $compatCh\ ch \implies c1 \approx_{01} c1 \implies c2 \approx_{01} c2 \implies Ch\ ch\ c1\ c2 \approx_{01} Ch\ ch\ c1\ c2$
 $properL\ cs \implies (\bigwedge c. c \in set\ cs \implies c \approx_s c) \implies ParT\ cs \approx_{01} ParT\ cs$
 $\langle proof \rangle$

lemma *discr-imp-Sbis*: $proper\ c \implies discr\ c \implies c \approx_s c$
 $\langle proof \rangle$

lemma *siso-imp-Sbis*: $siso\ c \implies c \approx_s c$
 $\langle proof \rangle$

lemma *Sbis-imp-ZObis*: $c \approx_s c \implies c \approx_{01} c$
 $\langle proof \rangle$

fun *SC-discr* **where**

$SC-discr\ Done \iff True$
 $| SC-discr\ (Atm\ atm) \iff presAtm\ atm$

```

| SC-discr (Seq c1 c2)  $\longleftrightarrow$  SC-discr c1  $\wedge$  SC-discr c2
| SC-discr (Ch ch c1 c2)  $\longleftrightarrow$  SC-discr c1  $\wedge$  SC-discr c2
| SC-discr (While tst c)  $\longleftrightarrow$  SC-discr c
| SC-discr (ParT cs)  $\longleftrightarrow$  ( $\forall c \in \text{set } cs. \textit{SC-discr } c$ )
| SC-discr (Par cs)  $\longleftrightarrow$  ( $\forall c \in \text{set } cs. \textit{SC-discr } c$ )

```

theorem *SC-discr-discr*[intro]: *proper* c \implies *SC-discr* c \implies *discr* c
 ⟨*proof*⟩

fun *SC-iso* **where**

```

  SC-iso Done  $\longleftrightarrow$  True
| SC-iso (Atm atm)  $\longleftrightarrow$  compatAtm atm
| SC-iso (Seq c1 c2)  $\longleftrightarrow$  SC-iso c1  $\wedge$  SC-iso c2
| SC-iso (Ch ch c1 c2)  $\longleftrightarrow$  compatCh ch  $\wedge$  SC-iso c1  $\wedge$  SC-iso c2
| SC-iso (While tst c)  $\longleftrightarrow$  compatTst tst  $\wedge$  SC-iso c
| SC-iso (Par cs)  $\longleftrightarrow$  ( $\forall c \in \text{set } cs. \textit{SC-iso } c$ )
| SC-iso (ParT cs)  $\longleftrightarrow$  ( $\forall c \in \text{set } cs. \textit{SC-iso } c$ )

```

theorem *SC-iso-iso*[intro]: *proper* c \implies *SC-iso* c \implies *iso* c
 ⟨*proof*⟩

fun *SC-Sbis* **where**

```

  SC-Sbis Done  $\longleftrightarrow$  True
| SC-Sbis (Atm atm)  $\longleftrightarrow$  compatAtm atm
| SC-Sbis (Seq c1 c2)  $\longleftrightarrow$  (SC-iso c1  $\wedge$  SC-Sbis c2)  $\vee$ 
  (SC-Sbis c1  $\wedge$  SC-discr c2)  $\vee$ 
  SC-discr (Seq c1 c2)  $\vee$  SC-iso (Seq c1 c2)
| SC-Sbis (Ch ch c1 c2)  $\longleftrightarrow$  (if compatCh ch
  then SC-Sbis c1  $\wedge$  SC-Sbis c2
  else (SC-discr (Ch ch c1 c2)  $\vee$  SC-iso (Ch ch c1 c2)))
| SC-Sbis (While tst c)  $\longleftrightarrow$  SC-discr (While tst c)  $\vee$  SC-iso (While tst c)
| SC-Sbis (Par cs)  $\longleftrightarrow$  ( $\forall c \in \text{set } cs. \textit{SC-Sbis } c$ )
| SC-Sbis (ParT cs)  $\longleftrightarrow$  SC-iso (ParT cs)  $\vee$  SC-discr (ParT cs)

```

theorem *SC-iso-SCbis*[intro]: *SC-iso* c \implies *SC-Sbis* c
 ⟨*proof*⟩

theorem *SC-discr-SCbis*[intro]: *SC-discr* c \implies *SC-Sbis* c
 ⟨*proof*⟩

declare *SC-iso.simps*[simp del]

declare *SC-discr.simps*[simp del]

theorem *SC-Sbis-Sbis*[intro]: *proper* c \implies *SC-Sbis* c \implies c \approx s c
 ⟨*proof*⟩

fun *SC-ZObis* **where**

```

  SC-ZObis Done  $\longleftrightarrow$  True

```

```

| SC-ZObis (Atm atm)       $\longleftrightarrow$  compatAtm atm
| SC-ZObis (Seq c1 c2)    $\longleftrightarrow$  (SC-siso c1  $\wedge$  SC-ZObis c2)  $\vee$ 
                             (SC-ZObis c1  $\wedge$  SC-discr c2)  $\vee$ 
                             SC-Sbis (Seq c1 c2)
| SC-ZObis (Ch ch c1 c2)  $\longleftrightarrow$  (if compatCh ch
                             then SC-ZObis c1  $\wedge$  SC-ZObis c2
                             else SC-Sbis (Ch ch c1 c2))
| SC-ZObis (While tst c)  $\longleftrightarrow$  SC-Sbis (While tst c)
| SC-ZObis (Par cs)      $\longleftrightarrow$  SC-Sbis (Par cs)
| SC-ZObis (ParT cs)    $\longleftrightarrow$  ( $\forall c \in \text{set } cs. \text{SC-Sbis } c$ )

```

theorem *SC-Sbis-SC-ZObis*[*intro*]: *SC-Sbis c* \implies *SC-ZObis c*
<proof>

declare *SC-Sbis.simps*[*simp del*]

theorem *SC-ZObis-ZObis*: *proper c* \implies *SC-ZObis c* \implies *c* ≈ 01 *c*
<proof>

end

end

6 Concrete setting

theory *Concrete*
imports *Syntactic-Criteria*
begin

datatype *level* = *Lo* | *Hi*

lemma [*simp*]: $\bigwedge l. l \neq \text{Hi} \longleftrightarrow l = \text{Lo}$ **and**
 [*simp*]: $\bigwedge l. \text{Hi} \neq l \longleftrightarrow \text{Lo} = l$ **and**
 [*simp*]: $\bigwedge l. l \neq \text{Lo} \longleftrightarrow l = \text{Hi}$ **and**
 [*simp*]: $\bigwedge l. \text{Lo} \neq l \longleftrightarrow \text{Hi} = l$
<proof>

lemma [*dest*]: $\bigwedge l A. \llbracket l \in A; \text{Lo} \notin A \rrbracket \implies l = \text{Hi}$ **and**
 [*dest*]: $\bigwedge l A. \llbracket l \in A; \text{Hi} \notin A \rrbracket \implies l = \text{Lo}$
<proof>

declare *level.split*[*split*]

instantiation *level* :: *complete-lattice*
begin

definition *top-level*: *top* \equiv *Hi*

definition *bot-level*: $bot \equiv Lo$
definition *inf-level*: $inf\ l1\ l2 \equiv if\ Lo \in \{l1, l2\}\ then\ Lo\ else\ Hi$
definition *sup-level*: $sup\ l1\ l2 \equiv if\ Hi \in \{l1, l2\}\ then\ Hi\ else\ Lo$
definition *less-eq-level*: $less-eq\ l1\ l2 \equiv (l1 = Lo \vee l2 = Hi)$
definition *less-level*: $less\ l1\ l2 \equiv l1 = Lo \wedge l2 = Hi$
definition *Inf-level*: $Inf\ L \equiv if\ Lo \in L\ then\ Lo\ else\ Hi$
definition *Sup-level*: $Sup\ L \equiv if\ Hi \in L\ then\ Hi\ else\ Lo$

instance

<proof>

end

lemma *sup-eq-Lo[simp]*: $sup\ a\ b = Lo \longleftrightarrow a = Lo \wedge b = Lo$

<proof>

datatype *var* = $h \mid h' \mid l \mid l'$

datatype *exp* = $Ct\ nat \mid Var\ var \mid Plus\ exp\ exp \mid Minus\ exp\ exp$

datatype *test* = $Tr \mid Eq\ exp\ exp \mid Gt\ exp\ exp \mid Non\ test$

datatype *atom* = $Assign\ var\ exp$

type-synonym *choice* = $real + test$

type-synonym *state* = $var \Rightarrow nat$

syntax

-assign :: $'a \Rightarrow 'a \Rightarrow 'a\ (- ::= - [1000, 61]\ 61)$

translations

$x ::= expr == CONST\ Atm\ (CONST\ Assign\ x\ expr)$

primrec *sec* **where**

sec $h = Hi$

| *sec* $h' = Hi$

| *sec* $l = Lo$

| *sec* $l' = Lo$

fun *eval* **where**

eval $(Ct\ n)\ s = n$

| *eval* $(Var\ x)\ s = s\ x$

| *eval* $(Plus\ e1\ e2)\ s = eval\ e1\ s + eval\ e2\ s$

| *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* $(Non\ e)\ s = (\neg\ tval\ e\ s)$

fun *aval* **where**

aval $(Assign\ x\ e)\ s = (s\ (x := eval\ e\ s))$

fun *cval* **where**

$cval (Inl p) s = \min 1 (\max 0 p)$
 $| cval (Inr tst) s = (if tval tst s then 1 else 0)$

definition $indis :: (state * state) \text{ setwhere}$
 $indis \equiv \{(s,t). ALL x. sec x = Lo \longrightarrow s x = t x\}$

interpretation *Example-PL: PL-Indis aval tval cval indis*
 $\langle proof \rangle$

fun $exprSec$ **where**
 $exprSec (Ct n) = Lo$
 $| exprSec (Var x) = sec x$
 $| exprSec (Plus e1 e2) = sup (exprSec e1) (exprSec e2)$
 $| exprSec (Minus e1 e2) = sup (exprSec e1) (exprSec e2)$

fun $tstSec$ **where**
 $tstSec Tr = Lo$
 $| tstSec (Eq e1 e2) = sup (exprSec e1) (exprSec e2)$
 $| tstSec (Gt e1 e2) = sup (exprSec e1) (exprSec e2)$
 $| tstSec (Non e) = tstSec e$

lemma $exprSec\text{-}Lo\text{-}eval\text{-}eq$: $exprSec\ expr = Lo \implies (s, t) \in indis \implies eval\ expr\ s = eval\ expr\ t$
 $\langle proof \rangle$

lemma $compatAtmSyntactic[simp]$: $exprSec\ expr = Lo \vee sec\ v = Hi \implies Example\text{-}PL.compatAtm (Assign\ v\ expr)$
 $\langle proof \rangle$

lemma $presAtmSyntactic[simp]$: $sec\ v = Hi \implies Example\text{-}PL.presAtm (Assign\ v\ expr)$
 $\langle proof \rangle$

lemma $compatTstSyntactic[simp]$: $tstSec\ tst = Lo \implies Example\text{-}PL.compatTst\ tst$
 $\langle proof \rangle$

lemma $compatPrchSyntactic[simp]$: $Example\text{-}PL.compatCh (Inl\ p)$
 $\langle proof \rangle$

lemma $compatIfchSyntactic[simp]$: $Example\text{-}PL.compatCh (Inr\ tst) \longleftrightarrow Example\text{-}PL.compatTst\ tst$
 $\langle proof \rangle$

abbreviation $Ch\text{-}half$ ($Ch_{\frac{1}{2}}$) **where** $Ch_{\frac{1}{2}} \equiv Ch (Inl (1/2))$
abbreviation If **where** $If\ tst \equiv Ch (Inr\ tst)$

abbreviation $siso\ c \equiv Example-PL.siso\ c$
abbreviation $discr\ c \equiv Example-PL.discr\ c$
abbreviation $Sbis-abbrev\ (\mathbf{infix}\ \approx_s\ 55)\ \mathbf{where}\ c1\ \approx_s\ c2 \equiv (c1, c2) \in Example-PL.Sbis$
abbreviation $ZObis-abbrev\ (\mathbf{infix}\ \approx_{01}\ 55)\ \mathbf{where}\ c1\ \approx_{01}\ c2 \equiv (c1, c2) \in Example-PL.ZObis$

abbreviation $SC-siso\ c \equiv Example-PL.SC-siso\ c$
abbreviation $SC-discr\ c \equiv Example-PL.SC-discr\ c$
abbreviation $SC-Sbis\ c \equiv Example-PL.SC-Sbis\ c$
abbreviation $SC-ZObis\ c \equiv Example-PL.SC-ZObis\ c$

lemma $SC-discr\ (h ::= Ct\ 0)$
 $\langle proof \rangle$

6.1 The secure programs from the paper's Example 3

definition $[simp]: d0 =$
 $h' ::= Ct\ 0\ ;;$
 $While\ (Gt\ (Var\ h)\ (Ct\ 0))$
 $(Ch_{\frac{1}{2}}\ (h ::= Ct\ 0)$
 $(h' ::= Plus\ (Var\ h')\ (Ct\ 1)))$

definition $[simp]: d1 =$
 $While\ (Gt\ (Var\ h)\ (Ct\ 0))$
 $(Ch_{\frac{1}{2}}\ (h ::= Minus\ (Var\ h)\ (Ct\ 1))$
 $(h ::= Plus\ (Var\ h)\ (Ct\ 1)))$

definition $[simp]: d2 =$
 $If\ (Eq\ (Var\ l)\ (Ct\ 0))$
 $(l' ::= Ct\ 1)$
 $d0$

definition $[simp]: d3 =$
 $h ::= Ct\ 5\ ;;$
 $ParT\ [d0, (l ::= Ct\ 1)]$

theorem $SC-discr\ d0$
 $SC-discr\ d1$
 $SC-Sbis\ d2$
 $SC-ZObis\ d2$
 $\langle proof \rangle$

theorem $discr\ d0$
 $discr\ d1$
 $d2 \approx_s\ d2$
 $d3 \approx_{01}\ d3$
 $\langle proof \rangle$

end

References

- [1] A. Popescu, J. Hölzl, and T. Nipkow. Formalizing probabilistic noninterference. In G. Gonthier and M. Norrish, editors, *Certified Programs and Proofs (CPP 2013)*, volume 8307 of *LNCS*, pages 259–275. Springer, 2013.
- [2] A. Popescu, J. Hölzl, and T. Nipkow. Noninterfering schedulers. In R. Heckel and S. Milius, editors, *Algebra and Coalgebra in Computer Science (CALCO 2013)*, volume 8089 of *LNCS*, pages 236–252. Springer, 2013.