

An Isabelle Correctness Proof for the Volpano/Smith Security Typing System

Gregor Snelting and Daniel Wasserrab
IPD Snelting
Universität Karlsruhe (TH)

December 12, 2009

Abstract

The Volpano/Smith/Irvine security type systems [2] requires that variables are annotated as high (secret) or low (public), and provides typing rules which guarantee that secret values cannot leak to public output ports. This property of a program is called confidentiality.

For a simple while-language without threads, our proof shows that typeability in the Volpano/Smith system guarantees noninterference. Noninterference means that if two initial states for program execution are low-equivalent, then the final states are low-equivalent as well. This indeed implies that secret values cannot leak to public ports. For more details on noninterference and security typing systems, see [1].

The proof defines an abstract syntax and operational semantics for programs, formalizes noninterference, and then proceeds by rule induction on the operational semantics. The mathematically most intricate part is the treatment of implicit flows. Note that the Volpano/Smith system is not flow-sensitive and thus quite unprecise, resulting in false alarms. However, due to the correctness property, all potential breaks of confidentiality are discovered.

Contents

1	The Language	3
1.1	Variables and Values	3
1.2	Expressions and Commands	3
1.3	State	4
1.4	Small Step Semantics	4
2	Security types	10
2.1	Security definitions	10
2.2	Lemmas concerning expressions	11
2.3	Noninterference definitions	15
2.3.1	Low Equivalence	15
2.3.2	Non Interference	15
3	Executing the small step semantics	25
3.1	An example taken from Volpano, Smith, Irvine	27

```

theory Semantics
imports Main
begin

```

1 The Language

1.1 Variables and Values

types *vname* = *string* — names for variables

```

datatype val
  = Bool bool      — Boolean value
  | Intg int       — integer value

```

```

abbreviation true == Bool True
abbreviation false == Bool False

```

1.2 Expressions and Commands

datatype *bop* = *Eq* | *And* | *Less* | *Add* | *Sub* — names of binary operations

```

datatype expr
  = Val val                — value
  | Var vname              — local variable
  | BinOp expr bop expr    (- «->» - [80,0,81] 80) — binary operation

```

Note: we assume that only type correct expressions are regarded as later proofs fail if expressions evaluate to *None* due to type errors. However there is [yet] no typing system

```

fun binop :: bop ⇒ val ⇒ val ⇒ val option
where binop Eq v1 v2          = Some(Bool(v1 = v2))
      | binop And (Bool b1) (Bool b2) = Some(Bool(b1 ∧ b2))
      | binop Less (Intg i1) (Intg i2) = Some(Bool(i1 < i2))
      | binop Add (Intg i1) (Intg i2) = Some(Intg(i1 + i2))
      | binop Sub (Intg i1) (Intg i2) = Some(Intg(i1 - i2))

      | binop bop v1 v2          = Some(Intg(0))

```

```

datatype com
  = Skip
  | LAss vname expr        (-:=- [70,70] 70) — local assignment
  | Seq com com           (-:;/ - [61,60] 60)
  | Cond expr com com    (if '(-) -/ else - [80,79,79] 70)
  | While expr com       (while '(-) - [80,79] 70)

```

```

fun fv :: expr ⇒ vname set — free variables in an expression
where

```

$FVc: fv (Val V) = \{\}$
 $| FVv: fv (Var V) = \{V\}$
 $| FVe: fv (e1 \ll bop \gg e2) = fv e1 \cup fv e2$

1.3 State

types

$state = vname \rightarrow val$

interpret silently assumes type correct expressions, i.e. no expression evaluates to None

fun *interpret* :: $expr \Rightarrow state \Rightarrow val option$ ($\llbracket - \rrbracket$)

where *Val*: $\llbracket Val v \rrbracket s = Some v$

$|$ *Var*: $\llbracket Var V \rrbracket s = s V$

$|$ *BinOp*: $\llbracket e1 \ll bop \gg e2 \rrbracket s = (case \llbracket e1 \rrbracket s of None \Rightarrow None$

$| Some v_1 \Rightarrow (case \llbracket e2 \rrbracket s of None \Rightarrow None$

$| Some v_2 \Rightarrow binop bop v_1 v_2))$

1.4 Small Step Semantics

inductive *red* :: $com * state \Rightarrow com * state \Rightarrow bool$

and *red'* :: $com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool$

$((\llbracket (1 \langle -, / - \rangle) \rrbracket \rightarrow / (1 \langle -, / - \rangle)) [0,0,0,0] 81)$

where

$\langle c1, s1 \rangle \rightarrow \langle c2, s2 \rangle == red (c1, s1) (c2, s2) |$

RedLAss:

$\langle V := e, s \rangle \rightarrow \langle Skip, s (V := (\llbracket e \rrbracket s)) \rangle$

$|$ *SeqRed*:

$\langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \Longrightarrow \langle c_1;; c_2, s \rangle \rightarrow \langle c_1';; c_2, s' \rangle$

$|$ *RedSeq*:

$\langle Skip;; c_2, s \rangle \rightarrow \langle c_2, s \rangle$

$|$ *RedCondTrue*:

$\llbracket b \rrbracket s = Some true \Longrightarrow \langle if (b) c_1 else c_2, s \rangle \rightarrow \langle c_1, s \rangle$

$|$ *RedCondFalse*:

$\llbracket b \rrbracket s = Some false \Longrightarrow \langle if (b) c_1 else c_2, s \rangle \rightarrow \langle c_2, s \rangle$

$|$ *RedWhileTrue*:

$\llbracket b \rrbracket s = Some true \Longrightarrow \langle while (b) c, s \rangle \rightarrow \langle c;; while (b) c, s \rangle$

$|$ *RedWhileFalse*:

$\llbracket b \rrbracket s = Some false \Longrightarrow \langle while (b) c, s \rangle \rightarrow \langle Skip, s \rangle$

lemmas *red-induct* = *red.induct*[*split-format* (*complete*)]

abbreviation *reds* :: $com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool$

$((\llbracket (1 \langle -, / - \rangle) \rrbracket \rightarrow * / (1 \langle -, / - \rangle)) [0,0,0,0] 81)$ **where**

$$\langle c, s \rangle \rightarrow^* \langle c', s' \rangle == \text{red}^{**} (c, s) (c', s')$$

lemma *Skip-reds*:

$\langle \text{Skip}, s \rangle \rightarrow^* \langle c', s' \rangle \implies s = s' \wedge c' = \text{Skip}$
by (*blast elim: converse-rtranclpE red.cases*)

lemma *LAss-reds*:

$\langle V := e, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \implies s' = s(V := \llbracket e \rrbracket s)$
proof (*induct c == V := e s rule: converse-rtranclp-induct2*)
case refl thus ?case by simp
next
case (*step c s c'' s''*)
hence $c'' = \text{Skip}$ **and** $s'' = s(V := (\llbracket e \rrbracket s))$ **by** (*auto elim:red.cases*)
with $\langle c'', s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$ **show** ?case **by** (*auto dest:Skip-reds*)
qed

lemma *Seq2-reds*:

$\langle \text{Skip};; c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \implies \langle c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$
by (*induct c == Skip;; c_2 s rule: converse-rtranclp-induct2*) (*auto elim:red.cases*)

lemma *Seq-reds*:

assumes $\langle c_1;; c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$
obtains s'' **where** $\langle c_1, s \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle$ **and** $\langle c_2, s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$
proof –
have $\exists s''. \langle c_1, s \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle \wedge \langle c_2, s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$
proof –
{ fix c c'
assume $\langle c, s \rangle \rightarrow^* \langle c', s' \rangle$ **and** $c = c_1;; c_2$ **and** $c' = \text{Skip}$
hence $\exists s''. \langle c_1, s \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle \wedge \langle c_2, s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$
proof (*induct arbitrary: c_1 rule: converse-rtranclp-induct2*)
case refl thus ?case by simp
next
case (*step c s c'' s''*)
note $IH = \langle \bigwedge e_1. \llbracket c'' = c_1;; c_2; c' = \text{Skip} \rrbracket$
 $\implies \exists sx. \langle c_1, s'' \rangle \rightarrow^* \langle \text{Skip}, sx \rangle \wedge \langle c_2, sx \rangle \rightarrow^* \langle \text{Skip}, s' \rangle$
from step
have $\langle c_1;; c_2, s \rangle \rightarrow \langle c'', s'' \rangle$ **by simp**
hence $(c_1 = \text{Skip} \wedge c'' = c_2 \wedge s = s'') \vee$
 $(\exists c_1'. \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \wedge c'' = c_1';; c_2)$
by (*auto elim:red.cases*)
thus ?case
proof
assume $c_1 = \text{Skip} \wedge c'' = c_2 \wedge s = s''$
with $\langle c'', s'' \rangle \rightarrow^* \langle c', s' \rangle$ $\langle c' = \text{Skip} \rangle$
show ?thesis **by auto**
next
assume $\exists c_1'. \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \wedge c'' = c_1';; c_2$
then obtain c_1' **where** $\langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle$ **and** $c'' = c_1';; c_2$ **by blast**
from $IH[OF \langle c'' = c_1';; c_2 \rangle \langle c' = \text{Skip} \rangle]$

```

obtain  $sx$  where  $\langle c_1', s' \rangle \rightarrow^* \langle Skip, sx \rangle$  and  $\langle c_2, sx \rangle \rightarrow^* \langle Skip, s \wedge \rangle$ 
  by blast
from  $\langle \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \rangle \langle \langle c_1', s' \rangle \rightarrow^* \langle Skip, sx \rangle \rangle$ 
have  $\langle c_1, s \rangle \rightarrow^* \langle Skip, sx \rangle$  by (auto intro: converse-rtranclp-into-rtranclp)
with  $\langle \langle c_2, sx \rangle \rightarrow^* \langle Skip, s \wedge \rangle \rangle$  show ?thesis by auto
qed
qed }
with  $\langle \langle c_1;; c_2, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \rangle$  show ?thesis by simp
qed
with that show ?thesis by blast
qed

```

lemma *Cond-True-or-False*:

```

 $\langle \text{if } (b) \ c_1 \ \text{else } c_2, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \implies \llbracket b \rrbracket s = \text{Some true} \vee \llbracket b \rrbracket s = \text{Some false}$ 
by (induct c==if (b) c1 else c2 s rule: converse-rtranclp-induct2) (auto elim: red.cases)

```

lemma *CondTrue-reds*:

```

 $\langle \text{if } (b) \ c_1 \ \text{else } c_2, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \implies \llbracket b \rrbracket s = \text{Some true} \implies \langle c_1, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle$ 
by (induct c==if (b) c1 else c2 s rule: converse-rtranclp-induct2) (auto elim: red.cases)

```

lemma *CondFalse-reds*:

```

 $\langle \text{if } (b) \ c_1 \ \text{else } c_2, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \implies \llbracket b \rrbracket s = \text{Some false} \implies \langle c_2, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle$ 
by (induct c==if (b) c1 else c2 s rule: converse-rtranclp-induct2) (auto elim: red.cases)

```

lemma *WhileFalse-reds*:

```

 $\langle \text{while } (b) \ cx, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \implies \llbracket b \rrbracket s = \text{Some false} \implies s = s'$ 
proof (induct c==while (b) cx s rule: converse-rtranclp-induct2)
  case refl thus ?case by simp
next
  case step thus ?case by (auto elim: red.cases dest: Skip-reds)
qed

```

lemma *WhileTrue-reds*:

```

 $\langle \text{while } (b) \ cx, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \implies \llbracket b \rrbracket s = \text{Some true}$ 
 $\implies \exists sx. \langle cx, s \rangle \rightarrow^* \langle Skip, sx \rangle \wedge \langle \text{while } (b) \ cx, sx \rangle \rightarrow^* \langle Skip, s \wedge \rangle$ 
proof (induct c==while (b) cx s rule: converse-rtranclp-induct2)
  case refl thus ?case by simp
next
  case (step c s c'' s'')
  hence  $c'' = cx;; \text{while } (b) \ cx \wedge s'' = s$  by (auto elim: red.cases)
  with  $\langle \langle c'', s'' \rangle \rightarrow^* \langle Skip, s \wedge \rangle \rangle$  show ?case by (auto dest: Seq-reds)
qed

```

lemma *While-True-or-False*:

```

 $\langle \text{while } (b) \ com, s \rangle \rightarrow^* \langle Skip, s \wedge \rangle \implies \llbracket b \rrbracket s = \text{Some true} \vee \llbracket b \rrbracket s = \text{Some false}$ 
by (induct c==while (b) com s rule: converse-rtranclp-induct2) (auto elim: red.cases)

```

inductive *red-n* :: *com* \Rightarrow *state* \Rightarrow *nat* \Rightarrow *com* \Rightarrow *state* \Rightarrow *bool*
 (((1⟨-,/-⟩) \rightarrow^* (1⟨-,/-⟩)) [0,0,0,0,0] 81)
where *red-n-Base*: $\langle c,s \rangle \rightarrow^0 \langle c,s \rangle$
 | *red-n-Rec*: $\llbracket \langle c,s \rangle \rightarrow \langle c'',s'' \rangle; \langle c'',s'' \rangle \rightarrow^n \langle c',s' \rangle \rrbracket \Longrightarrow \langle c,s \rangle \rightarrow^{Suc\ n} \langle c',s' \rangle$

lemma *Seq-red-nE*: **assumes** $\langle c_1;;c_2,s \rangle \rightarrow^n \langle Skip,s' \rangle$
obtains $i\ j\ s''$ **where** $\langle c_1,s \rangle \rightarrow^i \langle Skip,s'' \rangle$ **and** $\langle c_2,s'' \rangle \rightarrow^j \langle Skip,s' \rangle$
and $n = i + j + 1$

proof –

from $\langle \langle c_1;;c_2,s \rangle \rightarrow^n \langle Skip,s' \rangle \rangle$

have $\exists i\ j\ s''. \langle c_1,s \rangle \rightarrow^i \langle Skip,s'' \rangle \wedge \langle c_2,s'' \rangle \rightarrow^j \langle Skip,s' \rangle \wedge n = i + j + 1$

proof(*induct* $c \equiv c_1;;c_2\ s\ n\ c' \equiv Skip\ s'$ *arbitrary:c₁ rule:red-n.induct*)

case (*red-n-Base* $c\ s$)

from $\langle c = c_1;;c_2 \rangle \langle c = Skip \rangle$ **have** *False* **by** *simp*

thus *?case* **by** *simp*

next

case (*red-n-Rec* $c\ s\ c''\ s''\ n\ c'\ s'$)

note $IH = \langle \bigwedge c_1. \llbracket c'' = c_1;;c_2; c' = Skip \rrbracket$

$\Longrightarrow \exists i\ j\ sx. \langle c_1,s'' \rangle \rightarrow^i \langle Skip,sx \rangle \wedge \langle c_2,sx \rangle \rightarrow^j \langle Skip,s' \rangle \wedge n = i + j + 1$

from $\langle \langle c,s \rangle \rightarrow \langle c'',s'' \rangle \rangle \langle c = c_1;;c_2 \rangle$

have $(c_1 = Skip \wedge c'' = c_2 \wedge s = s'') \vee$

$(\exists c_1'. c'' = c_1';c_2 \wedge \langle c_1,s \rangle \rightarrow \langle c_1',s'' \rangle)$

by(*induct rule:red-induct*) *auto*

thus *?case*

proof

assume $c_1 = Skip \wedge c'' = c_2 \wedge s = s''$

hence $c_1 = Skip$ **and** $c'' = c_2$ **and** $s = s''$ **by** *simp-all*

from $\langle c_1 = Skip \rangle$ **have** $\langle c_1,s \rangle \rightarrow^0 \langle Skip,s \rangle$ **by**(*fastsimp intro:red-n-Base*)

with $\langle \langle c'',s'' \rangle \rightarrow^n \langle c',s' \rangle \rangle \langle c'' = c_2 \rangle \langle c' = Skip \rangle \langle s = s'' \rangle$

show *?thesis* **by**(*rule-tac x=0 in exI*) *auto*

next

assume $\exists c_1'. c'' = c_1';c_2 \wedge \langle c_1,s \rangle \rightarrow \langle c_1',s'' \rangle$

then obtain c_1' **where** $c'' = c_1';c_2$ **and** $\langle c_1,s \rangle \rightarrow \langle c_1',s'' \rangle$ **by** *blast*

from $IH[OF\ c'' = c_1';c_2\ \langle c' = Skip \rangle]$ **obtain** $i\ j\ sx$

where $\langle c_1',s'' \rangle \rightarrow^i \langle Skip,sx \rangle$ **and** $\langle c_2,sx \rangle \rightarrow^j \langle Skip,s' \rangle$

and $n = i + j + 1$ **by** *blast*

from $\langle \langle c_1,s \rangle \rightarrow \langle c_1',s'' \rangle \rangle \langle \langle c_1',s'' \rangle \rightarrow^i \langle Skip,sx \rangle \rangle$

have $\langle c_1,s \rangle \rightarrow^{Suc\ i} \langle Skip,sx \rangle$ **by**(*rule red-n.red-n-Rec*)

with $\langle \langle c_2,sx \rangle \rightarrow^j \langle Skip,s' \rangle \rangle \langle n = i + j + 1 \rangle$ **show** *?thesis*

by(*rule-tac x=Suc i in exI*) *auto*

qed

qed

with that show *?thesis* **by** *blast*

qed

lemma *while-red-nE*:

$\langle while\ (b)\ cx,s \rangle \rightarrow^n \langle Skip,s' \rangle$

$\implies (\llbracket b \rrbracket s = \text{Some false} \wedge s = s' \wedge n = 1) \vee$
 $(\exists i j s''. \llbracket b \rrbracket s = \text{Some true} \wedge \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle \wedge$
 $\langle \text{while } (b) \text{ } cx, s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \wedge n = i + j + 2)$
proof(*induct* $c \equiv \text{while } (b) \text{ } cx \text{ } s \text{ } n \text{ } c' \equiv \text{Skip } s' \text{ arbitrary:}c \text{ rule:red-n.induct}$)
case (*red-n-Base* $c \text{ } s$)
from $\langle c = \text{while } (b) \text{ } cx \rangle \langle c = \text{Skip} \rangle$ **have** *False* **by** *simp*
thus *?case* **by** *simp*
next
case (*red-n-Rec* $c \text{ } s \text{ } c'' \text{ } s'' \text{ } n \text{ } c' \text{ } s'$)
from $\langle \langle c, s \rangle \rightarrow \langle c'', s'' \rangle \rangle \langle c = \text{while } (b) \text{ } cx \rangle$
have $(\llbracket b \rrbracket s = \text{Some false} \wedge c'' = \text{Skip} \wedge s'' = s) \vee$
 $(\llbracket b \rrbracket s = \text{Some true} \wedge c'' = cx;;\text{while } (b) \text{ } cx \wedge s'' = s)$
by(*induct* *rule:red-induct*) *auto*
thus *?case*
proof
assume $\llbracket b \rrbracket s = \text{Some false} \wedge c'' = \text{Skip} \wedge s'' = s$
hence $\llbracket b \rrbracket s = \text{Some false}$ **and** $c'' = \text{Skip}$ **and** $s'' = s$ **by** *simp-all*
with $\langle \langle c'', s'' \rangle \rightarrow^n \langle c', s' \rangle \rangle$ **have** $s = s'$ **and** $n = 0$
by(*induct* *rule:red-n.induct*, *auto* *elim:red.cases*)
with $\langle \llbracket b \rrbracket s = \text{Some false} \rangle$ **show** *?thesis* **by** *fastsimp*
next
assume $\llbracket b \rrbracket s = \text{Some true} \wedge c'' = cx;;\text{while } (b) \text{ } cx \wedge s'' = s$
hence $\llbracket b \rrbracket s = \text{Some true}$ **and** $c'' = cx;;\text{while } (b) \text{ } cx$
and $s'' = s$ **by** *simp-all*
with $\langle \langle c'', s'' \rangle \rightarrow^n \langle c', s' \rangle \rangle \langle c' = \text{Skip} \rangle$
obtain $i j sx$ **where** $\langle cx, s \rangle \rightarrow^i \langle \text{Skip}, sx \rangle$ **and** $\langle \text{while } (b) \text{ } cx, sx \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$
and $n = i + j + 1$ **by**(*fastsimp* *elim:Seq-red-nE*)
with $\langle \llbracket b \rrbracket s = \text{Some true} \rangle$ **show** *?thesis* **by** *fastsimp*
qed
qed

lemma *while-red-n-induct* [*consumes 1, case-names false true*]:

assumes *major*: $\langle \text{while } (b) \text{ } cx, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$
and *IHfalse*: $\bigwedge s. \llbracket b \rrbracket s = \text{Some false} \implies P \text{ } s \text{ } s$
and *IHtrue*: $\bigwedge s i j s''. \llbracket b \rrbracket s = \text{Some true}; \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle;$
 $\langle \text{while } (b) \text{ } cx, s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle; P \text{ } s'' \text{ } s'' \implies P \text{ } s \text{ } s'$

shows $P \text{ } s \text{ } s'$

using *major*

proof(*induct* $n \text{ arbitrary:}s \text{ rule:nat-less-induct}$)

fix $n \text{ } s$

assume *IHall*: $\forall m < n. \forall x. \langle \text{while } (b) \text{ } cx, x \rangle \rightarrow^m \langle \text{Skip}, s' \rangle \longrightarrow P \text{ } x \text{ } s'$

and $\langle \text{while } (b) \text{ } cx, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$

from $\langle \langle \text{while } (b) \text{ } cx, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle \rangle$

have $(\llbracket b \rrbracket s = \text{Some false} \wedge s = s' \wedge n = 1) \vee$

$(\exists i j s''. \llbracket b \rrbracket s = \text{Some true} \wedge \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \wedge$

$\langle \text{while } (b) \text{ } cx, s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \wedge n = i + j + 2)$

by(*rule* *while-red-nE*)

thus $P \text{ } s \text{ } s'$

proof

assume $\llbracket b \rrbracket s = \text{Some false} \wedge s = s' \wedge n = 1$
hence $\llbracket b \rrbracket s = \text{Some false}$ **and** $s = s'$ **by** *auto*
from $\text{IHfalse}[OF \langle \llbracket b \rrbracket s = \text{Some false} \rangle]$ **have** $P s s'$.
with $\langle s = s' \rangle$ **show** *?thesis* **by** *simp*

next

assume $\exists i j s'' . \llbracket b \rrbracket s = \text{Some true} \wedge \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle \wedge$
 $\langle \text{while } (b) \text{ } cx, s' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \wedge n = i + j + 2$
then obtain $i j s''$ **where** $\llbracket b \rrbracket s = \text{Some true}$
and $\langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle$ **and** $\langle \text{while } (b) \text{ } cx, s' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$
and $n = i + j + 2$ **by** *blast*
with IHall **have** $P s'' s'$
apply(*erule-tac x=j in allE*) **apply** *clarsimp* **done**
from $\text{IHtrue}[OF \langle \llbracket b \rrbracket s = \text{Some true} \rangle \langle \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle \rangle$
 $\langle \langle \text{while } (b) \text{ } cx, s' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \rangle$ **this]** **show** *?thesis* .

qed

qed

lemma *reds-to-red-n*: $\langle c, s \rangle \rightarrow^* \langle c', s' \rangle \implies \exists n . \langle c, s \rangle \rightarrow^n \langle c', s' \rangle$
by(*induct rule:converse-rtranclp-induct2, auto intro:red-n.intros*)

lemma *red-n-to-reds*: $\langle c, s \rangle \rightarrow^n \langle c', s' \rangle \implies \langle c, s \rangle \rightarrow^* \langle c', s' \rangle$
by(*induct rule:red-n.induct, auto intro:converse-rtranclp-into-rtranclp*)

lemma *while-reds-induct*[*consumes 1, case-names false true*]:
 $\llbracket \langle \text{while } (b) \text{ } cx, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle; \wedge s . \llbracket b \rrbracket s = \text{Some false} \implies P s s' ;$
 $\wedge s s'' . \llbracket \llbracket b \rrbracket s = \text{Some true}; \langle cx, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle;$
 $\langle \text{while } (b) \text{ } cx, s' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle; P s'' s' \rrbracket \implies P s s' \rrbracket$
 $\implies P s s'$
apply(*drule reds-to-red-n,clarsimp*)
apply(*erule while-red-n-induct,clarsimp*)
by(*auto dest:red-n-to-reds*)

lemma *red-det*:

$\llbracket \langle c, s \rangle \rightarrow \langle c_1, s_1 \rangle; \langle c, s \rangle \rightarrow \langle c_2, s_2 \rangle \rrbracket \implies c_1 = c_2 \wedge s_1 = s_2$

proof(*induct arbitrary:c2 rule:red-induct*)

case (*SeqRed* $c_1 s c_1' s' c_2'$)

note $\text{IH} = \langle \wedge c_2 . \langle c_1, s \rangle \rightarrow \langle c_2, s_2 \rangle \implies c_1' = c_2 \wedge s' = s_2 \rangle$

from $\langle \langle c_1; c_2', s \rangle \rightarrow \langle c_2, s_2 \rangle \rangle$ **have** $c_1 = \text{Skip} \vee (\exists cx . c_2 = cx; c_2' \wedge \langle c_1, s \rangle \rightarrow$
 $\langle cx, s_2 \rangle)$

by(*fastsimp elim:red.cases*)

thus *?case*

proof

assume $c_1 = \text{Skip}$

with $\langle \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \rangle$ **have** *False* **by**(*fastsimp elim:red.cases*)

```

    thus ?thesis by simp
  next
    assume  $\exists cx. c_2 = cx;;c_2' \wedge \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle$ 
    then obtain  $cx$  where  $c_2 = cx;;c_2'$  and  $\langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle$  by blast
    from IH[OF  $\langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle$ ] have  $c_1' = cx \wedge s' = s_2$  .
    with  $\langle c_2 = cx;;c_2' \rangle$  show ?thesis by simp
  qed
qed (fastsimp elim:red.cases)+

```

theorem *reds-det*:

```

[[ $\langle c, s \rangle \rightarrow^* \langle Skip, s_1 \rangle; \langle c, s \rangle \rightarrow^* \langle Skip, s_2 \rangle$ ]]  $\implies s_1 = s_2$ 
proof(induct rule:converse-rtranclp-induct2)
  case refl
    from  $\langle \langle Skip, s_1 \rangle \rightarrow^* \langle Skip, s_2 \rangle \rangle$  show ?case
    by -(erule converse-rtranclpE, auto elim:red.cases)
  next
    case (step  $c'' s'' c' s'$ )
    note IH =  $\langle \langle c', s' \rangle \rightarrow^* \langle Skip, s_2 \rangle \implies s_1 = s_2 \rangle$ 
    from step have  $\langle c'', s'' \rangle \rightarrow \langle c', s' \rangle$ 
    by simp
    from  $\langle \langle c'', s'' \rangle \rightarrow^* \langle Skip, s_2 \rangle \rangle$  this have  $\langle c', s' \rangle \rightarrow^* \langle Skip, s_2 \rangle$ 
    by -(erule converse-rtranclpE, auto elim:red.cases dest:red-det)
    from IH[OF this] show ?thesis .
  qed

```

end

```

theory secTypes
imports Semantics
begin

```

2 Security types

2.1 Security definitions

```

datatype secLevel = Low | High

```

```

types typeEnv = vname  $\rightarrow$  secLevel

```

```

inductive secExprTyping :: typeEnv  $\Rightarrow$  expr  $\Rightarrow$  secLevel  $\Rightarrow$  bool (-  $\vdash$  - : -)
where typeVal:  $\Gamma \vdash \text{Val } V : lev$ 

```

```

| typeVar:  $\Gamma \text{ Vn} = \text{Some lev} \implies \Gamma \vdash \text{Var Vn} : lev$ 

```

```

| typeBinOp1:  $[[\Gamma \vdash e1 : Low; \Gamma \vdash e2 : Low]] \implies \Gamma \vdash e1 \llbop\rangle e2 : Low$ 

```

```

| typeBinOp2:  $[[\Gamma \vdash e1 : High; \Gamma \vdash e2 : lev]] \implies \Gamma \vdash e1 \llbop\rangle e2 : High$ 

```

| *typeBinOp3*: $[\Gamma \vdash e1 : \text{lev}; \Gamma \vdash e2 : \text{High}] \implies \Gamma \vdash e1 \ll\text{bop}\gg e2 : \text{High}$

inductive *secComTyping* :: *typeEnv* \Rightarrow *secLevel* \Rightarrow *com* \Rightarrow *bool* (-,- \vdash -)
where *typeSkip*: $\Gamma, T \vdash \text{Skip}$

| *typeAssH*: $\Gamma V = \text{Some High} \implies \Gamma, T \vdash V := e$

| *typeAssL*: $[\Gamma \vdash e : \text{Low}; \Gamma V = \text{Some Low}] \implies \Gamma, \text{Low} \vdash V := e$

| *typeSeq*: $[\Gamma, T \vdash c1; \Gamma, T \vdash c2] \implies \Gamma, T \vdash c1;;c2$

| *typeWhile*: $[\Gamma \vdash b : T; \Gamma, T \vdash c] \implies \Gamma, T \vdash \text{while } (b) \ c$

| *typeIf*: $[\Gamma \vdash b : T; \Gamma, T \vdash c1; \Gamma, T \vdash c2] \implies \Gamma, T \vdash \text{if } (b) \ c1 \ \text{else } \ c2$

| *typeConvert*: $\Gamma, \text{High} \vdash c \implies \Gamma, \text{Low} \vdash c$

2.2 Lemmas concerning expressions

lemma *exprTypeable*:

assumes $\text{fv } e \subseteq \text{dom } \Gamma$ **obtains** T **where** $\Gamma \vdash e : T$

proof –

from $\langle \text{fv } e \subseteq \text{dom } \Gamma \rangle$ **have** $\exists T. \Gamma \vdash e : T$

proof(*induct e*)

case (*Val V*)

have $\Gamma \vdash \text{Val } V : \text{Low}$ **by**(*rule typeVal*)

thus *?case* **by** (*rule exI*)

next

case (*Var V*)

have $V \in \text{fv } (V)$ **by** *simp*

with $\langle \text{fv } (V) \subseteq \text{dom } \Gamma \rangle$ **have** $V \in \text{dom } \Gamma$ **by** *simp*

then obtain T **where** $\Gamma V = \text{Some } T$ **by** *auto*

hence $\Gamma \vdash \text{Var } V : T$ **by** (*rule typeVar*)

thus *?case* **by** (*rule exI*)

next

case (*BinOp e1 bop e2*)

note $IH1 = \langle \text{fv } e1 \subseteq \text{dom } \Gamma \implies \exists T. \Gamma \vdash e1 : T \rangle$

note $IH2 = \langle \text{fv } e2 \subseteq \text{dom } \Gamma \implies \exists T. \Gamma \vdash e2 : T \rangle$

from $\langle \text{fv } (e1 \ll\text{bop}\gg e2) \subseteq \text{dom } \Gamma \rangle$

have $\text{fv } e1 \subseteq \text{dom } \Gamma$ **and** $\text{fv } e2 \subseteq \text{dom } \Gamma$ **by** *auto*

from $IH1[OF \langle \text{fv } e1 \subseteq \text{dom } \Gamma \rangle]$ **obtain** $T1$ **where** $\Gamma \vdash e1 : T1$ **by** *auto*

from $IH2[OF \langle \text{fv } e2 \subseteq \text{dom } \Gamma \rangle]$ **obtain** $T2$ **where** $\Gamma \vdash e2 : T2$ **by** *auto*

show *?case*

proof (*cases T1*)

case *Low*

show *?thesis*

```

proof (cases T2)
  case Low
    with  $\langle \Gamma \vdash e1 : T1 \rangle \langle \Gamma \vdash e2 : T2 \rangle \langle T1 = Low \rangle$ 
    have  $\Gamma \vdash e1 \ll bop \gg e2 : Low$  by (simp add:typeBinOp1)
    thus ?thesis by (rule exI)
  next
    case High
    with  $\langle \Gamma \vdash e1 : T1 \rangle \langle \Gamma \vdash e2 : T2 \rangle \langle T1 = Low \rangle$ 
    have  $\Gamma \vdash e1 \ll bop \gg e2 : High$  by (simp add:typeBinOp3)
    thus ?thesis by (rule exI)
  qed
next
  case High
  with  $\langle \Gamma \vdash e1 : T1 \rangle \langle \Gamma \vdash e2 : T2 \rangle$ 
  have  $\Gamma \vdash e1 \ll bop \gg e2 : High$  by (simp add:typeBinOp2)
  thus ?thesis by (rule exI)
  qed
qed
with that show ?thesis by blast
qed

```

```

lemma exprBinopTypeable:
  assumes  $\Gamma \vdash e1 \ll bop \gg e2 : T$ 
  shows  $(\exists T1. \Gamma \vdash e1 : T1) \wedge (\exists T2. \Gamma \vdash e2 : T2)$ 
using assms by (auto elim:secExprTyping.cases)

```

```

lemma exprTypingHigh:
  assumes  $\Gamma \vdash e : T$  and  $x \in fv\ e$  and  $\Gamma\ x = Some\ High$ 
  shows  $\Gamma \vdash e : High$ 
using assms
proof (induct e arbitrary:T)
  case (Val V) show ?case by (rule typeVal)
next
  case (Var V)
  from  $\langle x \in fv\ (Var\ V) \rangle$  have  $x = V$  by simp
  with  $\langle \Gamma\ x = Some\ High \rangle$  show ?case by (simp add:typeVar)
next
  case (BinOp e1 bop e2)
  note IH1 =  $\langle \bigwedge T. [\Gamma \vdash e1 : T; x \in fv\ e1; \Gamma\ x = Some\ High] \implies \Gamma \vdash e1 : High \rangle$ 
  note IH2 =  $\langle \bigwedge T. [\Gamma \vdash e2 : T; x \in fv\ e2; \Gamma\ x = Some\ High] \implies \Gamma \vdash e2 : High \rangle$ 
  from  $\langle \Gamma \vdash e1 \ll bop \gg e2 : T \rangle$ 
  have  $T : (\exists T1. \Gamma \vdash e1 : T1) \wedge (\exists T2. \Gamma \vdash e2 : T2)$  by (auto intro!:exprBinopTypeable)
  then obtain T1 where  $\Gamma \vdash e1 : T1$  by auto
  from T obtain T2 where  $\Gamma \vdash e2 : T2$  by auto
  from  $\langle x \in fv\ (e1 \ll bop \gg e2) \rangle$  have  $x \in (fv\ e1 \cup fv\ e2)$  by simp
  hence  $x \in fv\ e1 \vee x \in fv\ e2$  by auto

```

```

thus ?case
proof
  assume  $x \in \text{fv } e1$ 
  from  $IH1[OF \langle \Gamma \vdash e1 : T1 \rangle \text{ this } \langle \Gamma x = \text{Some High} \rangle]$  have  $\Gamma \vdash e1 : \text{High}$  .
  with  $\langle \Gamma \vdash e2 : T2 \rangle$  show ?thesis by(simp add:typeBinOp2)
next
  assume  $x \in \text{fv } e2$ 
  from  $IH2[OF \langle \Gamma \vdash e2 : T2 \rangle \text{ this } \langle \Gamma x = \text{Some High} \rangle]$  have  $\Gamma \vdash e2 : \text{High}$  .
  with  $\langle \Gamma \vdash e1 : T1 \rangle$  show ?thesis by(simp add:typeBinOp3)
qed
qed

```

lemma *exprTypingLow*:

```

assumes  $\Gamma \vdash e : \text{Low}$  and  $x \in \text{fv } e$  shows  $\Gamma x = \text{Some Low}$ 
using assms
proof (induct e)
  case (Val V)
  have  $\text{fv } (\text{Val } V) = \{\}$  by (rule FVc)
  with  $\langle x \in \text{fv } (\text{Val } V) \rangle$  have False by auto
  thus ?thesis by simp
next
  case (Var V)
  from  $\langle x \in \text{fv } (\text{Var } V) \rangle$  have  $xV: x = V$  by simp
  from  $\langle \Gamma \vdash \text{Var } V : \text{Low} \rangle$  have  $\Gamma V = \text{Some Low}$  by (auto elim:secExprTyping.cases)
  with  $xV$  show ?thesis by simp
next
  case (BinOp e1 bop e2)
  note  $IH1 = \langle [\Gamma \vdash e1 : \text{Low}; x \in \text{fv } e1] \implies \Gamma x = \text{Some Low} \rangle$ 
  note  $IH2 = \langle [\Gamma \vdash e2 : \text{Low}; x \in \text{fv } e2] \implies \Gamma x = \text{Some Low} \rangle$ 
  from  $\langle \Gamma \vdash e1 \ll bop \gg e2 : \text{Low} \rangle$  have  $\Gamma \vdash e1 : \text{Low}$  and  $\Gamma \vdash e2 : \text{Low}$ 
  by(auto elim:secExprTyping.cases)
  from  $\langle x \in \text{fv } (e1 \ll bop \gg e2) \rangle$  have  $x \in \text{fv } e1 \cup \text{fv } e2$  by (simp add:FVe)
  hence  $x \in \text{fv } e1 \vee x \in \text{fv } e2$  by auto
  thus ?case
  proof
    assume  $x \in \text{fv } e1$ 
    with  $IH1[OF \langle \Gamma \vdash e1 : \text{Low} \rangle]$  show ?thesis by auto
  next
    assume  $x \in \text{fv } e2$ 
    with  $IH2[OF \langle \Gamma \vdash e2 : \text{Low} \rangle]$  show ?thesis by auto
  qed
qed

```

lemma *typeableFreevars*:

```

assumes  $\Gamma \vdash e : T$  shows  $\text{fv } e \subseteq \text{dom } \Gamma$ 
using assms
proof(induct e arbitrary:T)

```

```

case (Val V)
have  $fv (Val V) = \{\}$  by (rule FVc)
thus ?case by simp
next
case (Var V)
show ?case
proof
  fix x assume  $x \in fv (Var V)$ 
  hence  $x = V$  by simp
  from  $\langle \Gamma \vdash Var V : T \rangle$  have  $\Gamma V = Some T$  by (auto elim:secExprTyping.cases)
  with  $\langle x = V \rangle$  show  $x \in dom \Gamma$  by auto
qed
next
case (BinOp e1 bop e2)
note IH1 =  $\langle \wedge T. \Gamma \vdash e1 : T \implies fv e1 \subseteq dom \Gamma \rangle$ 
note IH2 =  $\langle \wedge T. \Gamma \vdash e2 : T \implies fv e2 \subseteq dom \Gamma \rangle$ 
show ?case
proof
  fix x assume  $x \in fv (e1 \ll bop \gg e2)$ 
  from  $\langle \Gamma \vdash e1 \ll bop \gg e2 : T \rangle$ 
  have  $Q: (\exists T1. \Gamma \vdash e1 : T1) \wedge (\exists T2. \Gamma \vdash e2 : T2)$ 
  by (rule exprBinopTypeable)
  then obtain T1 where  $\Gamma \vdash e1 : T1$  by blast
  from Q obtain T2 where  $\Gamma \vdash e2 : T2$  by blast
  from IH1[OF  $\langle \Gamma \vdash e1 : T1 \rangle$ ] have  $fv e1 \subseteq dom \Gamma$  .
  moreover
  from IH2[OF  $\langle \Gamma \vdash e2 : T2 \rangle$ ] have  $fv e2 \subseteq dom \Gamma$  .
  ultimately have  $(fv e1) \cup (fv e2) \subseteq dom \Gamma$  by auto
  hence  $fv (e1 \ll bop \gg e2) \subseteq dom \Gamma$  by (simp add:FVe)
  with  $\langle x \in fv (e1 \ll bop \gg e2) \rangle$  show  $x \in dom \Gamma$  by auto
qed
qed

```

```

lemma exprNotNone:
assumes  $\Gamma \vdash e : T$  and  $fv e \subseteq dom s$ 
shows  $\llbracket e \rrbracket s \neq None$ 
using assms
proof (induct e arbitrary:  $\Gamma T s$ )
  case (Val v)
  show ?case by (simp add:Val)
next
  case (Var V)
  have  $\llbracket Var V \rrbracket s = s V$  by (simp add:Var)
  have  $V \in fv (Var V)$  by (auto simp add:FVv)
  with  $\langle fv (Var V) \subseteq dom s \rangle$  have  $V \in dom s$  by simp
  thus ?case by auto
next

```

```

case (BinOp e1 bop e2)
note IH1 =  $\langle \wedge T. [\Gamma \vdash e1 : T; fv\ e1 \subseteq dom\ s] \implies [e1]\ s \neq None \rangle$ 
note IH2 =  $\langle \wedge T. [\Gamma \vdash e2 : T; fv\ e2 \subseteq dom\ s] \implies [e2]\ s \neq None \rangle$ 
from  $\langle \Gamma \vdash e1 \ll bop \gg e2 : T \rangle$  have  $(\exists T1. \Gamma \vdash e1 : T1) \wedge (\exists T2. \Gamma \vdash e2 : T2)$ 
  by(rule exprBinopTypeable)
then obtain T1 T2 where  $\Gamma \vdash e1 : T1$  and  $\Gamma \vdash e2 : T2$  by blast
from  $\langle fv\ (e1 \ll bop \gg e2) \subseteq dom\ s \rangle$  have  $fv\ e1 \cup fv\ e2 \subseteq dom\ s$  by(simp
add:FVe)
hence  $fv\ e1 \subseteq dom\ s$  and  $fv\ e2 \subseteq dom\ s$  by auto
from IH1[OF  $\langle \Gamma \vdash e1 : T1 \rangle \langle fv\ e1 \subseteq dom\ s \rangle$ ] have  $[e1]\ s \neq None$  .
moreover from IH2[OF  $\langle \Gamma \vdash e2 : T2 \rangle \langle fv\ e2 \subseteq dom\ s \rangle$ ] have  $[e2]\ s \neq None$  .
ultimately show ?case
  apply(cases bop) apply auto
  apply(case-tac y,auto,case-tac ya,auto)+
done
qed

```

2.3 Noninterference definitions

2.3.1 Low Equivalence

Low Equivalence is reflexive even if the involved states are undefined. But in non-reflexive situations low variables must be initialized (i.e. $\in dom\ state$), otherwise the proof will not work. This is not a restriction, but a natural requirement, and could be formalized as part of a standard type system.

Low equivalence is also symmetric and transitive (see lemmas) hence an equivalence relation.

definition *lowEquiv* :: *typeEnv* \Rightarrow *state* \Rightarrow *state* \Rightarrow *bool* ($- \vdash - \approx_L -$)
where $\Gamma \vdash s1 \approx_L s2 \equiv \forall v \in dom\ \Gamma. \Gamma\ v = Some\ Low \longrightarrow (s1\ v = s2\ v)$

lemma *lowEquivReflexive*: $\Gamma \vdash s1 \approx_L s1$
by(*simp add:lowEquiv-def*)

lemma *lowEquivSymmetric*:
 $\Gamma \vdash s1 \approx_L s2 \implies \Gamma \vdash s2 \approx_L s1$
by(*simp add:lowEquiv-def*)

lemma *lowEquivTransitive*:
 $[\Gamma \vdash s1 \approx_L s2; \Gamma \vdash s2 \approx_L s3] \implies \Gamma \vdash s1 \approx_L s3$
by(*simp add:lowEquiv-def*)

2.3.2 Non Interference

definition *nonInterference* :: *typeEnv* \Rightarrow *com* \Rightarrow *bool*
where *nonInterference* $\Gamma\ c \equiv$
 $(\forall s1\ s2\ s1'\ s2'. (\Gamma \vdash s1 \approx_L s2 \wedge \langle c, s1 \rangle \rightarrow^* \langle Skip, s1 \rangle \wedge \langle c, s2 \rangle \rightarrow^* \langle Skip, s2 \rangle)$
 $\longrightarrow \Gamma \vdash s1' \approx_L s2')$

lemma nonInterferenceI:

$$\llbracket \bigwedge s1\ s2\ s1'\ s2'. \llbracket \Gamma \vdash s1 \approx_L s2; \langle c, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \rangle; \langle c, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1' \approx_L s2' \rrbracket \implies \text{nonInterference } \Gamma\ c$$
by(*auto simp:nonInterference-def*)

lemma interpretLow:
assumes $\Gamma \vdash s1 \approx_L s2$ **and** $\text{all}:\forall V \in \text{fv } e. \Gamma\ V = \text{Some Low}$
shows $\llbracket e \rrbracket s1 = \llbracket e \rrbracket s2$
using *all*
proof (*induct e*)
 case (*Val v*)
 show *?case* **by** (*simp add: Val*)
next
 case (*Var V*)
 have $\llbracket \text{Var } V \rrbracket s1 = s1\ V$ **and** $\llbracket \text{Var } V \rrbracket s2 = s2\ V$ **by**(*auto simp:Var*)
 have $V \in \text{fv } (\text{Var } V)$ **by**(*simp add:FVv*)
 from $\langle V \in \text{fv } (\text{Var } V) \rangle \langle \forall X \in \text{fv } (\text{Var } V). \Gamma\ X = \text{Some Low} \rangle$ **have** $\Gamma\ V = \text{Some Low}$ **by** *simp*
 with *assms* **have** $s1\ V = s2\ V$ **by**(*auto simp add:lowEquiv-def*)
 thus *?case* **by** *auto*
next
 case (*BinOp e1 bop e2*)
 note *IH1* = $\langle \forall V \in \text{fv } e1. \Gamma\ V = \text{Some Low} \implies \llbracket e1 \rrbracket s1 = \llbracket e1 \rrbracket s2 \rangle$
 note *IH2* = $\langle \forall V \in \text{fv } e2. \Gamma\ V = \text{Some Low} \implies \llbracket e2 \rrbracket s1 = \llbracket e2 \rrbracket s2 \rangle$
 from $\langle \forall V \in \text{fv } (e1 \llbracket bop \rrbracket e2). \Gamma\ V = \text{Some Low} \rangle$ **have** $\forall V \in \text{fv } e1. \Gamma\ V = \text{Some Low}$ **and** $\forall V \in \text{fv } e2. \Gamma\ V = \text{Some Low}$ **by** *auto*
 from *IH1* [*OF* $\langle \forall V \in \text{fv } e1. \Gamma\ V = \text{Some Low} \rangle$] **have** $\llbracket e1 \rrbracket s1 = \llbracket e1 \rrbracket s2$.
 moreover
 from *IH2* [*OF* $\langle \forall V \in \text{fv } e2. \Gamma\ V = \text{Some Low} \rangle$] **have** $\llbracket e2 \rrbracket s1 = \llbracket e2 \rrbracket s2$.
 ultimately show *?case* **by**(*cases* $\llbracket e1 \rrbracket s2, \text{auto}$)
qed

lemma interpretLow2:
assumes $\Gamma \vdash e : \text{Low}$ **and** $\Gamma \vdash s1 \approx_L s2$ **shows** $\llbracket e \rrbracket s1 = \llbracket e \rrbracket s2$
proof –
 from $\langle \Gamma \vdash e : \text{Low} \rangle$ **have** $\text{fv } e \subseteq \text{dom } \Gamma$ **by**(*auto dest:typeableFreevars*)
 have $\forall x \in \text{fv } e. \Gamma\ x = \text{Some Low}$
 proof
 fix *x* **assume** $x \in \text{fv } e$
 with $\langle \Gamma \vdash e : \text{Low} \rangle$ **show** $\Gamma\ x = \text{Some Low}$ **by**(*auto intro:exprTypingLow*)
 qed
 with $\langle \Gamma \vdash s1 \approx_L s2 \rangle$ **show** *?thesis* **by**(*rule interpretLow*)
qed

lemma assignNIhighlemma:

```

assumes  $\Gamma \vdash s1 \approx_L s2$  and  $\Gamma V = \text{Some High}$  and  $s1' = s1(V := \llbracket e \rrbracket s1)$ 
and  $s2' = s2(V := \llbracket e \rrbracket s2)$ 
shows  $\Gamma \vdash s1' \approx_L s2'$ 
proof -
{ fix  $V'$  assume  $V' \in \text{dom } \Gamma$  and  $\Gamma V' = \text{Some Low}$ 
  from  $\langle \Gamma \vdash s1 \approx_L s2 \rangle$   $\langle \Gamma V' = \text{Some Low} \rangle$  have  $s1 V' = s2 V'$ 
    by(auto simp add:lowEquiv-def)
  have  $s1' V' = s2' V'$ 
  proof(cases V' = V)
    case True
      with  $\langle \Gamma V' = \text{Some Low} \rangle$   $\langle \Gamma V = \text{Some High} \rangle$  have False by simp
      thus ?thesis by simp
    next
      case False
        with  $\langle s1' = s1(V := \llbracket e \rrbracket s1) \rangle$   $\langle s2' = s2(V := \llbracket e \rrbracket s2) \rangle$ 
        have  $s1 V' = s1' V'$  and  $s2 V' = s2' V'$  by auto
        with  $\langle s1 V' = s2 V' \rangle$  show ?thesis by simp
      qed
    }
  thus ?thesis by(auto simp add:lowEquiv-def)
qed

```

lemma *assignNlLowlemma*:

```

assumes  $\Gamma \vdash s1 \approx_L s2$  and  $\Gamma V = \text{Some Low}$  and  $\Gamma \vdash e : \text{Low}$ 
and  $s1' = s1(V := \llbracket e \rrbracket s1)$  and  $s2' = s2(V := \llbracket e \rrbracket s2)$ 
shows  $\Gamma \vdash s1' \approx_L s2'$ 
proof -
{ fix  $V'$  assume  $V' \in \text{dom } \Gamma$  and  $\Gamma V' = \text{Some Low}$ 
  from  $\langle \Gamma \vdash s1 \approx_L s2 \rangle$   $\langle \Gamma V' = \text{Some Low} \rangle$ 
  have  $s1 V' = s2 V'$  by(auto simp add:lowEquiv-def)
  have  $s1' V' = s2' V'$ 
  proof(cases V' = V)
    case True
      with  $\langle s1' = s1(V := \llbracket e \rrbracket s1) \rangle$   $\langle s2' = s2(V := \llbracket e \rrbracket s2) \rangle$ 
      have  $s1' V' = \llbracket e \rrbracket s1$  and  $s2' V' = \llbracket e \rrbracket s2$  by auto
      from  $\langle \Gamma \vdash e : \text{Low} \rangle$   $\langle \Gamma \vdash s1 \approx_L s2 \rangle$  have  $\llbracket e \rrbracket s1 = \llbracket e \rrbracket s2$ 
        by(auto intro:interpretLow2)
      with  $\langle s1' V' = \llbracket e \rrbracket s1 \rangle$   $\langle s2' V' = \llbracket e \rrbracket s2 \rangle$  show ?thesis by simp
    next
      case False
        with  $\langle s1' = s1(V := \llbracket e \rrbracket s1) \rangle$   $\langle s2' = s2(V := \llbracket e \rrbracket s2) \rangle$ 
        have  $s1' V' = s1 V'$  and  $s2' V' = s2 V'$  by auto
        with  $\langle s1 V' = s2 V' \rangle$  have  $s1' V' = s2' V'$  by simp
        with False  $\langle s1' V' = s1 V' \rangle$   $\langle s2' V' = s2 V' \rangle$ 
        show ?thesis by auto
      qed
    }
  qed
}

```

thus *?thesis* **by**(*simp add:lowEquiv-def*)
qed

Sequential Compositionality is given the status of a theorem because compositionality is no longer valid in case of concurrency

theorem *SeqCompositionality*:

assumes *nonInterference* Γ *c1* **and** *nonInterference* Γ *c2*
shows *nonInterference* Γ (*c1*;;*c2*)
proof(*rule nonInterferenceI*)
fix *s1 s2 s1' s2'*
assume $\Gamma \vdash s1 \approx_L s2$ **and** $\langle c1;;c2,s1 \rangle \rightarrow^* \langle Skip,s1' \rangle$
and $\langle c1;;c2,s2 \rangle \rightarrow^* \langle Skip,s2' \rangle$
from $\langle \langle c1;;c2,s1 \rangle \rightarrow^* \langle Skip,s1' \rangle \rangle$ **obtain** *s1''* **where** $\langle c1,s1 \rangle \rightarrow^* \langle Skip,s1'' \rangle$
and $\langle c2,s1'' \rangle \rightarrow^* \langle Skip,s1' \rangle$ **by**(*auto dest:Seq-reds*)
from $\langle \langle c1;;c2,s2 \rangle \rightarrow^* \langle Skip,s2' \rangle \rangle$ **obtain** *s2''* **where** $\langle c1,s2 \rangle \rightarrow^* \langle Skip,s2'' \rangle$
and $\langle c2,s2'' \rangle \rightarrow^* \langle Skip,s2' \rangle$ **by**(*auto dest:Seq-reds*)
from $\langle \Gamma \vdash s1 \approx_L s2 \rangle \langle \langle c1,s1 \rangle \rightarrow^* \langle Skip,s1'' \rangle \rangle \langle \langle c1,s2 \rangle \rightarrow^* \langle Skip,s2'' \rangle \rangle$
 $\langle nonInterference \Gamma c1 \rangle$
have $\Gamma \vdash s1'' \approx_L s2''$ **by**(*auto simp:nonInterference-def*)
with $\langle \langle c2,s1'' \rangle \rightarrow^* \langle Skip,s1' \rangle \rangle \langle \langle c2,s2'' \rangle \rightarrow^* \langle Skip,s2' \rangle \rangle \langle nonInterference \Gamma c2 \rangle$
show $\Gamma \vdash s1' \approx_L s2'$ **by**(*auto simp:nonInterference-def*)
qed

lemma *WhileStepInduct*:

assumes *while*: $\langle while (b) c,s1 \rangle \rightarrow^* \langle Skip,s2 \rangle$
and *body*: $\bigwedge s1 s2. \langle c,s1 \rangle \rightarrow^* \langle Skip,s2 \rangle \implies \Gamma \vdash s1 \approx_L s2$ **and** $\Gamma, High \vdash c$
shows $\Gamma \vdash s1 \approx_L s2$
using *while*
proof (*induct rule:while-reds-induct*)
case (*false s*) **thus** *?case* **by**(*auto simp add:lowEquiv-def*)
next
case (*true s1 s3*)
from *body*[*OF* $\langle \langle c,s1 \rangle \rightarrow^* \langle Skip,s3 \rangle \rangle$] **have** $\Gamma \vdash s1 \approx_L s3$ **by** *simp*
with $\langle \Gamma \vdash s3 \approx_L s2 \rangle$ **show** *?case* **by**(*auto intro:lowEquivTransitive*)
qed

In case control conditions from if/while are high, the body of an if/while must not change low variables in order to prevent implicit flow. That is, start and end state of any if/while body must be low equivalent.

theorem *highBodies*:

assumes $\Gamma, High \vdash c$ **and** $\langle c,s1 \rangle \rightarrow^* \langle Skip,s2 \rangle$
shows $\Gamma \vdash s1 \approx_L s2$
— all intermediate states must be well formed, otherwise the proof does not work for uninitialized variables. Thus it is propagated through the theorem conclusion
using *assms*
proof(*induct c arbitrary:s1 s2 rule:com.induct*)
case *Skip*

```

from  $\langle \langle \text{Skip}, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rangle$  have  $s1 = s2$  by (auto dest:Skip-reds)
thus ?case by (simp add:lowEquiv-def)
next
  case (LAss V e)
from  $\langle \Gamma, \text{High} \vdash V := e \rangle$  have  $\Gamma V = \text{Some High}$  by (auto elim:secComTyping.cases)
from  $\langle \langle V := e, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rangle$  have  $s2 = s1 (V := \llbracket e \rrbracket s1)$  by (auto intro:LAss-reds)
{ fix  $V'$  assume  $V' \in \text{dom } \Gamma$  and  $\Gamma V' = \text{Some Low}$ 
  have  $s1 V' = s2 V'$ 
  proof (cases V' = V)
    case True
    with  $\langle \Gamma V' = \text{Some Low} \rangle$   $\langle \Gamma V = \text{Some High} \rangle$  have False by simp
    thus ?thesis by simp
  next
    case False
    with  $\langle s2 = s1 (V := \llbracket e \rrbracket s1) \rangle$  show ?thesis by simp
  qed
}
thus ?case by (auto simp add:lowEquiv-def)
next
  case (Seq c1 c2)
note  $IH1 = \langle \wedge s1 s2. \llbracket \Gamma, \text{High} \vdash c1; \langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1 \approx_L s2 \rangle$ 
note  $IH2 = \langle \wedge s1 s2. \llbracket \Gamma, \text{High} \vdash c2; \langle c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1 \approx_L s2 \rangle$ 
from  $\langle \Gamma, \text{High} \vdash c1 ;; c2 \rangle$  have  $\Gamma, \text{High} \vdash c1$  and  $\Gamma, \text{High} \vdash c2$ 
  by (auto elim:secComTyping.cases)
from  $\langle \langle c1 ;; c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rangle$ 
have  $\exists s3. \langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s3 \rangle \wedge \langle c2, s3 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle$  by (auto intro:Seq-reds)
then obtain  $s3$  where  $\langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s3 \rangle$  and  $\langle c2, s3 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle$  by auto
from  $IH1 [OF \langle \Gamma, \text{High} \vdash c1 \rangle \langle \langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s3 \rangle \rangle]$ 
have  $\Gamma \vdash s1 \approx_L s3$  by simp
from  $IH2 [OF \langle \Gamma, \text{High} \vdash c2 \rangle \langle \langle c2, s3 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rangle]$ 
have  $\Gamma \vdash s3 \approx_L s2$  by simp
from  $\langle \Gamma \vdash s1 \approx_L s3 \rangle$   $\langle \Gamma \vdash s3 \approx_L s2 \rangle$  show ?case
  by (auto intro:lowEquivTransitive)
next
  case (Cond b c1 c2)
note  $IH1 = \langle \wedge s1 s2. \llbracket \Gamma, \text{High} \vdash c1; \langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1 \approx_L s2 \rangle$ 
note  $IH2 = \langle \wedge s1 s2. \llbracket \Gamma, \text{High} \vdash c2; \langle c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1 \approx_L s2 \rangle$ 
from  $\langle \Gamma, \text{High} \vdash \text{if } (b) c1 \text{ else } c2 \rangle$  have  $\Gamma, \text{High} \vdash c1$  and  $\Gamma, \text{High} \vdash c2$ 
  by (auto elim:secComTyping.cases)
from  $\langle \langle \text{if } (b) c1 \text{ else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rangle$ 
have  $\llbracket b \rrbracket s1 = \text{Some true} \vee \llbracket b \rrbracket s1 = \text{Some false}$  by (auto dest:Cond-True-or-False)
thus ?case
proof
  assume  $\llbracket b \rrbracket s1 = \text{Some true}$ 
  with  $\langle \langle \text{if } (b) c1 \text{ else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rangle$  have  $\langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle$ 
  by (auto intro:CondTrue-reds)
  from  $IH1 [OF \langle \Gamma, \text{High} \vdash c1 \rangle \text{this}]$  show ?thesis .
next
  assume  $\llbracket b \rrbracket s1 = \text{Some false}$ 

```

with $\langle \text{if } (b) \ c1 \ \text{else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle$ **have** $\langle c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle$
by (*auto intro: CondFalse-reds*)
from $IH2[OF \ \langle \Gamma, High \vdash c2 \rangle \ \text{this}]$ **show** *?thesis* .
qed
next
case (*While b c'*)
note $IH = \langle \bigwedge s1 \ s2. \llbracket \Gamma, High \vdash c'; \langle c', s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1 \approx_L s2 \rangle$
from $\langle \Gamma, High \vdash \text{while } (b) \ c' \rangle$ **have** $\Gamma, High \vdash c'$ **by** (*auto elim: secComTyping.cases*)
from $IH[OF \ \text{this}]$
have $\bigwedge s1 \ s2. \llbracket \langle c', s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle \rrbracket \implies \Gamma \vdash s1 \approx_L s2$.
with $\langle \text{while } (b) \ c', s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle$ $\langle \Gamma, High \vdash c' \rangle$
show *?case* **by** (*auto dest: WhileStepInduct*)
qed

lemma *CondHighCompositionality*:
assumes $\Gamma, High \vdash \text{if } (b) \ c1 \ \text{else } c2$
shows *nonInterference* $\Gamma \ \langle \text{if } (b) \ c1 \ \text{else } c2 \rangle$
proof (*rule nonInterferenceI*)
fix $s1 \ s2 \ s1' \ s2'$
assume $\Gamma \vdash s1 \approx_L s2$ **and** $\langle \text{if } (b) \ c1 \ \text{else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1' \rangle$
and $\langle \text{if } (b) \ c1 \ \text{else } c2, s2 \rangle \rightarrow^* \langle \text{Skip}, s2' \rangle$
show $\Gamma \vdash s1' \approx_L s2'$
proof –
from $\langle \Gamma, High \vdash \text{if } (b) \ c1 \ \text{else } c2 \rangle$ $\langle \text{if } (b) \ c1 \ \text{else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1' \rangle$
have $\Gamma \vdash s1 \approx_L s1'$ **by** (*auto dest: highBodies*)
from $\langle \Gamma, High \vdash \text{if } (b) \ c1 \ \text{else } c2 \rangle$ $\langle \text{if } (b) \ c1 \ \text{else } c2, s2 \rangle \rightarrow^* \langle \text{Skip}, s2' \rangle$
have $\Gamma \vdash s2 \approx_L s2'$ **by** (*auto dest: highBodies*)
with $\langle \Gamma \vdash s1 \approx_L s2 \rangle$ **have** $\Gamma \vdash s1 \approx_L s2'$ **by** (*auto intro: lowEquivTransitive*)
from $\langle \Gamma \vdash s1 \approx_L s1' \rangle$ **have** $\Gamma \vdash s1' \approx_L s1$ **by** (*auto intro: lowEquivSymmetric*)
with $\langle \Gamma \vdash s1 \approx_L s2' \rangle$ **show** *?thesis* **by** (*auto intro: lowEquivTransitive*)
qed
qed

lemma *CondLowCompositionality*:
assumes *nonInterference* $\Gamma \ c1$ **and** *nonInterference* $\Gamma \ c2$ **and** $\Gamma \vdash b : Low$
shows *nonInterference* $\Gamma \ \langle \text{if } (b) \ c1 \ \text{else } c2 \rangle$
proof (*rule nonInterferenceI*)
fix $s1 \ s2 \ s1' \ s2'$
assume $\Gamma \vdash s1 \approx_L s2$ **and** $\langle \text{if } (b) \ c1 \ \text{else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1' \rangle$
and $\langle \text{if } (b) \ c1 \ \text{else } c2, s2 \rangle \rightarrow^* \langle \text{Skip}, s2' \rangle$
from $\langle \Gamma \vdash b : Low \rangle$ $\langle \Gamma \vdash s1 \approx_L s2 \rangle$ **have** $\llbracket b \rrbracket s1 = \llbracket b \rrbracket s2$
by (*auto intro: interpretLow2*)
from $\langle \text{if } (b) \ c1 \ \text{else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1' \rangle$
have $\llbracket b \rrbracket s1 = \text{Some true} \vee \llbracket b \rrbracket s1 = \text{Some false}$ **by** (*auto dest: Cond-True-or-False*)
thus $\Gamma \vdash s1' \approx_L s2'$

proof

assume $\llbracket b \rrbracket s1 = \text{Some true}$
with $\langle \llbracket b \rrbracket s1 = \llbracket b \rrbracket s2 \rangle$ **have** $\llbracket b \rrbracket s2 = \text{Some true}$ **by**(*auto intro:CondTrue-reds*)
from $\langle \llbracket b \rrbracket s1 = \text{Some true} \rangle \langle \langle \text{if } (b) \text{ } c1 \text{ else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle$
have $\langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle$ **by**(*auto intro:CondTrue-reds*)
from $\langle \llbracket b \rrbracket s2 = \text{Some true} \rangle \langle \langle \text{if } (b) \text{ } c1 \text{ else } c2, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle \rangle$
have $\langle c1, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle$ **by**(*auto intro:CondTrue-reds*)
with $\langle \langle c1, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle \langle \Gamma \vdash s1 \approx_L s2 \rangle \langle \text{nonInterference } \Gamma \text{ } c1 \rangle$
show *?thesis* **by**(*auto simp:nonInterference-def*)

next

assume $\llbracket b \rrbracket s1 = \text{Some false}$
with $\langle \llbracket b \rrbracket s1 = \llbracket b \rrbracket s2 \rangle$ **have** $\llbracket b \rrbracket s2 = \text{Some false}$ **by**(*auto intro:CondTrue-reds*)
from $\langle \llbracket b \rrbracket s1 = \text{Some false} \rangle \langle \langle \text{if } (b) \text{ } c1 \text{ else } c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle$
have $\langle c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle$ **by**(*auto intro:CondFalse-reds*)
from $\langle \llbracket b \rrbracket s2 = \text{Some false} \rangle \langle \langle \text{if } (b) \text{ } c1 \text{ else } c2, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle \rangle$
have $\langle c2, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle$ **by**(*auto intro:CondFalse-reds*)
with $\langle \langle c2, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle \langle \Gamma \vdash s1 \approx_L s2 \rangle \langle \text{nonInterference } \Gamma \text{ } c2 \rangle$
show *?thesis* **by**(*auto simp:nonInterference-def*)

qed

qed

lemma *WhileHighCompositionality*:

assumes $\Gamma, \text{High} \vdash \text{while } (b) \text{ } c'$
shows $\text{nonInterference } \Gamma \text{ } (\text{while } (b) \text{ } c')$

proof(*rule nonInterferenceI*)

fix $s1 \ s2 \ s1' \ s2'$

assume $\Gamma \vdash s1 \approx_L s2$ **and** $\langle \text{while } (b) \text{ } c', s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle$
and $\langle \text{while } (b) \text{ } c', s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle$

show $\Gamma \vdash s1' \approx_L s2'$

proof –

from $\langle \Gamma, \text{High} \vdash \text{while } (b) \text{ } c' \rangle \langle \langle \text{while } (b) \text{ } c', s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle$
have $\Gamma \vdash s1 \approx_L s1'$ **by**(*auto dest:highBodies*)
from $\langle \Gamma, \text{High} \vdash \text{while } (b) \text{ } c' \rangle \langle \langle \text{while } (b) \text{ } c', s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle \rangle$
have $\Gamma \vdash s2 \approx_L s2'$ **by**(*auto dest:highBodies*)
with $\langle \Gamma \vdash s1 \approx_L s2 \rangle$ **have** $\Gamma \vdash s1 \approx_L s2'$ **by**(*auto intro:lowEquivTransitive*)
from $\langle \Gamma \vdash s1 \approx_L s1' \rangle$ **have** $\Gamma \vdash s1' \approx_L s1$ **by**(*auto intro:lowEquivSymmetric*)
with $\langle \Gamma \vdash s1 \approx_L s2' \rangle$ **show** *?thesis* **by**(*auto intro:lowEquivTransitive*)

qed

qed

lemma *WhileLowStepInduct*:

assumes $\text{while1}: \langle \text{while } (b) \text{ } c', s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle$

and $\text{while2}: \langle \text{while } (b) \text{ } c', s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle$

and $\Gamma \vdash b : \text{Low}$

and $\text{body}: \bigwedge s1 \ s1' \ s2 \ s2'. \llbracket \langle c', s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle; \langle c', s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle; \Gamma \vdash s1 \approx_L s2 \rrbracket \implies \Gamma \vdash s1' \approx_L s2'$

and $le: \Gamma \vdash s1 \approx_L s2$
shows $\Gamma \vdash s1' \approx_L s2'$
using *while1 le while2*
proof (*induct arbitrary:s2 rule:while-reds-induct*)
case (*false s1*)
from $\langle \Gamma \vdash b : Low \rangle \langle \Gamma \vdash s1 \approx_L s2 \rangle$ **have** $\llbracket b \rrbracket s1 = \llbracket b \rrbracket s2$ **by** (*auto intro:interpretLow2*)
with $\langle \llbracket b \rrbracket s1 = Some\ false \rangle$ **have** $\llbracket b \rrbracket s2 = Some\ false$ **by** *simp*
with $\langle \langle while\ (b)\ c',s2 \rangle \rightarrow^* \langle Skip,s2 \rangle \rangle$ **have** $s2 = s2'$ **by** (*auto intro:WhileFalse-reds*)
with $\langle \Gamma \vdash s1 \approx_L s2 \rangle$ **show** *?case* **by** *auto*
next
case (*true s1 s1''*)
note $IH = \langle \bigwedge s2''. \llbracket \Gamma \vdash s1'' \approx_L s2''; \langle while\ (b)\ c',s2'' \rangle \rightarrow^* \langle Skip,s2'' \rangle \rrbracket$
 $\implies \Gamma \vdash s1' \approx_L s2'$
from $\langle \Gamma \vdash b : Low \rangle \langle \Gamma \vdash s1 \approx_L s2 \rangle$ **have** $\llbracket b \rrbracket s1 = \llbracket b \rrbracket s2$ **by** (*auto intro:interpretLow2*)
with $\langle \llbracket b \rrbracket s1 = Some\ true \rangle$ **have** $\llbracket b \rrbracket s2 = Some\ true$ **by** *simp*
with $\langle \langle while\ (b)\ c',s2 \rangle \rightarrow^* \langle Skip,s2 \rangle \rangle$ **obtain** $s2''$ **where** $\langle c',s2 \rangle \rightarrow^* \langle Skip,s2'' \rangle$
and $\langle while\ (b)\ c',s2'' \rangle \rightarrow^* \langle Skip,s2'' \rangle$ **by** (*auto dest:WhileTrue-reds*)
from *body[OF* $\langle \langle c',s1 \rangle \rightarrow^* \langle Skip,s1'' \rangle \rangle \langle \langle c',s2 \rangle \rightarrow^* \langle Skip,s2'' \rangle \rangle \langle \Gamma \vdash s1 \approx_L s2 \rangle$
have $\Gamma \vdash s1'' \approx_L s2''$.
from IH [*OF this* $\langle \langle while\ (b)\ c',s2'' \rangle \rightarrow^* \langle Skip,s2'' \rangle \rangle$] **show** *?case* .
qed

lemma *WhileLowCompositionality*:

assumes *nonInterference* $\Gamma\ c'$ **and** $\Gamma \vdash b : Low$ **and** $\Gamma, Low \vdash c'$
shows *nonInterference* $\Gamma\ (while\ (b)\ c')$
proof (*rule nonInterferenceI*)
fix $s1\ s2\ s1'\ s2'$
assume $\Gamma \vdash s1 \approx_L s2$ **and** $\langle while\ (b)\ c',s1 \rangle \rightarrow^* \langle Skip,s1' \rangle$
and $\langle while\ (b)\ c',s2 \rangle \rightarrow^* \langle Skip,s2' \rangle$
{ fix $s1\ s2\ s1''\ s2''$
assume $\langle c',s1 \rangle \rightarrow^* \langle Skip,s1'' \rangle$ **and** $\langle c',s2 \rangle \rightarrow^* \langle Skip,s2'' \rangle$
and $\Gamma \vdash s1 \approx_L s2$
with $\langle nonInterference\ \Gamma\ c' \rangle$ **have** $\Gamma \vdash s1'' \approx_L s2''$
by (*auto simp:nonInterference-def*)
}
hence $\bigwedge s1\ s1''\ s2\ s2''. \llbracket \langle c',s1 \rangle \rightarrow^* \langle Skip,s1'' \rangle; \langle c',s2 \rangle \rightarrow^* \langle Skip,s2'' \rangle; \Gamma \vdash s1 \approx_L s2 \rrbracket \implies \Gamma \vdash s1'' \approx_L s2''$ **by** *auto*
with $\langle \langle while\ (b)\ c',s1 \rangle \rightarrow^* \langle Skip,s1' \rangle \rangle \langle \langle while\ (b)\ c',s2 \rangle \rightarrow^* \langle Skip,s2' \rangle \rangle$
 $\langle \Gamma \vdash b : Low \rangle \langle \Gamma \vdash s1 \approx_L s2 \rangle$ **show** $\Gamma \vdash s1' \approx_L s2'$
by (*auto intro:WhileLowStepInduct*)
qed

and now: the main theorem:

theorem *secTypeImpliesNonInterference*:

$\Gamma, T \vdash c \implies nonInterference\ \Gamma\ c$
proof (*induct c arbitrary:T rule:com.induct*)
case *Skip*

```

show ?case
proof(rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume  $\Gamma \vdash s1 \approx_L s2$  and  $\langle \text{Skip}, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle$  and  $\langle \text{Skip}, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle$ 
  from  $\langle \langle \text{Skip}, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle$  have  $s1 = s1'$  by(auto dest:Skip-reds)
  from  $\langle \langle \text{Skip}, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle \rangle$  have  $s2 = s2'$  by(auto dest:Skip-reds)
  from  $\langle \Gamma \vdash s1 \approx_L s2 \rangle$  and  $\langle s1 = s1' \rangle$  and  $\langle s2 = s2' \rangle$ 
  show  $\Gamma \vdash s1' \approx_L s2'$  by simp
qed
next
case (LAss V e)
from  $\langle \Gamma, T \vdash V := e \rangle$ 
have varprem:  $(\Gamma V = \text{Some High}) \vee (\Gamma V = \text{Some Low} \wedge \Gamma \vdash e : \text{Low} \wedge T = \text{Low})$ 
by (auto elim:secComTyping.cases)
show ?case
proof(rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume  $\Gamma \vdash s1 \approx_L s2$  and  $\langle V := e, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle$  and  $\langle V := e, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle$ 
  from  $\langle \langle V := e, s1 \rangle \rightarrow^* \langle \text{Skip}, s1 \wedge \rangle \rangle$  have  $s1' = s1(V := \llbracket e \rrbracket s1)$  by(auto intro:LAss-reds)
  from  $\langle \langle V := e, s2 \rangle \rightarrow^* \langle \text{Skip}, s2 \wedge \rangle \rangle$  have  $s2' = s2(V := \llbracket e \rrbracket s2)$  by(auto intro:LAss-reds)
  from varprem show  $\Gamma \vdash s1' \approx_L s2'$ 
  proof
    assume  $\Gamma V = \text{Some High}$ 
    with  $\langle \Gamma \vdash s1 \approx_L s2 \rangle$   $\langle s1' = s1(V := \llbracket e \rrbracket s1) \rangle$   $\langle s2' = s2(V := \llbracket e \rrbracket s2) \rangle$ 
    show ?thesis by(auto intro:assignNIhighlemma)
  next
    assume  $\Gamma V = \text{Some Low} \wedge \Gamma \vdash e : \text{Low} \wedge T = \text{Low}$ 
    with  $\langle \Gamma \vdash s1 \approx_L s2 \rangle$   $\langle s1' = s1(V := \llbracket e \rrbracket s1) \rangle$   $\langle s2' = s2(V := \llbracket e \rrbracket s2) \rangle$ 
    show ?thesis by(auto intro:assignNILowlemma)
  qed
qed
next
case (Seq c1 c2)
note IH1 =  $\langle \wedge T. \Gamma, T \vdash c1 \implies \text{nonInterference } \Gamma c1 \rangle$ 
note IH2 =  $\langle \wedge T. \Gamma, T \vdash c2 \implies \text{nonInterference } \Gamma c2 \rangle$ 
show ?case
proof (cases T)
  case High
  with  $\langle \Gamma, T \vdash c1 ;; c2 \rangle$  have  $\Gamma, \text{High} \vdash c1$  and  $\Gamma, \text{High} \vdash c2$ 
  by(auto elim:secComTyping.cases)
  from IH1[OF  $\langle \Gamma, \text{High} \vdash c1 \rangle$ ] have nonInterference  $\Gamma c1$  .
  moreover
  from IH2[OF  $\langle \Gamma, \text{High} \vdash c2 \rangle$ ] have nonInterference  $\Gamma c2$  .
  ultimately show ?thesis by (auto intro:SeqCompositionality)
next

```

```

case Low
with  $\langle \Gamma, T \vdash c1 ;; c2 \rangle$ 
have  $(\Gamma, Low \vdash c1 \wedge \Gamma, Low \vdash c2) \vee \Gamma, High \vdash c1 ;; c2$ 
  by(auto elim:secComTyping.cases)
thus ?thesis
proof
  assume  $\Gamma, Low \vdash c1 \wedge \Gamma, Low \vdash c2$ 
  hence  $\Gamma, Low \vdash c1$  and  $\Gamma, Low \vdash c2$  by simp-all
  from IH1[OF  $\langle \Gamma, Low \vdash c1 \rangle$ ] have nonInterference  $\Gamma c1$  .
  moreover
  from IH2[OF  $\langle \Gamma, Low \vdash c2 \rangle$ ] have nonInterference  $\Gamma c2$  .
  ultimately show ?thesis by(auto intro:SeqCompositionality)
next
  assume  $\Gamma, High \vdash c1 ;; c2$ 
  hence  $\Gamma, High \vdash c1$  and  $\Gamma, High \vdash c2$  by(auto elim:secComTyping.cases)
  from IH1[OF  $\langle \Gamma, High \vdash c1 \rangle$ ] have nonInterference  $\Gamma c1$  .
  moreover
  from IH2[OF  $\langle \Gamma, High \vdash c2 \rangle$ ] have nonInterference  $\Gamma c2$  .
  ultimately show ?thesis by(auto intro:SeqCompositionality)
qed
qed
next
case (Cond b c1 c2)
note IH1 =  $\langle \bigwedge T. \Gamma, T \vdash c1 \implies nonInterference \Gamma c1 \rangle$ 
note IH2 =  $\langle \bigwedge T. \Gamma, T \vdash c2 \implies nonInterference \Gamma c2 \rangle$ 
show ?case
proof (cases T)
  case High
  with  $\langle \Gamma, T \vdash if (b) c1 else c2 \rangle$  show ?thesis
  by(auto intro:CondHighCompositionality)
next
  case Low
  with  $\langle \Gamma, T \vdash if (b) c1 else c2 \rangle$ 
  have  $(\Gamma \vdash b : Low \wedge \Gamma, Low \vdash c1 \wedge \Gamma, Low \vdash c2) \vee \Gamma, High \vdash if (b) c1 else c2$ 
  by(auto elim:secComTyping.cases)
  thus ?thesis
  proof
    assume  $\Gamma \vdash b : Low \wedge \Gamma, Low \vdash c1 \wedge \Gamma, Low \vdash c2$ 
    hence  $\Gamma \vdash b : Low$  and  $\Gamma, Low \vdash c1$  and  $\Gamma, Low \vdash c2$  by simp-all
    from IH1[OF  $\langle \Gamma, Low \vdash c1 \rangle$ ] have nonInterference  $\Gamma c1$  .
    moreover
    from IH2[OF  $\langle \Gamma, Low \vdash c2 \rangle$ ] have nonInterference  $\Gamma c2$  .
    ultimately show ?thesis using  $\langle \Gamma \vdash b : Low \rangle$ 
    by(auto intro:CondLowCompositionality)
  next
    assume  $\Gamma, High \vdash if (b) c1 else c2$ 
    thus ?thesis by(auto intro:CondHighCompositionality)
  qed
qed

```

```

next
  case (While b c')
  note IH = ⟨ $\wedge T. \Gamma, T \vdash c' \implies \text{nonInterference } \Gamma \ c'$ ⟩
  show ?case
  proof(cases T)
    case High
    with  $\langle \Gamma, T \vdash \text{while } (b) \ c' \rangle$  show ?thesis by(auto intro: WhileHighCompositionality)
  next
  case Low
  with  $\langle \Gamma, T \vdash \text{while } (b) \ c' \rangle$ 
  have  $(\Gamma \vdash b : \text{Low} \wedge \Gamma, \text{Low} \vdash c') \vee \Gamma, \text{High} \vdash \text{while } (b) \ c'$ 
    by(auto elim:secComTyping.cases)
  thus ?thesis
  proof
    assume  $\Gamma \vdash b : \text{Low} \wedge \Gamma, \text{Low} \vdash c'$ 
    hence  $\Gamma \vdash b : \text{Low}$  and  $\Gamma, \text{Low} \vdash c'$  by simp-all
    moreover
    from IH[OF  $\langle \Gamma, \text{Low} \vdash c' \rangle$ ] have nonInterference  $\Gamma \ c'$  .
    ultimately show ?thesis by(auto intro: WhileLowCompositionality)
  next
  assume  $\Gamma, \text{High} \vdash \text{while } (b) \ c'$ 
  thus ?thesis by(auto intro: WhileHighCompositionality)
  qed
qed
qed
end

```

```

theory Execute
imports secTypes
begin

```

3 Executing the small step semantics

```
code-pred (modes: i => o => bool as exec-red) red .
```

```
thm red.equation
```

```
definition [code]: one-step x = Predicate.the (exec-red x)
```

```
lemmas [code-pred-intro] = typeVal[where lev = Low] typeVal[where lev = High]
  typeVar[unfolded Predicate.eq-is-eq[symmetric]]
  typeBinOp1 typeBinOp2[where lev = Low] typeBinOp2[where lev = High] type-
  BinOp3[where lev = Low]
```

```
code-pred (modes: i => i => o => bool as compute-secExprTyping,
  i => i => i => bool as check-secExprTyping) secExprTyping
proof -
```

```

case secExprTyping
from this(1) show thesis
proof
  fix  $\Gamma$   $V$  lev assume  $a1 = \Gamma$   $a2 = \text{Val } V$   $a3 = \text{lev}$ 
  from secExprTyping(2-3) this show thesis by (cases lev) auto
next
  fix  $\Gamma$   $Vn$  lev
  assume  $a1 = \Gamma$   $a2 = \text{Var } Vn$   $a3 = \text{lev}$   $\Gamma$   $Vn = \text{Some } \text{lev}$ 
  from secExprTyping(4) this show thesis by (auto simp add: Predicate.eq-is-eq)
next
  fix  $\Gamma$   $e1$   $e2$  bop
  assume  $a1 = \Gamma$   $a2 = e1 \ll bop \gg e2$   $a3 = \text{Low}$ 
   $\Gamma \vdash e1 : \text{Low}$   $\Gamma \vdash e2 : \text{Low}$ 
  from secExprTyping(5) this show thesis by auto
next
  fix  $\Gamma$   $e1$   $e2$  lev bop
  assume  $a1 = \Gamma$   $a2 = e1 \ll bop \gg e2$   $a3 = \text{High}$ 
   $\Gamma \vdash e1 : \text{High}$   $\Gamma \vdash e2 : \text{lev}$ 
  from secExprTyping(6-7) this show thesis by (cases lev) (auto, blast)
next
  fix  $\Gamma$   $e1$   $e2$  lev bop
  assume  $a1 = \Gamma$   $a2 = e1 \ll bop \gg e2$   $a3 = \text{High}$ 
   $\Gamma \vdash e1 : \text{lev}$   $\Gamma \vdash e2 : \text{High}$ 
  from secExprTyping(7-8) this show thesis by (cases lev) (auto, blast)
qed
qed

```

```

lemmas [code-pred-intro] = typeSkip[where T=Low] typeSkip[where T=High]
  typeAssH[where T = Low] typeAssH[where T=High]
  typeAssL typeSeq typeWhile typeIf typeConvert

```

```

code-pred (modes: i => o => i => bool as compute-secComTyping,
  i => i => i => bool as check-secComTyping) secComTyping

```

```

proof –
  case secComTyping
  from this(1) show thesis
  proof
    fix  $\Gamma$   $T$  assume  $a1 = \Gamma$   $a2 = T$   $a3 = \text{Skip}$ 
    from secComTyping(2-3) this show thesis by (cases T) auto
  next
    fix  $\Gamma$   $V$   $T$   $e$  assume  $a1 = \Gamma$   $a2 = T$   $a3 = V := e$   $\Gamma$   $V = \text{Some } \text{High}$ 
    from secComTyping(4-5) this show thesis by (cases T) (auto, blast)
  next
    fix  $\Gamma$   $e$   $V$ 
    assume  $a1 = \Gamma$   $a2 = \text{Low}$   $a3 = V := e$   $\Gamma \vdash e : \text{Low}$   $\Gamma$   $V = \text{Some } \text{Low}$ 
    from secComTyping(6) this show thesis by auto
  next
    fix  $\Gamma$   $T$   $c1$   $c2$ 
    assume  $a1 = \Gamma$   $a2 = T$   $a3 = \text{Seq } c1$   $c2$   $\Gamma, T \vdash c1$   $\Gamma, T \vdash c2$ 

```

```

    from secComTyping(7) this show thesis by auto
next
fix  $\Gamma$  b T c
assume  $a1 = \Gamma$   $a2 = T$   $a3 = \text{while } (b) c \Gamma \vdash b : T \Gamma, T \vdash c$ 
from secComTyping(8) this show thesis by auto
next
fix  $\Gamma$  b T c1 c2
assume  $a1 = \Gamma$   $a2 = T$   $a3 = \text{if } (b) c1 \text{ else } c2 \Gamma \vdash b : T \Gamma, T \vdash c1 \Gamma, T \vdash c2$ 
from secComTyping(9) this show thesis by blast
next
fix  $\Gamma$  c
assume  $a1 = \Gamma$   $a2 = \text{Low}$   $a3 = c \Gamma, \text{High} \vdash c$ 
from secComTyping(10) this show thesis by blast
qed
qed

```

thm *secComTyping.equation*

3.1 An example taken from Volpano, Smith, Irvine

definition $com = \text{if } (\text{Var } "x" \ll \text{Eq} \gg \text{Val } (\text{Intg } 1)) ("y" := \text{Val } (\text{Intg } 1)) \text{ else } ("y" := \text{Val } (\text{Intg } 0))$

definition $Env = \text{map-of } [("x", \text{High}), ("y", \text{High})]$

values $\{T. Env \vdash (\text{Var } "x" \ll \text{Eq} \gg \text{Val } (\text{Intg } 1)): T\}$

value $[code] Env, \text{High} \vdash com$

value $[code] Env, \text{Low} \vdash com$

values $1 \{T. Env, T \vdash com\}$

definition $Env' = \text{map-of } [("x", \text{Low}), ("y", \text{High})]$

value $[code] Env', \text{Low} \vdash com$

value $[code] Env', \text{High} \vdash com$

values $1 \{T. Env, T \vdash com\}$

end

References

- [1] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
- [2] Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2-3):167–187, 1996.