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

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

March 2, 2020

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
type-synonym 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 — 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(Bool(i1 + i2))
| binop Sub (Intg i1) (Intg i2) = Some(Bool(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
1.3 State

type-synonym 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 
where Val : [Val v] s = Some v
| Var : [Var V] s = s V
| BinOp : [e1 \ll bop \gg e2] s = (case [e1] s of None \Rightarrow None
| Some v1 \Rightarrow (case [e2] s of None \Rightarrow None
| Some v2 \Rightarrow binop bop v1 v2))

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
((((1 \langle -,/ \rangle) \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:=[e] s) \rangle
| SeqRed:
\langle c1,s \rangle \rightarrow \langle c1’,s’ \rangle \implies \langle c1;;c2,s \rangle \rightarrow \langle c1’,c2,s’ \rangle
| RedSeq:
\langle Skip;;c2,s \rangle \rightarrow \langle c2,s \rangle
| RedCondTrue:
\langle b,s \rangle = Some true \implies \langle if (b) c1 else c2,s \rangle \rightarrow \langle c1,s \rangle
| RedCondFalse:
\langle b,s \rangle = Some false \implies \langle if (b) c1 else c2,s \rangle \rightarrow \langle c2,s \rangle
| RedWhileTrue:
\langle b,s \rangle = Some true \implies \langle while (b) c,s \rangle \rightarrow \langle c;;while (b) c,s \rangle
| RedWhileFalse:
\langle b,s \rangle = Some false \implies \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
(((1 \langle -,/ \rangle) \rightarrow+/ (1 \langle -,/ \rangle)) [0,0,0,0] 81) where
\langle c,s \rangle \rightarrow+/ \langle c’,s’ \rangle == red” (c,s) (c’,s’)

4
lemma Skip-reds:
\[(\text{Skip}, s) \rightarrow^* (c', s') \implies s = s' \land c' = \text{Skip}\]
by (blast elim: converse-rtranclpE red_cases)

lemma LAss-reds:
\[(V := e, s) \rightarrow^* (\text{Skip}, s') \implies s' = s(V := [e] s)\]
proof (induct \(V := e\) s rule: converse-rtranclp-induct2)
  case (step s c'' s'')
  hence \(c'' = \text{Skip}\) and \(s'' = s(V := ([e] s))\) by (auto elim: red_cases)
  with \(\langle c', s'\rangle \rightarrow^* (\text{Skip}, s')\): show \(?case\) by (auto dest: Skip-reds)
qed

lemma Seq2-reds:
\[(\text{Skip}; c_2, s) \rightarrow^* (\text{Skip}, s') \implies (c_2, s) \rightarrow^* (\text{Skip}, s')\]
by (induct \(c := \text{Skip}; c_2\) s rule: converse-rtranclp-induct2) (auto elim: red_cases)

lemma Seq-reds:
assumes \(\langle c_1; c_2, s\rangle \rightarrow^* (\text{Skip}, s')\)
obtains \(s''\) where \(\langle c_1, s\rangle \rightarrow^* (\text{Skip}, s'')\) and \(\langle c_2, s''\rangle \rightarrow^* (\text{Skip}, s')\)
proof
  have \(\exists s''. \langle c_1, s\rangle \rightarrow^* (\text{Skip}, s'') \land (c_2, s'') \rightarrow^* (\text{Skip}, s')\)
  proof
  \{ fix \(c, c'\)
  assume \(\langle c, s\rangle \rightarrow^* (c', s')\) and \(c = c_1 ; c_2\) and \(c' = \text{Skip}\)
  hence \(\exists s''. \langle c_1, s\rangle \rightarrow^* (\text{Skip}, s'') \land (c_2, s'') \rightarrow^* (\text{Skip}, s')\)
  proof (induct arbitrary: \(c_1\) rule: converse-rtranclp-induct2)
  case refl thus \(?case\) by simp
  next
  case (step c s c'' s'')
  note IH = \(\langle c', s''\rangle = c_1 ; c_2; c' = \text{Skip}\)
  \(\implies \exists sx. \langle c_1, s''\rangle \rightarrow^* (\text{Skip}, sx) \land (c_2, sx) \rightarrow^* (\text{Skip}, s')\)
  from step
  have \(\langle c_1 ; c_2, s\rangle \rightarrow (c'', s'')\) by simp
  hence \(\langle c_1 = \text{Skip} \land c'' = c_2 \land s = s''\rangle \lor
\(\exists c_1'. \langle c_1, s\rangle \rightarrow (c_1', s'') \land c'' = c_1 ; c_2\)
  by (auto elim: red_cases)
  thus \(?case\)
  proof
  assume \(c_1 = \text{Skip} \land c'' = c_2 \land s = s''\)
  with \(\langle c', s''\rangle \rightarrow^* (c', s')\) \(c' = \text{Skip}\)
  show \(?thesis\) by auto
  next
  assume \(\exists c_1'. \langle c_1, s\rangle \rightarrow (c_1', s'') \land c'' = c_1 ; c_2\)
  then obtain \(c_1'\) where \(\langle c_1, s\rangle \rightarrow (c_1', s'')\) and \(c'' = c_1 ; c_2\) by blast
  from IH[OF \(c' = c_1' ; c_2; c' = \text{Skip}\)]
  obtain sx where \(\langle c_1, s''\rangle \rightarrow^* (\text{Skip}, sx)\) and \(\langle c_2, sx\rangle \rightarrow^* (\text{Skip}, s')\)
  by blast
  from \(\langle c_1, s\rangle \rightarrow (c_1', s'')\) \(\langle c_1', s''\rangle \rightarrow^* (\text{Skip}, sx)\):
have \((c_1,s) \rightarrow^* \langle \text{Skip},s' \rangle\) by \(\text{auto intro:converse-rtranclp-into-rtranclp}\)
with \(\langle c_2,sx \rangle \rightarrow^* \langle \text{Skip},s' \rangle\) show \(\text{thesis}\) by \(\text{auto}\)
qed

lemma \(\text{Cond-True-or-False}\):
\[(\text{if } (b) \text{ c}_1 \text{ else } c_2,s) \rightarrow^* \langle \text{Skip},s' \rangle \implies [b] s = \text{Some true} \lor [b] s = \text{Some false}\]
by \(\text{induct } c = = \text{if } (b) \text{ c}_1 \text{ else } c_2 s\text{ rule:converse-rtranclp-induct2}(\text{auto elim:red.cases})\)

lemma \(\text{CondFalse-reds}\):
\[(\text{if } (b) \text{ c}_1 \text{ else } c_2,s) \rightarrow^* \langle \text{Skip},s' \rangle \implies [b] s = \text{Some false} \implies (c_1,s) \rightarrow^* \langle \text{Skip},s' \rangle\]
by \(\text{induct } c = = \text{if } (b) \text{ c}_1 \text{ else } c_2 s\text{ rule:converse-rtranclp-induct2}(\text{auto elim:red.cases})\)

lemma \(\text{WhileFalse-reds}\):
\[(\text{while } (b) cx,s) \rightarrow^* \langle \text{Skip},s' \rangle \implies [b] s = \text{Some false} \implies s = s'\]
proof \(\text{induct while } (b) cx s\text{ rule:converse-rtranclp-induct2}\)
\(\text{case } \text{step }\) thus \(\text{?case}\) by \(\text{auto elim:red.cases dest: Skip-reds}\)
qed

lemma \(\text{WhileTrue-reds}\):
\[(\text{while } (b) cx,s) \rightarrow^* \langle \text{Skip},s' \rangle \implies [b] s = \text{Some true} \implies \exists sx. (cx,s) \rightarrow^* \langle \text{Skip},s' \rangle\]
proof \(\text{induct while } (b) cx s\text{ rule:converse-rtranclp-induct2}\)
\(\text{case } s' \text{ c''}\)
\(\text{hence } c'' = cx\); while \((b) cx \land s'' = s\) by \(\text{auto elim:red.cases}\)
with \(\langle c'',s'' \rangle \rightarrow^* \langle \text{Skip},s' \rangle\) show \(\text{?case}\) by \(\text{auto dest:Seq-reds}\)
qed

lemma \(\text{While-True-or-False}\):
\[(\text{while } (b) \text{ com},s) \rightarrow^* \langle \text{Skip},s' \rangle \implies [b] s = \text{Some true} \lor [b] s = \text{Some false}\]
by \(\text{induct } c = = \text{while } (b) \text{ com } s\text{ rule:converse-rtranclp-induct2}(\text{auto elim:red.cases})\)

inductive \text{red-n} :: \text{com }\Rightarrow \text{state }\Rightarrow \text{nat }\Rightarrow \text{com }\Rightarrow \text{state }\Rightarrow \text{bool}
\[
\langle 1(\cdot),\cdot \rangle \Rightarrow^* (\cdot,\cdot) \quad \text{[0,0,0,0,0]} \quad 81
\]
where \text{red-n-Base} : \langle c,s \rangle \rightarrow^* \langle c,s \rangle
| \text{red-n-Rec} : \langle c,s \rangle \rightarrow^* \langle c'',s'' \rangle ; \langle c'',s'' \rangle \rightarrow^n \langle c',s' \rangle \implies \langle c,s \rangle \rightarrow^n \langle c',s' \rangle

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

proof –
from $\langle c_1; c_2, s \rangle \rightarrow^n \langle \text{Skip}, s \rangle$
have $\exists i \cdot j \cdot s''. \langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \langle c_2, s'' \rangle \rightarrow^j \langle \text{Skip}, s \rangle \land n = i + j + 1$
proof(induct $c_1; c_2$ s n Skip $s'$ arbitrary; c rule:red-n.induct)
case (red-n-Rec $s \cdot c'' \cdot s''' \cdot n \cdot s'$)
  note IH = $\langle \land c_1, \cdot c'' = c_1; c_2 \land i \cdot j \cdot s \rangle$ 
  from $\langle c_1; c_2, s \rangle \rightarrow \langle c''', s''' \rangle$
  have $\langle c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \rangle \lor$
    $(\exists c_1'. \cdot c'' = c_1'; c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle)$
  by(induct $c_1; c_2$ - - rule:red-induct) auto
thus \{\case\}
proof
  assume $c_1 = \text{Skip} \land c'' = c_2 \land s = s''$
  hence $c_1 = \text{Skip}$ and $c'' = c_2$ and $s = s''$ by simp-all
  from $\langle c_1 = \text{Skip} \rangle$ have $\langle c_1, s \rangle \rightarrow^0 \langle \text{Skip}, s \rangle$ by(fastforce intro:red-n-Base)
  with $\langle c''', s''' \rangle \rightarrow^n \langle \text{Skip}, s'' \rangle$; $c'' = c_2$; $s = s''$
  show ?thesis by(rule-tac $x=0$ in exfI) auto
next
  assume $\exists c_1'. \cdot c'' = c_1'; c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle$
  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 $c'' = c_1'; c_2$] obtain $i \cdot j \cdot s$
    where $\langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, s \rangle$ and $\langle c_2, s \rangle \rightarrow^j \langle \text{Skip}, s \rangle$
    and $n = i + j + 1$ by blast
  from $\langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle$; $\langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, s \rangle$
  have $\langle c_1, s \rangle \rightarrow \langle c_2, s \rangle$ by(rule red-n.red-n.Rec)
  with $\langle c_2, s \rangle \rightarrow^j \langle \text{Skip}, s \rangle$; $n = i + j + 1$ show ?thesis
    by(rule-tac $x=\text{Suc} \cdot i$ in exfI) auto
  qed
qed
with that show ?thesis by blast
qed

lemma while-red-nE:
$\langle \text{while} \ (b) \ cx, s \rangle \rightarrow^n \langle \text{Skip}, s \rangle$
  $\Rightarrow \ (\{ b \} \ s = \text{Some false} \land s = s' \land n = 1) \lor$
    $(\exists i \cdot j \cdot s'''. \{ b \} \ s = \text{Some true} \land \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s''' \rangle \land$
      $\langle \text{while} \ (b) \ cx, s''' \rangle \rightarrow^j \langle \text{Skip}, s \rangle \land n = i + j + 2)$
proof(induct while (b) cx s Skip s' rule:red-n.induct)
case (red-n-Rec $s \cdot c'' \cdot s''' \cdot n \cdot s'$)
from $\langle \text{while} \ (b) \ cx, s \rangle \rightarrow \langle c''', s''' \rangle$
  have $\{ b \} \ s = \text{Some false} \land c'' = \text{Skip} \land s'' = s$ \lor
    $(\{ b \} \ s = \text{Some true} \land c'' = cx; while (b) \ cx \land s'' = s)$
  by(induct while (b) cx - - rule:red-induct) auto
thus \{\case\}
proof

7
assume \([b]\) \(s = \text{Some false} \land c'' = \text{Skip} \land s'' = s\)  

hence \([b]\) \(s = \text{Some false} \land c'' = \text{Skip} \land s'' = s\) by simp-all  

with \((c'',s'') \rightarrow^n (\text{Skip},s')\) have \(s = s'\) and \(n = 0\)  

by (induct - \(\text{- Skip}\) - rule:red-n.induct,auto elim:red.cases)  

with \([b]\) \(s = \text{Some false}\) show \(?thesis by fastforce\)

next  

assume \([b]\) \(s = \text{Some true} \land c'' = cx;;\text{while } (b) cx \land s'' = s\)  

hence \([b]\) \(s = \text{Some true} \land c'' = cx;;\text{while } (b) cx\)  

and \(s'' = s\) by simp-all  

with \((c'',s'') \rightarrow^n (\text{Skip},s')\) obtain \(i\) \(j\) \(x\) where \((cx,s) \rightarrow^i (\text{Skip},sx)\) and \((\text{while } (b) cx,sx) \rightarrow^j (\text{Skip},s')\)  

and \(n = i + j + 1\) by (fastforce elim:Seq-red-nE)  

with \([b]\) \(s = \text{Some true}\) show \(?thesis by fastforce\)

qed  

qed

\textbf{lemma while-red-n-induct [consumes 1, case-names false true]:}

assumes major: \((\text{while } (b) cx,s) \rightarrow^n (\text{Skip},s')\)  

and \(IHfalse:\forall s. \ [b]\ s = \text{Some false} \Longrightarrow P s s\)  

and \(IHtrue:\forall i\ j\ s'. \ [b]\ s = \text{Some true}; (\text{cx},s) \rightarrow^i (\text{Skip},s'');\)  

\((\text{while } (b) \text{cx},s'') \rightarrow^j (\text{Skip},s'); P s'' \Longrightarrow P s s'\)

shows \(P s s'\)  

using major

proof (induct \(n\) arbitrary:s rule: nat-less-induct)  

fix \(n\) \(s\)

assumption \(IHHall: \forall m < n. \forall x. (\text{while } (b) cx,x) \rightarrow^m (\text{Skip},s') \Longrightarrow P x s'\)

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

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

have \([b]\ s = \text{Some false} \land s = s' \land n = 1\) \lor

\((\exists i\ j\ s''. \ [b]\ s = \text{Some true}; (\text{cx},s) \rightarrow^i (\text{Skip},s'') \land

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

by (rule while-red-nE)

thus \(P s s'\)

proof

assumption \([b]\ s = \text{Some false} \land s = s' \land n = 1\)

hence \([b]\ s = \text{Some false} \land s = s'\) by auto

from \(IHfalse\) \((OF \ [b]\ s = \text{Some false})\) have \(P s s\).

with \((s = s')\) show \(?thesis by simp\)

next

assumption \(\exists i\ j\ s''. \ [b]\ s = \text{Some true} \land (\text{cx},s) \rightarrow^i (\text{Skip},s'') \land

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

then obtain \(i\ j\ s''\) where \([b]\ s = \text{Some true}\)

and \((\text{cx},s) \rightarrow^i (\text{Skip},s'')\) and \((\text{while } (b) \text{cx},s'') \rightarrow^j (\text{Skip},s')\)

and \(n = i + j + 2\) by blast

with \(IH Hall\) have \(P s'' s'\)

apply (crule-tac \(x=j\) in allE) apply clarsimp done

from \(IHtrue\) \((OF \ [b]\ s = \text{Some true}; (\text{cx},s) \rightarrow^i (\text{Skip},s'')\)
\langle \text{while} \ (b) \ cx,s' \rangle \rightarrow^j \langle \text{Skip},s' \rangle \text{ this} \ \textbf{show} \ \langle \text{?thesis} \rangle .

\textbf{qed}

\textbf{lemma} \ reds-to-red-n: \langle c,s \rangle \rightarrow^* \langle c',s' \rangle \quad \text{\textbf{implies}} \quad \exists \ n. \ \langle c,s \rangle \rightarrow^N \langle c',s' \rangle

\textbf{by}( \text{induct \ rule: converse-rtranclp-induct2, auto intro:red-n.intros} )

\textbf{lemma} \ red-n-to-reds: \langle c,s \rangle \rightarrow^N \langle c',s' \rangle \quad \text{\textbf{implies}} \quad \langle c,s \rangle \rightarrow^* \langle c',s' \rangle

\textbf{by}( \text{induct \ rule:red-n.induct, auto intro:converse-rtranclp-into-rtranclp} )

\textbf{lemma} \ while-reds-induct[consumes 1, case-names false true]:

\[[\text{while} \ (b) \ cx,s \rangle \rightarrow^* \langle \text{Skip},s' \rangle; \ \text{\text{\textbb{\Vdash}}}s. \ [b] \ s = \text{Some false} \rightarrow \ P \ s \ s;\]

\[\wedge \ s s''', \ [b] \ s = \text{Some true}; \ \langle cx,s \rangle \rightarrow^* \langle \text{Skip},s'' \rangle;\]

\[\langle \text{while} \ (b) \ cx,s'' \rangle \rightarrow^* \langle \text{Skip},s'' \rangle; \ P \ s'' s ] \rightarrow \ P \ s s]\n
\[\rightarrow \ P \ s s'.\]

\textbf{apply}( \text{drule \ reds-to-red-n, clarsimp})

\textbf{apply}( \text{erule \ while-red-n-induct, clarsimp})

\textbf{by}( \text{auto \ dest:red-n-to-reds} )

\textbf{lemma} \ red-det:

\[[\langle c,s \rangle \rightarrow \langle c_1,s_1 \rangle; \ \langle c,s \rangle \rightarrow \langle c_2,s_2 \rangle] \rightarrow c_1 = c_2 \ \wedge \ s_1 = s_2\]

\textbf{proof}( \text{induct arbitrary;c_2 \ rule:red-induct})

\textbf{case} (SeqRed \ c_1 \ s \ c_1' \ s' \ c_2')

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

\textbf{from} \ \langle c_1';c_2',s \rangle \rightarrow \langle c_2',s_2 \rangle \ \textbf{have} \ c_1 = \text{Skip} \ \vee \ \exists \ cx. \ c_2 = \text{cx};c_2' \ \wedge \ c_1 \rightarrow \langle \text{cx},s_2 \rangle \]

\textbf{by}( \text{fastforce \ elim:red.cases})

\textbf{thus} \ \langle \text{?case} \rangle

\textbf{proof}

\textbf{assume} \ c_1 = \text{Skip}

\textbf{with} \ \langle c_1,s \rangle \rightarrow \langle c_1',s' \rangle \ \textbf{have} \ \text{False} \ \textbf{by}( \text{fastforce \ elim:red.cases})

\textbf{thus} \ \langle \text{?thesis \ by \ simp} \rangle

\textbf{next}

\textbf{assume} \ \exists \ cx. \ c_2 = \text{cx};c_2' \ \wedge \ (c_1,s) \rightarrow \langle \text{cx},s_2 \rangle

\textbf{then obtain} \ cx \ \text{where} \ c_2 = \text{cx};c_2' \ \text{and} \ \langle c_1,s \rangle \rightarrow \langle \text{cx},s_2 \rangle \ \textbf{by } \text{blast}

\textbf{from} \ \text{IH}[OF \ \langle c_1,s \rangle \rightarrow \langle cx,s_2 \rangle] \ \textbf{have} \ c_1' = \text{cx} \ \wedge \ s' = s_2 .

\textbf{with} \ \langle c_2 = \text{cx};c_2' \ \text{show} \ \langle \text{?thesis \ by \ simp} \rangle \]

\textbf{qed}

\textbf{qed} \ ( \text{fastforce \ elim:red.cases})+

\textbf{theorem} \ reds-det:

\[[\langle c,s \rangle \rightarrow^* \langle \text{Skip},s_1 \rangle; \ \langle c,s \rangle \rightarrow^* \langle \text{Skip},s_2 \rangle] \rightarrow s_1 = s_2\]

\textbf{proof}( \text{induct \ rule:converse-rtranclp-induct2})

\textbf{case} refl
from \langle \text{Skip}, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \ show \ ?\text{case}
by \neg\text{erule converse-rtranclpE,auto elim:red\_cases)}

next
case (\text{step} \ c'' \ s'' \ c' \ s')
\text{note} \ IH = \langle (c',s') \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \Rightarrow s_1 = s_2 \rangle

from \text{step} \ have \langle c'',s'' \rangle \rightarrow \langle c',s' \rangle
\text{by simp}

from \langle c'',s'' \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \ \text{this have} \langle c',s' \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle
\text{by \neg\text{erule converse-rtranclpE,auto elim:red\_cases dest:red\_det)}

from IH[\text{OF this}] \ show \ ?\text{thesis} .
qed

datatype \text{secLevel} = \text{Low} | \text{High}

type-synonym \text{typeEnv} = \text{vname} \rightarrow \text{secLevel}

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

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

| \text{typeBinOp1}: [\Gamma \vdash e_1 : \text{Low}; \Gamma \vdash e_2 : \text{Low}] \Rightarrow \Gamma \vdash e_1 \langle bop \rangle e_2 : \text{Low}

| \text{typeBinOp2}: [\Gamma \vdash e_1 : \text{High}; \Gamma \vdash e_2 : \text{lev}] \Rightarrow \Gamma \vdash e_1 \langle bop \rangle e_2 : \text{High}

| \text{typeBinOp3}: [\Gamma \vdash e_1 : \text{lev}; \Gamma \vdash e_2 : \text{High}] \Rightarrow \Gamma \vdash e_1 \langle bop \rangle e_2 : \text{High}

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

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

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

| \text{typeSeq}: [\Gamma, T \vdash c_1; \Gamma, T \vdash c_2] \Rightarrow \Gamma, T \vdash c_1;c_2

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

2 Security types

2.1 Security definitions

datatype \text{secLevel} = \text{Low} \mid \text{High}

type-synonym \text{typeEnv} = \text{vname} \rightarrow \text{secLevel}

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

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

| \text{typeBinOp1}: [\Gamma \vdash e_1 : \text{Low}; \Gamma \vdash e_2 : \text{Low}] \Rightarrow \Gamma \vdash e_1 \langle bop \rangle e_2 : \text{Low}

| \text{typeBinOp2}: [\Gamma \vdash e_1 : \text{High}; \Gamma \vdash e_2 : \text{lev}] \Rightarrow \Gamma \vdash e_1 \langle bop \rangle e_2 : \text{High}

| \text{typeBinOp3}: [\Gamma \vdash e_1 : \text{lev}; \Gamma \vdash e_2 : \text{High}] \Rightarrow \Gamma \vdash e_1 \langle bop \rangle e_2 : \text{High}

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

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

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

| \text{typeSeq}: [\Gamma, T \vdash c_1; \Gamma, T \vdash c_2] \Rightarrow \Gamma, T \vdash c_1;c_2

| \text{typeWhile}: [\Gamma \vdash b : T; \Gamma, T \vdash c] \Rightarrow \Gamma, T \vdash \text{while} (b) c
\[
\text{typeIf: } \quad \Gamma \vdash b : T; \Gamma, T \vdash c1; \Gamma, T \vdash c2 \quad \implies \quad \Gamma, T \vdash \text{if } (b) \text{ c1 else c2}
\]

\[
\text{typeConvert: } \Gamma, \text{High} \vdash c \quad \implies \quad \Gamma, \text{Low} \vdash c
\]

### 2.2 Lemmas concerning expressions

**lemma exprTypeable:**

assumes \( fv e \subseteq \text{dom } \Gamma \) obtains \( T \) where \( \Gamma \vdash e : T \)

**proof**

- from \( \langle fv e \subseteq \text{dom } \Gamma \rangle \) have \( \exists T. \Gamma \vdash e : T \)

**proof**

- \( \text{induct e} \)

  - case \( \text{Val V} \)
    - have \( \Gamma \vdash \text{Val V} : \text{Low} \) by \( \text{rule typeVal} \)
    - thus \( ?\text{case by (rule exI)} \)
  
  - case \( \text{Var V} \)
    - have \( V \in \text{fv}(\text{Var V}) \) by \( \text{simp} \)
    - with \( \langle \text{fv}(\text{Var V}) \subseteq \text{dom } \Gamma \rangle \) have \( V \in \text{dom } \Gamma \) by \( \text{simp} \)
    - then obtain \( T \) where \( \Gamma \vdash \text{Var V} : T \) by \( \text{auto} \)
    - hence \( \Gamma \vdash \text{Var V} : T \) by \( \text{rule typeVar} \)
    - thus \( ?\text{case by (rule exI)} \)

- case \( \text{BinOp e1 bop e2} \)

  - note \( \text{IH1} = \langle fv e1 \subseteq \text{dom } \Gamma \implies \exists T. \Gamma \vdash e1 : T \rangle \)
  
  - note \( \text{IH2} = \langle fv e2 \subseteq \text{dom } \Gamma \implies \exists T. \Gamma \vdash e2 : T \rangle \)

  - from \( \langle \text{fv}(\text{e1} \ll bop \gg \text{e2}) \subseteq \text{dom } \Gamma \rangle \) have \( \langle \Gamma \vdash e1 \ll bop \gg \text{e2} : \text{Low} \rangle \) by \( \text{auto} \)

  - show \( ?\text{case} \)

**proof**

- \( \text{cases T1} \)

  - case \( \text{Low} \)
    - show \( ?\text{thesis} \)

  - proof \( \text{(cases T2)} \)

    - case \( \text{Low} \)
      - with \( \langle \Gamma \vdash e1 : T1; \Gamma \vdash e2 : T2 \rangle (T1 = \text{Low}) \)
      - have \( \Gamma \vdash e1 \ll bop \gg e2 : \text{Low} \) by \( \text{(simp add: typeBinOp1)} \)
      
    - thus \( ?\text{thesis by (rule exI)} \)

    - case \( \text{High} \)
      - with \( \langle \Gamma \vdash e1 : T1; \Gamma \vdash e2 : T2 \rangle (T1 = \text{Low}) \)
      - have \( \Gamma \vdash e1 \ll bop \gg e2 : \text{High} \) by \( \text{(simp add: typeBinOp3)} \)
      
    - thus \( ?\text{thesis by (rule exI)} \)

**qed**

next

- case \( \text{High} \)

  - with \( \langle \Gamma \vdash e1 : T1; \Gamma \vdash e2 : T2 \rangle \)
  
  - have \( \Gamma \vdash e1 \ll bop \gg e2 : \text{High} \) by \( \text{(simp add: typeBinOp2)} \)
thus ?thesis by (rule exI)
qed
qed
with that show ?thesis by blast
qed

lemma exprBinopTypeable:
assumes \( \Gamma \vdash e_1 \langle \text{bop} \rangle e_2 : T \)
shows \( \exists T_1. \Gamma \vdash e_1 : T_1 \land (\exists T_2. \Gamma \vdash e_2 : T_2) \)
using assms by (auto elim:secExprTyping.cases)

lemma exprTypingHigh:
assumes \( \Gamma \vdash e : T \) and \( x \in \text{fv } e \) and \( \Gamma \vdash x = \text{Some High} \)
shows \( \Gamma \vdash e : \text{High} \)
using assms
proof (induct e arbitrary: \( T \))
case (Val V) show ?case by (rule typeVal)
next
case (Var V)
from \( x \in \text{fv } \text{(Var V)} \) have \( x = V \) by simp
with \( \Gamma \vdash x = \text{Some High} \) show ?case by (simp add: typeVar)
next
case (BinOp e1 bop e2)
note IH1 = \( \langle \forall T. [\Gamma \vdash e_1 : T; x \in \text{fv } e_1; \Gamma \vdash x = \text{Some High}] \implies \Gamma \vdash e_1 : \text{High} \rangle \)
note IH2 = \( \langle \forall T. [\Gamma \vdash e_2 : T; x \in \text{fv } e_2; \Gamma \vdash x = \text{Some High}] \implies \Gamma \vdash e_2 : \text{High} \rangle \)
from \( \Gamma \vdash e_1 \langle \text{bop} \rangle e_2 : T \) have \( T:(\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2) \) by (auto intro!:exprBinopTypeable)
then obtain T1 where \( \Gamma \vdash e_1 : T_1 \) by auto
from T obtain T2 where \( \Gamma \vdash e_2 : T_2 \) by auto
from \( x \in \text{fv } (e_1 \langle \text{bop} \rangle e_2) \) have \( x \in (\text{fv } e_1 \cup \text{fv } e_2) \) by simp
hence \( x \in \text{fv } e_1 \lor x \in \text{fv } e_2 \) by auto
thus ?case
proof
assume \( x \in \text{fv } e_1 \)
from IH1[OF \( \Gamma \vdash e_1 : T_1 \) this \( \Gamma \vdash x = \text{Some High} \)] have \( \Gamma \vdash e_1 : \text{High} \).
with \( \Gamma \vdash e_2 : T_2 \) show ?thesis by (simp add: typeBinOp2)
next
assume \( x \in \text{fv } e_2 \)
from IH2[OF \( \Gamma \vdash e_2 : T_2 \) this \( \Gamma \vdash x = \text{Some High} \)] have \( \Gamma \vdash e_2 : \text{High} \).
with \( \Gamma \vdash e_1 : T_1 \) show ?thesis by (simp add: typeBinOp3)
qed
qed

lemma exprTypingLow:
assumes \( \Gamma \vdash e : \text{Low} \) and \( x \in \text{fv } e \) shows \( \Gamma \vdash x = \text{Some Low} \)
using assms

proof (induct e)
  case (Val V)
  have \( \text{fv} (\text{Val} V) = \{\} \) by (rule FVc)
  with \( \langle x \in \text{fv} (\text{Val} V): \text{have} \ \text{False} \ \text{by} \ \text{auto} \rangle \)
  thus ?thesis by simp
next
  case (Var V)
  have \( x \in \text{fv} (\text{Var} V): \text{have} \ \text{False} \ \text{by} \ \text{auto} \)
  thus ?thesis by simp
  qed

next
  case (BinOp e1 bop e2)
  note IH1 = \( \langle \forall T. \text{\textbf{\Gamma} } \vdash e1 : T \Rightarrow \text{\textbf{\Gamma} } x = \text{Some Low} \rangle \)
  note IH2 = \( \langle \forall T. \text{\textbf{\Gamma} } \vdash e2 : T \Rightarrow \text{\textbf{\Gamma} } x = \text{Some Low} \rangle \)
  have \( \langle \text{\textbf{\Gamma} } \vdash e1 \ll bop \gg e2 : \text{Low} \rangle \)
  by (auto elim: secExprTyping.cases)
  note IH3 = \( \langle \forall T. \text{\textbf{\Gamma} } \vdash e1 \ll bop \gg e2 : \text{Low} \Rightarrow \text{\textbf{\Gamma} } x = \text{Some Low} \rangle \)
  with IH1 IH2 show ?thesis by simp

lemma typeableFreevars:
  assumes \( \text{\textbf{\Gamma} } \vdash e : T \)
  shows \( \text{fv} e \subseteq \text{dom} \ \text{\textbf{\Gamma} } \)
using assms

proof (induct e arbitrary: T)
  case (Val V)
  have \( \text{fv} (\text{Val} V) = \{\} \) by (rule FVc)
  thus ?case by simp
next
  case (Var V)
  show ?case
  proof
    fix x assume \( x \in \text{fv} (\text{Var} V) \)
    hence \( x = V \) by simp
    from \( \text{\textbf{\Gamma} } \vdash \text{Var} V : T \)
    have \( \text{\textbf{\Gamma} } V = \text{Some T} \) by (auto elim: secExprTyping.cases)
    with \( x = V \) show \( x \in \text{dom} \ \text{\textbf{\Gamma} } \) by auto
    qed
next
  case (BinOp e1 bop e2)
  note IH1 = \( \langle \forall T. \text{\textbf{\Gamma} } \vdash e1 : T \Rightarrow \text{fv} e1 \subseteq \text{dom} \ \text{\textbf{\Gamma} } \rangle \)
note IH2 = \{\bigwedge T. \Gamma \vdash e_2 : T \implies \text{fv } e_2 \subseteq \text{dom } \Gamma\}
show \ ?case
proof
  fix x assume x \in \text{fv } (e_1 \ <\text{bop}> \ e_2)
  from \Gamma \vdash e_1 \ <\text{bop}> \ e_2 : T;
  have Q: (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)
    by (rule exprBinopTypeable)
then obtain T_1 where \Gamma \vdash e_1 : T_1 by blast
from Q obtain T_2 where \Gamma \vdash e_2 : T_2 by blast
from IH1[OF \Gamma \vdash e_1 : T_1] have \text{fv } e_1 \subseteq \text{dom } \Gamma .
moreover from IH2[OF \Gamma \vdash e_2 : T_2] have \text{fv } e_2 \subseteq \text{dom } \Gamma .
ultimately have \text{fv } (e_1 \ <\text{bop}> \ e_2) \subseteq \text{dom } \Gamma by auto
hence \text{fv } (e_1 \ <\text{bop}> \ e_2) \subseteq \text{dom } \Gamma by (simp add: FVe)
with \langle x \in \text{fv } (e_1 \ <\text{bop}> \ e_2) \rangle show \ x \in \text{dom } \Gamma by auto
qed
qed

lemma exprNotNone:
assumes \Gamma \vdash e : T and \text{fv } e \subseteq \text{dom } s
shows [e] s \not= None
using assms
proof (induct e arbitrary: \Gamma T s)
  case (Val v)
  show \ ?case by (simp add: Val)
next
  case (Var V)
  have [\text{Var } V] s = s V by (simp add: Var)
  have V \in \text{fv } (\text{Var } V) by (auto simp add: FVe)
  with [\text{fv } (\text{Var } V)] \subseteq \text{dom } s have V \in \text{dom } s by (simp add: FVe)
thus \ ?case by auto
next
  case (BinOp e_1 \text{ bop } e_2)
  note IH1 = \{\bigwedge T. \Gamma \vdash e_1 : T; \text{fv } e_1 \subseteq \text{dom } s \} \implies [e_1] s \not= None
  note IH2 = \{\bigwedge T. \Gamma \vdash e_2 : T; \text{fv } e_2 \subseteq \text{dom } s \} \implies [e_2] s \not= None
from \Gamma \vdash e_1 \ <\text{bop}> \ e_2 : T have \langle \exists T_1. \Gamma \vdash e_1 : T_1 \rangle \land \langle \exists T_2. \Gamma \vdash e_2 : T_2 \rangle
  by (rule exprBinopTypeable)
then obtain T_1 T_2 where \Gamma \vdash e_1 : T_1 and \Gamma \vdash e_2 : T_2 by blast
from [\text{fv } (e_1 \ <\text{bop}> \ e_2)] \subseteq \text{dom } s have \text{fv } e_1 \cup \text{fv } e_2 \subseteq \text{dom } s by (simp add: FVe)
hence \text{fv } e_1 \subseteq \text{dom } s and \text{fv } e_2 \subseteq \text{dom } s by auto
from IH1[OF \Gamma \vdash e_1 : T_1; \text{fv } e_1 \subseteq \text{dom } s] have [e_1] s \not= None .
moreover from IH2[OF \Gamma \vdash e_2 : T_2; \text{fv } e_2 \subseteq \text{dom } s] have [e_2] s \not= None .
ultimately show \ ?case
  apply (cases \text{bop}) apply auto
  apply (case_tac y, auto, case_tac ya, auto)+
done
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. $\in \text{dom state}$), 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** \( \text{lowEquiv} :: \text{typeEnv} \Rightarrow \text{state} \Rightarrow \text{state} \Rightarrow \text{bool} \)

\[
\Gamma \vdash s_1 \approx_L s_2 \equiv \forall v \in \text{dom } \Gamma. \Gamma v = \text{Some Low} \rightarrow (s_1 v = s_2 v)
\]

**Lemma** \( \text{lowEquivReflexive} : \Gamma \vdash s_1 \approx_L s_1 \)

by (simp add: lowEquiv-def)

**Lemma** \( \text{lowEquivSymmetric} : \Gamma \vdash s_1 \approx_L s_2 \Rightarrow \Gamma \vdash s_2 \approx_L s_1 \)

by (simp add: lowEquiv-def)

**Lemma** \( \text{lowEquivTransitive} : \left[ [\Gamma \vdash s_1 \approx_L s_2; \Gamma \vdash s_2 \approx_L s_3] \Rightarrow \Gamma \vdash s_1 \approx_L s_3 \right] \)

by (simp add: lowEquiv-def)

2.3.2 Non Interference

**Definition** \( \text{nonInterference} :: \text{typeEnv} \Rightarrow \text{com} \Rightarrow \text{bool} \)

\[
\text{nonInterference } \Gamma c \equiv \left( \forall s_1 s_2 s_1' s_2'. (\Gamma \vdash s_1 \approx_L s_2 \land \langle c, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle \land \langle c, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle) \right)
\]

\[
\Rightarrow \Gamma \vdash s_1' \approx_L s_2'
\]

**Lemma** \( \text{nonInterferenceI} : \left[ \Gamma \vdash s_1 \approx_L s_2; \Gamma \vdash s_1 \approx_L s_2; \langle c, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle; \langle c, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle \right] \Rightarrow \Gamma \vdash s_1' \approx_L s_2' \)

by (auto simp: nonInterference-def)

**Lemma** \( \text{interpretLow} : \)

assumes \( \Gamma \vdash s_1 \approx_L s_2 \) and \( \forall V \in \text{fv } e. \Gamma V = \text{Some Low} \)

shows \( \left[ e \right] s_1 = \left[ e \right] s_2 \)

using all

proof (induct e)

case (Val v)

show \( \text{case} \) by (simp add: Val)

next

case (Var V)
have \([\text{Var } V] s_1 = s_1 V \text{ and } [\text{Var } V] s_2 = s_2 V\) \(\text{by (auto simp: Var)}\)

have \(V \in \text{fv } (\text{Var } V)\) \(\text{by (simp add: FV v)}\)

from \((V \in \text{fv } (\text{Var } V)). \forall X \in \text{fv } (\text{Var } V). \Gamma X = \text{Some Low} \) have \(\Gamma V = \text{Some Low}\) \(\text{by simp}\)

with \(\text{assms have } s_1 V = s_2 V\) \(\text{by (auto simp add: lowEquiv-def)}\)

next

\text{case } (\text{BinOp } e_1 \text{ bop } e_2)

\text{note } IH1 = \(\forall V \in \text{fv } e_1. \Gamma V = \text{Some Low} \Rightarrow [e_1] s_1 = [e_1] s_2\)

\text{note } IH2 = \(\forall V \in \text{fv } e_2. \Gamma V = \text{Some Low} \Rightarrow [e_2] s_1 = [e_2] s_2\)

from \((\forall V \in \text{fv } (e_1 <\text{bop}> e_2). \Gamma V = \text{Some Low} )\) have \(\forall V \in \text{fv } e_1. \Gamma V = \text{Some Low}\) \(\text{by (auto simp add: lowEquiv-def)}\)

moreover

from \(IH2[\text{OF } \forall V \in \text{fv } e_2. \Gamma V = \text{Some Low}]\) have \([e_2] s_1 = [e_2] s_2\).

ultimately show \(\text{?case by (cases } [e_1] s_2, \text{auto)}\)

qed

\text{lemma } interpretLow2:

\text{assumes } \(\Gamma \vdash e : \text{Low} \text{ and } \Gamma \vdash s_1 \approx_{\text{L}} s_2 \text{ shows } [e] s_1 = [e] s_2\)

\text{proof –}

from \(\Gamma \vdash e : \text{Low}\) have \(\text{fv } e \subseteq \text{dom } \Gamma\) \(\text{by (auto dest: typeableFreevars)}\)

have \(\forall x \in \text{fv } e. \Gamma x = \text{Some Low}\)

\text{proof}

\(\text{fix } x \text{ assume } x \in \text{fv } e\)

with \(\Gamma \vdash e : \text{Low}\) show \(\Gamma x = \text{Some Low}\) \(\text{by (auto intro: exprTypingLow)}\)

qed

with \(\Gamma \vdash s_1 \approx_{\text{L}} s_2\) show \(\text{?thesis by (rule interpretLow)}\)

qed

\text{lemma } assignNIhighlemma:

\text{assumes } \(\Gamma \vdash s_1 \approx_{\text{L}} s_2\) \(\text{ and } \Gamma V = \text{Some High and } s_1 = s_1(\text{V := } [e] s_1)\)

\text{and } \(s_2' = s_2(\text{V := } [e] s_2)\)

\text{shows } \(\Gamma \vdash s_1' \approx_{\text{L}} s_2'\)

\text{proof –}

\(\{ \text{fix } V' \text{ assume } V' \in \text{dom } \Gamma \text{ and } \Gamma V' = \text{Some Low} \}

\text{from } \(\Gamma \vdash s_1 \approx_{\text{L}} s_2\) \(\Gamma V' = \text{Some Low} \) have \(s_1 V' = s_2 V'\)

\text{by (auto simp add: lowEquiv-def)}

\text{have } \(s_1' V' = s_2' V'\)

\text{proof (cases } V' = V\)

\text{case } True

\text{with } \(\Gamma V' = \text{Some Low}\) \(\Gamma V = \text{Some High}\) have \(\text{False by simp}\)

\text{thus } \text{?thesis by simp}\n
next

\text{case } False

\text{with } \(s_1' = s_1(\text{V := } [e] s_1)\) \(s_2' = s_2(\text{V := } [e] s_2)\)
proof
−

theorem SeqCompositionality
compositionality is no longer valid in case of concurrency

lemma assignNIlowlemma:
assumes \( \Gamma \vdash s_1 \approx_L s_2 \) and \( \Gamma V = \text{Some Low} \) and \( \Gamma \vdash e : \text{Low} \) and \( s_1' = s_1(V := [e] s_1) \) and \( s_2' = s_2(V := [e] s_2) \)
shows \( \Gamma \vdash s_1' \approx_L s_2' \)
proof
\begin{align*}
\{ & \text{fix } V' \text{ assume } V' \in \text{dom } \Gamma \text{ and } \Gamma V' = \text{Some Low} \\
& \text{from } \Gamma \vdash s_1 \approx_L s_2 \text{ } (\Gamma V' = \text{Some Low}) \\
& \text{have } s_1 V' = s_2 V' \text{ by (auto simp add:lowEquiv-def)} \\
& \text{have } s_1' V' = s_2' V' \\
\end{align*}
proof(cases V' = V)

next

case True

with \( (s_1' = s_1(V := [e] s_1)) \) \( (s_2' = s_2(V := [e] s_2)) \)

have \( s_1' V' = [e] s_1 \) \( \text{ and } s_2' V' = [e] s_2 \) by auto

from \( \Gamma \vdash e : \text{Low} \) \( \Gamma \vdash s_1 \approx_L s_2 \) have \([e] s_1 = [e] s_2 \)

by (auto intro:interpretLow2)

with \( (s_1' V' = [e] s_1) \) \( (s_2' V' = [e] s_2) \) show \( \text{thesis by simp} \)

next

case False

with \( (s_1' = s_1(V := [e] s_1)) \) \( (s_2' = s_2(V := [e] s_2)) \)

have \( s_1' V' = s_1 V' \) \( \text{ and } s_2' V' = s_2 V' \) by auto

with \( (s_1 V' = s_2 V') \) \( \text{ have } s_1' V' = s_2' V' \) by simp

with \( (s_1' V' = s_2 V') \) \( (s_2' V' = s_2 V') \) show \( \text{thesis by auto} \)

qed

\begin{align*}
\text{thus } & \text{thesis by (simp add:lowEquiv-def)} \\
\text{qed}
\end{align*}

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

theorem SeqCompositionality:
assumes nonInterference \( \Gamma c_1 \) and nonInterference \( \Gamma c_2 \)
shows nonInterference \( \Gamma (c_1;c_2) \)
proof(rule nonInterferenceI)
fix \( s_1 s_2 s_1' s_2' \)
assume \( \Gamma \vdash s_1 \approx_L s_2 \) and \( (c_1;c_2,s_1) \rightarrow* (\text{Skip},s_1') \)
and \( (c_1;c_2,s_2) \rightarrow* (\text{Skip},s_2') \)
from \( (c_1;c_2,s_1) \rightarrow* (\text{Skip},s_1') \) obtain \( s_1'' \) where \( (c_1,s_1) \rightarrow* (\text{Skip},s_1'') \)
and \( (c_2,s_1'') \rightarrow* (\text{Skip},s_1') \) by (auto dest:Seq-reds)
from \( (c_1;c_2,s_2) \rightarrow* (\text{Skip},s_2') \) obtain \( s_2'' \) where \( (c_1,s_2) \rightarrow* (\text{Skip},s_2'') \)
and } ⟨c2,s2''⟩ ⟷ ⟹ ⟨(Skip,s2'')⟩ by (auto dest:Seg-reds)
from } Γ ⊢ s1 ≈L s2 : ⟨⟨c1,s1⟩ ⟷ ⟹ ⟨(Skip,s1'')⟩ : ⟨⟨c1,s2⟩ ⟷ ⟹ ⟨(Skip,s2'')⟩)

nonInterference } Γ c1

have } Γ ⊢ s1'' ≈L s2'' by (auto simp:nonInterference-def)
with } ⟨⟨c2,s1''⟩ ⟷ ⟹ ⟨(Skip,s1'')⟩ : ⟨⟨c2,s2''⟩ ⟷ ⟹ ⟨(Skip,s2'')⟩ ; (nonInterference } Γ c2

show } Γ ⊢ s1' ≈L s2' by (auto simp:nonInterference-def)

qed

lemma WhileStepInduct:

assumes } while:⟨while (b) c,s1⟩ ⟷ ⟹ ⟨(Skip,s2)
and } body:⟨s2 ,⟨c,s1⟩ ⟷ ⟹ ⟨(Skip,s2)⟩ ⟷ ⟹ Γ ⊢ s1 ≈L s2 and } Γ,High ⊢ c

shows } Γ ⊢ s1 ≈L s2

using while proof (induct rule:while-reds-induct)
case } (false s) thus } thesis by (auto simp add:lowEquiv-def)
next
case } (true s1 s2)
from } body:⟨OF (⟨c,s1⟩ ⟷ ⟹ ⟨(Skip,s3)⟩)⟩ have } Γ ⊢ s1 ≈L s3 by simp
with } (Γ ⊢ s3 ≈L s2) show } thesis 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 } Γ,High ⊢ c and } ⟨c,s1⟩ ⟷ ⟹ ⟨(Skip,s2)⟩

shows } Γ ⊢ s1 ≈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 } ⟨(Skip,s1) ⟷ ⟹ ⟨(Skip,s2)⟩ have } s1 = s2 by (auto dest:Skip-reds)

thus } thesis by (simp add:lowEquiv-def)
next
case } (LAss V e)
from } Γ,High ⊢ V := e have } Γ V = Some High by (auto elim:secComTyping.cases)
from } ⟨V := e,s1⟩ ⟷ ⟹ ⟨(Skip,s2,s2)⟩ have } s2 = s1(V := [e]s1) by (auto intro:LAss-reds)
{ fix } V' assume } V' ∈ dom } Γ and } Γ V' = Some Low

have } s1 V' = s2 V'
proof (cases V' = V)
case } True
with } Γ V' = Some Low } Γ V = Some High have } False by simp

thus } thesis by simp
next
case } False
with } (s2 = s1(V := [e]s1)) show } thesis by simp


qed
}
thus ?case by(auto simp add:lowEquiv-def)
next
case (Seq c1 c2)
note IH1 = \(\{s1 \Rightarrow \{c1,s1\} \mapsto \{\text{Skip},s2\}\} \Rightarrow s1 \approx_L s2\)
note IH2 = \(\{s1 \Rightarrow \{c2,s1\} \mapsto \{\text{Skip},s2\}\} \Rightarrow s1 \approx_L s2\)
from \(\Gamma,\text{High} \vdash c1; c2\) have \(\Gamma,\text{High} \vdash c1\) and \(\Gamma,\text{High} \vdash c2\)
  by(auto elim:secComTyping.cases)
from \(\{c1;c2,s1\} \mapsto \{\text{Skip},s2\}\)
have \(\exists s3. (c1,s1) \mapsto \{\text{Skip},s3\} \land (c2,s3) \mapsto \{\text{Skip},s2\}\) by(auto intro:Seq-reds)
then obtain s3 where \((c1,s1) \mapsto \{\text{Skip},s3\}\) and \((c2,s3) \mapsto \{\text{Skip},s2\}\) by auto
from IH1[OF \(\Gamma,\text{High} \vdash c1\)] \[(c1,s1) \mapsto \{\text{Skip},s3\}\]
have \(\Gamma \vdash s1 \approx_L s3\) by simp
from IH2[OF \(\Gamma,\text{High} \vdash c2\)] \[(c2,s3) \mapsto \{\text{Skip},s2\}\]
have \(\Gamma \vdash s3 \approx_L s2\) by simp
from \(\Gamma \vdash s1 \approx_L s3\); \(\Gamma \vdash s3 \approx_L s2\) show ?case
  by(auto intro:LowEquivTransitive)
next
case (Cond b c1 c2)
note IH1 = \(\{s1 \Rightarrow \{c1,s1\} \mapsto \{\text{Skip},s2\}\} \Rightarrow s1 \approx_L s2\)
note IH2 = \(\{s1 \Rightarrow \{c2,s1\} \mapsto \{\text{Skip},s2\}\} \Rightarrow s1 \approx_L s2\)
from \(\Gamma,\text{High} \vdash \text{if } (b) \text{ c1 else } c2\) have \(\Gamma,\text{High} \vdash c1\) and \(\Gamma,\text{High} \vdash c2\)
  by(auto elim:secComTyping.cases)
from \(\{\text{if } (b) \text{ c1 else } c2,s1\} \mapsto \{\text{Skip},s2\}\)
have \([b],s1 = \text{Some true} \lor [b],s1 = \text{Some false}\) by(auto dest:Cond-True-or-False)
thus ?case
proof
  assume \([b],s1 = \text{Some true}\)
  with \(\{\text{if } (b) \text{ c1 else } c2,s1\} \mapsto \{\text{Skip},s2\}\) have \(\{c1,s1\} \mapsto \{\text{Skip},s2\}\)
    by(auto intro:CondTrue-reds)
  from IH1[OF \(\Gamma,\text{High} \vdash c1\) \(\text{this}\)] show ?thesis .
next
  assume \([b],s1 = \text{Some false}\)
  with \(\{\text{if } (b) \text{ c1 else } c2,s1\} \mapsto \{\text{Skip},s2\}\) have \(\{c2,s1\} \mapsto \{\text{Skip},s2\}\)
    by(auto intro:CondFalse-reds)
  from IH2[OF \(\Gamma,\text{High} \vdash c2\) \(\text{this}\)] show ?thesis .
qed
next
case (While b c)
note IH = \(\{s1 \Rightarrow \{c',s1\} \mapsto \{\text{Skip},s2\}\} \Rightarrow s1 \approx_L s2\)
from \(\Gamma,\text{High} \vdash \text{while } (b) \text{ c'}\) have \(\Gamma,\text{High} \vdash c'\) by(auto elim:secComTyping.cases)
from IH[OF this]
have \(\{s1 \Rightarrow \{c',s1\} \mapsto \{\text{Skip},s2\}\} \Rightarrow s1 \approx_L s2\).
with \(\{\text{while } (b) \text{ c'},s1\} \mapsto \{\text{Skip},s2\}\) \(\Gamma,\text{High} \vdash c'\)
show ?case by(auto dest:WhileStepInduct)
qed
lemma CondHighCompositionality:
assumes \( \Gamma', \text{High} \vdash \text{if } (b) \ c1 \text{ else } c2 \)
shows \( \text{nonInterference } \Gamma \ (if \ (b) \ c1 \text{ else } c2) \)
proof (rule nonInterferenceI)
  fix \( s1 \ s2 \ s1' \ s2' \)
  assume \( \Gamma \vdash s1 \approx_L s2 \) and \(\langle (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle\)
and \(\langle (if \ (b) \ c1 \text{ else } c2, s2) \rightarrow^* \langle \text{Skip}, s2' \rangle\)
show \( \Gamma \vdash s1' \approx_L s2' \)
proof
  from \( \Gamma, \text{High} \vdash \text{if } (b) \ c1 \text{ else } c2; \langle (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle\)
  have \( s1 \approx_L s1' \) by (auto dest: highBodies)
  from \( \Gamma, \text{High} \vdash \text{if } (b) \ c1 \text{ else } c2; \langle (if \ (b) \ c1 \text{ else } c2, s2) \rightarrow^* \langle \text{Skip}, s2' \rangle\)
  have \( s2 \approx_L s2' \) by (auto dest: highBodies)
  with \( \Gamma \vdash s1 \approx_L s2 \)
  have \( \Gamma \vdash s1 \approx_L s2' \) by (auto intro: lowEquivTransitive)
  from \( \Gamma \vdash s1 \approx_L s2 \)
  have \( \Gamma \vdash s1 \approx_L s1' \) by (auto intro: lowEquivSymmetric)
  with \( \Gamma \vdash s1 \approx_L s2' \)
  show \( \text{thesis} \) by (auto intro: lowEquivSymmetric)
qed

lemma CondLowCompositionality:
assumes \( \text{nonInterference } \Gamma \ c1 \) and \( \text{nonInterference } \Gamma \ c2 \) and \( \Gamma \vdash b : \text{Low} \)
shows \( \text{nonInterference } \Gamma \ (if \ (b) \ c1 \text{ else } c2) \)
proof (rule nonInterferenceI)
  fix \( s1 \ s2 \ s1' \ s2' \)
  assume \( \Gamma \vdash s1 \approx_L s2 \) and \(\langle (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle\)
and \(\langle (if \ (b) \ c1 \text{ else } c2, s2) \rightarrow^* \langle \text{Skip}, s2' \rangle\)
from \( \Gamma \vdash b : \text{Low} \)
  have \( [b] \ s1 = [b] \ s2 \) by (auto intro: interpretLow2)
  from \(\langle (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle\)
  have \( [b] \ s1 = \text{Some false} \) by (auto dest: Cond-True-or-False)
  thus \( \Gamma \vdash s1' \approx_L s2' \)
proof
  assume \( [b] \ s1 = \text{Some true} \)
with \( \langle [b] \ s1 = [b] \ s2 \rangle \)
  have \( [b] \ s2 = \text{Some true} \) by (auto intro: CondTrue-reds)
from \( \langle [b] \ s1 = \text{Some true} \rangle \)
  have \( \langle (c1, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle \) by (auto intro: CondTrue-reds)
from \( \langle [b] \ s2 = \text{Some true} \rangle \)
  have \( \langle (c1, s2) \rightarrow^* \langle \text{Skip}, s2' \rangle \) by (auto intro: CondTrue-reds)
with \(\langle (c1, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle, \Gamma \vdash s1 \approx_L s2, \text{nonInterference } \Gamma \ c1 \)
  show \( \text{thesis} \) by (auto simp: nonInterference_def)
next
  assume \( [b] \ s1 = \text{Some false} \)
with \( \langle [b] \ s1 = [b] \ s2 \rangle \)
  have \( [b] \ s2 = \text{Some false} \) by (auto intro: CondTrue-reds)
from \( \langle [b] \ s1 = \text{Some false} \rangle \)
  have \( \langle (c1, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle \) by (auto intro: CondFalse-reds)
from \( \langle [b] \ s2 = \text{Some false} \rangle \)
  have \( \langle (c2, s2) \rightarrow^* \langle \text{Skip}, s2' \rangle \) by (auto intro: CondFalse-reds)
from \( \langle [b] \ s2 = \text{Some false} \rangle \)
  have \( \langle (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow^* \langle \text{Skip}, s1' \rangle \)
lemma WhileHighCompositionality:
assumes Γ, High ⊢ while (b) c'
shows nonInterference Γ (while (b) c')
proof (rule nonInterferenceI)
fix s1 s2 s1' s2'
assume Γ ⊢ s1 ≈ _L s2 and ⟨while (b) c',s1⟩ →* ⟨Skip,s1'⟩
and ⟨while (b) c',s2⟩ →* ⟨Skip,s2'⟩
show Γ ⊢ s1' ≈ _L s2'
proof –
from Γ, High ⊢ while (b) c' ⟹ ⟨while (b) c',s1⟩ →* ⟨Skip,s1'⟩
have Γ ⊢ s1 ≈ _L s1' by (auto dest:highBodies)
from Γ, High ⊢ while (b) c' ⟹ ⟨while (b) c',s2⟩ →* ⟨Skip,s2'⟩
have Γ ⊢ s2 ≈ _L s2' by (auto dest:highBodies)
with Γ ⊢ s1 ≈ _L s2 have Γ ⊢ s1 ≈ _L s2' by (auto intro:lowEquivTransitive)
from Γ ⊢ s1 ≈ _L s1' have Γ ⊢ s1' ≈ _L s1 by (auto intro:lowEquivSymmetric)
with Γ ⊢ s1 ≈ _L s2' show ?thesis by (auto intro:lowEquivTransitive)
qed

lemma WhileLowStepInduct:
assumes while1: ⟨while (b) c',s1⟩ →* ⟨Skip,s1'⟩
and while2: ⟨while (b) c',s2⟩ →* ⟨Skip,s2'⟩
and b : Low
and body: ⟨b⟩⟨s1 s1' s2 s2'⟩ ⟹ ⟨⟨c',s1⟩ →* ⟨Skip,s1'⟩; ⟨c',s2⟩ →* ⟨Skip,s2'⟩; Γ ⊢ s1 ≈ _L s2⟩ ⟹ Γ ⊢ s1' ≈ _L s2'
and le: Γ ⊢ s1 ≈ _L s2
shows Γ ⊢ s1' ≈ _L s2'
using while1 le while2
proof (induct arbitrary:s2 rule:while-reds-induct)
case (false s1)
from Γ ⊢ b : Low; Γ ⊢ s1 ≈ _L s2; have [b] s1 = [b] s2
by (auto intro:interpretLow2)
with [b] s1 = Some false have [b] s2 = Some false by simp
with ⟨while (b) c',s2⟩ →* ⟨Skip,s2'⟩ have s2 = s2' by (auto intro:WhileFalse-reds)
with Γ ⊢ s1 ≈ _L s2 show ?case by auto
next
case (true s1 s1')

note IH = ⟨s2'', Γ ⊢ s1'' ≈ _L s2''; ⟨while (b) c',s2''⟩ →* ⟨Skip,s2''⟩ ⟹ Γ ⊢ s1' ≈ _L s2' ⟹ Γ ⊢ b : Low; Γ ⊢ s1 ≈ _L s2; have [b] s1 = [b] s2
by (auto intro:interpretLow2)
with [b] s1 = Some true have [b] s2 = Some true by simp
with \((\text{while } (b) c',s2) \rightarrow\ast (\text{Skip},s2')\) obtain \(s2''\) where \((c',s2) \rightarrow\ast (\text{Skip},s2')\)
and \((\text{while } (b) c',s2') \rightarrow\ast (\text{Skip},s2')\) by (auto dest:WhileTrue-reds)
from body[OF \((\langle c',s1 \rangle \rightarrow\ast (\text{Skip},s1') \mid (c',s2) \rightarrow\ast (\text{Skip},s2')\); \(\Gamma \vdash s1 \approx_L s2\)]
have \(\Gamma \vdash s1'' \approx_L s2''\).
from IH[OF this \((\langle (b) c',s2'\rangle \rightarrow\ast (\text{Skip},s2')\)] show \(?case\).
qed

lemma WhileLowCompositionality:
assumes nonInterference \(\Gamma c'\) and \(\Gamma \vdash b : \text{Low}\) and \(\Gamma,\text{Low} \vdash c'\)
shows nonInterference \(\Gamma (\text{while } (b) c')\)
proof (rule nonInterferenceI)
fix \(s1\) \(s2\) \(s1'\) \(s2'\)
assume \(\Gamma \vdash s1 \approx_L s2\) and \((\text{while } (b) c',s1) \rightarrow\ast (\text{Skip},s1')\)
and \((\text{while } (b) c',s2) \rightarrow\ast (\text{Skip},s2')\)
\{ fix \(s1\) \(s2\) \(s1''\) \(s2''\)
assume \((c',s1) \rightarrow\ast (\text{Skip},s1')\) and \((c',s2) \rightarrow\ast (\text{Skip},s2')\)
and \(\Gamma \vdash s1 \approx_L s2\)
with (nonInterference \(\Gamma c'\) have \(\Gamma \vdash s1'' \approx_L s2''\)
  by (auto simp:nonInterference-def)
\}
hence \(\bigwedge s1\) \(s1''\) \(s2\) \(s2''\). [(\(c',s1\) \rightarrow\ast (\text{Skip},s1'); \(c',s2) \rightarrow\ast (\text{Skip},s2');\]
\(\Gamma \vdash s1 \approx_L s2\) \(\implies\) \(\Gamma \vdash s1'' \approx_L s2''\) by auto
with \((\langle (b) c',s1\rangle \rightarrow\ast (\text{Skip},s1') \mid (\langle (b) c',s2\rangle \rightarrow\ast (\text{Skip},s2')\)
\(\Gamma \vdash b : \text{Low}\) \(\Gamma \vdash s1 \approx_L s2\) show \(\Gamma \vdash s1' \approx_L s2'\)
by (auto intro:WhileLowStepInduct)
qed

and now: the main theorem:

theorem secTypeImpliesNonInterference:
\(\Gamma, T \vdash c \implies \text{nonInterference } \Gamma c\)
proof (induct c arbitrary: \(T\) rule:com.induct)
case \(\text{Skip}\)
show \(?case\)
proof (rule nonInterferenceI)
fix \(s1\) \(s2\) \(s1'\) \(s2'\)
assume \(\Gamma \vdash s1 \approx_L s2\) and \((\text{Skip},s1) \rightarrow\ast (\text{Skip},s1')\) and \((\text{Skip},s2) \rightarrow\ast (\text{Skip},s2')\)
from \((\text{Skip},s1) \rightarrow\ast (\text{Skip},s1')\) have \(s1 = s1'\) by (auto dest:Skip-reds)
from \((\text{Skip},s2) \rightarrow\ast (\text{Skip},s2')\) have \(s2 = s2'\) by (auto dest:Skip-reds)
from \(\Gamma \vdash s1 \approx_L s2\) and \((\text{Skip},s1) \rightarrow\ast (\text{Skip},s1')\) and \(s2 = s2'\)
show \(\Gamma \vdash s1' \approx_L s2'\) by simp
qed

next
case \((\text{LAss } V e)\)
from \(\Gamma, T \vdash V := e\)
have varprem: \((\Gamma V = \text{Some High}) \vee (\Gamma V = \text{Some Low} \land \Gamma \vdash e : \text{Low} \land T = \text{Low})\)
  by (auto elim:secComTyping.cases)

22
show \( ?\text{case} \)
proof (rule nonInterferenceI)
  fix \( s_1 \) \( s_2 \) \( s_1' \) \( s_2' \)
  assume \( \Gamma \vdash s_1 \approx_L s_2 \) and \( (V:=e,s_1) \rightarrow^* (\text{Skip},s_1') \) and \( (V:=e,s_2) \rightarrow^* (\text{Skip},s_2') \)
  from \( (V:=e,s_1) \rightarrow^* (\text{Skip},s_1') \) have \( s_1' = s_1(V:=[e] s_1) \) by (auto intro:LAss-reds)
  from \( (V:=e,s_2) \rightarrow^* (\text{Skip},s_2') \) have \( s_2' = s_2(V:=[e] s_2) \) by (auto intro:LAss-reds)
  from \( \text{varprem} \) show \( \Gamma \vdash s_1' \approx_L s_2' \)
proof
  assume \( \Gamma \vdash V = \text{Some High} \)
  with \( \Gamma' \vdash s_1 \approx_L s_2 \) \( (s_1' = s_1(V:=[e] s_1)) \) \( (s_2' = s_2(V:=[e] s_2)) \)
  show \( ?\text{thesis} \) by (auto intro:assignNIhighlemma)
next
  assume \( \Gamma \vdash V = \text{Some Low} \) and \( \Gamma \vdash \text{Low} \land T = \text{Low} \)
  with \( \Gamma' \vdash s_1 \approx_L s_2 \) \( (s_1' = s_1(V:=[e] s_1)) \) \( (s_2' = s_2(V:=[e] s_2)) \)
  show \( ?\text{thesis} \) by (auto intro:assignNIlowlemma)
qed
qed

next
  case (Seq \( c_1 \) \( c_2 \))
  note IH1 = \( \langle \land T, \Gamma, T \vdash c_1 \Longrightarrow \text{nonInterference} \Gamma \; c_1 \rangle \)
  note IH2 = \( \langle \land T, \Gamma, T \vdash c_2 \Longrightarrow \text{nonInterference} \Gamma \; c_2 \rangle \)
  show \( ?\text{case} \)
proof (cases \( T \))
  case High
  with \( \Gamma, T \vdash c_1 ; ; c_2 \) have \( \Gamma, \text{High} \vdash c_1 \) and \( \Gamma, \text{High} \vdash c_2 \)
  by (auto elim:secComTyping.cases)
  from IH1[OF \( \Gamma, \text{High} \vdash c_1 \)] have nonInterference \( \Gamma \; c_1 \).
  moreover
  from IH2[OF \( \Gamma, \text{High} \vdash c_2 \)] have nonInterference \( \Gamma \; c_2 \).
  ultimately show \( ?\text{thesis} \) by (auto intro:SeqCompositionality)
next
  case Low
  with \( \Gamma, T \vdash c_1 ; ; c_2 \) have \( \Gamma, \text{Low} \vdash c_1 \) and \( \Gamma, \text{Low} \vdash c_2 \)
  by (auto elim:secComTyping.cases)
  thus \( ?\text{thesis} \)
proof
  assume \( \Gamma, \text{Low} \vdash c_1 \) and \( \Gamma, \text{Low} \vdash c_2 \)
  hence \( \Gamma, \text{Low} \vdash c_1 \) and \( \Gamma, \text{Low} \vdash c_2 \) by simp-all
  from IH1[OF \( \Gamma, \text{Low} \vdash c_1 \)] have nonInterference \( \Gamma \; c_1 \).
  moreover
  from IH2[OF \( \Gamma, \text{Low} \vdash c_2 \)] have nonInterference \( \Gamma \; c_2 \).
  ultimately show \( ?\text{thesis} \) by (auto intro:SeqCompositionality)
next
  assume \( \Gamma, \text{High} \vdash c_1 ; ; c_2 \)
  hence \( \Gamma, \text{High} \vdash c_1 \) and \( \Gamma, \text{High} \vdash c_2 \) by (auto elim:secComTyping.cases)

23
from \(IH1\) \([OF \Gamma, \text{High} \vdash c1]\) have nonInterference \(\Gamma c1\).
moreover
from \(IH2\) \([OF \Gamma, \text{High} \vdash c2]\) have nonInterference \(\Gamma c2\).
ultimately show \(?thesis\) by(auto intro:SeqCompositionality)
qed

next
case 
  \(\text{Cond} \ b \ c1 \ c2\)
  note \(IH1\) = \(<\forall \ T. \Gamma, T \vdash c1 = \Rightarrow \text{nonInterference} \Gamma c1>\)
  note \(IH2\) = \(<\forall \ T. \Gamma, T \vdash c2 = \Rightarrow \text{nonInterference} \Gamma c2>\)
  show \(?thesis\)
    proof (cases \(T\))
      case High
      with \(\Gamma, T \vdash \text{if} (b) \ c1 \ \text{else} \ c2\) show \(?thesis\)
        by(auto intro:CondHighCompositionality)
    next
      case Low
      with \(\Gamma, T \vdash \text{if} (b) \ c1 \ \text{else} \ c2\) have \(\Gamma \vdash b : \text{Low} \ \& \ \Gamma, \text{Low} \vdash c1 \ \& \ \Gamma, \text{Low} \vdash c2\) \(\lor \Gamma, \text{High} \vdash \text{if} (b) \ c1 \ \text{else} \ c2\)
        by(auto elim:secComTyping.cases)
      thus \(?thesis\)
        proof
          assume \(\Gamma \vdash b : \text{Low} \ \& \ \Gamma, \text{Low} \vdash c1 \ \& \ \Gamma, \text{Low} \vdash c2\)
          hence \(\Gamma \vdash b : \text{Low} \ \& \ \Gamma, \text{Low} \vdash c1 \ \& \ \Gamma, \text{Low} \vdash c2\) by simp-all
          from \(IH1\) \([OF \ \Gamma, \text{Low} \vdash c1]\) have nonInterference \(\Gamma c1\).
          moreover
          from \(IH2\) \([OF \ \Gamma, \text{Low} \vdash c2]\) have nonInterference \(\Gamma c2\).
          ultimately show \(?thesis\) using \(\Gamma \vdash b : \text{Low}\)
            by(auto intro:CondLowCompositionality)
        qed
    qed
next
case 
  \(\text{While} \ b \ c'\)
  note \(IH\) = \(<\forall \ T. \Gamma, T \vdash c' = \Rightarrow \text{nonInterference} \Gamma c'>\)
  show \(?thesis\)
    proof (cases \(T\))
      case High
      with \(\Gamma, T \vdash \text{while} (b) \ c'\) show \(?thesis\) by(auto intro:WhileHighCompositionality)
    next
      case Low
      with \(\Gamma, T \vdash \text{while} (b) \ c'\) have \(\Gamma \vdash b : \text{Low} \ \& \ \Gamma, \text{Low} \vdash c' \ \lor \Gamma, \text{High} \vdash \text{while} (b) \ c'\)
        by(auto elim:secComTyping.cases)
      thus \(?thesis\)
        proof
          assume \(\Gamma \vdash b : \text{Low} \ \& \ \Gamma, \text{Low} \vdash c'\)
hence $\Gamma \vdash b : \text{Low}$ and $\Gamma, \text{Low} \vdash c'$ by simp-all
moreover from $IH[\text{OF} \langle \Gamma, \text{Low} \vdash c' \rangle]$ have nonInterference $\Gamma \vdash c'$.
ultimately show $?\text{thesis}$ by(auto intro:WhileLowCompositionality)
next
assume $\Gamma, \text{High} \vdash \text{while}(b) \ c'$
thus $?\text{thesis}$ by(auto intro:WhileHighCompositionality)
qed
qed
end

theory Execute
imports secTypes
begin

3 Executing the small step semantics

code-pred (modes: $i => o => \text{bool}$ as exec-red, $i => i * o => \text{bool}$, $i => o * i => \text{bool}$, $i => i => \text{bool}$) red.

thm red.equation

definition [code]: one-step $x = \text{Predicate.the (exec-red} \ x)$

lemmas [code-pred-intro] = typeVal[where lev = \text{Low}] typeVal[where lev = \text{High}]
typeVar typeBinOp1 typeBinOp2[where lev = \text{Low}] typeBinOp2[where lev = \text{High}] typeBinOp3[where lev = \text{Low}]

code-pred (modes: $i => i => o => \text{bool}$ as compute-secExprTyping,
$i => i => i => \text{bool}$ as check-secExprTyping) secExprTyping

proof -
case secExprTyping
from secExprTyping.prems show thesis
proof
  fix $\Gamma \ V \ lev$ assume $x = \Gamma \ xa = \text{Val} \ V \ xb = \text{lev}$
  from secExprTyping(1 - 2) this show thesis by (cases lev) auto
next
  fix $\Gamma \ Vn \ lev$
  assume $x = \Gamma \ xa = \text{Var} \ Vn \ xb = \text{lev} \ \Gamma \ Vn = \text{Some} \ lev$
  from secExprTyping(3) this show thesis by (auto simp add: Predicate.eq-is-eq)
next
  fix $\Gamma \ \text{e1} \ \text{e2} \ \text{bop}$
  assume $x = \Gamma \ xa = \text{e1} \ bop \ \text{e2} \ xb = \text{Low}$
  $\Gamma \vdash e1 : \text{Low} \ \Gamma \vdash e2 : \text{Low}$
  from secExprTyping(4) this show thesis by auto
next
\begin{verbatim}

  fix \( \Gamma \) \( e_1 \) \( e_2 \) \( \text{lev} \) \( \text{bop} \)
  assume \( x = \Gamma \) \( xa = e_1 \ll bop \gg e_2 \) \( xb = \text{High} \)
  \( \Gamma \vdash e_1 : \text{High} \) \( \Gamma \vdash e_2 : \text{lev} \)
  from \text{secExprTyping}(5-6) this show thesis by (cases \text{lev}) (auto)

next

  fix \( \Gamma \) \( e_1 \) \( e_2 \) \( \text{lev} \) \( \text{bop} \)
  assume \( x = \Gamma \) \( xa = e_1 \ll bop \gg e_2 \) \( xb = \text{High} \)
  \( \Gamma \vdash e_1 : \text{lev} \) \( \Gamma \vdash e_2 : \text{High} \)
  from \text{secExprTyping}(6-7) this show thesis by (cases \text{lev}) (auto)

qed

lemmas [code-pred-intro] = typeSkip[where \( T = \text{Low} \)] typeSkip[where \( T = \text{High} \)]
  typeAssH[where \( T = \text{Low} \)] typeAssH[where \( T = \text{High} \)]
  typeAssL typeSeq typeWhile typeIf typeConvert

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

proof -

  case \text{secComTyping}

  from \text{secComTyping.prems} show thesis

proof

  fix \( \Gamma \) \( T \) assume \( x = \Gamma \) \( xa = T \) \( xb = \text{Skip} \)
  from \text{secComTyping}(1-2) this show thesis by (cases \( T \)) auto

next

  fix \( \Gamma \) \( V \) \( T \) \( e \) assume \( x = \Gamma \) \( xa = T \) \( xb = V:=e \) \( \Gamma \) \( V = \text{Some High} \)
  from \text{secComTyping}(3-4) this show thesis by (cases \( T \)) (auto)

next

  fix \( \Gamma \) \( e \) \( V \)
  assume \( x = \Gamma \) \( xa = \text{Low} \) \( xb = V:=e \) \( \Gamma \vdash e : \text{Low} \) \( \Gamma \) \( V = \text{Some Low} \)
  from \text{secComTyping}(5) this show thesis by auto

next

  fix \( \Gamma \) \( T \) \( c_1 \) \( c_2 \)
  assume \( x = \Gamma \) \( xa = T \) \( xb = \text{Seq} \) \( c_1 \) \( c_2 \) \( \Gamma , T \vdash c_1 \) \( \Gamma , T \vdash c_2 \)
  from \text{secComTyping}(6) this show thesis by auto

next

  fix \( \Gamma \) \( b \) \( T \) \( c \)
  assume \( x = \Gamma \) \( xa = T \) \( xb = \text{while} \) \( (b) \) \( c \) \( \Gamma \vdash b : T \) \( \Gamma , T \vdash c \)
  from \text{secComTyping}(7) this show thesis by auto

next

  fix \( \Gamma \) \( b \) \( T \) \( c_1 \) \( c_2 \)
  assume \( x = \Gamma \) \( xa = T \) \( xb = \text{if} \) \( (b) \) \( c_1 \) \( \text{else} \) \( c_2 \) \( \Gamma \vdash b : T \) \( \Gamma , T \vdash c_1 \) \( \Gamma , T \vdash c_2 \)
  from \text{secComTyping}(8) this show thesis by blast

next

  fix \( \Gamma \) \( c \)
  assume \( x = \Gamma \) \( xa = \text{Low} \) \( xb = c \) \( \Gamma , \text{High} \vdash c \)
  from \text{secComTyping}(9) this show thesis by blast

qed

qed

\end{verbatim}
\textbf{thm} \texttt{secComTyping.equation}

3.1 An example taken from Volpano, Smith, Irvine

\textbf{definition} \texttt{com} = \text{if (Var "x" ≪Eq\ Val (Intg 1)) ("y" := Val (Intg 1)) else ("y" := Val (Intg 0))}

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

\textbf{values} \{ \texttt{T. Env ⊢ (Var "x" ≪Eq Val (Intg 1)): T} \}

\textbf{value} \texttt{Env, High ⊢ com}

\textbf{value} \texttt{Env, Low ⊢ com}

\textbf{values} 1 \{ \texttt{T. Env, T ⊢ com} \}

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

\textbf{value} \texttt{Env', Low ⊢ com}

\textbf{value} \texttt{Env', High ⊢ com}

\textbf{values} 1 \{ \texttt{T. Env, T ⊢ com} \}

end

\textbf{References}
