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

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

April 20, 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 imprecise, resulting in false alarms. However, due to the correctness property, all potential breaks of confidentiality are discovered.
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(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
\[ FVc: \text{fv} (\text{Val } V) = \{ \} \]
\[ | \quad FVv: \text{fv} (\text{Var } V) = \{ V \} \]
\[ | \quad FVe: \text{fv} (e1 \leftarrow bop \rightarrow e2) = \text{fv } e1 \cup \text{fv } e2 \]

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 (\cdot \cdot)
**where** Val: [\text{Val } v] s = Some v
| Var: [\text{Var } V] s = s V
| BinOp: [\text{e1} \leftarrow bop \rightarrow e2] s = (case [\text{e1}] s of None \Rightarrow None
| Some v1 \Rightarrow (case [\text{e2}] s of None \Rightarrow None
| Some v2 \Rightarrow \text{binop } bop \text{ 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(\cdot,\cdot)) -/ (1(\cdot,\cdot))) [0,0,0,0] 81)\]
**where**
\((c1,s1) \rightarrow (c2,s2) = \text{red } (c1,s1) \rightarrow (c2,s2)\]
| RedLAss:
\((\text{V:=e},s) \rightarrow (\text{Skip},s(\text{V:=[e]} s)))\]
| | \rightarrow (c2,s)
| SeqRed:
\((\text{Skip;:c2},s) \rightarrow (c2,s)\]
| | \rightarrow (c1;:c2,s)\]
| RedCondTrue:
\([b] s = \text{Some true} \Rightarrow (\text{if } (b) c1 \text{ else } c2,s) \rightarrow (c1,s)\]
| RedCondFalse:
\([b] s = \text{Some false} \Rightarrow (\text{if } (b) c1 \text{ else } c2,s) \rightarrow (c2,s)\]
| RedWhileTrue:
\([b] s = \text{Some true} \Rightarrow (\text{while } (b) c,s) \rightarrow (c;:\text{while } (b) c,s)\]
| RedWhileFalse:
\([b] s = \text{Some false} \Rightarrow (\text{while } (b) c,s) \rightarrow (\text{Skip},s)\]

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

**abbreviation** reds :: com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool
\(((1(\cdot,\cdot)) -/ (1(\cdot,\cdot))) [0,0,0,0] 81)\]
**where**
\((c,s) \rightarrow (c',s') = \text{red}^* (c,s) \rightarrow (c',s')\]
lemma \textit{Skip-\text{reds}}:
\[(\text{Skip},s) \Rightarrow (c',s') \quad \Rightarrow \quad s = s' \land c' = \text{Skip}\]
by (\text{blast elim:converse-rtranclpE red.cases})

lemma \textit{LAss-\text{reds}}:
\[(V := c \cdot s) \Rightarrow (\text{Skip},s') \quad \Rightarrow \quad s' = s(V := [c] s)\]
proof (\text{induct } V := c \cdot s \text{ rule:converse-rtranclp-induct2})
\quad \text{case } (\text{step } s \quad c'' \cdot s''\]
\quad \text{hence } c'' = \text{Skip and } s'' = s(V := ([c] s)) by (\text{auto elim:red.cases})
\quad \text{with } \langle c'',s'' \rangle \Rightarrow \ast (\text{Skip},s') \quad \text{show } ?\text{case by}(\text{auto dest:Skip-\text{reds}})
\end{proof}
\begin{proof}
\text{qed}
\end{proof}

lemma \textit{Seq2-\text{reds}}:
\[(\text{Skip};c_2,s) \Rightarrow (\text{Skip},s') \quad \Rightarrow \quad c_2,s) \Rightarrow \ast (\text{Skip},s')\]
by (\text{induct } c := \text{Skip};c_2 \text{ s rule:converse-rtranclp-induct2})(\text{auto elim:red.cases})

lemma \textit{Seq-\text{reds}}:
assumes \[(c_1; c_2,s) \Rightarrow \ast (\text{Skip},s')\]
obtains \[s'' \quad \text{where } \langle c_1,s \rangle \Rightarrow \ast (\text{Skip},s'') \land \langle c_2,s' \rangle \Rightarrow \ast (\text{Skip},s')\]
proof
\quad \text{have } \exists s''. \langle c_1,s \rangle \Rightarrow \ast (\text{Skip},s'') \land (c_2,s'') \Rightarrow \ast (\text{Skip},s')
\quad \text{proof}
\quad \quad \{ \text{ fix } c, c'\]
\quad \quad \quad \text{assume } \langle c,s \rangle \Rightarrow \ast (c',s') \land c = c_1; c_2 \text{ and } c' = \text{Skip}
\quad \quad \quad \text{hence } \exists s''. \langle c_1,s \rangle \Rightarrow \ast (\text{Skip},s'') \land (c_2,s'') \Rightarrow \ast (\text{Skip},s')
\quad \quad \text{proof (induct arbitrary: } c_1 \text{ rule:converse-rtranclp-induct2) }
\quad \quad \quad \text{case refl thus } ?\text{case by simp}
\quad \quad \text{next}
\quad \quad \quad \text{case } (\text{step } s \quad c c'' \cdot s''\]
\quad \quad \quad \text{note } \text{IH } = \langle \forall c_1. \ [c'' = c_1; c_2; c' = \text{Skip}\]
\quad \quad \quad \quad \quad \Rightarrow \exists sx. \langle c_1,s'' \rangle \Rightarrow \ast (\text{Skip},sx) \land (c_2,sx) \Rightarrow \ast (\text{Skip},s')\]
\quad \quad \quad \text{from step}
\quad \quad \quad \text{have } \langle c_1; c_2, s \rangle \Rightarrow \ast (c'',s'') \text{ by simp}
\quad \quad \quad \text{hence } (c_1 = \text{Skip} \land c'' = c_2 \land s = s'') \lor
\quad \quad \quad (\exists c'_1. \langle c_1,s \rangle \Rightarrow \ast (c'_1,s'') \land c'' = c_1'; c_2)
\quad \quad \quad \text{by (auto elim:red.cases)}
\quad \quad \text{thus } ?\text{case}
\quad \text{proof}
\quad \quad \quad \text{assume } c_1 = \text{Skip} \land c'' = c_2 \land s = s''
\quad \quad \quad \text{with } \langle (c'',s'') \Rightarrow \ast (c',s') \rangle \quad (c' = \text{Skip})
\quad \quad \quad \text{show } ?\text{thesis by auto}
\quad \text{next}
\quad \quad \quad \text{assume } \exists c'_1. \langle c_1,s \rangle \Rightarrow \ast (c'_1,s'') \land c'' = c_1'; c_2
\quad \quad \quad \text{then obtain } c'_1 \text{ where } \langle c_1,s \rangle \Rightarrow \ast (c_1',s'') \land c'' = c_1'; c_2 \text{ by blast}
\quad \quad \quad \text{from } \text{IH}[OF \ (c'' = c_1'; c_2; c' = \text{Skip}]
\quad \quad \quad \text{obtain sx where } \langle c_1,s' \rangle \Rightarrow \ast (\text{Skip},sx) \text{ and } \langle c_2,sx \rangle \Rightarrow \ast (\text{Skip},s')
\quad \quad \quad \text{by blast}
\quad \quad \quad \text{from } \langle \langle c_1,s \rangle \Rightarrow \ast (c_1',s'' \rangle \Rightarrow (c_1',s'') \Rightarrow \ast (\text{Skip},sx)\rangle
\end{proof}
lemma Seq-red-nE
  with \langle c_1;c_2,s \rangle \rightarrow\{\text{Skip},s'\}
  show \neg \text{thesis} by auto
qed

lemma red-n-Base
  with \langle c_1;\bot,\top \rangle \rightarrow\{\text{Skip},s'\}
  show \neg \text{thesis} by simp
qed

with that show \neg \text{thesis} by blast
qed

lemma While-True-or-False:
\langle \text{if} \ (b) \ c_1 \ \text{else} \ c_2,s \ \rangle \rightarrow\{\text{Skip},s'\} \Longrightarrow [b] \ s = \text{Some true} \lor [b] \ s = \text{Some false}
by (induct c\equiv if \ (b) \ c_1 \ \text{else} \ c_2 \ s \ \text{rule: converse-rtranclp-induct2}) (auto elim:red.cases)

lemma CondTrue-reds:
\langle \text{if} \ (b) \ c_1 \ \text{else} \ c_2,s \ \rangle \rightarrow\{\text{Skip},s'\} \Longrightarrow [b] \ s = \text{Some true} \Longrightarrow \langle c_1,s \rangle \rightarrow\{\text{Skip},s'\}
by (induct c\equiv if \ (b) \ c_1 \ \text{else} \ c_2 \ s \ \text{rule: converse-rtranclp-induct2}) (auto elim:red.cases)

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

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

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

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

inductive red-n :: com \Rightarrow state \Rightarrow nat \Rightarrow com \Rightarrow state \Rightarrow bool
\langle \{1;\cdot\}\rangle \rightarrow\{1;\cdot\} [0,0,0,0,0] 81
where red-n-Base: \langle c,s \rangle \rightarrow\{ c,s \}
| red-n-Rec: \langle c,s \rangle \rightarrow\{ c'';s'' \}; \langle c'',s'' \rangle \rightarrow^n \langle c',s' \rangle \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 \{\text{Skip},s'\}
obtains \( i \ j \ s'' \) 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 \( \exists i j s''. \langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \langle c_2, s'' \rangle \rightarrow^j \langle \text{Skip}, s \rangle \land n = i + j + 1 \)
proof(induct \( c_1::c_2 \) \( s \) \( n \) \( \text{Skip} \) \( s' \) arbitrary; \( c_1 \) rule:red-n.induct)
case (red-n-Rec \( s \) \( c'' \) \( s'' \) \( n \) \( s' \))
note \( IH = \langle \land c_1. \ c'' = c_1::c_2 \Rightarrow \exists i sx. \langle c_1, s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle \land \langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s \rangle \land n = i + j + 1 \)
from \( \langle c_1::c_2, s \rangle \rightarrow \langle c''', s''' \rangle \)
have \( \langle c_1 = \text{Skip} \land c''' = c_2 \land s = s'' \rangle \land \\
(\exists c_1'. c''' = c_1::c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle) \)
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 \( c_1 = \text{Skip} \land c'' = c_2 \) and \( s = s'' \) by simp-all
from \( c_1 = \text{Skip} \) have \( \langle c_1, s \rangle \rightarrow^0 \langle \text{Skip}, s \rangle \) by (fastforce intro:red-n.Base)
with \( \langle c''', s''' \rangle \rightarrow^0 \langle \text{Skip}, s \rangle \), \( c''' = c_2 \) \( s = s'' \)
show \?thesis by(rule-tac \( x=0 \) in \text{exf}) auto
next
assume \( \exists c_1'. c''' = c_1::c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \)
then obtain \( c_1' \) where \( c''' = c_1::c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \) by blast
from \( IH[\text{OF c'''} = c_1::c_2] \) obtain \( i j sx \)
where \( \langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle \) and \( \langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s \rangle \)
and \( n = i + j + 1 \) by blast
from \( \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \) \( \langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle \)
have \( \langle c_1, s \rangle \rightarrow^{\text{Suc i}} \langle \text{Skip}, sx \rangle \) by(rule red-n.red-n-Rec)
with \( \langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s \rangle \) \( n = i + j + 1 \) show \?thesis
by(rule-tac \( x=\text{Suc i} \) in \text{exf}) auto
qed
qed
with that show \?thesis by blast
qed

lemma while-red-nE:
\( (\text{while} \ (b) \ cx, s) \rightarrow^n \langle \text{Skip}, s \rangle \) \Rightarrow \\
(\langle b \rangle \ s = \text{Some false} \land s = s' \land n = 1) \lor \\
(\exists i j s''. \langle b \rangle \ s = \text{Some true} \land \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \\
(\text{while} \ (b) \ cx, s'') \rightarrow^j \langle \text{Skip}, s \rangle \land n = i + j + 2) \)
proof(induct while \( b \) \( cx \) \( s \) arbitrary; \( b \) rule:red-n.induct)
case (red-n-Rec \( s \) \( c'' \) \( s'' \) \( n \) \( s' \))
from \( (\text{while} \ (b) \ cx, s) \rightarrow \langle c''', s''' \rangle \)
have \( \langle b \rangle \ s = \text{Some false} \land c''' = \text{Skip} \land s''' = s \) \lor \\
(\langle b \rangle \ 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 \(\langle c'',s'' \rangle \to^n \langle \text{Skip},s'' \rangle\) have \(s = s'\) and \(n = 0\) 

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

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

next 

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

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

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

with \(\langle c'',s'' \rangle \to^n \langle \text{Skip},s'' \rangle\) 

obtain \(i \ j \ sx\) where \(\langle cx,s \rangle \to^i \langle \text{Skip},sx \rangle\) and \(\langle while (b) cx,sx \rangle \to^j \langle \text{Skip},s' \rangle\) 

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

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

qed 

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

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

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

and IHtrue: \(\forall i \ j \ s''. \ [b] \ s = \text{Some true} ; \langle cx,s \rangle \to^i \langle \text{Skip},s'' \rangle; \ (\langle while (b) cx,s'' \rangle) \to^j \langle \text{Skip},s' \rangle; \ P \ s'' \ s' \implies P \ s \ s'\) 

shows \(P \ s \ s'\) 

using major 

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

fix \(n \ s\) 

assume IHHall: \(\forall m < n. \ \forall x. \langle while (b) cx,x \rangle \to^m \langle \text{Skip},s' \rangle \to P \ x \ s'\) 

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

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

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

\((\exists i \ j \ s''. \ [b] \ s = \text{Some true} ; \langle cx,s \rangle \to^i \langle \text{Skip},s'' \rangle; \ (\langle while (b) cx,s'' \rangle) \to^j \langle \text{Skip},s' \rangle \land n = i + j + 2)\) 

by (rule while-red-nE) 

thus \(P \ s \ s'\) 

proof 

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

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 

assume \(\exists i \ j \ s''. \ [b] \ s = \text{Some true} \land \langle cx,s \rangle \to^i \langle \text{Skip},s'' \rangle \land (\langle while (b) cx,s'' \rangle) \to^j \langle \text{Skip},s' \rangle \land n = i + j + 2)\) 

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

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

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

with IHHall have \(P \ s'' \ s'\) 

apply (rule_tac \(x=j\) in allE) apply clarsimp done 

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

\text{qed}

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

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

\text{lemma while-reds-induct [consumes 1, case-names false true]:}
\begin{align*}
[(\text{while } (b) cx, s) \rightarrow^* \langle \text{Skip}, s' \rangle; \text{All s. } [b] s = \text{Some false } & \Rightarrow P s s; \\
\text{All s s'' [b] s = Some true; } (cx, s) \rightarrow^* \langle \text{Skip}, s'' \rangle; \\
(\text{while } (b) cx, s'') \rightarrow^* \langle \text{Skip}, s' \rangle; P s'' s] & \Rightarrow P s s']
\end{align*}
\text{apply(drue reds-to-red-n, clarsimp)}
\text{apply(erule while-red-n-induct, clarsimp)}
\text{by(auto dest:red-n-to-reds)}

\text{lemma red-det:}
\begin{align*}
[cx, s] \rightarrow \langle c_1, s_1 \rangle; [c, s] \rightarrow \langle c_2, s_2 \rangle \implies c_1 = c_2 \land s_1 = s_2
\end{align*}
\text{proof(induct arbitrary: } c_2 \text{ rule:red-induct)}
\text{case } (\text{SeqRed } c_1 s c_1' s' c_2')
\text{note } IH = (\text{All } c_1; (c_1, s) \rightarrow \langle c_2, s_2 \rangle \implies c_1' = c_2 \land s' = s_2)
\text{from } (c_1; c_2', s) \rightarrow \langle c_2, s_2 \rangle \text{ have } c_1 = \text{Skip } \lor (\exists cx. c_2 = cx; c_2' \land (c_1, s) \rightarrow \langle cx, s_2 \rangle)
\text{by(fastforce elim:red.cases)}
\text{thus } ?\text{case}
\text{proof}
\begin{align*}
&\text{assume } c_1 = \text{Skip} \\
&\text{with } (c_1, s) \rightarrow \langle c_1', s'_1 \rangle \text{ have False by(fastforce elim:red.cases)}
\end{align*}
\text{thus } ?\text{thesis by simp}
\text{next}
\begin{align*}
&\text{assume } \exists cx. c_2 = cx; c_2' \land (c_1, s) \rightarrow \langle cx, s_2 \rangle \\
&\text{then obtain cx where } c_2 = cx; c_2' \land (c_1, s) \rightarrow \langle cx, s_2 \rangle \text{ by blast}
\end{align*}
\text{from } IH \{ OF \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle \} \text{ have } c_1' = cx \land s' = s_2 .
\text{with } (c_2 = cx; c_2') \text{ show } ?\text{thesis by simp}
\text{qed}
\text{qed (fastforce elim:red.cases)+}

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

end 
theory \text{secTypes} 
imports Semantics 
begin 

\hspace{1cm} 2 \ Security \ types 

\hspace{1cm} 2.1 \ Security \ definitions 

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{typeVar:} \quad \Gamma \vdash \text{Val} V : \text{lev} 
  | \text{typeBinOp1:} \quad [ \Gamma \vdash e_1 : \text{Low}; \Gamma \vdash e_2 : \text{Low}] \Longrightarrow \Gamma \vdash e_1 \langle \text{bop} \rangle e_2 : \text{Low} 
  | \text{typeBinOp2:} \quad [ \Gamma \vdash e_1 : \text{High}; \Gamma \vdash e_2 : \text{lev}] \Longrightarrow \Gamma \vdash e_1 \langle \text{bop} \rangle e_2 : \text{High} 
  | \text{typeBinOp3:} \quad [ \Gamma \vdash e_1 : \text{lev}; \Gamma \vdash e_2 : \text{High}] \Longrightarrow \Gamma \vdash e_1 \langle \text{bop} \rangle e_2 : \text{High} 

inductive \text{secComTyping} :: \text{typeEnv} \Rightarrow \text{secLevel} \Rightarrow \text{com} \Rightarrow \text{bool} \ (-,- \vdash -) 
where 
  \text{typeSkip:} \quad \Gamma, T \vdash \text{Skip} 
  | \text{typeAssH:} \quad \Gamma \vdash V = \text{Some High} \Longrightarrow \Gamma, T \vdash V := e 
  | \text{typeAssL:} \quad \Gamma \vdash e : \text{Low}; \Gamma \vdash V = \text{Some Low} \Longrightarrow \Gamma, \text{Low} \vdash V := e 
  | \text{typeSeq:} \quad [ \Gamma, T \vdash c_1; \Gamma, T \vdash c_2] \Longrightarrow \Gamma, T \vdash c_1;;c_2 
  | \text{typeWhile:} \quad [ \Gamma \vdash b : T; \Gamma, T \vdash c] \Longrightarrow \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 \(fv \ e \subseteq \text{dom} \ \Gamma\) obtains \(T\) where \(\Gamma \vdash e : T\)

**proof**

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

proof(induct \(e\))

case \((\text{Val} \ V)\)

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

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

next

case \((\text{Var} \ V)\)

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

with \((fv (\text{Var} \ V) \subseteq \text{dom} \ \Gamma)\) 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 \((\text{BinOp} \ e1 \ bop \ e2)\)

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

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

from \((fv (\text{BinOp} \ e1 \ bop \ e2) \subseteq \text{dom} \ \Gamma)\) have \(fv \ e1 \subseteq \text{dom} \ \Gamma\) and \(fv \ e2 \subseteq \text{dom} \ \Gamma\) by auto

from \(IH1[\text{OF} \ (fv \ e1 \subseteq \text{dom} \ \Gamma)]\) obtain \(T1\) where \(\Gamma \vdash e1 : T1\) by auto

from \(IH2[\text{OF} \ (fv \ e2 \subseteq \text{dom} \ \Gamma)]\) obtain \(T2\) where \(\Gamma \vdash e2 : T2\) by auto

show \(?case\)

proof (cases \(T1\))

case \(\text{Low}\)

show \(?thesis\)

proof (cases \(T2\))

case \(\text{Low}\)

with \((\Gamma \vdash e1 : T1) \ (\Gamma \vdash e2 : T2) \ (T1 = \text{Low})\)

have \(\Gamma \vdash e1 \ \text{bop} \ e2 : \text{Low}\) by (simp add: typeBinOp1)

thus \(?thesis\) by (rule exI)

next

case \(\text{High}\)

with \((\Gamma \vdash e1 : T1) \ (\Gamma \vdash e2 : T2) \ (T1 = \text{Low})\)

have \(\Gamma \vdash e1 \ \text{bop} \ e2 : \text{High}\) by (simp add: typeBinOp3)

thus \(?thesis\) by (rule exI)

qed

next

case \(\text{High}\)

with \((\Gamma \vdash e1 : T1) \ (\Gamma \vdash e2 : T2)\)

have \(\Gamma \vdash e1 \ \text{bop} \ e2 : \text{High}\) by (simp add: typeBinOp2)
thus ?thesis by (rule exI)
qed
qed

with that show ?thesis by blast
qed

lemma exprBinopTypeable:
  assumes Γ ⊢ e1 ≪ bop ≫ e2 : T
  shows (∃ T1. Γ ⊢ e1 : T1) ∧ (∃ T2. Γ ⊢ e2 : T2)
using assms by(auto elim:secExprTyping.cases)

lemma exprTypingHigh:
  assumes Γ ⊢ e : T and x ∈ fv e and Γ x = Some High
  shows Γ ⊢ e : High
using assms
proof (induct e arbitrary: T)
  case (Val V) show ?case by (rule typeVal)
  next
case (Var V)
  from ⟨x ∈ fv (Var V)⟩ have x = V by simp
with ⟨Γ x = Some High⟩
  show ?case by(simp add:typVar)
  next
case (BinOp e1 bop e2)
  note IH1 = ⟨∀ T. (Γ ⊢ e1 : T; x ∈ fv e1; Γ x = Some High) ⟹ Γ ⊢ e1 : High⟩
  note IH2 = ⟨∀ T. (Γ ⊢ e2 : T; x ∈ fv e2; Γ x = Some High) ⟹ Γ ⊢ e2 : High⟩
  from (Γ ⊢ e1 ≪ bop ≫ e2 : T)
  have T:∃ T1. Γ ⊢ e1 : T1) ∧ (∃ T2. Γ ⊢ e2 : T2) by (auto intro!:exprBinopTypeable)
then obtain T1 where Γ ⊢ e1 : T1 by auto
from T obtain T2 where Γ ⊢ e2 : T2 by auto
from ⟨x ∈ fv (e1 ≪ bop ≫ e2)⟩ have x ∈ (fv e1 ∪ fv e2) by simp
hence x ∈ fv e1 ∨ x ∈ fv e2 by auto
thus ?case
proof
  assume x ∈ fv e1
  from IH1[OF Γ ⊢ e1 : T1] this ⟨Γ x = Some High⟩ have Γ ⊢ e1 : High .
  with Γ ⊢ e1 : T1 show ?thesis by(simp add:typBinOp2)
  next
  assume x ∈ fv e2
  from IH2[OF Γ ⊢ e2 : T2] this ⟨Γ x = Some High⟩ have Γ ⊢ e2 : High .
  with Γ ⊢ e2 : T2 show ?thesis by(simp add:typBinOp3)
qed
qed

lemma exprTypingLow:
  assumes Γ ⊢ e : Low and x ∈ fv e shows Γ x = Some Low

using assms

proof (induct e)
  case (Val V)
  have $fv (Val V) = \{\}$ by (rule FVc)
  with $x \in fv (Val V)$ have False by auto
  thus ?thesis by simp
next
case (Var V)
  from $x \in fv (Var V)$ have $xV : x = V$ by simp
  from $\Gamma \vdash Var V : Low$ have $\Gamma V = Some Low$ by (auto elim:secExprTyping.cases)
  with $xV$ show ?thesis by simp
next
case (BinOp e1 bop e2)
  note IH1 = $\langle \forall T \cdot \Gamma \vdash e1 : T \implies \Gamma x = Some Low \rangle$
  note IH2 = $\langle \forall T \cdot \Gamma \vdash e2 : T \implies \Gamma x = Some Low \rangle$
  from $\Gamma \vdash e1 <bop> e2 : Low$ have $\Gamma \vdash e1 : Low$ and $\Gamma \vdash e2 : Low$
    by (auto elim:secExprTyping.cases)
  from $x \in fv (e1 <bop> e2)$ have $x \in fv e1 \cup fv e2$ by (simp add:FVc)
  hence $x \in fv e1 \lor x \in fv e2$ by auto
  thus ?case
  proof
    assume $x \in fv e1$
    with IH1[OF $\Gamma \vdash e1 : Low$] show ?thesis by auto
  next
    assume $x \in fv e2$
    with IH2[OF $\Gamma \vdash e2 : Low$] show ?thesis by auto
  qed
qed

lemma typeableFreevars:
  assumes $\Gamma \vdash e : T$ shows $fv e \subseteq dom \Gamma$
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 $\Gamma \vdash Var V : T$ have $\Gamma V = Some T$ by (auto elim:secExprTyping.cases)
      with $x = V$ show $x \in dom \Gamma$ by auto
  qed
next
case (BinOp e1 bop e2)
  note IH1 = $\langle \forall T \cdot \Gamma \vdash e1 : T \implies fv e1 \subseteq dom \Gamma \rangle$
note \( IH2 = \forall T. \Gamma \vdash e_2 : T \implies \text{fv } e_2 \subseteq \text{dom } \Gamma \)

show \(?\text{case}\)

proof

fix \( x \) assume \( x \in \text{fv } (e_1 \langle bop \rangle e_2) \)

from \( \Gamma \vdash e_1 \langle bop \rangle e_2 : T \)

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 \( IH1[\text{OF } \Gamma \vdash e_1 : T_1] \) have \( \text{fv } e_1 \subseteq \text{dom } \Gamma \).

moreover from \( IH2[\text{OF } \Gamma \vdash e_2 : T_2] \) have \( \text{fv } e_2 \subseteq \text{dom } \Gamma \).

ultimately have \( (\text{fv } e_1) \cup (\text{fv } e_2) \subseteq \text{dom } \Gamma \) by auto

hence \( \text{fv } (e_1 \langle bop \rangle e_2) \subseteq \text{dom } \Gamma \) by (simp add: FVv)

with \( x \in \text{fv } (e_1 \langle bop \rangle e_2) \) show \( x \in \text{dom } \Gamma \) by auto

qed

lemma \( \text{exprNotNone} \):

assumes \( \Gamma \vdash e : T \) and \( \text{fv } e \subseteq \text{dom } s \)

shows \( [e]s \neq \text{None} \)

using \( \text{assms} \)

proof (induct \( e \) arbitrary: \( \Gamma T s \))

next case (Val \( v \))

show \(?\text{case}\) 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: FVv)

with \( \text{fv } (\text{Var } V) \subseteq \text{dom } s \) have \( V \in \text{dom } s \) by simp

thus \(?\text{case}\) by auto

next case (BinOp \( e_1 \) \( bop \) \( e_2 \))

note \( IH1 = \forall T. \Gamma \vdash e_1 : T; \text{fv } e_1 \subseteq \text{dom } s \implies [e_1]s \neq \text{None} \)

note \( IH2 = \forall T. \Gamma \vdash e_2 : T; \text{fv } e_2 \subseteq \text{dom } s \implies [e_2]s \neq \text{None} \)

from \( \Gamma \vdash e_1 \langle bop \rangle e_2 : T \) have \( (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2) \)

by (rule exprBinopTypeable)

then obtain \( T_1 T_2 \) where \( \Gamma \vdash e_1 : T_1 \) and \( \Gamma \vdash e_2 : T_2 \) by blast

from \( \text{fv } (e_1 \langle bop \rangle e_2) \subseteq \text{dom } s \) have \( \text{fv } e_1 \cup \text{fv } e_2 \subseteq \text{dom } s \) by (simp add: FVv)

hence \( \text{fv } e_1 \subseteq \text{dom } s \) and \( \text{fv } e_2 \subseteq \text{dom } s \) by auto

from \( IH1[\text{OF } \Gamma \vdash e_1 : T_1; \text{fv } e_1 \subseteq \text{dom } s] \) have \( [e_1]s \neq \text{None} \).

moreover from \( IH2[\text{OF } \Gamma \vdash e_2 : T_2; \text{fv } e_2 \subseteq \text{dom } s] \) have \( [e_2]s \neq \text{None} \).

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

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 transitiv (see lemmas) hence an equivalence relation.

**Definition** lowEquiv :: typeEnv ⇒ state ⇒ state ⇒ 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** lowEquivReflexive: \( \Gamma \vdash s_1 \approx_L s_1 \)

by (simp add: lowEquiv-def)

**Lemma** lowEquivSymmetric: \( \Gamma \vdash s_1 \approx_L s_2 \rightarrow \Gamma \vdash s_2 \approx_L s_1 \)

by (simp add: lowEquiv-def)

**Lemma** lowEquivTransitive: \[ [\Gamma \vdash s_1 \approx_L s_2; \Gamma \vdash s_2 \approx_L s_3] \rightarrow \Gamma \vdash s_1 \approx_L s_3 \]

by (simp add: lowEquiv-def)

2.3.2 Non Interference

**Definition** nonInterference :: typeEnv ⇒ com ⇒ bool

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

**Lemma** nonInterferenceI: \[ [\bigwedge s_1 s_2 s_1' s_2'. [\Gamma \vdash s_1 \approx_L s_2; \langle c, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle; \langle c, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle]] \rightarrow \Gamma \vdash s_1' \approx_L s_2' \]

by (auto simp: nonInterference-def)

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

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

using all

proof (induct e)

  case (Val v)

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

next

  case (Var V)
have \[\text{Var} V\] \(s1 = s1 V\) and \[\text{Var} V\] \(s2 = s2 V\) by (auto simp: Var)

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

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

with \text{assms} have \(s1 V = s2 V\) by (auto simp add: lowEquiv-def)

thus \(?\text{case}\) by \text{auto}

next

case (\text{BinOp} e1 bop e2)

note \(IH1 = \forall V \in \text{fv} e1. \Gamma V = \text{Some Low} \implies [e1] s1 = [e1] s2\)

note \(IH2 = \forall V \in \text{fv} e2. \Gamma V = \text{Some Low} \implies [e2] s1 = [e2] s2\)

from \(\forall V \in \text{fv} (e1 <bop> e2)\). \(\Gamma V = \text{Some Low}\) 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 \text{auto}

from \(IH1[\text{OF} \forall V \in \text{fv} e1. \Gamma V = \text{Some Low}]\) have \([e1] s1 = [e1] s2\)

moreover

from \(IH2[\text{OF} \forall V \in \text{fv} e2. \Gamma V = \text{Some Low}]\) have \([e2] s1 = [e2] s2\)

ultimately show \(?\text{case}\) by (cases \([e1] s2, \text{auto}\))

qed

lemma interpretLow2:

assumes \(\Gamma \vdash e : \text{Low}\) and \(\Gamma \vdash s1 \approx_L s2\) shows \([e] s1 = [e] s2\)

proof

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

qed

with \(\Gamma \vdash s1 \approx_L s2\) show \(?\text{thesis}\) by (rule interpretLow)

qed

lemma assignNlhighlemma:

assumes \(\Gamma \vdash s1 \approx_L s2\) and \(\Gamma V = \text{Some High}\) and \(s1' = s1(V := [e] s1)\)

and \(s2' = s2(V := [e] s2)\)

shows \(\Gamma \vdash s1' \approx_L s2'\)

proof

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

from \(\Gamma \vdash s1 \approx_L s2\) \(\Gamma V' = \text{Some Low}\) have \(s1 V' = s2 V'\)

by (auto simp add: lowEquiv-def)

have \(s1' V' = s2' V'\)

proof (cases \(V' = V\))

  case True

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

  thus \(?\text{thesis}\) by simp

next

  case False

  with \(s1' = s1(V := [e] s1)\) \(s2' = s2(V := [e] s2)\)
have \( s_1 V' = s_1' V' \) and \( s_2 V' = s_2' V' \) by \textit{auto}
with \( s_1 V' = s_2 V' \) show \( ?\text{thesis} \) by \textit{simp}
\textbf{qed}

\textbf{lemmas} \textit{assignNIlowlemma}:
\begin{align*}
\text{assumes} & \quad \Gamma \vdash s_1 \approx_L s_2 \quad \text{and} \quad \Gamma \vdash e : \text{Low} \\
\text{and} & \quad s_1' = s_1(V := [e] s_1) \quad \text{and} \quad s_2' = s_2(V := [e] s_2) \\
\text{shows} & \quad \Gamma \vdash s_1' \approx_L s_2'
\end{align*}
proof (cases \( V' = V \))
\begin{align*}
\text{case} & \quad \text{True} \\
\text{with} & \quad (s_1' = s_1(V := [e] s_1)) \quad (s_2' = s_2(V := [e] s_2)) \\
\text{have} & \quad s_1' V' = [e] s_1 \quad \text{and} \quad s_2' V' = [e] s_2 \text{ by \textit{auto}}
\end{align*}
from \( \Gamma \vdash e : \text{Low} \) \( \Gamma \vdash s_1 \approx_L s_2 \) have \( [e] s_1 = [e] s_2 \) by (\textit{auto intro:interpretLow2})
with \( (s_1' V' = [e] s_1) \quad (s_2' V' = [e] s_2) \) show \( ?\text{thesis} \) by \textit{simp}
next
case \( \text{False} \)
with \( (s_1' = s_1(V := [e] s_1)) \quad (s_2' = s_2(V := [e] s_2)) \)
\text{have} \( s_1' V' = s_1 V' \) and \( s_2' V' = s_2 V' \) by \textit{auto}
with \( s_1' V' = s_2 V' \) have \( s_1' V' = s_2' V' \) by \textit{simp}
show \( ?\text{thesis} \) by \textit{auto}
\textbf{qed}

\textbf{thus} \( ?\text{thesis} \) by (\textit{simp add:lowEquiv-def})
\textbf{qed}

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

\textbf{theorem} \textit{SeqCompositionality}:
\begin{align*}
\text{assumes} & \quad \text{nonInterference} \quad \Gamma \vdash c_1 \quad \text{and} \quad \text{nonInterference} \quad \Gamma \vdash c_2 \\
\text{shows} & \quad \text{nonInterference} \quad \Gamma \vdash (c_1 ; ; c_2)
\end{align*}
proof (\textit{rule nonInterferenceI})
fix \( s_1 \quad s_2 \quad s_1' \quad s_2' \)
assume \( \Gamma \vdash s_1 \approx_L s_2 \) \quad and \quad \( \Gamma \vdash (c_1 ; ; c_2, s_1) \rightarrow^* (\text{Skip}, s_1') \)
\quad and \quad \( \Gamma \vdash (c_1 ; ; c_2, s_2) \rightarrow^* (\text{Skip}, s_2') \)
from \( (c_1 ; ; c_2, s_1) \rightarrow^* (\text{Skip}, s_1') \) obtain \( s_1'' \) where \( (c_1, s_1) \rightarrow^* (\text{Skip}, s_1'') \)
\quad and \quad \( (c_2, s_1') \rightarrow^* (\text{Skip}, s_1') \) by (\textit{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 \(\langle c_2,s_2''\rangle \rightarrow* \langle\text{Skip},s_2'\rangle\) by (auto dest:Seg-reds)

from \(\Gamma \vdash s_1 \approx_L s_2\) \(\langle c_1,s_1\rangle \rightarrow* \langle\text{Skip},s_1''\rangle\) \(\langle c_1,s_2\rangle \rightarrow* \langle\text{Skip},s_2''\rangle\)

\(\text{nonInterference}\ \Gamma\ \ c_1\)

have \(\Gamma \vdash s_1'' \approx_L s_2''\) by (auto simp:nonInterference-def)

with \(\langle c_2,s_1''\rangle \rightarrow* \langle\text{Skip},s_1''\rangle\) \(\langle c_2,s_2''\rangle \rightarrow* \langle\text{Skip},s_2''\rangle\) (nonInterference \(\Gamma\ c_2\))

show \(\Gamma \vdash s_1' \approx_L s_2'\) by (auto simp:nonInterference-def)

qed

lemma WhileStepInduct:

assumes while: \(\langle\text{while } (b) \ c,s_1\rangle \rightarrow* \langle\text{Skip},s_2\rangle\)

and body: \(\Gamma \vdash s_2.\ (c,s_1) \rightarrow* \langle\text{Skip},s_2'\rangle\) \(~\Rightarrow\ \Gamma \vdash s_1 \approx_L s_2\) and \(\Gamma,\text{High} \vdash c\)

shows \(\Gamma \vdash s_1 \approx_L s_2\)

using while

proof (induct rule: while-reds-induct)

case (false s) thus \(?case\ by\ (auto\ simp\ add:lowEquiv-def)\)

next

case (true s1 s2)

from body: \(OF \ (c,s_1) \rightarrow* \langle\text{Skip},s_3'\rangle\) have \(\Gamma \vdash s_1 \approx_L s_3\) by simp

with \(\Gamma \vdash s_3 \approx_L s_2\) 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 \(\Gamma,\text{High} \vdash c\) and \(\langle c,s_1\rangle \rightarrow* \langle\text{Skip},s_2\rangle\)

shows \(\Gamma \vdash s_1 \approx_L s_2\)

— 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: \(s_1\ s_2\) rule: com.induct)

case Skip

from \(\langle\text{Skip},s_1\rangle \rightarrow* \langle\text{Skip},s_2\rangle\) have \(s_1 = s_2\) by (auto dest:Skip-reds)

thus \(?case\ by\ (simp\ add:lowEquiv-def)\)

next

case \(\text{LAss} \ V\ e\)

from \(\Gamma,\text{High} \vdash V:=e\) have \(\Gamma\ V = \text{Some High}\) by (auto elim: secComTyping.cases)

from \(\langle V:=e,s_1\rangle \rightarrow* \langle\text{Skip},s_2'\rangle\) have \(s_2 = s_1(V:= [e]\ s_1)\) by (auto intro:LAss-reds)

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

have \(s_1\ V' = s_2\ V'\)

proof (cases \(V' = V\))

  case True

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

  thus \(?thesis\ by\ simp\)

next

case False

with \(s_2 = s_1(V:= [e]\ s_1)\) show \(?thesis\ by\ simp\)

next
qed
}
thus ?case by (auto simp add: lowEquiv-def)

next
case (Seq c1 c2)
note IH1 = (\A s1 s2. [Γ,High ⊢ c1; {c1,s1} →∗ {Skip,s2}] =⇒ Γ ⊢ s1 ≈L s2)
note IH2 = (\A s1 s2. [Γ,High ⊢ c2; {c2,s1} →∗ {Skip,s2}] =⇒ Γ ⊢ s1 ≈L s2)
from Γ,High ⊢ c1;c2 have Γ,High ⊢ c1 and Γ,High ⊢ c2
by (auto elim:secComTyping.cases)
from ⟨c1;c2,s1⟩ →∗ ⟨Skip,s2⟩ have ∃ s3. ⟨c1,s1⟩ →∗ ⟨Skip,s3⟩ ∧ ⟨c2,s3⟩ →∗ ⟨Skip,s2⟩ by (auto intro:Seq-reds)
then obtain s3 where ⟨c1,s1⟩ →∗ ⟨Skip,s3⟩ and ⟨c2,s3⟩ →∗ ⟨Skip,s2⟩ by auto
from IH1[OF Γ,High ⊢ c1; ⟨c1,s1⟩ →∗ ⟨Skip,s3⟩] have Γ ⊢ s1 ≈L s3 by simp
from IH2[OF Γ,High ⊢ c2; ⟨c2,s3⟩ →∗ ⟨Skip,s2⟩] have Γ ⊢ s3 ≈L s2 by simp
from Γ ⊢ s1 ≈L s3; Γ ⊢ s3 ≈L s2 show ?case
by (auto intro:lowEquivTransitive)

next
case (Cond b c1 c2)
note IH1 = (\A s1 s2. [Γ,High ⊢ c1; {c1,s1} →∗ {Skip,s2}] =⇒ Γ ⊢ s1 ≈L s2)
note IH2 = (\A s1 s2. [Γ,High ⊢ c2; {c2,s1} →∗ {Skip,s2}] =⇒ Γ ⊢ s1 ≈L s2)
from Γ,High ⊢ if (b) c1 else c2 have Γ,High ⊢ c1 and Γ,High ⊢ c2
by (auto elim:secComTyping.cases)
from ⟨if (b) c1 else c2,s1⟩ →∗ ⟨Skip,s2⟩ have [b] s1 = Some true ∨ [b] s1 = Some false by (auto dest:Cond-True-or-False)
thus ?case

proof
assume [b] s1 = Some true
with ⟨if (b) c1 else c2,s1⟩ →∗ ⟨Skip,s2⟩ have ⟨c1,s1⟩ →∗ ⟨Skip,s2⟩
by (auto intro:CondTrue-reds)
from IH1[OF Γ,High ⊢ c1; this] show ?thesis .

next
assume [b] s1 = Some false
with ⟨if (b) c1 else c2,s1⟩ →∗ ⟨Skip,s2⟩ have ⟨c2,s1⟩ →∗ ⟨Skip,s2⟩
by (auto intro:CondFalse-reds)
from IH2[OF Γ,High ⊢ c2; this] show ?thesis .

qed

next
case (While b c’)
note IH = (\A s1 s2. [Γ,High ⊢ c’; {c’,s1} →∗ {Skip,s2}] =⇒ Γ ⊢ s1 ≈L s2)
from Γ,High ⊢ while (b) c’ have Γ,High ⊢ c’ by (auto elim:secComTyping.cases)
from IH[OF this] have \A s1 s2. [(c’,s1) →∗ {Skip,s2}] =⇒ Γ ⊢ s1 ≈L s2 .
with ⟨while (b) c’,s1⟩ →∗ ⟨Skip,s2⟩; Γ,High ⊢ c’ show ?case by (auto dest:WhileStepInduct)

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

lemma CondLowCompositionality:  
assumes \text{nonInterference } Γ \, c1 \text{ and } \text{nonInterference } Γ \, c2 \text{ and } Γ \vdash b : \text{Low}  
shows \text{nonInterference } Γ \, (if \, (b) \, c1 \text{ else } c2)  
proof (rule nonInterferenceI)  
  fix \ s1 \ s2 \ s1' \ s2'  
  assume Γ \vdash s1 \approx_{L} s2 \text{ and } \langle if \, (b) \, c1 \text{ else } c2, s1 \rangle \rightarrow^{*} \langle \text{Skip}, s1' \rangle  
  and \langle if \, (b) \, c1 \text{ else } c2, s2 \rangle \rightarrow^{*} \langle \text{Skip}, s2' \rangle  
  from ⟨\text{if } (b) \, \text{Low}; Γ \vdash s1 \approx_{L} s2⟩ \; \text{have } [b] \, s1 = [b] \, s2  
  by (auto intro:interpretLow2)  
  from ⟨\langle if \, (b) \, c1 \text{ else } c2, s1 \rangle \rightarrow^{*} \langle \text{Skip}, s1' \rangle⟩  
  have [b] \, s1 = \text{Some true } \vee [b] \, s1 = \text{Some false } \text{by (auto dest:Cond-True-or-False)}  
  thus Γ \vdash s1' \approx_{L} s2'  
  proof  
  
  assume [b] \, s1 = \text{Some true }  
  with [b] \, s1 = [b] \, s2 \; \text{have } [b] \, s2 = \text{Some true } \text{by (auto intro:CondTrue-reds)}  
  from [b] \, s1 = \text{Some true } \langle if \, (b) \, c1 \text{ else } c2, s1 \rangle \rightarrow^{*} \langle \text{Skip}, s1' \rangle  
  have ⟨c1, s1⟩ \rightarrow^{*} \langle \text{Skip}, s1' \rangle \text{ by (auto intro:CondTrue-reds)}  
  from [b] \, s2 = \text{Some true } \langle if \, (b) \, c1 \text{ else } c2, s2 \rangle \rightarrow^{*} \langle \text{Skip}, s2' \rangle  
  have ⟨c1, s2⟩ \rightarrow^{*} \langle \text{Skip}, s2' \rangle \text{ by (auto intro:CondTrue-reds)}  
  with ⟨c1, s1⟩ \rightarrow^{*} \langle \text{Skip}, s1' \rangle, Γ \vdash s1 \approx_{L} s2, \text{nonInterference } Γ \, c1⟩  
  show \text{?thesis by (auto simp:nonInterference-def)}  

next  
  assume [b] \, s1 = \text{Some false }  
  with [b] \, s1 = [b] \, s2 \; \text{have } [b] \, s2 = \text{Some false } \text{by (auto intro:CondTrue-reds)}  
  from [b] \, s1 = \text{Some false } \langle if \, (b) \, c1 \text{ else } c2, s1 \rangle \rightarrow^{*} \langle \text{Skip}, s1' \rangle  
  have ⟨c2, s1⟩ \rightarrow^{*} \langle \text{Skip}, s1' \rangle \text{ by (auto intro:CondFalse-reds)}  
  from [b] \, s2 = \text{Some false } \langle if \, (b) \, c1 \text{ else } c2, s2 \rangle \rightarrow^{*} \langle \text{Skip}, s2' \rangle  

20
have \((c_2,s_2) \rightarrow^* \langle \text{Skip}, s_2' \rangle\) by (auto intro: CondFalse-reds)
with \((c_2,s_1) \rightarrow^* \langle \text{Skip}, s_1' \rangle\); \(\Gamma \vdash s_1 \approx_L s_2\); \langle \text{nonInterference} \ \Gamma \ \ c_2 \rangle
show \(\text{thesis}\) by (auto simp: nonInterference_def)
qed

lemma WhileHighCompositionality:
assumes \(\Gamma, \text{High} \vdash \text{while} \ (b) \ c'\)
shows \(\text{nonInterference} \ \Gamma \ \ (\text{while} \ (b) \ c')\)
proof (rule nonInterferenceI)
fix \(s_1 \ \ s_2 \ \ s_1' \ \ s_2'\)
assume \(\Gamma \vdash s_1 \approx_L s_2\) and \(\langle \text{while} \ (b) \ c', s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle\)
and \(\langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle\)
show \(\Gamma \vdash s_1' \approx_L s_2'\)
proof
–
from \(\Gamma, \text{High} \vdash \text{while} \ (b) \ c' \langle \text{while} \ (b) \ c', s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle\)
have \(\Gamma \vdash s_1 \approx_L s_1'\) by (auto dest: highBodies)
from \(\Gamma, \text{High} \vdash \text{while} \ (b) \ c' \langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle\)
have \(\Gamma \vdash s_2 \approx_L s_2'\) by (auto dest: highBodies)
with \(\Gamma \vdash s_1 \approx_L s_2\) have \(\Gamma \vdash s_1' \approx_L s_2'\) by (auto intro: low EQUIV Transitive)
from \(\Gamma \vdash s_1 \approx_L s_1'\) have \(\Gamma \vdash s_1' \approx_L s_1\) by (auto intro: low EQUIV Symmetric)
with \(\Gamma \vdash s_1 \approx_L s_2\) show \(\text{thesis}\) by (auto intro: low EQUIV Transitive)
qed

lemma WhileLowStepInduct:
assumes \(\text{while1} : \langle \text{while} \ (b) \ c', s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle\)
and \(\text{while2} : \langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle\)
and \(\Gamma \vdash b : \text{Low}\)
and \(\text{body} : \langle s_1 \ \ s_1' \ \ s_2 \ \ s_2' \rangle ; \langle c', s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle; \langle c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle; \Gamma \vdash s_1 \approx_L s_2\)
\(\Rightarrow\) \(\Gamma \vdash s_1' \approx_L s_2'\)
and \(\text{le} : \Gamma \vdash s_1 \approx_L s_2\)
shows \(\Gamma \vdash s_1' \approx_L s_2'\)
using \(\text{while1 le while2}\)
proof (induct arbitrary: \(s_2\) rule: while-reds-induct)
case (false \(s_1\))
from \(\Gamma \vdash b : \text{Low}; \Gamma \vdash s_1 \approx_L s_2\) have \([b] \ s_1 = [b] \ s_2\) by (auto intro: interpretLow2)
with \([b] \ s_1 = \text{Some false}\) have \([b] \ s_2 = \text{Some false}\) by simp
with \(\langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle\) have \(s_2 = s_2'\) by (auto intro: WhileFalse-reds)
with \(\Gamma \vdash s_1 \approx_L s_2\) show \(\text{thesis}\) by auto
next
case (true \(s_1 \ s_1'\))
note \(\text{IH} = \langle s_2'' \rangle; \Gamma \vdash s_1'' \approx_L s_2''; \langle \text{while} \ (b) \ c', s_2'' \rangle \rightarrow^* \langle \text{Skip}, s_2'' \rangle\)
\(\Rightarrow\) \(\Gamma \vdash s_1' \approx_L s_2'\)
from \(\Gamma \vdash b : \text{Low}; \Gamma \vdash s_1 \approx_L s_2\) have \([b] \ s_1 = [b] \ s_2\) by (auto intro: interpretLow2)
with \([b] \ s_1 = \text{Some true}\) have \([b] \ s_2 = \text{Some true}\) by simp
with \( \langle \text{while} (b) \ c',s2' \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) obtain \( s2'' \) where \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \) and \( \langle \text{while} (b) \ c',s2'' \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \) by(auto dest:WhileTrue-reds)
from body[OF \( \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \); \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \)]
have \( \Gamma \vdash s1' \approx_L s2'' \).
from IH[OF this \( \langle \text{while} (b) \ c',s2'' \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \)] 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 \( \langle \text{while} (b) \ c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) and \( \langle \text{while} (b) \ c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
\{ fix \( s1 \ s2 \ s1'' \ s2'' \)
assume \( \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) and \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \)
and \( \Gamma \vdash s1 \approx_L s2 \)
with nonInterference \( \Gamma \ c' \) have \( \Gamma \vdash s1'' \approx_L s2'' \)
by(auto simp:nonInterference-def)
\}

hence \( \forall s1 \ s2'' s2''. \ [ \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle ; \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle ;
\Gamma \vdash s1 \approx_L s2 ] 
\Rightarrow \Gamma \vdash s1'' \approx_L s2'' \) by auto
with \( \langle \text{while} (b) \ c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \); \( \langle \text{while} (b) \ c',s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \)
\( \Gamma \vdash b : \text{Low} \) \( \Gamma \vdash s1 \approx_L s2 \) show \( \Gamma \vdash s1' \approx_L s2' \)
by(auto intro:WhileLowStepInduct)
qed

and now: the main theorem:

theorem seeTypeImpliesNonInterference:
\( \Gamma, T \vdash c \Longrightarrow \text{nonInterference} \ \Gamma \ c \)
proof (induct c arbitrary:T rule:com.induct)
case \text{Skip}
show \( ?\text{case} \)
proof(rule nonInterferenceI)
fix \( s1 \ s2 \ s1' \ s2' \)
assume \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle \text{Skip},s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) and \( \langle \text{Skip},s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
from \( \langle \text{Skip},s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) have \( s1 = s1' \) by(auto dest:Skip-reds)
from \( \langle \text{Skip},s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) have \( s2 = s2' \) by(auto dest:Skip-reds)
from \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle s1 = s1' \rangle \) and \( \langle s2 = s2' \rangle \)
show \( \Gamma \vdash s1' \approx_L s2' \) by simp
qed
next
case \( \text{LA} s \ V \ e \)
from \( \Gamma, T \vdash V := e \)
have \( \text{varprem} : (\Gamma \ V = \text{Some High}) \lor (\Gamma \ V = \text{Some Low} \land \Gamma \vdash e : \text{Low} \land T = \text{Low}) \)
by(auto elim:seeComTyping.cases)
show ?case
proof (rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume Γ ⊢ s1 ≈_L s2 and ⟨V:=e,s1⟩ →* ⟨Skip,s1'⟩ and ⟨V:=e,s2⟩ →*
⟨Skip,s2'⟩
  from ⟨V:=e,s1⟩ →* ⟨Skip,s1'⟩ have s1' = s1(V:=e) s1 by(auto intro:LAss-reds)
  from ⟨V:=e,s2⟩ →* ⟨Skip,s2'⟩ have s2' = s2(V:=e) s2 by(auto intro:LAss-reds)
  from varprem show Γ ⊢ s1' ≈_L s2'
proof
  assume Γ V = Some High
  with Γ ⊢ s1 ≈_L s2 (s1' = s1(V:=e) s1) (s2' = s2(V:=e) s2)
  show ?thesis by(auto intro:assignNIhighlemma)
next
  assume Γ V = Some Low ∧ Γ ⊢ e : Low ∧ T = Low
  with Γ ⊢ s1 ≈_L s2 (s1' = s1(V:=e) s1) (s2' = s2(V:=e) s2)
  show ?thesis by(auto intro:assignNIlowlemma)
qed
next
  case (Seq c1 c2)
  note IH1 = ⟨∧T. Γ,T ⊢ c1 ⟹ nonInterference Γ c1⟩
  note IH2 = ⟨∧T. Γ,T ⊢ c2 ⟹ nonInterference Γ c2⟩
  show ?case
proof (cases T)
  case High
  with Γ,T ⊢ c1;;c2 have Γ,High ⊢ c1 and Γ,High ⊢ c2
  by(auto elim:secComTyping.cases)
  from IH1[OF Γ,High ⊢ c1] have nonInterference Γ c1 .
  moreover
  from IH2[OF Γ,High ⊢ c2] have nonInterference Γ c2 .
  ultimately show ?thesis by (auto intro:SeqCompositionality)
next
  case Low
  with Γ,T ⊢ c1;;c2 have (Γ,Low ⊢ c1 ∧ Γ,Low ⊢ c2) ∨ Γ,High ⊢ c1;;c2
  by(auto elim:secComTyping.cases)
  thus ?thesis
proof
  assume Γ,Low ⊢ c1 ∧ Γ,Low ⊢ c2
  hence Γ,Low ⊢ c1 and Γ,Low ⊢ c2 by simp-all
  from IH1[OF Γ,Low ⊢ c1] have nonInterference Γ c1 .
  moreover
  from IH2[OF Γ,Low ⊢ c2] have nonInterference Γ c2 .
  ultimately show ?thesis by(auto intro:SeqCompositionality)
next
  assume Γ,High ⊢ c1;;c2
  hence Γ,High ⊢ c1 and Γ,High ⊢ c2 by(auto elim:secComTyping.cases)

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

next case (Cond b c1 c2)

note IH1 = \( \langle \forall T. \Gamma, T \vdash c1 \Rightarrow \text{nonInterference} \Gamma \vdash c1 \rangle \)

note IH2 = \( \langle \forall T. \Gamma, T \vdash c2 \Rightarrow \text{nonInterference} \Gamma \vdash c2 \rangle \)

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 \( (\Gamma \vdash b : \text{Low} \land \Gamma, \text{Low} \vdash c1 \land \Gamma, \text{Low} \vdash c2) \lor \Gamma, \text{High} \vdash \text{if} (b) c1 \text{ else } c2 \)

by(auto elim:secComTyping.cases)

thus \(?\)thesis proof

assume \( \Gamma \vdash b : \text{Low} \land \Gamma, \text{Low} \vdash c1 \land \Gamma, \text{Low} \vdash c2 \)

hence \( \Gamma \vdash b : \text{Low} \land \Gamma, \text{Low} \vdash c1 \land \Gamma, \text{Low} \vdash c2 \) by simp-all

from IH1[OF \( \text{OF} \Gamma, \text{Low} \vdash c1 \)] have nonInterference \( \Gamma \vdash c1 \).

moreover

from IH2[OF \( \Gamma, \text{Low} \vdash c2 \)] have nonInterference \( \Gamma \vdash c2 \).

ultimately show \(?\)thesis using \( \Gamma \vdash b : \text{Low} \)

by(auto intro:CondLowCompositionality)

next

assume \( \Gamma, \text{High} \vdash \text{if} (b) c1 \text{ else } c2 \)

thus \(?\)thesis by(auto intro:CondHighCompositionality)

qed

next

next case (While b c')

note IH = \( \langle \forall T. \Gamma, T \vdash c' \Rightarrow \text{nonInterference} \Gamma \vdash c' \rangle \)

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 \( (\Gamma \vdash b : \text{Low} \land \Gamma, \text{Low} \vdash c') \lor \Gamma, \text{High} \vdash \text{while} (b) c' \)

by(auto elim:secComTyping.cases)

thus \(?\)thesis proof

assume \( \Gamma \vdash b : \text{Low} \land \Gamma, \text{Low} \vdash c' \)
hence \( \Gamma \vdash b : \text{Low} \) and \( \Gamma, \text{Low} \vdash c' \) by simp-all

moreover

from IH[\( \text{OF } \Gamma, \text{Low} \vdash c' \)] 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: \( i \Rightarrow o \Rightarrow \text{bool} \) as exec-red, \( i \Rightarrow i \Rightarrow o \Rightarrow \text{bool} \) as compute-secExprTyping, \( i \Rightarrow i \Rightarrow i \Rightarrow \text{bool} \) as check-secExprTyping) secExprTyping

proof

− case secExprTyping

from secExprTyping.prems show thesis

proof

fix \( \Gamma \) \( V \) \( \text{lev} \) assume \( x = \Gamma \ xa = \text{Val } V \ xb = \text{lev} \)

from secExprTyping(1-2) this show thesis by (cases \text{lev}) auto

next

fix \( \Gamma \) \( V \text{n} \) \( \text{lev} \)

assume \( x = \Gamma \ xa = \text{Var } V \text{n} \ xb = \text{lev} \) \( \Gamma \) \( V \text{n} = \text{Some } \text{lev} \)

from secExprTyping(3) this show thesis by (auto simp add: Predicate.eq-is-eq)

next

fix \( \Gamma \) \( e_1 \) \( e_2 \) \( \text{bop} \)

assume \( x = \Gamma \ xa = e_1 \text{<bop}> e_2 \ xb = \text{Low} \)

\( \Gamma \vdash e_1 : \text{Low} \) \( \Gamma \vdash e_2 : \text{Low} \)

from secExprTyping(4) this show thesis by auto

next
\[\begin{align*}
&\text{fix } \Gamma e_1 e_2 \text{ lev bop} \\
&\text{assume } x = \Gamma x a = e_1 \text{ bop } e_2 x b = \text{High} \\
&\Gamma \vdash e_1 : \text{High } \Gamma \vdash e_2 : \text{lev} \\
&\text{from } \text{secExprTyping}(5-6) \text{ this show thesis by (cases lev) (auto)} \\
&\text{next} \\
&\text{fix } \Gamma e_1 e_2 \text{ lev bop} \\
&\text{assume } x = \Gamma x a = e_1 \text{ bop } e_2 x b = \text{High} \\
&\Gamma \vdash e_1 : \text{lev } \Gamma \vdash e_2 : \text{High} \\
&\text{from } \text{secExprTyping}(6-7) \text{ this show thesis by (cases lev) (auto)} \\
&\text{next} \\
&\text{lemmas [code-pred-intro] = typeSkip[where } T=\text{Low} \text{ typeSkip[where } T=\text{High}] \\
&\text{typeAssH[where } T=\text{Low} \text{ typeAssH[where } T=\text{High}] \\
&\text{typeAssL typeSeq typeWhile typeIf typeConvert} \\
&\text{code-pred (modes: } i => o => i => \text{bool as compute-secComTyping,} \\
&\quad i => i => i => \text{bool as check-secComTyping) secComTyping} \\
&\text{proof} - \\
&\quad \text{case secComTyping} \\
&\quad \text{from } \text{secComTyping.prems show thesis} \\
&\quad \text{proof} \\
&\quad \quad \text{fix } \Gamma T \text{ assume } x = \Gamma x a = T x b = \text{Skip} \\
&\quad \quad \text{from } \text{secComTyping}(1-2) \text{ this show thesis by (cases } T \text{) auto} \\
&\quad \text{next} \\
&\quad \quad \text{fix } \Gamma V T e \text{ assume } x = \Gamma x a = T x b = V := e \Gamma V = \text{Some High} \\
&\quad \quad \text{from } \text{secComTyping}(3-4) \text{ this show thesis by (cases } T \text{) (auto)} \\
&\quad \text{next} \\
&\quad \quad \text{fix } \Gamma e V \text{ assume } x = \Gamma x a = \text{Low x b = V := e } \Gamma V = \text{Some Low} \\
&\quad \quad \text{from } \text{secComTyping}(5) \text{ this show thesis by auto} \\
&\quad \text{next} \\
&\quad \quad \text{fix } \Gamma T c_1 c_2 \text{ assume } x = \Gamma x a = T x b = \text{Seq c_1 c_2 } \Gamma, T \vdash c_1 \Gamma, T \vdash c_2 \\
&\quad \quad \text{from } \text{secComTyping}(6) \text{ this show thesis by auto} \\
&\quad \text{next} \\
&\quad \quad \text{fix } \Gamma b T c \text{ assume } x = \Gamma x a = T x b = \text{while } (b) c \text{ } \Gamma \vdash b : T \Gamma, T \vdash c \\
&\quad \quad \text{from } \text{secComTyping}(7) \text{ this show thesis by auto} \\
&\quad \text{next} \\
&\quad \quad \text{fix } \Gamma b T c_1 c_2 \text{ assume } x = \Gamma x a = T x b = \text{if } (b) c_1 \text{ else } c_2 \text{ } \Gamma \vdash b : T \Gamma, T \vdash c_1 \Gamma, T \vdash c_2 \\
&\quad \quad \text{from } \text{secComTyping}(8) \text{ this show thesis by blast} \\
&\quad \text{next} \\
&\quad \quad \text{fix } \Gamma c \text{ assume } x = \Gamma x a = \text{Low x b = e } \Gamma, \text{High } \vdash c \\
&\quad \quad \text{from } \text{secComTyping}(9) \text{ this show thesis by blast} \\
&\text{qed} \\
&\text{qed} \\
\end{align*}\]
3.1 An example taken from Volpano, Smith, Irvine

\begin{verbatim}
\begin{definition}
com = if (Var "x" ≪ Eq Val (Intg 1)) ("y" := Val (Intg 1)) else ("y" := Val (Intg 0))
\end{definition}

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

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

\begin{value}
Env, High ⊢ com
\end{value}

\begin{values}
1 \{ T. Env, T ⊢ com \}
\end{values}

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

\begin{value}
Env', Low ⊢ com
\end{value}

\begin{value}
Env', High ⊢ com
\end{value}

\begin{values}
1 \{ T. Env, T ⊢ com \}
\end{values}

\end{verbatim}

References
