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

Gregor Snelting and Daniel Wasserrab
IPD Snelting
Universität Karlsruhe (TH)
August 16, 2018

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 (−<−) − [80,0,81] 80 — binary operation

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

fun binop :: bop ⇒ val ⇒ val ⇒ val option
where
  binop Eq v1 v2 = Some(Bool(v1 = v2))
  | binop And (Bool b1) (Bool b2) = Some(Bool(b1 ∧ b2))
  | binop Less (Intg i1) (Intg i2) = Some(Bool(i1 < i2))
  | binop Add (Intg i1) (Intg i2) = Some(Intg(i1 + i2))
  | binop Sub (Intg i1) (Intg i2) = Some(Intg(i1 - i2))
  | binop bop v1 v2 = Some(Intg(0))

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

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

type-synonym state = vname ⇒ val

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

fun interpret :: expr ⇒ state ⇒ val option ([-] -)
where Val: [Val v] s = Some v
| Var: [Var V] s = s V
| BinOp: [e1 ≪ bop ≫ e2] s = (case [e1] s of None ⇒ None
  | Some v1 ⇒ (case [e2] s of None ⇒ None
  | Some v2 ⇒ binop bop v1 v2))

1.4 Small Step Semantics

inductive red :: com * state ⇒ com * state ⇒ bool
and red’ :: com ⇒ state ⇒ com ⇒ state ⇒ bool
(((1⟨-,/-⟩) →/ (1⟨-,/-⟩)) [0,0,0,0] 81)
where
  ⟨c1,s1⟩ → ⟨c2,s2⟩ == red (c1,s1) (c2,s2) |
RedLAss:
  ⟨V:=e,s⟩ → ⟨Skip,s(V:=[e] s))⟩
  | SeqRed:
    ⟨c1,s⟩ → ⟨c1′,s⟩ ⇒⇒ ⟨c1;;c2,s⟩ → ⟨c1′;;c2,s⟩
  | RedSeq:
    ⟨Skip;;c2,s⟩ → ⟨c2,s⟩
  | RedCondTrue:
    [b] s = Some true ⇒⇒ ⟨if (b) c1 else c2,s⟩ → ⟨c1,s⟩
  | RedCondFalse:
    [b] s = Some false ⇒⇒ ⟨if (b) c1 else c2,s⟩ → ⟨c2,s⟩
  | RedWhileTrue:
    [b] s = Some true ⇒⇒ ⟨while (b) c,s⟩ → ⟨c;;while (b) c,s⟩
  | RedWhileFalse:
    [b] s = Some false ⇒⇒ ⟨while (b) c,s⟩ → ⟨Skip,s⟩

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

abbreviation reds ::com ⇒ state ⇒ com ⇒ state ⇒ bool
    ((((1⟨-,/-⟩) →*/ (1⟨-,/-⟩)) [0,0,0,0] 81) where
    ⟨c,s⟩ →*/ ⟨c′,s′⟩ == red** (c,s) (c′,s′)
lemma \textit{Skip-reds}:
\[
\langle \text{Skip}, s \rangle \rightarrow^* (c', s') \quad \text{implies} \quad s = s' \land c' = \text{Skip}
\]
by (blast elim: converse-rtranclpI red.cases)

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

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

lemma \textit{Seq-reds}:
\[
\text{assumes} \quad \langle c_1; c_2, s \rangle \rightarrow^* (\text{Skip}, s')
\]
\[
\text{obtains} \quad s'' \text{ where} \quad \langle c_1, s \rangle \rightarrow^* (\text{Skip}, s'') \quad \text{and} \quad \langle c_2, s'' \rangle \rightarrow^* (\text{Skip}, s')
\]
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 \(IH = \langle \forall c_1. \ [c'' = c_1; c_2; c' = \text{Skip}] \Rightarrow \exists sx. \ (c_1, s'') \rightarrow^* (\text{Skip}, sx) \land (c_2, sx) \rightarrow^* (\text{Skip}, s')\)
      from step
      have \(\langle c_1; c_2, s \rangle \rightarrow (c'', s'')\) by simp
      hence \(\langle c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \rangle \lor \langle \exists c_1'. \ \langle c_1, s \rangle \rightarrow (c_1', s'') \land c'' = c_1' ; c_2 \rangle\)
      by (auto elim: red.cases)
      thus \(?case\)
    proof
      assume \(c_1 = \text{Skip} \land c'' = c_2 \land s = s''\)
      with \((c'', s'') \rightarrow^* (c', s')\) \((c' = \text{Skip})\)
      show \(\text{thesis}\) 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 \(IH[OFF \ (c'' = c_1' ; c_2; c' = \text{Skip})]\)
      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 \(\langle c_1, s \rangle \rightarrow (c_1', s'')\) \((c_1', s'') \rightarrow^* (\text{Skip}, sx)\)
have \((c_1, s) \rightarrow^* (\text{Skip}, s')\) by\((auto \ intro:converse-rtranclp-into-rtranclp)\)
with \(((c_2, s) \rightarrow^* (\text{Skip}, s'))\) show \(\text{thesis}\) by auto
qed

\textbf{lemma} \textit{Cond-True-or-False}:
\[(\text{if} \ (b) \ c_1 \ \text{else} \ c_2, s) \rightarrow^* (\text{Skip}, s') \Rightarrow [b] \ s = \text{Some \ true} \lor [b] \ s = \text{Some \ false}\]
by\((\text{induct} \ c = \text{if} \ (b) \ c_1 \ \text{else} \ c_2 \ s \ \text{rule:} converse-rtranclp-induct2)(\text{auto \ elim:} \text{red.cases})\)

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

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

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

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

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

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

\textbf{lemma} \textit{Seq-red-nE}: assumes \(\langle c_1, \cdot; c_2, s \rangle \rightarrow^\text{n} \langle \text{Skip}, s' \rangle\)
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 \( \langle e_1; e_2, s \rangle \rightarrow^n \langle \text{Skip}, s \rangle \)
have \( \exists i j s'. \langle e_1, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle \land \langle e_2, s' \rangle \rightarrow^j \langle \text{Skip}, s \rangle \land n = i + j + 1 \)
proof
\begin{enumerate}
\item \( \text{induct } c_1; c_2 \ s n \text{ Skip } s' \text{ arbitrary;} c_1 \text{ rule:red-n-induct) } \)
\item \( \text{case (red-n-Rec } s c' s'' n s') \)
\item \( \text{note } IH = \langle \land \rangle c_1. c'' = c_1; c_2 \)
\item \( \implies \exists i j s. \langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s \rangle \land \langle e_2, s \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1 \)
\item \( \text{from } \langle e_1; e_2, s \rangle \rightarrow \langle e''', s'' \rangle \)
\item \( \text{have } \langle c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \rangle \lor \langle \exists c_1'. c'' = c_1; c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \rangle \)
\end{enumerate}
by(\( \text{induct } c_1; c_2 \) - - rule:red-induct) auto
thus ?case
proof
\begin{enumerate}
\item \( \text{assume } c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \)
\item \( \text{hence } c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \text{ by simp-all) } \)
\item \( \text{from } \langle c_1 = \text{Skip} \rangle \text{ have } \langle c_1, s \rangle \rightarrow^0 \langle \text{Skip}, s \rangle \text{ by (fastforce intro:red-n-Base) } \)
\item \( \text{with } \langle c''', s'' \rangle \rightarrow^n \langle \text{Skip}, s'' \rangle ; \langle c'' = c_2 \rangle (s = s'') \)
\item \( \text{show } \langle \text{thesis by rule-tac } x = 0 \text{ in exf) auto } \rangle \)
\end{enumerate}
next
\begin{enumerate}
\item \( \text{assume } \exists c_1'. c'' = c_1'; c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \)
\item \( \text{then obtain } c_1' \text{ where } c'' = c_1'; c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \text{ by blast) } \)
\item \( \text{from } IH[\langle c'' = c_1'; c_2 \rangle \text{ obtain } i j s \)
\item \( \text{where } \langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle \land \langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s'' \rangle \)
\item \( \text{and } n = i + j + 1 \text{ by blast) } \)
\item \( \text{from } \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \land \langle c_1', s'' \rangle \rightarrow^j \langle \text{Skip}, sx \rangle \)
\item \( \text{have } \langle c_1, s \rangle \rightarrow \text{Suc } i \langle \text{Skip}, sx \rangle \text{ by (rule red-n.red-n-Rec) } \)
\item \( \text{with } \langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s'' \rangle ; n = i + j + 1 \text{ show } \langle \text{thesis by rule-tac } x = \text{Suc } i \text{ in exf) auto } \rangle \)
\end{enumerate}
qed
with that show \( \langle \text{thesis by blast) } \rangle \)
qed

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

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 \(\langle c'', s'' \rangle \rightarrow^n (\text{Skip}, s')\) 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 \(?\text{thesis by fastforce}\)
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 \(\langle c'', s'' \rangle \rightarrow^n (\text{Skip}, s')\)
 obtain \(i\) \(j\) \(s\) \(x\) where \(\langle cx, s \rangle \rightarrow^i (\text{Skip}, sx)\) and \(\langle while (b) cx, sx \rangle \rightarrow^j (\text{Skip}, s')\)
and \(n = i + j + 1\) by(fastforce elim:Seq-red-nE)
with \([b]\) \(s = \text{Some true}\) show \(?\text{thesis by fastforce}\)
qed
qed

lemma while-red-n-induct [consumes 1, case-names false true]:
assumes major: \(\langle while (b) cx, s \rangle \rightarrow^n (\text{Skip}, s')\)
and IHfalse:\(\forall s. \ [b]\) \(s = \text{Some false} \implies P \ s \ s\)
and IHtrue:\(\forall i \ j \ s'\). \ [b]\) \(s = \text{Some true}; \ \langle cx, s \rangle \rightarrow^i (\text{Skip}, s'');\)
\(\langle while (b) cx, s'' \rangle \rightarrow^j (\text{Skip}, s'); P \ s'' \ s' \implies P \ s \ s'\)
shows \(P \ s \ s'\)
using major
proof (induct \(n\) arbitrary:s rule:nat-less-induct)
fix \(n\) \(s\)
assume IHall: \(\forall m<n. \ \forall x. \ \langle while (b) cx, x \rangle \rightarrow^m (\text{Skip}, s') \implies P \ x \ s'\)
and \(\langle while (b) cx, s \rangle \rightarrow^n (\text{Skip}, s')\)
from \(\langle while (b) cx, s \rangle \rightarrow^n (\text{Skip}, s')\)
have \([b]\) \(s = \text{Some false} \land s = s' \land n = 1\) \(\lor\)
\((\exists i \ j \ s''). \ [b]\) \(s = \text{Some true}; \ \langle cx, s \rangle \rightarrow^i (\text{Skip}, s'');\)
\(\langle while (b) cx, s'' \rangle \rightarrow^j (\text{Skip}, s') \land n = i + j + 2\)
by(rule while-red-nE)
thus \(P \ s \ s'\)
proof
assume IHall: \(\forall m<n. \ \forall x. \ \langle while (b) cx, x \rangle \rightarrow^m (\text{Skip}, s') \implies P \ x \ s'\)
and \(\langle while (b) cx, s \rangle \rightarrow^n (\text{Skip}, s')\)
from IHfalse[OF \([b]\) \(s = \text{Some false}\)] have \(P \ s \ s\).
with \(s = s'\) show \(?\text{thesis by simp}\)
next
assume \(\exists i \ j \ s''. \ [b]\) \(s = \text{Some true} \land \langle cx, s \rangle \rightarrow^i (\text{Skip}, s'') \land \langle while (b) cx, s'' \rangle \rightarrow^j (\text{Skip}, s') \land n = i + j + 2\)
then obtain \(i\) \(j\) \(s''\) where \([b]\) \(s = \text{Some true}\)
and \(\langle cx, s \rangle \rightarrow^i (\text{Skip}, s'')\) and \(\langle while (b) cx, s'' \rangle \rightarrow^j (\text{Skip}, s')\)
and \(n = i + j + 2\) by blast
with IHall have \(P \ s'' \ s'\)
apply(crule-tac \(x=j\) in allE) apply clarsimp done
from IHtrue[OF \([b]\) \(s = \text{Some true}\)] \(\langle cx, s \rangle \rightarrow^i (\text{Skip}, s'')\)
\langle \text{while} \ (b) \ cx, s' \rangle \rightarrow^3 \langle \text{Skip}, s' \rangle \text{ this} \ \text{show} \ ?\text{thesis}.

\text{qed}

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

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

\text{lemma while-reds-induct[consumes 1, case-names false true]:}
\langle \text{while} \ (b) \ cx, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle; \ \langle s s' \rangle \Longrightarrow P \ s \ s;
\langle s s'' s'' \rangle \Longrightarrow P \ s \ s'
\text{apply (erule 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:}
\langle c, s \rangle \rightarrow \langle c_1, s_1 \rangle; \ \langle c, s \rangle \rightarrow \langle c_2, s_2 \rangle \Longrightarrow c_1 = c_2 \land s_1 = s_2
\text{proof (induct arbitrary;c rule:red-induct)}
\text{case (SeqRed c_1 s c_1' s_1' c_2')}
\text{note IH = } \langle \text{while} (b) \ cx, s \rangle \rightarrow \langle c_2, s_2 \rangle \Longrightarrow c_1' = c_2 \land s' = s_2
\text{from } \langle c_1', c_2', s \rangle \rightarrow \langle c_2, s_2 \rangle \text{ have } c_1 = \text{Skip} \lor \exists \ cx. \ c_2 = cx::c_2' \land \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle
\text{by (fastforce elim:red.cases)}
\text{thus } ?\text{case}
\text{proof}
\text{assume } c_1 = \text{Skip}
\text{with } \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle \text{ have False by (fastforce elim:red.cases)}
\text{thus } ?\text{thesis by simp}
\text{next}
\text{assume } \exists cx. \ c_2 = cx::c_2' \land \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle
\text{then obtain } cx \text{ where } c_2 = cx::c_2' \text{ and } \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle \text{ by blast}
\text{from IH[OF } \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle]\text{ have } c_1' = \text{cx} \land s' = s_2, \text{ with } \langle c_2 = cx::c_2' \rangle \text{ show } ?\text{thesis by simp}
\text{qed}

\text{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 \Longrightarrow s_1 = s_2
\text{proof (induct rule: converse-rtranclp-induct2)}
\text{case refl}
from ⟨(Skip,s₁) →∗ (Skip,s₂)⟩ show ?case
  by ¬(erule converse-rtranclpE,auto elim:red.cases)
next
case (step c'' s'' c' s')
  note IH = ⟨⟨c'',s''⟩→∗ ⟨c',s'⟩⟩
  from step have ⟨⟨c'',s''⟩→ ⟨c',s'⟩⟩
    by simp
  from ⟨⟨c'',s''⟩→∗ ⟨Skip,s₂⟩⟩ this have ⟨⟨c',s'⟩→∗ ⟨Skip,s₂⟩⟩
    by ¬(erule converse-rtranclpE,auto elim:red.cases dest:red-det)
  from IH[OF this] show ?thesis .
qed

end
theory secTypes
imports Semantics
begin

2 Security types

2.1 Security definitions
datatype secLevel = Low | High
type-synonym typeEnv = vname ⇒ secLevel

inductive secExprTyping :: typeEnv ⇒ expr ⇒ secLevel ⇒ bool (- |- : -)
where typeVal: Γ ⊢ Val V : lev
| typeVar: Γ Vn = Some lev ⇒ Γ ⊢ Var Vn : lev
| typeBinOp1: Γ ⊢ e1 : Low; Γ ⊢ e2 : Low ⇒ Γ ⊢ e1 ≪bop≫ e2 : Low
| typeBinOp2: Γ ⊢ e1 : High; Γ ⊢ e2 : lev ⇒ Γ ⊢ e1 ≪bop≫ e2 : High
| typeBinOp3: Γ ⊢ e1 : lev; Γ ⊢ e2 : High ⇒ Γ ⊢ e1 ≪bop≫ e2 : High

inductive secComTyping :: typeEnv ⇒ secLevel ⇒ com ⇒ bool (-,- |- -)
where typeSkip: Γ,T ⊢ Skip
| typeAssH: Γ V = Some High ⇒ Γ,T ⊢ V := e
| typeAssL: Γ ⊢ e : Low; Γ V = Some Low ⇒ Γ,Low ⊢ V := e
| typeSeq: Γ,T ⊢ c1; Γ,T ⊢ c2 ⇒ Γ,T ⊢ c1;;c2
| typeWhile: Γ ⊢ b : T; Γ,T ⊢ c ⇒ Γ,T ⊢ while (b) c
\( \text{typeIf:} \quad \left[ \Gamma \vdash b : T ; \Gamma, T \vdash c_1 ; \Gamma, T \vdash c_2 \right] \implies \Gamma, T \vdash \text{if} \ (b \ \text{c_1 else c_2} \)

\( \text{typeConvert:} \quad \Gamma, \text{High} \vdash c \implies \Gamma, \text{Low} \vdash c \)

2.2 Lemmas concerning expressions

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

**proof**

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

**proof (induct e)**

**case** \( \text{Val V} \)
have \( \Gamma \vdash \text{Val V} : \text{Low} \) by \( \text{rule typeVal} \)
thus \( \exists \text{case by (rule exI)} \)

**next**

**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 \( \Gamma \ V = \text{Some} \ T \) by \( \text{auto} \)

**hence** \( \Gamma \vdash \text{Var V} : T \) by \( \text{rule typeVar} \)
thus \( \exists \text{case by (rule exI)} \)

**next**

**case** \( \text{BinOp e}_1 \ bop \ e_2 \)

**note** \( \text{IH1} = \langle \text{fv} \ e_1 \subseteq \text{dom} \ \Gamma \implies \exists T. \ \Gamma \vdash e_1 : T \rangle \)

**note** \( \text{IH2} = \langle \text{fv} \ e_2 \subseteq \text{dom} \ \Gamma \implies \exists T. \ \Gamma \vdash e_2 : T \rangle \)

from \( \langle \text{fv} (\text{e}_1 \ \text{bop} \ \text{e}_2) \subseteq \text{dom} \ \Gamma \rangle \) have \( \exists T. \ \Gamma \vdash e_1 \ \text{bop} \ e_2 : T \) by \( \text{auto} \)

**show \( \exists \text{case by (rule exI)} \)

**proof (cases T1)**

**case** \( \text{Low} \)
**show \( \exists \text{thesis} \)

**proof (cases T2)**

**case** \( \text{Low} \)

**with** \( \Gamma \vdash e_1 : T_1 ; \Gamma \vdash e_2 : T_2 \) \( (T_1 = \text{Low}) \)
**have** \( \Gamma \vdash e_1 \ \text{bop} \ e_2 : \text{Low} \) by \( \text{simp add: typeBinOp1} \)

**thus \( \exists \text{thesis by (rule exI)} \)

**next**

**case** \( \text{High} \)

**with** \( \Gamma \vdash e_1 : T_1 ; \Gamma \vdash e_2 : T_2 \) \( (T_1 = \text{Low}) \)
**have** \( \Gamma \vdash e_1 \ \text{bop} \ e_2 : \text{High} \) by \( \text{simp add: typeBinOp3} \)

**thus \( \exists \text{thesis by (rule exI)} \)

**qed**

**next**

**case** \( \text{High} \)

**with** \( \Gamma \vdash e_1 : T_1 ; \Gamma \vdash e_2 : T_2 \)
**have** \( \Gamma \vdash e_1 \ \text{bop} \ e_2 : \text{High} \) by \( \text{simp add: typeBinOp2} \)
thus \(?thesis\) by (rule exI)
qed
qed
with \(\text{that show} \ ?thesis\) by blast
qed

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

lemma \textit{exprTypingHigh}:
assumes \(\Gamma \vdash e : T\) and \(x \in \text{fv} \ e\) and \(\Gamma \ x = \text{Some High}\)
shows \(\Gamma \vdash e : \text{High}\)
using \textit{assms}
proof (induct \(e\) arbitrary; \(T\))
case (Val \(V\)) show \(?case\) by (rule typeVal)
next
case (Var \(V\))
from \(\langle x \in \text{fv} \ (\text{Var} \ V) \rangle\) have \(x = V\) by simp
with \(\Gamma \ x = \text{Some High}\) show \(?case\) by (simp add: typeVar)
next
case (BinOp \(e_1\) \(bop\) \(e_2\))
note IH1 = \((\forall T. \ \Gamma \vdash e_1 : T; x \in \text{fv} \ e_1; \ \Gamma \ x = \text{Some High}) \Rightarrow \Gamma \vdash e_1 : \text{High})
note IH2 = \((\forall T. \ \Gamma \vdash e_2 : T; x \in \text{fv} \ e_2; \ \Gamma \ x = \text{Some High}) \Rightarrow \Gamma \vdash e_2 : \text{High})
from \(\Gamma \vdash e_1 \leftarrow bop \rightarrow e_2 : T)\)
have \(T; (\exists \ T_1. \ \Gamma \vdash e_1 : T_1) \land (\exists \ T_2. \ \Gamma \vdash e_2 : T_2)\) by (auto intro!: \textit{exprBinopTypeable})
then obtain \(T_1\) where \(\Gamma \vdash e_1 : T_1\) by auto
from \(T\) obtain \(T_2\) where \(\Gamma \vdash e_2 : T_2\) by auto
from \(\langle x \in \text{fv} \ (e_1 \leftarrow bop \rightarrow e_2) \rangle\) 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 \(\Gamma \ x = \text{Some High}\)] 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 \(\Gamma \ x = \text{Some High}\)] have \(\Gamma \vdash e_2 : \text{High}\).
with \(\Gamma \vdash e_1 : T_1\) show \(?thesis\) by (simp add: typeBinOp3)
qed
qed

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

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

next
  case (Var V)
  from \( x \in \text{fv} (\text{Var } V) \) have \( xV \colon x = V \) by simp
  thus \(?thesis\) by simp

next
  case (BinOp e1 bop e2)
  note IH1 = \( \{[\Gamma \vdash e1 : \text{Low}; x \in \text{fv } e1] \implies \Gamma x = \text{Some Low}\} \)
  note IH2 = \( \{[\Gamma \vdash e2 : \text{Low}; x \in \text{fv } e2] \implies \Gamma x = \text{Some Low}\} \)
  from \( \Gamma \vdash e1 \langle bop \rangle e2 : \text{Low} \) have \( \Gamma \vdash e1 : \text{Low} \) and \( \Gamma \vdash e2 : \text{Low} \)
  by (auto elim: secExprTyping.cases)
  with \( x \in \text{fv } (e1 \langle bop \rangle e2) \) have \( x \in \text{fv } e1 \cup \text{fv } e2 \) by (simp add:FVc)
  hence \( x \in \text{fv } e1 \lor x \in \text{fv } e2 \) by auto
  thus \(?case\) proof
    assume \( x \in \text{fv } e1 \)
    with \( \text{IH1}[OF \Gamma \vdash e1 : \text{Low}] \) show \(?thesis\) by auto
  next
    assume \( x \in \text{fv } e2 \)
    with \( \text{IH2}[OF \Gamma \vdash e2 : \text{Low}] \) show \(?thesis\) by auto
  qed

qed

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

proof (induct e arbitrary: T)
  case (Val V)
  have \( \text{fv} (\text{Val } V) = \{\} \) 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 \( \Gamma \vdash \text{Var } V : T \) have \( \Gamma V = \text{Some } T \) by (auto elim: secExprTyping.cases)
    with \( x = V \) show \( x \in \text{dom } \Gamma \) by auto
  qed

next
  case (BinOp e1 bop e2)
  note IH1 = \( \{\bigland T. \Gamma \vdash e1 : T \implies \text{fv } e1 \subseteq \text{dom } \Gamma \} \)
note \( IH2 = \{ \land T. \Gamma \vdash e_2 : T \implies \text{fv} \ e_2 \subseteq \text{dom} \ \Gamma \} \)

show \( ?\text{case} \)

proof

fix \( x \) assume \( x \in \text{fv} \ (e_1 \ <\ bop > \ e_2) \)

from \( \Gamma \vdash e_1 \ <\ bop > \ e_2 : T \)

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

hence \( \text{fv} \ (e_1 \ <\ bop > \ e_2) \subseteq \text{dom} \ \Gamma \) by (simp add: FVe)

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

qed

qed

lemma \( \text{exprNotNone} \):

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

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

using assms

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

  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: FVe)

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

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

  from \( \Gamma \vdash e_1 \ <\ bop > \ e_2 : T \)

  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 \( \text{fv} \ (e_1 \ <\ bop > \ e_2) \subseteq \text{dom} \ s \) 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 \( \{ e_1 \} s \neq \text{None} \).

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

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

  apply (cases 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. \( s1 \approx_L s2 \equiv \forall v \in \text{dom } state, \Gamma v = \text{Some Low} \rightarrow (s1 v = s2 v) \)), 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**

\[
\text{lowEquiv} :: \text{typeEnv} \Rightarrow state \Rightarrow state \Rightarrow \text{bool} (\vdash s1 \approx_L s2)
\]

where

\[
\Gamma \vdash s1 \approx_L s2 \equiv \forall v \in \text{dom } \Gamma. \Gamma v = \text{Some Low} \rightarrow (s1 v = s2 v)
\]

**Lemma**

\[
\text{lowEquivReflexive}: \Gamma \vdash s1 \approx_L s1
\]

by (simp add: lowEquiv-def)

**Lemma**

\[
\text{lowEquivSymmetric}: \Gamma \vdash s1 \approx_L s2 \Rightarrow \Gamma \vdash s2 \approx_L s1
\]

by (simp add: lowEquiv-def)

**Lemma**

\[
\text{lowEquivTransitive}: \left[ \left[ \Gamma \vdash s1 \approx_L s2 \land \forall v \in \text{dom } \Gamma, \Gamma v = \text{Some Low} \rightarrow (s1 v = s2 v) \right] \right] \Rightarrow \Gamma \vdash s1 \approx_L s3
\]

by (simp add: lowEquiv-def)

2.3.2 Non Interference

**Definition**

\[
\text{nonInterference} :: \text{typeEnv} \Rightarrow \text{com} \Rightarrow \text{bool}
\]

where

\[
\text{nonInterference } \Gamma c \equiv (\forall\ s1 \approx_L s2 \land (\forall v \in \text{dom } \Gamma, \Gamma v = \text{Some Low} \rightarrow (s1 v = s2 v))
\]

\[
\Gamma \vdash s1' \approx_L s2'
\]

**Lemma**

\[
\text{nonInterferenceI}: \left[ \left[ \exists\ s1 s2 s1' s2'. \Gamma \vdash s1 \approx_L s2 \land (\exists\ s1' \land (\forall v \in \text{dom } \Gamma, \Gamma v = \text{Some Low} \rightarrow (s1 v = s2 v)))
\right] \Rightarrow \Gamma \vdash s1' \approx_L s2' \right] \Rightarrow \text{nonInterference } \Gamma c
\]

by (auto simp: nonInterference-def)

**Lemma**

\[
\text{interpretLow}: \quad \text{assumes } \Gamma \vdash s1 \approx_L s2 \quad \text{and} \quad \forall V \in \text{fv } e, \Gamma v = \text{Some Low}
\]

\[
\text{shows } [e] s1 = [e] s2
\]

using \text{all}

**Proof** (induct \( e \))

case (Val \( v \))

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

next

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

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

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

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

thus \case by auto

next

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

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

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

from \(\forall V \in \text{fv} (e1 \langle \text{bop } e2). \Gamma V = \text{Some Low}\) \text{ have } \forall V \in \text{fv} e1. \Gamma V = \text{Some Low}

and \(\forall V \in \text{fv } e2. \Gamma V = \text{Some Low} \text{ by auto}\)

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

moreover

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

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

qed

lemma \(\text{interpretLow2}\):

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

proof –

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

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

proof

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

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

qed

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

qed

lemma \(\text{assignNIhighlemma}\):

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

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

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

proof

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

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

by (auto simp add: lowEquiv-def)

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

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

\text{case True}

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

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

next

\text{case False}

with \(s1' = s1(V := [e] s1)\) \text{ and } \(s2' = s2(V := [e] s2)\)
proof

SeqCompositionality theorem compositionality is no longer valid in case of concurrency

lemma assignNIlowlemma:
assumes Γ ⊢ s1 ≈L s2 and Γ V = Some Low and Γ ⊢ e : Low
and s1' = s1(V := [e] s1) and s2' = s2(V := [e] s2)
shows Γ ⊢ s1' ≈L s2'

proof

−

⟨

from ⟨Γ s1 s2 s1⟩

fix shows nonInterference {fix nonInterference ?thesis thus ?thesis}

⟨

⟨Γ s1 s2 s1⟩

fix nonInterference

shows s1 and assumes Γ thus ?thesis

⟨

by (auto intro:interpretLow2)

with ⟨Γ s1 s2 s1⟩ ⟨s2' V' = [e] s2⟩ show ?thesis

next

case True

with ⟨s1' = s1(V := [e] s1)⟩ ⟨s2' = s2(V := [e] s2)⟩

have s1' V' = [e] s1 and s2' V' = [e] s2 by auto

from ⟨Γ ⊢ e : Low ⟩ Γ ⊢ s1 ≈L s2 ⟩ have [e] s1 = [e] s2

by (auto intro:interpretLow2)

with ⟨s1' V' = [e] s1⟩ ⟨s2' V' = [e] s2⟩ show ?thesis by simp

⟨

with ⟨⟨Γ s1 s2 s1⟩ ⟨s2' V' = [e] s2⟩ ⟩ show ?thesis by auto

qed

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

qed

lemma assignNIlowlemma:
assumes Γ ⊢ s1 ≈L s2 and Γ V = Some Low and Γ ⊢ e : Low
and s1' = s1(V := [e] s1) and s2' = s2(V := [e] s2)
shows Γ ⊢ s1' ≈L s2'

proof

−

⟨

from ⟨Γ s1 s2 s1⟩

fix shows nonInterference {fix nonInterference ?thesis thus ?thesis}

⟨

⟨Γ s1 s2 s1⟩

fix nonInterference

shows s1 and assumes Γ thus ?thesis

⟨

by (auto intro:interpretLow2)

with ⟨Γ s1 s2 s1⟩ ⟨s2' V' = [e] s2⟩ show ?thesis

next

case True

with ⟨s1' = s1(V := [e] s1)⟩ ⟨s2' = s2(V := [e] s2)⟩

have s1' V' = [e] s1 and s2' V' = [e] s2 by auto

from ⟨Γ ⊢ e : Low ⟩ Γ ⊢ s1 ≈L s2 ⟩ have [e] s1 = [e] s2

by (auto intro:interpretLow2)

with ⟨s1' V' = [e] s1⟩ ⟨s2' V' = [e] s2⟩ show ?thesis by simp

⟨

with ⟨⟨Γ s1 s2 s1⟩ ⟨s2' V' = [e] s2⟩ ⟩ show ?thesis by auto

qed

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

qed

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

theorem SeqCompositionality:
assumes nonInterference Γ c1 and nonInterference Γ c2
shows nonInterference Γ (c1;;c2)

proof (rule nonInterferenceI)

fix s1 s2 s1' s2'

assume Γ ⊢ s1 ≈L s2 and ⟨c1;;c2,s1⟩ →* ⟨Skip,s1'⟩

and ⟨c1;;c2,s2⟩ →* ⟨Skip,s2'⟩

from ⟨⟨c1;;c2,s1⟩ ⟩ obtain s1'' where ⟨c1,s1⟩ →* ⟨Skip,s1''⟩

and ⟨c2,s1''⟩ →* ⟨Skip,s1''⟩ by (auto dest:Seq-reds)

from ⟨⟨c1;;c2,s2⟩ ⟩ obtain s2'' where ⟨c1,s2⟩ →* ⟨Skip,s2''⟩

⟨

have s1' V' = s1' V' and s2' V' = s2' V' by auto

with ⟨s1' V' = s2' V' ⟩ show ?thesis by simp

qed
and \(\langle c_2, s_2''\rangle \rightarrow^* \langle \text{Skip}, s_2''\rangle\) by (auto dest: Seg-reds)

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

(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: \(\forall s_2\cdot \langle c, s_1\rangle \rightarrow^* \langle \text{Skip}, s_2\rangle\) \(\implies \Gamma \vdash s_1 \approx_L s_2\) 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\(\langle\text{while}\ (b)\ c, s_1\rangle \rightarrow^* \langle \text{Skip}, s_2\rangle\) have \(\Gamma \vdash s_1 \approx_L s_2\) 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:s1 s2 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 (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 \(s_2 = s_1(V := \llbracket e \rrbracket s_1)\) 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 \(\Gamma\ V' = \text{Some Low}\) have \(\Gamma\ V = \text{Some High}\) by simp

thus \(?thesis\ by\ simp\)

next
case False

with \(s_2 = s_1(V := \llbracket e \rrbracket s_1)\) show \(?thesis\ by\ simp\)

18
qed
}
thus ?case by (auto simp add: lowEquiv-def)

next
case (Seq c1 c2)

note IH1 = \[\lambda s1 s2. \Gamma, \text{High} \vdash c1 ; \{c1, s1\} \rightarrow* \{\text{Skip}, s2\} \implies \Gamma \vdash s1 \approx_L s2\]

note IH2 = \[\lambda s1 s2. \Gamma, \text{High} \vdash c2 ; \{c2, s1\} \rightarrow* \{\text{Skip}, s2\} \implies \Gamma \vdash s1 \approx_L s2\]

from \[\Gamma, \text{High} \vdash c1 ; c2\] have \[\Gamma, \text{High} \vdash c1 \text{ and } \Gamma, \text{High} \vdash c2\]
by (auto elim: secComTyping.cases)

from \[(c1;c2,s1) \rightarrow* \{\text{Skip}, s2\}\]
have \[\exists s3. (c1,s1) \rightarrow* \{\text{Skip}, s3\} \land (c2,s3) \rightarrow* \{\text{Skip}, s2\}\]
by (auto intro: Seq-reds)

then obtain s3 where \[(c1,s1) \rightarrow* \{\text{Skip}, s3\} \land (c2,s3) \rightarrow* \{\text{Skip}, s2\}\]
by auto

from IH1[OF \[\Gamma, \text{High} \vdash c1 ; \{c1,s1\} \rightarrow* \{\text{Skip}, s3\}\]]
have \[\Gamma \vdash s1 \approx_L s3\]
by simp

from IH2[OF \[\Gamma, \text{High} \vdash c2 ; \{c2,s3\} \rightarrow* \{\text{Skip}, s2\}\]]
have \[\Gamma \vdash s3 \approx_L s2\]
by simp

from \[\Gamma \vdash s1 \approx_L s3\] have \[\Gamma \vdash s3 \approx_L s2\]
show ?case
by (auto intro: lowEquivTransitive)

next
case (Cond b c1 c2)

note IH1 = \[\lambda s1 s2. \Gamma, \text{High} \vdash c1 ; \{c1, s1\} \rightarrow* \{\text{Skip}, s2\} \implies \Gamma \vdash s1 \approx_L s2\]

note IH2 = \[\lambda s1 s2. \Gamma, \text{High} \vdash c2 ; \{c2, s1\} \rightarrow* \{\text{Skip}, s2\} \implies \Gamma \vdash s1 \approx_L s2\]

from \[\Gamma, \text{High} \vdash b;\] if \[(b)\] c1 else c2 have \[\Gamma, \text{High} \vdash c1 \text{ and } \Gamma, \text{High} \vdash c2\]
by (auto elim: secComTyping.cases)

from \[(\text{if } (b)\) c1 else c2, s1) \rightarrow* \{\text{Skip}, s2\}\]
have \[\exists b. s1 = \text{Some true} \lor \exists b. s1 = \text{Some false}\]
by (auto dest: Cond-True-or-False)

thus ?case

proof

assume \[\exists b. s1 = \text{Some true}\]

with \[(\text{if } (b)\) c1 else c2, s1) \rightarrow* \{\text{Skip}, s2\}\] have \[\{c1, s1\} \rightarrow* \{\text{Skip}, s2\}\]
by (auto intro: CondTrue-reds)

from IH1[OF \[\Gamma, \text{High} \vdash c1 ; \text{this}\] show ?thesis .

next

assume \[\exists b. s1 = \text{Some false}\]

with \[(\text{if } (b)\) c1 else c2, s1) \rightarrow* \{\text{Skip}, s2\}\] have \[\{c2, s1\} \rightarrow* \{\text{Skip}, s2\}\]
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 = \[\lambda s1 s2. \Gamma, \text{High} \vdash c' ; \{c', s1\} \rightarrow* \{\text{Skip}, s2\} \implies \Gamma \vdash s1 \approx_L s2\]

from \[\Gamma, \text{High} \vdash \text{while } (b)\ c'\] have \[\Gamma, \text{High} \vdash c'\]
by (auto elim: secComTyping.cases)

from IH[OF \[\text{this}\]
have \[\lambda s1 s2. \{c', s1\} \rightarrow* \{\text{Skip}, s2\} \implies \Gamma \vdash s1 \approx_L s2 .

with \[(\text{while } (b)\ c', s1) \rightarrow* \{\text{Skip}, s2\}; \Gamma, \text{High} \vdash c'\]
show ?case by (auto dest: WhileStepInduct)

qed
lemma CondHighCompositionality:
assumes $\Gamma, \text{High} \vdash \text{if } (b) \ c_1 \text{ else } c_2$
shows nonInterference $\Gamma \ (\text{if } (b) \ c_1 \text{ else } c_2)$
proof (rule nonInterferenceI)
  \begin{align*}
  & 
  \text{fix } s_1 \ s_2 \ s_1' \ s_2' \\
  & \text{assume } \Gamma \vdash s_1 \approx_L s_2 \text{ and } \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \\
  & \quad \text{and } \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_2) \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \\
  & \text{show } \Gamma \vdash s_1' \approx_L s_2' \\
  & \text{proof --}
  \end{align*}
  \begin{align*}
  & \quad \text{from } \Gamma, \text{High} \vdash \text{if } (b) \ c_1 \text{ else } c_2; \langle \text{if } (b) \ c_1 \text{ else } c_2, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \\
  & \quad \text{have } \Gamma \vdash s_1 \approx_L s_1' \text{ by (auto dest:highBodies)} \\
  & \quad \text{from } \Gamma, \text{High} \vdash \text{if } (b) \ c_1 \text{ else } c_2; \langle \text{if } (b) \ c_1 \text{ else } c_2, s_2 \rangle \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \\
  & \quad \text{have } \Gamma \vdash s_2 \approx_L s_2' \text{ by (auto dest:highBodies)} \\
  & \quad \text{with } \Gamma \vdash s_1 \approx_L s_2 \vdash \Gamma \vdash s_1' \approx_L s_1' \text{ by (auto intro:lowEquivTransitive)} \\
  & \quad \text{with } \Gamma \vdash s_1 \approx_L s_2' \vdash \text{show } ?\text{thesis by (auto intro:lowEquivSymmetric)} \\
  & \text{qed}
  \end{align*}
proof (rule nonInterferenceI)
  \begin{align*}
  & \text{fix } s_1 \ s_2 \ s_1' \ s_2' \\
  & \text{assume } \Gamma \vdash s_1 \approx_L s_2 \text{ and } \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \\
  & \quad \text{and } \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_2) \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \\
  & \text{from } \langle b \rangle \ b \vdash \text{if } (b) \ c_1 \text{ else } c_2, s_1 \vdash \text{have } [b] \ s_1 = [b] \ s_2 \\
  & \quad \text{by (auto intro:interpretLow2)} \\
  & \text{from } \langle b \rangle \ b \vdash \text{if } (b) \ c_1 \text{ else } c_2, s_1 \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \\
  & \text{have } [b] \ s_1 = \text{Some true by (auto dest:Cond-True-or-False)} \\
  & \text{thus } \Gamma \vdash s_1' \approx_L s_2' \\
  & \text{proof}
  \end{align*}
  \begin{align*}
  & \text{assume } [b] \ s_1 = \text{Some true}
  \end{align*}
  \begin{align*}
  & \quad \text{with } [b] \ s_1 = [b] \ s_2; \text{ have } [b] \ s_2 = \text{Some true by (auto intro:CondTrue-reds)} \\
  & \text{from } [b] \ s_1 = \text{Some true}; \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \\
  & \quad \text{have } (c_1, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \text{ by (auto intro:CondTrue-reds)} \\
  & \text{from } [b] \ s_2 = \text{Some true}; \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_2) \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \\
  & \quad \text{have } (c_1, s_2) \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \text{ by (auto intro:CondTrue-reds)} \\
  & \quad \text{with } (c_1, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle; \Gamma \vdash s_1' \approx_L s_2; \langle \text{nonInterference } \Gamma \ c_1 \rangle \\
  & \text{show } ?\text{thesis by (auto simp:nonInterference-def)} \\
  & \text{next}
  \end{align*}
  \begin{align*}
  & \text{assume } [b] \ s_1 = \text{Some false}
  \end{align*}
  \begin{align*}
  & \quad \text{with } [b] \ s_1 = [b] \ s_2; \text{ have } [b] \ s_2 = \text{Some false by (auto intro:CondTrue-reds)} \\
  & \text{from } [b] \ s_1 = \text{Some false}; \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \\
  & \quad \text{have } (c_1, s_1) \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \text{ by (auto intro:CondFalse-reds)} \\
  & \text{from } [b] \ s_2 = \text{Some false}; \langle b \rangle \ (\text{if } (b) \ c_1 \text{ else } c_2, s_2) \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \\
  \end{align*}
have \((c_2,s_2) \rightarrow^* \langle \text{Skip}, s_2' \rangle\) by (auto intro:CondFalse-reds)
with \((c_2,s_1) \rightarrow^* \langle \text{Skip}, s_1' \rangle\); \(\Gamma \vdash s_1 \approx_L s_2\); nonInterference \(\Gamma c_2\)
show \(?thesis\) by (auto simp:nonInterference-def)

qed

lemma \(\text{WhileHighCompositionality}\):
assumes \(\Gamma, \text{High} \vdash \text{while} (b) \ c'
shows \(\text{nonInterference} \ (\text{while} (b) \ c')\)
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:lowEquivStransitive)
from \(\Gamma \vdash s_1 \approx_L s_1'\) have \(\Gamma \vdash s_1' \approx_L s_1\) by (auto intro:lowEquivSymmetric)
with \(\Gamma \vdash s_1 \approx_L s_2\) show \(?thesis\) by (auto intro:lowEquivStransitive)

qed

lemma \(\text{WhileLowStepInduct}\):
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 \(\text{and} b : \text{Low}\)
and \(\text{body}: \forall s_1 \ s_1' \ s_2 \ s_2' . \ ((c',s_1) \rightarrow^* \langle \text{Skip}, s_1' \rangle; \ c',s_2 \rightarrow^* \langle \text{Skip}, s_2' \rangle; \ s_1 \approx_L s_2) \implies \Gamma \vdash s_1' \approx_L s_2'\)
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:s2 rule:while-reds-induct)
case \(\text{false} \ s_1\)
from \(\Gamma \vdash b : \text{Low}\); \(\Gamma \vdash s_1 \approx_L s_2\); \(\text{have} \ [b] \ s_1 \equiv [b] \ s_2\) 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 \(?case\) by auto
next
case \(\text{true} \ s_1 \ s_1''\)
note \(IH = \langle s_2'', \ [s] \ s_1'' \approx_L s_2''; \ (c',s_2'') \rightarrow^* \langle \text{Skip}, s_2' \rangle\;\implies \Gamma \vdash s_1' \approx_L s_2'\)\)
from \(\Gamma \vdash b : \text{Low}\); \(\Gamma \vdash s_1 \approx_L s_2\); \(\text{have} \ [b] \ s_1 \equiv [b] \ s_2\) by (auto intro:interpretLow2)
with \([b] \ s_1 = \text{Some true}\) have \([b] \ s_2 = \text{Some true}\) by simp


with \((\text{while } (b) c', s_2) \rightarrow (\text{Skip}, s_2')\) obtain \(s_2''\) where \((c', s_2) \rightarrow (\text{Skip}, s_2')\)

and \((\text{while } (b) c', s_2') \rightarrow (\text{Skip}, s_2)\) by (auto dest:WhileTrue-reds)

from body[OF \((c', s_1) \rightarrow (\text{Skip}, s_1')\); \((c', s_2) \rightarrow (\text{Skip}, s_2')\); \(\Gamma \vdash s_1 \approx_L s_2\)]

have \(\Gamma \vdash s_1'' \approx_L s_2''\).

from IH[OF this \((\text{while } (b) c', s_2') \rightarrow (\text{Skip}, s_2')\)] show \(?\text{case }\).

qed


**lemma** WhileLowCompositionality:

assumes \(\text{nonInterference } \Gamma c' \and \Gamma \vdash b : \text{Low} \and \Gamma, \text{Low} \vdash c'\)

shows \(\text{nonInterference } \Gamma (\text{while } (b) c')\)

proof (rule nonInterferenceI)

fix \(s_1 \ s_2 \ s_1' \ s_2'\)

assume \(\Gamma \vdash s_1 \approx_L s_2 \and (\text{while } (b) c', s_1) \rightarrow (\text{Skip}, s_1')\)

and \((\text{while } (b) c', s_2) \rightarrow (\text{Skip}, s_2')\)

{ fix \(s_1 \ s_2 \ s_1'' \ s_2''\)

assume \((c', s_1) \rightarrow (\text{Skip}, s_1')\) and \((c', s_2) \rightarrow (\text{Skip}, s_2')\)

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

with \(\text{nonInterference } c'\) have \(\Gamma \vdash s_1'' \approx_L s_2''\)

by (auto simp:nonInterference-def)

} hence \(\forall s_1 \ s_1'' \ s_2 \ s_2''. [(c', s_1) \rightarrow (\text{Skip}, s_1''); (c', s_2) \rightarrow (\text{Skip}, s_2''); \Gamma \vdash s_1 \approx_L s_2] \Longrightarrow \Gamma \vdash s_1'' \approx_L s_2''\) by auto

with \((\text{while } (b) c', s_1) \rightarrow (\text{Skip}, s_1')\); \((\text{while } (b) c', s_2) \rightarrow (\text{Skip}, s_2')\)

\((\Gamma \vdash b : \text{Low}) \Gamma \vdash s_1 \approx_L s_2\) show \(\Gamma \vdash s_1' \approx_L s_2'\)

by (auto intro:WhileLowStepInduct)

qed

and now: the main theorem:

**theorem** secTypeImpliesNonInterference:

\(\Gamma, T \vdash c \Longrightarrow \text{nonInterference } \Gamma c\)

proof (induct c arbitrary: \(T\) rule:com.induct)

case \(\text{Skip}\)

show \(?\text{case}\)

proof (rule nonInterferenceI)

fix \(s_1 \ s_2 \ s_1' \ s_2'\)

assume \(\Gamma \vdash s_1 \approx_L s_2 \and (\text{Skip}, s_1) \rightarrow (\text{Skip}, s_1')\) and \((\text{Skip}, s_2) \rightarrow (\text{Skip}, s_2')\)

from \((\text{Skip}, s_1) \rightarrow (\text{Skip}, s_1')\) have \(s_1 = s_1'\) by (auto dest:Skip-reds)

from \((\text{Skip}, s_2) \rightarrow (\text{Skip}, s_2')\) have \(s_2 = s_2'\) by (auto dest:Skip-reds)

from \(\Gamma \vdash s_1 \approx_L s_2\) and \(\langle s_1 = s_1', s_2 = s_2'\rangle\)

show \(\Gamma \vdash s_1' \approx_L s_2'\) by simp

qed

next

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

from \(\Gamma, T \vdash V := e\)

have \(\text{varPrem:}(\Gamma \vdash \text{Some High}) \lor (\Gamma \vdash \text{Some Low} \and \Gamma \vdash e : \text{Low} \and T = \text{Low})\)

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 \(\langle V := e, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle\) and \(\langle V := e, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle\)
from \(\langle V := e, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle\) have \(s_1' = s_1(V := [e] s_1)\) by (auto intro:LAss-reds)
from \(\langle V := e, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle\) 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 \(\Gamma \vdash s_1 \approx_L s_2\) \(\langle s_1' = s_1(V := [e] s_1)\rangle \langle s_2' = s_2(V := [e] s_2)\rangle\)
show \(\text{?thesis}\) by (auto intro:assignNIhighlemma)
next
assume \(\Gamma \vdash V = \text{Some Low} \land \Gamma \vdash e : \text{Low} \land T = \text{Low}\)
with \(\Gamma \vdash s_1 \approx_L s_2\) \(\langle s_1' = s_1(V := [e] s_1)\rangle \langle s_2' = s_2(V := [e] s_2)\rangle\)
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 \(IH1[\text{OF} \: \Gamma, \text{High} \vdash c_1]\) have \(\text{nonInterference} \Gamma c_1\)
moreover
from \(IH2[\text{OF} \: \Gamma, \text{High} \vdash c_2]\) have \(\text{nonInterference} \Gamma c_2\)
ultimately show \(\text{?thesis}\) by (auto intro:SeqCompositionality)
next
case Low
with \(\Gamma, T \vdash c_1; c_2\) have \(\Gamma, \text{Low} \vdash c_1 \land \Gamma, \text{Low} \vdash c_2\)
\lor \(\Gamma, \text{High} \vdash c_1; c_2\)
by (auto elim:secComTyping.cases)
thus \(\text{?thesis}\)
proof
assume \(\Gamma, \text{Low} \vdash c_1 \land \Gamma, \text{Low} \vdash c_2\)
hence \(\Gamma, \text{Low} \vdash c_1\) and \(\Gamma, \text{Low} \vdash c_2\) by simp-all
from \(IH1[\text{OF} \: \Gamma, \text{Low} \vdash c_1]\) have \(\text{nonInterference} \Gamma c_1\)
moreover
from \(IH2[\text{OF} \: \Gamma, \text{Low} \vdash c_2]\) have \(\text{nonInterference} \Gamma c_2\)
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[\langle \Gamma, \text{High} \vdash \text{c1} \rangle] have nonInterference \Gamma \text{c1}.

moreover
from IH2[\langle \Gamma, \text{High} \vdash \text{c2} \rangle] have nonInterference \Gamma \text{c2}.

ultimately show \text{thesis} by(auto intro:SeqCompositionality)

qed

next

next case (Cond \text{b} \text{c1} \text{c2})

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

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

show \text{thesis} using \langle \Gamma \vdash \text{b} : \text{Low} \rangle

by(auto intro:CondLowCompositionality)

next

next case

next case

next case

next case

next case (While \text{b} \text{c'})

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

show \text{thesis} using \langle \Gamma \vdash \text{b} : \text{Low} \rangle

by(auto intro:WhileHighCompositionality)

next

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case

next case
hence $\Gamma \vdash b : \text{Low}$ and $\Gamma, \text{Low} \vdash c'$ by simp-all
moreover from IH[OF $\Gamma, \text{Low} \vdash c'$] have nonInterference $\Gamma \vdash c'$.
ultimately show thesis by(auto intro:WhileLowCompositionality)

next
assume $\Gamma, \text{High} \vdash \text{while}(b) \ c'$
thus thesis by(auto intro:WhileHighCompositionality)
qed
qed

end

theory Execute
imports secTypes
begin

3 Executing the small step semantics

code-pred (modes: $i \Rightarrow o \Rightarrow \text{bool}$ as exec-red, $i \Rightarrow i \Rightarrow i \Rightarrow \text{bool}$ 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 $\Gamma \ V \ \text{lev} \ \text{assume} \ x = \Gamma \ xa = \text{Val} \ V \ xb = \text{lev}$
from secExprTyping(1-2) this show thesis by (cases \text{lev}) auto
next
fix $\Gamma \ Vn \ \text{lev} \ \text{assume} \ x = \Gamma \ xa = \text{Var} \ Vn \ xb = \text{lev} \ \Gamma \ Vn = \text{Some} \ \text{lev}$
from secExprTyping(3) this show thesis by (auto simp add: Predicate.eq-is-eq)
next
fix $\Gamma \ e1 \ e2 \ \text{bop} \ \text{assume} \ x = \Gamma \ xa = e1 \langle \text{bop} \rangle \ e2 \ xb = \text{Low}$
$\Gamma \vdash e1 : \text{Low} \ \Gamma \vdash e2 : \text{Low}$
from secExprTyping(4) this show thesis by auto
next

\textbf{fix} \ \Gamma \ e_1 \ e_2 \ \text{lev} \ \text{bop} \\
\text{assume} \ x = \Gamma \ xa = e_1 <\text{bop} > \ e_2 \ xb = \text{High} \ \\
\Gamma \vdash e_1 : \text{High} \ \Gamma \vdash e_2 : \text{lev} \\
\text{from \ secExprTyping(5–6) this show thesis by (cases lev) (auto)} \\
\text{next} \\
\text{fix} \ \Gamma \ e_1 \ e_2 \ \text{lev} \ \text{bop} \\
\text{assume} \ x = \Gamma \ xa = e_1 <\text{bop} > \ e_2 \ xb = \text{High} \ \\
\Gamma \vdash e_1 : \text{lev} \ \Gamma \vdash e_2 : \text{High} \\
\text{from \ secExprTyping(6–7) this show thesis by (cases lev) (auto)} \\
\text{qed} \\
\text{qed}

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

\textbf{code-pred} \ (\text{modes: i => o => i => bool as compute-secComTyping,} \\
\text{i => i => i => bool as check-secComTyping}) \ \text{secComTyping} \\
\text{proof} – \\
\text{case \ secComTyping} \\
\text{from \ secComTyping.prems show thesis} \\
\text{proof} \\
\text{fix} \ \Gamma \ T \ \text{assume} \ x = \Gamma \ xa = T \ xb = \text{Skip} \\
\text{from \ secComTyping(1–2) this show thesis by (cases T) auto} \\
\text{next} \\
\text{fix} \ \Gamma \ V \ T \ e \ \text{assume} \ x = \Gamma \ xa = T \ xb = V := e \ \Gamma \ V = \text{Some High} \\
\text{from \ secComTyping(3–4) this show thesis by (cases T) (auto)} \\
\text{next} \\
\text{fix} \ \Gamma \ e \ V \\
\text{assume} \ x = \Gamma \ xa = \text{Low} \ xb = V := e \ \Gamma \vdash e : \text{Low} \ \Gamma \ V = \text{Some Low} \\
\text{from \ secComTyping(5) this show thesis by auto} \\
\text{next} \\
\text{fix} \ \Gamma \ T \ c_1 \ c_2 \\
\text{assume} \ x = \Gamma \ xa = T \ xb = \text{Seq c1 c2} \ \Gamma, T \vdash c_1 \ \Gamma, T \vdash c_2 \\
\text{from \ secComTyping(6) this show thesis by auto} \\
\text{next} \\
\text{fix} \ \Gamma \ b \ T \ c \\
\text{assume} \ x = \Gamma \ xa = T \ xb = \text{while (b) c} \ \Gamma \vdash b : T \ \Gamma, T \vdash c \\
\text{from \ secComTyping(7) this show thesis by auto} \\
\text{next} \\
\text{fix} \ \Gamma \ b \ T \ c_1 \ c_2 \\
\text{assume} \ x = \Gamma \ xa = T \ xb = \text{if (b) c1 else c2} \ \Gamma \vdash b : T \ \Gamma, T \vdash c_1 \ \Gamma, T \vdash c_2 \\
\text{from \ secComTyping(8) this show thesis by blast} \\
\text{next} \\
\text{fix} \ \Gamma \ c \\
\text{assume} \ x = \Gamma \ xa = \text{Low} \ xb = c \ \Gamma, \text{High} \vdash c \\
\text{from \ secComTyping(9) this show thesis by blast} \\
\text{qed} \\
\text{qed}
3.1 An example taken from Volpano, Smith, Irvine

definition com = if (Var "x" ≪Eq≫ Val (Intg 1)) ("y" := Val (Intg 1)) else ("y" := Val (Intg 0))
definition Env = map-of [(["x", High], ("y", High))]
values { T. Env ⊢ (Var "x" ≪Eq≫ Val (Intg 1)): T}
value Env, High ⊢ com
value Env, Low ⊢ com
values 1 { T. Env, T ⊢ com}

end

References
