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

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

February 25, 2020

Abstract

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

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

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

1 **The Language** 3  
  1.1 Variables and Values ................................. 3  
  1.2 Expressions and Commands ............................ 3  
  1.3 State ................................................. 4  
  1.4 Small Step Semantics ................................. 4  

2 **Security types** 10  
  2.1 Security definitions ................................. 10  
  2.2 Lemmas concerning expressions ........................ 11  
  2.3 Noninterference definitions ........................... 15  
    2.3.1 Low Equivalence ................................. 15  
    2.3.2 Non Interference ................................. 15  

3 **Executing the small step semantics** 25  
  3.1 An example taken from Volpano, Smith, Irvine ........... 27
theory Semantics
imports Main
begin

1 The Language

1.1 Variables and Values

type-synonym vname = string — names for variables

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

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

1.2 Expressions and Commands

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

datatype expr
  = Val val — value
  | Var vname — local variable
  | BinOp expr bop expr ( ≪ [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 Skip-reds:
\[(\text{Skip}, s) \rightarrow^* (c', s') \implies s = s' \land c' = \text{Skip}\]
by (blast elim: converse-rtranclpE red_cases)

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

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

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

with \(\langle c_1; c_2, s\rangle \rightarrow\) \((\text{Skip}, s')\) show ?thesis by simp
qed
with that show ?thesis by blast
qed


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


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


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


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


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


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


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


lemma Seq-red-nE: assumes \(\langle c_1; c_2, s\rangle \rightarrow^n \langle \text{Skip}, s'\rangle\)

obtains \( i \, j \, s'' \) where \( \langle c_1, s \rangle \rightarrow (\langle \text{Skip}, s' \rangle \) \text{ and } \langle c_2, s'' \rangle \rightarrow (\langle \text{Skip}, s \rangle \)
and \( n = i + j + 1 \)

proof –

from \( \langle c_1;i;c_2,s \rangle \rightarrow^n (\langle \text{Skip}, s \rangle \)
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 \( c_1;i;c_2 \) \text{ s n Skip s' arbitrary; } c_1 \text{ rule:red-n.induct})

case (\text{red-n-Rec s c'' s'''} n s')

note \( \text{IH} = \langle \langle c_1, c'' = c_1;i;c_2 \rangle \rightarrow^n \langle c''', s'' \rangle \)

from \( \langle c_1;i;c_2,s \rangle \rightarrow (\langle c''', s'' \rangle) \)

have \( \langle c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \rangle \lor 
(\exists c_1'. c'' = c_1':c_2 \land (\langle c_1, s \rangle \rightarrow \langle c_1', s''' \rangle) \)

by (induct \( c_1;i;c_2 \rightarrow^n \text{ rule:red-induct} \) auto

thus \?thesis

proof

assume \( c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \)

hence \( c_1 = \text{Skip} \) and \( c'' = c_2 \) and \( s = s'' \) by simp-all

from \( c_1 = \text{Skip} \) have \( \langle c_1, s \rangle \rightarrow \langle \text{Skip}, s \rangle \) by (fastforce intro: \text{red-n-Base})

with \( \langle c'', s'' \rangle \rightarrow (\langle \text{Skip}, s'' \rangle) \) \( c'' = c_2 \) \( (s = s'') \)

show \?thesis by (rule-tac \( x = 0 \) in \text{exfI}) auto

next

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

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

from \( \text{IH}[OF \langle c'' = c_1';c_2 \rangle] \) obtain \( i \, j \, s \)

where \( \langle c_1', s'' \rangle \rightarrow (\langle \text{Skip}, s \rangle \) \land \langle c_2, s \rangle \rightarrow (\langle \text{Skip}, s \rangle \)

and \( n = i + j + 1 \) by blast

from \( (c_1, s) \rightarrow (\langle c_1', s'' \rangle) \rightarrow (\langle c_1', s'' \rangle) \rightarrow (\langle \text{Skip}, s \rangle) \)

have \( (c_1, s) \rightarrow \langle \text{Skip}, s \rangle \) by (rule \text{red-n.red-n-Rec})

with \( \langle c_2, s \rangle \rightarrow (\langle \text{Skip}, s \rangle) \) \( (n = i + j + 1) \) \text{ show } ?thesis

by (rule-tac \( x = \text{Suc i in exfI} \) auto

qed

qed

with that show \?thesis by blast

qed

lemma \text{while-red-nE:}

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

\( \implies (\langle b \rangle \; \text{cx,s} \rightarrow (\langle \text{some false } \implies s = s' \land n = 1) \lor 
(\exists i \, j \, s'''. \; b \; \text{cx,s} \rightarrow i (\langle \text{skip}, s''' \rangle \) \land 
(\langle \text{while } (b) \; \text{cx,s} \rangle \rightarrow j (\langle \text{skip}, s \rangle \) \land n = i + j + 2)

proof (induct while \( (b) \; \text{cx,s} \rightarrow \langle \text{some false } \) \( c'' = \text{skip } \land s'' = s \) \lor 
(\{b \} \; s = \text{some true } \land c'' = \text{cx } \); \text{while } (b) \; \text{cx } \land s'' = s)

by (induct while \( (b) \; \text{cx } \rightarrow^n \text{ rule:red-induct} \) auto

thus \?thesis

proof


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

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

with \((c'',s'') \rightarrow^n (\text{Skip},s')\) 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 \(\text{thesis}\) by fastforce

next

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

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

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

with \((c'',s'') \rightarrow^n (\text{Skip},s')\)

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

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

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

qed

qed

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

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

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

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

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

shows \(P s s'\)

using \textit{major}

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

fix \(n s\)

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

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

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

have \([b] s = \text{Some false} \wedge s = s' \wedge n = 1\) \text{ or } \(\exists i j s''. \ [b] s = \text{Some true}; \langle cx,s \rangle \rightarrow^i (\text{Skip},s'')\)

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

by (rule \text{while-red-nE})

thus \(P s s'\)

proof

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

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

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

with \(s = s'\) show \(\text{thesis}\) by simp

next

assume \(\exists i j s''. \ [b] s = \text{Some true} \wedge \langle cx,s \rangle \rightarrow^i (\text{Skip},s'') \wedge\)

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

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

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

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

with \(\text{IHall}\) have \(P s'' s'\)

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

from \(\text{IHtrue}[OF \ [b] s = \text{Some true}]; \langle cx,s \rangle \rightarrow^3 (\text{Skip},s'')\)
⟨while (b) cx,s′⟩ →∗ ⟨Skip,s′⟩
this show \textit{?thesis}.

qed
qed

lemma reds-to-red-n:(c,s) →∗ ⟨c′,s′⟩ =⇒ ∃ n. (c,s) →^n ⟨c′,s′⟩
by(induct rule:converse-rtranclp-induct2.auto intro:red-n.intros)

lemma red-n-to-reds:(c,s) →^n ⟨c′,s′⟩ =⇒ ⟨c,s⟩ →∗ ⟨c′,s′⟩
by(induct rule:red-n.induct,auto intro:converse-rtranclp-into-rtranclp)

lemma while-reds-induct[consumes 1, case-names false true]:
[(while (b) cx,s) →∗ ⟨Skip,s′⟩; \forall s. [b] s = Some false =⇒ P s s;]
\forall s s''. [b] s = Some true; ⟨cx,s⟩ →∗ ⟨Skip,s''⟩;
⟨while (b) cx,s''⟩ →∗ ⟨Skip,s′⟩; P s'' s =⇒ P s s'
apply(drule reds-to-red-n,clarsimp)
apply(erule while-red-n-induct,clarsimp)
by(auto dest:red-n-to-reds)

lemma red-det:
[(⟨c,s⟩ → ⟨c_1,s_1⟩; ⟨c,s⟩ → ⟨c_2,s_2⟩)] =⇒ c_1 = c_2 ∧ s_1 = s_2
proof(induct arbitrary;c_2 rule:red-induct)
case (SeqRed c_1 s c_1 s' c_2')
note IH = (\forall c_2. ⟨c_1,s⟩ → ⟨c_2,s_2⟩ =⇒ c_1' = c_2 ∧ s' = s_2)
from ⟨⟨c_1'; c_2',s⟩⟩ → ⟨⟨c_2',s_2⟩⟩ have c_1 = Skip ∨ (\exists cx. c_2 = cx;:c_2' ∧ ⟨c_1,s⟩ → ⟨cx,s_2⟩)
by(fastforce elim:red.cases)
thus \textit{?case}
proof
assume c_1 = Skip
with ⟨⟨c_1,s⟩⟩ → ⟨⟨c_1',s'⟩⟩ have False by(fastforce elim:red.cases)
thus \textit{?thesis} by simp
next
assume \exists cx. c_2 = cx;c_2' ∧ ⟨c_1,s⟩ → ⟨cx,s_2⟩
then obtain cx where c_2 = cx;c_2' and ⟨c_1,s⟩ → ⟨cx,s_2⟩ by blast
from IH[OF ⟨⟨c_1,s⟩⟩ → ⟨⟨cx,s_2⟩⟩] have c_1' = cx ∧ s' = s_2.
with ⟨c_2 = cx;c_2'⟩ show \textit{?thesis} by simp
qed
qed (fastforce elim:red.cases)+

theorem reds-det:
[(⟨c,s⟩ →∗ ⟨Skip,s_1⟩; ⟨c,s⟩ →∗ ⟨Skip,s_2⟩)] =⇒ s_1 = s_2
proof(induct rule:converse-rtranclp-induct2)
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'⟩⟩ →* ⟨Skip,s₂⟩ =⇒ s₁ = s₂
  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

end
| 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: $\Gamma, \text{High} \vdash e \implies \Gamma, \text{Low} \vdash e$ |

## 2.2 Lemmas concerning expressions

**lemma** `exprTypeable`:

assumes $\text{fv } e \subseteq \text{dom } \Gamma$

obtains $T$ where $\Gamma \vdash e : T$

**proof** —

from $(\text{fv } e \subseteq \text{dom } \Gamma)$ 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 $?\text{case by (rule exI)}$

next

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

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

with $(\text{fv } (\text{Var } V) \subseteq \text{dom } \Gamma)$ 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 $?\text{case by (rule exI)}$

next

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

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

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

from $(\text{fv } (e1 \ bop \ e2) \subseteq \text{dom } \Gamma)$

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

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

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

show $?\text{case}$

**proof (cases T1)**

**case** Low

show $?\text{thesis}$

**proof (cases T2)**

**case** Low

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

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

thus $?\text{thesis by (rule exI)}$

next

**case** High

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

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

thus $?\text{thesis by (rule exI)}$

qed

next

**case** High

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

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

qed

qed

with that show \(\text{thesis by blast}\)

qed

lemma \texttt{exprBinopTypeable}:

assumes \(\Gamma \vdash e_1 \typeof{\text{bop}} e_2 : T\)

shows \(\exists T_1. \Gamma \vdash e_1 : T_1 \land \exists T_2. \Gamma \vdash e_2 : T_2\)

using \texttt{assms} by (auto elim: \text{secExprTyping.cases})

lemma \texttt{exprTypingHigh}:

assumes \(\Gamma \vdash e : T \land x \in \text{fv } e \land \Gamma x = \text{Some High}\)

shows \(\Gamma \vdash e : \text{High}\)

using \texttt{assms}

proof (induct \(e\) arbitrary: \text{T})

next

next

note \(\text{IH1} = (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)\) by (auto intro!: \texttt{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 \(x \in \text{fv} (e_1 \typeof{\text{bop}} e_2)\) have \(x \in (\text{fv } e_1 \cup \text{fv } e_2)\) by simp

hence \(x \in \text{fv } e_1 \lor x \in \text{fv } e_2\) by auto

thus \(\text{thesis}\)

proof

assume \(x \in \text{fv } e_1\)

from \(\text{IH1}[OF \Gamma \vdash e_1 : T_1]\) this \(\Gamma x = \text{Some High}\) have \(\Gamma \vdash e_1 : \text{High}\).

with \(\Gamma \vdash e_2 : T_2\) show \(\text{thesis}\) by (simp add: \text{typeBinOp2})

next

assume \(x \in \text{fv } e_2\)

from \(\text{IH2}[OF \Gamma \vdash e_2 : T_2]\) this \(\Gamma x = \text{Some High}\) have \(\Gamma \vdash e_2 : \text{High}\).

with \(\Gamma \vdash e_1 : T_1\) show \(\text{thesis}\) by (simp add: \text{typeBinOp3})

qed

qed

lemma \texttt{exprTypingLow}:

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

proof (induct e)
  case (Val V)
  have \( \text{fv} (\text{Val} \ V) = \{\} \) by (rule FVc)
  with \( \langle x \in \text{fv} (\text{Val} \ V) \rangle \) have \( \text{False} \) by auto
  thus ?thesis by simp
next
  case (Var V)
  from \( \langle x \in \text{fv} (\text{Var} \ V) \rangle \) have \( xV \colon x = V \) by simp
  with \( \text{fv} e \) show ?thesis by simp
next
  case (BinOp e1 bop e2)
  note IH1 = \( \langle \Gamma \vdash e1 : \text{Low} ; x \in \text{fv} e1 \rangle \implies \Gamma \ x = \text{Some Low} \)
  note IH2 = \( \langle \Gamma \vdash e2 : \text{Low} ; x \in \text{fv} e2 \rangle \implies \Gamma \ x = \text{Some Low} \)
  from \( \Gamma \vdash e1 \ bop \ e2 : \text{Low} \) have \( \Gamma \vdash e1 : \text{Low} \) and \( \Gamma \vdash e2 : \text{Low} \)
    by (auto elim:secExprTyping_cases)
  from \( \langle x \in \text{fv} (e1 \ bop \ e2) \rangle \) have \( x \in \text{fv} e1 \cup \text{fv} e2 \) by (simp add:FVc)
  hence \( x \in \text{fv} e1 \cup x \in \text{fv} e2 \) by auto
  thus ?case
  proof
    assume \( x \in \text{fv} e1 \)
    with IH1[OF \( \Gamma \vdash e1 : \text{Low} \)] show ?thesis by auto
  next
    assume \( x \in \text{fv} e2 \)
    with 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 \( \langle x = V \rangle \) show \( x \in \text{dom} \ \Gamma \) by auto
  qed
next
  case (BinOp e1 bop e2)
  note IH1 = \( \langle \bigwedge T. \Gamma \vdash e1 : T \implies \text{fv} e1 \subseteq \text{dom} \ \Gamma \rangle \)
\[
\text{note } IH2 = (\forall T. \Gamma \vdash e_2 : T \implies \text{fv } e_2 \subseteq \text{dom } \Gamma)
\]

\text{show } ?\text{case}

\text{proof}

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

\text{from } \Gamma \vdash e_1 \ <\ bop > \ e_2 : T;

\text{have } Q:\ (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)

\text{by (rule exprBinopTypeable)}

\text{then obtain } T_1 \text{ where } \Gamma \vdash e_1 : T_1 \text{ by blast}

\text{from } Q \text{ obtain } T_2 \text{ where } \Gamma \vdash e_2 : T_2 \text{ by blast}

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

\text{moreover}

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

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

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

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

\text{qed}

\text{qed}

\text{lemma } exprNotNone:

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

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

\text{using assms}

\text{proof (induct } e \text{ arbitrary: } \Gamma \ T \ s)

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

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

\text{next}

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

\text{have } [\text{Var } V] s = s \ V \text{ by (simp add:Var)}

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

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

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

\text{next}

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

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

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

\text{from } \Gamma \vdash e_1 \ <\ bop > \ e_2 : T \text{ have } (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)

\text{by (rule exprBinopTypeable)}

\text{then obtain } T_1 \ T_2 \text{ where } \Gamma \vdash e_1 : T_1 \text{ and } \Gamma \vdash e_2 : T_2 \text{ by blast}

\text{from } \text{fv } (e_1 \ <\ bop > \ e_2) \subseteq \text{dom } s \text{ have } \text{fv } e_1 \cup \text{fv } e_2 \subseteq \text{dom } s \text{ by (simp add:FVc)}

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

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

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

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

\text{apply(cases bop) apply auto}

\text{apply(case-tac y,auto,case-tac ya,auto)+}

\text{done}
2.3 Noninterference definitions

2.3.1 Low Equivalence

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

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

**definition** lowEquiv :: typeEnv ⇒ state ⇒ state ⇒ bool (\(-\vdash - \approx_L -\))

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** lowEquivReflexive: \( \Gamma \vdash s1 \approx_L s1 \)

**by (simp add: lowEquiv-def)**

**lemma** lowEquivSymmetric:

\( \Gamma \vdash s1 \approx_L s2 \Rightarrow \Gamma \vdash s2 \approx_L s1 \)

**by (simp add: lowEquiv-def)**

**lemma** lowEquivTransitive:

\[ [ [\Gamma \vdash s1 \approx_L s2]; \Gamma \vdash s2 \approx_L s3 ] \Rightarrow \Gamma \vdash s1 \approx_L s3 \]

**by (simp add: lowEquiv-def)**

2.3.2 Non Interference

**definition** nonInterference :: typeEnv ⇒ com ⇒ bool

where

\[ \text{nonInterference } \Gamma c \equiv (\forall s1 s2 s1' s2'. (\Gamma \vdash s1 \approx_L s2 \land (c,s1) \rightarrow^* (\text{Skip},s1') \land (c,s2) \rightarrow^* (\text{Skip},s2'))) \rightarrow \Gamma \vdash s1' \approx_L s2' ) \]

**lemma** nonInterferenceI:

\[ [\land s1 \approx_L s1'; s2 \approx_L s2'; [\Gamma \vdash s1 \approx_L s2; (c,s1) \rightarrow^* (\text{Skip},s1'); (c,s2) \rightarrow^* (\text{Skip},s2') ] \rightarrow \Gamma \vdash s1' \approx_L s2' ] \Rightarrow \text{nonInterference } \Gamma c \]

**by (auto simp: nonInterference-def)**

**lemma** interpretLow:

assumes \( \Gamma \vdash s1 \approx_L s2 \) and all:\( \forall V \in \text{fv } e. \Gamma V = \text{Some Low} \)

shows \([e] s1 = [e] s2\)

using all

**proof (induct e)**

  case (Val v)
  show ?case by (simp add: Val)

  next
  case (Var V)

qed
proof

lemma assignNIhighlemma

qed

interpretLow2

next

proof

by ⟨⟨from fixes Γ assumes Γ and s2 with ⟨qed

thus ?case by auto

next

case (BinOp e1 bop e2)

note IH1 = ∀V∈fv e1. Γ V = Some Low ⇒ [e1]s1 = [e1]s2.

note IH2 = ∀V∈fv e2. Γ V = Some Low ⇒ [e2]s1 = [e2]s2.

from (∀V∈fv (e1 <bop> e2)). Γ V = Some Low have ∀V∈fv e1. Γ V = Some Low

and (∃V∈fv e2). Γ V = Some Low by auto

from IH1[OF (∀V∈fv e1. Γ V = Some Low)] have [e1]s1 = [e1]s2 .

moreover

from IH2[OF (∀V∈fv e2. Γ V = Some Low)] have [e2]s1 = [e2]s2 .

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

qed

lemma interpretLow2:

assumes Γ ⊢ e : Low and Γ ⊢ s1 ≈_L s2 shows [e] s1 = [e] s2

proof –

from Γ ⊢ e : Low have fv e ⊆ dom Γ by (auto dest: typeableFreevars)

have (∀x∈fv e. Γ x = Some Low)

proof

fix x assume x ∈ fv e

with Γ ⊢ e : Low show Γ x = Some Low by (auto intro: exprTypingLow)

qed

with ⟨Γ ⊢ s1 ≈_L s2⟩ show ?thesis by (rule interpretLow)

qed

lemma assignNIhighlemma:

assumes Γ ⊢ s1 ≈_L s2 and Γ V = Some High and s1’ = s1(V:= [e] s1)

and s2’ = s2(V:= [e] s2)

shows Γ ⊢ s1’ ≈_L s2’

proof

{ fix V’ assume V’ ∈ dom Γ and Γ V’ = Some Low

from Γ ⊢ s1 ≈_L s2. Γ V’ = 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 Γ V’ = Some Low ⟨Γ V = Some High⟩ have False by simp

thus ?thesis by simp

next

case False

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

SeqCompositionality
compositionality is no longer valid in case of concurrency

assignNIlowlemma

⟨
from Γ
assume s1 s2 s1
shows nonInterference
thus ?thesis
⟩→∗⟨
from Γ
assume s1 s2 s1
⟨
from Γ
assume s1 s2 s1
have s1 V = s2 V by auto
with ⟨s1 V = s2 V⟩ show ?thesis by simp
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 −
{ fix V′ assume V′ ∈ dom Γ and Γ V′ = Some Low
  from Γ ⊢ s1 ≈L s2 and Γ V′ = 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 ⟨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 ⊢ 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
next
  case False
  with ⟨s1′ = s1(V := [e] s1), s2′ = s2(V := [e] s2)⟩
  have s1′ V′ = s1 V′ and s2′ V′ = s2 V′ by auto
  with False ⟨s1′ V′ = s1 V′, s2′ V′ = s2 V′⟩ 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⟩ →∗ ⟨Skip,s1′⟩ obtain s1″ where ⟨c1,s1⟩ →∗ ⟨Skip,s1″⟩
and ⟨c2,s1″⟩ →∗ ⟨Skip,s1″⟩ by(auto dest:Seq-reds)
from ⟨c1;;c2,s2⟩ →∗ ⟨Skip,s2′⟩ obtain s2″ where ⟨c1,s2⟩ →∗ ⟨Skip,s2″⟩

17
lemma WhileStepInduct:
  assumes while:\langle\text{while } (b) \ c, s_1 \rangle \rightarrow \langle \text{Skip}, s_2 \rangle
  and body:\langle s_2, (c, s_1) \rangle \rightarrow \langle \text{Skip}, s_2 \rangle\quad \Rightarrow \quad \Gamma \vdash s_1 \approx_L s_2 \quad \text{and} \quad \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 s_1 s_2)
  from body:\langle c, s_1 \rangle \rightarrow \langle \text{Skip}, s_3 \rangle\quad \text{have} \quad \Gamma \vdash s_1 \approx_L s_3 \quad \text{by simp}
with \langle \Gamma \vdash s_3 \approx_L s_2 \rangle\quad \text{show} \quad \?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 \langle \text{High}, c \rangle \vdash c \quad \text{and} \quad \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\quad \text{have} \quad s_1 = s_2 \quad \text{by } (\text{auto dest:Skip-reds})
  thus \?case by (simp add:lowEquiv-def)
next
case \langle \text{LAss } V \ e \rangle
  from \langle \text{High} \vdash V := e \rangle\quad \text{have} \quad \Gamma \vdash \text{Some High} \quad (\text{auto elim:secComTyping.cases})
  from \langle V := e, s_1 \rangle \rightarrow \langle \text{Skip}, s_2 \rangle\quad \text{have} \quad s_2 = s_1(V := \llbracket e \rrbracket s_1) \quad (\text{auto intro:LAss-reds})
  \{ \text{fix } V' \quad \text{assume} \quad V' \in \text{dom } \Gamma \quad \text{and} \quad \Gamma \vdash V' = \text{Some Low} \}
  \quad \quad \quad \text{have} \quad s_1 V' = s_2 V'
proof (cases V' = V)
case True
  with \langle \Gamma \vdash V' = \text{Some Low} \rangle\quad \langle \Gamma \vdash \text{Some High} \rangle\quad \text{have} \quad \text{False} \quad \text{by simp}
  thus \?thesis by simp
next
case False
  with \langle s_2 = s_1(V := \llbracket e \rrbracket s_1) \rangle\quad \text{show} \quad \?thesis by simp
qed

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

next

case (Seq \( c\) 1 c2)

note IH1 = \( \text{l}s1 \, s2. \, [\text{\( \Gamma \)}, \text{High} \vdash \, c\,; \, \{c\,s1\} \rightarrow \{\text{Skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_{L} s2 \)

note IH2 = \( \text{l}s1 \, s2. \, [\text{\( \Gamma \)}, \text{High} \vdash \, c\,; \, \{c\,s1\} \rightarrow \{\text{Skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_{L} s2 \)

from \( \Gamma, \text{High} \vdash \, c\,; c2 \) have \( \Gamma, \text{High} \vdash \, c\) and \( \Gamma, \text{High} \vdash \, c2 \)

by (auto elim: secComTyping.cases)

from \( \langle c1;; c2, s1\rangle \rightarrow * \{\text{Skip}, s2\}\rangle \)

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 \( (c1, s1) \rightarrow \{\text{Skip}, s3\} \) and \( (c2, s3) \rightarrow \{\text{Skip}, s2\} \) by auto

from IH1[OF \( \Gamma, \text{High} \vdash \, c\,; \, \langle c1, s1\rangle \rightarrow \{\text{Skip}, s3\} \)]

have \( \Gamma \vdash s1 \approx_{L} s3 \) by simp

from IH2[OF \( \Gamma, \text{High} \vdash \, c\,; \, \langle c2, s3\rangle \rightarrow \{\text{Skip}, s2\} \)]

have \( \Gamma \vdash s3 \approx_{L} s2 \) by simp

from \( \Gamma \vdash s1 \approx_{L} s3 \) and \( \Gamma \vdash s3 \approx_{L} s2 \) show \( \text{case} \)

by (auto intro: lowEquivTransitive)

next

case (Cond \( b\) c1 c2)

note IH1 = \( \text{l}s1 \, s2. \, [\text{\( \Gamma \)}, \text{High} \vdash \, c\,; \, \{c\,s1\} \rightarrow \{\text{Skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_{L} s2 \)

note IH2 = \( \text{l}s1 \, s2. \, [\text{\( \Gamma \)}, \text{High} \vdash \, c\,; \, \{c\,s1\} \rightarrow \{\text{Skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_{L} s2 \)

from \( \Gamma, \text{High} \vdash \text{if} \ (b) \ c1 \ else \ c2 \) have \( \Gamma, \text{High} \vdash \, c\) and \( \Gamma, \text{High} \vdash \, c2 \)

by (auto elim: secComTyping.cases)

from \( \langle \text{if} \ (b) \ c1 \ else \ c2, s1\rangle \rightarrow * \{\text{Skip}, s2\}\rangle \)

have \( \langle b \rangle, s1 = \text{Some true} \lor \langle b \rangle, s1 = \text{Some false} \) by (auto dest: Cond-True-or-False)

thus \( \text{case} \)

proof

assume \( \langle b \rangle, s1 = \text{Some true} \)

with \( \langle \text{if} \ (b) \ c1 \ else \ c2, s1\rangle \rightarrow * \{\text{Skip}, s2\}\rangle \langle c1, s1\rangle \rightarrow * \langle\text{Skip}, s2\rangle \)

by (auto intro: CondTrue-reds)

from IH1[OF \( \Gamma, \text{High} \vdash \, c\,; \, \langle c1, s1\rangle \rightarrow \text{this} \)] show \( \text{thesis} \)

next

assume \( \langle b \rangle, s1 = \text{Some false} \)

with \( \langle \text{if} \ (b) \ c1 \ else \ c2, s1\rangle \rightarrow * \{\text{Skip}, s2\}\rangle \langle c2, s1\rangle \rightarrow * \langle\text{Skip}, s2\rangle \)

by (auto intro: CondFalse-reds)

from IH2[OF \( \Gamma, \text{High} \vdash \, c\,; \, \langle c2, s1\rangle \rightarrow \text{this} \)] show \( \text{thesis} \)

qed

next

case (While \( b\) c')

note IH = \( \text{l}s1 \, s2. \, [\text{\( \Gamma \)}, \text{High} \vdash \, c'\; \{c',s1\} \rightarrow * \{\text{Skip},s2\}] \Rightarrow \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 this]

have \( \langle s1, s2\rangle \rightarrow * \{\text{Skip},s2\} \Rightarrow \Gamma \vdash s1 \approx_{L} s2 \)

with \( \langle \text{while} \ (b) \ c',s1\rangle \rightarrow * \{\text{Skip},s2\}\rangle \Gamma, \text{High} \vdash c' \)

show \( \text{case} \) by (auto dest: WhileStepInduct)

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

lemma \textit{CondLowCompositionality}:
\begin{itemize}
\item assumes \text{nonInterference}\ \Gamma\ \text{c1 and nonInterference}\ \Gamma\ \text{c2 and }\Gamma \vdash b : \text{Low}$
\item shows \text{nonInterference}\ \Gamma\ \text{if}\ (b)\ c1\ \text{else}\ c2$
\end{itemize}
\begin{proof}[rule \text{nonInterferenceI}]
\begin{itemize}
\item fix $s1\ s2\ s1'\ s2'$
\item assume $\Gamma \vdash s1 \approx_L s2$ and $\langle (b)\ c1\ \text{else}\ c2, s1 \rangle \rightarrow^\ast \langle \text{Skip}, s1' \rangle$
\item and $\langle (b)\ c1\ \text{else}\ c2, s2 \rangle \rightarrow^\ast \langle \text{Skip}, s2' \rangle$
\item from $\langle (b)\ c1\ \text{else}\ c2, s1 \rangle \rightarrow^\ast \langle \text{Skip}, s1' \rangle\ \text{have } [b]\ s1 = [b]\ s2$
\item by (auto intro: interpretLow2)
\item from $\langle (b)\ c1\ \text{else}\ c2, s1 \rangle \rightarrow^\ast \langle \text{Skip}, s1' \rangle\ \text{have } [b]\ s1 = \text{Some true by (auto dest: Cond-True-or-False)}$
\item thus $\Gamma \vdash s1' \approx_L s2'$
\item proof
\begin{itemize}
\item assume $[b] \ s1 = \text{Some true}$
\item with $[b] \ s1 = [b] \ s2\ \text{have } [b] \ s2 = \text{Some true by (auto intro: CondTrue-reds)}$
\item from $[b] \ s1 = \text{Some true}\ \langle (b)\ c1\ \text{else}\ c2, s1 \rangle \rightarrow^\ast \langle \text{Skip}, s1' \rangle\ \text{have } (c1, s1) \rightarrow^\ast \langle \text{Skip}, s1' \rangle \text{ by (auto intro: CondTrue-reds)}$
\item from $[b] \ s2 = \text{Some true}\ \langle (b)\ c1\ \text{else}\ c2, s2 \rangle \rightarrow^\ast \langle \text{Skip}, s2' \rangle\ \text{have } (c1, s2) \rightarrow^\ast \langle \text{Skip}, s2' \rangle \text{ by (auto intro: CondTrue-reds)}$
\item with $\langle (c1, s1) \rightarrow^\ast \langle \text{Skip}, s1' \rangle\ \text{nonInterference } \Gamma \ c1\ \text{show } ?\text{thesis by (auto simp: nonInterference-def)}$
\item next
\begin{itemize}
\item assume $[b] \ s1 = \text{Some false}$
\item with $[b] \ s1 = [b] \ s2\ \text{have } [b] \ s2 = \text{Some false by (auto intro: CondTrue-reds)}$
\item from $[b] \ s1 = \text{Some false}\ \langle (b)\ c1\ \text{else}\ c2, s1 \rangle \rightarrow^\ast \langle \text{Skip}, s1' \rangle\ \text{have } (c2, s1) \rightarrow^\ast \langle \text{Skip}, s1' \rangle \text{ by (auto intro: CondFalse-reds)}$
\item from $[b] \ s2 = \text{Some false}\ \langle (b)\ c1\ \text{else}\ c2, s2 \rangle \rightarrow^\ast \langle \text{Skip}, s2' \rangle$
\end{itemize}
\end{itemize}
\end{itemize}
\begin{proof}[qed]
\end{proof}
have \((c_2,s_2) \rightarrow\ast \langle \text{Skip},s_2' \rangle\) by (auto intro:CondFalse-reds)
with \((c_2,s_1) \rightarrow\ast \langle \text{Skip},s_1' \rangle\): \(\Gamma \vdash s_1 \approx_L s_2\): \(\text{nonInterference } \Gamma c_2\)
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 \(\langle \text{while } (b) \ c',s_1 \rangle \rightarrow\ast \langle \text{Skip},s_1' \rangle\)
and \(\langle \text{while } (b) \ c',s_2 \rangle \rightarrow\ast \langle \text{Skip},s_2' \rangle\)
show \(\Gamma \vdash s_1' \approx_L s_2'\)
proof
from \(\Gamma;\text{High} \vdash \text{while } (b) \ c'\): \(\langle \text{while } (b) \ c',s_1 \rangle \rightarrow\ast \langle \text{Skip},s_1' \rangle\)
have \(\Gamma \vdash s_1 \approx_L s_1'\) by (auto dest:highBodies)
from \(\Gamma;\text{High} \vdash \text{while } (b) \ c'\): \(\langle \text{while } (b) \ c',s_2 \rangle \rightarrow\ast \langle \text{Skip},s_2' \rangle\)
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:lowEquivTransitive)
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 \(\text{thesis}\) by (auto intro:lowEquivTransitive)
qed

lemma WhileLowStepInduct:
assumes \(\text{while1: } \langle \text{while } (b) \ c',s_1 \rangle \rightarrow\ast \langle \text{Skip},s_1' \rangle\)
and \(\text{while2: } \langle \text{while } (b) \ c',s_2 \rangle \rightarrow\ast \langle \text{Skip},s_2' \rangle\)
and \(\Gamma \vdash b ; \text{Low}\) and \(\text{body};\Gamma;\text{le}\) : \(\Gamma \vdash s_1 \approx_L s_2\)
shows \(\Gamma \vdash s_1' \approx_L s_2'\)
using \(\text{while1 le while2}\)
proof (induct arbitrary:s2 rule:while-reds-induct)
case (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 \(\langle \text{while } (b) \ c',s_2 \rangle \rightarrow\ast \langle \text{Skip},s_2' \rangle\) 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 \(IH = \langle s_2'' \rangle, \Gamma \vdash s_1' \approx_L s_2''\): \(\langle \text{while } (b) \ c',s_2' \rangle \rightarrow\ast \langle \text{Skip},s_2' \rangle\)
\(\quad \Longrightarrow \Gamma \vdash s_1' \approx_L s_2'\)
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 \( \text{auto dest: WhileTrue-reds} \)
from body[\( \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)]
\( \Gamma \vdash s1 \approx_L s2 \) have \( \Gamma \vdash s1'' \approx_L s2'' \).
from IH[\( \langle \text{while} \ (b) \ c',s2'' \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)] show \( ?\text{case} \).
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\( \text{rule nonInterferenceI} \)
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 \( \langle \text{while} \ (b) \ c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
\( \text{fix} \ s1 \ s2 \ s1'' \ s2'' \)
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\( \text{auto simp: nonInterference-def} \)
}\)

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 \( \text{auto} \)
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\( \text{auto intro: WhileLowStepInduct} \)

qed

and now: the main theorem:

**theorem** secTypeImpliesNonInterference:
\( \Gamma . T \vdash c \Rightarrow \text{nonInterference} \ \Gamma \ c \)
proof\( \text{induct} \ c \ \text{arbitrary}: \text{T rule: com.induct} \)
case \( \text{Skip} \)
show \( ?\text{case} \)
proof\( \text{rule nonInterferenceI} \)
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\( \text{auto dest: Skip-reds} \)
from \( \langle \text{Skip},s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) have \( s2 = s2' \) by\( \text{auto dest: Skip-reds} \)
from \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle s1 \rangle = s1' \) and \( s2 = s2' \)
show \( \Gamma \vdash s1' \approx_L s2' \) by simp
qed
next
case \( \text{LA}ss \ V \ e \)
from \( \Gamma . T \vdash V := e \)
have \( \text{varPrem}(\Gamma \ V = \text{Some High}) \lor (\Gamma \ V = \text{Some Low} \ \wedge \ \Gamma \vdash c : \text{Low} \ \wedge \ T = \text{Low}) \)
by\( \text{auto elim: secComTyping_cases} \)

22
show \?case
proof (rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle V:=e,s1 \rangle \rightarrowiid \langle \text{Skip},s1' \rangle \) and \( \langle V:=e,s2 \rangle \rightarrowiid \langle \text{Skip},s2' \rangle \)
  from \( \langle V:=e,s1 \rangle \rightarrowiid \langle \text{Skip},s1' \rangle \) have \( s1' = s1(V:=\lceil e \rceil s1) \) by (auto intro:LAss-reds)
  from \( \langle V:=e,s2 \rangle \rightarrowiid \langle \text{Skip},s2' \rangle \) have \( s2' = s2(V:=\lceil e \rceil s2) \) by (auto intro:LAss-reds)
  from \( \text{varprem show } \Gamma \vdash s1' \approx_L s2' \)
  proof
    assume \( \Gamma \vdash \Gamma \vdash e : \text{Low} \wedge T = \text{Low} \)
    with \( \Gamma \vdash s1 \approx_L s2 \) \( \langle s1' = s1(V:=\lceil e \rceil s1) \rangle \langle s2' = s2(V:=\lceil e \rceil s2) \rangle \)
    show \?thesis by (auto intro:assignNIlowlemma)
  qed
next
  case (Seq c1 c2)
  note IH1 = \( \langle T. \Gamma,T \vdash c1 \Longrightarrow \text{nonInterference } \Gamma \vdash c1 \rangle \)
  note IH2 = \( \langle T. \Gamma,T \vdash c2 \Longrightarrow \text{nonInterference } \Gamma \vdash c2 \rangle \)
  show \?case
  proof (cases T)
    case High
    with \( \Gamma,T \vdash c1;\vdash c2 \) have \( \Gamma,\text{High} \vdash c1 \) and \( \Gamma,\text{High} \vdash c2 \)
    by (auto elim:secComTyping.cases)
    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)
  next
    case Low
    with \( \Gamma,T \vdash c1;\vdash c2 \)
    have \( \langle \Gamma,\text{Low} \vdash c1 \wedge \Gamma,\text{Low} \vdash c2 \rangle \vee \Gamma,\text{High} \vdash c1;\vdash c2 \)
    by (auto elim:secComTyping.cases)
    thus \?thesis
  proof
    assume \( \Gamma,\text{Low} \vdash c1 \wedge \Gamma,\text{Low} \vdash c2 \)
    hence \( \Gamma,\text{Low} \vdash c1 \) and \( \Gamma,\text{Low} \vdash c2 \) by simp-all
    from IH1[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 by (auto intro:SeqCompositionality)
  next
    assume \( \Gamma,\text{High} \vdash c1;\vdash c2 \)
    hence \( \Gamma,\text{High} \vdash c1 \) and \( \Gamma,\text{High} \vdash c2 \) by (auto elim:secComTyping.cases)
from IH1[\text{OF } \Gamma, \text{High} \vdash c1] have \text{nonInterference } \Gamma c1.

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

ultimately show ?thesis by(auto intro:SeqCompositionality)

qed

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

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

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

show ?thesis

proof (cases T)

case High

with \(\Gamma, T \vdash \text{if } (b) \ c1 \text{ else } c2\) show ?thesis

by(auto intro:CondHighCompositionality)

next
case Low

with \(\Gamma, T \vdash \text{if } (b) \ c1 \text{ else } c2\)

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

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

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

moreover
from IH2[\text{OF } \Gamma, \text{Low} \vdash c2] have \text{nonInterference } \Gamma 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
case (\text{While } b \ c')

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

show ?thesis

proof (cases T)

case High

with \(\Gamma, T \vdash \text{while } (b) \ c'\) show ?thesis by(auto intro:WhileHighCompositionality)

next
case Low

with \(\Gamma, T \vdash \text{while } (b) \ c'\)

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

by(auto elim:secComTyping.cases)

thus ?thesis

proof

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

moreover

from $IH[OF \Gamma, Low \vdash c']$ have nonInterference $\Gamma \vdash c'$.

ultimately show thesis by (auto intro: WhileLowCompositionality)

next

assume $\Gamma, High \vdash \text{while} (b) \ c'$

thus thesis by (auto intro: WhileHighCompositionality)

qed

end

theory Execute

imports secTypes

begin

3 Executing the small step semantics

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

thm red.equation

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

lemmas [code-pred-intro] = typeVal[where lev = Low] typeVal[where lev = High]


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

proof –

case secExprTyping

from secExprTyping.prems show thesis

proof

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

from secExprTyping(1-2) this show thesis by (cases lev) auto

next

fix $\Gamma \ Vn \ lev$

assume $x = \Gamma \ xa = \text{Var \ Vn \ xb = lev} \ \Gamma \ Vn = \text{Some \ lev}$

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

next

fix $\Gamma \ c1 \ e2 \ bop$

assume $x = \Gamma \ xa = c1 \ bop \ e2 \ xb = \text{Low}$

$\Gamma \vdash c1 : Low \ \Gamma \vdash e2 : Low$

from secExprTyping(4) this show thesis by auto

next
\textbf{fix} \( \Gamma \ e1 \ e2 \ lev \ bop \)
\textbf{assume} \( x = \Gamma \ xa = e1 <bop> e2 \ xb = \text{High} \)
\( \Gamma \vdash e1 : \text{High} \ \Gamma \vdash e2 : \text{lev} \)
\textbf{from} \textit{secExprTyping}(5−6) \textbf{this show thesis by} \ (\text{cases} \ \text{lev}) \ \text{(auto)}
\textbf{next}
\textbf{fix} \( \Gamma \ e1 \ e2 \ lev \ bop \)
\textbf{assume} \( x = \Gamma \ xa = e1 <bop> e2 \ xb = \text{High} \)
\( \Gamma \vdash e1 : \text{lev} \ \Gamma \vdash e2 : \text{High} \)
\textbf{from} \textit{secExprTyping}(6−7) \textbf{this show thesis by} \ (\text{cases} \ \text{lev}) \ \text{(auto)}
\textbf{qed}
\textbf{next}
\textbf{lemmas} \ [\textit{code-pred-intro} = \textit{typeSkip[where} \ T=\text{Low}] \ \textit{typeSkip[where} \ T=\text{High}] \]
\textit{typeAssH[where} \ T = \text{Low}] \ \textit{typeAssH[where} \ T=\text{High}] \)
\textit{typeAssL} \ \textit{typeSeq} \ \textit{typeWhile} \ \textit{typeIf} \ \textit{typeConvert}
\textbf{code-pred} \ (\text{modes}: i => o => i => \text{bool as} \ \textit{compute-secComTyping}, \ i => i => i => \text{bool as} \ \textit{check-secComTyping}) \ \textit{secComTyping}
\textbf{proof} –
\textbf{case} \ \textit{secComTyping}
\textbf{from} \ \textit{secComTyping.prems} \textbf{show thesis}
\textbf{proof}
\textbf{fix} \( \Gamma \ \ T \ \textbf{assume} \ x = \Gamma \ xa = T \ xb = \text{Skip} \)
\textbf{from} \ \textit{secComTyping}(1−2) \textbf{this show thesis by} \ (\text{cases} \ \text{T}) \ \text{(auto)}
\textbf{next}
\textbf{fix} \( \Gamma \ \ V \ \ T \ e \ \textbf{assume} \ x = \Gamma \ xa = T \ xb = V:=e \ \Gamma \ V = \text{Some High} \)
\textbf{from} \ \textit{secComTyping}(3−4) \textbf{this show thesis by} \ (\text{cases} \ \text{T}) \ \text{(auto)}
\textbf{next}
\textbf{fix} \( \Gamma \ \ e \ \ V \)
\textbf{assume} \( x = \Gamma \ xa = \text{Low} \ xb = V:=e \ \Gamma \vdash e : \text{Low} \ \Gamma \ V = \text{Some Low} \)
\textbf{from} \ \textit{secComTyping}(5) \textbf{this show thesis by} \ \text{(auto)}
\textbf{next}
\textbf{fix} \( \Gamma \ \ T \ c1 \ c2 \)
\textbf{assume} \( x = \Gamma \ xa = T \ xb = \text{Seq} \ c1 \ c2 \ \Gamma ,T \vdash \ \Gamma ,T \vdash \ c1 \)
\textbf{from} \ \textit{secComTyping}(6) \textbf{this show thesis by} \ \text{(auto)}
\textbf{next}
\textbf{fix} \( \Gamma \ \ b \ \ T \ c \)
\textbf{assume} \( x = \Gamma \ xa = T \ xb = \text{while} (b) \ c \ \Gamma \vdash b : T \ \Gamma ,T \vdash c \)
\textbf{from} \ \textit{secComTyping}(7) \textbf{this show thesis by} \ \text{(auto)}
\textbf{next}
\textbf{fix} \( \Gamma \ \ b \ \ T \ c1 \ c2 \)
\textbf{assume} \( x = \Gamma \ xa = T \ xb = \text{if} (b) \ c1 \ \text{else} \ c2 \ \Gamma \vdash b : T \ \Gamma ,T \vdash \ c1 \ \Gamma ,T \vdash c2 \)
\textbf{from} \ \textit{secComTyping}(8) \textbf{this show thesis by} \ \text{blast}
\textbf{next}
\textbf{fix} \( \Gamma \ e \)
\textbf{assume} \( x = \Gamma \ xa = \text{Low} \ xb = c \ \Gamma ,\text{High} \vdash c \)
\textbf{from} \ \textit{secComTyping}(9) \textbf{this show thesis by} \ \text{blast}
\textbf{qed}
\textbf{qed}

26
3.1 An example taken from Volpano, Smith, Irvine

definition \( com = \text{if} \ (\text{Var} \ "x" \not<:\not \text{Eq} \ \text{Val} \text{(Intg} \ 1)) \ ("y" := \ \text{Val} \text{(Intg} \ 0)) \ \text{else} \ ("y" := \ \text{Val} \text{(Intg} \ 0)) \)

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

values \{ T. \ \text{Env} \vdash (\text{Var} \ "x" \not<:\not \text{Eq} \ \text{Val} \text{(Intg} \ 1)): T \} \)

value \( \text{Env}, \text{High} \vdash \text{com} \)

value \( \text{Env}, \text{Low} \vdash \text{com} \)

values \{ T. \ \text{Env}, T \vdash \text{com} \} \)

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

value \( \text{Env}', \text{Low} \vdash \text{com} \)

value \( \text{Env}', \text{High} \vdash \text{com} \)

values \{ T. \ \text{Env}, T \vdash \text{com} \} \)

References
