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**  
   1.1 Variables and Values  
   1.2 Expressions and Commands  
   1.3 State  
   1.4 Small Step Semantics  

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

3. **Executing the small step semantics**  
   3.1 An example taken from Volpano, Smith, Irvine
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:
\(\langle\text{Skip},s\rangle \rightarrow^* \langle c',s' \rangle \Longrightarrow s = s' \land c' = \text{Skip}\)
by (blast elim: converse-rtranclpE red_cases)

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

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

lemma Seq-reds:
assumes \(\langle c_1;c_2,s\rangle \rightarrow^* \langle\text{Skip},s' \rangle\)
obtains \(s''\) where \(\langle c_1,s \rangle \rightarrow^* \langle\text{Skip},s'' \rangle\) and \(\langle c_2,s'' \rangle \rightarrow^* \langle\text{Skip},s' \rangle\)
proof
  have \(\exists s''. \langle c_1,s \rangle \rightarrow \langle\text{Skip},s'' \rangle \land \langle c_2,s'' \rangle \rightarrow \langle\text{Skip},s' \rangle\)
  proof
    \{ fix \(c\ c'\)
      assume \(\langle c,s \rangle \rightarrow \langle c',s' \rangle\) and \(c = c_1;c_2\) and \(c' = \text{Skip}\)
      hence \(\exists s''. \langle c_1,s \rangle \rightarrow \langle\text{Skip},s'' \rangle \land \langle c_2,s'' \rangle \rightarrow \langle\text{Skip},s' \rangle\)
    proof (induct arbitrary: \(c_1\) rule: converse-rtranclp-induct2)
      case refl thus \(?case\) by simp
    next
      case (step \(c\ s\ c''\ s''\))
      note IH = \(\forall c_1. \ [c'' = c_1;c_2; c' = \text{Skip}] \Longrightarrow \exists sx. \langle c_1,s'' \rangle \rightarrow \langle\text{Skip},sx \rangle \land \langle c_2,sx \rangle \rightarrow \langle\text{Skip},s' \rangle\)
      from step
      have \(\langle c_1;c_2,s \rangle \rightarrow \langle c'',s'' \rangle\) by simp
      hence \(\langle c_1 = \text{Skip} \land c'' = c_2 \land s = s''\) \(\lor\)
      \(\exists c'_1. \langle c_1\ s' \rangle \rightarrow \langle c'_1,s'' \rangle \land c'' = c_1';c_2\)
      by (auto elim:red_cases)
      thus \(?case\)
    proof
      assume \(c_1 = \text{Skip} \land c'' = c_2 \land s = s''\)
      with \(\langle c'',s'' \rangle \rightarrow \langle c',s' \rangle\) \(c' = \text{Skip}\)
      show \(?thesis\) by auto
    next
      assume \(\exists c'_1. \langle c_1,s \rangle \rightarrow \langle c'_1,s'' \rangle \land c'' = c_1';c_2\)
      then obtain \(c_1'\) where \(\langle c_1,s \rangle \rightarrow \langle c'_1,s'' \rangle\) and \(c'' = c_1';c_2\) by blast
      from IH \([OF \ (c'' = c_1';c_2; c' = \text{Skip}]\)
      obtain \(sx\) where \(\langle c_1,s' \rangle \rightarrow \langle\text{Skip},sx \rangle\) and \(\langle c_2,sx \rangle \rightarrow \langle\text{Skip},s' \rangle\)
      by blast
      from \(\langle c_1,s \rangle \rightarrow \langle c'_1,s'' \rangle\) \(\langle c'_1,s'' \rangle \rightarrow \langle\text{Skip},sx \rangle\)
lemma Cond-True-or-False:
\[
\langle \text{if } \langle b \rangle \rangle^* \boldsymbol{(\text{Skip}, s') } \Rightarrow [b] \ s = \text{Some true } \lor [b] \ s = \text{Some false}
\]
by(induct c: = if (b) c1 else c2 s rule: converse-rtranclp-induct2)(auto elim:red_cases)

lemma CondFalse-reds:
\[
\langle \text{if } \langle b \rangle \rangle^* \boldsymbol{(\text{Skip}, s') } \Rightarrow [b] \ s = \text{Some false } \Rightarrow \langle c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle
\]
by(induct c: = if (b) c1 else c2 s rule: converse-rtranclp-induct2)(auto elim:red_cases)

lemma WhileTrue-reds:
\[
\langle \text{while } \langle b \rangle \rangle^* \boldsymbol{(\text{Skip}, s') } \Rightarrow [b] \ s = \text{Some true } \Rightarrow \exists sz. \ \langle cz, sx \rangle \rightarrow \langle \text{Skip}, s' \rangle
\]
proof(induct while (b) cx s rule: converse-rtranclp-induct2)
  case step thus \( ? \) case by(auto elim:red_cases dest: Skipreds)
qed

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

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

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

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

proof(induct $c_1;c_2$ s n $\text{Skip}$ s' arbitrary; $c_1$ rule:red-n.induct)

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

note $IH = \langle \land c_1. c'' = c_1;c_2$

$\implies \exists i j s. \langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s \rangle \land \langle c_2, s \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1$

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

have $(c_1 = \text{Skip} \land c'' = c_2 \land s = s'') \lor

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

by(induct $c_1;c_2$ - - 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 $\langle c_1 = \text{Skip} \rangle$ have $\langle c_1, s \rangle \rightarrow^0 \langle \text{Skip}, s \rangle$ by(fastforce intro:red-n-Base)

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

show $?thesis$ by(rule-tac $x = 0$ in exfI) auto

next

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

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

from $IH[OF \langle c'' = c_1';c_2 \rangle]$ obtain $i j s x$

where $\langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, s x \rangle$ and $\langle c_2, s x \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$

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

from $\langle \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \rangle \langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, s x \rangle$

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

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

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

qed

with that show $?thesis$ by blast

qed

lemma while-red-nE:

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

$\sqrt{(\exists i j s''. (b) \text{ } s = \text{Some false }\land s = s'' \land n = 1) \lor$

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

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

proof(induct while (b) cx s n $\text{Skip}$ s' rule:red-n.induct)

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

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

have $\langle s \rangle = \text{Some false }\land c'' = \text{Skip }\land s'' = s \rangle \lor$

$\langle \langle \text{(while } (b) \text{ } cx, s'\rangle \rightarrow (c'', s'') \rangle$)

by(induct while (b) cx - - rule:red-induct) auto

thus $?case$

proof


proof

next

thus \( P \ s' s'' \)

\begin{verbatim}
proof

next

thus \( P \ s' s'' \)

\end{verbatim}
proof \langle while \ (b) \ cx,s' \rangle \rightarrow \langle \text{Skip},s' \rangle \ \text{this} \ \textbf{show} \ ?\text{thesis}.

\textbf{qed}

\textbf{proof}

\textbf{lemma} \ \text{reds-to-red-n}:(c,s) \rightarrow^{*} \langle c',s' \rangle \Longrightarrow \exists \ n. \ (c,s) \rightarrow^{n} \langle c',s' \rangle

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

\textbf{lemma} \ \text{red-n-to-reds}:(c,s) \rightarrow^{n} \langle c',s' \rangle \Longrightarrow \langle c,s \rangle \rightarrow^{\langle c',s' \rangle}

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

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

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

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

\langle \text{while} \ (b) \ cx,s'' \rangle \rightarrow^{*} \langle \text{Skip},s' \rangle; \ P \ s'' \ s \Longrightarrow P \ s \ s'\ \text{apply(} \text{drule} \ \text{reds-to-red-n} \text{, clarsimp)}

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

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

\textbf{lemma} \ \text{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

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

\textbf{case} \ \langle \text{SeqRed} \ c_1 \ s \ s_1 \ s' \ c_2' \rangle

\textbf{note} \ \text{IH} = \langle \exists \ c_2. \ \langle c_1,s \rangle \rightarrow \langle c_2,s_2 \rangle \Longrightarrow c_1 = c_2 \ \land \ s = s_2 \rangle

\textbf{from} \ \langle c_1';c_2',s \rangle \rightarrow \langle c_2,s_2 \rangle \ \textbf{have} \ c_1 = \text{Skip} \ \lor \ \exists \ cx. \ c_2 = cx;c_2' \ \land \ c_1 \rightarrow \langle cx,s_2 \rangle \ \langle cx,s_2 \rangle \ \text{by fastforce elim:red.cases)}

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

\textbf{proof}

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

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

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

\textbf{next}

\textbf{assume} \ \exists \ cx. \ c_2 = cx;c_2' \ \land \ c_1 \rightarrow \langle cx,s_2 \rangle

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

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

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

\textbf{qed}

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

\textbf{theorem} \ \text{reds-det}:

\langle (c,s) \rightarrow^{*} \langle \text{Skip},s_1 \rangle; \ \langle c,s \rangle \rightarrow^{*} \langle \text{Skip},s_2 \rangle \rangle \Longrightarrow s_1 = s_2

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

\textbf{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
| typeIf: \[ \Gamma \vdash b : T ; \Gamma,T \vdash c1 ; \Gamma,T \vdash c2 \] \implies \Gamma,T \vdash \text{if} (b) c1 \text{ else } c2 
| typeConvert: \Gamma, \text{High} \vdash c \implies \Gamma, \text{Low} \vdash c 

2.2 Lemmas concerning expressions

lemma exprTypeable:
assumes \( Hv \subseteq dom \Gamma \) obtains \( T \) where \( \Gamma \vdash e : T \)
proof
from \( Hv \subseteq dom \Gamma \) have \( \exists T . \Gamma \vdash e : T \)
proof (induct \( e \))
case (Val V)
have \( \Gamma \vdash \text{Val} V : \text{Low} \) by (rule typeVal)
thus \( \exists \text{case} \) by (rule exI)
next
case (Var V)
have \( V \in Hv (\text{Var} V) \) by simp
with \( Hv (\text{Var} V) \subseteq dom \Gamma \) have \( V \in dom \Gamma \) by simp
then obtain \( T \) where \( \Gamma \vdash V = \text{Some} T \) by auto
hence \( \Gamma \vdash \text{Var} V : T \) by (rule typeVar)
thus \( \exists \text{case} \) by (rule exI)
next
case (BinOp \( e1 \ bop e2 \))
note \( IH1 = (fv \subseteq dom \Gamma \implies \exists T . \Gamma \vdash e1 : T) \)
note \( IH2 = (fv \subseteq dom \Gamma \implies \exists T . \Gamma \vdash e2 : T) \)
from \( fv (e1 \ bop e2) \subseteq dom \Gamma \)
have \( Hv \subseteq dom \Gamma \) and \( Hv \subseteq dom \Gamma \) by auto
from \( IH1[OF Hv \subseteq dom \Gamma] \) obtain \( T1 \) where \( \Gamma \vdash e1 : T1 \) by auto
from \( IH2[OF Hv \subseteq dom \Gamma] \) obtain \( T2 \) where \( \Gamma \vdash e2 : T2 \) by auto
show \( \exists \text{case} \)
proof (cases \( T1 \))
case Low
show \( \exists \text{thesis} \)
proof (cases \( T2 \))
case Low
with \( \Gamma \vdash e1 : T1 \vdash e2 : T2 \) \( T1 = \text{Low} \)
have \( \Gamma \vdash e1 \ bop e2 : \text{Low} \) by (simp add: typeBinOp1)
thus \( \exists \text{thesis} \) by (rule exI)
next
case High
with \( \Gamma \vdash e1 : T1 \vdash e2 : T2 \) \( T1 = \text{Low} \)
have \( \Gamma \vdash e1 \ bop e2 : \text{High} \) by (simp add: typeBinOp3)
thus \( \exists \text{thesis} \) by (rule exI)
qed
next
case High
with \( \Gamma \vdash e1 : T1 \vdash e2 : T2 \)
have \( \Gamma \vdash e1 \ bop e2 : \text{High} \) by (simp add: typeBinOp2)

11
thus ?thesis by (rule exI)
qed
qed
with that show ?thesis by blast
qed

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

lemma exprTypingHigh:
assumes Γ ⊢ e : T and x ∈ fv e and Γ x = Some High
shows Γ ⊢ e : High
using assms
proof (induct e arbitrary: T)
case (Val V) show ?case by (rule typeVal)
next
case (Var V) from ⟨x ∈ fv (Var V)⟩ have x = V by simp
with ⟨Γ ⊢ e : T1⟩ obtain T1 where Γ ⊢ e : T1 by auto
then obtain T2 where Γ ⊢ e : T2 by auto
from ⟨x ∈ fv (e₁≪bop≫e₂)⟩ have x ∈ (fv e₁ ∪ fv e₂) by simp
hence x ∈ fv e₁ ∨ x ∈ fv e₂ by auto
thus ?case

proof
  assume x ∈ fv e₁
  from IH1[OF Γ ⊢ e₁ : T₁] this ⟨Γ x = Some High⟩ have Γ ⊢ e₁ : High.
  with Γ ⊢ e₂ : T₂ show ?thesis by (simp add: typeBinOp2)
next
  assume x ∈ fv e₂
  from IH2[OF Γ ⊢ e₂ : T₂] this ⟨Γ x = Some High⟩ have Γ ⊢ e₂ : High.
  with Γ ⊢ e₁ : T₁ show ?thesis by (simp add: typeBinOp3)
qed

lemma exprTypingLow:
assumes Γ ⊢ e : Low and x ∈ fv e shows Γ x = Some Low
using assms
proof (induct e)
  case (Val V)
  have \( \text{fv (Val V)} \subseteq \{\} \) by (rule FVc)
  with \( x \in \text{fv (Val V)} \) have False by auto
  thus \(?thesis\) by simp
next
  case (Var V)
  from \( x \in \text{fv (Var V)} \) have \( xV: x = V \) by simp
  from \( \Gamma \vdash \text{Var V : Low} \) have \( \Gamma \vdash V = \text{Some Low} \) by (auto elim:secExprTyping.cases)
  with \( xV \) show \(?thesis\) by simp
next
  case (BinOp e1 bop e2)
  note IH1 = \( \bigwedge T. \Gamma \vdash e1 : T \Rightarrow \text{fv e1} \subseteq \text{dom } \Gamma \)\)
  note IH2 = \( \bigwedge T. \Gamma \vdash e2 : T \Rightarrow \text{fv e2} \subseteq \text{dom } \Gamma \)\)
  from \( \Gamma \vdash e1 \ll bop \gg e2 : \text{Low} \) have \( \text{fv e1} \cup \text{fv e2} \subseteq \text{fv (BinOp e1 bop e2)} \) by (simp add:FVe)
  hence \( x \in \text{fv e1} \cup \text{fv e2} \) by auto
  thus \(?case\) proof
  
  fix \( x \)
  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

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 \( \text{fv (Val V)} \subseteq \{\} \) by (rule FVc)
  thus \(?case\) by simp
next
  case (Var V)
  show \(?case\)
  proof
    fix \( x \)
    assume \( x \in \text{fv (Var V)} \)
    hence \( x = V \) by simp
    from \( \Gamma \vdash \text{Var V : T} \) have \( \Gamma \vdash V = \text{Some T} \) 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 = \( \bigwedge T. \Gamma \vdash e1 : T \Rightarrow \text{fv e1} \subseteq \text{dom } \Gamma \)\)
note IH2 = (\{T. \Gamma \vdash e2 : T \implies \text{fv} e2 \subseteq \text{dom} \Gamma\)

show ?case

proof

fix x assume x \in \text{fv} (e1 \langle bop \rangle e2)

from \Gamma \vdash e1 \langle bop \rangle e2 : T; e2 \in \text{dom} s

have Q: (\{\exists T1. \Gamma \vdash e1 : T1\} \land (\{\exists T2. \Gamma \vdash e2 : T2\})

by (rule exprBinopTypeable)

then obtain T1 where \Gamma \vdash e1 : T1 by blast

from Q obtain T2 where \Gamma \vdash e2 : T2 by blast

from IH1[\{OF \Gamma \vdash e1 : T1\}] have \text{fv} e1 \subseteq \text{dom} \Gamma .

moreover

from IH2[\{OF \Gamma \vdash e2 : T2\}] have \text{fv} e2 \subseteq \text{dom} \Gamma .

ultimately have (\{\text{fv} e1\} \cup (\{\text{fv} e2\}) \subseteq \text{dom} \Gamma by auto

hence \text{fv} (e1 \langle bop \rangle e2) \subseteq \text{dom} \Gamma by (simp add: FVe)

with \langle x \in \text{fv} (e1 \langle bop \rangle e2) show x \in \text{dom} \Gamma by auto

qed

qed


lemma exprNotNone:

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

shows [e] s \neq None

using assms

proof (induct e arbitrary: \Gamma T s)

case (Val v)

show ?case by (simp add: Val)

next

case (Var V)

have [Var V] s = s V by (simp add: Var)

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

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

thus ?case by auto

next

case (BinOp e1 bop e2)

note IH1 = (\{\exists T1. \Gamma \vdash e1 : T1 \land \text{fv} e1 \subseteq \text{dom} s\}) \implies \langle e1 \rangle s \neq None

note IH2 = (\{\exists T2. \Gamma \vdash e2 : T2 \land \text{fv} e2 \subseteq \text{dom} s\}) \implies \langle e2 \rangle s \neq None

from \Gamma \vdash e1 \langle bop \rangle e2 : T; e2 \in \text{dom} s

have (\{\exists T1. \Gamma \vdash e1 : T1\} \land (\{\exists T2. \Gamma \vdash e2 : T2\})

by (rule exprBinopTypeable)

then obtain T1 T2 where \Gamma \vdash e1 : T1 and \Gamma \vdash e2 : T2 by blast

from \langle \text{fv} (e1 \langle bop \rangle e2) \subseteq \text{dom} s \rangle have \text{fv} e1 \cup \text{fv} e2 \subseteq \text{dom} s by (simp add: FVe)

hence \text{fv} e1 \subseteq \text{dom} s and \text{fv} e2 \subseteq \text{dom} s by auto

from IH1[\{OF \Gamma \vdash e1 : T1\}] have [e1] s \neq None .

moreover from IH2[\{OF \Gamma \vdash e2 : T2\}] have [e2] s \neq None .

ultimately show ?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. $\in \text{dom state}$), 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 $\Rightarrow$ state $\Rightarrow$ state $\Rightarrow$ bool $\quad$ (¬ $\vdash$ ¬ $\approx _L$)

where $\Gamma \vdash s_1 \approx _L s_2 \equiv \forall v \in \text{dom} \ \Gamma. \ \Gamma v = \text{Some Low} \rightarrow (s_1 v = s_2 v)$

**lemma** lowEquivReflexive: $\Gamma \vdash s_1 \approx _L s_1$

by (simp add: lowEquiv-def)

**lemma** lowEquivSymmetric: $\Gamma \vdash s_1 \approx _L s_2 = \Rightarrow \Gamma \vdash s_2 \approx _L s_1$

by (simp add: lowEquiv-def)

**lemma** lowEquivTransitive: $[[\Gamma \vdash s_1 \approx _L s_2; \Gamma \vdash s_2 \approx _L s_3]] = \Rightarrow \Gamma \vdash s_1 \approx _L s_3$

by (simp add: lowEquiv-def)

2.3.2 Non Interference

**definition** nonInterference :: typeEnv $\Rightarrow$ com $\Rightarrow$ bool $\quad$ where nonInterference $\Gamma c \equiv$

$(\forall s_1 s_2 s_1' s_2'. \ (\Gamma \vdash s_1 \approx _L s_2 \land \langle c, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \land \langle c, s_2 \rangle \rightarrow^{*} \langle \text{Skip}, s_2' \rangle) \rightarrow \Gamma \vdash s_1' \approx _L s_2')$

**lemma** nonInterferenceI: $[[\Gamma \vdash s_1 \approx _L s_2; \langle c, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_1' \rangle; \langle c, s_2 \rangle \rightarrow^{*} \langle \text{Skip}, s_2' \rangle]] = \Rightarrow \Gamma \vdash s_1' \approx _L s_2'$

by (auto simp: nonInterference-def)

**lemma** interpretLow: assumes $\Gamma \vdash s_1 \approx _L s_2$ and all: $\forall V \in \text{fv e}. \ \Gamma V = \text{Some Low}$

shows $[e] s_1 = [e] s_2$

using all

proof (induct e)

case (Val v)

show ?case by (simp add: Val)

next

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

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

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

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

thus ?case by auto

next

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

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

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

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

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

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

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

ultimately show ?case by (cases \([e1]\) \([e2]\), auto)

qed

lemma \( \text{interpretLow2}; \)

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

proof –

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

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

proof

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

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

qed

with \( \Gamma \vdash s1 \approxL s2 \) show \( ?thesis \) by (rule interpretLow)

qed

lemma \( \text{assignNIHighLemma}; \)

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

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

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

proof

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

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

by (auto simp add: lowEquiv-def)

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

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

case True

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

thus \( ?thesis \) by simp

next

case False

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

16
proof
−
SeqCompositionality

theorem compositionality is no longer valid in case of concurrency

assignNIlowlemma

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 ⟨ 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 ⟨ 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
  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’ = s2 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-reduces)
from ⟨ c1;;c2,s2 ⟩ →∗ ⟨ Skip,s2’ ⟩ obtain s2’’ where ⟨ c1,s2 ⟩ →∗ ⟨ Skip,s2’’ ⟩
and $\langle c_2, s_2'' \rangle \rightarrow^* \langle \text{Skip}, s_2'' \rangle$ by (auto dest:Seg-reds)
from $\Gamma \vdash s_1 \approx_L s_2 \quad \langle c_1, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle \quad \langle c_1, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle$
nonInterference $\Gamma \ c_1$
have $\Gamma \vdash s_1'' \approx_L s_2''$ by (auto simp:nonInterference-def)
with $\langle c_2, s_1'' \rangle \rightarrow^* \langle \text{Skip}, s_1'' \rangle \quad \langle c_2, s_2'' \rangle \rightarrow^* \langle \text{Skip}, s_2'' \rangle$ (nonInterference $\Gamma \ c_2$)
show $\Gamma \vdash s_1' \approx_L s_2'$ by (auto simp:nonInterference-def)
qed

lemma WhileStepInduct:
  assumes while: \langle while \ b \ c, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_2 \rangle
  and body: \forall s_2. \langle c, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_2 \rangle \Longrightarrow \Gamma \vdash s_1 \approx_L s_2 \text{ 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 s_1 s_2)
  from body: \langle 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: s_1 s_2 rule:com.induct)
case Skip
  from $\langle \text{Skip}, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_2 \rangle$ have $s_1 = s_2$ by (auto dest:Skip-reds)
  thus \?case by (simp add:lowEquiv-def)
next
case (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 $s_2 = s_1(V := \langle e \rangle s_1)$ 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 $\Gamma \ V' = \text{Some Low}$ \?thesis by simp
  thus \?thesis by simp
next
case False
  with $s_2 = s_1(V := \langle e \rangle s_1)$ show \?thesis by simp

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

next
case (Seq c1 c2)
note IH1 = (\A s1 s2. [\Gamma,High |- c1; \langle c1,s1 \rangle -> (\langle Skip,s2 \rangle)] ==\> \Gamma |- s1 \approx_L s2)
note IH2 = (\A s1 s2. [\Gamma,High |- c2; \langle c2,s1 \rangle -> (\langle Skip,s2 \rangle)] ==\> \Gamma |- s1 \approx_L s2)
from \Gamma,High |- c1;c2 have \Gamma,High |- c1 and \Gamma,High |- c2
  by(auto elim:secComTyping.cases)
from (\langle c1;c2,s1 \rangle -> (\langle Skip,s2 \rangle))
have \exists s3. (\langle c1,s1 \rangle -> (\langle Skip,s3 \rangle) \&\& (\langle c2,s3 \rangle -> (\langle Skip,s2 \rangle)) by(auto intro:Seq-ords)
then obtain s3 where (c1,s1) -> (\langle Skip,s3 \rangle) and (c2,s3) -> (\langle Skip,s2 \rangle) by auto
from IH1[OF \Gamma,High |- c1; \langle c1,s1 \rangle -> (\langle Skip,s3 \rangle)]
have \Gamma |- s1 \approx_L s3 by simp
from IH2[OF \Gamma,High |- c2; \langle c2,s3 \rangle -> (\langle Skip,s2 \rangle)]
have \Gamma |- s3 \approx_L s2 by simp
from \Gamma |- s1 \approx_L s3: \Gamma |- s3 \approx_L s2 show ?case
  by(auto intro:lowEquivTransitive)

next
case (Cond b c1 c2)
note IH1 = (\A s1 s2. [\Gamma,High |- c1; \langle c1,s1 \rangle -> (\langle Skip,s2 \rangle)] ==\> \Gamma |- s1 \approx_L s2)
note IH2 = (\A s1 s2. [\Gamma,High |- c2; \langle c2,s1 \rangle -> (\langle Skip,s2 \rangle)] ==\> \Gamma |- s1 \approx_L s2)
from \Gamma,High |- if (b) c1 else c2 have \Gamma,High |- c1 and \Gamma,High |- c2
  by(auto elim:secComTyping.cases)
from (\langle if (b) c1 else c2,s1 \rangle -> (\langle Skip,s2 \rangle))
have \[b\] s1 = Some true \&\& \[b\] s1 = Some false by(auto dest:Cond-True-or-False)
thus ?case
proof
  assume \[b\] s1 = Some true
  with (\langle if (b) c1 else c2,s1 \rangle -> (\langle Skip,s2 \rangle) have \langle c1,s1 \rangle -> (\langle Skip,s2 \rangle)
    by(auto intro:CondTrue-ords)
  from IH1[OF \Gamma,High |- c1; this] show ?thesis .
next
  assume \[b\] s1 = Some false
  with (\langle if (b) c1 else c2,s1 \rangle -> (\langle Skip,s2 \rangle) have \langle c2,s1 \rangle -> (\langle Skip,s2 \rangle)
    by(auto intro:CondFalse-ords)
  from IH2[OF \Gamma,High |- c2; this] show ?thesis .
qed

next
case (While b c)
note IH = (\A s1 s2. [\Gamma,High |- c; \langle c',s1 \rangle -> (\langle Skip,s2 \rangle)] ==\> \Gamma |- s1 \approx_L s2)
from \Gamma,High |- while (b) c have \Gamma,High |- c by(auto elim:secComTyping.cases)
from IH[OF this]
have \langle s1 s2. \[(c',s1) -> (\langle Skip,s2 \rangle)] ==\> \Gamma |- s1 \approx_L s2 .
  with (\langle while (b) c',s1 \rangle -> (\langle Skip,s2 \rangle) \Gamma,High |- c')
  show ?case by(auto dest:WhileStepInduct)
qed

19
lemma CondHighCompositionality:
assumes \( \Gamma, \text{High} \vdash \text{if} (b) \; c_1 \text{ else } c_2 \)
shows \( \text{nonInterference} \; \Gamma \; (\text{if} (b) \; c_1 \text{ else } c_2) \)
proof (rule nonInterferenceI)
fix \( s_1 \; s_2 \; s_1' \; s_2' \)
assume \( \Gamma \vdash s_1 \approx_L s_2 \) and \( (\text{if} (b) \; c_1 \text{ else } c_2, s_1) \to^{*} (\text{Skip}, s_1') \)
and \( (\text{if} (b) \; c_1 \text{ else } c_2, s_2) \to^{*} (\text{Skip}, s_2') \)
show \( \Gamma \vdash s_1' \approx_L s_2' \)
proof 
from \( \Gamma, \text{High} \vdash (\text{if} (b) \; c_1 \text{ else } c_2); (\text{if} (b) \; c_1 \text{ else } c_2, s_1) \to^{*} (\text{Skip}, s_1') \)
have \( \Gamma \vdash s_1 \approx_L s_1' \) by (auto dest: highBodies)
from \( \Gamma, \text{High} \vdash (\text{if} (b) \; c_1 \text{ else } c_2); (\text{if} (b) \; c_1 \text{ else } c_2, s_2) \to^{*} (\text{Skip}, s_2') \)
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 \( \Gamma \vdash s_1 \approx_L s_2' \) 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: lowEquivTransitive)
qed
qed

lemma CondLowCompositionality:
assumes \( \text{nonInterference} \; \Gamma \; c_1 \) and \( \text{nonInterference} \; \Gamma \; c_2 \) and \( \Gamma \vdash b : \text{Low} \)
shows \( \text{nonInterference} \; \Gamma \; (\text{if} (b) \; c_1 \text{ else } c_2) \)
proof (rule nonInterferenceI)
fix \( s_1 \; s_2 \; s_1' \; s_2' \)
assume \( \Gamma \vdash s_1 \approx_L s_2 \) and \( (\text{if} (b) \; c_1 \text{ else } c_2, s_1) \to^{*} (\text{Skip}, s_1') \)
and \( (\text{if} (b) \; c_1 \text{ else } c_2, s_2) \to^{*} (\text{Skip}, s_2') \)
from \( \Gamma \vdash b : \text{Low} \) have \( [b] \; s_1 = [b] \; s_2 \)
by (auto intro: interpretLow2)
from \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_1') \)
have \( [b] \; s_1 = \text{Some true} \) by (auto dest: CondTrue-or-False)
thus \( \Gamma \vdash s_1' \approx_L s_2' \)
proof 
assume \( [b] \; s_1 = \text{Some true} \)
with \( [b] \; s_1 = [b] \; s_2 \) have \( [b] \; s_2 = \text{Some true} \)
by (auto intro: CondTrue-reds)
from \( [b] \; s_1 = \text{Some true} \) have \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_1') \)
by (auto intro: CondTrue-reds)
from \( [b] \; s_2 = \text{Some true} \) have \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_2') \)
by (auto intro: CondTrue-reds)
from \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_1') \) have \( \Gamma \vdash s_1 \approx_L s_2 \) (nonInterference \( \Gamma \; c_1 \))
show \(?thesis\) by (auto simp: nonInterference_def)
next 
assume \( [b] \; s_1 = \text{Some false} \)
with \( [b] \; s_1 = [b] \; s_2 \) have \( [b] \; s_2 = \text{Some false} \)
by (auto intro: CondTrue-reds)
from \( [b] \; s_1 = \text{Some false} \) have \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_1') \)
by (auto intro: CondFalse-reds)
from \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_1') \) have \( (\text{if} (b) \; c_1 \text{ else } c_2, s_1) \to^{*} (\text{Skip}, s_1') \)
by (auto intro: CondFalse-reds)
from \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_1') \) have \( ([b] \; s_1 = [b] \; s_2) \to^{*} (\text{Skip}, s_2') \)
have \((c_2, s_2) \rightarrow^* \langle \text{Skip}, s_2' \rangle\) by \((\text{auto intro:CondFalse-reds})\)
with \((c_2, s_1) \rightarrow^* \langle \text{Skip}, s_1' \rangle; \Gamma \vdash s_1 \approx_L s_2; \langle \text{nonInterference} \Gamma \ c_2 \rangle\)
show ?thesis by \((\text{auto simp:nonInterference-def})\)
qed

lemma WhileHighCompositionality:
assumes \(\Gamma, \text{High} \vdash \text{while} \ (b) \ c'\)
shows \(\langle \text{nonInterference} \ \Gamma \ \langle \text{while} \ (b) \ c' \rangle\rangle\)
proof \((\text{rule nonInterferenceI})\)
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 \((\text{auto dest:highBodies})\)
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 \((\text{auto dest:highBodies})\)
with \(\Gamma \vdash s_1 \approx_L s_2\) have \(\Gamma \vdash s_1 \approx_L s_2\) by \((\text{auto intro:lowEquivTransitive})\)
from \(\Gamma \vdash s_1 \approx_L s_1'\) have \(\Gamma \vdash s_1' \approx_L s_1\) by \((\text{auto intro:lowEquivSymmetric})\)
with \(\Gamma \vdash s_1 \approx_L s_2\) show ?thesis by \((\text{auto intro:lowEquivTransitive})\)
qed

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

WhileLowCompositionality

lemma WhileLowCompositionality:

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

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

proof

rule nonInterferenceI

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

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

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

\{ 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 \( \bigwedge s_1 \ s_2 \ s_2''. \ [(\langle c',s_1 \rangle) \rightarrow\star (\text{Skip},s_1'); (\langle c',s_2 \rangle) \rightarrow\star (\text{Skip},s_2'); \)

\( \Gamma \vdash s_1 \approx_L s_2 \] \( \Rightarrow \) \( \Gamma \vdash s_1'' \approx_L s_2'' \) by auto

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

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

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

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

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

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

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 \( \text{varprem: } (\Gamma \ V = \text{Some High}) \lor (\Gamma \ V = \text{Some Low} \land \Gamma \vdash e : \text{Low} \land T = \text{Low}) \)

by (auto elim:secComTyping.cases)

22
show \( ?\text{case} \)

\[
\text{proof}\quad (\text{rule nonInterferenceI})
\]

\[
\text{fix } s_1\text{ } s_2\text{ } s_1'\text{ } s_2'\text{ }
\]

\[
\text{assume } \Gamma \vdash s_1 \approx_L s_2 \text{ and } \langle V := e, s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle \text{ and } \langle V := e, s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle
\]

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 varprem show \( \Gamma \vdash s_1' \approx_L s_2' \)

\[
\text{proof}
\]

\[
\text{assume } \Gamma \text{ } V = \text{ Some High}
\]

with \( \Gamma \vdash s_1 \approx_L s_2 \) \( \langle s_1' = s_1(V := [e] s_1) \rangle \) \( \langle s_2' = s_2(V := [e] s_2) \rangle \)

\[
\text{show } ?\text{thesis by(auto intro:assignNIhighlemma)}
\]

next

\[
\text{assume } \Gamma \text{ } V = \text{ Some Low } \land \Gamma \vdash e : \text{ Low } \land T = \text{ Low}
\]

with \( \Gamma \vdash s_1 \approx_L s_2 \) \( \langle s_1' = s_1(V := [e] s_1) \rangle \) \( \langle s_2' = s_2(V := [e] s_2) \rangle \)

\[
\text{show } ?\text{thesis by(auto intro:assignNILowlemma)}
\]

qed

next

case \( \langle \text{Seq } c_1 \text{ } c_2 \rangle \)

\[
\text{note IH1 } = \langle \land T, \Gamma, T \vdash c_1 \implies \text{nonInterference } \Gamma \text{ } c_1 \rangle
\]

\[
\text{note IH2 } = \langle \land T, \Gamma, T \vdash c_2 \implies \text{nonInterference } \Gamma \text{ } c_2 \rangle
\]

\[
\text{show } ?\text{case}
\]

\[
\text{proof}\quad (\text{cases } T)
\]

\[
\text{case } \text{High}
\]

\[
\text{with } \Gamma, T \vdash c_1 ;; c_2\text{ have } \Gamma, \text{High } \vdash c_1 \text{ and } \Gamma, \text{High } \vdash c_2
\]

\[
\text{by(auto elim:secComTyping.cases)}
\]

from \( \text{IH1}[\text{OF } \Gamma, \text{High } \vdash c_1] \) have nonInterference \( \Gamma \text{ } c_1 \).

moreover

from \( \text{IH2}[\text{OF } \Gamma, \text{High } \vdash c_2] \) have nonInterference \( \Gamma \text{ } c_2 \).

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

next

case \( \text{Low} \)

\[
\text{with } \Gamma, T \vdash c_1 ;; c_2\text{ have } \langle \Gamma, \text{Low } \vdash c_1 \land \Gamma, \text{Low } \vdash c_2 \rangle \lor \Gamma, \text{High } \vdash c_1 ;; c_2
\]

\[
\text{by(auto elim:secComTyping.cases)}
\]

thus \( ?\text{thesis} \)

\[
\text{proof}
\]

\[
\text{assume } \Gamma, \text{Low } \vdash c_1 \land \Gamma, \text{Low } \vdash c_2
\]

\[
\text{hence } \Gamma, \text{Low } \vdash c_1 \text{ and } \Gamma, \text{Low } \vdash c_2 \text{ by simp-all}
\]

from \( \text{IH1}[\text{OF } \Gamma, \text{Low } \vdash c_1] \) have nonInterference \( \Gamma \text{ } c_1 \).

moreover

from \( \text{IH2}[\text{OF } \Gamma, \text{Low } \vdash c_2] \) have nonInterference \( \Gamma \text{ } c_2 \).

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

next

\[
\text{assume } \Gamma, \text{High } \vdash c_1 ;; c_2
\]

\[
\text{hence } \Gamma, \text{High } \vdash c_1 \text{ and } \Gamma, \text{High } \vdash c_2 \text{ by(auto elim:secComTyping.cases)}
\]
from IH1[OF \( \Gamma,\text{High} \vdash c1 \)] have nonInterference \( \Gamma \ c1 \).
moreover
from IH2[OF \( \Gamma,\text{High} \vdash c2 \)] have nonInterference \( \Gamma \ c2 \).
ultimately show \( ?\text{thesis} \) by(auto intro:SeqCompositionality)
qed

next
case (Cond \( b \ c1 \ c2 \))

next
case (While \( b \ c' \))

next
hence $\Gamma \vdash b : \text{Low}$ and $\Gamma, \text{Low} \vdash c'$ by simp-all

moreover

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

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

next

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

thus $\text{thesis}$ by(auto intro:WhileHighCompositionality)

qed

qed

end

theory Execute

imports secTypes

begin

3 Executing the small step semantics

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

thm red.equation

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

lemmas [code-pred-intro] = typeVal[where $\text{lev} = \text{Low}$] typeVal[where $\text{lev} = \text{High}$]

typeVar

typeBinOp1 typeBinOp2[where $\text{lev} = \text{Low}$] typeBinOp2[where $\text{lev} = \text{High}$] type-BinOp3[where $\text{lev} = \text{Low}$]

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

proof –

case secExprTyping

from secExprTyping.prems show thesis

proof

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

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

next

fix $\Gamma \ V n \ \text{lev}$

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

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

next

fix $\Gamma \ c1 \ c2 \ \text{bop}$

assume $x = \Gamma \ xa = c1 <\text{bop}> c2 \ xb = \text{Low}$

$\Gamma \vdash c1 : \text{Low} \ \Gamma \vdash c2 : \text{Low}$

from secExprTyping(4) this show thesis by auto

next

25
\[
\begin{align*}
\text{fix } \Gamma &\ e1 \ e2 \ \text{lev} \ \text{bop} \\
\text{assume } x &\ = \Gamma \ xa = e1 \ l bop \ e2 \ xb = \text{High} \\
\Gamma &\vdash e1 : \text{High} \ \Gamma \vdash e2 : \text{lev} \\
\text{from secExprTyping}(5-6) \text{ this show thesis by (cases lev) (auto)} \\
\text{next} \\
\text{fix } \Gamma &\ e1 \ e2 \ \text{lev} \ \text{bop} \\
\text{assume } x &\ = \Gamma \ xa = e1 \ l bop \ e2 \ xb = \text{High} \\
\Gamma &\vdash e1 : \text{lev} \ \Gamma \vdash e2 : \text{High} \\
\text{from secExprTyping}(6-7) \text{ this show thesis by (cases lev) (auto)} \\
\text{qed} \\
\text{qed}
\end{align*}
\]

\textbf{lemmas} [code-pred-intro] = typeSkip[where \(T=\text{Low}\)] typeSkip[where \(T=\text{High}\)]

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

\textbf{proof} –

\textbf{case secComTyping}

\textbf{from secComTyping.prems show thesis}

\textbf{proof}

\textbf{fix } \Gamma &\ T \ \text{assume } x = \Gamma \ xa = T \ xb = \text{Skip} \\
\textbf{from secComTyping}(1-2) \text{ this show thesis by (cases } T \text{) (auto)} \\
\textbf{next} \\
\textbf{fix } \Gamma &\ V \ T \ e \ \text{assume } x = \Gamma \ xa = T \ xb = V:=e \ \Gamma \ V = \text{Some High} \\
\textbf{from secComTyping}(3-4) \text{ this show thesis by (cases } 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 secComTyping}(5) \text{ this show thesis by auto} \\
\textbf{next} \\
\textbf{fix } \Gamma &\ T \ c1 \ c2 \\
\textbf{assume } x = \Gamma \ xa = T \ xb = \text{Seq} c1 \ c2 \ \Gamma , T \vdash c1 \ \Gamma , T \vdash c2 \\
\textbf{from secComTyping}(6) \text{ this show thesis by 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 secComTyping}(7) \text{ this show thesis by 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 secComTyping}(8) \text{ this show thesis by blast} \\
\textbf{next} \\
\textbf{fix } \Gamma &\ c \\
\textbf{assume } x = \Gamma \ xa = \text{Low} \ xb = c \ \Gamma , \text{High} \vdash c \\
\textbf{from secComTyping}(9) \text{ 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}
definition Env' = map-of [("x", Low), ("y", High)]
value Env', Low ⊢ com
value Env', High ⊢ com
values 1 { T. Env, T ⊢ com}
end

References
