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

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

February 23, 2021

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 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
\[ FVc : \text{fv} (\text{Val } V) = \{ \} \]
\[ | FVc : \text{fv} (\text{Var } V) = \{ V \} \]
\[ | FVc : \text{fv} (e1 \left< \text{bop}\right> e2) = \text{fv } e1 \cup \text{fv } e2 \]

1.3 State

**type-synonym** state = ename \rightarrow val

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

**fun** interpret :: expr \Rightarrow state \Rightarrow val option

**where**

Val: [Val v] s = Some v

Var: [Var V] s = s V

BinOp: [e1 \left< \text{bop}\right> e2] s = (case [e1] s of None \Rightarrow None

| Some v1 \Rightarrow (case [e2] s of None \Rightarrow None

| Some v2 \Rightarrow \text{binop bop } v1 v2))

1.4 Small Step Semantics

**inductive** red :: com \* state \Rightarrow com \* state \Rightarrow bool

**and** red' :: com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool

(((1,{-,-}) \rightarrow/ (1,{-,-})) [0,0,0,0] 81)

**where**

\( \langle c1,s \rangle \rightarrow \langle c2,s' \rangle \Rightarrow \text{red } (c1,s1) (c2,s2) \mid \text{RedAss} \)

\( \langle V := e,s \rangle \rightarrow \langle \text{Skip},s(V := ([e] s)) \rangle \)

| SeqRed:

\( \langle c1,s \rangle \rightarrow \langle c1',s' \rangle \Rightarrow \langle c1';c2,s' \rangle \)

| RedSeq:

\( \langle \text{Skip};c2,s \rangle \rightarrow \langle c2,s \rangle \)

| RedCondTrue:

\( \langle b \rangle s = \text{Some true} \Rightarrow (\text{if } b \text{ else } c2,s) \rightarrow \langle c1,s \rangle \)

| RedCondFalse:

\( \langle b \rangle s = \text{Some false} \Rightarrow (\text{if } b \text{ else } c2,s) \rightarrow \langle c2,s \rangle \)

| RedWhileTrue:

\( \langle b \rangle s = \text{Some true} \Rightarrow (\text{while } b \text{ c},s) \rightarrow \langle c;\text{while } b \text{ c},s \rangle \)

| RedWhileFalse:

\( \langle b \rangle s = \text{Some false} \Rightarrow (\text{while } b \text{ c},s) \rightarrow \langle \text{Skip},s \rangle \)

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

**abbreviation** reds :: com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool

(((1,{-,-}) \rightarrow*/ (1,{-,-})) [0,0,0,0] 81) **where**

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

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

lemma Seq-reds:
assumes \((c_1; c_2, s) \rightarrow^{*} (\text{Skip}, s')\)
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 ?case by simp
    next
    case (step \(c\) \(s\) \(c''\) \(s''\))
    note IH = \((c'' = c_1; c_2; c' = \text{Skip}) \implies \exists s. (c_1, s') \rightarrow^{*} (\text{Skip}, s) \land (c_2, s) \rightarrow^{*} (\text{Skip}, s')\)
    from 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 ?case
    proof
      assume \(c_1 = \text{Skip} \land c'' = c_2 \land s = s''\)
      with \((c'', s'') \rightarrow^{*} (c', s')\) \(c' = \text{Skip}\)
      show \(\text{?thesis}\) by auto
    next
      assume \(\exists c_1'. (c_1, s) \rightarrow (c_1', s'') \land c'' = c_1'; c_2\)
      then obtain \(c_1'\) where \((c_1, s) \rightarrow (c_1', s'')\) and \(c'' = c_1'; c_2\) by blast
      from IH[OF \((c' = c_1'; c_2; c' = \text{Skip})\]
      obtain \(s\) where \((c_1', s'') \rightarrow^{*} (\text{Skip}, s)\) and \((c_2, s) \rightarrow (\text{Skip}, s)\)
      by blast
      from \((c_1, s) \rightarrow (c_1', s'')\) \((c_1', s'') \rightarrow^{*} (\text{Skip}, s)\)
have \((c_1, s) \rightarrow (\text{Skip}, s')\) by (auto intro: converse-rtranclp-into-rtranclp)
with \((c_2, s) \rightarrow (\text{Skip}, s')\) show \(?thesis\) by auto
qed

\section*{Proof}
\begin{proof}
\item[Lemma] \textbf{Cond-True-or-False:}
\begin{align*}
\langle \text{if } b \rangle \ c_1 \ \text{else} \ c_2, \ s \rangle \rightarrow (\text{Skip}, s') \Longrightarrow \ [b] \ s = \text{Some true} \lor [b] \ s = \text{Some false}
\end{align*}
by (induct \(c\) := if \(b\) \(c_1\) \(\text{else} \ c_2\) \(s\) \(rulenew: \text{converse-rtranclp-}\text{-induct2})\)(auto elim: red\_cases)

\item[Lemma] \textbf{CondFalse-reds:}
\begin{align*}
\langle \text{if } b \rangle \ c_1 \ \text{else} \ c_2, \ s \rangle \rightarrow (\text{Skip}, s') \Longrightarrow \ [b] \ s = \text{Some false} \Longrightarrow \ (c_1, s) \rightarrow (\text{Skip}, s')
\end{align*}
by (induct \(c\) := if \(b\) \(c_1\) \(\text{else} \ c_2\) \(s\) \(rulenew: \text{converse-rtranclp-}\text{-induct2})\)(auto elim: red\_cases)

\item[Lemma] \textbf{WhileFalse-reds:}
\begin{align*}
\langle \text{while } (b) \ cx, s \rangle \rightarrow (\text{Skip}, s') \Longrightarrow \ [b] \ s = \text{Some false} \Longrightarrow \ s = s'
\end{align*}
proof (induct \text{while } (b) \ cx \ s \ rulenew: converse-rtranclp-\text{-induct2})
\begin{itemize}
\item \textbf{case step thus \(?case\) by (auto elim: red\_cases \text{ dest: Skip-reds})}
\end{itemize}
qed

\item[Lemma] \textbf{WhileTrue-reds:}
\begin{align*}
\langle \text{while } (b) \ cx, s \rangle \rightarrow (\text{Skip}, s') \Longrightarrow \ [b] \ s = \text{Some true} \Longrightarrow \exists \ sx. \ \langle \text{cx, sx} \rangle \rightarrow (\text{Skip}, s') \land \langle \text{while } (b) \ cx, sx \rangle \rightarrow (\text{Skip}, s')
\end{align*}
proof (induct \text{while } (b) \ cx \ s \ rulenew: converse-rtranclp-\text{-induct2})
\begin{itemize}
\item \textbf{case step \(s\) \(c'' \ s''\)}
\item \textbf{hence \(c'' = cx;\text{while } (b) \ cx \land s'' = s\) by (auto elim: red\_cases)}
\item \textbf{with \(?case\) by (auto dest: Seq-reds)}
\end{itemize}
qed

\item[Lemma] \textbf{WhileTrue-\text{or-False}:}
\begin{align*}
\langle \text{while } (b) \ com, s \rangle \rightarrow (\text{Skip}, s') \Longrightarrow \ [b] \ s = \text{Some true} \lor [b] \ s = \text{Some false}
\end{align*}
by (induct \text{while } (b) \ com \ s \ rulenew: converse-rtranclp-\text{-induct2})\)(auto elim: red\_cases)

\item[Inductive] \textbf{red-n}: \(\text{com} \Rightarrow \text{state} \Rightarrow \text{nat} \Rightarrow \text{com} \Rightarrow \text{state} \Rightarrow \text{bool}\)
\begin{itemize}
\item[Base] \textbf{red-n-Base}: \(\langle c, s \rangle \rightarrow^0 (c, s)\)
\item[Rec] \textbf{red-n-Rec}: \[\langle c, s \rangle \rightarrow (c'', s''); \ (c'', s'') \rightarrow^n (c', s')\] \(\Longrightarrow \langle c, s \rangle \rightarrow^S (c', s)\)
\end{itemize}

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

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

proof

```latex
from \((c_1;c_2,s) \rightarrow^n (\text{Skip}, s')\)
```

have \( \exists i \) \( j \) \( s'' \). \((c_1, s) \rightarrow^i (\text{Skip}, s') \) \& \((c_2, s''') \rightarrow^j (\text{Skip}, s) \) \& \( n = i + j + 1 \)

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

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

```latex
\text{note } IH = \langle c_1, c'' = c_1;c_2 \rangle \rightarrow \exists i \) \( j \) \( s' \times s'' \rightarrow (\text{Skip}, s) \) \& \((c_2, s') \rightarrow^j (\text{Skip}, s) \) \& \( n = i + j + 1 \)
```

```latex
from \((c_1;c_2,s) \rightarrow (c'',s'')\)
```

```latex
\text{have } (c_1 = \text{Skip} \& c'' = c_2 \& s = s'') \lor \\
(\exists c_1'. c'' = c_1;c_2 \& (c_1, s) \rightarrow (c_1',s''))
```

```latex
\text{by(} \text{induct } c_1;c_2 - - \text{ rule: } \text{red-induct} \text{) auto}
```

thus ?case

```latex
proof
```

```latex
\text{assume } c_1 = \text{Skip} \& c'' = c_2 \& s = s''
```

```latex
\text{hence } c_1 = \text{Skip} \& c'' = c_2 \& s = s'' \text{ by } \text{simp-all}
```

```latex
\text{from } (c_1 = \text{Skip}) \text{ have } (c_1, s) \rightarrow^0 (\text{Skip}, s) \text{ by(} \text{fastforce intro: } \text{red-n-Base} \text{) }
```

```latex
\text{with } (c'',s'') \rightarrow^n (\text{Skip}, s') \text{ } (c'' = c_2) \& (s = s'')
```

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

next

```latex
\text{assume } \exists c_1'. c'' = c_1';c_2 \& (c_1, s) \rightarrow (c_1',s'')
```

```latex
\text{then obtain } c_1' \text{ where } c'' = c_1';c_2 \& (c_1, s) \rightarrow (c_1',s'') \text{ by blast}
```

```latex
\text{from } IH[\text{OF } (c'' = c_1';c_2)] \text{ obtain } i \) j \( s' \times s'' \times s'' \times s'' \rightarrow (\text{Skip}, s)
```

```latex
\text{and } n = i + j + 1 \text{ by blast}
```

```latex
\text{from } (c_1, s) \rightarrow (c_1',s'') \text{ } (c_1',s'') \rightarrow^i (\text{Skip}, s)
```

```latex
\text{have } (c_1, s) \rightarrow \text{Suc } i \text{ } (\text{Skip}, s) \text{ by(} \text{rule red-n-red-n-Rec} \text{) }
```

```latex
\text{with } (c_2, s) \rightarrow^j (\text{Skip}, s') \text{ } (n = i + j + 1) \text{ show } ?\text{thesis}
```

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

```latex
\text{qed}
```

```latex
\text{with that show } ?\text{thesis by blast}
```

```latex
\text{qed}
```

**lemma** \text{while-red-nE}:

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

```latex
\Rightarrow (\exists i \) j \( s''). (\text{[b]} s = \text{Some false} \& s = s' \& n = 1) \lor \\
(\exists i \) j \( s''). (\text{[b]} s = \text{Some true} \& (cx, s) \rightarrow^i (\text{Skip}, s') \& \\
(\text{while } (b) \text{ } cx, s'') \rightarrow^j (\text{Skip}, s) \& n = i + j + 2)
```

```latex
proof\( (\text{induct } \text{while } (b) \text{ } cx, s \text{ n } \text{Skip } s' \text{ rule: } \text{red-n-induct} \) \)
```

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

```latex
\text{from } (\text{while } (b) \text{ } cx, s) \rightarrow (c'',s'')\)
```

```latex
\text{have } (\text{[b]} s = \text{Some false} \& c'' = \text{Skip} \& s'' = s) \lor \\
(\text{[b]} s = \text{Some true} \& c'' = cx;;\text{while } (b) \text{ } cx \& s'' = s)
```

```latex
\text{by(} \text{induct } \text{while } (b) \text{ } cx - - \text{ rule: } \text{red-induct} \text{) auto}
```

thus ?case

```latex
proof
```
lemma while-red-n-induct [consumes 1, case-names false true]:
assumes major: ⟨while (b) cx,s⟩ →^n ⟨Skip,s'⟩
and IHfalse:\s. \[b\] s = Some false --- P s s
and IHtrue:\s i j s'. \[b\] s = Some true; ⟨cx,s⟩ →^i ⟨Skip,s'⟩;
⟨while (b) cx,s⟩ →^j ⟨Skip,s'⟩; P s s' \[s\] --- P s s'
shows P s s'
using major
proof (induct n arbitrary; s rule:nat-less-induct)
fix n s
assume IHall:\forall m<n. \forall x. ⟨while (b) cx,x⟩ →^m ⟨Skip,s'⟩ --- P x s'
and ⟨while (b) cx,s⟩ →^n ⟨Skip,s'⟩
from ⟨while (b) cx,s⟩ →^n ⟨Skip,s'⟩
have \[b\] s = Some false \land s = s' \land n = 1 \lor
(∃ i j s''. \[b\] s = Some true \land ⟨cx,s⟩ →^i ⟨Skip,s''⟩ \land
⟨while (b) cx,s''⟩ →^j ⟨Skip,s'⟩ \land n = i + j + 2)
by (rule while-red-nE)
thus P s s'
proof
assume \[b\] s = Some false \land s = s' \land n = 1
hence \[b\] s = Some false and s = s' by auto
from IHfalse[OF \[b\] s = Some false] have P s s .
with \(s = s'\) show \(\text{thesis by simp}\)
next
assume \(\exists i j s''. \[b\] s = Some true \land ⟨cx,s⟩ →^i ⟨Skip,s''⟩ \land
⟨while (b) cx,s''⟩ →^j ⟨Skip,s'⟩ \land n = i + j + 2\)
then obtain i j s'' where \[b\] s = Some true
and ⟨cx,s⟩ →^i ⟨Skip,s''⟩ and ⟨while (b) cx,s''⟩ →^j ⟨Skip,s'⟩
and n = i + j + 2 by blast
with IHall have P s'' s'
apply (erule_tac x=j in allE) apply clarsimp done
from IHtrue[OF \[b\] s = Some true] \(⟨cx,s⟩ →^1 ⟨Skip,s'⟩\)
\langle \text{while} \ (b) \ cx,s' \rangle \rightarrow^i \langle \text{Skip},s' \rangle \ \text{this by lemma reds-to-reds}; \ \text{this by lemma red-n-to-reds}; \ \text{this by lemma converse-rtranclp-induct2, auto intro:red-n.intros)}

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

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

\text{lemma 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} \implies P \ s \ s; 
\langle \text{while} \ (b) \ 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' \implies P \ s \ s' 
\text{apply(drule reds-to-red-n,clarsimp)} 
\text{apply(erule while-reds-induct,clarsimp)} 
\text{by(auto dest:red-n-to-reds)}

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

\text{theorem reds-det:}
\langle (c,s) \rightarrow^* \langle \text{Skip},s_1 \rangle; \ (c,s) \rightarrow^* \langle \text{Skip},s_2 \rangle \rangle \implies s_1 = s_2 
\text{proof(induct rule:converse-rtranclp-induct2)} 
\text{case refl}
from \((\langle \text{Skip}, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle)\) show ?case

by

\(-\) (erule converse-rtranclpE, auto elim:red_cases)

next

case (step \(c''\) \(s''\) \(c'\) \(s'\))

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

from step have \(\langle \langle c'', s'' \rangle \rangle \rightarrow \langle c', s' \rangle\)

by simp

from \(\langle \langle c'', s'' \rangle \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle\) this have \(\langle \langle c', s' \rangle \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle\)

by

\(-\) (erule converse-rtranclpE, auto elim:red_cases dest:red-det)

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

qed

datatype secLevel = Low | High

type-synonym typeEnv = vname \rightarrow secLevel

inductive secExprTyping :: typeEnv \Rightarrow expr \Rightarrow secLevel \Rightarrow bool (- \vdash - : -)

where
typeVal: \(\Gamma \vdash \text{Val} V : \text{lev}\)

typeVar: \(\Gamma Vn = \text{Some lev} \implies \Gamma \vdash \text{Var} Vn : \text{lev}\)

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

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

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

inductive secComTyping :: typeEnv \Rightarrow secLevel \Rightarrow com \Rightarrow bool (-,- \vdash -)

where
typeSkip: \(\Gamma,T \vdash \text{Skip}\)

typeAssH: \(\Gamma V = \text{Some High} \implies \Gamma,T \vdash V := e\)

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

| typeSeq: [\[ \Gamma,T \vdash c_1; \Gamma,T \vdash c_2\]] \implies \Gamma,T \vdash c_1;c_2

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

2 Security types

2.1 Security definitions

datatype secLevel = Low | High

type-synonym typeEnv = vname -> secLevel

inductive secExprTyping :: typeEnv => expr => secLevel => bool (- |- :: -)

where
typeVal: Gamma |- Val V : lev

typeVar: Gamma Vn = Some lev => Gamma |- Var Vn : lev

typeBinOp1: [([Gamma |- e_1 : Low; Gamma |- e_2 : Low])] => Gamma |- e_1 bop e_2 : Low

| typeBinOp2: [([Gamma |- e_1 : High; Gamma |- e_2 : lev])] => Gamma |- e_1 bop e_2 : High

| typeBinOp3: [([Gamma |- e_1 : lev; Gamma |- e_2 : High])] => Gamma |- e_1 bop e_2 : High

inductive secComTyping :: typeEnv => secLevel => com => bool (-,- |- :: -)

where
typeSkip: Gamma,T |- Skip

typeAssH: Gamma V = Some High => Gamma,T |- V := e

| typeAssL: ([Gamma |- e : Low; Gamma V = Some Low]) => Gamma,Low |- V := e

| typeSeq: ([Gamma,T |- c_1; Gamma,T |- c_2]) => Gamma,T |- c_1;c_2

| typeWhile: ([Gamma |- b : T; Gamma,T |- c]) => Gamma,T |- while (b) c
| typeIf: \[ \Gamma \vdash b : T; \Gamma, T \vdash c_1; \Gamma, T \vdash c_2 \] \implies \Gamma, T \vdash if (b) c_1 else c_2

| typeConvert: \( \Gamma, \text{High} \vdash e \) \implies \( \Gamma, \text{Low} \vdash e \)

2.2 Lemmas concerning expressions

lemma exprTypeable:
\begin{align*}
\text{assumes } & fv\ e \subseteq \text{dom } \Gamma \\
\text{obtains } & T \ 	ext{where } \Gamma \vdash e : T \\
\text{proof} & \quad \text{(induct e)} \\
\text{case } & (\text{Val}\ V) \\
\text{have } & \Gamma \vdash \text{Val}\ V : \text{Low} \ 	ext{by (rule typeVal)} \\
\text{thus } & \text{case by (rule exI)} \\
\text{next} & \\
\text{case } & (\text{Var}\ V) \\
\text{have } & V \in fv\ (\text{Var}\ V) \ 	ext{by simp} \\
\text{with } & (fv\ (\text{Var}\ V) \subseteq \text{dom } \Gamma) \ 	ext{have } V \in \text{dom } \Gamma \ 	ext{by simp} \\
\text{then obtain } & T \ 	ext{where } \Gamma \vdash V = \text{Some}\ T \ 	ext{by auto} \\
\text{hence } & \Gamma \vdash \text{Var}\ V : T \ 	ext{by (rule typeVar)} \\
\text{thus } & \text{case by (rule exI)} \\
\text{next} & \\
\text{case } & (\text{BinOp}\ e_1\ bop\ e_2) \\
\text{note } & IH1 = (fv\ e_1 \subseteq \text{dom } \Gamma) \implies \exists T. \Gamma \vdash e_1 : T \\
\text{note } & IH2 = (fv\ e_2 \subseteq \text{dom } \Gamma) \implies \exists T. \Gamma \vdash e_2 : T \\
\text{from } & (fv\ e_1 \ « bop »\ e_2) \subseteq \text{dom } \Gamma \\
\text{have } & fv\ e_1 \subseteq \text{dom } \Gamma \text{ and } fv\ e_2 \subseteq \text{dom } \Gamma \ 	ext{by auto} \\
\text{from } & IH1[\text{OF } (fv\ e_1 \subseteq \text{dom } \Gamma)] \ 	ext{obtain } T_1 \ 	ext{where } \Gamma \vdash e_1 : T_1 \ 	ext{by auto} \\
\text{from } & IH2[\text{OF } (fv\ e_2 \subseteq \text{dom } \Gamma)] \ 	ext{obtain } T_2 \ 	ext{where } \Gamma \vdash e_2 : T_2 \ 	ext{by auto} \\
\text{show } & \text{case} \\
\text{proof } & \text{(cases T1)} \\
\text{case } & \text{Low} \\
\text{show } & \text{?thesis} \\
\text{proof } & \text{(cases T2)} \\
\text{case } & \text{Low} \\
\text{with } & \Gamma \vdash e_1 : T_1; \Gamma \vdash e_2 : T_2; (T_1 = \text{Low}) \\
\text{have } & \Gamma \vdash e_1 \ « bop »\ e_2 : \text{Low} \ 	ext{by (simp add: typeBinOp1)} \\
\text{thus } & \text{?thesis by (rule exI)} \\
\text{next} & \\
\text{case } & \text{High} \\
\text{with } & \Gamma \vdash e_1 : T_1; \Gamma \vdash e_2 : T_2; (T_1 = \text{Low}) \\
\text{have } & \Gamma \vdash e_1 \ « bop »\ e_2 : \text{High} \ 	ext{by (simp add: typeBinOp3)} \\
\text{thus } & \text{?thesis by (rule exI)} \\
\text{qed} \\
\text{next} & \\
\text{case } & \text{High} \\
\text{with } & \Gamma \vdash e_1 : T_1; \Gamma \vdash e_2 : T_2; \\
\text{have } & \Gamma \vdash e_1 \ « bop »\ e_2 : \text{High} \ 	ext{by (simp add: typeBinOp2)}
thus thesis by (rule exI)
qed

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

lemma exprTypingHigh:
assumes "Γ ⊢ e : T and x ∈ fv e and Γ x = Some High"
shows "Γ ⊢ e : High"
using assms
proof (induct e arbitrary: T)
case (Val V) show ?case by (rule typeVal)
next
case (Var V)
from "x ∈ fv (Var V)" have "x = V" by simp
with "Γ x = Some High" show ?case by(simp add:typeVar)
next
case (BinOp e1 bop e2)
note "IH1 = (∀ T. [Γ ⊢ e1 : T; x ∈ fv e1; Γ x = Some High] ⇒ Γ ⊢ e1 : High)"
note "IH2 = (∀ T. [Γ ⊢ e2 : T; x ∈ fv e2; Γ x = Some High] ⇒ Γ ⊢ e2 : High)"
from "Γ ⊢ e1 "bop" e2 : T" have "T:(∃ T1. Γ ⊢ e1 : T1) ∧ (∃ T2. Γ ⊢ e2 : T2)" by (auto intro!:exprBinopTypeable)
then obtain T1 where "Γ ⊢ e1 : T1" by auto
from T obtain T2 where "Γ ⊢ e2 : T2" by auto
from "x ∈ fv (e1 "bop" e2)" have "x ∈ (fv e1 ∪ fv e2)" by simp
hence "x ∈ fv e1 ∨ x ∈ fv e2" by auto
thus ?case
proof
assume "x ∈ fv e1"
from IH1[OF "Γ ⊢ e1 : T1" this "Γ x = Some High"] have "Γ ⊢ e1 : High" .
with "Γ ⊢ 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 \( \text{fv} (\text{Val} V) = \{\} \) by (rule FVc)
  with \( x \in \text{fv} (\text{Val} V) \) have False by auto  
  thus ?thesis by simp

next
  case (Var V)
  from \( x \in \text{fv} (\text{Var} V) \) have \( x = V \) by simp
  from \( \Gamma \vdash \text{Var} V : \text{Low} \) have \( \Gamma = \text{Some} \text{Low} \) by (auto elim: secExprTyping.cases)
  with \( x = V \) show ?thesis by simp

next
  case (BinOp e1 bop e2)
  note IH1 = \( \forall T. \Gamma \vdash e1 : T \Rightarrow \text{fv} e1 \subseteq \text{dom} \Gamma \)
  note IH2 = \( \forall T. \Gamma \vdash e2 : T \Rightarrow \text{fv} e2 \subseteq \text{dom} \Gamma \)
  from \( \Gamma \vdash e1 \langle bop \rangle e2 : \text{Low} \) have \( \Gamma \vdash e1 : \text{Low} \) and \( \Gamma \vdash e2 : \text{Low} \)
    by (auto elim: secExprTyping.cases)
  from \( x \in \text{fv} (e1 \langle bop \rangle e2) \) have \( x \in \text{fv} e1 \cup \text{fv} e2 \) by (simp add:FVe)
  hence \( x \in \text{fv} e1 \lor x \in \text{fv} e2 \) by auto
  thus \( x \in \text{fv} e1 \cup \text{fv} e2 \) 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 \( \text{fv} (\text{Val} V) = \{\} \) 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 \( \Gamma = \text{Some} T \) 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 = \( \forall T. \Gamma \vdash e1 : T \Rightarrow \text{fv} e1 \subseteq \text{dom} \Gamma \)
note \( IH2 \) = \( \forall T. \Gamma \vdash e2 : T \implies fv e2 \subseteq \text{dom } \Gamma \)

show \(?\)case

proof

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

from \( \Gamma \vdash e1 \langle \text{bop} \rangle e2 : T \)

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 \( IH1[\langle \text{OF} \rangle \Gamma \vdash e1 : T1] \) have \( fv e1 \subseteq \text{dom } \Gamma \).

moreover from \( IH2[\langle \text{OF} \rangle \Gamma \vdash e2 : T2] \) have \( fv e2 \subseteq \text{dom } \Gamma \).

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

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

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

qed

qed

lemma \( \text{exprNotNone} \):

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

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

using asms

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

  case (Val \( v \))

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

next

  case (Var \( V \))

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

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

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

  thus \(?\)case by auto

next

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

  note \( IH1 \) = \( \forall T. \Gamma \vdash e1 : T \; \forall fv e1 \subseteq \text{dom } s \implies [e1] s \neq \text{None} \)

  note \( IH2 \) = \( \forall T. \Gamma \vdash e2 : T \; \forall fv e2 \subseteq \text{dom } s \implies [e2] s \neq \text{None} \)

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

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

  from \( IH1[\langle \text{OF} \rangle \Gamma \vdash e1 : T1] \) \( \langle \text{fv } e1 \subseteq \text{dom } s \rangle \) have \([e1]s \neq \text{None} \).

  moreover from \( IH2[\langle \text{OF} \rangle \Gamma \vdash e2 : T2] \) \( \langle \text{fv } e2 \subseteq \text{dom } s \rangle \) have \([e2]s \neq \text{None} \).

  ultimately show \(?\)case

  apply (cases \( \text{bop} \)) apply auto

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

  done

qed
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
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 \( \Rightarrow \) com \( \Rightarrow \) bool
where nonInterference \( \Gamma c \equiv \forall s1 s2 s1' s2'. (\Gamma \vdash s1 \approx_L s2 \land \langle c,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle \land \langle c,s2 \rangle \rightarrow^* \langle \text{Skip},s2' \rangle) \rightarrow \Gamma \vdash s1' \approx_L s2' \)

lemma nonInterferenceI: \( \Gamma \vdash s1 \approx_L s2 ; \Gamma \vdash s2 \approx_L s3 \] \( \Rightarrow \) \( \Gamma \vdash s1 \approx_L s3 \)
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 \( \llbracket e \rrbracket s1 = \llbracket e \rrbracket s2 \)
using all
proof (induct e)
  case (Val v)
  show ?case by (simp add: Val)
next
  case (Var V)
  have \( \llbracket \text{Var } V \rrbracket s1 = s1 V \) and \( \llbracket \text{Var } V \rrbracket s2 = s2 V \) by (auto simp: Var)
  have \( V \in \text{fv } \langle \text{Var } V \rangle \) by (simp add: FVv)

15
from \( \forall V \in \text{fv} (\text{Var} \ V) \land X \in \text{fv} (\text{Var} \ V) \). \( \Gamma \ X = \text{Some Low} \) have \( \Gamma \ V = \text{Some Low} \) by simp

with 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} \Rightarrow \[e1]s1 = [e1]s2 \)

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

  from \( \forall V \in \text{fv} \ (e1 <bop> e2) \). \( \Gamma \ V = \text{Some Low} \) 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[OF \( \forall V \in \text{fv} \ e1 \). \( \Gamma \ V = \text{Some Low} \)] have \([e1]s1 = [e1]s2 \).

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

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

  qed

lemma interpretLow2:

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

proof –

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

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

  proof

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

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

  qed

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

  qed

lemma assignNHiglemma:

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

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

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

proof

  { fix \( V' \) assume \( V' \in \text{dom} \ \Gamma \) and \( \Gamma \ V' = \text{Some Low} \)
    from \( \Gamma \vdash s1 \approx_L s2 \). \( \Gamma \ V' = \text{Some Low} \) have \( s1 \ V' = s2 \ V' \)

      by(auto simp add:lowEquiv-def)

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

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

    case True

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

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

  next

    case False

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

    have \( s1 \ V' = s1' \ V' \) and \( s2 \ V' = s2' \ V' \) by auto

    with \( \langle s1 \ V' = s2 \ V' \rangle \) show \( \text{thesis} \) by simp

  qed
lemma assignNllowerlemma:
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’⟩ have s1′ V’ = s2′ V’ by simp
  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:Seqreds)
from ⟨c1;c2,s2⟩ →∗ ⟨Skip,s2′⟩ obtain s2’’ where ⟨c1,s2⟩ →∗ ⟨Skip,s2’’⟩
and ⟨c2,s2’’⟩ →∗ ⟨Skip,s2⟩ by(auto dest:Seqreds)
from Γ ⊢ s1 ≈_L s2: ⟨⟨c1,s1⟩ →∗ ⟨Skip,s1’’⟩; ⟨⟨c1,s2⟩ →∗ ⟨Skip,s2’’⟩

17
⟨\text{nonInterference} \, \Gamma \, c1\rangle

\text{have} \, \Gamma \vdash s1'' \approx_L s2'' \text{ by} (\text{auto simp:nonInterference-def})

\text{with} \, \langle\langle c2, s1' \rangle \rightarrow^* \langle \text{Skip}, s1' \rangle\rangle \langle\langle c2, s2' \rangle \rightarrow^* \langle \text{Skip}, s2' \rangle\rangle \langle\text{nonInterference} \, \Gamma \, c2\rangle

\text{show} \, \Gamma \vdash s1' \approx_L s2' \text{ by} (\text{auto simp:nonInterference-def})
qed

\text{lemma WhileStepInduct:}

\text{assumes} \, \text{while:} \langle\langle \text{while} \ (b) \ c, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle\rangle \text{ and body:} \langle\langle c, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle\rangle \Rightarrow \Gamma \vdash s1 \approx_L s2 \text{ and} \, \Gamma, \text{High} \vdash c

\text{shows} \, \Gamma \vdash s1 \approx_L s2
using \text{while}

\text{proof (induct rule:while-reds-induct)}

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

\text{next}

\text{case (true s1 s3)}

\text{from body:OF \langle\langle c, s1 \rangle \rightarrow^* \langle \text{Skip}, s3 \rangle\rangle \, \text{have} \, \Gamma \vdash s1 \approx_L s3 \text{ by simp}}

\text{with} \, \Gamma \vdash s3 \approx_L s2 \, \text{show \ ?case by (auto intro:lowEquivTransitive)}
qed

\text{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.}

\text{theorem highBodies:}

\text{assumes} \, \Gamma, \text{High} \vdash c \text{ and} \, \langle\langle c, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle\rangle \text{ shows} \, \Gamma \vdash s1 \approx_L s2
— all intermediate states must be well formed, otherwise the proof does not work for uninitialized variables. Thus it is propagated through the theorem conclusion

\text{using \text{assms}}

\text{proof (induct c arbitrary:s1 s2 rule:com.induct)}

\text{case Skip}

\text{from} \, \langle\langle \text{Skip}, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle\rangle \, \text{have} \, s1 = s2 \text{ by (auto dest:Skip-reds)}

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

\text{next}

\text{case (LAss V e)}

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

\text{from} \, \langle\langle V := e, s1 \rangle \rightarrow^* \langle \text{Skip}, s2 \rangle\rangle \, \text{have} \, s2 = s1(V := [e]s1) \text{ by (auto intro:LAss-reds)}

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

\text{have} \, s1 \, V' = s2 \, V'

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

\text{case True}

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

\text{thus \ ?thesis by simp}

\text{next}

\text{case False}

\text{with} \, \langle\langle s2 = s1(V := [e]s1) \rangle \rangle \, \text{show \ ?thesis by simp}
qed

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

next
case (Seq \( c1 \ c2 \))

note IH1 = \( \forall s1 \ s2. [\Gamma,\text{High} \vdash c1; \langle c1,s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle] \Rightarrow \Gamma \vdash s1 \approx_L s2 \)

note IH2 = \( \forall s1 \ s2. [\Gamma,\text{High} \vdash c2; \langle c2,s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle] \Rightarrow \Gamma \vdash s1 \approx_L s2 \)

from \( \Gamma,\text{High} \vdash c1;c2 \) have \( \Gamma,\text{High} \vdash c1 \) and \( \Gamma,\text{High} \vdash c2 \)

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

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

have \( \exists s3. \langle c1,s1 \rangle \rightarrow \langle \text{Skip},s3 \rangle \land \langle c2,s3 \rangle \rightarrow \langle \text{Skip},s2 \rangle \) by (\text{auto intro:Seq-reds})

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

from IH1[OF \( \Gamma,\text{High} \vdash c1; \langle c1,s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle \)]

have \( \Gamma \vdash s1 \approx_L s3 \) by simp

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

have \( \Gamma \vdash s3 \approx_L s2 \) by simp

from \( \Gamma \vdash s1 \approx_L s3; \Gamma \vdash s3 \approx_L s2 \) show ?case

by (\text{auto intro:lowEquivTransitive})

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

note IH1 = \( \forall s1 \ s2. [\Gamma,\text{High} \vdash c1; \langle c1,s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle] \Rightarrow \Gamma \vdash s1 \approx_L s2 \)

note IH2 = \( \forall s1 \ s2. [\Gamma,\text{High} \vdash c2; \langle c2,s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle] \Rightarrow \Gamma \vdash s1 \approx_L s2 \)

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

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

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

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

thus ?case

proof

assume \( \text{[b]} \ s1 = \text{Some true} \)

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

by (\text{auto intro:CondTrue-reds})

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

next

assume \( \text{[b]} \ s1 = \text{Some false} \)

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

by (\text{auto intro:CondFalse-reds})

from IH2[OF \( \Gamma,\text{High} \vdash c2; \text{this} \)] show ?thesis .

qed

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

note IH = \( \forall s1 \ s2. [\Gamma,\text{High} \vdash c'; \langle c',s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle] \Rightarrow \Gamma \vdash s1 \approx_L s2 \)

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

from IH[OF \text{this}]

have \( \forall s1 \ s2. [\langle c',s1 \rangle \rightarrow \langle \text{Skip},s2 \rangle] \Rightarrow \Gamma \vdash s1 \approx_L s2 \)

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

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

qed


lemma CondHighCompositionality:
proof (rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume Γ ⊢ s1 ≈L s2 and (if (b) c1 else c2, s1) →∗ ⟨Skip, s1'⟩
  and (if (b) c1 else c2, s2) →∗ ⟨Skip, s2'⟩
  show Γ ⊢ s1' ≈L s2'
    proof
      from Γ, High ⊢ (if (b) c1 else c2, ⟨if (b) c1 else c2, s1⟩) →∗ ⟨Skip, s1'⟩
      have Γ ⊢ s1 ≈L s1' by (auto dest: highBodies)
    from Γ, High ⊢ (if (b) c1 else c2, ⟨if (b) c1 else c2, s2⟩) →∗ ⟨Skip, s2'⟩
    have Γ ⊢ s2 ≈L s2' by (auto dest: highBodies)
    with Γ ⊢ s1 ≈L s2 have Γ ⊢ s1 ≈L s1' by (auto intro: lowEquivTransitive)
    from Γ ⊢ s1 ≈L s1' have Γ ⊢ s1 ≈L s1 by (auto intro: lowEquivSymmetric)
    with Γ ⊢ s1 ≈L s2' show ?thesis by (auto intro: lowEquivTransitive)
  qed

lemma CondLowCompositionality:
  assumes nonInterference Γ c1 and nonInterference Γ c2 and Γ ⊢ b : Low
  shows nonInterference Γ (if (b) c1 else c2)
proof (rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume Γ ⊢ s1 ≈L s2 and (if (b) c1 else c2, s1) →∗ ⟨Skip, s1'⟩
  and (if (b) c1 else c2, s2) →∗ ⟨Skip, s2'⟩
  from ⟨if (b) c1 else c2, s1⟩ →∗ ⟨Skip, s1'⟩
  have ⟦b⟧ s1 = Some true by (auto intro: CondTrue-or-False)
  thus Γ ⊢ s1' ≈L s2'
    proof
      assume ⟦b⟧ s1 = Some true
      with ⟦b⟧ s1 = ⟦b⟧ s2 have ⟦b⟧ s2 = Some true by (auto intro: CondTrue-reds)
      from ⟦b⟧ s1 = Some true ⟦if (b) c1 else c2, s1⟧ →∗ ⟨Skip, s1'⟩
      have (c1, s1) →∗ ⟨Skip, s1'⟩ by (auto intro: CondTrue-reds)
      from ⟦b⟧ s2 = Some true ⟦if (b) c1 else c2, s2⟧ →∗ ⟨Skip, s2'⟩
      have (c1, s2) →∗ ⟨Skip, s2'⟩ by (auto intro: CondTrue-reds)
      with (c1, s1) →∗ ⟨Skip, s1'⟩ Γ ⊢ s1 ≈L s2 ⟦if b⟧ Γ ⊢ nonInterference Γ c1
      show ?thesis by (auto simp: nonInterference_def)
next
  assume ⟦b⟧ s1 = Some false
  with ⟦b⟧ s1 = ⟦b⟧ s2 have ⟦b⟧ s2 = Some false by (auto intro: CondTrue-reds)
  from ⟦b⟧ s1 = Some false ⟦if (b) c1 else c2, s1⟧ →∗ ⟨Skip, s1'⟩
  have (c2, s1) →∗ ⟨Skip, s1'⟩ by (auto intro: CondFalse-reds)
  from ⟦b⟧ s2 = Some false ⟦if (b) c1 else c2, s2⟧ →∗ ⟨Skip, s2'⟩
  have (c2, s2) →∗ ⟨Skip, s2'⟩ by (auto intro: CondFalse-reds)
  with (c2, s1) →∗ ⟨Skip, s1'⟩ Γ ⊢ s1 ≈L s2 ⟦if b⟧ Γ ⊢ nonInterference Γ c2
show \( \text{thesis} \) by (auto simp: nonInterference-def)
qed
qed

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

lemma WhileLowStepInduct:
assumes \( \text{while}1: \langle \text{while} \ (b) \ c', s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle \)
and \( \text{while}2: \langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle \)
and \( \Gamma \vdash b : \text{Low} \)
and \( \text{body} : \forall s_1 s_1' s_2 s_2'. \ (\langle c', s_1 \rangle \rightarrow^* \langle \text{Skip}, s_1' \rangle ; \langle c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle; \)
  \( \Gamma \vdash s_1 \approx_L s_2 \) \implies \( \Gamma \vdash s_1' \approx_L s_2' \)
and \( \text{le} : \Gamma \vdash s_1 \approx_L s_2 \)
shows \( \Gamma \vdash s_1' \approx_L s_2' \)
using \( \text{while}1 \) \( \text{while}2 \)
proof (induct arbitrary: \( s_2 \) rule: while-reds-induct)
  case (false \( s_1 \))
  from \( \Gamma \vdash b : \text{Low} \) \( \Gamma \vdash s_1 \approx_L s_2 \) have \( [b] \) \( s_1 = [b] \) \( s_2 \) by (auto intro: interpretLow2)
  with \( [b] \) \( s_1 = \text{Some false} \) have \( [b] \) \( s_2 = \text{Some false} \) by simp
  with \( \langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle \) have \( s_2 = s_2' \) by (auto intro: WhileFalse-reds)
  with \( \Gamma \vdash s_1 \approx_L s_2 \) show \( \text{thesis} \) by auto
next
case (true \( s_1 \) \( s_1' \))
  note 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 (auto intro: interpretLow2)
  with \( [b] \) \( s_1 = \text{Some true} \) have \( [b] \) \( s_2 = \text{Some true} \) by simp
  with \( \langle \text{while} \ (b) \ c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle \) obtain \( s_2'' \) where \( \langle c', s_2 \rangle \rightarrow^* \langle \text{Skip}, s_2'' \rangle \)
  and \( \langle \text{while} \ (b) \ c', s_2'' \rangle \rightarrow^* \langle \text{Skip}, s_2' \rangle \) by (auto dest: WhileTrue-reds)

21
from \( \text{body}(\text{OF} \ (c', s1) \rightarrow (\text{Skip}, s1')) \rightarrow* (\text{Skip}, s2') \) have \( \Gamma \vdash s1 \approx_L s2' \).

from \( \text{IH}(\text{OF} \ (c', s2') \rightarrow* (\text{Skip}, s2')) \) show \ ?case .

qed

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

proof (rule nonInterferenceI)

fix \( s1 \ s2 \ s1' \ s2' \)
assume \( \Gamma \vdash s1 \approx_L s2 \) and \( (\text{while} (b) c', s1) \rightarrow (\text{Skip}, s1') \)
and \( (\text{while} (b) c', s2) \rightarrow* (\text{Skip}, s2') \)

\{ fix \( s1 \ s2 \ s1'' \ s2'' \)
assume \( (c', s1) \rightarrow* (\text{Skip}, s1') \) and \( (c', s2') \rightarrow* (\text{Skip}, s2'') \)
and \( \Gamma \vdash s1 \approx_L s2 \)
with nonInterference \( \Gamma \ c' \) have \( \Gamma \vdash s1'' \approx_L s2'' \)
by (auto simp: nonInterference-def)
\}

hence \( \forall s1 s1'' s2 s2''. \ [(c', s1) \rightarrow* (\text{Skip}, s1'); (c', s2') \rightarrow* (\text{Skip}, s2''); \ \Gamma \vdash s1 \approx_L s2 \] \( \implies \Gamma \vdash s1'' \approx_L s2'' \) by auto

with \( (\text{while} (b) c', s1) \rightarrow* (\text{Skip}, s1') \) \( (\text{while} (b) c', s2) \rightarrow* (\text{Skip}, s2') \)
\( \Gamma \vdash b: \text{Low} \) \( \Gamma \vdash s1 \approx_L s2 \) show \( \Gamma \vdash s1'' \approx_L s2'' \)
by (auto intro: WhileLowStepInduct)

qed

and now: the main theorem:

theorem sectypeimpliesnoninterference:
\( \Gamma , T \vdash c \implies\) nonInterference \( \Gamma \ c \)

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

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

proof (rule nonInterferenceI)

fix \( s1 \ s2 \ s1' \ s2' \)
assume \( \Gamma \vdash s1 \approx_L s2 \) and \( (\text{Skip}, s1) \rightarrow* (\text{Skip}, s1') \) and \( (\text{Skip}, s2) \rightarrow* (\text{Skip}, s2') \)
from \( (\text{Skip}, s1) \rightarrow* (\text{Skip}, s1') \) have \( s1 = s1' \) by (auto dest: Skip-reds)
from \( (\text{Skip}, s2) \rightarrow* (\text{Skip}, s2') \) have \( s2 = s2' \) by (auto dest: Skip-reds)
from \( \Gamma \vdash s1 \approx_L s2 \) and \( s1 = s1' \) and \( s2 = s2' \)
show \( \Gamma \vdash s1' \approx_L s2' \) by simp

qed

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

proof (rule nonInterferenceI)

22
\[
\text{fix } s_1 \; s_2 \; s_1' \; s_2' \\
\text{assume } \Gamma \vdash s_1 \equiv_L s_2 \text{ and } (V := e, s_1) \rightarrow* (\text{Skip}, s_1') \text{ and } (V := e, s_2) \rightarrow* (\text{Skip}, s_2') \\
\text{from } (V := e, s_1) \rightarrow* (\text{Skip}, s_1') \text{ have } s_1' = s_1(V := [e] \; s_1) \text{ by (auto intro:LAss-reds)} \\
\text{from } (V := e, s_2) \rightarrow* (\text{Skip}, s_2') \text{ have } s_2' = s_2(V := [e] \; s_2) \text{ by (auto intro:LAss-reds)} \\
\text{from } \text{earpren} \; \text{show } \Gamma \vdash s_1' \equiv_L s_2' \\
\text{proof } \\
\text{assume } \Gamma \vdash V = \text{Some High} \\
\text{with } \Gamma \vdash s_1 \equiv_L s_2 ; (s_1' = s_1(V := [e] \; s_1)) \; (s_2' = s_2(V := [e] \; s_2)) \text{ have } \Gamma \vdash s_1' = s_2' \text{ by (auto intro:assignNlhighlemma)} \\
\text{next } \\
\text{assume } \Gamma \vdash V = \text{Some Low} \land \Gamma \vdash e : \text{Low} \land T = \text{Low} \\
\text{with } \Gamma \vdash s_1 \equiv_L s_2 ; (s_1' = s_1(V := [e] \; s_1)) \; (s_2' = s_2(V := [e] \; s_2)) \text{ have } \Gamma \vdash s_1' = s_2' \text{ by (auto intro:assignNlowlemma)} \\
\text{qed} \\
\text{qed} \\
\text{next } \\
\text{case } (\text{Seq } c_1 \; c_2) \\
\text{note } IH1 = (\bigwedge T. \; \Gamma, T \vdash c_1 \Rightarrow \text{nonInterference } \Gamma \; c_1) \\
\text{note } IH2 = (\bigwedge T. \; \Gamma, T \vdash c_2 \Rightarrow \text{nonInterference } \Gamma \; c_2) \\
\text{show } \exists \text{case } \\
\text{proof } (\text{cases } T) \\
\text{case } \text{High} \\
\text{with } \Gamma, T \vdash c_1 ; c_2 \text{ have } \Gamma \vdash \text{High} \vdash c_1 \text{ and } \Gamma \vdash \text{High} \vdash c_2 \\
\text{by (auto elim:secComTyping.cases) \\
\text{from } IH1(OF \; \Gamma, \text{High} \vdash c_1) \text{ have } \text{nonInterference } \Gamma \; c_1 \text{.} \\
\text{moreover } \\
\text{from } IH2(OF \; \Gamma, \text{High} \vdash c_2) \text{ have } \text{nonInterference } \Gamma \; c_2 \text{.} \\
\text{ultimately show } \exists \text{thesis by (auto intro:SeqCompositionality) \\
\text{next } \\
\text{case } \text{Low} \\
\text{with } \Gamma, T \vdash c_1 ; c_2 \text{ have } (\Gamma, \text{Low} \vdash c_1 \land \Gamma, \text{Low} \vdash c_2) \lor \Gamma \vdash \text{High} \vdash c_1 ; c_2 \\
\text{by (auto elim:secComTyping.cases) \\
\text{thus } \exists \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} \\
\text{from } IH1(OF \; \Gamma, \text{Low} \vdash c_1) \text{ have } \text{nonInterference } \Gamma \; c_1 \text{.} \\
\text{moreover } \\
\text{from } IH2(OF \; \Gamma, \text{Low} \vdash c_2) \text{ have } \text{nonInterference } \Gamma \; c_2 \text{.} \\
\text{ultimately show } \exists \text{thesis by (auto intro:SeqCompositionality) \\
\text{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) \\
\text{from } IH1(OF \; \Gamma, \text{High} \vdash c_1) \text{ have } \text{nonInterference } \Gamma \; c_1 \text{.} \\
\text{moreover } \\
\text{from } IH2(OF \; \Gamma, \text{High} \vdash c_2) \text{ have } \text{nonInterference } \Gamma \; c_2 \text{.} \\
\]
ultimately show ?thesis by(auto intro:SeqCompositionality)

qed

next

case (Cond b c1 c2)

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

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

show ?case

proof (cases T)

case High

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

by(auto intro:CondHighCompositionality)

next

case Low

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

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

by(auto elim:secComTyping.cases)

thus ?thesis

proof

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

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

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

moreover

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

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

by(auto intro:CondLowCompositionality)

next

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

thus ?thesis by(auto intro:CondHighCompositionality)

qed

qed

next

case (While b c')

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

show ?case

proof (cases T)

case High

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

next

case Low

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

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

by(auto elim:secComTyping.cases)

thus ?thesis

proof

assume \( \Gamma \vdash b : \text{Low} \land \Gamma, \text{Low} \vdash c' \)

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

moreover

from IH[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, \( i \Rightarrow i \Rightarrow i \Rightarrow \text{bool} \) as compute-secExprTyping, \( i \Rightarrow i \Rightarrow i \Rightarrow \text{bool} \) as check-secExprTyping) secExprTyping

proof –
case secExprTyping
from secExprTyping.prems show thesis
proof
fix \( \Gamma V \) lev assume \( x = \Gamma xa = \text{Val} V xb = \text{lev} \)
from secExprTyping(1−2) this show thesis by (cases lev) auto
next
fix \( \Gamma Vn \) lev
assume \( x = \Gamma xa = \text{Var} Vn xb = \text{lev} \) \( \Gamma Vn = \text{Some lev} \)
from secExprTyping(3) this show thesis by (auto simp add: Predicate.eq-is-eq)
next
fix \( \Gamma e1 e2 \) bop
assume \( x = \Gamma xa = e1\text{bop} \ e2 xb = \text{Low} \)
\( \Gamma \vdash e1 : \text{Low} \) \( \Gamma \vdash e2 : \text{Low} \)
from secExprTyping(4) this show thesis by auto
next
fix \( \Gamma e1 e2 \) lev bop
assume \( x = \Gamma xa = e1\text{bop} \ e2 xb = \text{High} \)
\( \Gamma \vdash e1 : \text{High} \) \( \Gamma \vdash e2 : \text{lev} \)
from secExprTyping(5–6) this show thesis by (cases lev) (auto)
next
fix \( \Gamma \) \( e1 \) \( e2 \) \( lev \) \( bop \)
assume \( x = \Gamma \) \( xa = e1 \ast bop \ast e2 \) \( xb = High \)
\( \Gamma \vdash e1 : lev \) \( \Gamma \vdash e2 : High \)
from secExprTyping(6–7) this show thesis by (cases lev) (auto)
qed
qed

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

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

thm secComTyping.equation
3.1 An example taken from Volpano, Smith, Irvine

definition \texttt{com} = \texttt{if (Var ”x” ”Eq” Val (Intg 1)) (”y” := Val (Intg 1)) else (”y” := Val (Intg 0))}
definition \texttt{Env} = \texttt{map-of [”x” High], (”y” High]}
values \{ T. Env \vdash (Var ”x” ”Eq” Val (Intg 1)): T \}
value \texttt{Env}, High \vdash \texttt{com}
value \texttt{Env}, Low \vdash \texttt{com}
values 1 \{ T. Env, T \vdash \texttt{com}\}

\texttt{Env’} = \texttt{map-of [”x” Low], (”y” High]}
value \texttt{Env’}, Low \vdash \texttt{com}
value \texttt{Env’}, High \vdash \texttt{com}
values 1 \{ T. Env, T \vdash \texttt{com}\}

end

References
