An Isabelle/HOL formalization of Strong Security

Sylvia Grewe, Alexander Lux, Heiko Mantel, Jens Sauer

April 20, 2020

Abstract

Research in information-flow security aims at developing methods to identify undesired information leaks within programs from private sources to public sinks. Noninterference captures this intuition. Strong security from [2] formalizes noninterference for concurrent systems.

We present an Isabelle/HOL formalization of strong security for arbitrary security lattices ([2] uses a two-element security lattice). The formalization includes compositionality proofs for strong security and a soundness proof for a security type system that checks strong security for programs in a simple while language with dynamic thread creation.

Our formalization of the security type system is abstract in the language for expressions and in the semantic side conditions for expressions. It can easily be instantiated with different syntactic approximations for these side conditions. The soundness proof of such an instantiation boils down to showing that these syntactic approximations imply the semantic side conditions.

Contents

1 Preliminary definitions 2
  1.1 Type synonyms ........................................... 2

2 Strong security 4
  2.1 Definition of strong security .......................... 4
  2.2 Proof technique for compositionality results ......... 8
  2.3 Proof of parallel compositionality .................... 10

3 Example language and compositionality proofs 13
  3.1 Example language with dynamic thread creation .... 13
  3.2 Proofs of atomic compositionality results .......... 15
  3.3 Proofs of non-atomic compositionality results ....... 18
1 Preliminary definitions

1.1 Type synonyms

The formalization is parametric in different aspects. Notably, it is parametric in the security lattice it supports.

For better readability, we use the following type synonyms in our formalization:

```
theory Types
imports Main
begin

— type parameters:
— 'exp: expressions (arithmetic, boolean...)
— 'val: values
— 'id: identifier names
— 'com: commands
— 'd: domains

This is a collection of type synonyms. Note that not all of these type synonyms are used within Strong-Security - some are used in WHATandWHERE-Security.

— type for memory states - map ids to values
```
type-synonym ('id, 'val) State = 'id ⇒ 'val
```

— type for evaluation functions mapping expressions to a values depending on a state
```
type-synonym ('exp, 'id, 'val) Evalfunction = 'exp ⇒ ('id, 'val) State ⇒ 'val
```

— define configurations with threads as pair of commands and states
```
type-synonym ('id, 'val, 'com) TConfig = 'com × ('id, 'val) State
```

— define configurations with thread pools as pair of command lists (thread pool) and states
```
type-synonym ('id, 'val, 'com) TPConfig = ('com list) × ('id, 'val) State
```

— type for program states (including the set of commands and a symbol for terminating - None)
```
type-synonym 'com ProgramState = 'com option
```
— type for configurations with program states
\textbf{type-synonym} ('id, 'val, 'com) \textit{PSConfig} =
\hspace{1em} 'com ProgramState × ('id, 'val) State

— type for labels with a list of spawned threads
\textbf{type-synonym} 'com Label = 'com list

— type for step relations from single commands to a program state, with a label
\textbf{type-synonym} ('exp, 'id, 'val, 'com) \textit{TLSteps} =
\hspace{1em} (('id, 'val, 'com) \textit{TConfig} × 'com Label
\hspace{1em} \times ('id, 'val, 'com) \textit{PSConfig}) set

— curried version of previously defined type
\textbf{type-synonym} ('exp, 'id, 'val, 'com) \textit{TLSteps-curry} =
\hspace{1em} 'com \Rightarrow ('id, 'val) State \Rightarrow 'com Label \Rightarrow 'com ProgramState
\hspace{1em} \Rightarrow ('id, 'val) State \Rightarrow bool

— type for step relations from thread pools to thread pools
\textbf{type-synonym} ('exp, 'id, 'val, 'com) \textit{TPSteps} =
\hspace{1em} ((('id, 'val, 'com) \textit{TPConfig} × ('id, 'val, 'com) \textit{TPConfig}) set

— curried version of previously defined type
\textbf{type-synonym} ('exp, 'id, 'val, 'com) \textit{TPSteps-curry} =
\hspace{1em} 'com list \Rightarrow ('id, 'val) State \Rightarrow 'com list \Rightarrow ('id, 'val) State \Rightarrow bool

— define type of step relations for single threads to thread pools
\textbf{type-synonym} ('exp, 'id, 'val, 'com) \textit{TSteps} =
\hspace{1em} ((('id, 'val, 'com) \textit{TConfig} × ('id, 'val, 'com) \textit{TPConfig}) set

— define the same type as \textit{TSteps}, but in a curried version (allowing syntax abbreviations)
\textbf{type-synonym} ('exp, 'id, 'val, 'com) \textit{TSteps-curry} =
\hspace{1em} 'com list \Rightarrow ('id, 'val) State \Rightarrow 'com list \Rightarrow ('id, 'val) State \Rightarrow bool

— type for simple domain assignments; 'd has to be an instance of order (partial order)
\textbf{type-synonym} ('id, 'd) \textit{DomainAssignment} = 'id \Rightarrow 'd::order

\textbf{type-synonym} 'com \textit{Bisimulation-type} = (('com list) × ('com list)) set

— type for escape hatches
\textbf{type-synonym} ('d, 'exp) \textit{Hatch} = 'd × 'exp

— type for sets of escape hatches
\textbf{type-synonym} ('d, 'exp) \textit{Hatches} = (('d, 'exp) \textit{Hatch}) set

— type for local escape hatches
\textbf{type-synonym} ('d, 'exp) l\textit{Hatch} = 'd × 'exp × nat
type-synonym (′d, ′exp) lHatches = ((′d, ′exp) lHatch) set

2 Strong security

2.1 Definition of strong security

We define strong security such that it is parametric in a security lattice (′d). The definition of strong security by itself is language-independent, therefore the definition is parametric in a programming language (′com) in addition.
— predicate for strong d-bisimulation

**Definition** Strong-d-Bisimulation :: 'd ⇒ 'com Bisimulation-type ⇒ bool

**Where**

Strong-d-Bisimulation d R ≡

(sym R) ∧

(∀ (V, V') ∈ R. length V = length V') ∧

∀ (V, V') ∈ R. ∀ i < length V. ∀ m1 m1' m2 W.

(V!i,m1) → (W,m2) ∧ m1 =<d m1' → (∃ W'. m2' (V!i,m1') → (W'.m2') ∧ (W,W') ∈ R ∧ m2 =<d m2'))

— union of all strong d-bisimulations

**Definition** USdB :: 'd ⇒ 'com Bisimulation-type α

**Where**

≈<d ≡ ⋃ {r. (Strong-d-Bisimulation d r)}

**Abbreviation** relatedbyUSdB :: 'com list ⇒ 'd ⇒ 'com list ⇒ bool

**Where**

V ≈<d V' ≡ (V, V') ∈ USdB d

— predicate to define when a program is strongly secure

**Definition** Strongly-Secure :: 'com list ⇒ bool

**Where**

Strongly-Secure V ≡ (∀ d. V ≈<d V)

— auxiliary lemma to obtain central strong d-Bisimulation property as Lemma in meta logic (allows instantiating all the variables manually if necessary)

**Lemma** strongdB-aux: (∀ i < length V ; (V, V') ∈ R; (V!i,m1) → (W,m2); m1 =<d m1') → (∃ W'. m2'. (V!i,m1') → (W'.m2') ∧ (W,W') ∈ R ∧ m2 =<d m2'))

**By** (simp add: Strong-d-Bisimulation-def, fastforce)

**Lemma** trivialpair-in-USdB:

[[] ≈<d []]

**By** (simp add: USdB-def Strong-d-Bisimulation-def, rule-tac x={([],[])}) in exI, simp add: sym-def)

**Lemma** USdBSym: sym (≈<d)

**By** (simp add: USdB-def Strong-d-Bisimulation-def sym-def, auto)

**Lemma** USdBEqLen:

V ≈<d V' ⇒ length V = length V'

**By** (simp add: USdB-def Strong-d-Bisimulation-def, auto)

**Lemma** USdB-Strong-d-Bisimulation:

Strong-d-Bisimulation d (≈<d)

**Proof** (simp add: Strong-d-Bisimulation-def, auto)

**Show** sym (≈<d) by (rule USdBSym)
next
  fix V V'
  show V \approx_d V' \implies \text{length } V = \text{length } V' by (rule USdBeqlen, auto)

next
  fix V V' m1 m1' m2 W i
  assume inUSdB: V \approx_d V'
  assume stepV: (V!i,m1) \to (W,m2)
  assume irange: i < \text{length } V
  assume dequal: m1 =_d m1'

from inUSdB obtain R where someR:
  Strong-d-Bisimulation \ d R \land (V,V') \in R
by (simp add: USdB-def, auto)

with strongdB-aux stepV irange dequal show
  \exists W' m2'. (V!i,m1') \to (W',m2') \land W \approx_d W' \land m2 =_d m2'
by (simp add: USdB-def, fastforce)

qed

lemma USdBtrans; trans (\approx_d)
proof (simp add: trans-def, auto)
  fix V V''
  assume p1: V \approx_d V'
  assume p2: V' \approx_d V''

  let ?R = \{(V, V''). \exists V'. V \approx_d V' \land V' \approx_d V''\}

from p1 p2 have inRest: (V, V'') \in ?R by auto

have SdB-rest: Strong-d-Bisimulation \ d ?R
proof (simp add: Strong-d-Bisimulation-def sym-def, auto)
  fix V V''
  assume p1: V \approx_d V'
  moreover
  assume p2: V' \approx_d V''
  moreover
  from p1 USdBsym have V' \approx_d V
    by (simp add: sym-def)
  moreover
  from p2 USdBsym have V'' \approx_d V' 
    by (simp add: sym-def)
  ultimately show
    \exists V', V'' \approx_d V' \land V' \approx_d V
    by (rule_tac x=V' in exI, auto)

next
  fix V V''
  assume p1: V \approx_d V'
moreover
assume \( p2 \): \( V' \approx_d V'' \)
moreover
from \( p1 \) USdB-eqlen[of \( V V' \)] have \( \text{length } V = \text{length } V' \)
  by auto
moreover
from \( p2 \) USdB-eqlen[of \( V' V'' \)] have \( \text{length } V' = \text{length } V'' \)
  by auto
ultimately show eqlen: \( \text{length } V = \text{length } V'' \) by auto
next
fix \( V V' V'' i m1 m1' W m2 \)
assume step: \( \langle V!i,m1 \rangle \rightarrow \langle W,m2 \rangle \)
assume dequal: \( m1 =_d m1' \)
assume \( p1 \): \( V \approx_d V' \)
assume \( p2 \): \( V' \approx_d V'' \)
assume irange: \( i < \text{length } V \)
from \( p1 \) USdB-eqlen[of \( V V' \)]
have leq: \( \text{length } V = \text{length } V' \)
  by force
have deq-same: \( m1' =_d m1' \) by (simp add: d-equal-def)
from irange step dequal \( p1 \) USdB-Strong-d-Bisimulation
strongdB-aux[of \( d \approx_d i V V' m1 W m2 m1' \)]
obtain \( W' m2' \) where \( p1concl: \)
  \( \langle V'!i,m1' \rangle \rightarrow \langle W',m2' \rangle \land W \approx_d W' \land m2 =_d m2' \)
  by auto
with deq-same leq USdB-Strong-d-Bisimulation
strongdB-aux[of \( d \approx_d i V' V'' m1' W' m2' m1' \)]
irange \( p2 \) dequal obtain \( W'' m2'' \) where \( p2concl: \)
  \( W' \approx_d W'' \land \langle V''!i,m1' \rangle \rightarrow \langle W'',m2'' \rangle \land m2' =_d m2'' \)
  by auto
from \( p1concl \) \( p2concl \) d-equal-trans have \( tt'': m2 =_d m2'' \)
  by blast
from \( p1concl \) \( p2concl \) have \( (W,W'') \in ?R \)
  by auto
with \( p2concl \) \( tt'' \) show \( \exists W'' m2''. \langle V''!i,m1' \rangle \rightarrow \langle W'',m2'' \rangle \land \)
  \( (\exists V'. W \approx_d V' \land V' \approx_d W'') \land m2 =_d m2'' \)
  by auto
qed

hence liftup: \( ?R \subseteq (\approx_d) \)
  by (simp add: USdB-def, auto)
with inRest show \( V \approx_d V'' \)
2.2 Proof technique for compositionality results

For proving compositionality results for strong security, we formalize the following “up-to technique” and prove it sound:

theory Up-To-Technique
imports Strong-Security
begin

context Strong-Security
begin

— define d-bisimulation ’up to’ union of strong d-Bisimulations

definition d-Bisimulation-Up-To-USdB ::
\[
d \Rightarrow \text{Bisimulation-type} \Rightarrow \text{bool}
\]

where

d-Bisimulation-Up-To-USdB d R
\[\equiv (\text{sym } R) \land (\forall (V,V') \in R. \text{length } V = \text{length } V') \land (\forall (V,V') \in R. \forall i < \text{length } V. \forall m1 m1' W m2. \\
\langle V!i,m1 \rangle \rightarrow \langle W,m2 \rangle \land (m1 =_d m1') \rightarrow (\exists W' m2'. \langle V'!i,m1' \rangle \rightarrow \langle W',m2' \rangle \\
\land (W,W') \in (R \cup (\approx_d)) \land (m2 =_d m2')))
\]

lemma UpTo-ux: \[\forall V V' m1 m1' m2 W i. [ d-Bisimulation-Up-To-USdB d R; \\
i < \text{length } V; (V,V') \in R; (V!i,m1) \rightarrow (W,m2); m1 =_d m1' ] \\
\Rightarrow (\exists W' m2'. (V'!i,m1') \rightarrow (W',m2') \\
\land (W,W') \in (R \cup (\approx_d)) \land (m2 =_d m2'))
\]

by (simp add: d-Bisimulation-Up-To-USdB-def, fastforce)

lemma RaUSdBeqlen:
\[\forall d-Bisimulation-Up-To-USdB d R; \\
(V,V') \in (R \cup (\approx_d)) \] 
\Rightarrow \text{length } V = \text{length } V'

by (auto, simp add: d-Bisimulation-Up-To-USdB-def, auto, rule USdBeqlen, auto)

lemma Up-To-Technique:

assumes upToR: d-Bisimulation-Up-To-USdB d R

shows R \subseteq \approx_d

proof –
define $S$ where $S = R \cup (\approx_d)$
from $S$-def have $R \subseteq S$
by auto
moreover have $S \subseteq (\approx_d)$
proof (simp add: USdB-def, auto, rule_tac x=S in exI, auto,
  simp add: Strong-d-Bisimulation-def, auto)
  — show symmetry
show $\text{sym} S$: $\text{sym}\ S$
proof
  from upToR have $R\text{sym}$: $\text{sym}\ R$
  by (simp add: d-Bisimulation-Up-To-USdB-def)
  with USdBsym have $Usym$: $\text{sym}\ (R \cup (\approx_d))$
  by (metis sym-Un)
  with $S$-def show $\exists\ \text{thesis}$
  by simp
qed
next
fix $V\ V'$
assume $\text{inS}: (V, V') \in S$
  — show equal length (by definition)
from $\text{inS}$ $S$-def upToR RuUSdBEqlen
show eqlen: $\text{length}\ V = \text{length}\ V'$
by simp
next
  — show general bisimulation property
fix $V\ V'\ W\ m1\ m1'\ m2\ i$
assume $\text{inS}: (V, V') \in S$
assume irange: $i < \text{length}\ V$
assume stepV: $\langle V!i, m1 \rangle \rightarrow \langle W, m2 \rangle$
assume dequal: $m1 =_d m1'$
from $\text{inS}$ show $\exists\ W'\ m2': (V!i, m1') \rightarrow (W', m2') \land\ (W, W') \in S \land m2 =_d m2'$
proof (simp add: $S$-def, auto)
  assume firstcase: $(V, V') \in R$
  with upToR dequal irange stepV
  $\text{UpTo-aux}\ [d\ R\ i\ V\ V'\ m1\ W\ m2\ m1']$
  show $\exists\ W'\ m2': (V!i, m1') \rightarrow (W', m2') \land\ (W, W') \in S \land m2 =_d m2'$
  by (auto simp add: $S$-def)
next
  assume secondcase: $V \approx_d V'$
from USdB-Strong-d-Bisimulation upToR
secondcase dequal irange stepV
...strongdB-aux[of d ≈di V V′ m1 W m2 m1′]
show ∃W′ m2′. ⟨V ′i,m1′⟩ → ⟨W′,m2′⟩ ∧
((W,W′) ∈ R ∨ W ≈d W′) ∧ m2 ≈d m2′
by auto
qed
qed
ultimately show thesis by auto
qed
end
end

2.3 Proof of parallel compositionality

We prove that strong security is preserved under composition of strongly
secure threads.

theory Parallel-Composition
imports Up-To-Technique
begin
context Strong-Security
begin

theorem parallel-composition:
assumes eqlen: length V = length V′
assumes partsrelated: ∀ i < length V. [V!i] ≈d [V′!i]
shows V ≈d V′
proof −
define R where R = {(V,V′). length V = length V′
∧ (∀ i < length V. [V!i] ≈d [V′!i])}
from eqlen partsrelated have inR: (V,V′) ∈ R
by (simp add: R-def)

have dB-Bisimulation-Up-To-USdB d R
proof (simp add: dB-Bisimulation-Up-To-USdB-def, auto)
from USdBsym show sym R
by (simp add: R-def sym-def)
next
fix V V′
assume (V,V′) ∈ R
with USdBeqlen show length V = length V′
by (simp add: R-def)
next
fix V V′ i m1 m1′ RS m2
assume inR: (V,V′) ∈ R
assume irange: i < length V
assume step:\ \langle V!i,m1 \rangle \rightarrow \langle RS,m2 \rangle  
assume dequal:\ m1 =_d m1' 

from inR have Vassump:\ 
- length \ V = length \ V' \land (\forall i < length \ V. [V!i] \approx_d [V'!i])  
  by (simp add: R-def)  

with step dequal USdB-Strong-d-Bisimulation irange 
strongdB-aux[of d \approx_d 0 \ [V!i] \ [V'!i] \ m1 \ RS \ m2 \ m1']  
show \ \exists RS' \ m2'. \ \langle V'!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land  
((RS,RS') \in R \lor RS \approx_d RS') \land m2 =_d m2'  
  by (simp, fastforce) 
qed 

hence \ R \subseteq (\approx_d)  
  by (rule Up-To-Technique)  
with inR show \ ?thesis by auto  
qed 

lemma \ parallel-decomposition:\  
assumes related:\ V \approx_d V'  
shows \ \forall i < length \ V. [V!i] \approx_d [V'!i]  
proof -  
  define \ R \ where\ R = \{(C,C'). \ \exists i W W'. \ W \approx_d W' \land i < length W  
  \land C = [W!i] \land C' = [W'!i]\}  

with related have inR: \ \forall i < length \ V. ([V!i],[V'!i]) \in R  
  by auto  
have d-Bisimulation-Up-To-USdB d R  
  proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)  
    from USdBsym USdBqeqen show sym R  
      by (simp add: sym-def R-def, metis)  
next 
  fix \ C C'  
  assume (C,C') \in R  
  with USdBqeqen show length C = length C'  
    by (simp add: R-def, auto)  
next 
  fix \ C C' i m1 m1' RS m2  
  assume inR: \ (C,C') \in R  
  assume irange: \ i < length C  
  assume step: \ \langle C!i,m1 \rangle \rightarrow \langle RS,m2 \rangle  
  assume dequal: \ m1 =_d m1'  

from inR obtain j W W' where \ Rassump:\  
- W \approx_d W' \land j < length W \land C = [W'j] \land C' = [W'j] 

11
by \((\text{simp add: } R\text{-def}, \text{auto})\)

with \(\text{irange have } i0: i = 0 \text{ by } \text{auto}\)

from \(\text{Rassump } i0 \text{ strongdB-aux}[\text{of } d \approx_d j \text{ W } W']\)

\(\text{USdB-Strong-d-Bisimulation step dequal}\)

show \(\exists RS' \text{ m2'}. \langle C'\#i', m1' \rangle \rightarrow \langle RS', m2' \rangle\)

\& ((\text{RS}, \text{RS}') \in R \lor RS \approx_d RS') \land m2 =_d m2'

by \(\text{auto}\)

qed

hence \(R \subseteq (\approx_d)\)

by \((\text{rule Up-To-Technique})\)

with \(\text{inR show } \text{?thesis}\)

by \(\text{auto}\)

\(\text{qed}\)

\(\text{lemma USdB-comp-head-tail:}\)

assumes \(\text{relatedhead: } [c] \approx_d [c']\)

assumes \(\text{relatedtail: } V \approx_d V'\)

shows \((c\#V) \approx_d (c'\#V')\)

proof

from \(\text{relatedtail USdBeqlen have eqlen: length } (c\#V) = \text{length } (c'\#V')\)

by \(\text{force}\)

from \(\text{relatedtail parallel-decomposition have singleV:}\)

\(\forall i < \text{length } V. [V!i] \approx_d [V'!i]\)

by \(\text{force}\)

with \(\text{relatedhead have intermediate:}\)

\(\forall i < \text{length } (c\#V). [(c\#V)!i] \approx_d [(c'\#V')!i]\)

by \((\text{auto, case-tac } i, \text{auto})\)

with \(\text{eqlen parallel-composition}\)

show \(\text{?thesis}\)

by \(\text{blast}\)

\(\text{qed}\)

\(\text{lemma USdB-decomp-head-tail:}\)

assumes \(\text{relatedlist: } (c\#V) \approx_d (c'\#V')\)

shows \([c] \approx_d [c'] \land V \approx_d V'\)

proof \(\text{auto}\)

from \(\text{relatedlist USdBeqlen[of c\#V c'\#V'] have eqlen: length } V = \text{length } V'\)

by \(\text{auto}\)
from relatedlist parallel-decomposition[of c#V c#V' d]

have intermediate:
\forall i < length (c#V). [(c#V)!i] \approx_d [(c#V')!i]
by auto
thus [c] \approx_d [c']
by force

from intermediate eqlen show V \approx_d V'
proof (case-tac V)
    assume Vcase1: V = []
    with eqlen have V' = [] by auto
    with Vcase1 trivialpair-in-USdB show V \approx_d V'
    by auto

next
fix c1 W
assume Vcase2: V = c1#W
hence Vlen: length V > 0 by auto

from intermediate have intermediate-aux:
\forall i. i < length V
\implies [V!i] \approx_d [V'!i]
by force

with parallel-composition[of V V'] eqlen
show V \approx_d V'
by blast

qed

end

end

3 Example language and compositionality proofs

3.1 Example language with dynamic thread creation

As in [2], we instantiate the language with a simple while language that sup-
ports dynamic thread creation via a fork command (Multi-threaded While
Language with fork, MWLf). Note that the language is still parametric in
the language used for Boolean and arithmetic expressions (’exp).

theory MWLf
imports Types
begin
— SYNTAX

— Commands for the multi-threaded while language with fork (to instantiate 'com)

```
datatype ('exp, 'id) MWLfCom
  = Skip (skip)
  | Assign 'id 'exp
      (\x:= [70,70] 70)
  | Seq ('exp, 'id) MWLfCom ('exp, 'id) MWLfCom
      (\x: [61,60] 60)
  | If-Else 'exp ('exp, 'id) MWLfCom ('exp, 'id) MWLfCom
     (if - then - else - fi [80,79] 70)
  | While-Do 'exp ('exp, 'id) MWLfCom
     (while - do - od [80,79] 70)
  | Fork ('exp, 'id) MWLfCom (('exp, 'id) MWLfCom) list
     (fork - [70,70] 70)
```

— SEMANTICS

locale MWLf-semantics =
fixes E :: ('exp, 'id, 'val)
Evalfunction
and BMap :: 'val ⇒ bool
begin

— steps semantics, set of deterministic steps from single threads to either single
threads or thread pools

inductive-set MWLfSteps-det :: ('exp, 'id, 'val, ('exp, 'id) MWLfCom) TSteps
and MWLfSteps-det' :: ('exp, 'id, 'val, ('exp, 'id) MWLfCom) TSteps-curry
  ((λ(\x/\x)) \to/ (λ(\x/\x)) [0,0,0,0] 81)
where
\langle c1,m1 \rangle \to \langle c2,m2 \rangle \equiv ((c1,m1),(c2,m2)) \in MWLfSteps-det |
skip: \langle skip,m \rangle \to (\ [],m) |
assign: (E c m) = v \to \langle x := e,m \rangle \to (\ [],m(x := v)) |
seq1: \langle c1,m \rangle \to (\ [],m') \Longrightarrow \langle c1;c2,m \rangle \to (\ [],m') |
seq2: \langle c1,m \rangle \to (\c1#V,m') \Longrightarrow \langle c1;c2,m \rangle \to (\c1';c2)c#V,m' |
iftrue: BMap (E b m) = True \to 
  (if b then c1 else c2 fi,m) \to (\c1,m) |
iffalse: BMap (E b m) = False \to 
  (if b then c1 else c2 fi,m) \to (\c2,m) |
whiletrue: BMap (E b m) = True \to 
  (while b do c od,m) \to (\c:while b do c od),m) |
whilefalse: BMap (E b m) = False \to 
  (while b do c od,m) \to (\ [],m) |
fork: \langle fork V,m \rangle \to (\c#V,m)
inductive-cases \texttt{MWLfSteps-det-cases}:
\begin{itemize}
  \item \(\langle \text{skip}, m \rangle \rightarrow \langle W, m' \rangle\)
  \item \(\langle x := e, m \rangle \rightarrow \langle W, m' \rangle\)
  \item \(\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi }, m \rangle \rightarrow \langle W, m' \rangle\)
  \item \(\langle \text{while } b \text{ do } c \text{ od } m \rangle \rightarrow \langle W, m' \rangle\)
  \item \(\langle \text{fork } c \text{ V }, m \rangle \rightarrow \langle W, m' \rangle\)
\end{itemize}

— non-deterministic, possibilistic system step (added for intuition, not used in the proofs)

\textbf{inductive-set} \texttt{MWLfSteps-ndet} :: (\langle 'exp', 'id', 'val', (\langle 'exp', 'id' \rangle \text{ MWLfCom}) TPSteps \\
and \texttt{MWLfSteps-ndet}' :: (\langle 'exp', 'id', 'val', (\langle 'exp', 'id' \rangle \text{ MWLfCom}) TPSteps-curry \\
((1\langle -,- \rangle) \Rightarrow (1\langle -,- \rangle) [0,0,0,0] 81) \\
\textbf{where} \begin{align*}
  \langle V1,m1 \rangle & \Rightarrow \langle V2,m2 \rangle \equiv ((V1,m1),(V2,m2)) \in \text{ MWLfSteps-ndet} \\
  \langle ci,m \rangle & \Rightarrow \langle c,m' \rangle \Rightarrow \langle Vf \ @ \ [ci] \ @ \ Va,m \rangle \Rightarrow \langle Vf \ @ \ c \ @ \ Va,m \rangle
\end{align*}

end end

\textbf{3.2 Proofs of atomic compositionality results}

We prove for each atomic command of our example programming language (i.e. a command that is not composed out of other commands) that it is strongly secure if the expressions involved are indistinguishable for an observer on security level \(d\).

\textbf{theory} \texttt{Strongly-Secure-Skip-Assign} \\
\textbf{imports} \texttt{MWLf Parallel-Composition} \\
\textbf{begin} \\
\textbf{locale} \texttt{Strongly-Secure-Programs} = \\
L?: \text{ MWLf-semantics} \ E \ BMap \\
+ SS?: \text{ Strong-Security MWLfSteps-det DA} \\
\textbf{for} \ E :: ('exp', 'id', 'val') \text{ Evalfunction} \\
\textbf{and} \ BMap :: 'val \Rightarrow \text{ bool} \\
\textbf{and} \ DA :: ('id', 'd::order') \text{ DomainAssignment} \\
\textbf{begin} \\
\textbf{abbreviation} \texttt{USdBname} :: 'd \Rightarrow (\langle 'exp', 'id' \rangle \text{ MWLfCom} \text{ Bisimulation-type} \\
\texttt{(\approx)} \texttt{ where} \approx_d \equiv \text{ USdB} \ d \\
\textbf{abbreviation} \texttt{relatedbyUSdB} :: (\langle 'exp','id' \rangle \text{ MWLfCom} \text{ list} \Rightarrow 'd}
⇒ (′exp,′id) MWLfpCom list ⇒ bool (infixr ≈. 65)
where V ≈_d V′ ≜ (V, V′) ∈ USdB d

— define when two expressions are indistinguishable with respect to a domain d

definition d-indistinguishable :: ′d::order ⇒ ′exp ⇒ ′exp ⇒ bool
where
d-indistinguishable d e1 e2 ≜
  ∀ m m′. ((m =_d m′) −→ ((E e1 m) = (E e2 m′)))

abbreviation d-indistinguishable′ :: ′exp ⇒ ′d::order ⇒ ′exp ⇒ bool
( ( ≈ ) )
where
e1 ≡_d e2 ≡ d-indistinguishable d e1 e2

— symmetry of d-indistinguishable

lemma d-indistinguishable-sym:
e ≡_d e′ ≡ e′ ≡_d e
by (simp add: d-indistinguishable-def d-equal-def, metis)

— transitivity of d-indistinguishable

lemma d-indistinguishable-trans:
[ e ≡_d e′; e′ ≡_d e″ ] ⇒ e ≡_d e″
by (simp add: d-indistinguishable-def d-equal-def, metis)

theorem Strongly-Secure-Skip:
[skip] ≈_d [skip]
proof −
define R0 where R0 = \{(V::(′exp,′id) MWLfpCom list, V′::(′exp,′id) MWLfpCom list).
V = [skip] ∧ V′ = [skip]\}

have uptoR0: d-Bisimulation-Up-To-USdB d R0
proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)
  show sym R0 by (simp add: R0-def sym-def)
next
  fix V V′
  assume (V, V′) ∈ R0
  thus length V = length V′
  by (simp add: R0-def)
next
  fix V V′ i m1 m1′ W m2
  assume inR0: (V, V′) ∈ R0
  assume irange: i < length V
  assume step: ⟨V!i,m1⟩ ↠ ⟨W,m2⟩
  assume dequal: m1 =_d m1′
  from inR0 have Vassump:
    V = [skip] ∧ V′ = [skip]
  by (simp add: R0-def)
with step irange have step1:
  \( W = \[] \land m_2 = m_1 \)
  by (simp, metis MWLf-semantics.MWLfSteps-det-cases(1))

from Vassamp irange obtain \( m_2' \) where step2:
  \( \langle V'!i, m_1' \rangle \rightarrow (\[\], m_2') \land m_2' = m_1' \)
  by (simp, metis MWLfSteps-det.skip)

with step1 dequal trivialpair-in-USdB show \( \exists W' m_2' . \)
  \( \langle V'!i, m_1' \rangle \rightarrow \langle W', m_2' \rangle \land (\langle W, W' \rangle \in R_0 \lor W \approx_d W') \land m_2 = d m_2' \)
  by auto

qed

hence \( R_0 \subseteq \approx_d \)
  by (rule Up-To-Technique)

thus \( \diamondthesis \)
  by (simp add: R0-def)

qed

theorem Strongly-Secure-Assign:
  assumes d-indistinguishable-exp: \( e \equiv_{DA} x e' \)
  shows \( \{ x := e \} \approx_d \{ x := e' \} \)
proof -
  define \( R_0 \) where \( R_0 = \{ (V, V') . \exists x e e'. V = \{ x := e \} \land V' = \{ x := e' \} \land e \equiv_{DA} x e' \} \)

from d-indistinguishable-exp have inR0: \( \{ x := e \}, \{ x := e' \} \in R_0 \)
  by (simp add: R0-def)

have d-Bisimulation-Up-To-USdB d R0
proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)
  from d-indistinguishable-sym show sym R0
  by (simp add: R0-def sym-def, fastforce)

next
  fix \( V, V' \)
  assume \( (V, V') \in R_0 \)
  thus \( \text{length } V = \text{length } V' \)
  by (simp add: R0-def, auto)

next
  fix \( V, V', i, m_1, m_1', W, m_2 \)
  assume inR0: \( (V, V') \in R_0 \)
  assume irange: \( i < \text{length } V \)
  assume step: \( \langle V!i, m_1 \rangle \rightarrow \langle W, m_2 \rangle \)
  assume dequal: \( m_1 =_d m_1' \)

17
from \textit{in}R0 obtain \( x \in \text{e} \in \text{e}' \) \textbf{where Vassump:}
\[
V = [x := \text{e}] \land V' = [x := \text{e}'] \land \\
\text{e} \equiv DA x \in \text{e}'
\]
by (simp add: R0-def, auto)

with \textit{step} irange obtain \( v \) \textbf{where step1:}
\[
E e \in m1 = v \land W = \emptyset \land m2 = m1(x := v)
\]
by (auto, metis MWLf-semantics.MWLfSteps-det-cases(2))

from Vassump irange obtain \( m2' \in v' \) \textbf{where step2:}
\[
E e' \in m1' = v' \land \langle V' ! i, m1' \rangle \rightarrow (\emptyset, m2') \land m2' = m1'(x := v')
\]
by (auto, metis MWLfSteps-det.assign)

with Vassump dequal step step1
\textbf{have dequalnext:} \( m1(x := v) =_d m1'(x := v') \)
by (simp add: d-equal-def, auto)

with step1 step1 trivialpair-in-USdB show \( \exists W' \in m2' \)
\[
\langle V' ! i, m1' \rangle \rightarrow (W', m2') \land (W, W') \in R0 \lor W =_d W'
\]
\land m2 =_d m2'
by auto

qed

hence \( R0 \subseteq \approx _d \)
by (rule Up-To-Technique)

with \textit{in}R0 show \( \text{thesis} \)
by auto

qed

end

3.3 Proofs of non-atomic compositionality results

We prove compositionality results for each non-atomic command of our example programming language (i.e. a command that is composed out of other commands): If the components are strongly secure and the expressions involved indistinguishable for an observer on security level \( d \), then the composed command is also strongly secure.

theory Language-Composition
imports Strongly-Secure-Skip-Assign
begin

context Strongly-Secure-Programs
begin

theorem Compositionality-Seq:
  assumes relatedpart1: \([c1] \approx_d [c1']\)
  assumes relatedpart2: \([c2] \approx_d [c2']\)
  shows \([c1;c2] \approx_d [c1';c2']\)

proof
  define \(R0\) where \(R0 = \{ (S1,S2). \exists c1 c1' c2 c2' W W'. S1 = (c1;c2)#W \land S2 = (c1';c2')#W' \land [c1] \approx_d [c1'] \land [c2] \approx_d [c2'] \land W \approx_d W' \}\)
  from relatedpart1 relatedpart2 trivialpair-in-USdB
  have \(inR0\): \((c1;c2],[c1';c2']\) \in R0
    by (simp add: R0-def)
  have \(uptoR0\): \(d\)-Bisimulation-Up-To-USdB \(d R0\)
    proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)
      from USdBsym
      show sym \(R0\)
        by (simp add: sym-def R0-def, fastforce)
    next
    fix S1 S2
    assume \(inR0\): \((S1,S2)\) \in R0
    with USdBlen show \(length S1 = length S2\)
      by (auto simp add: R0-def)
    next
    fix S1 S2 RS m1 m2 m1' i
    assume \(inR0\): \((S1,S2)\) \in R0
    assume irange: \(i < \text{length} S1\)
    assume S1step: \(\langle S1!,m1 \rangle \rightarrow \langle RS,m2 \rangle\)
    assume dequal: \(m1 =_d m1'\)
    from \(inR0\) obtain \(c1 c1' c2 c2' V V'\)
      where \(R0\)def: \(S1 = (c1;c2)#V \land S2 = (c1';c2')#V' \land [c1] \approx_d [c1'] \land [c2] \approx_d [c2'] \land V \approx_d V'\)
      by (simp add: R0-def, fastforce)
    with \(irange\) have \text{case-distinction1:}\n      \(i = 0 \lor (V \neq [] \land i \neq 0)\)
      by auto
    moreover
    have \text{case1:} \(i = 0 \Rightarrow \exists RS' m2'. \langle S2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land \langle (RS,RS')\) \in R0 \lor RS \approx_d RS' \rangle \land m2 =_d m2'\)
      proof
        assume \(i0: i = 0\)
        — get the two different sub-cases:
        with \(R0\)def \(S1\)step obtain \(c3 W\) \text{ where case-distinction:}\n        \(RS = [c2] \land \langle c1,m1 \rangle \rightarrow \langle [],m2 \rangle\)
∀ RS = (c3;c2)# W ∧ (c1,m1) → ⟨c3# W,m2⟩
by (simp, metis MWLfSteps-det-cases(3))

moreover
— Case 1: first command terminates

{ assume RSassump: RS = [c2] assume StepAssump: ⟨c1,m1⟩ → ⟨[],m2⟩

from USdBBeqlen[of []] StepAssump R0def’ USdB-Strong-d-Bisimulation dequal strongdB-aux[of d ≈ₜ i [c1] [c1 ′] m1 [] m2 m1 ′] i0 obtain W’ m2’ where c1c1’reason:
⟨c1,c2⟩ → ⟨[c2 ′],m2⟩ ∧ W’ = [] ∧ [] ≈ₜ W’ ∧ m2 =ₜ m2’
by fastforce

with c1c1’reason have conclpart:
⟨c1,c2⟩ → ⟨[c2 ′],m2⟩ ∧ m2 =ₜ m2’
by (simp add: MWLfSteps-det.seq1)

with RSassump R0def’ i0 have case1-concl:
∃ RS’ m2’, ⟨S2!i,m1 ′⟩ → ⟨RS’,m2⟩ ∧ ((RS,RS’) ∈ R0 ∨ RS ≈ₜ RS’) ∧ m2 =ₜ m2’
by (simp, rule-tac x=[c2 ′] in exI, auto)
}

moreover
— Case 2: first command does not terminate

{ assume RSassump: RS = (c3;c2)# W assume StepAssump: ⟨c1,m1⟩ → ⟨c3# W,m2⟩

from StepAssump R0def’ USdB-Strong-d-Bisimulation dequal strongdB-aux[of d ≈ₜ i [c1] [c1 ′] m1 c3# W m2 m1 ′] i0 obtain W” m2’ where c1c1’reason:
⟨c1,c2⟩ → ⟨W’,m2⟩ ∧ W” = [] ∧ (c3# W) ≈ₜ W” ∧ m2 =ₜ m2’
by fastforce

with USdBBeqlen[of c3# W V’”] obtain c3’ W’
where V’”reason:
V” = c3′# W’ ∧ length W = length W’
by (cases V”, force, force)

with c1c1’reason have conclpart1:
⟨c1,c2⟩ → ⟨(c3;c2)# W’,m2⟩ ∧ m2 =ₜ m2’
by (simp add: MWLfSteps-det.seq2)
ultimately show 

\[ \exists RS' m2'. \langle S2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land 
((RS,RS') \in R0 \lor RS \approx_d RS') \land m2 =_d m2' \]

by (auto simp add: R0-def)

qed

moreover

have case2-concl:
\[ \exists RS' m2'. \langle S2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land 
((RS,RS') \in R0 \lor RS \approx_d RS') \land m2 =_d m2' \]

by (blast)

proof

assume Vnonempt: \( V \neq [] \)

assume in0t0: \( i \neq 0 \)

with Vnonempt irange R0def' have i1range:
\( (i-Suc 0) < \text{ length } V \)

by simp

from in0t0 R0def' have S1seq: \( S1!i = V!(i-Suc 0) \)

by auto

from in0t0 R0def' have S2i: \( \exists RS' m2'. \langle S2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land 
((RS,RS') \in R0 \lor RS \approx_d RS') \land m2 =_d m2' \)

by (force)

qed

ultimately show \( \exists RS' m2'. \langle S2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land 
\]
((RS,RS') ∈ R0 ∨ RS ≈_d RS') ∧ m2 =_d m2'
by auto
qed

hence R0 ⊆ ≈_d
by (rule Up-To-Technique)

with inR0 show ?thesis
by auto

qed

theorem Compositionality-Fork:
fixes V::('exp,'id) MWLfCom list
assumes relatedmain: [c] ≈_d [c']
assumes relatedthreads: V ≈_d V'
shows [fork c V] ≈_d [fork c' V']
proof –
  define R0 where R0 = {(F1,F2). ∃c1 c1' W W'.
  F1 = [fork c1 W] ∧ F2 = [fork c1' W']
  ∧ [c1] ≈_d [c1'] ∧ W ≈_d W'}
from relatedmain relatedthreads
have inR0: ([fork c V],[fork c' V']) ∈ R0
  by (simp add: R0-def)

have uptoR0: d-Bisimulation-Up-To-USdB d R0
proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)
  from USdBsym show sym R0
  by (simp add: R0-def sym-def, auto)
next
fix F1 F2
assume inR0: (F1,F2) ∈ R0
with R0-def USdBeqlen show length F1 = length F2
by auto
next
fix F1 F2 c1V m1 m2 m1' i
assume inR0: (F1,F2) ∈ R0
assume irange: i < length F1
assume F1step: ⟨F1!,i,m1⟩ → ⟨c1V,m2⟩
assume dequal: m1 =_d m1'
from inR0 obtain c1 c1' V V'
  where R0def': F1 = [fork c1 V] ∧ F2 = [fork c1' V'] ∧
  [c1] ≈_d [c1'] ∧ V ≈_d V'
  by (simp add: R0-def, force)

from irange R0def' F1step
have rew: c1V = c1#V ∧ m2 = m1
  by (simp, metis MWLf-semantics.MWLfSteps-det-cases(6))
from irange R0def'MWLfSteps-det.fork have F2step:
  ⟨F2!i,m1⟩ → ⟨c1'!V',m1'⟩
by force

from R0def' USdB-comp-head-tail have conclpart:
  ((c1!V',c1'!V') ∈ R0 ∨ (c1!V') ≈d (c1'!V'))
by auto

with irange rew inR0 F1step dequal R0def' F2step
show ∃c1V' m2'. ⟨F2!i,m1⟩ → ⟨c1V',m1'⟩ ∧
((c1V',c1V') ∈ R0 ∨ c1V ≈d c1V') ∧ m2 =d m2'
by fastforce
qed

hence R0 ⊆≈d by (rule Up-To-Technique)
with inR0 show ?thesis
by auto

qed

theorem Compositionality-If:
  assumes dind-or-branchesrelated:
    b ≈d b' ∨ [c1] ≈d [c2] ∨ [c1'] ≈d [c2']
  assumes branch1related: [c1] ≈d [c1']
  assumes branch2related: [c2] ≈d [c2']
  shows [if b then c1 else c2 fi] ≈d [if b' then c1' else c2' fi]
proof –
  define R1 where R1 = {{II, I2}. ∃ c1 c1' c2 c2' b b'.
    II = [if b then c1 else c2 fi] ∧ I2 = [if b' then c1' else c2' fi] ∧
    [c1] ≈d [c1'] ∧ [c2] ≈d [c2'] ∧ b ≈d b'}

  define R2 where R2 = {{II, I2}. ∃ c1 c1' c2 c2' b b'.
    II = [if b then c1 else c2 fi] ∧ I2 = [if b' then c1' else c2' fi] ∧
    [c1] ≈d [c1'] ∧ [c2] ≈d [c2'] ∧
    ([c1] ≈d [c2] ∨ [c1'] ≈d [c2'])}

  define R0 where R0 = R1 ∪ R2

  from dind-or-branchesrelated branch1related branch2related
  have inR0: ([if b then c1 else c2 fi],[if b' then c1' else c2' fi]) ∈ R0
  by (simp add: R0-def R1-def R2-def)

  have uptoR0: d-Bisimulation-Up-To-USdB d R0
  proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)
    from USdBsym d-indistinguishable-sym
    have symR1: sym R1

  qed
by (simp add: sym-def R1-def, fastforce)
from USdBsym
have symR2: sym R2
  by (simp add: sym-def R2-def, fastforce)

from symR1 symR2 show sym R0
  by (simp add: sym-def R0-def)

next
fix I1 I2
assume inR0: (I1,I2) ∈ R0
thus length I1 = length I2
  by (simp add: R0-def R1-def R2-def, auto)

next
fix I1 I2 RS m1 m1' m2 i
assume inR0: (I1,I2) ∈ R0
assume irange: i < length I1
assume I1step: ⟨I1!i,m1⟩ → ⟨RS,m2⟩
assume dequal: m1 = d m1'
have inR1case: (I1,I2) ∈ R1
  ⟹ ∃ RS' m2'. ⟨I2!i,m1'⟩ → ⟨RS',m2'⟩ ∧
  
  ((RS,RS') ∈ R0 ∨ RS ≈_d RS') ∧ m2 = d m2'
  proof
  assume inR1: (I1,I2) ∈ R1
  
  then obtain c1 c1' c2 c2' b b' where R1def':
  
  I1 = [if b then c1 else c2 fi]
  ∧ I2 = [if b' then c1' else c2' fi] ∧
  [c1] ≈_d [c1'] ∧ [c2] ≈_d [c2'] ∧ b ≡_d b'
  by (simp add: R1-def, force)

  moreover
  — get the two different cases True and False from semantics:
  from irange R1def' I1step have case-distinction:
  
  RS = [c1] ∧ BMap (E b m1) = True ∨
  RS = [c2] ∧ BMap (E b m1) = False
  by (simp, metis MWLf-semantics.MWLfSteps-det-cases(4))

  moreover
  — Case 1: b evaluates to True
  
  { assume bevalT: BMap (E b m1) = True
  assume RSassump: RS = [c1] 
  from irange bevalT I1step R1def' RSassump have memeq:
  
  m2 = m1
  by (simp, metis MWLf-semantics.MWLfSteps-det-cases(4))

  from bevalT R1def' dequal have b'evalT:
  
  BMap (E b' m1') = True
  by (simp add: d-indistinguishable-def)
hence I2step-case1:
\((\text{if } b' \text{ then } c1' \text{ else } c2' f_i, m1') \rightarrow ([c1'], m1')\)
by (simp add: MWLSteps-det.simps iftrue)

with irange dequal RSassump memeq R1def'
have case1-concl:
\(\exists RS' m2'. \langle I2'i, m1' \rangle \rightarrow \langle RS', m2' \rangle \land\)
\((\langle RS, RS' \rangle \in R0 \lor RS \approx_d RS') \land m2 = d m2'\)
by auto

moreover
— Case 2: \(b\) evaluates to False

{ assume bevalF: BMap (E b m1) = False
assume RSassump: RS = [c2]
from irange bevalF I1step R1def' RSassump have memeq:
\(m1 = m2\)
by (simp, metis MWLf-semantics.MWLSteps-det-cases(4))

from bevalF R1def' dequal have b'evalF:
BMap (E b' m1') = False
by (simp add: d-indistinguishable-def)

hence I2step-case1:
\((\text{if } b' \text{ then } c1' \text{ else } c2' f_i, m1') \rightarrow ([c2'], m1')\)
by (simp add: MWLSteps-det.simps iffalse)

with irange dequal RSassump memeq R1def'
have case1-concl:
\(\exists RS' m2'. \langle I2'i, m1' \rangle \rightarrow \langle RS', m2' \rangle \land\)
\((\langle RS, RS' \rangle \in R0 \lor RS \approx_d RS') \land m2 = d m2'\)
by auto

ultimately show
\(\exists RS' m2'. \langle I2'i, m1' \rangle \rightarrow \langle RS', m2' \rangle \land\)
\((\langle RS, RS' \rangle \in R0 \lor RS \approx_d RS') \land m2 = d m2'\)
by auto

qed

have inR2case: \(\langle I1, I2 \rangle \in R2\)
\(\Rightarrow \exists RS' m2'. \langle I2'i, m1' \rangle \rightarrow \langle RS', m2' \rangle \land\)
\((\langle RS, RS' \rangle \in R0 \lor RS \approx_d RS') \land m2 = d m2'\)

proof —
assume inR2: \(\langle I1, I2 \rangle \in R2\)
then obtain \(c1 \ c1' \ c2 \ c2' \ b' \ b'\) where R2def' :
\(I1 = [\text{if } b \text{ then } c1 \text{ else } c2 f_i]\)
\(\land I2 = [\text{if } b' \text{ then } c1' \text{ else } c2' f_i]\)
\(\land \ [c1] \approx_d [c1'] \land [c2] \approx_d [c2'] \land\)
\[(c_1 \approx_d [c_2] \lor [c_1'] \approx_d [c_2'])\]
by (simp add: R2-def, force)

moreover
— get the two different cases for the result from semantics:
from irange R2def' Istep have case-distinction-left:
\[(RS = [c_1] \lor RS = [c_2]) \land m_2 = m_1\]
by (simp, metis MWLf-semantics.MWLfSteps-det-cases(4))

moreover
from irange R2def' dequal obtain RS' where I2step:
\[\langle I_2!i, m_1' \rangle \rightarrow \langle RS', m_1' \rangle\]
\[\land (RS' = [c_1'] \lor RS' = [c_2']) \land m_1 =_d m_1'\]
by (simp, metis MWLfSteps-det-iffalse MWLfSteps-det-iftrue)

moreover
from USdBtrans have \[[ [c_1] \approx_d [c_2]; [c_2] \approx_d [c_2'] ] \]
\[\implies [c_1] \approx_d [c_2']\]
by (unfold trans-def, blast)

moreover
from USdBtrans have \[[ [c_1] \approx_d [c_1']; [c_1'] \approx_d [c_2'] ] \]
\[\implies [c_1'] \approx_d [c_2']\]
by (unfold trans-def, blast)

moreover
from USdBsym have \[[c_1] \approx_d [c_2] \implies [c_2] \approx_d [c_1]\]
by (simp add: sym-def)

moreover
from USdBtrans have \[[ [c_2] \approx_d [c_1]; [c_1] \approx_d [c_1'] ] \]
\[\implies [c_2] \approx_d [c_1']\]
by (unfold trans-def, blast)

moreover
from USdBsym have \[[c_1'] \approx_d [c_2'] \implies [c_2'] \approx_d [c_1']\]
by (simp add: sym-def)

moreover
from USdBtrans have \[[ [c_2] \approx_d [c_2']; [c_2'] \approx_d [c_1'] ] \]
\[\implies [c_2'] \approx_d [c_1']\]
by (unfold trans-def, blast)

ultimately show

\[\exists RS' m_2'. \langle I_2!i, m_1' \rangle \rightarrow (RS', m_2') \land\]
\[(RS, RS') \in R_0 \lor RS \approx_d RS' \land m_2 =_d m_2'\]
by auto

qed

from inR0 inR1case inR2case show

\[\exists RS' m_2'. \langle I_2!i, m_1' \rangle \rightarrow (RS', m_2') \land\]
\[(RS, RS') \in R_0 \lor RS \approx_d RS' \land m_2 =_d m_2'\]
by (auto simp add: R0-def)

qed

hence \(R_0 \subseteq \approx_d\)
by (rule Up-To-Technique)
with inR0 show \(?thesis
by auto

qed

theorem Compositionality-While:
assumes dind: \( b \equiv_d b' \)
assumes bodyrelated: \( [c] \approx_d [c'] \)
shows \([\text{while } b \text{ do } c \text{ od}] \approx_d [\text{while } b' \text{ do } c'\text{ od}] \)
proof

define \( R1 \) where
\( R1 = \{ (S1,S2). \exists c1 c1' c2 c2' b b' W W'. \)
\( S1 = (c1;(\text{while } b \text{ do } c2 \text{ od}))#W \land \)
\( S2 = (c1';(\text{while } b' \text{ do } c2' \text{ od}))#W' \land \)
\( [c1] \approx_d [c1'] \land [c2] \approx_d [c2'] \land W \approx_d W' \land b \equiv_d b' \}

define \( R2 \) where
\( R2 = \{ (W1,W2). \exists c1 c1' b b'. \)
\( W1 = [\text{while } b \text{ do } c1 \text{ od}] \land \)
\( W2 = [\text{while } b' \text{ do } c1' \text{ od}] \land \)
\( [c1] \approx_d [c1'] \land b \equiv_d b' \}

define \( R0 \) where
\( R0 = R1 \cup R2 \)

from dind bodyrelated
have inR0: \([\text{while } b \text{ do } c \text{ od}],[\text{while } b' \text{ do } c' \text{ od}]\) \( \in R0 \)
by (simp add: R0-def R1-def R2-def)

have uptoR0: \( \text{d-Bisimulation-Up-To-USdB} \ d R0 \)
proof (simp add: d-Bisimulation-Up-To-USdB-def, auto)
  from USdBsym d-indistinguishable-sym have symR1: sym R1
  by (simp add: sym-def R1-def, fastforce)
  from USdBsym d-indistinguishable-sym have symR2: sym R2
  by (simp add: sym-def R2-def, fastforce)
  from symR1 symR2 show sym R0
  by (simp add: sym-def R0-def)

next
fix W1 W2
assume inR0: \((W1,W2) \in R0 \)
with USdBlen show length W1 = length W2
by (simp add: R0-def R1-def R2-def, force)

next
fix W1 W2 i m1 m1' RS m2
assume inR0: \((W1,W2) \in R0 \)
assume irange: \( i < \text{length } W1 \)
assume W1step: \( \langle W1!i,m1 \rangle \rightarrow \langle RS,m2 \rangle \)
assume dequal: \( m1 =_d m1' \)
from inR0 show \( \exists RS' m2'. \langle W2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land \)
\( ((RS,RS') \in R0 \lor RS \approx_d RS' \land m2 =_d m2' \)
proof (simp add: R0-def, auto)
  assume inR1: \((W1,W2) \in R1 \)
\end{proof}
then obtain \( c_1 c_1' c_2 c_2' b b' V V' \)

where \( R1\text{def}' : W_1 = (c_1; (\text{while } b \text{ do } c_2 \text{ od}))\#V \)
\( \land \ W_2 = (c_1'; (\text{while } b' \text{ do } c_2' \text{ od}))\#V' \)
\( \land [c_1] \approx_d [c_1'] \land [c_2] \approx_d [c_2'] \land V \approx_d V' \land b \equiv_d b' \)

by (simp add: R1-def, force)

with \( \text{irange have case-distinction}1: i = 0 \lor \)
\( (V \neq [] \land i \neq 0) \)

by auto

moreover

have \( \text{case}1: i = 0 \implies \)
\( \exists RS', m_2'. \langle W_2!i, m_1' \rangle \to \langle RS', m_2' \rangle \land \)
\( ((RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS') \land m_2 = d m_2' \)

proof

assume \( i0: i = 0 \)

— get the two different sub-cases:

with \( R1\text{def}' \ W1\text{step obtain} c_3 W \) where \( \text{case-distinction}:
\)
\( RS = [\text{while } b \text{ do } c_2 \text{ od}] \land \langle c_1, m_1 \rangle \to \langle [], m_2 \rangle \land \)
\( R = (c_3; (\text{while } b \text{ do } c_2 \text{ od}))\#W \land \langle c_1, m_1 \rangle \to \langle c_3\#W, m_2 \rangle \)

by (simp, metis MWLSteps-def-cases(3))

moreover

— Case 1: first command terminates

\[
\{
\text{assume RSassump: } RS = [\text{while } b \text{ do } c_2 \text{ od}]
\text{ assume StepAssump: } \langle c_1, m_1 \rangle \to \langle [], m_2 \rangle
\}
\]

from \( \text{USdB}\text{Beqlen}[\text{of } []] \) \( \text{StepAssump R1def}'
\)
\( \text{USdB-Strong-d-Bisimulation dequal}
\text{strongdB-aux}[\text{of } d \equiv_d i \]
\[
\langle c_1 \rangle \ [c_1'] m_1 \ [m_2 m_1'] \ i0 
\]

obtain \( W' m_2' \) where \( \text{c1c1' \ reason:}
\)
\( \langle c_1', m_1' \rangle \to \langle W', m_2' \rangle \land W' = [] \land \)
\( [] \approx_d W' \land m_2 = d m_2' \)

by fastforce

with \( \text{c1c1' \ reason have conclpart1:}
\)
\( \langle c_1', (\text{while } b' \text{ do } c_2' \text{ od}), m_1' \rangle \)
\( \to \langle [\text{while } b' \text{ do } c_2' \text{ od}], m_2' \rangle \land m_2 = d m_2' \)

by (simp add: MWLSteps-det-seq1)

from \( R1\text{def}' \) have \( \text{conclpart2:}
\)
\( ([\text{while } b \text{ do } c_2 \text{ od}],[\text{while } b' \text{ do } c_2' \text{ od}]) \in R2 \)

by (simp add: R2-def)

with \( \text{conclpart1 RSassump i0 R1def}'
\)

have \( \text{case1-concl:}
\)
\( \exists RS' m_2'. \langle W_2!i, m_1' \rangle \to \langle RS', m_2' \rangle \land \)
\[(RS,RS') \in R1 \lor (RS,RS') \in R2 \lor RS \approx_d RS')\]
\[\land m2 \approx_d m2'\]

by auto

\}

moreover

— Case 2: first command does not terminate

\{ 

assume RSassump: \(RS = (c3;(while \ b \ do \ c2 \ od))\#W\)

assume StepAssump: \((c1,m1) \rightarrow (c3\#W,m2)\)

from StepAssump R1def' USdB-Strong-d-Bisimulation dequal strongdB-aux[of d \approx_d i]
[\[c1\] [c1'] m1 c3\#W m2 m1'] i0

obtain \(V'' m2'\) where c1c1'reason:
\[\langle c1',m1' \rangle \rightarrow \langle V'',m2' \rangle\]
\[\land (c3\#W) \approx_d V'' \land m2 =_d m2'\]

by fastforce

with USdBBegin[of c3\#W V''] obtain c3' W'
where V''reason: \(V'' = c3'\#W'\)
by (cases V'', force, force)

with c1c1'reason have conclpart1:
\[\langle c1';(while \ b' \ do \ c2' \ od),m1' \rangle \rightarrow \langle (c3';(while \ b' \ do \ c2' \ od))\#W',m2' \rangle\]
\[\land m2 =_d m2'\]
by (simp add: MWLSteps-det.seq2)

from V''reason
c1c1'reason USdB-decomp-head-tail[of c3 W]
USdB-Strong-d-Bisimulation

have c3aWinUSDB:
\[\[c3\] \approx_d [c3'] \land W \approx_d W'\]
by blast

with R1def' have conclpart2:
\[\langle (c3';(while \ b' \ do \ c2' \ od))\#W',m2' \rangle \in R1\]
by (simp add: R1-def)

with i0 RSassump R1def' V''reason conclpart1
have case2-concl:
\[\exists RS' m2'. \langle W2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land \rangle\]
\[\langle RS,RS' \rangle \in R1 \lor \langle RS,RS' \rangle \in R2 \lor RS \approx_d RS')\]
\[\land m2 =_d m2'\]
by auto

\}

ultimately

show \[\exists RS' m2'. \langle W2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land \]
\[(RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS')\]
\[\land m2 = d \cdot m2'\]
by blast

qed

moreover

have case2: [ V ≠ []; i ≠ 0 ]

\[\Rightarrow \exists RS', m2'. (W2!i, m1) \rightarrow (RS', m2') \land
\[(RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS')\]
\[\land m2 = d \cdot m2'\]

proof –

assume Vnonempt: V ≠ []

assume inot0: i ≠ 0

with Vnonempt irange R1def’ have iirange: (i−Suc 0) < length V
by simp

from inot0 R1def’ have W1eq: W1!i = V!(i−Suc 0)
by auto

from inot0 R1def’ have W2!i = V′!(i−Suc 0)
by auto

with W1eq R1def’ W1step iirange dequal
USdB-Strong-d-Bisimulation
strongdB-aux[of d USdB d
i−Suc 0 V V' m1 RS m2 m1']

show \[\exists RS', m2'. (W2!i, m1) \rightarrow (RS', m2') \land
((RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS')\]
\[\land m2 = d \cdot m2'\]
by force

qed

ultimately show \[\exists RS', m2'. (W2!i, m1) \rightarrow (RS', m2') \land
((RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS')\]
\[\land m2 = d \cdot m2'\]
by auto

next

assume inR2: (W1, W2) ∈ R2

then obtain c1 c1' b b' where R2def':
W1 = [while b do c1 od] \land W2 = [while b' do c1' od] \land
[c1] ≈_d [c1'] \land b \equiv_d b'
by (auto simp add: R2-def)

— get the two different cases:

moreover

from irange R2def’ W1step have case-distinction:
RS = [c1; do b while b do c1 od] \land BMap (E b m1) = True \lor
RS = [] \land BMap (E b m1) = False
by (simp,metis MWLf-semantics.MWLSteps-det-cases(5))
moreover
— Case 1: \( b \) evaluates to True

\{
  \begin{align*}
  \text{assume } & \text{bevalT: } BMap (E \ b \ m1) \\
  \text{assume } & \text{RSassump: } RS = [c1;\{\text{while } b \text{ do } c1 \text{ od}\}] \\
  \text{from } & \text{irange bevalT W1step R2def' RSassump have } \text{memeq: } m2 = m1 \\
  & \text{by } (\text{simp,metis MWLf-semantics.MWLSteps-det-cases(5)}) \\
  \text{from } & \text{bevalT R2def' dequal have } b'\text{evalT: } BMap (E \ b' \ m1') \\
  & \text{by } (\text{simp add: d-indistinguishable-def}) \\
  \text{hence } & W2step-case1: \\
  & \langle \text{while } b' \text{ do } c1' \text{ od},m1' \rangle \\
  & \rightarrow \langle c1';\{\text{while } b' \text{ do } c1' \text{ od}\},m1' \rangle \\
  & \text{by } (\text{simp add: MWLfSteps-det.whiletrue}) \\
  \text{from } & \text{trivialpair-in-USdB R2def' have inWR2: } \\
  & ([c1;\{\text{while } b \text{ do } c1 \text{ od}\}], \\
  & [c1';\{\text{while } b' \text{ do } c1' \text{ od}\}]) \in R1 \\
  & \text{by } (\text{auto simp add: R1-def}) \\
  \text{with } & \text{irange dequal RSassump memeq W2step-case1 R2def'} \\
  \text{have case1-concl: } \\
  & \exists RS' m2'. \langle W2!i,m1' \rangle \rightarrow \langle RS',m2' \rangle \land \\
  & ((RS,RS') \in R1 \lor (RS,RS') \in R2 \lor RS \approx d RS') \\
  & \land m2 = d m2' \\
  & \text{by } \text{auto}
  \end{align*}
\}
moreover
— Case 2: \( b \) evaluates to False

\{
  \begin{align*}
  \text{assume } & \text{bevalF: } BMap (E \ b \ m1) = \text{False} \\
  \text{assume } & \text{RSassump: } RS = [] \\
  \text{from } & \text{irange bevalF W1step R2def' RSassump have } \text{memeq: } m2 = m1 \\
  & \text{by } (\text{simp,metis MWLf-semantics.MWLSteps-det-cases(5)}) \\
  \text{from } & \text{bevalF R2def' dequal have } b'\text{evalF: } BMap (E \ b' \ m1') = \text{False} \\
  & \text{by } (\text{simp add: d-indistinguishable-def}) \\
  \text{hence } & W2step-case2: \\
  & \langle \text{while } b' \text{ do } c1' \text{ od},m1' \rangle \rightarrow \langle [],m1' \rangle \\
  & \text{by } (\text{simp add: MWLfSteps-det.whilefalse}) \\
  \text{with } & \text{trivialpair-in-USdB irange dequal RSassump } \\
  & \text{memeq R2def'} \\
  \text{have case1-concl: }
  \end{align*}
\}
\[ \exists RS', m2'. \langle W2!i, m1' \rangle \rightarrow \langle RS', m2' \rangle \land \\
((RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS') \\
\land m2 =_d m2' \]
by force

ultimately

\textbf{show} \ \exists RS', m2'. \langle W2!i, m1' \rangle \rightarrow \langle RS', m2' \rangle \land \\
((RS, RS') \in R1 \lor (RS, RS') \in R2 \lor RS \approx_d RS') \\
\land m2 =_d m2' 
by auto

qed

\textbf{qed}

\textit{hence} \ \ R0 \subseteq \approx_d 
by (rule \ Up-To-Technique)

\textbf{with} \ \ \textit{inR0} \ \ \textbf{show} \ ?thesis 
by auto

qed

end

end

4 Security type system

4.1 Abstract security type system with soundness proof

We formalize an abstract version of the type system in [2] using locales [1]. Our formalization of the type system is abstract in the sense that the rules specify abstract semantic side conditions on the expressions within a command that satisfy for proving the soundness of the rules. That is, it can be instantiated with different syntactic approximations for these semantic side conditions in order to achieve a type system for a concrete language for Boolean and arithmetic expressions. Obtaining a soundness proof for such a concrete type system then boils down to proving that the concrete type system interprets the abstract type system.

We prove the soundness of the abstract type system by simply applying the compositionality results proven before.

theory \ Type-System
imports Language-Composition
begin

locale \ Type-System = 
SSP:\ : Strongly-Secure-Programs E BMap DA
for E :: ('exp, 'id, 'val) Evalfunction
and BMap :: 'val ⇒ bool
and DA :: ('id, 'd::order) DomainAssignment
+
fixes
AssignSideCondition :: 'id ⇒ 'exp ⇒ bool
and WhileSideCondition :: 'exp ⇒ bool
and IfSideCondition :: 'exp ⇒ ('exp, 'id) MWLfCom ⇒ bool
assumes semAssignSC: AssignSideCondition x e =⇒ e ≡ DA x e
and semWhileSC: WhileSideCondition e =⇒ ∀ d. e ≡ₚ d e
and semIfSC: IfSideCondition e c1 c2 =⇒ ∀ d. e ≡ₚ d e ∨ (c1) ≈ₚ d (c2)

begin

— Security typing rules for the language commands

inductive
ComSecTyping :: ('exp, 'id) MWLfCom ⇒ bool
(⊢ C -)
and ComSecTypingL :: ('exp, 'id) MWLfCom list ⇒ bool
(⊢ V -)
where

skip: ⊢ C skip |
Assign: [ AssignSideCondition x e ] ⊢ C x := e |
Fork: [ ⊢ C c; ⊢ V ] ⊢ C fork c V |
Seq: [ ⊢ C c1; ⊢ C c2 ] ⊢ C c1; c2 |
While: [ ⊢ C c; WhileSideCondition b ]

⇒ ⊢ C while b do c od |
If: [ ⊢ C c1; ⊢ C c2; IfSideCondition b c1 c2 ]

⇒ ⊢ C if b then c1 else c2 fi |
Parallel: [ ∀ i < length V. ⊢ C V!i ] ⇒ ⊢ V V

inductive-cases parallel-cases:

⊢ V V

— soundness proof of abstract type system

theorem ComSecTyping-single-is-sound:

⊢ C c =⇒ Strongly-Secure [c]

by (induct rule: ComSecTyping-ComSecTypingL.inducts(1))

[of - - Strongly-Secure],

auto simp add: Strongly-Secure-def,

metis Strongly-Secure-Skip,

metis Strongly-Secure-Assign semAssignSC,

metis Compositionality-Fork,

metis Compositionality-Seq,

metis Compositionality-While semWhileSC,

metis Compositionality-If semIfSC,

metis parallel-composition)

theorem ComSecTyping-list-is-sound:
⊢ \forall V \rightarrow \text{Strongly-Secure } V \\
\text{by (metis ComSecTyping-single-is-sound Strongly-Secure-def}} \\
\text{parallel-composition parallel-cases) }

end

end

4.2 Example language for Boolean and arithmetic expressions

As an example, we provide a simple example language for instantiating the parameter 'exp for the language for Boolean and arithmetic expressions.

theory Expr 
imports Types
begin

— type parameters:
— 'val: numbers, boolean constants...
— 'id: identifier names

type-synonym ('val) operation = ('val list ⇒ 'val)

datatype (dead 'id, dead 'val) Expr =
  Const 'val |
  Var 'id |
  Op 'val operation (('id, 'val) Expr) list

— defining a simple recursive evaluation function on this datatype
primrec ExprEval :: (('id, 'val) Expr, ('id, 'val) Evalfunction) and
ExprEvalL :: (('id, 'val) Expr) list ⇒ ('id, 'val) State ⇒ 'val list
where
  ExprEval (Const v) m = v |
  ExprEval (Var x) m = (m x) |
  ExprEval (Op f arglist) m = (f (ExprEvalL arglist m)) |
  ExprEvalL [] m = [] |
  ExprEvalL (e# V) m = (ExprEval e m)#(ExprEvalL V m)

end

4.3 Example interpretation of abstract security type system

Using the example instantiation of the language for Boolean and arithmetic expressions, we give an example instantiation of our abstract security type system, instantiating the parameter for domains 'd with a two-level security
lattice.

theory Domain-example
imports Expr
begin

— When interpreting, we have to instantiate the type for domains. As an example, we take a type containing 'low' and 'high' as domains.

datatype Dom = low | high

instantiation Dom :: order
begin

definition less-eq-Dom-def: \( d1 \leq d2 = (\text{if } d1 = d2 \text{ then True else (if } d1 = \text{ low then True else False}) \)

definition less-Dom-def: \( d1 < d2 = (\text{if } d1 = d2 \text{ then False else (if } d1 = \text{ low then True else False}) \)

instance proof
fix x y z :: Dom
show \((x < y) = (x \leq y \land \neg y \leq x)\)
  unfolding less-eq-Dom-def less-Dom-def by auto
show \(x \leq x\) unfolding less-eq-Dom-def by auto
show \([[x \leq y; y \leq z] \Rightarrow x \leq z\]
  unfolding less-eq-Dom-def by ((split if-split-asm)+, auto)
show \([[x \leq y; y \leq x] \Rightarrow x = y\]
  unfolding less-eq-Dom-def by ((split if-split-asm)+,
  auto, (split if-split-asm)+, auto)
qed

end

end

theory Type-System-example
imports Type-System Expr Domain-example
begin

— When interpreting, we have to instantiate the type for domains.
— As an example, we take a type containing 'low' and 'high' as domains.

consts DA :: (‘id,Dom) DomainAssignment
consts BMap :: ‘val ⇒ bool

35
abbreviation \(d\text{-indistinguishable}' :: ('id,'val) Expr \Rightarrow \text{Dom} \Rightarrow ('id,'val) Expr \Rightarrow \text{bool}

\begin{align*}
\text{where} \\
\ e1 \equiv_d e2 \\
\ \equiv \text{Strongly-Secure-Programs.d-indistinguishable} & \ \text{ExprEval DA d e1 e2}
\end{align*}

abbreviation relatedbyUSdB' :: (('id,'val) Expr, 'id) MWLfCom list \\
\Rightarrow \text{Dom} \Rightarrow (('id,'val) Expr, 'id) MWLfCom list \Rightarrow \text{bool} (\text{infixr} \approx 65)

\begin{align*}
\text{where} \\
\ V \approx_d V' \equiv (V',V) \in \text{Strong-Security.USdB} & \\
\ \text{(MWLf-semantics,MWLfSteps-det ExprEval BMap) DA d}
\end{align*}

— Security typing rules for expressions - will be part of a side condition

inductive

ExprSecTyping :: ('id, 'val) Expr \Rightarrow \text{Dom set} \Rightarrow \text{bool}

\begin{align*}
\lnot \cdot \cdot \cdot \cdot \\
\text{where} \\
\ \text{Consts:} \vdash \cdot \cdot \cdot \\
\ \text{Vars:} \vdash \cdot \cdot \cdot \\
\ \text{Ops:} \forall i < \text{length arglist}. \vdash \cdot \cdot \cdot \\
\ \Rightarrow \vdash \cdot \cdot \cdot
\end{align*}

definition \(\text{synAssignSC} :: 'id \Rightarrow ('id, 'val) Expr \Rightarrow \text{bool}

\begin{align*}
\text{where} \\
\ \text{synAssignSC x e} \equiv_\exists D. (\vdash e : D \land (\forall d \in D. (d \leq DA x)))
\end{align*}

definition \(\text{synWhileSC} :: ('id, 'val) Expr \Rightarrow \text{bool}

\begin{align*}
\text{where} \\
\ \text{synWhileSC e} \equiv_\exists D. (\vdash e : D \land (\forall d \in D. \forall d'. (d \leq d'))
\end{align*}

definition \(\text{synIfSC} :: ('id, 'val) Expr \Rightarrow (('id, 'val) Expr, 'id) MWLfCom

\begin{align*}
\Rightarrow (('id, 'val) Expr, 'id) MWLfCom \Rightarrow \text{bool}
\end{align*}

\begin{align*}
\text{where} \\
\ \text{synIfSC e c1 c2} \equiv \\
\ \forall d. (\vdash (e \equiv_d e \Rightarrow [c1] \approx_d [c2])
\end{align*}

lemma \(\text{ExprTypable-with-smallerD-implies-d-indistinguishable}:

\begin{align*}
[\vdash e : D; \forall d' \in D'. (d' \leq d)] \Rightarrow e \equiv_d e
\end{align*}

proof (induct rule: ExprSecTyping.induct,

\begin{align*}
\text{smp-all addl: Strongly-Secure-Programs.d-indistinguishable-def}
\text{Strong-Security.d-equal-def, auto)}
\end{align*}

fix \(d l \text{ and arglist:}((\text{id}', \text{val}) \text{Expr}) \text{ list} \text{ and} \ f:::\text{val list} \Rightarrow \text{val}

\begin{align*}
\text{and} m1::(\text{id}', \text{val}) \text{State} \text{ and} \ m2::(\text{id}', \text{val}) \text{State}
\end{align*}

assume main: \(\forall i < \text{length arglist}. \vdash \cdot \cdot \cdot \\
\ (\forall d' \in (dll)i). d' \leq d) \Rightarrow \\
\ (\forall m m'. \forall x. DA x \leq d \Rightarrow m x = m' x)

\begin{align*}
\Rightarrow \text{ExprEval (arglist!!i) m = ExprEval (arglist!!i) m')}
\end{align*}

assume smaller: \(\forall D. (\exists i < \text{length arglist}. D = (dll)i)

\begin{align*}
\Rightarrow (\forall d' \in D. d' \leq d)
\end{align*}

36
assume $eqstate: \forall x. DA x \leq d \rightarrow m1 x = m2 x$

from smaller have irangesubst:
\[ \forall i < \text{length arglist}. \forall d' \in (dl!i). d' \leq d \]
by auto

with $eqstate$ main have
\[ \forall i < \text{length arglist}. \text{ExprEval (arglist}!i) m1 = \text{ExprEval (arglist}!i) m2 \]
by force

hence substmap: \(\text{ExprEvalL arglist m1} = \text{ExprEvalL arglist m2}\)
by (induct arglist, auto, force)

show $f (\text{ExprEvalL arglist m1}) = f (\text{ExprEvalL arglist m2})$
by (subst substmap, auto)
qed

interpretation Type-System-example: Type-System ExprEval BMap DA
synAssignSC synWhileSC synIfSC
end

References
