A Formalization of Assumptions and Guarantees for Compositional Noninterference

Sylvia Grewe, Heiko Mantel, Daniel Schoepe

March 19, 2025

Abstract

Research in information-flow security aims at developing methods to identify undesired information leaks within programs from private (high) sources to public (low) sinks. For a concurrent system, it is desirable to have compositional analysis methods that allow for analyzing each thread independently and that nevertheless guarantee that the parallel composition of successfully analyzed threads satisfies a global security guarantee. However, such a compositional analysis should not be overly pessimistic about what an environment might do with shared resources. Otherwise, the analysis will reject many intuitively secure programs.

The paper "Assumptions and Guarantees for Compositional Noninterference" by Mantel et. al. [MSS11] presents one solution for this problem: an approach for compositionally reasoning about noninterference in concurrent programs via rely-guarantee-style reasoning. We present an Isabelle/HOL formalization of the concepts and proofs of this approach.

The formalization includes the following parts:

- Notion of SIFUM-security and preliminary concepts: `Preliminaries.thy`, `Security.thy`
- Compositionality proof: `Compositionality.thy`
- Example language: `Language.thy`
- Type system for ensuring SIFUM-security and soundness proof: `TypeSystem.thy`
- Type system for ensuring sound use of modes and soundness proof: `LocallySoundUseOfModes.thy`

Contents

1 Preliminaries

theory *Preliminaries*
imports *Main*
begin

unbundle *lattice-syntax*

Possible modes for variables:

datatype *Mode* = *AsmNoRead* | *AsmNoWrite* | *GuarNoRead* | *GuarNoWrite*

We consider a two-element security lattice:

datatype *Sec* = *High* | *Low*

notation
less-eq (**infix** ‹⊑› *50*) **and**
less (**infix** ‹⊏› *50*)

Sec forms a (complete) lattice:

instantiation *Sec* :: *complete-lattice*
begin

definition *top-Sec-def*: $\top$ = *High*
definition *sup-Sec-def*: *d1* ⊔ *d2* = (*if* (*d1* = *High* ∨ *d2* = *High*) *then High else Low*)
definition *inf-Sec-def*: *d1* ⊓ *d2* = (*if* (*d1* = *Low* ∨ *d2* = *Low*) *then Low else High*)

definition *bot-Sec-def*: $\bot = Low$
definition *less-eq-Sec-def*: $d1 \leq d2 = (d1 = d2 \lor d1 = Low)$
definition *less-Sec-def*: $d1 < d2 = (d1 = Low \land d2 = High)$
definition *Sup-Sec-def*: $\bigsqcup S = (if\ (High \in S)\ then\ High\ else\ Low)$
definition *Inf-Sec-def*: $\bigsqcap S = (if\ (Low \in S)\ then\ Low\ else\ High)$

instance
⟨*proof*⟩
end

Memories are mappings from variables to values

type-synonym $('var, 'val)\ Mem = 'var \Rightarrow 'val$

A mode state maps modes to the set of variables for which the given mode is set.

type-synonym $'var\ Mds = Mode \Rightarrow 'var\ set$

Local configurations:

type-synonym $('com, 'var, 'val)\ LocalConf = ('com \times 'var\ Mds) \times ('var, 'val)\ Mem$

Global configurations:

type-synonym $('com, 'var, 'val)\ GlobalConf = ('com \times 'var\ Mds)\ list \times ('var, 'val)\ Mem$

A locale to fix various parametric components in Mantel et. al, and assumptions about them:

locale *sifum-security* =
fixes $dma :: 'Var \Rightarrow Sec$
fixes $stop :: 'Com$
fixes $eval :: ('Com, 'Var, 'Val)\ LocalConf\ rel$
fixes $some\text{-}val :: 'Val$
fixes $some\text{-}val' :: 'Val$
assumes *stop-no-eval*: $\neg ((((stop, mds), mem), ((c', mds'), mem')) \in eval)$
assumes *deterministic*: $[\![(lc, lc') \in eval; (lc, lc'') \in eval]\!] \Longrightarrow lc' = lc''$
assumes *finite-memory*: $finite\ \{(x::'Var).\ True\}$
assumes *different-values*: $some\text{-}val \neq some\text{-}val'$

end

2 Definition of the SIFUM-Security Property

theory *Security*
imports *Main Preliminaries*
begin

context *sifum-security* **begin**

2.1 Evaluation of Concurrent Programs

abbreviation *eval-abv* :: ('*Com*, '*Var*, '*Val*) *LocalConf* ⇒ (-, -, -) *LocalConf* ⇒ *bool*
(**infixl** ‹⇝› *70*)
where
$x \rightsquigarrow y \equiv (x, y) \in eval$

abbreviation *conf-abv* :: '*Com* ⇒ '*Var Mds* ⇒ ('*Var*, '*Val*) *Mem* ⇒ (-,-,-) *LocalConf*
(‹⟨-, -, -⟩› [*0*, *0*, *0*] *1000*)
where
⟨ *c*, *mds*, *mem* ⟩ ≡ ((*c*, *mds*), *mem*)

inductive-set *meval* :: (-,-,-) *GlobalConf rel*
and *meval-abv* :: - ⇒ - ⇒ *bool* (**infixl** ‹→› *70*)
where
conf → *conf'* ≡ (*conf*, *conf'*) ∈ *meval* |
meval-intro [*iff*]: ⟦ (*cms* ! *n*, *mem*) ⇝ (*cm'*, *mem'*); *n* < *length cms* ⟧ ⟹
((*cms*, *mem*), (*cms* [*n* := *cm'*], *mem'*)) ∈ *meval*

inductive-cases *meval-elim* [*elim*!]: ((*cms*, *mem*), (*cms'*, *mem'*)) ∈ *meval*

abbreviation *meval-clos* :: - ⇒ - ⇒ *bool* (**infixl** ‹$\rightarrow^*$› *70*)
where
conf $\rightarrow^*$ *conf'* ≡ (*conf*, *conf'*) ∈ $meval^*$

fun *lc-set-var* :: (-, -, -) *LocalConf* ⇒ '*Var* ⇒ '*Val* ⇒ (-, -, -) *LocalConf*
where
lc-set-var (*c*, *mem*) *x* *v* = (*c*, *mem* (*x* := *v*))

fun *meval-k* :: *nat* ⇒ ('*Com*, '*Var*, '*Val*) *GlobalConf* ⇒ (-, -, -) *GlobalConf* ⇒ *bool*
where
meval-k 0 c c' = (*c* = *c'*) |
meval-k (*Suc n*) *c c'* = (∃ *c''*. *meval-k n c c''* ∧ *c''* → *c'*)

abbreviation *meval-k-abv* :: *nat* ⇒ (-, -, -) *GlobalConf* ⇒ (-, -, -) *GlobalConf* ⇒ *bool*
(‹- $\rightarrow_1$ -› [*100*, *100*] *80*)
where
gc $\rightarrow_k$ *gc'* ≡ *meval-k k gc gc'*

2.2 Low-equivalence and Strong Low Bisimulations

definition *low-eq* :: ('*Var*, '*Val*) *Mem* ⇒ (-, -) *Mem* ⇒ *bool* (**infixl** ‹$=^l$› *80*)
where

$mem_1 =^l mem_2 \equiv (\forall\ x.\ dma\ x = Low \longrightarrow mem_1\ x = mem_2\ x)$

definition *low-mds-eq* :: *'Var Mds* ⇒ (*'Var*, *'Val*) *Mem* ⇒ (-, -) *Mem* ⇒ *bool*
(‹- =ı^l -› [*100*, *100*] *80*)
where
$(mem_1 =_{mds}{}^l\ mem_2) \equiv (\forall\ x.\ dma\ x = Low \wedge x \notin mds\ AsmNoRead \longrightarrow mem_1\ x = mem_2\ x)$

definition mds_s :: *'Var Mds* **where**
$mds_s\ x = \{\}$

lemma [*simp*]: $mem =^l mem' \Longrightarrow mem =_{mds}{}^l\ mem'$
⟨*proof*⟩

lemma [*simp*]: $(\forall\ mds.\ mem =_{mds}{}^l\ mem') \Longrightarrow mem =^l mem'$
⟨*proof*⟩

definition *closed-glob-consistent* :: ((*'Com*, *'Var*, *'Val*) *LocalConf*) *rel* ⇒ *bool*
where
closed-glob-consistent $\mathcal{R}$ =
$(\forall\ c_1\ mds\ mem_1\ c_2\ mem_2.\ (\langle\ c_1,\ mds,\ mem_1\ \rangle, \langle\ c_2,\ mds,\ mem_2\ \rangle) \in \mathcal{R} \longrightarrow$
$(\forall\ x.\ ((dma\ x = High \wedge x \notin mds\ AsmNoWrite) \longrightarrow$
$(\forall\ v_1\ v_2.\ (\langle\ c_1,\ mds,\ mem_1\ (x := v_1)\ \rangle, \langle\ c_2,\ mds,\ mem_2\ (x := v_2)\ \rangle) \in \mathcal{R})) \wedge$
$((dma\ x = Low \wedge x \notin mds\ AsmNoWrite) \longrightarrow$
$(\forall\ v.\ (\langle\ c_1,\ mds,\ mem_1\ (x := v)\ \rangle, \langle\ c_2,\ mds,\ mem_2\ (x := v)\ \rangle) \in \mathcal{R}))))$

definition *strong-low-bisim-mm* :: ((*'Com*, *'Var*, *'Val*) *LocalConf*) *rel* ⇒ *bool*
where
strong-low-bisim-mm $\mathcal{R} \equiv$
sym $\mathcal{R} \wedge$
closed-glob-consistent $\mathcal{R} \wedge$
$(\forall\ c_1\ mds\ mem_1\ c_2\ mem_2.\ (\langle\ c_1,\ mds,\ mem_1\ \rangle, \langle\ c_2,\ mds,\ mem_2\ \rangle) \in \mathcal{R} \longrightarrow$
$(mem_1 =_{mds}{}^l\ mem_2) \wedge$
$(\forall\ c_1'\ mds'\ mem_1'.\ \langle\ c_1,\ mds,\ mem_1\ \rangle \leadsto \langle\ c_1',\ mds',\ mem_1'\ \rangle \longrightarrow$
$(\exists\ c_2'\ mem_2'.\ \langle\ c_2,\ mds,\ mem_2\ \rangle \leadsto \langle\ c_2',\ mds',\ mem_2'\ \rangle \wedge$
$(\langle\ c_1',\ mds',\ mem_1'\ \rangle, \langle\ c_2',\ mds',\ mem_2'\ \rangle) \in \mathcal{R})))$

inductive-set *mm-equiv* :: ((*'Com*, *'Var*, *'Val*) *LocalConf*) *rel*
and *mm-equiv-abv* :: (*'Com*, *'Var*, *'Val*) *LocalConf* ⇒
(*'Com*, *'Var*, *'Val*) *LocalConf* ⇒ *bool* (**infix** ‹≈› *60*)
where
mm-equiv-abv $x\ y \equiv (x,\ y) \in mm\text{-}equiv$ |
mm-equiv-intro [*iff*]: $[\![\ strong\text{-}low\text{-}bisim\text{-}mm\ \mathcal{R}\ ;\ (lc_1,\ lc_2) \in \mathcal{R}\ ]\!] \Longrightarrow (lc_1,\ lc_2) \in mm\text{-}equiv$

inductive-cases *mm-equiv-elim* [*elim*]: $\langle\ c_1,\ mds,\ mem_1\ \rangle \approx \langle\ c_2,\ mds,\ mem_2\ \rangle$

definition *low-indistinguishable* :: *'Var Mds* ⇒ *'Com* ⇒ *'Com* ⇒ *bool*
(‹- ∼1 -› [*100*, *100*] *80*)
where $c_1 \sim_{mds} c_2 = (\forall\ mem_1\ mem_2.\ mem_1 =_{mds}{}^l\ mem_2 \longrightarrow$
$\langle\ c_1,\ mds,\ mem_1\ \rangle \approx \langle\ c_2,\ mds,\ mem_2\ \rangle)$

2.3 SIFUM-Security

definition *com-sifum-secure* :: *'Com* ⇒ *bool*
where *com-sifum-secure* $c = c \sim_{mds_s} c$

definition *add-initial-modes* :: *'Com list* ⇒ (*'Com* × *'Var Mds*) *list*
where *add-initial-modes cmds* = *zip cmds* (*replicate* (*length cmds*) mds_s)

definition *no-assumptions-on-termination* :: *'Com list* ⇒ *bool*
where *no-assumptions-on-termination cmds* =
(∀ *mem mem' cms'*.
(*add-initial-modes cmds*, *mem*) $\rightarrow^*$ (*cms'*, *mem'*) ∧
list-all (λ *c*. *c* = *stop*) (*map fst cms'*) ⟶
(∀ *mds'* ∈ *set* (*map snd cms'*). *mds' AsmNoRead* = {} ∧ *mds' AsmNoWrite* = {}))

definition *prog-sifum-secure* :: *'Com list* ⇒ *bool*
where *prog-sifum-secure cmds* =
(*no-assumptions-on-termination cmds* ∧
$(\forall\ mem_1\ mem_2.\ mem_1 =^l mem_2 \longrightarrow$
$(\forall\ k\ cms_1'\ mem_1'.$
$(add\text{-}initial\text{-}modes\ cmds,\ mem_1) \rightarrow_k (cms_1',\ mem_1') \longrightarrow$
$(\exists\ cms_2'\ mem_2'.\ (add\text{-}initial\text{-}modes\ cmds,\ mem_2) \rightarrow_k (cms_2',\ mem_2') \wedge$
$map\ snd\ cms_1' = map\ snd\ cms_2' \wedge$
$length\ cms_2' = length\ cms_1' \wedge$
$(\forall\ x.\ dma\ x = Low \wedge (\forall\ i < length\ cms_1'.$
$x \notin snd\ (cms_1'\ !\ i)\ AsmNoRead) \longrightarrow mem_1'\ x = mem_2'\ x)))))$

2.4 Sound Mode Use

definition *doesnt-read* :: *'Com* ⇒ *'Var* ⇒ *bool*
where
doesnt-read c x = (∀ *mds mem c' mds' mem'*.
$\langle\ c,\ mds,\ mem\ \rangle \leadsto \langle\ c',\ mds',\ mem'\ \rangle \longrightarrow$
$((\forall\ v.\ \langle\ c,\ mds,\ mem\ (x := v)\ \rangle \leadsto \langle\ c',\ mds',\ mem'\ (x := v)\ \rangle) \vee$
$(\forall\ v.\ \langle\ c,\ mds,\ mem\ (x := v)\ \rangle \leadsto \langle\ c',\ mds',\ mem'\ \rangle)))$

definition *doesnt-modify* :: *'Com* ⇒ *'Var* ⇒ *bool*
where
doesnt-modify c x = (∀ *mds mem c' mds' mem'*. $(\langle\ c,\ mds,\ mem\ \rangle \leadsto \langle\ c',\ mds',\ mem'\ \rangle) \longrightarrow$

mem x = *mem′ x*)

inductive-set *loc-reach* :: (*′Com*, *′Var*, *′Val*) *LocalConf* ⇒ (*′Com*, *′Var*, *′Val*) *LocalConf set*
  for *lc* :: (-, -, -) *LocalConf*
  where
  refl : ⟨*fst* (*fst lc*), *snd* (*fst lc*), *snd lc*⟩ ∈ *loc-reach lc* |
  step : ⟦ ⟨*c′*, *mds′*, *mem′*⟩ ∈ *loc-reach lc*;
      ⟨*c′*, *mds′*, *mem′*⟩ ⇝ ⟨*c″*, *mds″*, *mem″*⟩ ⟧ ⟹
    ⟨*c″*, *mds″*, *mem″*⟩ ∈ *loc-reach lc* |
  mem-diff : ⟦ ⟨ *c′*, *mds′*, *mem′* ⟩ ∈ *loc-reach lc*;
      (∀ *x* ∈ *mds′ AsmNoWrite*. *mem′ x* = *mem″ x*) ⟧ ⟹
    ⟨ *c′*, *mds′*, *mem″* ⟩ ∈ *loc-reach lc*

definition *locally-sound-mode-use* :: (-, -, -) *LocalConf* ⇒ *bool*
  where
  locally-sound-mode-use lc =
  (∀ *c′ mds′ mem′*. ⟨ *c′*, *mds′*, *mem′* ⟩ ∈ *loc-reach lc* ⟶
   (∀ *x*. (*x* ∈ *mds′ GuarNoRead* ⟶ *doesnt-read c′ x*) ∧
     (*x* ∈ *mds′ GuarNoWrite* ⟶ *doesnt-modify c′ x*)))

definition *compatible-modes* :: (*′Var Mds*) *list* ⇒ *bool*
  where
  compatible-modes mdss = (∀ (*i* :: *nat*) *x*. *i* < *length mdss* ⟶
   (*x* ∈ (*mdss* ! *i*) *AsmNoRead* ⟶
    (∀ *j* < *length mdss*. *j* ≠ *i* ⟶ *x* ∈ (*mdss* ! *j*) *GuarNoRead*)) ∧
   (*x* ∈ (*mdss* ! *i*) *AsmNoWrite* ⟶
    (∀ *j* < *length mdss*. *j* ≠ *i* ⟶ *x* ∈ (*mdss* ! *j*) *GuarNoWrite*)))

definition *reachable-mode-states* :: (*′Com*, *′Var*, *′Val*) *GlobalConf* ⇒ ((*′Var Mds*) *list*) *set*
  where *reachable-mode-states gc* =
  {*mdss*. (∃ *cms′ mem′*. *gc* $\rightarrow^*$ (*cms′*, *mem′*) ∧ *map snd cms′* = *mdss*)}

definition *globally-sound-mode-use* :: (*′Com*, *′Var*, *′Val*) *GlobalConf* ⇒ *bool*
  where *globally-sound-mode-use gc* =
  (∀ *mdss*. *mdss* ∈ *reachable-mode-states gc* ⟶ *compatible-modes mdss*)

primrec *sound-mode-use* :: (-, -, -) *GlobalConf* ⇒ *bool*
  where
  sound-mode-use (*cms*, *mem*) =
   (*list-all* (λ *cm*. *locally-sound-mode-use* (*cm*, *mem*)) *cms* ∧
    globally-sound-mode-use (*cms*, *mem*))

lemma *mm-equiv-sym*:
  assumes *equivalent*: $\langle c_1, mds_1, mem_1\rangle \approx \langle c_2, mds_2, mem_2\rangle$
  shows $\langle c_2, mds_2, mem_2\rangle \approx \langle c_1, mds_1, mem_1\rangle$

⟨*proof*⟩

lemma *low-indistinguishable-sym*: $lc \sim_{mds} lc' \Longrightarrow lc' \sim_{mds} lc$
⟨*proof*⟩

lemma *mm-equiv-glob-consistent*: *closed-glob-consistent mm-equiv*
⟨*proof*⟩

lemma *mm-equiv-strong-low-bisim*: *strong-low-bisim-mm mm-equiv*
⟨*proof*⟩

end

end

3 Compositionality Proof for SIFUM-Security Property

theory *Compositionality*
imports *Main Security*
begin

context *sifum-security*
begin

definition *differing-vars* :: ('*Var*, '*Val*) *Mem* ⇒ (-, -) *Mem* ⇒ '*Var set*
where
differing-vars mem_1 mem_2 = $\{x.\ mem_1\ x \neq mem_2\ x\}$

definition *differing-vars-lists* :: ('*Var*, '*Val*) *Mem* ⇒ (-, -) *Mem* ⇒
((-, -) *Mem* × (-, -) *Mem*) *list* ⇒ *nat* ⇒ '*Var set*
where
differing-vars-lists mem_1 mem_2 *mems* *i* =
(*differing-vars* mem_1 (*fst* (*mems* ! *i*)) ∪ *differing-vars* mem_2 (*snd* (*mems* ! *i*)))

lemma *differing-finite*: *finite* (*differing-vars* mem_1 mem_2)
⟨*proof*⟩

lemma *differing-lists-finite*: *finite* (*differing-vars-lists* mem_1 mem_2 *mems* *i*)
⟨*proof*⟩

definition *subst* :: ('*a* ⇀ '*b*) ⇒ ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*)
where
subst f mem = (λ *x*. *case f x of*
None ⇒ *mem x* |
Some v ⇒ *v*)

abbreviation *subst-abv* :: $('a \Rightarrow 'b) \Rightarrow ('a \rightharpoonup 'b) \Rightarrow ('a \Rightarrow 'b)$ (‹- [↦-]› [*900*, *0*] *1000*)
where
$f\ [\mapsto \sigma] \equiv subst\ \sigma\ f$

lemma *subst-not-in-dom* : $[\![\ x \notin dom\ \sigma\ ]\!] \Longrightarrow mem\ [\mapsto \sigma]\ x = mem\ x$
⟨*proof*⟩

fun *makes-compatible* ::
$('Com, 'Var, 'Val)\ GlobalConf \Rightarrow$
$('Com, 'Var, 'Val)\ GlobalConf \Rightarrow$
$((\text{-}, \text{-})\ Mem \times (\text{-}, \text{-})\ Mem)\ list \Rightarrow$
bool
where
$makes\text{-}compatible\ (cms_1, mem_1)\ (cms_2, mem_2)\ mems =$
$(length\ cms_1 = length\ cms_2 \wedge length\ cms_1 = length\ mems \wedge$
$(\forall\ i.\ i < length\ cms_1 \longrightarrow$
$(\forall\ \sigma.\ dom\ \sigma = differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i \longrightarrow$
$(cms_1\ !\ i, (fst\ (mems\ !\ i))\ [\mapsto \sigma]) \approx (cms_2\ !\ i, (snd\ (mems\ !\ i))\ [\mapsto \sigma])) \wedge$
$(\forall\ x.\ (mem_1\ x = mem_2\ x \vee dma\ x = High) \longrightarrow$
$x \notin differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i)) \wedge$
$((length\ cms_1 = 0 \wedge mem_1 =^l mem_2) \vee (\forall\ x.\ \exists\ i.\ i < length\ cms_1 \wedge$
$x \notin differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i)))$

lemma *makes-compatible-intro* [*intro*]:
$[\![\ length\ cms_1 = length\ cms_2 \wedge length\ cms_1 = length\ mems;$
$(\bigwedge\ i\ \sigma.\ [\![\ i < length\ cms_1;\ dom\ \sigma = differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i\ ]\!]$
$\Longrightarrow$
$(cms_1\ !\ i, (fst\ (mems\ !\ i))\ [\mapsto \sigma]) \approx (cms_2\ !\ i, (snd\ (mems\ !\ i))\ [\mapsto \sigma]));$
$(\bigwedge\ i\ x.\ [\![\ i < length\ cms_1;\ mem_1\ x = mem_2\ x \vee dma\ x = High\ ]\!] \Longrightarrow$
$x \notin differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i);$
$(length\ cms_1 = 0 \wedge mem_1 =^l mem_2) \vee$
$(\forall\ x.\ \exists\ i.\ i < length\ cms_1 \wedge x \notin differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i)\ ]\!]$
$\Longrightarrow$
$makes\text{-}compatible\ (cms_1, mem_1)\ (cms_2, mem_2)\ mems$
⟨*proof*⟩

lemma *compat-low*:
$[\![\ makes\text{-}compatible\ (cms_1, mem_1)\ (cms_2, mem_2)\ mems;$
$i < length\ cms_1;$
$x \in differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i\ ]\!] \Longrightarrow dma\ x = Low$
⟨*proof*⟩

lemma *compat-different*:
$[\![\ makes\text{-}compatible\ (cms_1, mem_1)\ (cms_2, mem_2)\ mems;$
$i < length\ cms_1;$

$x \in$ *differing-vars-lists* mem_1 mem_2 *mems* i ⟧ $\Longrightarrow$ mem_1 $x \neq mem_2$ $x \wedge$ *dma* x = *Low*
⟨*proof*⟩

lemma *sound-modes-no-read* :
⟦ *sound-mode-use* (*cms*, *mem*); $x \in$ (*map snd cms* ! i) *GuarNoRead*; $i <$ *length cms* ⟧ $\Longrightarrow$
doesnt-read (*fst* (*cms* ! i)) x
⟨*proof*⟩

lemma *compat-different-vars*:
⟦ *fst* (*mems* ! i) x = *snd* (*mems* ! i) x;
$x \notin$ *differing-vars-lists* mem_1 mem_2 *mems* i ⟧ $\Longrightarrow$
mem_1 x = mem_2 x
⟨*proof*⟩

lemma *differing-vars-subst* [*rule-format*]:
assumes *domσ*: *dom* $\sigma \supseteq$ *differing-vars* mem_1 mem_2
shows mem_1 [$\mapsto \sigma$] = mem_2 [$\mapsto \sigma$]
⟨*proof*⟩

lemma *mm-equiv-low-eq*:
⟦ ⟨ c_1, *mds*, mem_1 ⟩ ≈ ⟨ c_2, *mds*, mem_2 ⟩ ⟧ $\Longrightarrow$ $mem_1 =_{mds}{}^{l}$ mem_2
⟨*proof*⟩

lemma *globally-sound-modes-compatible*:
⟦ *globally-sound-mode-use* (*cms*, *mem*) ⟧ $\Longrightarrow$ *compatible-modes* (*map snd cms*)
⟨*proof*⟩

lemma *compatible-different-no-read* :
assumes *sound-modes*: *sound-mode-use* (cms_1, mem_1)
sound-mode-use (cms_2, mem_2)
assumes *compat*: *makes-compatible* (cms_1, mem_1) (cms_2, mem_2) *mems*
assumes *modes-eq*: *map snd* cms_1 = *map snd* cms_2
assumes *ile*: $i <$ *length* cms_1
assumes *x*: $x \in$ *differing-vars-lists* mem_1 mem_2 *mems* i
shows *doesnt-read* (*fst* (cms_1 ! i)) $x \wedge$ *doesnt-read* (*fst* (cms_2 ! i)) x
⟨*proof*⟩

definition *func-le* :: ($'a \rightharpoonup 'b$) ⇒ ($'a \rightharpoonup 'b$) ⇒ *bool* (**infixl** ‹⪯› *60*)
where $f \preceq g = (\forall\ x \in dom\ f.\ f\ x = g\ x)$

fun *change-respecting* ::
($'Com$, $'Var$, $'Val$) *LocalConf* ⇒
($'Com$, $'Var$, $'Val$) *LocalConf* ⇒
$'Var$ *set* ⇒
(($'Var \rightharpoonup 'Val$) ⇒
($'Var \rightharpoonup 'Val$)) ⇒ *bool*

where *change-respecting* (cms, mem) (cms', mem') X g =
$((cms, mem) \rightsquigarrow (cms', mem') \wedge$
$(\forall\ \sigma.\ dom\ \sigma = X \longrightarrow g\ \sigma \preceq \sigma) \wedge$
$(\forall\ \sigma\ \sigma'.\ dom\ \sigma = X \wedge dom\ \sigma' = X \longrightarrow dom\ (g\ \sigma) = dom\ (g\ \sigma')) \wedge$
$(\forall\ \sigma.\ dom\ \sigma = X \longrightarrow (cms, mem\ [\mapsto \sigma]) \rightsquigarrow (cms', mem'\ [\mapsto g\ \sigma])))$

lemma *change-respecting-dom-unique*:
$[\![$ *change-respecting* $\langle c, mds, mem\rangle$ $\langle c', mds', mem'\rangle$ X g $]\!] \Longrightarrow$
$\exists\ d.\ \forall\ f.\ dom\ f = X \longrightarrow d = dom\ (g\ f)$
⟨*proof*⟩

lemma *func-le-restrict*: $[\![\ f \preceq g;\ X \subseteq dom\ f\ ]\!] \Longrightarrow f\ |^{\backprime}\ X \preceq g$
⟨*proof*⟩

definition *to-partial* :: $('a \Rightarrow 'b) \Rightarrow ('a \rightharpoonup 'b)$
where *to-partial* $f = (\lambda\ x.\ Some\ (f\ x))$

lemma *func-le-dom*: $f \preceq g \Longrightarrow dom\ f \subseteq dom\ g$
⟨*proof*⟩

lemma *doesnt-read-mutually-exclusive*:
assumes *noread*: *doesnt-read* c x
assumes *eval*: $\langle c, mds, mem\rangle \rightsquigarrow \langle c', mds', mem'\rangle$
assumes *unchanged*: $\forall\ v.\ \langle c, mds, mem\ (x := v)\rangle \rightsquigarrow \langle c', mds', mem'\ (x := v)\rangle$
shows $\neg\ (\forall\ v.\ \langle c, mds, mem\ (x := v)\rangle \rightsquigarrow \langle c', mds', mem'\rangle)$
⟨*proof*⟩

lemma *doesnt-read-mutually-exclusive'*:
assumes *noread*: *doesnt-read* c x
assumes *eval*: $\langle c, mds, mem\rangle \rightsquigarrow \langle c', mds', mem'\rangle$
assumes *overwrite*: $\forall\ v.\ \langle c, mds, mem\ (x := v)\rangle \rightsquigarrow \langle c', mds', mem'\rangle$
shows $\neg\ (\forall\ v.\ \langle c, mds, mem\ (x := v)\rangle \rightsquigarrow \langle c', mds', mem'\ (x := v)\rangle)$
⟨*proof*⟩

lemma *change-respecting-dom*:
assumes *cr*: *change-respecting* (cms, mem) (cms', mem') X g
assumes *domσ*: $dom\ \sigma = X$
shows $dom\ (g\ \sigma) \subseteq X$
⟨*proof*⟩

lemma *change-respecting-intro* [*iff*]:
$[\![\ \langle\ c, mds, mem\ \rangle \rightsquigarrow \langle\ c', mds', mem'\ \rangle;$
$\bigwedge f.\ dom\ f = X \Longrightarrow$
$g\ f \preceq f\ \wedge$
$(\forall\ f'.\ dom\ f' = X \longrightarrow dom\ (g\ f) = dom\ (g\ f'))\ \wedge$
$(\langle\ c, mds, mem\ [\mapsto f]\ \rangle \rightsquigarrow \langle\ c', mds', mem'\ [\mapsto g\ f]\ \rangle)\ ]\!]$
$\Longrightarrow$ *change-respecting* $\langle c, mds, mem\rangle$ $\langle c', mds', mem'\rangle$ X g
⟨*proof*⟩

lemma *conjI3*: $[\![A;\ B;\ C]\!] \Longrightarrow A \wedge B \wedge C$
⟨*proof*⟩

lemma *noread-exists-change-respecting*:
assumes *fin*: *finite* $(X :: 'Var\ set)$
assumes *eval*: $\langle\ c,\ mds,\ mem\ \rangle \rightsquigarrow \langle\ c',\ mds',\ mem'\ \rangle$
assumes *noread*: $\forall\ x \in X.\ doesnt\text{-}read\ c\ x$
shows $\exists\ (g :: ('Var \rightharpoonup 'Val) \Rightarrow ('Var \rightharpoonup 'Val)).\ \ change\text{-}respecting\ \langle c,\ mds,\ mem\rangle\ \langle c',\ mds',\ mem'\rangle\ X\ g$
⟨*proof*⟩

lemma *differing-vars-neg*: $x \notin differing\text{-}vars\text{-}lists\ mem1\ mem2\ mems\ i \Longrightarrow$
$(fst\ (mems\ !\ i)\ x = mem1\ x \wedge snd\ (mems\ !\ i)\ x = mem2\ x)$
⟨*proof*⟩

lemma *differing-vars-neg-intro*:
$[\![mem_1\ x = fst\ (mems\ !\ i)\ x;$
$mem_2\ x = snd\ (mems\ !\ i)\ x]\!] \Longrightarrow x \notin differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i$
⟨*proof*⟩

lemma *differing-vars-elim* [*elim*]:
$x \in differing\text{-}vars\text{-}lists\ mem_1\ mem_2\ mems\ i \Longrightarrow$
$(fst\ (mems\ !\ i)\ x \neq mem_1\ x) \vee (snd\ (mems\ !\ i)\ x \neq mem_2\ x)$
⟨*proof*⟩

lemma *subst-overrides*: $dom\ \sigma = dom\ \tau \Longrightarrow mem\ [\mapsto \tau]\ [\mapsto \sigma] = mem\ [\mapsto \sigma]$
⟨*proof*⟩

lemma *dom-restrict-total*: $dom\ (to\text{-}partial\ f\ |`\ X) = X$
⟨*proof*⟩

lemma *update-nth-eq*:
$[\![xs = ys;\ n < length\ xs]\!] \Longrightarrow xs = ys\ [n := xs\ !\ n]$
⟨*proof*⟩

This property is obvious, so an unreadable apply-style proof is acceptable here:

lemma *mm-equiv-step*:
assumes *bisim*: $(cms_1,\ mem_1) \approx (cms_2,\ mem_2)$
assumes *modes-eq*: $snd\ cms_1 = snd\ cms_2$
assumes *step*: $(cms_1,\ mem_1) \rightsquigarrow (cms_1',\ mem_1')$
shows $\exists\ c_2'\ mem_2'.\ (cms_2,\ mem_2) \rightsquigarrow \langle\ c_2',\ snd\ cms_1',\ mem_2'\ \rangle \wedge$
$(cms_1',\ mem_1') \approx \langle\ c_2',\ snd\ cms_1',\ mem_2'\ \rangle$
⟨*proof*⟩

lemma *change-respecting-doesnt-modify*:
assumes *cr*: $change\text{-}respecting\ (cms,\ mem)\ (cms',\ mem')\ X\ g$
assumes *eval*: $(cms,\ mem) \rightsquigarrow (cms',\ mem')$

assumes *domf*: *dom f* = *X*
assumes *x-in-dom*: $x \in$ *dom* (*g f*)
assumes *noread*: *doesnt-read* (*fst cms*) *x*
shows *mem x* = *mem′ x*
⟨*proof*⟩

type-synonym ($'var$, $'val$) *adaptation* = $'var \rightharpoonup ('val \times 'val)$

definition *apply-adaptation* ::
bool ⇒ ($'Var$, $'Val$) *Mem* ⇒ ($'Var$, $'Val$) *adaptation* ⇒ ($'Var$, $'Val$) *Mem*
where *apply-adaptation first mem A* =
(λ *x*. *case* (*A x*) *of*
Some (v_1, v_2) ⇒ *if first then* v_1 *else* v_2
| *None* ⇒ *mem x*)

abbreviation *apply-adaptation*$_1$ ::
($'Var$, $'Val$) *Mem* ⇒ ($'Var$, $'Val$) *adaptation* ⇒ ($'Var$, $'Val$) *Mem*
(‹- [‖$_1$ -]› [*900*, *0*] *1000*)
where *mem* [‖$_1$ *A*] ≡ *apply-adaptation True mem A*

abbreviation *apply-adaptation*$_2$ ::
($'Var$, $'Val$) *Mem* ⇒ ($'Var$, $'Val$) *adaptation* ⇒ ($'Var$, $'Val$) *Mem*
(‹- [‖$_2$ -]› [*900*, *0*] *1000*)
where *mem* [‖$_2$ *A*] ≡ *apply-adaptation False mem A*

definition *restrict-total* :: ($'a \Rightarrow 'b$) ⇒ $'a$ *set* ⇒ $'a \rightharpoonup 'b$
where *restrict-total f A* = *to-partial f* |‘ *A*

lemma *differing-empty-eq*:
⟦ *differing-vars mem mem′* = {} ⟧ ⟹ *mem* = *mem′*
⟨*proof*⟩

definition *globally-consistent-var* :: ($'Var$, $'Val$) *adaptation* ⇒ $'Var$ *Mds* ⇒ $'Var$ ⇒ *bool*
where *globally-consistent-var A mds x* ≡
(*case A x of*
Some (*v*, *v′*) ⇒ $x \notin$ *mds AsmNoWrite* ∧ (*dma x* = *Low* ⟶ *v* = *v′*)
| *None* ⇒ *True*)

definition *globally-consistent* :: ($'Var$, $'Val$) *adaptation* ⇒ $'Var$ *Mds* ⇒ *bool*
where *globally-consistent A mds* ≡ *finite* (*dom A*) ∧
(∀ $x \in$ *dom A*. *globally-consistent-var A mds x*)

definition *gc2* :: ($'Var$, $'Val$) *adaptation* ⇒ $'Var$ *Mds* ⇒ *bool*
where *gc2 A mds* = (∀ $x \in$ *dom A*. *globally-consistent-var A mds x*)

lemma *globally-consistent-dom*:

$[\![$ *globally-consistent A mds*; $X \subseteq$ *dom A* $]\!] \Longrightarrow$ *globally-consistent* $(A \mid` X)$ *mds*
⟨*proof*⟩

lemma *globally-consistent-writable*:
$[\![$ $x \in$ *dom A*; *globally-consistent A mds* $]\!] \Longrightarrow x \notin$ *mds AsmNoWrite*
⟨*proof*⟩

lemma *globally-consistent-loweq*:
assumes *globally-consistent*: *globally-consistent A mds*
assumes *some*: $A\ x = Some\ (v,\ v')$
assumes *low*: $dma\ x = Low$
shows $v = v'$
⟨*proof*⟩

lemma *globally-consistent-adapt-bisim*:
assumes *bisim*: $\langle c_1,\ mds,\ mem_1\rangle \approx \langle c_2,\ mds,\ mem_2\rangle$
assumes *globally-consistent*: *globally-consistent A mds*
shows $\langle c_1,\ mds,\ mem_1\ [\|_1\ A]\rangle \approx \langle c_2,\ mds,\ mem_2\ [\|_2\ A]\rangle$
⟨*proof*⟩

lemma *makes-compatible-invariant*:
assumes *sound-modes*: *sound-mode-use* $(cms_1,\ mem_1)$
sound-mode-use $(cms_2,\ mem_2)$
assumes *compat*: *makes-compatible* $(cms_1,\ mem_1)\ (cms_2,\ mem_2)$ *mems*
assumes *modes-eq*: $map\ snd\ cms_1 = map\ snd\ cms_2$
assumes *eval*: $(cms_1,\ mem_1) \to (cms_1',\ mem_1')$
obtains $cms_2'\ mem_2'\ mems'$ **where**
$map\ snd\ cms_1' = map\ snd\ cms_2' \wedge$
$(cms_2,\ mem_2) \to (cms_2',\ mem_2') \wedge$
makes-compatible $(cms_1',\ mem_1')\ (cms_2',\ mem_2')\ mems'$
⟨*proof*⟩

The Isar proof language provides a readable way of specifying assumptions while also giving them names for subsequent usage.

lemma *compat-low-eq*:
assumes *compat*: *makes-compatible* $(cms_1,\ mem_1)\ (cms_2,\ mem_2)$ *mems*
assumes *modes-eq*: $map\ snd\ cms_1 = map\ snd\ cms_2$
assumes *x-low*: $dma\ x = Low$
assumes *x-readable*: $\forall\ i < length\ cms_1.\ x \notin snd\ (cms_1\ !\ i)\ AsmNoRead$
shows $mem_1\ x = mem_2\ x$
⟨*proof*⟩

lemma *loc-reach-subset*:
assumes *eval*: $\langle c,\ mds,\ mem\rangle \leadsto \langle c',\ mds',\ mem'\rangle$
shows *loc-reach* $\langle c',\ mds',\ mem'\rangle \subseteq$ *loc-reach* $\langle c,\ mds,\ mem\rangle$
⟨*proof*⟩

lemma *locally-sound-modes-invariant*:

assumes *sound-modes*: *locally-sound-mode-use* ⟨*c*, *mds*, *mem*⟩
assumes *eval*: ⟨*c*, *mds*, *mem*⟩ $\leadsto$ ⟨*c*′, *mds*′, *mem*′⟩
shows *locally-sound-mode-use* ⟨*c*′, *mds*′, *mem*′⟩
⟨*proof*⟩

lemma *reachable-modes-subset*:
assumes *eval*: (*cms*, *mem*) → (*cms*′, *mem*′)
shows *reachable-mode-states* (*cms*′, *mem*′) ⊆ *reachable-mode-states* (*cms*, *mem*)
⟨*proof*⟩

lemma *globally-sound-modes-invariant*:
assumes *globally-sound*: *globally-sound-mode-use* (*cms*, *mem*)
assumes *eval*: (*cms*, *mem*) → (*cms*′, *mem*′)
shows *globally-sound-mode-use* (*cms*′, *mem*′)
⟨*proof*⟩

lemma *loc-reach-mem-diff-subset*:
assumes *mem-diff*: $\forall\ x.\ x \in mds\ AsmNoWrite \longrightarrow mem_1\ x = mem_2\ x$
shows ⟨*c*′, *mds*′, *mem*′⟩ ∈ *loc-reach* ⟨*c*, *mds*, mem_1⟩ ⟹ ⟨*c*′, *mds*′, *mem*′⟩ ∈ *loc-reach* ⟨*c*, *mds*, mem_2⟩
⟨*proof*⟩

lemma *loc-reach-mem-diff-eq*:
assumes *mem-diff*: ∀ *x*. *x* ∈ *mds AsmNoWrite* ⟶ *mem*′ *x* = *mem x*
shows *loc-reach* ⟨*c*, *mds*, *mem*⟩ = *loc-reach* ⟨*c*, *mds*, *mem*′⟩
⟨*proof*⟩

lemma *sound-modes-invariant*:
assumes *sound-modes*: *sound-mode-use* (*cms*, *mem*)
assumes *eval*: (*cms*, *mem*) → (*cms*′, *mem*′)
shows *sound-mode-use* (*cms*′, *mem*′)
⟨*proof*⟩

lemma *makes-compatible-eval-k*:
assumes *compat*: *makes-compatible* (cms_1, mem_1) (cms_2, mem_2) *mems*
assumes *modes-eq*: *map snd* cms_1 = *map snd* cms_2
assumes *sound-modes*: *sound-mode-use* (cms_1, mem_1) *sound-mode-use* (cms_2, mem_2)
assumes *eval*: (cms_1, mem_1) $\rightarrow_k$ ($cms_1{}'$, $mem_1{}'$)
shows ∃ $cms_2{}'$ $mem_2{}'$ *mems*′. *sound-mode-use* ($cms_1{}'$, $mem_1{}'$) ∧
sound-mode-use ($cms_2{}'$, $mem_2{}'$) ∧
map snd $cms_1{}'$ = *map snd* $cms_2{}'$ ∧
(cms_2, mem_2) $\rightarrow_k$ ($cms_2{}'$, $mem_2{}'$) ∧
makes-compatible ($cms_1{}'$, $mem_1{}'$) ($cms_2{}'$, $mem_2{}'$) *mems*′
⟨*proof*⟩

lemma *differing-vars-initially-empty*:
$i < n \Longrightarrow x \notin$ *differing-vars-lists* mem_1 mem_2 (*zip* (*replicate n* mem_1) (*replicate n* mem_2)) *i*

⟨*proof*⟩

lemma *compatible-refl*:
assumes *coms-secure*: *list-all com-sifum-secure cmds*
assumes *low-eq*: $mem_1 =^l mem_2$
shows *makes-compatible* (*add-initial-modes cmds*, mem_1)
(*add-initial-modes cmds*, mem_2)
(*replicate* (*length cmds*) (mem_1, mem_2))
⟨*proof*⟩

theorem *sifum-compositionality*:
assumes *com-secure*: *list-all com-sifum-secure cmds*
assumes *no-assms*: *no-assumptions-on-termination cmds*
assumes *sound-modes*: ∀ *mem*. *sound-mode-use* (*add-initial-modes cmds*, *mem*)
shows *prog-sifum-secure cmds*
⟨*proof*⟩

end

end

4 Language for Instantiating the SIFUM-Security Property

theory *Language*
imports *Main Preliminaries*
begin

4.1 Syntax

datatype *'var ModeUpd* = *Acq 'var Mode* (**infix** ‹$+=_m$› *75*)
| *Rel 'var Mode* (**infix** ‹$-=_m$› *75*)

datatype (*'var*, *'aexp*, *'bexp*) *Stmt* = *Assign 'var 'aexp* (**infix** ‹←› *130*)
| *Skip*
| *ModeDecl* (*'var*, *'aexp*, *'bexp*) *Stmt 'var ModeUpd* (‹-@[-]› [*0*, *0*] *150*)
| *Seq* (*'var*, *'aexp*, *'bexp*) *Stmt* (*'var*, *'aexp*, *'bexp*) *Stmt* (**infixr** ‹;;› *150*)
| *If 'bexp* (*'var*, *'aexp*, *'bexp*) *Stmt* (*'var*, *'aexp*, *'bexp*) *Stmt*
| *While 'bexp* (*'var*, *'aexp*, *'bexp*) *Stmt*
| *Stop*

type-synonym (*'var*, *'aexp*, *'bexp*) *EvalCxt* = (*'var*, *'aexp*, *'bexp*) *Stmt list*

locale *sifum-lang* =
fixes $eval_A$ :: (*'Var*, *'Val*) *Mem* ⇒ *'AExp* ⇒ *'Val*
fixes $eval_B$ :: (*'Var*, *'Val*) *Mem* ⇒ *'BExp* ⇒ *bool*
fixes *aexp-vars* :: *'AExp* ⇒ *'Var set*
fixes *bexp-vars* :: *'BExp* ⇒ *'Var set*

fixes *dma* :: *'Var* ⇒ *Sec*
assumes *Var-finite* : *finite* {(*x* :: *'Var*). *True*}
assumes *eval-vars-det*$_A$: ⟦ ∀ *x* ∈ *aexp-vars e*. *mem*$_1$ *x* = *mem*$_2$ *x* ⟧ ⟹ *eval*$_A$ *mem*$_1$ *e* = *eval*$_A$ *mem*$_2$ *e*
assumes *eval-vars-det*$_B$: ⟦ ∀ *x* ∈ *bexp-vars b*. *mem*$_1$ *x* = *mem*$_2$ *x* ⟧ ⟹ *eval*$_B$ *mem*$_1$ *b* = *eval*$_B$ *mem*$_2$ *b*

context *sifum-lang*
begin

notation (*latex* **output**)
Seq (‹-; -› *60*)

notation (*Rule* **output**)
Seq (‹- ; -› *60*)

notation (*Rule* **output**)
If (‹*if* - *then* - *else* - *fi*› *50*)

notation (*Rule* **output**)
While (‹*while* - *do* - *done*›)

abbreviation *conf*$_w$*-abv* :: (*'Var*, *'AExp*, *'BExp*) *Stmt* ⇒
'Var Mds ⇒ (*'Var*, *'Val*) *Mem* ⇒ (-,-,-) *LocalConf*
(‹⟨-, -, -⟩$_w$› [*0*, *120*, *120*] *100*)
where
⟨ *c*, *mds*, *mem* ⟩$_w$ ≡ ((*c*, *mds*), *mem*)

4.2 Semantics

primrec *update-modes* :: *'Var ModeUpd* ⇒ *'Var Mds* ⇒ *'Var Mds*
where
update-modes (*Acq x m*) *mds* = *mds* (*m* := *insert x* (*mds m*)) |
update-modes (*Rel x m*) *mds* = *mds* (*m* := {*y*. *y* ∈ *mds m* ∧ *y* ≠ *x*})

fun *updated-var* :: *'Var ModeUpd* ⇒ *'Var*
where
updated-var (*Acq x* -) = *x* |
updated-var (*Rel x* -) = *x*

fun *updated-mode* :: *'Var ModeUpd* ⇒ *Mode*
where
updated-mode (*Acq* - *m*) = *m* |
updated-mode (*Rel* - *m*) = *m*

inductive-set *eval*$_w$*-simple* :: ((*'Var*, *'AExp*, *'BExp*) *Stmt* × (*'Var*, *'Val*) *Mem*) *rel*

and $eval_w$*-simple-abv* :: (('Var, 'AExp, 'BExp) Stmt × ('Var, 'Val) Mem) ⇒
('Var, 'AExp, 'BExp) Stmt × ('Var, 'Val) Mem ⇒ bool
(**infixr** ‹$\rightsquigarrow_s$› *60*)
where
$c \rightsquigarrow_s c' \equiv (c, c') \in eval_w\text{-}simple$ |
assign: $((x \leftarrow e, mem), (Stop, mem\ (x := eval_A\ mem\ e))) \in eval_w\text{-}simple$ |
skip: $((Skip, mem), (Stop, mem)) \in eval_w\text{-}simple$ |
seq-stop: $((Seq\ Stop\ c, mem), (c, mem)) \in eval_w\text{-}simple$ |
if-true: $[\![\ eval_B\ mem\ b\ ]\!] \Longrightarrow ((If\ b\ t\ e, mem), (t, mem)) \in eval_w\text{-}simple$ |
if-false: $[\![\ \neg\ eval_B\ mem\ b\ ]\!] \Longrightarrow ((If\ b\ t\ e, mem), (e, mem)) \in eval_w\text{-}simple$ |
while: $((While\ b\ c, mem), (If\ b\ (c\ ;;\ While\ b\ c)\ Stop, mem)) \in eval_w\text{-}simple$

primrec *cxt-to-stmt* :: ('Var, 'AExp, 'BExp) EvalCxt ⇒ ('Var, 'AExp, 'BExp) Stmt
⇒ ('Var, 'AExp, 'BExp) Stmt
where
cxt-to-stmt [] c = c |
cxt-to-stmt (c # cs) c' = Seq c' (*cxt-to-stmt* cs c)

inductive-set $eval_w$ :: (('Var, 'AExp, 'BExp) Stmt, 'Var, 'Val) LocalConf rel
and $eval_w$*-abv* :: (('Var, 'AExp, 'BExp) Stmt, 'Var, 'Val) LocalConf ⇒
(('Var, 'AExp, 'BExp) Stmt, 'Var, 'Val) LocalConf ⇒ bool
(**infixr** ‹$\rightsquigarrow_w$› *60*)
where
$c \rightsquigarrow_w c' \equiv (c, c') \in eval_w$ |
unannotated: $[\![\ (c, mem) \rightsquigarrow_s (c', mem')\ ]\!]$
$\Longrightarrow (\langle cxt\text{-}to\text{-}stmt\ E\ c, mds, mem\rangle_w, \langle cxt\text{-}to\text{-}stmt\ E\ c', mds, mem'\rangle_w) \in eval_w$ |
seq: $[\![\ \langle c_1, mds, mem\rangle_w \rightsquigarrow_w \langle c_1', mds', mem'\rangle_w\ ]\!] \Longrightarrow (\langle (c_1\ ;;\ c_2), mds, mem\rangle_w, \langle (c_1'\ ;;\ c_2), mds', mem'\rangle_w) \in eval_w$ |
decl: $[\![\ \langle c, update\text{-}modes\ mu\ mds, mem\rangle_w \rightsquigarrow_w \langle c', mds', mem'\rangle_w\ ]\!] \Longrightarrow$
$(\langle cxt\text{-}to\text{-}stmt\ E\ (ModeDecl\ c\ mu), mds, mem\rangle_w, \langle cxt\text{-}to\text{-}stmt\ E\ c', mds', mem'\rangle_w) \in eval_w$

4.3 Semantic Properties

The following lemmas simplify working with evaluation contexts in the soundness proofs for the type system(s).

inductive-cases *eval-elim*: $(((c, mds), mem), ((c', mds'), mem')) \in eval_w$
inductive-cases *stop-no-eval'* [*elim*]: $((Stop, mem), (c', mem')) \in eval_w\text{-}simple$
inductive-cases *assign-elim'* [*elim*]: $((x \leftarrow e, mem), (c', mem')) \in eval_w\text{-}simple$
inductive-cases *skip-elim'* [*elim*]: $(Skip, mem) \rightsquigarrow_s (c', mem')$

lemma *cxt-inv*:
$[\![\ cxt\text{-}to\text{-}stmt\ E\ c = c'\ ;\ \bigwedge p\ q.\ c' \neq Seq\ p\ q\ ]\!] \Longrightarrow E = [] \wedge c' = c$
⟨*proof*⟩

lemma *cxt-inv-assign*:
$[\![$ *cxt-to-stmt E c* $= x \leftarrow e$ $]\!] \Longrightarrow c = x \leftarrow e \wedge E = []$
⟨*proof*⟩

lemma *cxt-inv-skip*:
$[\![$ *cxt-to-stmt E c* = *Skip* $]\!] \Longrightarrow c =$ *Skip* $\wedge E = []$
⟨*proof*⟩

lemma *cxt-inv-stop*:
cxt-to-stmt E c = *Stop* $\Longrightarrow c =$ *Stop* $\wedge E = []$
⟨*proof*⟩

lemma *cxt-inv-if*:
cxt-to-stmt E c = *If e p q* $\Longrightarrow c =$ *If e p q* $\wedge E = []$
⟨*proof*⟩

lemma *cxt-inv-while*:
cxt-to-stmt E c = *While e p* $\Longrightarrow c =$ *While e p* $\wedge E = []$
⟨*proof*⟩

lemma *skip-elim* [*elim*]:
$\langle Skip, mds, mem\rangle_w \leadsto_w \langle c', mds', mem'\rangle_w \Longrightarrow c' = Stop \wedge mds = mds' \wedge mem = mem'$
⟨*proof*⟩

lemma *assign-elim* [*elim*]:
$\langle x \leftarrow e, mds, mem\rangle_w \leadsto_w \langle c', mds', mem'\rangle_w \Longrightarrow c' = Stop \wedge mds = mds' \wedge mem' = mem\ (x := eval_A\ mem\ e)$
⟨*proof*⟩

inductive-cases *if-elim'* [*elim*!]: $(If\ b\ p\ q,\ mem) \leadsto_s (c',\ mem')$

lemma *if-elim* [*elim*]:
$\bigwedge P.$
$[\![\langle If\ b\ p\ q, mds, mem\rangle_w \leadsto_w \langle c', mds', mem'\rangle_w ;$
$[\![c' = p; mem' = mem ; mds' = mds ; eval_B\ mem\ b]\!] \Longrightarrow P ;$
$[\![c' = q; mem' = mem ; mds' = mds ; \neg\ eval_B\ mem\ b]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

inductive-cases *while-elim'* [*elim*!]: $(While\ e\ c,\ mem) \leadsto_s (c',\ mem')$

lemma *while-elim* [*elim*]:
$[\![\langle While\ e\ c, mds, mem\rangle_w \leadsto_w \langle c', mds', mem'\rangle_w]\!] \Longrightarrow c' = If\ e\ (c\ ;;\ While\ e\ c)\ Stop \wedge mds' = mds \wedge mem' = mem$
⟨*proof*⟩

inductive-cases *upd-elim'* [*elim*]: $(c@[upd],\ mem) \leadsto_s (c',\ mem')$

lemma *upd-elim* [*elim*]:

$\langle c@[upd],\ mds,\ mem\rangle_w \rightsquigarrow_w \langle c',\ mds',\ mem'\rangle_w \Longrightarrow \langle c,\ update\text{-}modes\ upd\ mds,\ mem\rangle_w \rightsquigarrow_w \langle c',\ mds',\ mem'\rangle_w$
⟨*proof*⟩

lemma *cxt-seq-elim* [*elim*]:
$c_1\ ;;\ c_2 = cxt\text{-}to\text{-}stmt\ E\ c \Longrightarrow (E = [] \wedge c = c_1\ ;;\ c_2) \vee (\exists\ c'\ cs.\ E = c' \# cs \wedge c = c_1 \wedge c_2 = cxt\text{-}to\text{-}stmt\ cs\ c')$
⟨*proof*⟩

inductive-cases *seq-elim′* [*elim*]: $(c_1\ ;;\ c_2,\ mem) \rightsquigarrow_s (c',\ mem')$

lemma *stop-no-eval*: $\neg (\langle Stop,\ mds,\ mem\rangle_w \rightsquigarrow_w \langle c',\ mds',\ mem'\rangle_w)$
⟨*proof*⟩

lemma *seq-stop-elim* [*elim*]:
$\langle Stop\ ;;\ c,\ mds,\ mem\rangle_w \rightsquigarrow_w \langle c',\ mds',\ mem'\rangle_w \Longrightarrow c' = c \wedge mds' = mds \wedge mem' = mem$
⟨*proof*⟩

lemma *cxt-stmt-seq*:
$c\ ;;\ cxt\text{-}to\text{-}stmt\ E\ c' = cxt\text{-}to\text{-}stmt\ (c' \# E)\ c$
⟨*proof*⟩

lemma *seq-elim* [*elim*]:
$[\![\ \langle c_1\ ;;\ c_2,\ mds,\ mem\rangle_w \rightsquigarrow_w \langle c',\ mds',\ mem'\rangle_w\ ;\ c_1 \neq Stop\ ]\!] \Longrightarrow$
$(\exists\ c_1'.\ \langle c_1,\ mds,\ mem\rangle_w \rightsquigarrow_w \langle c_1',\ mds',\ mem'\rangle_w \wedge c' = c_1'\ ;;\ c_2)$
⟨*proof*⟩

lemma *stop-cxt*: $Stop = cxt\text{-}to\text{-}stmt\ E\ c \Longrightarrow c = Stop$
⟨*proof*⟩

end

end

5 Type System for Ensuring SIFUM-Security of Commands

theory *TypeSystem*
imports *Main Preliminaries Security Language Compositionality*
begin

5.1 Typing Rules

type-synonym *Type* = *Sec*

type-synonym *′Var TyEnv* = *′Var* ⇀ *Type*

locale *sifum-types* =
  sifum-lang ev_A ev_B + *sifum-security dma Stop* $eval_w$
  for ev_A :: ('*Var*, '*Val*) *Mem* ⇒ '*AExp* ⇒ '*Val*
  and ev_B :: ('*Var*, '*Val*) *Mem* ⇒ '*BExp* ⇒ *bool*

context *sifum-types*
begin

abbreviation *mm-equiv-abv2* :: (-, -, -) *LocalConf* ⇒ (-, -, -) *LocalConf* ⇒ *bool*
(**infix** ‹≈› *60*)
  where *mm-equiv-abv2 c c'* ≡ *mm-equiv-abv c c'*

abbreviation *eval-abv2* :: (-, '*Var*, '*Val*) *LocalConf* ⇒ (-, -, -) *LocalConf* ⇒ *bool*
  (**infixl** ‹⇝› *70*)
  where
  $x \leadsto y \equiv (x, y) \in eval_w$

abbreviation *low-indistinguishable-abv* :: '*Var Mds* ⇒ ('*Var*, '*AExp*, '*BExp*) *Stmt*
⇒ (-, -, -) *Stmt* ⇒ *bool*
  (‹- ∼ı -› [*100*, *100*] *80*)
  where
  $c \sim_{mds} c' \equiv$ *low-indistinguishable mds c c'*

definition *to-total* :: '*Var TyEnv* ⇒ '*Var* ⇒ *Sec*
where *to-total* Γ *v* ≡ *if v* ∈ *dom* Γ *then the* (Γ *v*) *else dma v*

definition *max-dom* :: *Sec set* ⇒ *Sec*
  where *max-dom xs* ≡ *if High* ∈ *xs then High else Low*

inductive *type-aexpr* :: '*Var TyEnv* ⇒ '*AExp* ⇒ *Type* ⇒ *bool* (‹- $\vdash_a$ - ∈ -› [*120*,
120, *120*] *1000*)
  where
  type-aexpr [*intro*!]: Γ $\vdash_a$ *e* ∈ *max-dom* (*image* (λ *x*. *to-total* Γ *x*) (*aexp-vars e*))

inductive-cases *type-aexpr-elim* [*elim*]: Γ $\vdash_a$ *e* ∈ *t*

inductive *type-bexpr* :: '*Var TyEnv* ⇒ '*BExp* ⇒ *Type* ⇒ *bool* (‹- $\vdash_b$ - ∈ - › [*120*,
120, *120*] *1000*)
  where
  type-bexpr [*intro*!]: Γ $\vdash_b$ *e* ∈ *max-dom* (*image* (λ *x*. *to-total* Γ *x*) (*bexp-vars e*))

inductive-cases *type-bexpr-elim* [*elim*]: Γ $\vdash_b$ *e* ∈ *t*

definition *mds-consistent* :: '*Var Mds* ⇒ '*Var TyEnv* ⇒ *bool*
  where *mds-consistent mds* Γ ≡

$dom\ \Gamma = \{(x :: 'Var).\ (dma\ x = Low \wedge x \in mds\ AsmNoRead) \vee$
$(dma\ x = High \wedge x \in mds\ AsmNoWrite)\}$

fun $add\text{-}anno\text{-}dom :: 'Var\ TyEnv \Rightarrow 'Var\ ModeUpd \Rightarrow 'Var\ set$
where
$add\text{-}anno\text{-}dom\ \Gamma\ (Acq\ v\ AsmNoRead) = (if\ dma\ v = Low\ then\ dom\ \Gamma \cup \{v\}\ else\ dom\ \Gamma)\ |$
$add\text{-}anno\text{-}dom\ \Gamma\ (Acq\ v\ AsmNoWrite) = (if\ dma\ v = High\ then\ dom\ \Gamma \cup \{v\}\ else\ dom\ \Gamma)\ |$
$add\text{-}anno\text{-}dom\ \Gamma\ (Acq\ v\ \text{-}) = dom\ \Gamma\ |$
$add\text{-}anno\text{-}dom\ \Gamma\ (Rel\ v\ AsmNoRead) = (if\ dma\ v = Low\ then\ dom\ \Gamma - \{v\}\ else\ dom\ \Gamma)\ |$
$add\text{-}anno\text{-}dom\ \Gamma\ (Rel\ v\ AsmNoWrite) = (if\ dma\ v = High\ then\ dom\ \Gamma - \{v\}\ else\ dom\ \Gamma)\ |$
$add\text{-}anno\text{-}dom\ \Gamma\ (Rel\ v\ \text{-}) = dom\ \Gamma$

definition $add\text{-}anno :: 'Var\ TyEnv \Rightarrow 'Var\ ModeUpd \Rightarrow 'Var\ TyEnv$ (**infix** ‹⊕› 60)
where
$\Gamma \oplus upd = ((\lambda x.\ Some\ (to\text{-}total\ \Gamma\ x))\ |`\ add\text{-}anno\text{-}dom\ \Gamma\ upd)$

definition $context\text{-}le :: 'Var\ TyEnv \Rightarrow 'Var\ TyEnv \Rightarrow bool$ (**infixr** ‹$\sqsubseteq_c$› 100)
where
$\Gamma \sqsubseteq_c \Gamma' \equiv (dom\ \Gamma = dom\ \Gamma') \wedge (\forall\ x \in dom\ \Gamma.\ the\ (\Gamma\ x) \sqsubseteq the\ (\Gamma'\ x))$

inductive $has\text{-}type :: 'Var\ TyEnv \Rightarrow ('Var, 'AExp, 'BExp)\ Stmt \Rightarrow 'Var\ TyEnv \Rightarrow bool$
(‹⊢ - {-} -› $[120, 120, 120]\ 1000$)
where
$stop\text{-}type\ [intro]: \vdash \Gamma\ \{Stop\}\ \Gamma\ |$
$skip\text{-}type\ [intro] : \vdash \Gamma\ \{Skip\}\ \Gamma\ |$
$assign_1 : [\![\ x \notin dom\ \Gamma\ ;\ \Gamma \vdash_a e \in t;\ t \sqsubseteq dma\ x\ ]\!] \Longrightarrow \vdash \Gamma\ \{x \leftarrow e\}\ \Gamma\ |$
$assign_2 : [\![\ x \in dom\ \Gamma\ ;\ \Gamma \vdash_a e \in t\ ]\!] \Longrightarrow has\text{-}type\ \Gamma\ (x \leftarrow e)\ (\Gamma\ (x := Some\ t))\ |$
$if\text{-}type\ [intro]: [\![\ \Gamma \vdash_b e \in High \longrightarrow$
$((\forall\ mds.\ mds\text{-}consistent\ mds\ \Gamma \longrightarrow (low\text{-}indistinguishable\ mds\ c_1\ c_2)) \wedge$
$(\forall\ x \in dom\ \Gamma'.\ \Gamma'\ x = Some\ High))$
$;\ \vdash \Gamma\ \{\ c_1\ \}\ \Gamma'$
$;\ \vdash \Gamma\ \{\ c_2\ \}\ \Gamma'\ ]\!] \Longrightarrow$
$\vdash \Gamma\ \{\ If\ e\ c_1\ c_2\ \}\ \Gamma'\ |$
$while\text{-}type\ [intro]: [\![\ \Gamma \vdash_b e \in Low\ ;\ \vdash \Gamma\ \{\ c\ \}\ \Gamma\ ]\!] \Longrightarrow \vdash \Gamma\ \{\ While\ e\ c\ \}\ \Gamma\ |$
$anno\text{-}type\ [intro]: [\![\ \Gamma' = \Gamma \oplus upd\ ;\ \vdash \Gamma'\ \{\ c\ \}\ \Gamma''\ ;\ c \neq Stop\ ;$
$\forall\ x.\ to\text{-}total\ \Gamma\ x \sqsubseteq to\text{-}total\ \Gamma'\ x\ ]\!] \Longrightarrow \vdash \Gamma\ \{\ c@[upd]\ \}\ \Gamma''\ |$
$seq\text{-}type\ [intro]: [\![\ \vdash \Gamma\ \{\ c_1\ \}\ \Gamma'\ ;\ \vdash \Gamma'\ \{\ c_2\ \}\ \Gamma''\ ]\!] \Longrightarrow \vdash \Gamma\ \{\ c_1\ ;;\ c_2\ \}\ \Gamma''\ |$
$sub : [\![\ \vdash \Gamma_1\ \{\ c\ \}\ \Gamma_1'\ ;\ \Gamma_2 \sqsubseteq_c \Gamma_1\ ;\ \Gamma_1' \sqsubseteq_c \Gamma_2'\ ]\!] \Longrightarrow \vdash \Gamma_2\ \{\ c\ \}\ \Gamma_2'$

5.2 Typing Soundness

The following predicate is needed to exclude some pathological cases, that abuse the *Stop* command which is not allowed to occur in actual programs.

fun *has-annotated-stop* :: (*'Var*, *'AExp*, *'BExp*) *Stmt* ⇒ *bool*
where
has-annotated-stop (*c*@[-]) = (*if c* = *Stop then True else has-annotated-stop c*) |
has-annotated-stop (*Seq p q*) = (*has-annotated-stop p* ∨ *has-annotated-stop q*) |
has-annotated-stop (*If - p q*) = (*has-annotated-stop p* ∨ *has-annotated-stop q*) |
has-annotated-stop (*While - p*) = *has-annotated-stop p* |
has-annotated-stop - = *False*

inductive-cases *has-type-elim*: ⊢ Γ { *c* } Γ′
inductive-cases *has-type-stop-elim*: ⊢ Γ { *Stop* } Γ′

definition *tyenv-eq* :: *'Var TyEnv* ⇒ (*'Var*, *'Val*) *Mem* ⇒ (*'Var*, *'Val*) *Mem* ⇒ *bool*
(**infix** ‹=ı› *60*)
where $mem_1 =_\Gamma mem_2 \equiv \forall\ x.\ (\textit{to-total}\ \Gamma\ x = Low \longrightarrow mem_1\ x = mem_2\ x)$

lemma *tyenv-eq-sym*: $mem_1 =_\Gamma mem_2 \Longrightarrow mem_2 =_\Gamma mem_1$
⟨*proof*⟩

inductive-set $\mathcal{R}_1$ :: *'Var TyEnv* ⇒ ((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *LocalConf rel*
and $\mathcal{R}_1$-*abv* :: *'Var TyEnv* ⇒
((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *LocalConf* ⇒
((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *LocalConf* ⇒
bool (‹- $\mathcal{R}^1$ı -› [*120*, *120*] *1000*)
for Γ′ :: *'Var TyEnv*
where
$x\ \mathcal{R}^1{}_\Gamma\ y \equiv (x, y) \in \mathcal{R}_1\ \Gamma$ |
intro [*intro*!] : ⟦ ⊢ Γ { *c* } Γ′ ; *mds-consistent mds* Γ ; $mem_1 =_\Gamma mem_2$ ⟧ ⟹ $\langle c, mds, mem_1\rangle\ \mathcal{R}^1{}_{\Gamma'}\ \langle c, mds, mem_2\rangle$

inductive-set $\mathcal{R}_2$ :: *'Var TyEnv* ⇒ ((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *LocalConf rel*
and $\mathcal{R}_2$-*abv* :: *'Var TyEnv* ⇒
((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *LocalConf* ⇒
((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *LocalConf* ⇒
bool (‹- $\mathcal{R}^2$ı -› [*120*, *120*] *1000*)
for Γ′ :: *'Var TyEnv*
where
$x\ \mathcal{R}^2{}_\Gamma\ y \equiv (x, y) \in \mathcal{R}_2\ \Gamma$ |
intro [*intro*!] : ⟦ $\langle c_1, mds, mem_1\rangle \approx \langle c_2, mds, mem_2\rangle$;
∀ *x* ∈ *dom* Γ′. Γ′ *x* = *Some High* ;
$\vdash \Gamma_1\ \{\ c_1\ \}\ \Gamma'$; $\vdash \Gamma_2\ \{\ c_2\ \}\ \Gamma'$;
mds-consistent mds Γ_1 ; *mds-consistent mds* Γ_2 ⟧ ⟹
$\langle c_1, mds, mem_1\rangle\ \mathcal{R}^2{}_{\Gamma'}\ \langle c_2, mds, mem_2\rangle$

inductive $\mathcal{R}_3$-*aux* :: *'Var TyEnv* ⇒ ((*'Var*, *'AExp*, *'BExp*) *Stmt*, *'Var*, *'Val*) *Lo-*

calConf ⇒
(('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf* ⇒
bool (‹- $\mathcal{R}^3$1 -› [*120*, *120*] *1000*)
and $\mathcal{R}_3$ :: '*Var TyEnv* ⇒ (('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf rel*
where
$\mathcal{R}_3$ Γ' ≡ {(lc_1, lc_2). $\mathcal{R}_3$-*aux* Γ' lc_1 lc_2} |
$intro_1$ [*intro*] : ⟦ ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^1{}_\Gamma$ ⟨c_2, *mds*, mem_2⟩; ⊢ Γ { *c* } Γ' ⟧ ⟹
⟨*Seq* c_1 *c*, *mds*, mem_1⟩ $\mathcal{R}^3{}_{\Gamma'}$ ⟨*Seq* c_2 *c*, *mds*, mem_2⟩ |
$intro_2$ [*intro*] : ⟦ ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^2{}_\Gamma$ ⟨c_2, *mds*, mem_2⟩; ⊢ Γ { *c* } Γ' ⟧ ⟹
⟨*Seq* c_1 *c*, *mds*, mem_1⟩ $\mathcal{R}^3{}_{\Gamma'}$ ⟨*Seq* c_2 *c*, *mds*, mem_2⟩ |
$intro_3$ [*intro*] : ⟦ ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^3{}_\Gamma$ ⟨c_2, *mds*, mem_2⟩; ⊢ Γ { *c* } Γ' ⟧ ⟹
⟨*Seq* c_1 *c*, *mds*, mem_1⟩ $\mathcal{R}^3{}_{\Gamma'}$ ⟨*Seq* c_2 *c*, *mds*, mem_2⟩

definition *weak-bisim* :: (('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf rel* ⇒
(('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf rel* ⇒ *bool*
where *weak-bisim* $\mathcal{T}_1$ $\mathcal{T}$ ≡ ∀ c_1 c_2 *mds* mem_1 mem_2 c_1' *mds*' mem_1'.
((⟨c_1, *mds*, mem_1⟩, ⟨c_2, *mds*, mem_2⟩) ∈ $\mathcal{T}_1$ ∧
(⟨c_1, *mds*, mem_1⟩ ⇝ ⟨c_1', *mds*', mem_1'⟩)) ⟶
(∃ c_2' mem_2'. ⟨c_2, *mds*, mem_2⟩ ⇝ ⟨c_2', *mds*', mem_2'⟩ ∧
(⟨c_1', *mds*', mem_1'⟩, ⟨c_2', *mds*', mem_2'⟩) ∈ $\mathcal{T}$)

inductive-set $\mathcal{R}$ :: '*Var TyEnv* ⇒
(('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf rel*
and $\mathcal{R}$-*abv* :: '*Var TyEnv* ⇒
(('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf* ⇒
(('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf* ⇒
bool (‹- $\mathcal{R}^u$1 -› [*120*, *120*] *1000*)
for Γ :: '*Var TyEnv*
where
x $\mathcal{R}^u{}_\Gamma$ *y* ≡ (*x*, *y*) ∈ $\mathcal{R}$ Γ |
$intro_1$: *lc* $\mathcal{R}^1{}_\Gamma$ *lc*' ⟹ (*lc*, *lc*') ∈ $\mathcal{R}$ Γ |
$intro_2$: *lc* $\mathcal{R}^2{}_\Gamma$ *lc*' ⟹ (*lc*, *lc*') ∈ $\mathcal{R}$ Γ |
$intro_3$: *lc* $\mathcal{R}^3{}_\Gamma$ *lc*' ⟹ (*lc*, *lc*') ∈ $\mathcal{R}$ Γ

inductive-cases $\mathcal{R}_1$-*elim* [*elim*]: ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^1{}_\Gamma$ ⟨c_2, *mds*, mem_2⟩
inductive-cases $\mathcal{R}_2$-*elim* [*elim*]: ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^2{}_\Gamma$ ⟨c_2, *mds*, mem_2⟩
inductive-cases $\mathcal{R}_3$-*elim* [*elim*]: ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^3{}_\Gamma$ ⟨c_2, *mds*, mem_2⟩

inductive-cases $\mathcal{R}$-*elim* [*elim*]: (⟨c_1, *mds*, mem_1⟩, ⟨c_2, *mds*, mem_2⟩) ∈ $\mathcal{R}$ Γ
inductive-cases $\mathcal{R}$-*elim*': (⟨c_1, *mds*, mem_1⟩, ⟨c_2, mds_2, mem_2⟩) ∈ $\mathcal{R}$ Γ
inductive-cases $\mathcal{R}_1$-*elim*' : ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^1{}_\Gamma$ ⟨c_2, mds_2, mem_2⟩
inductive-cases $\mathcal{R}_2$-*elim*' : ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^2{}_\Gamma$ ⟨c_2, mds_2, mem_2⟩
inductive-cases $\mathcal{R}_3$-*elim*' : ⟨c_1, *mds*, mem_1⟩ $\mathcal{R}^3{}_\Gamma$ ⟨c_2, mds_2, mem_2⟩

lemma $\mathcal{R}_1$-*sym*: *sym* ($\mathcal{R}_1$ Γ)

⟨*proof*⟩

lemma $\mathcal{R}_2$-*sym*: *sym* ($\mathcal{R}_2$ Γ)
⟨*proof*⟩

lemma $\mathcal{R}_3$-*sym*: *sym* ($\mathcal{R}_3$ Γ)
⟨*proof*⟩

lemma $\mathcal{R}$-*mds* [*simp*]: $\langle c_1,\ mds,\ mem_1\rangle\ \mathcal{R}^u{}_\Gamma\ \langle c_2,\ mds',\ mem_2\rangle \Longrightarrow mds = mds'$
⟨*proof*⟩

lemma $\mathcal{R}$-*sym*: *sym* ($\mathcal{R}$ Γ)
⟨*proof*⟩

lemma $\mathcal{R}_1$-*closed-glob-consistent*: *closed-glob-consistent* ($\mathcal{R}_1$ Γ′)
⟨*proof*⟩

lemma $\mathcal{R}_2$-*closed-glob-consistent*: *closed-glob-consistent* ($\mathcal{R}_2$ Γ′)
⟨*proof*⟩

fun *closed-glob-helper* :: ′*Var TyEnv* ⇒ ((′*Var*, ′*AExp*, ′*BExp*) *Stmt*, ′*Var*, ′*Val*) *LocalConf* ⇒ ((′*Var*, ′*AExp*, ′*BExp*) *Stmt*, ′*Var*, ′*Val*) *LocalConf* ⇒ *bool*
where
closed-glob-helper $\Gamma'\ \langle c_1,\ mds,\ mem_1\rangle\ \langle c_2,\ mds_2,\ mem_2\rangle$ =
$(\forall\ x.\ ((dma\ x = High \land x \notin mds\ AsmNoWrite) \longrightarrow$
$(\forall\ v_1\ v_2.\ (\langle\ c_1,\ mds,\ mem_1\ (x := v_1)\ \rangle,\ \langle\ c_2,\ mds,\ mem_2\ (x := v_2)\ \rangle) \in \mathcal{R}_3\ \Gamma')) \land$
$((dma\ x = Low \land x \notin mds\ AsmNoWrite) \longrightarrow$
$(\forall\ v.\ (\langle\ c_1,\ mds,\ mem_1\ (x := v)\ \rangle,\ \langle\ c_2,\ mds,\ mem_2\ (x := v)\ \rangle) \in \mathcal{R}_3\ \Gamma')))$

lemma $\mathcal{R}_3$-*closed-glob-consistent*:
assumes *R3*: $\langle c_1,\ mds,\ mem_1\rangle\ \mathcal{R}^3{}_{\Gamma'}\ \langle c_2,\ mds,\ mem_2\rangle$
shows ∀ *x*.
$(dma\ x = High \land x \notin mds\ AsmNoWrite \longrightarrow$
$(\forall v_1\ v_2.\ (\langle c_1,\ mds,\ mem_1(x := v_1)\rangle,\ \langle c_2,\ mds,\ mem_2(x := v_2)\rangle) \in \mathcal{R}_3\ \Gamma'))$
$\land$
$(dma\ x = Low \land x \notin mds\ AsmNoWrite \longrightarrow (\forall v.\ (\langle c_1,\ mds,\ mem_1(x := v)\rangle,\ \langle c_2,\ mds,\ mem_2(x := v)\rangle) \in \mathcal{R}_3\ \Gamma'))$
⟨*proof*⟩

lemma $\mathcal{R}$-*closed-glob-consistent*: *closed-glob-consistent* ($\mathcal{R}$ Γ′)
⟨*proof*⟩

lemma *type-low-vars-low*:
  assumes *typed*: $\Gamma \vdash_a e \in Low$
  assumes *mds-cons*: *mds-consistent mds* Γ
  assumes *x-in-vars*: $x \in$ *aexp-vars e*
  shows *to-total* $\Gamma\ x = Low$
  ⟨*proof*⟩

lemma *type-low-vars-low-b*:
  assumes *typed* : $\Gamma \vdash_b e \in Low$
  assumes *mds-cons*: *mds-consistent mds* Γ
  assumes *x-in-vars*: $x \in$ *bexp-vars e*
  shows *to-total* $\Gamma\ x = Low$
  ⟨*proof*⟩

lemma *mode-update-add-anno*:
  mds-consistent mds $\Gamma \Longrightarrow$ *mds-consistent* (*update-modes upd mds*) ($\Gamma \oplus$ *upd*)
  ⟨*proof*⟩

lemma *context-le-trans*: $[\![\Gamma \sqsubseteq_c \Gamma' ; \Gamma' \sqsubseteq_c \Gamma'']\!] \Longrightarrow \Gamma \sqsubseteq_c \Gamma''$
  ⟨*proof*⟩

lemma *context-le-refl* [*simp*]: $\Gamma \sqsubseteq_c \Gamma$
  ⟨*proof*⟩

lemma *stop-cxt* :
  $[\![\vdash \Gamma \ \{ \ c \ \} \ \Gamma' ; c = Stop]\!] \Longrightarrow \Gamma \sqsubseteq_c \Gamma'$
  ⟨*proof*⟩

lemma *preservation*:
  assumes *typed*: $\vdash \Gamma \ \{ \ c \ \} \ \Gamma'$
  assumes *eval*: $\langle c, mds, mem\rangle \rightsquigarrow \langle c', mds', mem'\rangle$
  shows $\exists\ \Gamma''.\ (\vdash \Gamma'' \ \{ \ c' \ \} \ \Gamma') \wedge$ (*mds-consistent mds* $\Gamma \longrightarrow$ *mds-consistent mds'*
Γ'')
  ⟨*proof*⟩

lemma $\mathcal{R}_1$-*mem-eq*: $\langle c_1, mds, mem_1\rangle\ \mathcal{R}^1{}_{\Gamma'}\ \langle c_2, mds, mem_2\rangle \Longrightarrow mem_1 =_{mds}{}^l$
mem_2
  ⟨*proof*⟩

lemma $\mathcal{R}_2$-*mem-eq*: $\langle c_1, mds, mem_1\rangle\ \mathcal{R}^2{}_{\Gamma'}\ \langle c_2, mds, mem_2\rangle \Longrightarrow mem_1 =_{mds}{}^l$
mem_2
  ⟨*proof*⟩

fun *bisim-helper* :: (('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf* ⇒
  (('*Var*, '*AExp*, '*BExp*) *Stmt*, '*Var*, '*Val*) *LocalConf* ⇒ *bool*
  where
  bisim-helper $\langle c_1, mds, mem_1\rangle\ \langle c_2, mds_2, mem_2\rangle = mem_1 =_{mds}{}^l\ mem_2$

lemma $\mathcal{R}_3$-*mem-eq*: $\langle c_1,\ mds,\ mem_1\rangle\ \mathcal{R}^3{}_{\Gamma'}\ \langle c_2,\ mds,\ mem_2\rangle \Longrightarrow mem_1 =_{mds}{}^l$ mem_2
⟨*proof*⟩

lemma $\mathcal{R}_2$-*bisim-step*:
assumes *case2*: $\langle c_1,\ mds,\ mem_1\rangle\ \mathcal{R}^2{}_{\Gamma'}\ \langle c_2,\ mds,\ mem_2\rangle$
assumes *eval*: $\langle c_1,\ mds,\ mem_1\rangle \rightsquigarrow \langle c_1',\ mds',\ mem_1'\rangle$
shows $\exists\ c_2'\ mem_2'.\ \langle c_2,\ mds,\ mem_2\rangle \rightsquigarrow \langle c_2',\ mds',\ mem_2'\rangle \wedge \langle c_1',\ mds',\ mem_1'\rangle$ $\mathcal{R}^2{}_{\Gamma'}\ \langle c_2',\ mds',\ mem_2'\rangle$
⟨*proof*⟩

lemma $\mathcal{R}_2$-*weak-bisim*:
weak-bisim $(\mathcal{R}_2\ \Gamma')\ (\mathcal{R}\ \Gamma')$
⟨*proof*⟩

lemma $\mathcal{R}_2$-*bisim*: *strong-low-bisim-mm* $(\mathcal{R}_2\ \Gamma')$
⟨*proof*⟩

lemma *annotated-no-stop*: ⟦ ¬ *has-annotated-stop* (*c*@[*upd*]) ⟧ ⟹ ¬ *has-annotated-stop* *c*
⟨*proof*⟩

lemma *typed-no-annotated-stop*:
$[\![\vdash \Gamma\ \{\ c\ \}\ \Gamma']\!] \Longrightarrow \neg$ *has-annotated-stop c*
⟨*proof*⟩

lemma *not-stop-eval*:
$[\![\ c \neq$ *Stop* ; ¬ *has-annotated-stop c* $]\!] \Longrightarrow$
$\forall\ mds\ mem.\ \exists\ c'\ mds'\ mem'.\ \langle c,\ mds,\ mem\rangle \rightsquigarrow \langle c',\ mds',\ mem'\rangle$
⟨*proof*⟩

lemma *stop-bisim*:
assumes *bisim*: $\langle Stop,\ mds,\ mem_1\rangle \approx \langle c,\ mds,\ mem_2\rangle$
assumes *typeable*: $\vdash \Gamma\ \{\ c\ \}\ \Gamma'$
shows $c = Stop$
⟨*proof*⟩

lemma $\mathcal{R}$-*typed-step*:
$[\![\vdash \Gamma\ \{\ c_1\ \}\ \Gamma'$;
mds-consistent mds Γ ;

$mem_1 =_\Gamma mem_2$;
$\langle c_1, mds, mem_1\rangle \rightsquigarrow \langle c_1', mds', mem_1'\rangle$ $]\!] \Longrightarrow$
$(\exists\ c_2'\ mem_2'.\ \langle c_2, mds, mem_2\rangle \rightsquigarrow \langle c_2', mds', mem_2'\rangle \wedge$
$\langle c_1', mds', mem_1'\rangle\ \mathcal{R}^u{}_{\Gamma'}\ \langle c_2', mds', mem_2'\rangle)$
⟨*proof*⟩

lemma $\mathcal{R}_1$*-weak-bisim*:
weak-bisim $(\mathcal{R}_1\ \Gamma')$ $(\mathcal{R}\ \Gamma')$
⟨*proof*⟩

lemma $\mathcal{R}$*-to-*$\mathcal{R}_3$: $[\![\ \langle c_1, mds, mem_1\rangle\ \mathcal{R}^u{}_\Gamma\ \langle c_2, mds, mem_2\rangle$; $\vdash \Gamma\ \{\ c\ \}\ \Gamma'\ ]\!] \Longrightarrow$
$\langle c_1$;; $c, mds, mem_1\rangle\ \mathcal{R}^3{}_{\Gamma'}\ \langle c_2$;; $c, mds, mem_2\rangle$
⟨*proof*⟩

lemma $\mathcal{R}_2$*-implies-typeable*: $\langle c_1, mds, mem_1\rangle\ \mathcal{R}^2{}_{\Gamma'}\ \langle c_2, mds, mem_2\rangle \Longrightarrow \exists\ \Gamma_1.\ \vdash$
$\Gamma_1\ \{\ c_2\ \}\ \Gamma'$
⟨*proof*⟩

lemma $\mathcal{R}_3$*-weak-bisim*:
weak-bisim $(\mathcal{R}_3\ \Gamma')$ $(\mathcal{R}\ \Gamma')$
⟨*proof*⟩

lemma $\mathcal{R}$*-bisim*: *strong-low-bisim-mm* $(\mathcal{R}\ \Gamma')$
⟨*proof*⟩

lemma *Typed-in-*$\mathcal{R}$:
assumes *typeable*: $\vdash \Gamma\ \{\ c\ \}\ \Gamma'$
assumes *mds-cons*: *mds-consistent mds* Γ
assumes *mem-eq*: $\forall\ x.$ *to-total* $\Gamma\ x = Low \longrightarrow mem_1\ x = mem_2\ x$
shows $\langle c, mds, mem_1\rangle\ \mathcal{R}^u{}_{\Gamma'}\ \langle c, mds, mem_2\rangle$
⟨*proof*⟩

theorem *type-soundness*:
assumes *well-typed*: $\vdash \Gamma\ \{\ c\ \}\ \Gamma'$
assumes *mds-cons*: *mds-consistent mds* Γ
assumes *mem-eq*: $\forall\ x.$ *to-total* $\Gamma\ x = Low \longrightarrow mem_1\ x = mem_2\ x$
shows $\langle c, mds, mem_1\rangle \approx \langle c, mds, mem_2\rangle$
⟨*proof*⟩

definition Γ_0 :: *'Var TyEnv*
where $\Gamma_0\ x = None$

inductive *type-global* :: (*'Var*, *'AExp*, *'BExp*) *Stmt list* ⇒ *bool*
(‹⊢ -› [*120*] *1000*)
where

$\llbracket$ *list-all* (λ *c*. $\vdash \Gamma_0$ { *c* } Γ_0) *cs* ;
$\forall$ *mem*. *sound-mode-use* (*add-initial-modes cs*, *mem*) $\rrbracket \Longrightarrow$
type-global cs

inductive-cases *type-global-elim*: $\vdash$ *cs*

lemma mds_s*-consistent*: *mds-consistent* mds_s Γ_0
⟨*proof*⟩

lemma *typed-secure*:
$\llbracket \vdash \Gamma_0$ { *c* } $\Gamma_0 \rrbracket \Longrightarrow$ *com-sifum-secure c*
⟨*proof*⟩

lemma $\llbracket$ *mds-consistent mds* Γ_0 ; *dma x* = *Low* $\rrbracket \Longrightarrow x \notin$ *mds AsmNoRead*
⟨*proof*⟩

lemma *list-all-set*: $\forall$ $x \in$ *set xs*. *P x* $\Longrightarrow$ *list-all P xs*
⟨*proof*⟩

theorem *type-soundness-global*:
assumes *typeable*: $\vdash$ *cs*
assumes *no-assms-term*: *no-assumptions-on-termination cs*
shows *prog-sifum-secure cs*
⟨*proof*⟩

end
end

6 Type System for Ensuring Locally Sound Use of Modes

theory *LocallySoundModeUse*
imports *Main Security Language*
begin

6.1 Typing Rules

locale *sifum-modes* = *sifum-lang* ev_A ev_B +
sifum-security dma Stop $eval_w$
for ev_A :: (*'Var*, *'Val*) *Mem* $\Rightarrow$ *'AExp* $\Rightarrow$ *'Val*
and ev_B :: (*'Var*, *'Val*) *Mem* $\Rightarrow$ *'BExp* $\Rightarrow$ *bool*

context *sifum-modes*
begin

abbreviation *eval-abv-modes* :: (-, *'Var*, *'Val*) *LocalConf* $\Rightarrow$ (-, -, -) *LocalConf* $\Rightarrow$ *bool*
(**infixl** ‹$\leadsto$› *70*)

where
$x \rightsquigarrow y \equiv (x, y) \in eval_w$

fun *update-annos* :: *'Var Mds* ⇒ *'Var ModeUpd list* ⇒ *'Var Mds*
(**infix** ‹⊕› *140*)
where
update-annos mds [] = *mds* |
update-annos mds (*a* # *as*) = *update-annos* (*update-modes a mds*) *as*

fun *annotate* :: (*'Var*, *'AExp*, *'BExp*) *Stmt* ⇒ *'Var ModeUpd list* ⇒ (*'Var*, *'AExp*, *'BExp*) *Stmt*
(**infix** ‹⊗› *140*)
where
annotate c [] = *c* |
annotate c (*a* # *as*) = (*annotate c as*)@[*a*]

inductive *mode-type* :: *'Var Mds* ⇒
(*'Var*, *'AExp*, *'BExp*) *Stmt* ⇒
'Var Mds ⇒ *bool* (‹⊢ - { - } -›)
where
skip: ⊢ *mds* { *Skip* ⊗ *annos* } (*mds* ⊕ *annos*) |
assign: ⟦ $x \notin$ *mds GuarNoWrite* ; *aexp-vars e* ∩ *mds GuarNoRead* = {} ⟧ ⟹
⊢ *mds* { ($x \leftarrow e$) ⊗ *annos* } (*mds* ⊕ *annos*) |
if-: ⟦ ⊢ (*mds* ⊕ *annos*) { c_1 } *mds''* ;
⊢ (*mds* ⊕ *annos*) { c_2 } *mds''* ;
bexp-vars e ∩ *mds GuarNoRead* = {} ⟧ ⟹
⊢ *mds* { *If e* c_1 c_2 ⊗ *annos* } *mds''* |
while: ⟦ *mds'* = *mds* ⊕ *annos* ; ⊢ *mds'* { *c* } *mds'* ; *bexp-vars e* ∩ *mds' GuarNoRead* = {} ⟧ ⟹
⊢ *mds* { *While e c* ⊗ *annos* } *mds'* |
seq: ⟦ ⊢ *mds* { c_1 } *mds'* ; ⊢ *mds'* { c_2 } *mds''* ⟧ ⟹ ⊢ *mds* { c_1 ;; c_2 } *mds''* |
sub: ⟦ ⊢ mds_2 { *c* } mds_2' ; $mds_1 \leq mds_2$; $mds_2' \leq mds_1'$ ⟧ ⟹
⊢ mds_1 { *c* } mds_1'

6.2 Soundness of the Type System

lemma *cxt-eval*:
⟦ ⟨*cxt-to-stmt* [] *c*, *mds*, *mem*⟩ ⇝ ⟨*cxt-to-stmt* [] *c'*, *mds'*, *mem'*⟩ ⟧ ⟹
⟨*c*, *mds*, *mem*⟩ ⇝ ⟨*c'*, *mds'*, *mem'*⟩
⟨*proof*⟩

lemma *update-preserves-le*:
$mds_1 \leq mds_2 \Longrightarrow (mds_1 \oplus annos) \leq (mds_2 \oplus annos)$
⟨*proof*⟩

lemma *doesnt-read-annos*:
doesnt-read c x ⟹ *doesnt-read* (*c* ⊗ *annos*) *x*
⟨*proof*⟩

lemma *doesnt-modify-annos*:
$doesnt\text{-}modify\ c\ x \Longrightarrow doesnt\text{-}modify\ (c \otimes annos)\ x$
$\langle proof \rangle$

lemma *stop-loc-reach*:
$[\![\ \langle c', mds', mem' \rangle \in loc\text{-}reach\ \langle Stop, mds, mem \rangle\ ]\!] \Longrightarrow$
$c' = Stop \wedge mds' = mds$
$\langle proof \rangle$

lemma *stop-doesnt-access*:
$doesnt\text{-}modify\ Stop\ x \wedge doesnt\text{-}read\ Stop\ x$
$\langle proof \rangle$

lemma *skip-eval-step*:
$\langle Skip \otimes annos, mds, mem \rangle \rightsquigarrow \langle Stop, mds \oplus annos, mem \rangle$
$\langle proof \rangle$

lemma *skip-eval-elim*:
$[\![\ \langle Skip \otimes annos, mds, mem \rangle \rightsquigarrow \langle c', mds', mem' \rangle\ ]\!] \Longrightarrow c' = Stop \wedge mds' = mds \oplus annos \wedge mem' = mem$
$\langle proof \rangle$

lemma *skip-doesnt-read*:
$doesnt\text{-}read\ (Skip \otimes annos)\ x$
$\langle proof \rangle$

lemma *skip-doesnt-write*:
$doesnt\text{-}modify\ (Skip \otimes annos)\ x$
$\langle proof \rangle$

lemma *skip-loc-reach*:
$[\![\ \langle c', mds', mem' \rangle \in loc\text{-}reach\ \langle Skip \otimes annos, mds, mem \rangle\ ]\!] \Longrightarrow$
$(c' = Stop \wedge mds' = (mds \oplus annos)) \vee (c' = Skip \otimes annos \wedge mds' = mds)$
$\langle proof \rangle$

lemma *skip-doesnt-access*:
$[\![\ lc \in loc\text{-}reach\ \langle Skip \otimes annos, mds, mem \rangle\ ;\ lc = \langle c', mds', mem' \rangle\ ]\!] \Longrightarrow doesnt\text{-}read\ c'\ x \wedge doesnt\text{-}modify\ c'\ x$
$\langle proof \rangle$

lemma *assign-doesnt-modify*:
$[\![\ x \neq y\ ]\!] \Longrightarrow doesnt\text{-}modify\ ((x \leftarrow e) \otimes annos)\ y$
$\langle proof \rangle$

lemma *assign-annos-eval*:
$\langle (x \leftarrow e) \otimes annos, mds, mem \rangle \rightsquigarrow \langle Stop, mds \oplus annos, mem\ (x := ev_A\ mem\ e) \rangle$

$\langle proof \rangle$

lemma *assign-annos-eval-elim*:
$[\![\langle (x \leftarrow e) \otimes annos,\ mds,\ mem \rangle \rightsquigarrow \langle c',\ mds',\ mem' \rangle]\!] \Longrightarrow$
$c' = Stop \wedge mds' = mds \oplus annos$
$\langle proof \rangle$

lemma *mem-upd-commute*:
$[\![x \neq y]\!] \Longrightarrow mem\ (x := v_1,\ y := v_2) = mem\ (y := v_2,\ x := v_1)$
$\langle proof \rangle$

lemma *assign-doesnt-read*:
$[\![y \notin aexp\text{-}vars\ e]\!] \Longrightarrow doesnt\text{-}read\ ((x \leftarrow e) \otimes annos)\ y$
$\langle proof \rangle$

lemma *assign-loc-reach*:
$[\![\langle c',\ mds',\ mem' \rangle \in loc\text{-}reach\ \langle (x \leftarrow e) \otimes annos,\ mds,\ mem \rangle]\!] \Longrightarrow$
$(c' = Stop \wedge mds' = (mds \oplus annos)) \vee (c' = (x \leftarrow e) \otimes annos \wedge mds' = mds)$
$\langle proof \rangle$

lemma *if-doesnt-modify*:
$doesnt\text{-}modify\ (If\ e\ c_1\ c_2 \otimes annos)\ x$
$\langle proof \rangle$

lemma *vars-eval$_B$*:
$x \notin bexp\text{-}vars\ e \Longrightarrow ev_B\ mem\ e = ev_B\ (mem\ (x := v))\ e$
$\langle proof \rangle$

lemma *if-doesnt-read*:
$x \notin bexp\text{-}vars\ e \Longrightarrow doesnt\text{-}read\ (If\ e\ c_1\ c_2 \otimes annos)\ x$
$\langle proof \rangle$

lemma *if-eval-true*:
$[\![ev_B\ mem\ e]\!] \Longrightarrow$
$\langle If\ e\ c_1\ c_2 \otimes annos,\ mds,\ mem \rangle \rightsquigarrow \langle c_1,\ mds \oplus annos,\ mem \rangle$
$\langle proof \rangle$

lemma *if-eval-false*:
$[\![\neg\ ev_B\ mem\ e]\!] \Longrightarrow$
$\langle If\ e\ c_1\ c_2 \otimes annos,\ mds,\ mem \rangle \rightsquigarrow \langle c_2,\ mds \oplus annos,\ mem \rangle$
$\langle proof \rangle$

lemma *if-eval-elim*:
$[\![\langle If\ e\ c_1\ c_2 \otimes annos,\ mds,\ mem \rangle \rightsquigarrow \langle c',\ mds',\ mem' \rangle]\!] \Longrightarrow$
$((c' = c_1 \wedge ev_B\ mem\ e) \vee (c' = c_2 \wedge \neg\ ev_B\ mem\ e)) \wedge mds' = mds \oplus annos \wedge mem' = mem$
$\langle proof \rangle$

lemma *if-eval-elim'*:

$[\![\langle \mathit{If}\ e\ c_1\ c_2,\ \mathit{mds},\ \mathit{mem}\rangle \rightsquigarrow \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle\ ]\!] \Longrightarrow$
$((c' = c_1 \wedge \mathit{ev}_B\ \mathit{mem}\ e) \vee (c' = c_2 \wedge \neg\ \mathit{ev}_B\ \mathit{mem}\ e)) \wedge \mathit{mds}' = \mathit{mds} \wedge \mathit{mem}' = \mathit{mem}$
$\langle \mathit{proof}\rangle$

lemma *loc-reach-refl'*:
$\langle c,\ \mathit{mds},\ \mathit{mem}\rangle \in \mathit{loc\text{-}reach}\ \langle c,\ \mathit{mds},\ \mathit{mem}\rangle$
$\langle \mathit{proof}\rangle$

lemma *if-loc-reach*:
$[\![\langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle \mathit{If}\ e\ c_1\ c_2 \otimes \mathit{annos},\ \mathit{mds},\ \mathit{mem}\rangle\ ]\!] \Longrightarrow$
$(c' = \mathit{If}\ e\ c_1\ c_2 \otimes \mathit{annos} \wedge \mathit{mds}' = \mathit{mds}) \vee$
$(\exists\ \mathit{mem}''.\ \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_1,\ \mathit{mds} \oplus \mathit{annos},\ \mathit{mem}''\rangle) \vee$
$(\exists\ \mathit{mem}''.\ \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_2,\ \mathit{mds} \oplus \mathit{annos},\ \mathit{mem}''\rangle)$
$\langle \mathit{proof}\rangle$

lemma *if-loc-reach'*:
$[\![\langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle \mathit{If}\ e\ c_1\ c_2,\ \mathit{mds},\ \mathit{mem}\rangle\ ]\!] \Longrightarrow$
$(c' = \mathit{If}\ e\ c_1\ c_2 \wedge \mathit{mds}' = \mathit{mds}) \vee$
$(\exists\ \mathit{mem}''.\ \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_1,\ \mathit{mds},\ \mathit{mem}''\rangle) \vee$
$(\exists\ \mathit{mem}''.\ \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_2,\ \mathit{mds},\ \mathit{mem}''\rangle)$
$\langle \mathit{proof}\rangle$

lemma *seq-loc-reach*:
$[\![\langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_1\ ;;\ c_2,\ \mathit{mds},\ \mathit{mem}\rangle\ ]\!] \Longrightarrow$
$(\exists\ c''.\ c' = c''\ ;;\ c_2 \wedge \langle c'',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_1,\ \mathit{mds},\ \mathit{mem}\rangle) \vee$
$(\exists\ c''\ \mathit{mds}''\ \mathit{mem}''.\ \langle \mathit{Stop},\ \mathit{mds}'',\ \mathit{mem}''\rangle \in \mathit{loc\text{-}reach}\ \langle c_1,\ \mathit{mds},\ \mathit{mem}\rangle \wedge$
$\langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_2,\ \mathit{mds}'',\ \mathit{mem}''\rangle)$
$\langle \mathit{proof}\rangle$

lemma *seq-doesnt-read*:
$[\![\mathit{doesnt\text{-}read}\ c\ x\ ]\!] \Longrightarrow \mathit{doesnt\text{-}read}\ (c\ ;;\ c')\ x$
$\langle \mathit{proof}\rangle$

lemma *seq-doesnt-modify*:
$[\![\mathit{doesnt\text{-}modify}\ c\ x\ ]\!] \Longrightarrow \mathit{doesnt\text{-}modify}\ (c\ ;;\ c')\ x$
$\langle \mathit{proof}\rangle$

inductive-cases *seq-stop-elim'*: $\langle \mathit{Stop}\ ;;\ c,\ \mathit{mds},\ \mathit{mem}\rangle \rightsquigarrow \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle$

lemma *seq-stop-elim*: $\langle \mathit{Stop}\ ;;\ c,\ \mathit{mds},\ \mathit{mem}\rangle \rightsquigarrow \langle c',\ \mathit{mds}',\ \mathit{mem}'\rangle \Longrightarrow$
$c' = c \wedge \mathit{mds}' = \mathit{mds} \wedge \mathit{mem}' = \mathit{mem}$
$\langle \mathit{proof}\rangle$

lemma *seq-split*:
$[\![\langle \mathit{Stop},\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_1\ ;;\ c_2,\ \mathit{mds},\ \mathit{mem}\rangle\ ]\!] \Longrightarrow$
$\exists\ \mathit{mds}''\ \mathit{mem}''.\ \langle \mathit{Stop},\ \mathit{mds}'',\ \mathit{mem}''\rangle \in \mathit{loc\text{-}reach}\ \langle c_1,\ \mathit{mds},\ \mathit{mem}\rangle \wedge$
$\langle \mathit{Stop},\ \mathit{mds}',\ \mathit{mem}'\rangle \in \mathit{loc\text{-}reach}\ \langle c_2,\ \mathit{mds}'',\ \mathit{mem}''\rangle$
$\langle \mathit{proof}\rangle$

lemma *while-eval*:
$\langle While\ e\ c \otimes annos,\ mds,\ mem\rangle \rightsquigarrow \langle(If\ e\ (c\ ;;\ While\ e\ c)\ Stop),\ mds \oplus annos,\ mem\rangle$
⟨*proof*⟩

lemma *while-eval′*:
$\langle While\ e\ c,\ mds,\ mem\rangle \rightsquigarrow \langle If\ e\ (c\ ;;\ While\ e\ c)\ Stop,\ mds,\ mem\rangle$
⟨*proof*⟩

lemma *while-eval-elim*:
$[\![\ \langle While\ e\ c \otimes annos,\ mds,\ mem\rangle \rightsquigarrow \langle c',\ mds',\ mem'\rangle\ ]\!] \Longrightarrow$
$(c' = If\ e\ (c\ ;;\ While\ e\ c)\ Stop \wedge mds' = mds \oplus annos \wedge mem' = mem)$
⟨*proof*⟩

lemma *while-eval-elim′*:
$[\![\ \langle While\ e\ c,\ mds,\ mem\rangle \rightsquigarrow \langle c',\ mds',\ mem'\rangle\ ]\!] \Longrightarrow$
$(c' = If\ e\ (c\ ;;\ While\ e\ c)\ Stop \wedge mds' = mds \wedge mem' = mem)$
⟨*proof*⟩

lemma *while-doesnt-read*:
$[\![\ x \notin bexp\text{-}vars\ e\ ]\!] \Longrightarrow doesnt\text{-}read\ (While\ e\ c \otimes annos)\ x$
⟨*proof*⟩

lemma *while-doesnt-modify*:
$doesnt\text{-}modify\ (While\ e\ c \otimes annos)\ x$
⟨*proof*⟩

lemma *disjE3*:
$[\![\ A \vee B \vee C\ ;\ A \Longrightarrow P\ ;\ B \Longrightarrow P\ ;\ C \Longrightarrow P\ ]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *disjE5*:
$[\![\ A \vee B \vee C \vee D \vee E\ ;\ A \Longrightarrow P\ ;\ B \Longrightarrow P\ ;\ C \Longrightarrow P\ ;\ D \Longrightarrow P\ ;\ E \Longrightarrow P\ ]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *if-doesnt-read′*:
$x \notin bexp\text{-}vars\ e \Longrightarrow doesnt\text{-}read\ (If\ e\ c_1\ c_2)\ x$
⟨*proof*⟩

theorem *mode-type-sound*:
assumes *typeable*: $\vdash mds_1\ \{\ c\ \}\ mds_1'$
assumes *mode-le*: $mds_2 \leq mds_1$
shows $\forall\ mem.\ (\langle Stop,\ mds_2',\ mem'\rangle \in loc\text{-}reach\ \langle c,\ mds_2,\ mem\rangle \longrightarrow mds_2' \leq mds_1') \wedge$
$locally\text{-}sound\text{-}mode\text{-}use\ \langle c,\ mds_2,\ mem\rangle$
⟨*proof*⟩

end

end

References

[MSS11] Heiko Mantel, David Sands, and Henning Sudbrock. Assumptions and Guarantees for Compositional Noninterference. In *CSF*, pages 218–232. IEEE Computer Society, 2011.