Compositional Security-Preserving Refinement for Concurrent Imperative Programs

Toby Murray, Robert Sison, Edward Pierzchalski and Christine Rizkallah

February 23, 2021

Abstract

The paper "Compositional Verification and Refinement of Concurrent Value-Dependent Noninterference" by Murray et. al. [MSPR16] presents a compositional theory of refinement for a value-dependent noninterference property, defined in [Mur15], for concurrent programs. This development formalises that refinement theory, and demonstrates its application on some small examples.

The formalisation is contained in the theory `CompositionalRefinement.thy`.

Examples are also present in the formalisation in the `Examples/` directory.

Contents

theory *CompositionalRefinement*
imports *Dependent-SIFUM-Type-Systems.Compositionality*
begin

lemma *inj-card-le*:

inj (*f*::'*a* ⇒ '*b*) ⟹ *finite* (*UNIV*::'*b set*) ⟹ *card* (*UNIV*::'*a set*) ≤ *card* (*UNIV*::'*b set*)
⟨*proof*⟩

We define a generic locale for capturing refinement between an abstract and a concrete program. We then define and prove sufficient, conditions that preserve local security from the abstract to the concrete program.

Below we define a second locale that is more restrictive than this one. Specifically, this one allows the concrete program to have extra variables not present in the abstract one. These variables might be used, for instance, to implement a runtime stack that was implicit in the semantics of the abstract program; or as temporary storage for expression evaluation that may (appear to be) atomic in the abstract semantics.

The simpler locale below forbids extra variables in the concrete program, making the necessary conditions for preservation of local security simpler.

locale *sifum-refinement* =
abs: *sifum-security* dma_A $\mathcal{C}$-$vars_A$ $\mathcal{C}_A$ $eval_A$ *some-val* +
conc: *sifum-security* dma_C $\mathcal{C}$-$vars_C$ $\mathcal{C}_C$ $eval_C$ *some-val*
for dma_A :: ('Var_A,'*Val*) *Mem* ⇒ 'Var_A ⇒ *Sec*
and dma_C :: ('Var_C,'*Val*) *Mem* ⇒ 'Var_C ⇒ *Sec*
and $\mathcal{C}$-$vars_A$ :: 'Var_A ⇒ 'Var_A *set*
and $\mathcal{C}$-$vars_C$ :: 'Var_C ⇒ 'Var_C *set*
and $\mathcal{C}_A$ :: 'Var_A *set*
and $\mathcal{C}_C$ :: 'Var_C *set*
and $eval_A$ :: ('Com_A, 'Var_A, '*Val*) *LocalConf rel*
and $eval_C$ :: ('Com_C, 'Var_C, '*Val*) *LocalConf rel*
and *some-val* :: '*Val* +
fixes var_C-*of* :: 'Var_A ⇒ 'Var_C
assumes var_C-*of-inj*: *inj* var_C-*of*
assumes *dma-consistent*:
dma_A (λx_A. mem_C (var_C-*of* x_A)) x_A = dma_C mem_C (var_C-*of* x_A)
assumes $\mathcal{C}$-*vars-consistent*:
(var_C-*of* ‘ $\mathcal{C}$-$vars_A$ x_A) = $\mathcal{C}$-$vars_C$ (var_C-*of* x_A)

assumes *control-vars-are-A-vars*:
$\mathcal{C}_C$ = var_C-*of* ‘ $\mathcal{C}_A$

1 General Compositional Refinement

The type of state relations between the abstract and compiled components. The job of a certifying compiler will be to exhibit one of these for each component it compiles. Below we'll define the conditions that such a relation needs to satisfy to give compositional refinement.

type-synonym ('Com_A, 'Var_A, '*Val*, 'Com_C, 'Var_C) *state-relation* =
(('Com_A, 'Var_A, '*Val*) *LocalConf* × ('Com_C, 'Var_C, '*Val*) *LocalConf*) *set*

context *sifum-refinement* **begin**

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

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

abbreviation
eval-abv$_A$ :: (*'Com*$_A$, *'Var*$_A$, *'Val*) *LocalConf* ⇒ (-, -, -) *LocalConf* ⇒ *bool*
(**infixl** ⇝$_A$ *70*)
where
x ⇝$_A$ *y* ≡ (*x*, *y*) ∈ *eval*$_A$

abbreviation
eval-abv$_C$ :: (*'Com*$_C$, *'Var*$_C$, *'Val*) *LocalConf* ⇒ (-, -, -) *LocalConf* ⇒ *bool*
(**infixl** ⇝$_C$ *70*)
where
x ⇝$_C$ *y* ≡ (*x*, *y*) ∈ *eval*$_C$

definition
preserves-modes-mem :: (*'Com*$_A$, *'Var*$_A$, *'Val*, *'Com*$_C$, *'Var*$_C$) *state-relation* ⇒ *bool*
where
preserves-modes-mem $\mathcal{R}$ ≡
(∀ *c*$_A$ *mds*$_A$ *mem*$_A$ *c*$_C$ *mds*$_C$ *mem*$_C$. (⟨ *c*$_A$, *mds*$_A$, *mem*$_A$ ⟩$_A$, ⟨ *c*$_C$, *mds*$_C$, *mem*$_C$ ⟩$_C$) ∈ $\mathcal{R}$ ⟶
(∀ *x*$_A$. (*mem*$_A$ *x*$_A$) = (*mem*$_C$ (*var*$_C$-*of x*$_A$))) ∧
(∀ *m*. *var*$_C$-*of* ‘ *mds*$_A$ *m* = (*range var*$_C$-*of*) ∩ *mds*$_C$ *m*))

definition
mem$_A$-*of* :: (*'Var*$_C$, *'Val*) *Mem* ⇒ (*'Var*$_A$, *'Val*) *Mem*
where
mem$_A$-*of mem*$_C$ ≡ (λ*x*$_A$. (*mem*$_C$ (*var*$_C$-*of x*$_A$)))

definition
mds$_A$-*of* :: *'Var*$_C$ *Mds* ⇒ *'Var*$_A$ *Mds*
where
mds$_A$-*of mds*$_C$ ≡ (λ *m*. (*inv var*$_C$-*of*) ‘ (*range var*$_C$-*of* ∩ *mds*$_C$ *m*))

lemma *low-mds-eq-from-conc-to-abs*:
conc.low-mds-eq mds mem mem' ⟹ *abs.low-mds-eq* (*mds*$_A$-*of mds*) (*mem*$_A$-*of mem*) (*mem*$_A$-*of mem'*)

⟨*proof*⟩

definition
var$_A$*-of* :: '*Var*$_C$ ⇒ '*Var*$_A$
where
var$_A$*-of* ≡ *inv var*$_C$*-of*

lemma *preserves-modes-mem-mem*$_A$*-simp*:
(∀ *x*$_A$. (*mem*$_A$ *x*$_A$) = (*mem*$_C$ (*var*$_C$*-of x*$_A$))) ⟹
mem$_A$ = *mem*$_A$*-of mem*$_C$
⟨*proof*⟩

lemma *preserves-modes-mem-mds*$_A$*-simp*:
(∀ *m*. *var*$_C$*-of* ‘ *mds*$_A$ *m* = *range* (*var*$_C$*-of*) ∩ *mds*$_C$ *m*) ⟹
mds$_A$ = *mds*$_A$*-of mds*$_C$
⟨*proof*⟩

This version might be more useful. Not sure yet.

lemma *preserves-modes-mem-def2*:
preserves-modes-mem $\mathcal{R}$ =
(∀ *c*$_A$ *mds*$_A$ *mem*$_A$ *c*$_C$ *mds*$_C$ *mem*$_C$. (⟨ *c*$_A$, *mds*$_A$, *mem*$_A$ ⟩$_A$, ⟨ *c*$_C$, *mds*$_C$, *mem*$_C$ ⟩$_C$) ∈ $\mathcal{R}$ ⟶
mem$_A$ = *mem*$_A$*-of mem*$_C$ ∧
mds$_A$ = *mds*$_A$*-of mds*$_C$)
⟨*proof*⟩

definition
closed-others :: ('*Com*$_A$, '*Var*$_A$, '*Val*, '*Com*$_C$, '*Var*$_C$) *state-relation* ⇒ *bool*
where
closed-others $\mathcal{R}$ ≡
(∀ *c*$_A$ *c*$_C$ *mds*$_C$ *mem*$_C$ *mem*$_C$'. (⟨ *c*$_A$, *mds*$_A$*-of mds*$_C$, *mem*$_A$*-of mem*$_C$ ⟩$_A$, ⟨ *c*$_C$, *mds*$_C$, *mem*$_C$ ⟩$_C$) ∈ $\mathcal{R}$ ⟶
(∀ *x*. *mem*$_C$ *x* ≠ *mem*$_C$' *x* ⟶ ¬ *var-asm-not-written mds*$_C$ *x*) ⟶
(∀ *x*. *dma*$_C$ *mem*$_C$ *x* ≠ *dma*$_C$ *mem*$_C$' *x* ⟶ ¬ *var-asm-not-written mds*$_C$ *x*) ⟶
(⟨ *c*$_A$, *mds*$_A$*-of mds*$_C$, *mem*$_A$*-of mem*$_C$' ⟩$_A$, ⟨ *c*$_C$, *mds*$_C$, *mem*$_C$' ⟩$_C$) ∈ $\mathcal{R}$)

definition
stops$_C$ :: ('*Com*$_C$, '*Var*$_C$, '*Val*) *LocalConf* ⇒ *bool*
where
stops$_C$ *c* ≡ ∀ *c*'. ¬ (*c* ⇝$_C$ *c*')

lemmas *neval-induct* = *abs.neval.induct*[*consumes 1*, *case-names Zero Suc*]

lemma *strong-low-bisim-neval'*:
abs.neval c$_1$ *n c*$_n$ ⟹ (*c*$_1$,*c*$_1$') ∈ $\mathcal{R}_A$ ⟹ *snd* (*fst c*$_1$) = *snd* (*fst c*$_1$') ⟹ *abs.strong-low-bisim-mm* $\mathcal{R}_A$ ⟹
∃ *c*$_n$'. *abs.neval c*$_1$' *n c*$_n$' ∧ (*c*$_n$,*c*$_n$') ∈ $\mathcal{R}_A$ ∧ *snd* (*fst c*$_n$) = *snd* (*fst* (*c*$_n$'))

⟨*proof*⟩

lemma *strong-low-bisim-neval*:
abs.neval ⟨c_1,mds_1,mem_1⟩$_A$ *n* ⟨c_n,mds_n,mem_n⟩$_A$ ⟹ (⟨c_1,mds_1,mem_1⟩$_A$,⟨c_1′,mds_1,mem_1′⟩$_A$) ∈ $\mathcal{R}_A$ ⟹ *abs.strong-low-bisim-mm* $\mathcal{R}_A$ ⟹
∃ c_n′ mem_n′. *abs.neval* ⟨c_1′,mds_1,mem_1′⟩$_A$ *n* ⟨c_n′,mds_n,mem_n′⟩$_A$ ∧ (⟨c_n,mds_n,mem_n⟩$_A$,⟨c_n′,mds_n,mem_n′⟩$_A$) ∈ $\mathcal{R}_A$
⟨*proof*⟩

lemma *in-$\mathcal{R}$-dma′*:
assumes *preserves*: *preserves-modes-mem* $\mathcal{R}$
assumes *in-$\mathcal{R}$*: (⟨c_A,mds_A,mem_A⟩$_A$,⟨c_C,mds_C,mem_C⟩$_C$) ∈ $\mathcal{R}$
shows dma_A mem_A x_A = dma_C mem_C (var_C-*of* x_A)
⟨*proof*⟩

lemma *in-$\mathcal{R}$-dma*:
assumes *preserves*: *preserves-modes-mem* $\mathcal{R}$
assumes *in-$\mathcal{R}$*: (⟨c_A,mds_A,mem_A⟩$_A$,⟨c_C,mds_C,mem_C⟩$_C$) ∈ $\mathcal{R}$
shows dma_A mem_A = (dma_C mem_C ∘ var_C-*of*)
⟨*proof*⟩

definition
new-vars-private :: ('Com_A, 'Var_A, '*Val*, 'Com_C, 'Var_C) *state-relation* ⇒ *bool*
where
new-vars-private $\mathcal{R}$ ≡
(∀ c_{1A} mds_A mem_{1A} c_{1C} mds_C mem_{1C}.
(⟨ c_{1A}, mds_A, mem_{1A} ⟩$_A$, ⟨ c_{1C}, mds_C, mem_{1C} ⟩$_C$) ∈ $\mathcal{R}$ ⟶
(∀ c_{1C}′ mds_C′ mem_{1C}′. ⟨ c_{1C}, mds_C, mem_{1C} ⟩$_C$ ⇝$_C$ ⟨ c_{1C}′, mds_C′, mem_{1C}′ ⟩$_C$ ⟶
(∀ v_C. (mem_{1C}′ v_C ≠ mem_{1C} v_C ∨ dma_C mem_{1C}′ v_C < dma_C mem_{1C} v_C) ∧ v_C ∉ *range* var_C-*of* ⟶ v_C ∈ mds_C′ *AsmNoReadOrWrite*) ∧
(mds_C *AsmNoReadOrWrite* − (*range* var_C-*of*)) ⊆ (mds_C′ *AsmNoReadOrWrite* − (*range* var_C-*of*))))

lemma *not-less-eq-is-greater-Sec*:
(¬ *a* ≤ (*b*::*Sec*)) = (*a* > *b*)
⟨*proof*⟩

lemma *doesnt-have-mode*:
(*x* ∉ mds_A-*of* mds_C *m*) = (var_C-*of* *x* ∉ mds_C *m*)
⟨*proof*⟩

lemma *new-vars-private-does-the-thing*:
assumes *nice*: *new-vars-private* $\mathcal{R}$
assumes *in-$\mathcal{R}_1$*: (⟨ c_{1A}, mds_A-*of* mds_C, mem_A-*of* mem_{1C} ⟩$_A$, ⟨ c_{1C}, mds_C, mem_{1C} ⟩$_C$) ∈ $\mathcal{R}$
assumes *in-$\mathcal{R}_2$*: (⟨ c_{2A}, mds_A-*of* mds_C, mem_A-*of* mem_{2C} ⟩$_A$, ⟨ c_{2C}, mds_C, mem_{2C} ⟩$_C$) ∈ $\mathcal{R}$

assumes $step_{1C}$: $\langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C$
assumes $step_{2C}$: $\langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \rightsquigarrow_C \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C$
assumes *low-mds-eq*$_C$: *conc.low-mds-eq* mds_C mem_{1C} mem_{2C}
assumes *low-mds-eq*$_A$$'$: *abs.low-mds-eq* ($mds_A$-*of* mds_C') (mem_A-*of* mem_{1C}') (mem_A-*of* mem_{2C}')
shows *conc.low-mds-eq* mds_C' mem_{1C}' mem_{2C}'
⟨*proof*⟩

Perhaps surprisingly, we don't necessarily care whether the refinement preserves termination or divergence behaviour from the source to the target program. It can do whatever it likes, so long as it transforms two source programs that are low bisimilar (i.e. perform the same low actions at the same time), into two target ones that perform the same low actions at the same time.

Having the concrete step correspond to zero abstract ones is like expanding abstract code out (think e.g. of side-effect free expression evaluation). Having the concrete step correspond to more than one abstract step is like optimising out abstract code. But importantly, the optimisation needs to look the same for abstract-bisimilar code.

Additionally, we allow the instantiation of this theory to supply an arbitrary predicate that can be used to restrict our consideration to pairs of concrete steps that correspond to each other in terms of progress. This is particularly important for distinguishing between multiple concrete steps derived from the expansion of a single abstract step.

definition
secure-refinement :: ('Com_A, 'Var_A, '*Val*) *LocalConf rel* ⇒ ('Com_A, 'Var_A, '*Val*, 'Com_C, 'Var_C) *state-relation* ⇒
('Com_C, 'Var_C, '*Val*) *LocalConf rel* ⇒ *bool*
where
secure-refinement $\mathcal{R}_A$ $\mathcal{R}$ P ≡
closed-others $\mathcal{R}$ ∧
preserves-modes-mem $\mathcal{R}$ ∧
new-vars-private $\mathcal{R}$ ∧
conc.closed-glob-consistent P ∧
(∀ c_{1A} mds_A mem_{1A} c_{1C} mds_C mem_{1C}.
(⟨ c_{1A}, mds_A, mem_{1A} ⟩$_A$, ⟨ c_{1C}, mds_C, mem_{1C} ⟩$_C$) ∈ $\mathcal{R}$ ⟶
(∀ c_{1C}' mds_C' mem_{1C}'. ⟨ c_{1C}, mds_C, mem_{1C} ⟩$_C$ ⇝$_C$ ⟨ c_{1C}', mds_C', mem_{1C}' ⟩$_C$ ⟶
(∃ n c_{1A}' mds_A' mem_{1A}'. *abs.neval* ⟨ c_{1A}, mds_A, mem_{1A} ⟩$_A$ n ⟨ c_{1A}', mds_A', mem_{1A}' ⟩$_A$ ∧
(⟨ c_{1A}', mds_A', mem_{1A}' ⟩$_A$, ⟨ c_{1C}', mds_C', mem_{1C}' ⟩$_C$) ∈ $\mathcal{R}$ ∧
(∀ c_{2A} mem_{2A} c_{2C} mem_{2C} c_{2A}' mem_{2A}'.
(⟨ c_{1A}, mds_A, mem_{1A} ⟩$_A$, ⟨ c_{2A}, mds_A, mem_{2A} ⟩$_A$) ∈ $\mathcal{R}_A$ ∧
(⟨ c_{2A}, mds_A, mem_{2A} ⟩$_A$, ⟨ c_{2C}, mds_C, mem_{2C} ⟩$_C$) ∈ $\mathcal{R}$ ∧
(⟨ c_{1C}, mds_C, mem_{1C} ⟩$_C$, ⟨ c_{2C}, mds_C, mem_{2C} ⟩$_C$) ∈ P ∧
abs.neval ⟨ c_{2A}, mds_A, mem_{2A} ⟩$_A$ n ⟨ c_{2A}', mds_A', mem_{2A}' ⟩$_A$ ⟶
(∃ c_{2C}' mem_{2C}'. ⟨ c_{2C}, mds_C, mem_{2C} ⟩$_C$ ⇝$_C$ ⟨ c_{2C}', mds_C', mem_{2C}' ⟩$_C$

$\wedge$
$(\langle\ c_{2A}{}',\ mds_A{}',\ mem_{2A}{}'\ \rangle_A,\ \langle\ c_{2C}{}',\ mds_C{}',\ mem_{2C}{}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle\ c_{1C}{}',\ mds_C{}',\ mem_{1C}{}'\ \rangle_C,\ \langle\ c_{2C}{}',\ mds_C{}',\ mem_{2C}{}'\ \rangle_C) \in P)))))$

lemma *preserves-modes-memD*:
$[\![$*preserves-modes-mem* $\mathcal{R}$; $(\langle c_A,\ mds_A,\ mem_A\rangle_A,\ \langle c_C,\ mds_C,\ mem_C\rangle_C) \in \mathcal{R}]\!] \Longrightarrow$
$mem_A = mem_A$*-of* $mem_C \wedge mds_A = mds_A$*-of* mds_C
$\langle$*proof*$\rangle$

lemma *secure-refinement-def2*:
secure-refinement $\mathcal{R}_A\ \mathcal{R}\ P \equiv$
closed-others $\mathcal{R} \wedge$
preserves-modes-mem $\mathcal{R} \wedge$
new-vars-private $\mathcal{R} \wedge$
conc.closed-glob-consistent $P \wedge$
$(\forall\ c_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{1C}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}{}'\ mds_C{}'\ mem_{1C}{}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}{}',\ mds_C{}',\ mem_{1C}{}'\ \rangle_C \longrightarrow$
$(\exists\ n\ c_{1A}{}'.$ *abs.neval* $\langle\ c_{1A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{1C}\ \rangle_A\ n\ \langle\ c_{1A}{}',\ mds_A$*-of* $mds_C{}',\ mem_A$*-of* $mem_{1C}{}'\ \rangle_A \wedge$
$(\langle\ c_{1A}{}',\ mds_A$*-of* $mds_C{}',\ mem_A$*-of* $mem_{1C}{}'\ \rangle_A,\ \langle\ c_{1C}{}',\ mds_C{}',\ mem_{1C}{}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\forall c_{2A}\ c_{2C}\ mem_{2C}\ c_{2A}{}'\ mem_{2A}{}'.$
$(\langle\ c_{1A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{1C}\ \rangle_A,\ \langle\ c_{2A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{2C}\ \rangle_A) \in \mathcal{R}_A \wedge$
$(\langle\ c_{2A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{2C}\ \rangle_A,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in P \wedge$
abs.neval $\langle\ c_{2A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{2C}\ \rangle_A\ n\ \langle\ c_{2A}{}',\ mds_A$*-of* $mds_C{}',\ mem_{2A}{}'\ \rangle_A\ \longrightarrow$
$(\exists\ c_{2C}{}'\ mem_{2C}{}'.\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \rightsquigarrow_C \langle\ c_{2C}{}',\ mds_C{}',\ mem_{2C}{}'\ \rangle_C \wedge$
$(\langle\ c_{2A}{}',\ mds_A$*-of* $mds_C{}',\ mem_{2A}{}'\ \rangle_A,\ \langle\ c_{2C}{}',\ mds_C{}',\ mem_{2C}{}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle\ c_{1C}{}',\ mds_C{}',\ mem_{1C}{}'\ \rangle_C,\ \langle\ c_{2C}{}',\ mds_C{}',\ mem_{2C}{}'\ \rangle_C) \in P)))))$
$\langle$*proof*$\rangle$

lemma *extra-vars-are-not-control-vars*:
$x \notin$ *range* var_C*-of* $\Longrightarrow x \notin \mathcal{C}_C$
$\langle$*proof*$\rangle$

definition
R_C*-of* ::
$((('Com_A \times (Mode \Rightarrow 'Var_A\ set)) \times ('Var_A \Rightarrow 'Val)) \times$
$('Com_A \times (Mode \Rightarrow 'Var_A\ set)) \times ('Var_A \Rightarrow 'Val))\ set \Rightarrow$
$('Com_A,\ 'Var_A,\ 'Val,\ 'Com_C,\ 'Var_C)$ *state-relation* $\Rightarrow$
$((('Com_C \times (Mode \Rightarrow 'Var_C\ set)) \times ('Var_C \Rightarrow 'Val)) \times$
$('Com_C \times (Mode \Rightarrow 'Var_C\ set)) \times ('Var_C \Rightarrow 'Val))\ set \Rightarrow$

$(((\mathit{'Com}_C \times (\mathit{Mode} \Rightarrow \mathit{'Var}_C\ \mathit{set})) \times (\mathit{'Var}_C \Rightarrow \mathit{'Val})) \times$
$(\mathit{'Com}_C \times (\mathit{Mode} \Rightarrow \mathit{'Var}_C\ \mathit{set})) \times (\mathit{'Var}_C \Rightarrow \mathit{'Val}))\ \mathit{set}$
where
$R_C\text{-}\mathit{of}\ \mathcal{R}_A\ \mathcal{R}\ P \equiv \{(x,y).\ \exists\, x_A\ y_A.\ (x_A,x) \in \mathcal{R} \wedge (y_A,y) \in \mathcal{R} \wedge (x_A,y_A) \in \mathcal{R}_A \wedge$
$\mathit{snd}\ (\mathit{fst}\ x) = \mathit{snd}\ (\mathit{fst}\ y)$ — TODO: annoying to have to say $\wedge$
$\mathit{conc.low\text{-}mds\text{-}eq}\ (\mathit{snd}\ (\mathit{fst}\ x))\ (\mathit{snd}\ x)\ (\mathit{snd}\ y) \wedge$
$(x,y) \in P\}$

lemma *abs-low-mds-eq-dma$_C$-eq*:
assumes $\mathit{abs.low\text{-}mds\text{-}eq}\ (\mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds})\ (\mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{1C})\ \ (\mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{2C})$
shows $\mathit{dma}_C\ \mathit{mem}_{1C} = \mathit{dma}_C\ \mathit{mem}_{2C}$
⟨*proof*⟩

lemma *R_C-ofD*:
assumes *rr*: $\mathit{secure\text{-}refinement}\ \mathcal{R}_A\ \mathcal{R}\ P$
assumes *in-R*: $(\langle c_{1C},\ \mathit{mds}_C,\ \mathit{mem}_{1C}\rangle_C,\ \langle c_{2C},\ \mathit{mds}_C{}',\ \mathit{mem}_{2C}\rangle_C) \in R_C\text{-}\mathit{of}\ \mathcal{R}_A\ \mathcal{R}\ P$
shows
$(\exists\, c_{1A}\ c_{2A}.\ (\langle c_{1A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{1C}\rangle_A,\ \langle c_{1C},\ \mathit{mds}_C,\ \mathit{mem}_{1C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{2A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{2C}\rangle_A,\ \langle c_{2C},\ \mathit{mds}_C,\ \mathit{mem}_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{1C}\rangle_A,\ \langle c_{2A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{2C}\rangle_A) \in \mathcal{R}_A) \wedge$
$(\mathit{mds}_C{}' = \mathit{mds}_C) \wedge$
$\mathit{conc.low\text{-}mds\text{-}eq}\ \mathit{mds}_C\ \mathit{mem}_{1C}\ \mathit{mem}_{2C} \wedge$
$(\langle c_{1C},\ \mathit{mds}_C,\ \mathit{mem}_{1C}\rangle_C,\ \langle c_{2C},\ \mathit{mds}_C{}',\ \mathit{mem}_{2C}\rangle_C) \in P$
⟨*proof*⟩

lemma *R_C-ofI*:
$(\langle c_{1A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{1C}\rangle_A,\ \langle c_{1C},\ \mathit{mds}_C,\ \mathit{mem}_{1C}\rangle_C) \in \mathcal{R} \Longrightarrow$
$(\langle c_{2A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{2C}\rangle_A,\ \langle c_{2C},\ \mathit{mds}_C,\ \mathit{mem}_{2C}\rangle_C) \in \mathcal{R} \Longrightarrow$
$(\langle c_{1A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{1C}\rangle_A,\ \langle c_{2A},\ \mathit{mds}_A\text{-}\mathit{of}\ \mathit{mds}_C,\ \mathit{mem}_A\text{-}\mathit{of}\ \mathit{mem}_{2C}\rangle_A) \in \mathcal{R}_A \Longrightarrow$
$\mathit{conc.low\text{-}mds\text{-}eq}\ \mathit{mds}_C\ \mathit{mem}_{1C}\ \mathit{mem}_{2C} \Longrightarrow$
$(\langle c_{1C},\ \mathit{mds}_C,\ \mathit{mem}_{1C}\rangle_C,\ \langle c_{2C},\ \mathit{mds}_C,\ \mathit{mem}_{2C}\rangle_C) \in P \Longrightarrow$
$(\langle c_{1C},\ \mathit{mds}_C,\ \mathit{mem}_{1C}\rangle_C,\ \langle c_{2C},\ \mathit{mds}_C,\ \mathit{mem}_{2C}\rangle_C) \in R_C\text{-}\mathit{of}\ \mathcal{R}_A\ \mathcal{R}\ P$
⟨*proof*⟩

lemma *R_C-of-sym*:
assumes $\mathit{sym}\ \mathcal{R}_A$
assumes *P-sym*: $\mathit{sym}\ P$
assumes *rr*: $\mathit{secure\text{-}refinement}\ \mathcal{R}_A\ \mathcal{R}\ P$
assumes *mm*:
$\bigwedge c_1\ \mathit{mds}\ \mathit{mem}_1\ c_2\ \mathit{mds}\ \mathit{mem}_2.\ (\langle c_1,\ \mathit{mds},\ \mathit{mem}_1\rangle_A,\ \langle c_2,\ \mathit{mds},\ \mathit{mem}_2\rangle_A) \in \mathcal{R}_A \Longrightarrow$
$\mathit{abs.low\text{-}mds\text{-}eq}\ \mathit{mds}\ \mathit{mem}_1\ \mathit{mem}_2$
shows $\mathit{sym}\ (R_C\text{-}\mathit{of}\ \mathcal{R}_A\ \mathcal{R}\ P)$
⟨*proof*⟩

lemma R_C*-of-simp*:
assumes *rr*: *secure-refinement* $\mathcal{R}_A$ $\mathcal{R}$ P
shows $(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in R_C$*-of* $\mathcal{R}_A$ $\mathcal{R}$ P =
$((\exists\ c_{1A}\ c_{2A}.\ (\langle c_{1A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{1C}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{2A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{1C}\rangle_A,\ \langle c_{2A},\ mds_A$*-of* $mds_C,\ mem_A$*-of* $mem_{2C}\rangle_A) \in \mathcal{R}_A) \wedge$
conc.low-mds-eq mds_C mem_{1C} $mem_{2C} \wedge$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P)$
⟨*proof*⟩

definition
A_A*-of* :: $('Var_C,'Val)$ *adaptation* ⇒ $('Var_A,'Val)$ *adaptation*
where
A_A*-of* $A \equiv \lambda x_A.$ *case* A (var_C*-of* x_A) *of None* ⇒ *None* |
Some (v,v') ⇒ *Some* (v,v')

lemma $var\text{-}writable_A$:
¬ *var-asm-not-written* mds_C (var_C*-of* x) ⟹ ¬ *var-asm-not-written* (mds_A*-of* mds_C) x
⟨*proof*⟩

lemma A_A*-asm-mem*:
assumes A_C*-asm-mem*: ∀ x. *case* A_C x *of None* ⇒ *True*
| *Some* $(v,\ v')$ ⇒
$mem_{1C}\ x \neq v \vee mem_{2C}\ x \neq v' \longrightarrow$ ¬ *var-asm-not-written* mds_C x
shows *case* (A_A*-of* A_C) x *of None* ⇒ *True*
| *Some* $(v,\ v')$ ⇒
(mem_A*-of* mem_{1C}) $x \neq v \vee$ (mem_A*-of* mem_{2C}) $x \neq v' \longrightarrow$ ¬ *var-asm-not-written* (mds_A*-of* mds_C) x
⟨*proof*⟩

lemma dma_A*-adaptation-eq*:
dma_A ((mem_A*-of* mem_{1C}) [$\|_1$ A_A*-of* A_C]) $x_A = dma_C$ (mem_{1C} [$\|_1$ A_C]) (var_C*-of* x_A)
⟨*proof*⟩

lemma A_A*-asm-dma*:
assumes A_C*-asm-dma*: ∀ x. dma_C (mem_{1C} [$\|_1$ A_C]) $x \neq dma_C\ mem_{1C}\ x \longrightarrow$ ¬ *var-asm-not-written* mds_C x
shows dma_A ((mem_A*-of* mem_{1C}) [$\|_1$ (A_A*-of* A_C)]) $x_A \neq dma_A$ (mem_A*-of* mem_{1C}) $x_A \longrightarrow$ ¬ *var-asm-not-written* (mds_A*-of* mds_C) x_A
⟨*proof*⟩

lemma var_C*-of-in-*$\mathcal{C}_C$:

assumes $x_A \in \mathcal{C}_A$
shows $var_C\text{-}of\ x_A \in \mathcal{C}_C$
⟨*proof*⟩

lemma $doesnt\text{-}have\text{-}mode_C$:
$x \notin mds_A\text{-}of\ mds_C\ m \Longrightarrow var_C\text{-}of\ x \notin mds_C\ m$
⟨*proof*⟩

lemma $has\text{-}mode_A$: $var_C\text{-}of\ x \in mds_C\ m \Longrightarrow x \in mds_A\text{-}of\ mds_C\ m$
⟨*proof*⟩

lemma $A_A\text{-}sec$:
assumes $A_C\text{-}sec$: $\forall x.\ dma_C\ (mem_{1C}\ [\|_1\ A_C])\ x = Low \wedge (x \notin mds_C\ AsmNoReadOrWrite \vee x \in \mathcal{C}_C) \longrightarrow$
$mem_{1C}\ [\|_1\ A_C]\ x = mem_{2C}\ [\|_2\ A_C]\ x$
shows $dma_A\ ((mem_A\text{-}of\ mem_{1C})\ [\|_1\ A_A\text{-}of\ A_C])\ x = Low \wedge (x \notin mds_A\text{-}of\ mds_C\ AsmNoReadOrWrite \vee x \in \mathcal{C}_A) \longrightarrow$
$(mem_A\text{-}of\ mem_{1C})\ [\|_1\ A_A\text{-}of\ A_C]\ x = (mem_A\text{-}of\ mem_{2C})\ [\|_2\ A_A\text{-}of\ A_C]\ x$
⟨*proof*⟩

lemma $apply\text{-}adaptation_A$:
$(mem_A\text{-}of\ mem_{1C})\ [\|_1\ A_A\text{-}of\ A_C] = mem_A\text{-}of\ (mem_{1C}\ [\|_1\ A_C])$
$(mem_A\text{-}of\ mem_{1C})\ [\|_2\ A_A\text{-}of\ A_C] = mem_A\text{-}of\ (mem_{1C}\ [\|_2\ A_C])$
⟨*proof*⟩

lemma $R_C\text{-}of\text{-}closed\text{-}glob\text{-}consistent$:
assumes mm:
$\bigwedge c_1\ mds\ mem_1\ c_2\ mds\ mem_2.\ (\langle c_1,\ mds,\ mem_1\rangle_A,\ \langle c_2,\ mds,\ mem_2\rangle_A) \in \mathcal{R}_A \Longrightarrow$
$abs.low\text{-}mds\text{-}eq\ mds\ mem_1\ mem_2$
assumes cgc: $abs.closed\text{-}glob\text{-}consistent\ \mathcal{R}_A$
assumes rr: $secure\text{-}refinement\ \mathcal{R}_A\ \mathcal{R}\ P$
shows $conc.closed\text{-}glob\text{-}consistent\ (R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P)$
⟨*proof*⟩

lemma $R_C\text{-}of\text{-}local\text{-}preservation$:
assumes rr: $secure\text{-}refinement\ \mathcal{R}_A\ \mathcal{R}\ P$
assumes $bisim$: $abs.strong\text{-}low\text{-}bisim\text{-}mm\ \mathcal{R}_A$
assumes $in\text{-}R_C\text{-}of$: $(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
assumes $step_{1C}$: $\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C \rightsquigarrow_C \langle c_{1C}',\ mds_C',\ mem_{1C}'\rangle_C$
shows $\exists\, c_{2C}'\ mem_{2C}'.$
$\langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}',\ mds_C',\ mem_{2C}'\rangle_C \wedge$
$(\langle c_{1C}',\ mds_C',\ mem_{1C}'\rangle_C,\ \langle c_{2C}',\ mds_C',\ mem_{2C}'\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
⟨*proof*⟩

Security of the concrete system should follow straightforwardly from security

of the abstract one, via the compositionality theorem, presuming that the compiler also preserves the sound use of modes.

lemma *R_C-of-strong-low-bisim-mm*:
assumes *abs*: *abs.strong-low-bisim-mm* $\mathcal{R}_A$
assumes *rr*: *secure-refinement* $\mathcal{R}_A$ $\mathcal{R}$ *P*
assumes *P-sym*: *sym P*
shows *conc.strong-low-bisim-mm* (*R_C-of* $\mathcal{R}_A$ $\mathcal{R}$ *P*)
⟨*proof*⟩

2 A Simpler Proof Principle for General Compositional Refinement

Here we make use of the fact that the source language we are working in is assumed deterministic. This allows us to invert the direction of refinement and thereby to derive a simpler condition for secure compositional refinement.

The simpler condition rests on an ordinary definition of refinement, and has the user prove separately that the coupling invariant P is self-preserving. This allows proofs about coupling invariant properties to be disentangled from the proof of refinement itself.

Given a bisimulation $\mathcal{R}_A$, this definition captures the essence of the extra requirements on a refinement relation $\mathcal{R}$ needed to ensure that the refined program is also secure. These requirements are essentially that:

1. The enabledness of the compiled code depends only on Low abstract data;
2. The length of the abstract program to which a single step of the concrete program corresponds depends only on Low abstract data;
3. The coupling invariant is maintained.

The second requirement we express via the parameter *abs-steps* that, given an abstract and corresponding concrete configuration, yields the number of execution steps of the abstract configuration to which a single step of the concrete configuration corresponds.

Note that a more specialised version of this definition, fixing the coupling invariant P to be the one that relates all configurations with identical programs and mode states, appeared in Murray et al., CSF 2016. Here we generalise the theory to support a wider class of coupling invariants.

definition
simpler-refinement-safe
where
simpler-refinement-safe $\mathcal{R}_A$ $\mathcal{R}$ *P abs-steps* ≡

$\forall\, c_{1A}\ mds_A\ mem_{1A}\ c_{2A}\ mem_{2A}\ c_{1C}\ mds_C\ mem_{1C}\ c_{2C}\ mem_{2C}.\ (\langle c_{1A}, mds_A, mem_{1A}\rangle_A, \langle c_{2A}, mds_A, mem_{2A}\rangle_A) \in \mathcal{R}_A \wedge$
$(\langle c_{1A}, mds_A, mem_{1A}\rangle_A, \langle c_{1C}, mds_C, mem_{1C}\rangle_C) \in \mathcal{R} \wedge (\langle c_{2A}, mds_A, mem_{2A}\rangle_A, \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C}, mds_C, mem_{1C}\rangle_C, \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \in P \longrightarrow$
$(stops_C\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C = stops_C\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \wedge$
$(abs\text{-}steps\ \langle c_{1A}, mds_A, mem_{1A}\rangle_A\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C = abs\text{-}steps\ \langle c_{2A}, mds_A, mem_{2A}\rangle_A\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \wedge$
$(\forall\, mds_{1C}'\ mds_{2C}'\ mem_{1C}'\ mem_{2C}'\ c_{1C}'\ c_{2C}'.\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C \rightsquigarrow_C \langle c_{1C}', mds_{1C}', mem_{1C}'\rangle_C \wedge$
$\langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}', mds_{2C}', mem_{2C}'\rangle_C \longrightarrow$
$(\langle c_{1C}', mds_{1C}', mem_{1C}'\rangle_C, \langle c_{2C}', mds_{2C}', mem_{2C}'\rangle_C) \in P \wedge$
$mds_{1C}' = mds_{2C}')$

definition
secure-refinement-simpler
where
secure-refinement-simpler $\mathcal{R}_A\ \mathcal{R}\ P\ abs\text{-}steps \equiv$
closed-others $\mathcal{R} \wedge$
preserves-modes-mem $\mathcal{R} \wedge$
new-vars-private $\mathcal{R} \wedge$
simpler-refinement-safe $\mathcal{R}_A\ \mathcal{R}\ P\ abs\text{-}steps \wedge$
conc.closed-glob-consistent $P \wedge$
$(\forall\ c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A}, mds_A, mem_{1A}\ \rangle_A, \langle\ c_{1C}, mds_C, mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C}, mds_C, mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}', mds_C', mem_{1C}'\ \rangle_C \longrightarrow$
$(\exists\ c_{1A}'\ mds_A'\ mem_{1A}'.\ abs.neval\ \langle\ c_{1A}, mds_A, mem_{1A}\ \rangle_A\ (abs\text{-}steps\ \langle c_{1A}, mds_A, mem_{1A}\rangle_A\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C)\ \langle\ c_{1A}', mds_A', mem_{1A}'\ \rangle_A \wedge$
$(\langle\ c_{1A}', mds_A', mem_{1A}'\ \rangle_A, \langle\ c_{1C}', mds_C', mem_{1C}'\ \rangle_C) \in \mathcal{R})))$

lemma *secure-refinement-simpler*:
assumes *rrs*: *secure-refinement-simpler* $\mathcal{R}_A\ \mathcal{R}\ P\ abs\text{-}steps$
shows *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
⟨*proof*⟩

3 Simple Bisimulations and Simple Refinement

We derive the theory of simple refinements from Murray et al. CSF 2016 from the above *simpler* theory of secure refinement.

definition
bisim-simple
where
bisim-simple $\mathcal{R}_A \equiv \forall\, c_{1A}\ mds\ mem_{1A}\ c_{2A}\ mem_{2A}.\ (\langle c_{1A}, mds, mem_{1A}\rangle_A, \langle c_{2A}, mds, mem_{2A}\rangle_A) \in \mathcal{R}_A \longrightarrow$
$c_{1A} = c_{2A}$

definition
simple-refinement-safe
where
simple-refinement-safe $\mathcal{R}_A$ $\mathcal{R}$ *abs-steps* $\equiv$
$\forall\, c_A\ mds_A\ mem_{1A}\ mem_{2A}\ c_C\ mds_C\ mem_{1C}\ mem_{2C}.\ (\langle c_A,mds_A,mem_{1A}\rangle_A,\langle c_A,mds_A,mem_{2A}\rangle_A) \in \mathcal{R}_A \wedge$
$(\langle c_A,mds_A,mem_{1A}\rangle_A,\langle c_C,\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R} \wedge (\langle c_A,mds_A,mem_{2A}\rangle_A,\langle c_C,\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \longrightarrow$
$(stops_C\ \langle c_C,\ mds_C,\ mem_{1C}\rangle_C = stops_C\ \langle c_C,\ mds_C,\ mem_{2C}\rangle_C) \wedge$
$(abs\text{-}steps\ \langle c_A,mds_A,mem_{1A}\rangle_A\ \langle c_C,\ mds_C,\ mem_{1C}\rangle_C = abs\text{-}steps\ \langle c_A,mds_A,mem_{2A}\rangle_A\ \langle c_C,\ mds_C,\ mem_{2C}\rangle_C) \wedge$
$(\forall\, mds_{1C}'\ mds_{2C}'\ mem_{1C}'\ mem_{2C}'\ c_{1C}'\ c_{2C}'.\ \langle c_C,\ mds_C,\ mem_{1C}\rangle_C \rightsquigarrow_C \langle c_{1C}',\ mds_{1C}',\ mem_{1C}'\rangle_C \ \wedge$
$\langle c_C,\ mds_C,\ mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}',\ mds_{2C}',\ mem_{2C}'\rangle_C \longrightarrow$
$c_{1C}' = c_{2C}' \wedge mds_{1C}' = mds_{2C}')$

definition
secure-refinement-simple
where
secure-refinement-simple $\mathcal{R}_A$ $\mathcal{R}$ *abs-steps* $\equiv$
closed-others $\mathcal{R}$ $\wedge$
preserves-modes-mem $\mathcal{R}$ $\wedge$
new-vars-private $\mathcal{R}$ $\wedge$
simple-refinement-safe $\mathcal{R}_A$ $\mathcal{R}$ *abs-steps* $\wedge$
bisim-simple $\mathcal{R}_A$ $\wedge$
$(\forall\ c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C \longrightarrow$
$(\exists\ c_{1A}'\ mds_A'\ mem_{1A}'.\ abs.neval\ \langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A\ (abs\text{-}steps\ \langle c_{1A},mds_A,mem_{1A}\rangle_A\ \langle c_{1C},mds_C,mem_{1C}\rangle_C)\ \langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A \wedge$
$(\langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A,\ \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R})))$

definition
$\mathcal{I}simple$
where
$\mathcal{I}simple \equiv \{(\langle c,mds,mem\rangle_C,\langle c',mds',mem'\rangle_C) |\ c\ mds\ mem\ c'\ mds'\ mem'.\ c = c'\}$

lemma $\mathcal{I}simple\text{-}closed\text{-}glob\text{-}consistent$:
conc.closed-glob-consistent $\mathcal{I}simple$
$\langle proof\rangle$

lemma *secure-refinement-simple*:
assumes *srs*: *secure-refinement-simple* $\mathcal{R}_A$ $\mathcal{R}$ *abs-steps*
shows *secure-refinement-simpler* $\mathcal{R}_A$ $\mathcal{R}$ $\mathcal{I}simple$ *abs-steps*
$\langle proof\rangle$

4 Sound Mode Use Preservation

Prove that

> acquiring a mode on the concrete version of an abstract variable x, and then mapping the new concrete mode state to the corresponding abstract mode state,

is equivalent to

> first mapping the initial concrete mode state to its corresponding abstract mode state and then acquiring the mode on the abstract variable x.

This lemma essentially justifies why a concrete program doing *Acq* (var_C-*of* x) *SomeMode* is a the right way to implement the abstract program doing *Acq x SomeMode*.

lemma *mode-acquire-refinement-helper*:
mds_A-*of* (mds_C(*SomeMode* := *insert* (var_C-*of* x) (mds_C *SomeMode*))) =
(mds_A-*of* mds_C)(*SomeMode* := *insert* x (mds_A-*of* mds_C *SomeMode*))
⟨*proof*⟩

lemma *mode-release-refinement-helper*:
mds_A-*of* (mds_C(*SomeMode* := {$y \in mds_C$ *SomeMode*. $y \neq$ (var_C-*of* x)})) =
(mds_A-*of* mds_C)(*SomeMode* := {$y \in$ (mds_A-*of* mds_C *SomeMode*). $y \neq x$})
⟨*proof*⟩

definition
preserves-locally-sound-mode-use :: ('Com_A, 'Var_A, '*Val*, 'Com_C, 'Var_C) *state-relation* ⇒ *bool*
where
preserves-locally-sound-mode-use $\mathcal{R}$ ≡
∀ lc_A lc_C.
(*abs.locally-sound-mode-use* lc_A ∧ (lc_A, lc_C) ∈ $\mathcal{R}$ ⟶
conc.locally-sound-mode-use lc_C)

lemma *secure-refinement-loc-reach*:
assumes *rr*: *secure-refinement* $\mathcal{R}_A$ $\mathcal{R}$ P
assumes *in-*$\mathcal{R}$: (⟨c_A, mds_A, mem_A⟩$_A$, ⟨c_C, mds_C, mem_C⟩$_C$) ∈ $\mathcal{R}$
assumes *loc-reach*$_C$: ⟨c_C', mds_C', mem_C'⟩$_C$ ∈ *conc.loc-reach* ⟨c_C, mds_C, mem_C⟩$_C$
shows ∃ c_A' mds_A' mem_A'.
(⟨c_A', mds_A', mem_A'⟩$_A$, ⟨c_C', mds_C', mem_C'⟩$_C$) ∈ $\mathcal{R}$ ∧
⟨c_A', mds_A', mem_A'⟩$_A$ ∈ *abs.loc-reach* ⟨c_A, mds_A, mem_A⟩$_A$
⟨*proof*⟩

definition *preserves-local-guarantee-compliance* ::
('Com_A, 'Var_A, '*Val*, 'Com_C, 'Var_C) *state-relation* ⇒ *bool*
where

preserves-local-guarantee-compliance $\mathcal{R} \equiv$
$\forall\, cm_A\ mem_A\ cm_C\ mem_C.$
abs.respects-own-guarantees $cm_A \wedge$
$((cm_A,\ mem_A),\ (cm_C,\ mem_C)) \in \mathcal{R} \longrightarrow$
conc.respects-own-guarantees cm_C

lemma *preserves-local-guarantee-compliance-def2*:
preserves-local-guarantee-compliance $\mathcal{R} \equiv$
$\forall\, c_A\ mds_A\ mem_A\ c_C\ mds_C\ mem_C.$
abs.respects-own-guarantees $(c_A,\ mds_A) \wedge$
$(\langle c_A,\ mds_A,\ mem_A\rangle_A,\ \langle c_C,\ mds_C,\ mem_C\rangle_C) \in \mathcal{R} \longrightarrow$
conc.respects-own-guarantees $(c_C,\ mds_C)$
⟨*proof*⟩

lemma *locally-sound-mode-use-preservation*:
assumes *rr*: *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
assumes *preserves-guarantee-compliance*: *preserves-local-guarantee-compliance* $\mathcal{R}$
shows *preserves-locally-sound-mode-use* $\mathcal{R}$
⟨*proof*⟩

end

5 Refinement without changing the Memory Model

Here we define a locale which restricts the refinement to be between an abstract and concrete programs that share identical memory models: i,e. have the same set of variables. This allows us to derive simpler versions of the conditions that are likely to be easier to work with for initial experimentation.

locale *sifum-refinement-same-mem* =
abs: *sifum-security dma* $\mathcal{C}$*-vars* $\mathcal{C}$ $eval_A$ *some-val* +
conc: *sifum-security dma* $\mathcal{C}$*-vars* $\mathcal{C}$ $eval_C$ *some-val*
for *dma* :: (*'Var*,*'Val*) *Mem* ⇒ *'Var* ⇒ *Sec*
and $\mathcal{C}$*-vars* :: *'Var* ⇒ *'Var set*
and $\mathcal{C}$ :: *'Var set*
and $eval_A$ :: (*'Com*$_A$, *'Var*, *'Val*) *LocalConf rel*
and $eval_C$ :: (*'Com*$_C$, *'Var*, *'Val*) *LocalConf rel*
and *some-val* :: *'Val*

sublocale *sifum-refinement-same-mem* ⊆
gen-refine: *sifum-refinement dma dma* $\mathcal{C}$*-vars* $\mathcal{C}$*-vars* $\mathcal{C}$ $\mathcal{C}$ $eval_A$ $eval_C$ *some-val id*
⟨*proof*⟩

context *sifum-refinement-same-mem* **begin**

lemma [*simp*]:

gen-refine.new-vars-private $\mathcal{R}$
⟨*proof*⟩

definition
preserves-modes-mem :: ('Com_A, '*Var*, '*Val*, 'Com_C, '*Var*) *state-relation* ⇒ *bool*
where
preserves-modes-mem $\mathcal{R} \equiv$
$(\forall\ c_A\ mds_A\ mem_A\ c_C\ mds_C\ mem_C.\ (\langle\ c_A, mds_A, mem_A\ \rangle_A, \langle\ c_C, mds_C, mem_C\ \rangle_C) \in \mathcal{R} \longrightarrow$
$mem_A = mem_C \wedge mds_A = mds_C)$

definition
closed-others :: ('Com_A, '*Var*, '*Val*, 'Com_C, '*Var*) *state-relation* ⇒ *bool*
where
closed-others $\mathcal{R} \equiv$
$(\forall\ c_A\ mds\ mem\ c_C\ mem'.\ (\langle\ c_A, mds, mem\ \rangle_A, \langle\ c_C, mds, mem\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall x.\ mem\ x \neq mem'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds\ x) \longrightarrow$
$(\forall x.\ dma\ mem\ x \neq dma\ mem'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds\ x) \longrightarrow$
$(\langle\ c_A, mds, mem'\ \rangle_A, \langle\ c_C, mds, mem'\ \rangle_C) \in \mathcal{R})$

lemma [*simp*]:
gen-refine.mds_A*-of* $x = x$
⟨*proof*⟩

lemma [*simp*]:
gen-refine.mem_A*-of* $x = x$
⟨*proof*⟩

lemma [*simp*]:
preserves-modes-mem $\mathcal{R} \Longrightarrow$
gen-refine.closed-others $\mathcal{R}$ = *closed-others* $\mathcal{R}$
⟨*proof*⟩

lemma [*simp*]:
gen-refine.preserves-modes-mem $\mathcal{R}$ = *preserves-modes-mem* $\mathcal{R}$
⟨*proof*⟩

definition
secure-refinement :: ('Com_A, '*Var*, '*Val*) *LocalConf rel* ⇒ ('Com_A, '*Var*, '*Val*, 'Com_C, '*Var*) *state-relation* ⇒
('Com_C, '*Var*, '*Val*) *LocalConf rel* ⇒ *bool*
where
secure-refinement $\mathcal{R}_A\ \mathcal{R}\ P \equiv$
closed-others $\mathcal{R} \wedge$
preserves-modes-mem $\mathcal{R} \wedge$
conc.closed-glob-consistent $P \wedge$
$(\forall\ c_{1A}\ mds\ mem_1\ c_{1C}.$
$(\langle\ c_{1A}, mds, mem_1\ \rangle_A, \langle\ c_{1C}, mds, mem_1\ \rangle_C) \in \mathcal{R} \longrightarrow$

$(\forall\ c_{1C}'\ mds'\ mem_1'.\ \langle\ c_{1C},\ mds,\ mem_1\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}',\ mds',\ mem_1'\ \rangle_C \longrightarrow$
$(\exists\ n\ c_{1A}'.\ abs.neval\ \langle\ c_{1A},\ mds,\ mem_1\ \rangle_A\ n\ \langle\ c_{1A}',\ mds',\ mem_1'\ \rangle_A \wedge$
$(\langle\ c_{1A}',\ mds',\ mem_1'\ \rangle_A, \langle\ c_{1C}',\ mds',\ mem_1'\ \rangle_C) \in \mathcal{R} \wedge$
$(\forall\ c_{2A}\ mem_2\ c_{2C}\ c_{2A}'\ mem_2'.$
$(\langle\ c_{1A},\ mds,\ mem_1\ \rangle_A, \langle\ c_{2A},\ mds,\ mem_2\ \rangle_A) \in \mathcal{R}_A \wedge$
$(\langle\ c_{2A},\ mds,\ mem_2\ \rangle_A, \langle\ c_{2C},\ mds,\ mem_2\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C},\ mds,\ mem_1\rangle_C, \langle c_{2C},\ mds,\ mem_2\rangle_C) \in P\ \wedge$
$abs.neval\ \langle\ c_{2A},\ mds,\ mem_2\ \rangle_A\ n\ \langle\ c_{2A}',\ mds',\ mem_2'\ \rangle_A\ \longrightarrow$
$(\exists\ c_{2C}'\ .\ \langle\ c_{2C},\ mds,\ mem_2\ \rangle_C \rightsquigarrow_C \langle\ c_{2C}',\ mds',\ mem_2'\ \rangle_C \wedge$
$(\langle\ c_{2A}',\ mds',\ mem_2'\ \rangle_A, \langle\ c_{2C}',\ mds',\ mem_2'\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C}',\ mds',\ mem_1'\rangle_C, \langle c_{2C}',\ mds',\ mem_2'\rangle_C) \in P\ )))))$

lemma *preserves-modes-memD*:
preserves-modes-mem $\mathcal{R} \Longrightarrow$
$(\langle\ c_A,\ mds_A,\ mem_A\ \rangle_A, \langle\ c_C,\ mds_C,\ mem_C\ \rangle_C) \in \mathcal{R} \Longrightarrow$
$mem_A = mem_C \wedge mds_A = mds_C$
⟨*proof*⟩

lemma [*simp*]:
gen-refine.secure-refinement $\mathcal{R}_A\ \mathcal{R}\ P$ = *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
⟨*proof*⟩

lemma R_C*-of-strong-low-bisim-mm*:
assumes *abs*: *abs.strong-low-bisim-mm* $\mathcal{R}_A$
assumes *rr*: *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
assumes *P-sym*: *sym* P
shows *conc.strong-low-bisim-mm* (*gen-refine.*R_C*-of* $\mathcal{R}_A\ \mathcal{R}\ P$)
⟨*proof*⟩

end

context *sifum-refinement* **begin**
lemma *use-secure-refinement-helper*:
secure-refinement $\mathcal{R}_A\ \mathcal{R}\ P \Longrightarrow$
$((cm_A,mem_A),(cm_C,mem_C)) \in \mathcal{R} \Longrightarrow (cm_C,mem_C) \rightsquigarrow_C (cm_C',mem_C') \Longrightarrow$
$(\exists\ cm_A'\ mem_A'\ n.\ abs.neval\ (cm_A,mem_A)\ n\ (cm_A',mem_A') \wedge$
$((cm_A',mem_A'), (cm_C',mem_C')) \in \mathcal{R})$
⟨*proof*⟩

lemma *closed-othersD*:
closed-others $\mathcal{R} \Longrightarrow$
$(\langle c_A,\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_C\rangle_A, \langle c_C,\ mds_C,\ mem_C\rangle_C) \in \mathcal{R} \Longrightarrow$
$(\bigwedge x.\ mem_C'\ x \neq mem_C\ x \vee dma_C\ mem_C'\ x \neq dma_C\ mem_C\ x \Longrightarrow \neg$
var-asm-not-written $mds_C\ x) \Longrightarrow$
$(\langle c_A,\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_C'\rangle_A, \langle c_C,\ mds_C,\ mem_C'\rangle_C) \in \mathcal{R}$
⟨*proof*⟩
end

record $('a,\ 'Val,\ 'Var_C,\ 'Com_C,\ 'Var_A,\ 'Com_A)$ *componentwise-refinement* =

priv-mem :: *'Var$_C$ set*
$\mathcal{R}_A$-rel :: (*'Com$_A$*, *'Var$_A$*, *'Val*) *LocalConf rel*
$\mathcal{R}$-rel :: (*'Com$_A$*, *'Var$_A$*, *'Val*, *'Com$_C$*, *'Var$_C$*) *state-relation*
P-rel :: (*'Com$_C$*, *'Var$_C$*, *'Val*) *LocalConf rel*

6 Whole System Refinement

A locale to capture componentwise refinement of an entire system.

locale *sifum-refinement-sys* =
sifum-refinement dma$_A$ dma$_C$ $\mathcal{C}$-vars$_A$ $\mathcal{C}$-vars$_C$ $\mathcal{C}_A$ $\mathcal{C}_C$ eval$_A$ eval$_C$ some-val var$_C$-of
for *dma$_A$* :: (*'Var$_A$*,*'Val*) *Mem* ⇒ *'Var$_A$* ⇒ *Sec*
and *dma$_C$* :: (*'Var$_C$*,*'Val*) *Mem* ⇒ *'Var$_C$* ⇒ *Sec*
and *$\mathcal{C}$-vars$_A$* :: *'Var$_A$* ⇒ *'Var$_A$ set*
and *$\mathcal{C}$-vars$_C$* :: *'Var$_C$* ⇒ *'Var$_C$ set*
and *$\mathcal{C}_A$* :: *'Var$_A$ set*
and *$\mathcal{C}_C$* :: *'Var$_C$ set*
and *eval$_A$* :: (*'Com$_A$*, *'Var$_A$*, *'Val*) *LocalConf rel*
and *eval$_C$* :: (*'Com$_C$*, *'Var$_C$*, *'Val*) *LocalConf rel*
and *some-val* :: *'Val*
and *var$_C$-of* :: *'Var$_A$* ⇒ *'Var$_C$* +
fixes *cms* :: (*'a*::*wellorder*, *'Val*, *'Var$_C$*, *'Com$_C$*, *'Var$_A$*, *'Com$_A$*) *component-wise-refinement list*
fixes *priv-mem$_C$* :: *'Var$_C$ set list*
defines *priv-mem$_C$-def*: *priv-mem$_C$* ≡ *map priv-mem cms*
assumes *priv-mem-disjoint*: *i* < *length cms* ⟹ *j* < *length cms* ⟹ *i* ≠ *j* ⟹ *priv-mem$_C$* ! *i* ∩ *priv-mem$_C$* ! *j* = {}
assumes *new-vars-priv*: − *range var$_C$-of* = ⋃ (*set priv-mem$_C$*)
assumes *new-privs-preserved*: ⟨*c*, *mds*, *mem*⟩$_C$ ⇝$_C$ ⟨*c'*, *mds'*, *mem'*⟩$_C$ ⟹ *x* ∉ *range var$_C$-of* ⟹
(*x* ∈ *mds m*) = (*x* ∈ *mds' m*)
assumes *secure-refinements*:
i < *length cms* ⟹ *secure-refinement* (*$\mathcal{R}_A$-rel* (*cms* ! *i*)) (*$\mathcal{R}$-rel* (*cms* ! *i*)) (*P-rel* (*cms* ! *i*))
assumes *local-guarantee-preservation*:
i < *length cms* ⟹ *preserves-local-guarantee-compliance* (*$\mathcal{R}$-rel* (*cms* ! *i*))
assumes *bisims*:
i < *length cms* ⟹ *abs.strong-low-bisim-mm* (*$\mathcal{R}_A$-rel* (*cms* ! *i*))
assumes *Ps-sym*:
⋀*a b*. *i* < *length cms* ⟹ *sym* (*P-rel* (*cms* ! *i*))
assumes *Ps-refl-on-low-mds-eq*:
i < *length cms* ⟹ *conc.low-mds-eq mds$_C$ mem$_C$ mem$_C$'* ⟹ (⟨*c$_C$*, *mds$_C$*, *mem$_C$*⟩$_C$, ⟨*c$_C$*, *mds$_C$*, *mem$_C$'*⟩$_C$) ∈ (*P-rel* (*cms* ! *i*))

context *sifum-security* **begin**
lemma *neval-modifies-helper*:
assumes *nevaln*: *neval lcn m lcn'*

assumes *lcn-def*: $lcn = (cms\ !\ n,\ mem)$
assumes *lcn'-def*: $lcn' = (cmn',\ mem')$
assumes *len*: $n < length\ cms$
assumes *modified*: $mem\ x \neq mem'\ x \vee dma\ mem\ x \neq dma\ mem'\ x$
shows $\exists k\ cmn''\ mem''\ cmn'''\ mem'''.\ k < m \wedge neval\ (cms\ !\ n,mem)\ k\ (cmn'',mem'') \wedge$
$(cmn'',mem'') \rightsquigarrow (cmn''',\ mem''') \wedge$
$(mem''\ x \neq mem'''\ x \vee dma\ mem''\ x \neq dma\ mem'''\ x)$
⟨*proof*⟩

lemma *neval-sched-Nil* [*simp*]:
$(cms,\ mem) \rightarrow_{[]} (cms,\ mem)$
⟨*proof*⟩

lemma *reachable-mode-states-refl*:
$map\ snd\ cms \in reachable\text{-}mode\text{-}states\ (cms,\ mem)$
⟨*proof*⟩

lemma *neval-reachable-mode-states*:
assumes *neval*: $neval\ lc\ n\ lc'$
assumes *lc-def*: $lc = (cms\ !\ k,\ mem)$
assumes *len*: $k < length\ cms$
shows $map\ snd\ (cms[k := (fst\ lc')]) \in reachable\text{-}mode\text{-}states\ (cms,\ mem)$
⟨*proof*⟩

lemma *meval-sched-sound-mode-use*:
$sound\text{-}mode\text{-}use\ gc \Longrightarrow meval\text{-}sched\ sched\ gc\ gc' \Longrightarrow sound\text{-}mode\text{-}use\ gc'$
⟨*proof*⟩

lemma *neval-meval*:
$neval\ lcn\ k\ lcn' \Longrightarrow n < length\ cms \Longrightarrow lcn = (cms\ !\ n,mem) \Longrightarrow lcn' = (cmn',\ mem') \Longrightarrow$
$meval\text{-}sched\ (replicate\ k\ n)\ (cms,mem)\ (cms[n := cmn'],mem')$
⟨*proof*⟩

lemma *meval-sched-app*:
$meval\text{-}sched\ as\ gc\ gc' \Longrightarrow meval\text{-}sched\ bs\ gc'\ gc'' \Longrightarrow meval\text{-}sched\ (as@bs)\ gc\ gc''$
⟨*proof*⟩

end

context *sifum-refinement-sys* **begin**

lemma *conc-respects-priv*:
assumes *xnin*: $x_C \notin range\ var_C\text{-}of$
assumes *modified*$_C$: $mem_C\ x_C \neq mem_C'\ x_C \vee dma_C\ mem_C\ x_C \neq dma_C\ mem_C'\ x_C$
assumes *eval*$_C$: $(cms_C\ !\ n,\ mem_C) \rightsquigarrow_C (cm_C n',\ mem_C')$

assumes *in-$\mathcal{R}n$*: $((cms_A\ !\ n,\ mem_A),\ cms_C\ !\ n,\ mem_C) \in \mathcal{R}n$
assumes *preserves*: *preserves-local-guarantee-compliance* $\mathcal{R}n$
assumes *sound-mode-use$_A$*: *abs.sound-mode-use* $(cms_A,\ mem_A)$
assumes *nlen*: $n < length\ cms$
assumes *len-eq*: $length\ cms_A = length\ cms$
assumes *len-eq'*: $length\ cms_C = length\ cms$
shows $x_C \notin (snd\ (cms_C\ !\ n))\ GuarNoWrite \wedge x_C \notin (snd\ (cms_C\ !\ n))\ GuarNoReadOrWrite$
⟨*proof*⟩

lemma *modified-variables-are-not-assumed-not-written*:
fixes $cms_A\ mem_A\ cms_C\ mem_C\ cm_C n'\ mem_C'\ \mathcal{R}n\ cm_A n'\ mem_A'\ m_A\ \mathcal{R}i$
assumes *sound-mode-use$_A$*: *abs.sound-mode-use* $(cms_A,\ mem_A)$
assumes *pmmn*: *preserves-modes-mem* $\mathcal{R}n$
assumes *in-$\mathcal{R}n$*: $((cms_A\ !\ n,\ mem_A),\ (cms_C\ !\ n,\ mem_C)) \in \mathcal{R}n$
assumes *pmmi*: *preserves-modes-mem* $\mathcal{R}i$
assumes *in-$\mathcal{R}i$*: $((cms_A\ !\ i,\ mem_A),\ (cms_C\ !\ i,\ mem_C)) \in \mathcal{R}i$
assumes *nlen*: $n < length\ cms$
assumes *len$_A$*: $length\ cms_A = length\ cms$
assumes *len$_C$*: $length\ cms_C = length\ cms$
assumes *priv-is-asm-priv*: $\bigwedge i.\ i < length\ cms \Longrightarrow priv\text{-}mem_C\ !\ i \subseteq snd\ (cms_C\ !\ i)\ AsmNoReadOrWrite$
assumes *priv-is-guar-priv*: $\bigwedge i\ j.\ i < length\ cms \Longrightarrow j < length\ cms \Longrightarrow i \neq j \Longrightarrow priv\text{-}mem_C\ !\ i \subseteq snd\ (cms_C\ !\ j)\ GuarNoReadOrWrite$
assumes *new-asms-only-for-priv*: $\bigwedge i.\ i < length\ cms \Longrightarrow (snd\ (cms_C\ !\ i)\ AsmNoReadOrWrite \cup snd\ (cms_C\ !\ i)\ AsmNoWrite) \cap (-\ range\ var_C\text{-}of) \subseteq priv\text{-}mem_C\ !\ i$
assumes *eval$_C n$*: $(cms_C\ !\ n, mem_C) \rightsquigarrow_C (cm_C n', mem_C')$
assumes *neval$_A n$*: *abs.neval* $(cms_A\ !\ n, mem_A)\ m_A\ (cm_A n', mem_A')$
assumes *in-$\mathcal{R}n'$*: $((cm_A n', mem_A'), (cm_C n', mem_C')) \in \mathcal{R}n$
assumes *modified$_C$*: $mem_C\ x_C \neq mem_C'\ x_C \vee dma_C\ mem_C\ x_C \neq dma_C\ mem_C'\ x_C$
assumes *neq*: $i \neq n$
assumes *ilen*: $i < length\ cms$
assumes *preserves*: *preserves-local-guarantee-compliance* $\mathcal{R}n$
shows $\neg$ *var-asm-not-written* $(snd\ (cms_C\ !\ i))\ x_C$
⟨*proof*⟩

definition
priv-is-asm-priv :: *'Var$_C$ Mds list* ⇒ *bool*
where
priv-is-asm-priv $mdss_C \equiv \forall i < length\ cms.\ priv\text{-}mem_C\ !\ i \subseteq (mdss_C\ !\ i)\ AsmNoReadOrWrite$

definition
priv-is-guar-priv :: *'Var$_C$ Mds list* ⇒ *bool*
where
priv-is-guar-priv $mdss_C \equiv$
$\forall i < length\ cms.\ (\forall j < length\ cms.\ i \neq j \longrightarrow priv\text{-}mem_C\ !\ i \subseteq (mdss_C\ !\ j)$

GuarNoReadOrWrite)

definition
new-asms-only-for-priv :: *'Var$_C$ Mds list* ⇒ *bool*
where
new-asms-only-for-priv mdss$_C$ ≡
∀ *i* < *length cms*.
((*mdss$_C$* ! *i*) *AsmNoReadOrWrite* ∪ (*mdss$_C$* ! *i*) *AsmNoWrite*) ∩ (− *range var$_C$-of*) ⊆ *priv-mem$_C$* ! *i*

definition
new-asms-NoReadOrWrite-only :: *'Var$_C$ Mds list* ⇒ *bool*
where
new-asms-NoReadOrWrite-only mdss$_C$ ≡
∀ *i* < *length cms*.
(*mdss$_C$* ! *i*) *AsmNoWrite* ∩ (− *range var$_C$-of*) = {}

definition
modes-respect-priv :: *'Var$_C$ Mds list* ⇒ *bool*
where
modes-respect-priv mdss$_C$ ≡ *priv-is-asm-priv mdss$_C$* ∧ *priv-is-guar-priv mdss$_C$* ∧
new-asms-only-for-priv mdss$_C$ ∧
new-asms-NoReadOrWrite-only mdss$_C$

definition
ignores-old-vars :: (*'Var$_C$ Mds list* ⇒ *bool*) ⇒ *bool*
where
ignores-old-vars P ≡ ∀ *mdss mdss'*. *length mdss* = *length mdss'* ∧ *length mdss'* = *length cms* ⟶
(*map* (λ*x m*. *x m* ∩ (− *range var$_C$-of*)) *mdss*) = (*map* (λ*x m*. *x m* ∩ (− *range var$_C$-of*)) *mdss'*) ⟶
P mdss = *P mdss'*

lemma *ignores-old-vars-conj*:
assumes *Rdef*: (⋀*x*. *R x* = (*P x* ∧ *Q x*))
assumes *iP*: *ignores-old-vars P*
assumes *iQ*: *ignores-old-vars Q*
shows *ignores-old-vars R*
⟨*proof*⟩

lemma *nth-map-eq'*:
length xs = *length ys* ⟹ *map f xs* = *map g ys* ⟹ *i* < *length xs* ⟹ *f* (*xs* ! *i*) = *g* (*ys* ! *i*)
⟨*proof*⟩

lemma *nth-map-eq*:
map f xs = *map g ys* ⟹ *i* < *length xs* ⟹ *f* (*xs* ! *i*) = *g* (*ys* ! *i*)

⟨*proof*⟩

lemma *nth-in-Union-over-set*:
$i < length\ xs \Longrightarrow xs\ !\ i \subseteq \bigcup (set\ xs)$
⟨*proof*⟩

lemma *priv-are-new-vars*:
$x \in priv\text{-}mem_C\ !\ i \Longrightarrow i < length\ cms \Longrightarrow x \notin range\ var_C\text{-}of$
⟨*proof*⟩

lemma *priv-is-asm-priv-ignores-old-vars*:
ignores-old-vars priv-is-asm-priv
⟨*proof*⟩

lemma *priv-is-guar-priv-ignores-old-vars*:
ignores-old-vars priv-is-guar-priv
⟨*proof*⟩

lemma *new-asms-only-for-priv-ignores-old-vars*:
ignores-old-vars new-asms-only-for-priv
⟨*proof*⟩

lemma *new-asms-NoReadOrWrite-only-ignores-old-vars*:
ignores-old-vars new-asms-NoReadOrWrite-only
⟨*proof*⟩

lemma *modes-respect-priv-ignores-old-vars*:
ignores-old-vars modes-respect-priv
⟨*proof*⟩

lemma *ignores-old-varsD*:
$ignores\text{-}old\text{-}vars\ P \Longrightarrow length\ mdss = length\ mdss' \Longrightarrow length\ mdss' = length\ cms \Longrightarrow$
$(map\ (\lambda x\ m.\ x\ m \cap (-\ range\ var_C\text{-}of))\ mdss) = (map\ (\lambda x\ m.\ x\ m \cap (-\ range\ var_C\text{-}of))\ mdss') \Longrightarrow$
$P\ mdss = P\ mdss'$
⟨*proof*⟩

lemma *new-privs-preserved'*:
$\langle c,\ mds,\ mem\rangle_C \rightsquigarrow_C \langle c',\ mds',\ mem'\rangle_C \Longrightarrow (mds\ m \cap (-\ range\ var_C\text{-}of)) = (mds'\ m \cap (-\ range\ var_C\text{-}of))$
⟨*proof*⟩

lemma *map-nth-eq*:
$length\ xs = length\ ys \Longrightarrow (\bigwedge i.\ i < length\ xs \Longrightarrow f\ (xs\ !\ i) = g\ (ys\ !\ i)) \Longrightarrow$
$map\ f\ xs = map\ g\ ys$
⟨*proof*⟩

lemma *ignores-old-vars-conc-meval*:

assumes *ignores*: *ignores-old-vars P*
assumes *meval*: *conc.meval-abv* gc_C *n* gc_C'
assumes *len-eq*: *length* (*fst* gc_C) = *length cms*
shows *P* (*map snd* (*fst* gc_C)) = *P* (*map snd* (*fst* gc_C'))
⟨*proof*⟩

lemma *ignores-old-vars-conc-meval-sched*:
assumes *ignores*: *ignores-old-vars P*
assumes *meval-sched*: *conc.meval-sched sched* gc_C gc_C'
assumes *len-eq*: *length* (*fst* gc_C) = *length cms*
shows *P* (*map snd* (*fst* gc_C)) = *P* (*map snd* (*fst* gc_C'))
⟨*proof*⟩

lemma *meval-sched-modes-respect-priv*:
conc.meval-sched sched gc_C $gc_C' \Longrightarrow$ *length* (*fst* gc_C) = *length cms* $\Longrightarrow$
modes-respect-priv (*map snd* (*fst* gc_C)) $\Longrightarrow$
modes-respect-priv (*map snd* (*fst* gc_C'))
⟨*proof*⟩

lemma *meval-modes-respect-priv*:
conc.meval-abv gc_C *n* $gc_C' \Longrightarrow$ *length* (*fst* gc_C) = *length cms* $\Longrightarrow$
modes-respect-priv (*map snd* (*fst* gc_C)) $\Longrightarrow$
modes-respect-priv (*map snd* (*fst* gc_C'))
⟨*proof*⟩

lemma *traces-refinement*:
$\bigwedge gc_C\ gc_C'\ sched_C\ gc_A$. *conc.meval-sched* $sched_C$ gc_C $gc_C' \Longrightarrow$
length (*fst* gc_A) = *length cms* $\Longrightarrow$ *length* (*fst* gc_C) = *length cms* $\Longrightarrow$
($\bigwedge i$. $i <$ *length cms* $\Longrightarrow$ ((*fst* gc_A ! i, *snd* gc_A), (*fst* gc_C ! i, *snd* gc_C)) $\in \mathcal{R}$-*rel* (*cms* ! i)) $\Longrightarrow$
abs.sound-mode-use $gc_A \Longrightarrow$ *modes-respect-priv* (*map snd* (*fst* gc_C)) $\Longrightarrow$
$\exists\, sched_A\ gc_A'$. *abs.meval-sched* $sched_A$ gc_A $gc_A' \wedge$
($\forall i$. $i <$ *length cms* $\longrightarrow$ ((*fst* gc_A' ! i, *snd* gc_A'), (*fst* gc_C' ! i, *snd* gc_C')) $\in \mathcal{R}$-*rel* (*cms* ! i)) $\wedge$
abs.sound-mode-use gc_A'
⟨*proof*⟩

end

context *sifum-security* **begin**

definition
restrict-modes :: *'Var Mds list* ⇒ *'Var set* ⇒ *'Var Mds list*
where
restrict-modes mdss X ≡ *map* (λ*mds m*. *mds m* ∩ *X*) *mdss*

lemma *restrict-modes-length* [*simp*]:
length (*restrict-modes mdss X*) = *length mdss*

⟨*proof*⟩

lemma *compatible-modes-by-case-distinction*:
assumes *compat-X*: *compatible-modes* (*restrict-modes mdss X*)
assumes *compat-compX*: *compatible-modes* (*restrict-modes mdss* (−*X*))
shows *compatible-modes mdss*
⟨*proof*⟩

lemma *in-restrict-modesD*:
i < *length mdss* ⟹ *x* ∈ ((*restrict-modes mdss X*) ! *i*) *m* ⟹ *x* ∈ *X* ∧ *x* ∈ (*mdss* ! *i*) *m*
⟨*proof*⟩

lemma *in-restrict-modesI*:
i < *length mdss* ⟹ *x* ∈ *X* ⟹ *x* ∈ (*mdss* ! *i*) *m* ⟹ *x* ∈ ((*restrict-modes mdss X*) ! *i*) *m*
⟨*proof*⟩

lemma *meval-sched-length*:
meval-sched sched gc gc′ ⟹ *length* (*fst gc′*) = *length* (*fst gc*)
⟨*proof*⟩

end

context *sifum-refinement-sys* **begin**

lemma *compatible-modes-old-vars*:
assumes *compatible-modes*$_A$: *abs.compatible-modes* (*map snd cms*$_A$)
assumes *len*$_A$: *length cms*$_A$ = *length cms*
assumes *len*$_C$: *length cms*$_C$ = *length cms*
assumes *in-*$\mathcal{R}$: (∀ *i*<*length cms*. ((*cms*$_A$! *i*, *mem*$_A$), *cms*$_C$! *i*, *mem*$_C$) ∈ $\mathcal{R}$-*rel* (*cms* ! *i*))
shows *conc.compatible-modes* (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$-*of*))
⟨*proof*⟩

lemma *compatible-modes-new-vars*:
length mdss = *length cms* ⟹ *modes-respect-priv mdss* ⟹ *conc.compatible-modes* (*conc.restrict-modes mdss* (− *range var*$_C$-*of*))
⟨*proof*⟩

lemma *sound-mode-use-preservation*:
⋀*gc*$_C$ *gc*$_A$.
length (*fst gc*$_A$) = *length cms* ⟹ *length* (*fst gc*$_C$) = *length cms* ⟹
(⋀*i*. *i* < *length cms* ⟹ ((*fst gc*$_A$! *i*, *snd gc*$_A$), (*fst gc*$_C$! *i*, *snd gc*$_C$)) ∈ $\mathcal{R}$-*rel* (*cms* ! *i*)) ⟹
abs.sound-mode-use gc$_A$ ⟹ *modes-respect-priv* (*map snd* (*fst gc*$_C$)) ⟹
conc.sound-mode-use gc$_C$

⟨*proof*⟩

lemma *refined-prog-secure*:
assumes len_A [*simp*]: *length* cms_C = *length cms*
assumes len_C [*simp*]: *length* cms_A = *length cms*
assumes *in-*$\mathcal{R}$: ($\bigwedge i\ mem_C$. $i <$ *length cms* $\Longrightarrow$ ((cms_A ! i,mem_A-*of* mem_C),(cms_C ! i, mem_C)) $\in$ $\mathcal{R}$-*rel* (*cms* ! i))
assumes *in-*$\mathcal{R}_A$: ($\bigwedge i\ mem_C\ mem_C'$. ⟦$i <$ *length cms*; *conc.low-mds-eq* (*snd* (cms_C ! i)) mem_C mem_C'⟧
$\Longrightarrow$ ((cms_A ! i, mem_A-*of* mem_C), (cms_A ! i, mem_A-*of* mem_C')) $\in$ $\mathcal{R}_A$-*rel* (*cms* ! i))
assumes *sound-mode-use*$_A$: ($\bigwedge$ mem_A. *abs.sound-mode-use* (cms_A, mem_A))
assumes *modes-respect-priv*: *modes-respect-priv* (*map snd* cms_C)
shows *conc.prog-sifum-secure-cont* cms_C
⟨*proof*⟩

lemma *refined-prog-secure'*:
assumes len_A [*simp*]: *length* cms_C = *length cms*
assumes len_C [*simp*]: *length* cms_A = *length cms*
assumes *in-*$\mathcal{R}$: ($\bigwedge i\ mem_C$. $i <$ *length cms* $\Longrightarrow$ ((cms_A ! i,mem_A-*of* mem_C),(cms_C ! i, mem_C)) $\in$ $\mathcal{R}$-*rel* (*cms* ! i))
assumes *in-*$\mathcal{R}_A$: ($\bigwedge i\ mem_A\ mem_A'$. ⟦$i <$ *length cms*; *abs.low-mds-eq* (*snd* (cms_A ! i)) mem_A mem_A'⟧
$\Longrightarrow$ ((cms_A ! i, mem_A), (cms_A ! i, mem_A')) $\in$ $\mathcal{R}_A$-*rel* (*cms* ! i))
assumes *sound-mode-use*$_A$: ($\bigwedge$ mem_A. *abs.sound-mode-use* (cms_A, mem_A))
assumes *modes-respect-priv*: *modes-respect-priv* (*map snd* cms_C)
shows *conc.prog-sifum-secure-cont* cms_C
⟨*proof*⟩

end

context *sifum-security* **begin**

definition
reachable-mems :: (*'Com* × (*Mode* ⇒ *'Var set*)) *list* ⇒ (*'Var*,*'Val*) *Mem* ⇒ (*'Var*,*'Val*) *Mem set*
where
reachable-mems cms mem ≡ {*mem'*. ∃ *sched cms'*. *meval-sched sched* (*cms*,*mem*) (*cms'*,*mem'*)}

lemma *reachable-mems-refl*:
mem ∈ *reachable-mems cms mem*
⟨*proof*⟩

end

context *sifum-refinement-sys* **begin**

lemma *reachable-mems-refinement*:

assumes *sys-nonempty*: *length cms* > *0*
assumes len_A [*simp*]: *length* cms_C = *length cms*
assumes len_C [*simp*]: *length* cms_A = *length cms*
assumes *in-*$\mathcal{R}$: ($\bigwedge i\ mem_C$. $i <$ *length cms* $\Longrightarrow$ ((cms_A ! i,mem_A-*of* mem_C),(cms_C ! i, mem_C)) $\in$ $\mathcal{R}$-*rel* (*cms* ! i))
assumes *sound-mode-use*$_A$: ($\bigwedge$ mem_A. *abs.sound-mode-use* (cms_A, mem_A))
assumes *modes-respect-priv*: *modes-respect-priv* (*map snd* cms_C)
assumes *reachable*$_C$: $mem_C{'} \in$ *conc.reachable-mems* cms_C mem_C
shows mem_A-*of* $mem_C{'} \in$ *abs.reachable-mems* cms_A (mem_A-*of* mem_C)
⟨*proof*⟩

end

end

References

[MSPR16] Toby Murray, Robert Sison, Edward Pierzchalski, and Christine Rizkallah. Compositional verification and refinement of concurrent value-dependent noninterference. In *IEEE Computer Security Foundations Symposium*, Lisbon, Portugal, June 2016.

[Mur15] Toby Murray. On high-assurance information-flow-secure programming languages. In *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, pages 43–48, Prague, Czech Republic, July 2015.