Compositional Security-Preserving Refinement for Concurrent Imperative Programs

Toby Murray, Robert Sison, Edward Pierzchalski and Christine Rizkallah

March 17, 2025

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*:

$inj\ (f{::}'a \Rightarrow 'b) \Longrightarrow finite\ (UNIV{::}'b\ set) \Longrightarrow card\ (UNIV{::}'a\ set) \leq card\ (UNIV{::}'b\ set)$
⟨*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 :: $('Var_A,'Val)\ Mem \Rightarrow 'Var_A \Rightarrow Sec$
  and dma_C :: $('Var_C,'Val)\ Mem \Rightarrow 'Var_C \Rightarrow Sec$
  and $\mathcal{C}$-$vars_A$ :: $'Var_A \Rightarrow 'Var_A\ set$
  and $\mathcal{C}$-$vars_C$ :: $'Var_C \Rightarrow 'Var_C\ set$
  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 :: $'Var_A \Rightarrow 'Var_C$
  assumes var_C-*of-inj*: *inj* var_C-*of*
  assumes *dma-consistent*:
    $dma_A\ (\lambda x_A.\ mem_C\ (var_C\text{-}of\ x_A))\ x_A = dma_C\ mem_C\ (var_C\text{-}of\ x_A)$
  assumes $\mathcal{C}$-*vars-consistent*:
    $(var_C\text{-}of\ `\ \mathcal{C}\text{-}vars_A\ x_A) = \mathcal{C}\text{-}vars_C\ (var_C\text{-}of\ x_A)$

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

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* =
  $(('Com_A,\ 'Var_A,\ 'Val)\ LocalConf \times ('Com_C,\ 'Var_C,\ 'Val)\ LocalConf)\ set$

context *sifum-refinement* **begin**

abbreviation
$conf\text{-}abv_A$ :: $'Com_A \Rightarrow 'Var_A\ Mds \Rightarrow ('Var_A, 'Val)\ Mem \Rightarrow$ (-,-,-) *LocalConf*
(‹⟨-, -, -⟩$_A$› [0, 0, 0] 1000)
where
$\langle\ c,\ mds,\ mem\ \rangle_A \equiv ((c,\ mds),\ mem)$

abbreviation
$conf\text{-}abv_C$ :: $'Com_C \Rightarrow 'Var_C\ Mds \Rightarrow ('Var_C, 'Val)\ Mem \Rightarrow$ (-,-,-) *LocalConf*
(‹⟨-, -, -⟩$_C$› [0, 0, 0] 1000)
where
$\langle\ c,\ mds,\ mem\ \rangle_C \equiv ((c,\ mds),\ mem)$

abbreviation
$eval\text{-}abv_A$ :: $('Com_A, 'Var_A, 'Val)$ *LocalConf* $\Rightarrow$ (-, -, -) *LocalConf* $\Rightarrow$ *bool*
(**infixl** ‹$\leadsto_A$› 70)
where
$x \leadsto_A y \equiv (x,\ y) \in eval_A$

abbreviation
$eval\text{-}abv_C$ :: $('Com_C, 'Var_C, 'Val)$ *LocalConf* $\Rightarrow$ (-, -, -) *LocalConf* $\Rightarrow$ *bool*
(**infixl** ‹$\leadsto_C$› 70)
where
$x \leadsto_C y \equiv (x,\ y) \in eval_C$

definition
preserves-modes-mem :: $('Com_A, 'Var_A, 'Val, 'Com_C, 'Var_C)$ *state-relation* $\Rightarrow$ *bool*
where
preserves-modes-mem $\mathcal{R} \equiv$
$(\forall\ c_A\ mds_A\ mem_A\ c_C\ mds_C\ mem_C.\ (\langle\ c_A,\ mds_A,\ mem_A\ \rangle_A,\ \langle\ c_C,\ mds_C,\ mem_C\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall x_A.\ (mem_A\ x_A) = (mem_C\ (var_C\text{-}of\ x_A))) \land$
$(\forall m.\ var_C\text{-}of\ `\ mds_A\ m = (range\ var_C\text{-}of) \cap mds_C\ m))$

definition
$mem_A\text{-}of$ :: $('Var_C, 'Val)\ Mem \Rightarrow ('Var_A, 'Val)\ Mem$
where
$mem_A\text{-}of\ mem_C \equiv\ (\lambda x_A.\ (mem_C\ (var_C\text{-}of\ x_A)))$

definition
$mds_A\text{-}of$ :: $'Var_C\ Mds \Rightarrow 'Var_A\ Mds$
where
$mds_A\text{-}of\ mds_C \equiv (\lambda\ m.\ (inv\ var_C\text{-}of)\ `\ (range\ var_C\text{-}of \cap mds_C\ m))$

lemma *low-mds-eq-from-conc-to-abs*:
conc.low-mds-eq mds mem mem' $\Longrightarrow$ *abs.low-mds-eq* $(mds_A\text{-}of\ mds)\ (mem_A\text{-}of\ mem)\ (mem_A\text{-}of\ mem')$

$\langle proof \rangle$

definition
$var_A\text{-}of$ :: $'Var_C \Rightarrow 'Var_A$
where
$var_A\text{-}of \equiv inv\ var_C\text{-}of$

lemma *preserves-modes-mem-mem*$_A$*-simp*:
$(\forall x_A.\ (mem_A\ x_A) = (mem_C\ (var_C\text{-}of\ x_A))) \Longrightarrow$
$mem_A = mem_A\text{-}of\ mem_C$
$\langle proof \rangle$

lemma *preserves-modes-mem-mds*$_A$*-simp*:
$(\forall m.\ var_C\text{-}of\ `\ mds_A\ m = range\ (var_C\text{-}of) \cap mds_C\ m) \Longrightarrow$
$mds_A = mds_A\text{-}of\ mds_C$
$\langle proof \rangle$

This version might be more useful. Not sure yet.

lemma *preserves-modes-mem-def2*:
$preserves\text{-}modes\text{-}mem\ \mathcal{R} =$
$(\forall\ c_A\ mds_A\ mem_A\ c_C\ mds_C\ mem_C.\ (\langle\ c_A,\ mds_A,\ mem_A\ \rangle_A,\ \langle\ c_C,\ mds_C,\ mem_C\ \rangle_C) \in \mathcal{R} \longrightarrow$
$mem_A = mem_A\text{-}of\ mem_C\ \wedge$
$mds_A = mds_A\text{-}of\ mds_C)$
$\langle proof \rangle$

definition
$closed\text{-}others$ :: $('Com_A,\ 'Var_A,\ 'Val,\ 'Com_C,\ 'Var_C)\ state\text{-}relation \Rightarrow bool$
where
$closed\text{-}others\ \mathcal{R} \equiv$
$(\forall\ c_A\ c_C\ mds_C\ mem_C\ mem_C'.\ (\langle\ c_A,\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_C\ \rangle_A,\ \langle\ c_C,\ mds_C,\ mem_C\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall x.\ mem_C\ x \neq mem_C'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds_C\ x) \longrightarrow$
$(\forall x.\ dma_C\ mem_C\ x \neq dma_C\ mem_C'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds_C\ x) \longrightarrow$
$(\langle\ c_A,\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_C'\ \rangle_A,\ \langle\ c_C,\ mds_C,\ mem_C'\ \rangle_C) \in \mathcal{R})$

definition
$stops_C$ :: $('Com_C,\ 'Var_C,\ 'Val)\ LocalConf \Rightarrow bool$
where
$stops_C\ c \equiv \forall c'.\ \neg\ (c \leadsto_C c')$

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

lemma *strong-low-bisim-neval'*:
$abs.neval\ c_1\ n\ c_n \Longrightarrow (c_1,c_1') \in \mathcal{R}_A \Longrightarrow snd\ (fst\ c_1) = snd\ (fst\ c_1') \Longrightarrow abs.strong\text{-}low\text{-}bisim\text{-}mm\ \mathcal{R}_A \Longrightarrow$
$\exists c_n'.\ abs.neval\ c_1'\ n\ c_n' \wedge (c_n,c_n') \in \mathcal{R}_A \wedge snd\ (fst\ c_n) = snd\ (fst\ (c_n'))$

$\langle proof \rangle$

lemma *strong-low-bisim-neval*:
$abs.neval\ \langle c_1, mds_1, mem_1 \rangle_A\ n\ \langle c_n, mds_n, mem_n \rangle_A \Longrightarrow (\langle c_1, mds_1, mem_1 \rangle_A, \langle c_1', mds_1, mem_1' \rangle_A) \in \mathcal{R}_A \Longrightarrow abs.strong\text{-}low\text{-}bisim\text{-}mm\ \mathcal{R}_A \Longrightarrow$
$\exists\, c_n'\ mem_n'.\ abs.neval\ \langle c_1', mds_1, mem_1' \rangle_A\ n\ \langle c_n', mds_n, mem_n' \rangle_A \wedge (\langle c_n, mds_n, mem_n \rangle_A, \langle c_n', mds_n, mem_n' \rangle_A) \in \mathcal{R}_A$
$\langle proof \rangle$

lemma *in-$\mathcal{R}$-dma'*:
assumes *preserves*: $preserves\text{-}modes\text{-}mem\ \mathcal{R}$
assumes *in-$\mathcal{R}$*: $(\langle c_A, mds_A, mem_A \rangle_A, \langle c_C, mds_C, mem_C \rangle_C) \in \mathcal{R}$
shows $dma_A\ mem_A\ x_A = dma_C\ mem_C\ (var_C\text{-}of\ x_A)$
$\langle proof \rangle$

lemma *in-$\mathcal{R}$-dma*:
assumes *preserves*: $preserves\text{-}modes\text{-}mem\ \mathcal{R}$
assumes *in-$\mathcal{R}$*: $(\langle c_A, mds_A, mem_A \rangle_A, \langle c_C, mds_C, mem_C \rangle_C) \in \mathcal{R}$
shows $dma_A\ mem_A = (dma_C\ mem_C \circ var_C\text{-}of)$
$\langle proof \rangle$

definition
$new\text{-}vars\text{-}private :: ('Com_A, 'Var_A, 'Val, 'Com_C, 'Var_C)\ state\text{-}relation \Rightarrow bool$
where
$new\text{-}vars\text{-}private\ \mathcal{R} \equiv$
$(\forall\ c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \leadsto_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C \longrightarrow$
$(\forall\, v_C.\ (mem_{1C}'\ v_C \neq mem_{1C}\ v_C \vee dma_C\ mem_{1C}'\ v_C < dma_C\ mem_{1C}\ v_C) \wedge v_C \notin range\ var_C\text{-}of \longrightarrow v_C \in mds_C'\ AsmNoReadOrWrite) \wedge$
$(mds_C\ AsmNoReadOrWrite - (range\ var_C\text{-}of)) \subseteq (mds_C'\ AsmNoReadOrWrite - (range\ var_C\text{-}of))))$

lemma *not-less-eq-is-greater-Sec*:
$(\neg\ a \leq (b::Sec)) = (a > b)$
$\langle proof \rangle$

lemma *doesnt-have-mode*:
$(x \notin mds_A\text{-}of\ mds_C\ m) = (var_C\text{-}of\ x \notin mds_C\ m)$
$\langle proof \rangle$

lemma *new-vars-private-does-the-thing*:
assumes *nice*: $new\text{-}vars\text{-}private\ \mathcal{R}$
assumes *in-$\mathcal{R}_1$*: $(\langle\ c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R}$
assumes *in-$\mathcal{R}_2$*: $(\langle\ c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\ \rangle_A,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in \mathcal{R}$

assumes $step_{1C}$: $\langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \leadsto_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C$
assumes $step_{2C}$: $\langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \leadsto_C \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C$
assumes $\textit{low-mds-eq}_C$: *conc.low-mds-eq* $mds_C\ mem_{1C}\ mem_{2C}$
assumes $\textit{low-mds-eq}_A{}'$: *abs.low-mds-eq* $(mds_A\textit{-of}\ mds_C')\ (mem_A\textit{-of}\ mem_{1C}')\ (mem_A\textit{-of}\ mem_{2C}')$
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 :: $('Com_A,\ 'Var_A,\ 'Val)\ \textit{LocalConf rel} \Rightarrow ('Com_A,\ 'Var_A,\ 'Val,\ 'Com_C,\ 'Var_C)\ \textit{state-relation} \Rightarrow$
$('Com_C,\ 'Var_C,\ 'Val)\ \textit{LocalConf rel} \Rightarrow \textit{bool}$
where
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 $P\ \wedge$
$(\forall\ c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$((\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \leadsto_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C \longrightarrow$
$(\exists\ n\ c_{1A}'\ mds_A'\ mem_{1A}'.\ abs.neval\ \langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A\ n\ \langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A\ \wedge$
$(\langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A,\ \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R}\ \wedge$
$(\forall\ c_{2A}\ mem_{2A}\ c_{2C}\ mem_{2C}\ c_{2A}'\ mem_{2A}'.$
$((\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{2A},\ mds_A,\ mem_{2A}\ \rangle_A) \in \mathcal{R}_A\ \wedge$
$((\langle\ c_{2A},\ mds_A,\ mem_{2A}\ \rangle_A,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in \mathcal{R}\ \wedge$
$((\langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in P\ \wedge$
$abs.neval\ \langle\ c_{2A},\ mds_A,\ mem_{2A}\ \rangle_A\ n\ \langle\ c_{2A}',\ mds_A',\ mem_{2A}'\ \rangle_A \longrightarrow$
$(\exists\ c_{2C}'\ mem_{2C}'.\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \leadsto_C \langle\ c_{2C}',\ mds_C',\ mem_{2C}'$

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

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

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

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

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

$((('Com_C \times (Mode \Rightarrow 'Var_C\ set)) \times ('Var_C \Rightarrow 'Val)) \times$
$('Com_C \times (Mode \Rightarrow 'Var_C\ set)) \times ('Var_C \Rightarrow 'Val))\ set$

where

$R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P \equiv \{(x,y).\ \exists\, x_A\ y_A.\ (x_A,x) \in \mathcal{R} \wedge (y_A,y) \in \mathcal{R} \wedge (x_A,y_A) \in \mathcal{R}_A \wedge$
$snd\ (fst\ x) = snd\ (fst\ y)$ — TODO: annoying to have to say $\wedge$
$conc.low\text{-}mds\text{-}eq\ (snd\ (fst\ x))\ (snd\ x)\ (snd\ y) \wedge$
$(x,y) \in P\}$

lemma $abs\text{-}low\text{-}mds\text{-}eq\text{-}dma_C\text{-}eq$:
assumes $abs.low\text{-}mds\text{-}eq\ (mds_A\text{-}of\ mds)\ (mem_A\text{-}of\ mem_{1C})\ (mem_A\text{-}of\ mem_{2C})$
shows $dma_C\ mem_{1C} = dma_C\ mem_{2C}$
$\langle proof \rangle$

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

lemma $R_C\text{-}ofI$:
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R} \Longrightarrow$
$(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \Longrightarrow$
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A) \in \mathcal{R}_A \Longrightarrow$
$conc.low\text{-}mds\text{-}eq\ mds_C\ mem_{1C}\ mem_{2C} \Longrightarrow$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P \Longrightarrow$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
$\langle proof \rangle$

lemma $R_C\text{-}of\text{-}sym$:
assumes $sym\ \mathcal{R}_A$
assumes $P\text{-}sym$: $sym\ P$
assumes rr: $secure\text{-}refinement\ \mathcal{R}_A\ \mathcal{R}\ P$
assumes mm:
$\bigwedge c_1\ mds\ mem_1\ c_2\ mds\ mem_2.\ (\langle c_1,\ mds,\ mem_1\rangle_A,\ \langle c_2,\ mds,\ mem_2\rangle_A) \in \mathcal{R}_A \Longrightarrow$
$abs.low\text{-}mds\text{-}eq\ mds\ mem_1\ mem_2$
shows $sym\ (R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P)$
$\langle proof \rangle$

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

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

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

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

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

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

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

assumes $x_A \in \mathcal{C}_A$
shows $var_C\text{-}of\ x_A \in \mathcal{C}_C$
$\langle proof \rangle$

lemma *doesnt-have-mode*$_C$:
$x \notin mds_A\text{-}of\ mds_C\ m \Longrightarrow var_C\text{-}of\ x \notin mds_C\ m$
$\langle proof \rangle$

lemma *has-mode*$_A$: $var_C\text{-}of\ x \in mds_C\ m \Longrightarrow x \in mds_A\text{-}of\ mds_C\ m$
$\langle proof \rangle$

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

lemma *apply-adaptation*$_A$:
$(mem_A\text{-}of\ mem_{1C})\ [\|_1\ A_A\text{-}of\ A_C] = mem_A\text{-}of\ (mem_{1C}\ [\|_1\ A_C])$
$(mem_A\text{-}of\ mem_{1C})\ [\|_2\ A_A\text{-}of\ A_C] = mem_A\text{-}of\ (mem_{1C}\ [\|_2\ A_C])$
$\langle proof \rangle$

lemma R_C*-of-closed-glob-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-mds-eq $mds\ mem_1\ mem_2$
assumes *cgc*: *abs.closed-glob-consistent* $\mathcal{R}_A$
assumes *rr*: *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
shows *conc.closed-glob-consistent* $(R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P)$
$\langle proof \rangle$

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

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* $\equiv$

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

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

lemma *secure-refinement-simpler*:
assumes *rrs*: $secure\text{-}refinement\text{-}simpler\ \mathcal{R}_A\ \mathcal{R}\ P\ abs\text{-}steps$
shows $secure\text{-}refinement\ \mathcal{R}_A\ \mathcal{R}\ P$
⟨*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
$bisim\text{-}simple\ \mathcal{R}_A \equiv \forall\, c_{1A}\ mds\ mem_{1A}\ c_{2A}\ mem_{2A}.\ (\langle c_{1A}, mds, mem_{1A}\rangle_A, \langle c_{2A}, mds, mem_{2A}\rangle_A) \in \mathcal{R}_A \longrightarrow$
$c_{1A} = c_{2A}$

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

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 $\mathcal{R}_A$ $\wedge$
$(\forall\ c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \leadsto_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C \longrightarrow$
$(\exists\ c_{1A}'\ mds_A'\ mem_{1A}'.\ abs.neval\ \langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A\ (abs\text{-}steps\ \langle c_{1A}, mds_A, mem_{1A}\rangle_A\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C)\ \langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A \wedge$
$(\langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A,\ \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R})))$

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-closed-glob-consistent*:
conc.closed-glob-consistent $\mathcal{I}simple$
⟨*proof*⟩

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*
⟨*proof*⟩

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*:
$mds_A\text{-}of\ (mds_C(SomeMode := insert\ (var_C\text{-}of\ x)\ (mds_C\ SomeMode))) =$
$(mds_A\text{-}of\ mds_C)(SomeMode := insert\ x\ (mds_A\text{-}of\ mds_C\ SomeMode))$
⟨*proof*⟩

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

definition
$preserves\text{-}locally\text{-}sound\text{-}mode\text{-}use :: ('Com_A, 'Var_A, 'Val, 'Com_C, 'Var_C)\ state\text{-}relation \Rightarrow bool$
where
$preserves\text{-}locally\text{-}sound\text{-}mode\text{-}use\ \mathcal{R} \equiv$
$\forall\ lc_A\ lc_C.$
$(abs.locally\text{-}sound\text{-}mode\text{-}use\ lc_A \wedge (lc_A, lc_C) \in \mathcal{R} \longrightarrow$
$conc.locally\text{-}sound\text{-}mode\text{-}use\ lc_C)$

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

definition *preserves-local-guarantee-compliance* ::
$('Com_A, 'Var_A, 'Val, 'Com_C, 'Var_C)\ state\text{-}relation \Rightarrow bool$
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*]:

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

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

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

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

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

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

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

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

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

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 $\mathcal{R}_A\ \mathcal{R}\ P \Longrightarrow$
$((cm_A,mem_A),(cm_C,mem_C)) \in \mathcal{R} \Longrightarrow (cm_C,mem_C) \leadsto_C (cm_C{}',mem_C{}') \Longrightarrow$
$(\exists\ cm_A{}'\ mem_A{}'\ n.\ abs.neval\ (cm_A,mem_A)\ n\ (cm_A{}',mem_A{}') \wedge$
$((cm_A{}',mem_A{}'),\ (cm_C{}',mem_C{}')) \in \mathcal{R})$
⟨*proof*⟩

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

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

$priv\text{-}mem$ :: $'Var_C$ set
$\mathcal{R}_A\text{-}rel$ :: $('Com_A,\ 'Var_A,\ 'Val)$ $LocalConf$ rel
$\mathcal{R}\text{-}rel$ :: $('Com_A,\ 'Var_A,\ 'Val,\ 'Com_C,\ 'Var_C)$ $state\text{-}relation$
$P\text{-}rel$ :: $('Com_C,\ 'Var_C,\ 'Val)$ $LocalConf$ 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 $\mathcal{C}\text{-}vars_A$ $\mathcal{C}\text{-}vars_C$ $\mathcal{C}_A$ $\mathcal{C}_C$ $eval_A$ $eval_C$ *some-val* $var_C\text{-}of$
for dma_A :: $('Var_A,'Val)\ Mem \Rightarrow 'Var_A \Rightarrow Sec$
and dma_C :: $('Var_C,'Val)\ Mem \Rightarrow 'Var_C \Rightarrow Sec$
and $\mathcal{C}\text{-}vars_A$ :: $'Var_A \Rightarrow 'Var_A\ set$
and $\mathcal{C}\text{-}vars_C$ :: $'Var_C \Rightarrow 'Var_C\ set$
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 $var_C\text{-}of$ :: $'Var_A \Rightarrow 'Var_C$ +
fixes *cms* :: $('a{::}wellorder,\ 'Val,\ 'Var_C,\ 'Com_C,\ 'Var_A,\ 'Com_A)$ *componentwise-refinement list*
fixes $priv\text{-}mem_C$ :: $'Var_C\ set\ list$
defines $priv\text{-}mem_C\text{-}def$: $priv\text{-}mem_C \equiv map\ priv\text{-}mem\ cms$
assumes *priv-mem-disjoint*: $i < length\ cms \Longrightarrow j < length\ cms \Longrightarrow i \neq j \Longrightarrow priv\text{-}mem_C\ !\ i \cap priv\text{-}mem_C\ !\ j = \{\}$
assumes *new-vars-priv*: $-\ range\ var_C\text{-}of = \bigcup\ (set\ priv\text{-}mem_C)$
assumes *new-privs-preserved*: $\langle c,\ mds,\ mem\rangle_C \rightsquigarrow_C \langle c',\ mds',\ mem'\rangle_C \Longrightarrow x \notin range\ var_C\text{-}of \Longrightarrow$
$$(x \in mds\ m) = (x \in mds'\ m)$$
assumes *secure-refinements*:
$i < length\ cms \Longrightarrow secure\text{-}refinement\ (\mathcal{R}_A\text{-}rel\ (cms\ !\ i))\ (\mathcal{R}\text{-}rel\ (cms\ !\ i))\ (P\text{-}rel\ (cms\ !\ i))$
assumes *local-guarantee-preservation*:
$i < length\ cms \Longrightarrow preserves\text{-}local\text{-}guarantee\text{-}compliance\ (\mathcal{R}\text{-}rel\ (cms\ !\ i))$
assumes *bisims*:
$i < length\ cms \Longrightarrow abs.strong\text{-}low\text{-}bisim\text{-}mm\ (\mathcal{R}_A\text{-}rel\ (cms\ !\ i))$
assumes *Ps-sym*:
$\bigwedge a\ b.\ i < length\ cms \Longrightarrow sym\ (P\text{-}rel\ (cms\ !\ i))$
assumes *Ps-refl-on-low-mds-eq*:
$i < length\ cms \Longrightarrow conc.low\text{-}mds\text{-}eq\ mds_C\ mem_C\ mem_C' \Longrightarrow (\langle c_C,\ mds_C,\ mem_C\rangle_C,\ \langle c_C,\ mds_C,\ mem_C'\rangle_C) \in (P\text{-}rel\ (cms\ !\ i))$

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
$priv\text{-}is\text{-}asm\text{-}priv\ mdss_C \equiv\ \forall\, i < length\ cms.\ priv\text{-}mem_C \ ! \ i \subseteq (mdss_C \ ! \ i)\ AsmNoReadOrWrite$

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

$GuarNoReadOrWrite$)

definition
$new\text{-}asms\text{-}only\text{-}for\text{-}priv :: {}'Var_C\ Mds\ list \Rightarrow bool$
where
$new\text{-}asms\text{-}only\text{-}for\text{-}priv\ mdss_C \equiv$
$\forall\ i < length\ cms.$
$((mdss_C\ !\ i)\ AsmNoReadOrWrite \cup (mdss_C\ !\ i)\ AsmNoWrite) \cap (-\ range\ var_C\text{-}of) \subseteq priv\text{-}mem_C\ !\ i$

definition
$new\text{-}asms\text{-}NoReadOrWrite\text{-}only :: {}'Var_C\ Mds\ list \Rightarrow bool$
where
$new\text{-}asms\text{-}NoReadOrWrite\text{-}only\ mdss_C \equiv$
$\forall\ i < length\ cms.$
$(mdss_C\ !\ i)\ AsmNoWrite \cap (-\ range\ var_C\text{-}of) = \{\}$

definition
$modes\text{-}respect\text{-}priv :: {}'Var_C\ Mds\ list \Rightarrow bool$
where
$modes\text{-}respect\text{-}priv\ mdss_C \equiv priv\text{-}is\text{-}asm\text{-}priv\ mdss_C \wedge priv\text{-}is\text{-}guar\text{-}priv\ mdss_C \wedge$
$new\text{-}asms\text{-}only\text{-}for\text{-}priv\ mdss_C \wedge$
$new\text{-}asms\text{-}NoReadOrWrite\text{-}only\ mdss_C$

definition
$ignores\text{-}old\text{-}vars :: ({}'Var_C\ Mds\ list \Rightarrow bool) \Rightarrow bool$
where
$ignores\text{-}old\text{-}vars\ P \equiv \forall\, mdss\ mdss'.\ length\ mdss = length\ mdss' \wedge length\ mdss' = length\ cms \longrightarrow$
$(map\ (\lambda x\ m.\ x\ m \cap (-\ range\ var_C\text{-}of))\ mdss) = (map\ (\lambda x\ m.\ x\ m \cap (-\ range\ var_C\text{-}of))\ mdss') \longrightarrow$
$P\ mdss = P\ mdss'$

lemma *ignores-old-vars-conj*:
assumes *Rdef*: $(\bigwedge x.\ R\ x = (P\ x \wedge Q\ x))$
assumes *iP*: $ignores\text{-}old\text{-}vars\ P$
assumes *iQ*: $ignores\text{-}old\text{-}vars\ Q$
shows $ignores\text{-}old\text{-}vars\ R$
⟨*proof*⟩

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

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

$\langle proof \rangle$

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

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

lemma *priv-is-asm-priv-ignores-old-vars*:
ignores-old-vars priv-is-asm-priv
$\langle proof \rangle$

lemma *priv-is-guar-priv-ignores-old-vars*:
ignores-old-vars priv-is-guar-priv
$\langle proof \rangle$

lemma *new-asms-only-for-priv-ignores-old-vars*:
ignores-old-vars new-asms-only-for-priv
$\langle proof \rangle$

lemma *new-asms-NoReadOrWrite-only-ignores-old-vars*:
ignores-old-vars new-asms-NoReadOrWrite-only
$\langle proof \rangle$

lemma *modes-respect-priv-ignores-old-vars*:
ignores-old-vars modes-respect-priv
$\langle proof \rangle$

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

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

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

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 $gc_C{}'$ ⟹ *length* (*fst* gc_C) = *length cms* ⟹
modes-respect-priv (*map snd* (*fst* gc_C)) ⟹
modes-respect-priv (*map snd* (*fst* $gc_C{}'$))
⟨*proof*⟩

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

lemma *traces-refinement*:
$\bigwedge gc_C\ gc_C{}'\ sched_C\ gc_A$. *conc.meval-sched* $sched_C\ gc_C\ gc_C{}'$ ⟹
length (*fst* gc_A) = *length cms* ⟹ *length* (*fst* gc_C) = *length cms* ⟹
($\bigwedge i$. *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)) ⟹
∃ $sched_A\ gc_A{}'$. *abs.meval-sched* $sched_A\ gc_A\ gc_A{}'$ ∧
(∀ *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 $gc_A{}'$
⟨*proof*⟩

end

context *sifum-security* **begin**

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

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

$\langle proof \rangle$

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*
$\langle proof \rangle$

lemma *in-restrict-modesD*:
  $i < length\ mdss \Longrightarrow x \in ((restrict\text{-}modes\ mdss\ X)\ !\ i)\ m \Longrightarrow x \in X \land x \in (mdss\ !\ i)\ m$
  $\langle proof \rangle$

lemma *in-restrict-modesI*:
  $i < length\ mdss \Longrightarrow x \in X \Longrightarrow x \in (mdss\ !\ i)\ m \Longrightarrow x \in ((restrict\text{-}modes\ mdss\ X)\ !\ i)\ m$
  $\langle proof \rangle$

lemma *meval-sched-length*:
  $meval\text{-}sched\ sched\ gc\ gc' \Longrightarrow length\ (fst\ gc') = length\ (fst\ gc)$
  $\langle proof \rangle$

end

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

lemma *compatible-modes-old-vars*:
  assumes $compatible\text{-}modes_A$: *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-*$\mathcal{R}$: $(\forall\, i{<}length\ cms.\ ((cms_A\ !\ i,\ mem_A),\ cms_C\ !\ i,\ mem_C) \in \mathcal{R}\text{-}rel\ (cms\ !\ i))$
  shows *conc.compatible-modes* (*conc.restrict-modes* ($map\ snd\ cms_C$) ($range\ var_C\text{-}of$))
$\langle proof \rangle$

lemma *compatible-modes-new-vars*:
  $length\ mdss = length\ cms \Longrightarrow modes\text{-}respect\text{-}priv\ mdss \Longrightarrow conc.compatible\text{-}modes\ (conc.restrict\text{-}modes\ mdss\ (-\ range\ var_C\text{-}of))$
$\langle proof \rangle$

lemma *sound-mode-use-preservation*:
  $\bigwedge gc_C\ gc_A.$
    $length\ (fst\ gc_A) = length\ cms \Longrightarrow length\ (fst\ gc_C) = length\ cms \Longrightarrow$
    $(\bigwedge i.\ i < length\ cms \Longrightarrow ((fst\ gc_A\ !\ i,\ snd\ gc_A),\ (fst\ gc_C\ !\ i,\ snd\ gc_C)) \in \mathcal{R}\text{-}rel\ (cms\ !\ i)) \Longrightarrow$
    $abs.sound\text{-}mode\text{-}use\ gc_A \Longrightarrow modes\text{-}respect\text{-}priv\ (map\ snd\ (fst\ gc_C)) \Longrightarrow$
    $conc.sound\text{-}mode\text{-}use\ gc_C$

$\langle proof \rangle$

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

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

end

context *sifum-security* **begin**

definition
reachable-mems :: $('Com \times (Mode \Rightarrow 'Var\ set))\ list \Rightarrow ('Var,'Val)\ Mem \Rightarrow ('Var,'Val)\ Mem\ set$
where
reachable-mems cms mem $\equiv \{mem'.\ \exists\ sched\ cms'.\ meval\text{-}sched\ sched\ (cms,mem)\ (cms',mem')\}$

lemma *reachable-mems-refl*:
$mem \in reachable\text{-}mems\ cms\ mem$
$\langle proof \rangle$

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-*$\mathcal{R}$: $(\bigwedge i\ mem_C.\ i < length\ cms \Longrightarrow ((cms_A\ !\ i, mem_A\text{-}of\ mem_C),(cms_C\ !\ i,\ mem_C)) \in \mathcal{R}\text{-}rel\ (cms\ !\ i))$
assumes $sound\text{-}mode\text{-}use_A$: $(\bigwedge\ mem_A.\ abs.sound\text{-}mode\text{-}use\ (cms_A,\ mem_A))$
assumes *modes-respect-priv*: $modes\text{-}respect\text{-}priv\ (map\ snd\ cms_C)$
assumes $reachable_C$: $mem_C' \in conc.reachable\text{-}mems\ cms_C\ mem_C$
shows $mem_A\text{-}of\ mem_C' \in abs.reachable\text{-}mems\ cms_A\ (mem_A\text{-}of\ mem_C)$
⟨*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.