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*)
by (*blast intro*: *card-inj-on-le*)

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'*)

apply(*clarsimp simp*: *abs.low-mds-eq-def conc.low-mds-eq-def* mem_A*-of-def* mds_A*-of-def*)
using var_C*-of-inj*
by (*metis IntI control-vars-are-A-vars dma-consistent image-eqI inv-f-f rangeI*)

definition
var_A*-of* :: $'Var_C \Rightarrow 'Var_A$
where
var_A*-of* $\equiv$ *inv* var_C*-of*

lemma *preserves-modes-mem-*mem_A*-simp*:
$(\forall x_A.\ (mem_A\ x_A) = (mem_C\ (var_C\text{-}of\ x_A))) \Longrightarrow$
$mem_A = mem_A\text{-}of\ mem_C$
unfolding mem_A*-of-def* **by** *blast*

lemma *preserves-modes-mem-*mds_A*-simp*:
$(\forall m.\ var_C\text{-}of\ `\ mds_A\ m = range\ (var_C\text{-}of) \cap mds_C\ m) \Longrightarrow$
$mds_A = mds_A\text{-}of\ mds_C$
unfolding mds_A*-of-def*
apply(*rule ext*)
apply(*drule-tac x=m* **in** *spec*)
apply(*rule equalityI*)
apply *clarsimp*
apply(*rename-tac* x_A)
apply(*drule equalityD1*)
apply(*drule-tac* $c{=}var_C$*-of* x_A **in** *subsetD*)
apply *blast*
unfolding *image-def*
apply *clarsimp*
apply(*rule-tac* $x{=}var_C$*-of* x_A **in** *bexI*)
apply(*rule sym*)
apply(*rule inv-f-f*[*OF* var_C*-of-inj*])
apply(*drule inj-onD*[*OF* var_C*-of-inj*])
apply *blast+*
apply *clarsimp*
apply(*rename-tac* x_A)
apply(*simp add*: *inv-f-f*[*OF* var_C*-of-inj*])
apply(*drule equalityD2*)
apply(*drule-tac* $c{=}var_C$*-of* x_A **in** *subsetD*)
apply *blast*
apply *clarsimp*
apply(*drule inj-onD*[*OF* var_C*-of-inj*])
apply *blast+*
done

This version might be more useful. Not sure yet.

lemma *preserves-modes-mem-def2*:
preserves-modes-mem $\mathcal{R}$ =
$(\forall\ c_A\ mds_A\ mem_A\ c_C\ mds_C\ mem_C.\ (\langle\ c_A, mds_A, mem_A\ \rangle_A, \langle\ c_C, mds_C, mem_C$

$\rangle_C) \in \mathcal{R} \longrightarrow$
$mem_A = mem_A\text{-}of\ mem_C \wedge$
$mds_A = mds_A\text{-}of\ mds_C$)
unfolding *preserves-modes-mem-def*
apply(*rule iffI*)
apply(*blast dest*: *preserves-modes-mem-mem*$_A$*-simp preserves-modes-mem-mds*$_A$*-simp*)
apply *safe*
apply(*elim allE impE*, *assumption*, *elim conjE*)
apply(*simp add*: *mem*$_A$*-of-def*)
apply *blast*
apply *clarsimp*
apply(*rename-tac* x_A)
apply(*elim allE impE*, *assumption*, *elim conjE*)
apply *clarsimp*
apply(*clarsimp simp*: *mds*$_A$*-of-def image-def*)
apply(*simp add*: *inv-f-f*[*OF var*$_C$*-of-inj*])
apply *clarsimp*
apply(*rename-tac* x_A)
apply(*rule imageI*)
apply(*elim allE impE*, *assumption*, *elim conjE*)
apply(*clarsimp simp*: *mds*$_A$*-of-def*)
apply(*subst image-def*)
apply *clarify*
apply(*rule-tac x=var*$_C$*-of* x_A **in** *bexI*)
apply(*simp add*: *inv-f-f*[*OF var*$_C$*-of-inj*])
apply *blast*
done

definition
closed-others :: (*'Com*$_A$, *'Var*$_A$, *'Val*, *'Com*$_C$, *'Var*$_C$) *state-relation* $\Rightarrow$ *bool*
where
closed-others $\mathcal{R} \equiv$
($\forall$ c_A c_C *mds*$_C$ *mem*$_C$ *mem*$_C$*'*. ($\langle$ c_A, *mds*$_A$*-of mds*$_C$, *mem*$_A$*-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-asm-not-written mds*$_C$ x) $\longrightarrow$
($\forall x$. *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*$_A$*-of mds*$_C$, *mem*$_A$*-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$_C$ $c \equiv \forall c'. \neg (c \rightsquigarrow_C c')$

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-low-bisim-mm $\mathcal{R}_A \Longrightarrow$

```
  ∃ c_n'. abs.neval c_1' n c_n' ∧ (c_n,c_n') ∈ R_A ∧ snd (fst c_n) = snd (fst (c_n'))
proof(induct  arbitrary: c_1' rule: neval-induct)
  case (Zero c_1  c_n)
   hence abs.neval c_1' 0 c_1' ∧ (c_n, c_1') ∈ R_A ∧ snd (fst c_n) = snd (fst c_1')
     by(blast intro: abs.neval.intros(1))
   thus ?case by blast
next
  case (Suc lc_0 lc_1 n lc_n lc_0')
  obtain c_0 mds_0 mem_0
  where [simp]: lc_0 = ⟨c_0, mds_0, mem_0⟩_A by (case-tac lc_0, auto)
  obtain c_1 mds_1 mem_1
  where [simp]: lc_1 = ⟨c_1, mds_1, mem_1⟩_A by (case-tac lc_1, auto)
  from ‹snd (fst lc_0) = snd (fst lc_0')› obtain c_0' mem_0'
  where [simp]: lc_0' = ⟨c_0', mds_0, mem_0'⟩_A by (case-tac lc_0', auto)

 from ‹(lc_0, lc_0') ∈ R_A›[simplified] ‹lc_0 ⇝_A lc_1›[simplified] ‹abs.strong-low-bisim-mm
R_A›
  obtain c_1' mem_1' where a: ⟨c_0',mds_0, mem_0'⟩_A ⇝_A ⟨c_1',mds_1, mem_1'⟩_A and
         b: (⟨c_1,mds_1,mem_1⟩_A,⟨c_1',mds_1, mem_1'⟩_A) ∈ R_A
    unfolding abs.strong-low-bisim-mm-def
    by blast

  from this Suc.hyps Suc(6) obtain lc_S' where abs.neval ⟨c_1',mds_1,mem_1'⟩_A n
lc_S' and (lc_n, lc_S') ∈ R_A and snd (fst lc_n) = snd (fst lc_S')
    by force
  with Suc this a b show ?case by(fastforce intro: abs.neval.intros(2))
qed

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)
∈ R_A ⟹ abs.strong-low-bisim-mm 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)
∈ R_A
  by(drule strong-low-bisim-neval', simp+)

lemma in-R-dma':
  assumes preserves: preserves-modes-mem R
  assumes in-R: (⟨c_A,mds_A,mem_A⟩_A,⟨c_C,mds_C,mem_C⟩_C) ∈ R
  shows  dma_A mem_A x_A = dma_C mem_C (var_C-of x_A)
proof −
  from assms have
    mds_A-def: mds_A = mds_A-of mds_C and
    mem_A-def: mem_A = mem_A-of mem_C
    unfolding preserves-modes-mem-def2 by blast+

  have dma_A (mem_A-of mem_C) x_A = dma_C mem_C (var_C-of x_A)
    unfolding mem_A-of-def
    by(rule dma-consistent)
```

thus *?thesis*
by(*simp add*: mem_A*-def*)
qed

lemma *in-*$\mathcal{R}$*-dma*:
assumes *preserves*: *preserves-modes-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 $dma_A\ mem_A = (dma_C\ mem_C \circ var_C\text{-}of)$
unfolding *o-def*
using *assms* **by**(*blast intro*: *in-*$\mathcal{R}$*-dma'*)

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

lemma *not-less-eq-is-greater-Sec*:
$(\neg\ a \leq (b::Sec)) = (a > b)$
unfolding *less-Sec-def less-eq-Sec-def* **using** *Sec.exhaust* **by** *blast*

lemma *doesnt-have-mode*:
$(x \notin mds_A\text{-}of\ mds_C\ m) = (var_C\text{-}of\ x \notin mds_C\ m)$
apply(*clarsimp simp*: mds_A*-of-def image-def*)
apply(*rule iffI*)
apply *clarsimp*
apply(*drule-tac* $x{=}var_C\text{-}of\ x$ **in** *bspec*)
apply *blast*
apply(*simp add*: *inv-f-f*[*OF* var_C*-of-inj*])
apply(*clarify*)
apply(*simp add*: *inv-f-f*[*OF* var_C*-of-inj*])
done

lemma *new-vars-private-does-the-thing*:
assumes *nice*: *new-vars-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*$_{1C}$: $\langle\ c_{1C}, mds_C, mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}', mds_C', mem_{1C}'\ \rangle_C$
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$_{2C}$'*
unfolding *conc.low-mds-eq-def*
proof(*clarify*)
let *?mem$_{1A}$ = mem$_A$-of mem$_{1C}$*
let *?mem$_{2A}$ = mem$_A$-of mem$_{2C}$*
let *?mem$_{1A}$' = mem$_A$-of mem$_{1C}$'*
let *?mem$_{2A}$' = mem$_A$-of mem$_{2C}$'*
let *?mds$_A$ = mds$_A$-of mds$_C$*
let *?mds$_A$' = mds$_A$-of mds$_C$'*
fix *x$_C$*
assume *is-Low$_C$'*: *dma$_C$ mem$_{1C}$' x$_C$ = Low*
assume *is-readable$_C$'*: *x$_C$ ∈ $\mathcal{C}_C$ ∨ x$_C$ ∉ mds$_C$' AsmNoReadOrWrite*
show *mem$_{1C}$' x$_C$ = mem$_{2C}$' x$_C$*
proof(*cases dma$_C$ mem$_{1C}$' x$_C$ ≥ dma$_C$ mem$_{1C}$ x$_C$ ∧ mem$_{1C}$' x$_C$ = mem$_{1C}$ x$_C$ ∧ mem$_{2C}$' x$_C$ = mem$_{2C}$ x$_C$ ∧ (x$_C$ ∈ mds$_C$ AsmNoReadOrWrite ⟶ x$_C$ ∈ mds$_C$' AsmNoReadOrWrite)*)
assume *easy*: *dma$_C$ mem$_{1C}$' x$_C$ ≥ dma$_C$ mem$_{1C}$ x$_C$ ∧ mem$_{1C}$' x$_C$ = mem$_{1C}$ x$_C$ ∧ mem$_{2C}$' x$_C$ = mem$_{2C}$ x$_C$ ∧ (x$_C$ ∈ mds$_C$ AsmNoReadOrWrite ⟶ x$_C$ ∈ mds$_C$' AsmNoReadOrWrite)*
with *is-Low$_C$'* **have** *is-Low$_C$*: *dma$_C$ mem$_{1C}$ x$_C$ = Low* **by** (*simp add*: *less-eq-Sec-def*)
from *easy is-readable$_C$'* **have** *is-readable$_C$*: *x$_C$ ∈ $\mathcal{C}_C$ ∨ x$_C$ ∉ mds$_C$ AsmNoReadOrWrite* **by** *blast*
from *is-Low$_C$ is-readable$_C$ low-mds-eq$_C$* **have** *mem$_{1C}$ x$_C$ = mem$_{2C}$ x$_C$*
unfolding *conc.low-mds-eq-def* **by** *blast*
with *easy* **show** *?thesis* **by** *metis*
next
assume *a*: *¬ (dma$_C$ mem$_{1C}$ x$_C$ ≤ dma$_C$ mem$_{1C}$' x$_C$ ∧ mem$_{1C}$' x$_C$ = mem$_{1C}$ x$_C$ ∧ mem$_{2C}$' x$_C$ = mem$_{2C}$ x$_C$ ∧ (x$_C$ ∈ mds$_C$ AsmNoReadOrWrite ⟶ x$_C$ ∈ mds$_C$' AsmNoReadOrWrite))*
hence *a-disj*: *(dma$_C$ mem$_{1C}$ x$_C$ > dma$_C$ mem$_{1C}$' x$_C$ ∨ mem$_{1C}$' x$_C$ ≠ mem$_{1C}$ x$_C$ ∨ mem$_{2C}$' x$_C$ ≠ mem$_{2C}$ x$_C$ ∨ (x$_C$ ∈ mds$_C$ AsmNoReadOrWrite ∧ x$_C$ ∉ mds$_C$' AsmNoReadOrWrite))*
using *not-less-eq-is-greater-Sec* **by** *blast*
show *mem$_{1C}$' x$_C$ = mem$_{2C}$' x$_C$*
proof(*cases x$_C$ ∈ range var$_C$-of*)
assume *C-only-var*: *x$_C$ ∉ range var$_C$-of*
with *in-$\mathcal{R}_1$ step$_{1C}$ nice*
have *(mem$_{1C}$' x$_C$ ≠ mem$_{1C}$ x$_C$ ∨ dma$_C$ mem$_{1C}$' x$_C$ < dma$_C$ mem$_{1C}$ x$_C$) ⟶ x$_C$ ∈ mds$_C$' AsmNoReadOrWrite*
unfolding *new-vars-private-def* **by** *blast*
moreover from *C-only-var in-$\mathcal{R}_2$ step$_{2C}$ nice* **have** *(mem$_{2C}$' x$_C$ ≠ mem$_{2C}$ x$_C$) ⟶ x$_C$ ∈ mds$_C$' AsmNoReadOrWrite*
unfolding *new-vars-private-def* **by** *blast*

moreover from *C-only-var in-*$\mathcal{R}_1$ *step*$_{1C}$ *nice* **have** $x_C \in mds_C$ *AsmNoReadOrWrite* $\longrightarrow x_C \in mds_C'$ *AsmNoReadOrWrite* **unfolding** *new-vars-private-def* **by** *blast*

moreover from *C-only-var is-readable*$_C'$ **have** $x_C \notin mds_C'$ *AsmNoReadOrWrite*

using *control-vars-are-A-vars* **by** *blast*

ultimately have *False* **using** *a-disj* **by** *blast*

thus *?thesis* **by** *blast*

next

assume *in-val*$_C$*-of*: $x_C \in$ *range var*$_C$*-of*

from *this* **obtain** x_A **where** *x*$_C$*-def*: $x_C = var_C\text{-}of\ x_A$ **by** *blast*

from *is-Low*$_C'$ **have** *is-Low*$_A'$: $dma_A\ ?mem_{1A}'\ x_A = Low$

using *dma-consistent* **unfolding** *mem*$_A$*-of-def x*$_C$*-def* **by** *force*

from *is-readable*$_C'$ **have** *is-readable*$_A'$: $x_A \in \mathcal{C}_A \vee x_A \notin ?mds_A'$ *AsmNoReadOrWrite*

using *control-vars-are-A-vars x*$_C$*-def doesnt-have-mode*[*symmetric*] *var*$_C$*-of-inj inj-image-mem-iff* **by** *fast*

with *is-Low*$_A'$ *low-mds-eq*$_A'$ **have** *x*$_A$*-eq'*: $?mem_{1A}'\ x_A = ?mem_{2A}'\ x_A$

unfolding *abs.low-mds-eq-def* **by** *blast*

thus *?thesis* **by**(*simp add*: *mem*$_A$*-of-def x*$_C$*-def*)

qed

qed

qed

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 $\mathcal{R}_A\ \mathcal{R}\ P \equiv$
closed-others $\mathcal{R} \wedge$

preserves-modes-mem $\mathcal{R}$ $\wedge$
new-vars-private $\mathcal{R}$ $\wedge$
conc.closed-glob-consistent P $\wedge$
$(\forall\ c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C \longrightarrow$
$(\exists\ n\ c_{1A}'\ mds_A'\ mem_{1A}'.\ abs.neval\ \langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A\ n\ \langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A \wedge$
$(\langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A,\ \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\forall\ c_{2A}\ mem_{2A}\ c_{2C}\ mem_{2C}\ c_{2A}'\ mem_{2A}'.$
$(\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A,\ \langle\ c_{2A},\ mds_A,\ mem_{2A}\ \rangle_A) \in \mathcal{R}_A \wedge$
$(\langle\ c_{2A},\ mds_A,\ mem_{2A}\ \rangle_A,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in P \wedge$
$abs.neval\ \langle\ c_{2A},\ mds_A,\ mem_{2A}\ \rangle_A\ n\ \langle\ c_{2A}',\ mds_A',\ mem_{2A}'\ \rangle_A \longrightarrow$
$(\exists\ c_{2C}'\ mem_{2C}'.\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \rightsquigarrow_C \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C \wedge$
$(\langle\ c_{2A}',\ mds_A',\ mem_{2A}'\ \rangle_A,\ \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C,\ \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C) \in P)))))$

lemma *preserves-modes-memD*:
$[\![preserves\text{-}modes\text{-}mem\ \mathcal{R};\ (\langle c_A,\ mds_A,\ mem_A\rangle_A,\ \langle c_C,\ mds_C,\ mem_C\rangle_C) \in \mathcal{R}]\!] \Longrightarrow mem_A = mem_A\text{-}of\ mem_C \wedge mds_A = mds_A\text{-}of\ mds_C$
using *preserves-modes-mem-def2* **by** *blast*

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 $\wedge$
$(\forall\ c_{1A}\ c_{1C}\ mds_C\ mem_{1C}.$
$(\langle\ c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\ \rangle_A,\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C) \in \mathcal{R} \longrightarrow$
$(\forall\ c_{1C}'\ mds_C'\ mem_{1C}'.\ \langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C \rightsquigarrow_C \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C \longrightarrow$
$(\exists\ n\ c_{1A}'.\ abs.neval\ \langle\ c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\ \rangle_A\ n\ \langle\ c_{1A}',\ mds_A\text{-}of\ mds_C',\ mem_A\text{-}of\ mem_{1C}'\ \rangle_A \wedge$
$(\langle\ c_{1A}',\ mds_A\text{-}of\ mds_C',\ mem_A\text{-}of\ mem_{1C}'\ \rangle_A,\ \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\forall\ c_{2A}\ c_{2C}\ mem_{2C}\ c_{2A}'\ mem_{2A}'.$
$(\langle\ c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\ \rangle_A,\ \langle\ c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\ \rangle_A) \in \mathcal{R}_A \wedge$
$(\langle\ c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\ \rangle_A,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle\ c_{1C},\ mds_C,\ mem_{1C}\ \rangle_C,\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C) \in P \wedge$
$abs.neval\ \langle\ c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\ \rangle_A\ n\ \langle\ c_{2A}',\ mds_A\text{-}of\ mds_C',\ mem_{2A}'\ \rangle_A \longrightarrow$
$(\exists\ c_{2C}'\ mem_{2C}'.\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \rightsquigarrow_C \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C \wedge$

$(\langle\ c_{2A}{'},\ mds_A\text{-}of\ mds_C{'},\ mem_{2A}{'}\ \rangle_A,\ \langle\ c_{2C}{'},\ mds_C{'},\ mem_{2C}{'}\ \rangle_C) \in$
$\mathcal{R} \wedge$
$(\langle\ c_{1C}{'},\ mds_C{'},\ mem_{1C}{'}\ \rangle_C,\ \langle\ c_{2C}{'},\ mds_C{'},\ mem_{2C}{'}\ \rangle_C) \in P)))))$
apply(*rule eq-reflection*)
unfolding *secure-refinement-def*
apply(*rule conj-cong*)
apply(*fastforce*)
apply(*rule conj-cong*)
apply(*fastforce*)
apply(*rule conj-cong*)
apply(*fastforce*)
apply(*rule conj-cong*, *fastforce*)
apply(*rule iffI*)
apply(*intro allI conjI impI*)
apply((*drule spec*)+,*erule* (*1*) *impE*)
apply((*drule spec*)+,*erule* (*1*) *impE*)
using *preserves-modes-memD* **apply** *metis*
apply(*intro allI conjI impI*)
apply(*frule* (*1*) *preserves-modes-memD*, *clarify*)
apply((*drule spec*)+,*erule* (*1*) *impE*)
apply((*drule spec*)+,*erule* (*1*) *impE*)
using *preserves-modes-memD* **apply** *metis*
done

lemma *extra-vars-are-not-control-vars*:
$x \notin range\ var_C\text{-}of \Longrightarrow x \notin \mathcal{C}_C$
proof(*erule contrapos-nn*)
assume $x \in \mathcal{C}_C$
from *this* **obtain** x_A **where** $x = var_C\text{-}of\ x_A$
using *control-vars-are-A-vars* **by** *blast*
thus $x \in range\ var_C\text{-}of$ **by** *blast*
qed

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-low-mds-eq-dma*$_C$*-eq*:

assumes *abs.low-mds-eq* (*mds*$_A$*-of mds*) (*mem*$_A$*-of mem*$_{1C}$) (*mem*$_A$*-of mem*$_{2C}$)
shows *dma*$_C$ *mem*$_{1C}$ = *dma*$_C$ *mem*$_{2C}$
proof(*rule conc.dma-*$\mathcal{C}$, *rule ballI*)
fix x_C
assume $x_C \in \mathcal{C}_C$
from *this* **obtain** x_A **where** *var*$_C$*-of* $x_A = x_C$ **and** $x_A \in \mathcal{C}_A$ **using** *control-vars-are-A-vars* **by** *blast*
from *assms* ‹$x_A \in \mathcal{C}_A$› **have** (*mem*$_A$*-of mem*$_{1C}$) x_A = (*mem*$_A$*-of mem*$_{2C}$) x_A
unfolding *abs.low-mds-eq-def*
using *abs.*$\mathcal{C}$*-Low* **by** *blast*
thus (*mem*$_{1C}$ x_C) = (*mem*$_{2C}$ x_C)
using ‹*var*$_C$*-of* $x_A = x_C$› **unfolding** *mem*$_A$*-of-def* **by** *blast*
qed

lemma R_C*-ofD*:
assumes *rr*: *secure-refinement* $\mathcal{R}_A$ $\mathcal{R}$ P
assumes *in-R*: ($\langle c_{1C}, mds_C, mem_{1C}\rangle_C$, $\langle c_{2C}, mds_C{}', mem_{2C}\rangle_C$) $\in$ R_C*-of* $\mathcal{R}_A$ $\mathcal{R}$ P
shows
($\exists\, c_{1A}\ c_{2A}.$ ($\langle c_{1A}, mds_A$*-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* $mem_{2C}\rangle_A$) $\in \mathcal{R}_A$) $\wedge$
($mds_C{}' = mds_C$) $\wedge$
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 −
have $\mathcal{R}$*-preserves-modes-mem*: *preserves-modes-mem* $\mathcal{R}$
using *rr* **unfolding** *secure-refinement-def* **by** *blast*

from *in-R* **obtain** c_{1A} mds_{1A} mem_{1A} c_{2A} mds_{2A} mem_{2A} **where**
in-$\mathcal{R}_1$: ($\langle c_{1A}, mds_{1A}, mem_{1A}\rangle_A$, $\langle c_{1C}, mds_C, mem_{1C}\rangle_C$) $\in \mathcal{R}$ **and**
in-$\mathcal{R}_2$: ($\langle c_{2A}, mds_{2A}, mem_{2A}\rangle_A$, $\langle c_{2C}, mds_C{}', mem_{2C}\rangle_C$) $\in \mathcal{R}$ **and**
in-$\mathcal{R}_A$: ($\langle c_{1A}, mds_{1A}, mem_{1A}\rangle_A$, $\langle c_{2A}, mds_{2A}, mem_{2A}\rangle_A$) $\in \mathcal{R}_A$ **and**
pred-holds: ($\langle c_{1C}, mds_C, mem_{1C}\rangle_C$, $\langle c_{2C}, mds_C, mem_{2C}\rangle_C$) $\in P$ **and**
mds-eq: $mds_C = mds_C{}'$ **and**
mds-eq: *conc.low-mds-eq* mds_C mem_{1C} mem_{2C}
unfolding R_C*-of-def* **by** *force+*

from *this* $\mathcal{R}$*-preserves-modes-mem*[*simplified preserves-modes-mem-def2*, *rule-format*, *OF in-*$\mathcal{R}_1$] $\mathcal{R}$*-preserves-modes-mem*[*simplified preserves-modes-mem-def2*, *rule-format*, *OF in-*$\mathcal{R}_2$]
show *?thesis* **by** *blast*
qed

lemma R_C*-ofI*:
($\langle c_{1A}, mds_A$*-of* mds_C, mem_A*-of* $mem_{1C}\rangle_A$, $\langle c_{1C}, mds_C, mem_{1C}\rangle_C$) $\in \mathcal{R} \Longrightarrow$

$(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \Longrightarrow$
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A) \in \mathcal{R}_A \Longrightarrow$
conc.low-mds-eq $mds_C\ mem_{1C}\ mem_{2C} \Longrightarrow$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P \Longrightarrow$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
unfolding *R_C-of-def* **by** *fastforce*

lemma *R_C-of-sym*:
assumes *sym* $\mathcal{R}_A$
assumes *P-sym*: *sym* P
assumes *rr*: *secure-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-mds-eq $mds\ mem_1\ mem_2$
shows *sym* $(R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P)$
proof(*rule symI*, *clarify*)
fix $c_{1C}\ mds_C\ mem_{1C}\ \ c_{2C}\ mds_C{}'\ mem_{2C}$
assume *in-R_C-of*: $(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C{}',\ mem_{2C}\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
from *in-R_C-of* **obtain** $c_{1A}\ c_{2A}$ **where**
junk:
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A) \in \mathcal{R}_A \wedge$
$(mds_C{}' = mds_C) \wedge$ *conc.low-mds-eq* $mds_C\ mem_{1C}\ mem_{2C} \wedge$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P$
using *rr R_C-ofD* **by** *fastforce+*
hence *dma-eq*: $dma_C\ mem_{1C} = dma_C\ mem_{2C}$
using *abs-low-mds-eq-dma_C-eq*[*OF mm*] **by** *blast*
with *junk* **have** *junk′*:
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A) \in \mathcal{R}_A \wedge$
$(mds_C{}' = mds_C) \wedge$
conc.low-mds-eq $mds_C{}'\ mem_{2C}\ mem_{1C} \wedge$
$(\langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in P$
using ‹*sym* $\mathcal{R}_A$› *P-sym* **unfolding** *sym-def* **using** *conc.low-mds-eq-sym* **by** *metis*
thus $(\langle c_{2C},\ mds_C{}',\ mem_{2C}\rangle_C,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
using *R_C-ofI* **by** *auto*
qed

lemma *R_C-of-simp*:
assumes *rr*: *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
shows $(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P =$

$((\exists\, c_{1A}\ c_{2A}.\ (\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A) \in \mathcal{R}_A) \wedge$
$conc.low\text{-}mds\text{-}eq\ mds_C\ mem_{1C}\ mem_{2C} \wedge$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P)$
using *assms* **by**(*blast dest*: R_C*-ofD intro*: R_C*-ofI*)

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
apply(*simp add*: *var-asm-not-written-def* mds_A*-of-def*)
apply(*auto simp*: *inv-f-f*[*OF* var_C*-of-inj*])
done

lemma A_A*-asm-mem*:
assumes A_C*-asm-mem*: ∀ x. *case* A_C x *of None* ⇒ *True*
| *Some* $(v,\ v')$ ⇒
$mem_{1C}\ x \neq v \vee mem_{2C}\ x \neq v' \longrightarrow \neg$ *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}) $x \neq v' \longrightarrow \neg$ *var-asm-not-written* (mds_A*-of* mds_C) x
apply(*split option.splits*, *simp*, *intro allI impI*)
proof −
fix $v\ v'$
assume A_A*-not-None*: A_A*-of* A_C x = *Some* $(v,\ v')$
assume A_A*-updates-x*: mem_A*-of* mem_{1C} $x = v \longrightarrow mem_A$*-of* mem_{2C} $x \neq v'$
from A_A*-not-None* **have**
A_C*-not-None*: A_C (var_C*-of* x) = *Some* $(v,\ v')$
unfolding A_A*-of-def* **by** (*auto split*: *option.splits*)

from A_A*-updates-x* **have**
A_C*-updates-x*: mem_{1C} (var_C*-of* x) ≠ $v \vee mem_{2C}$ (var_C*-of* x) ≠ v'
unfolding mem_A*-of-def* **by** *fastforce*

from A_C*-not-None* A_C*-updates-x* A_C*-asm-mem* **have**
¬ *var-asm-not-written* mds_C (var_C*-of* x) **by** (*auto split*: *option.splits*)

thus ¬ *var-asm-not-written* (mds_A*-of* mds_C) x
by(*rule* $var\text{-}writable_A$)

qed

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$_A$*)
apply(*subst dma-consistent*[*folded mem$_A$-of-def*, *symmetric*])
apply(*rule-tac x*=*x$_A$* **in** *fun-cong*)
apply(*rule-tac f*=*dma$_A$* **in** *arg-cong*)
apply(*rule ext*)
apply(*clarsimp simp*: *apply-adaptation-def A$_A$-of-def mem$_A$-of-def split*: *option.splits*)
done

lemma *A$_A$-asm-dma*:
assumes *A$_C$-asm-dma*: $\forall$ *x*. *dma$_C$* (*mem$_{1C}$* [$\|_1$ *A$_C$*]) *x* $\neq$ *dma$_C$ mem$_{1C}$ x* $\longrightarrow$ $\neg$ *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$_A$* $\longrightarrow$ $\neg$ *var-asm-not-written* (*mds$_A$-of mds$_C$*) *x$_A$*
proof(*intro impI*)
assume *A$_A$-updates-dma*: *dma$_A$* ((*mem$_A$-of mem$_{1C}$*) [$\|_1$ *A$_A$-of A$_C$*]) *x$_A$* $\neq$ *dma$_A$* (*mem$_A$-of mem$_{1C}$*) *x$_A$*

with *dma-consistent*[*folded mem$_A$-of-def*] *dma$_A$-adaptation-eq*
have *dma$_C$* (*mem$_{1C}$* [$\|_1$ *A$_C$*]) (*var$_C$-of x$_A$*) $\neq$ *dma$_C$ mem$_{1C}$* (*var$_C$-of x$_A$*) **by**(*metis*)

with *A$_C$-asm-dma* **have** $\neg$ *var-asm-not-written mds$_C$* (*var$_C$-of x$_A$*) **by** *blast*

thus $\neg$ *var-asm-not-written* (*mds$_A$-of mds$_C$*) *x$_A$* **by** (*rule var-writable$_A$*)
qed

lemma *var$_C$-of-in-$\mathcal{C}_C$*:
assumes *x$_A$* $\in$ $\mathcal{C}_A$
shows *var$_C$-of x$_A$* $\in$ $\mathcal{C}_C$
proof −
from *assms* **obtain** *y$_A$* **where** *x$_A$* $\in$ *$\mathcal{C}$-vars$_A$ y$_A$*
unfolding *abs.$\mathcal{C}$-def* **by** *blast*

hence *var$_C$-of x$_A$* $\in$ *$\mathcal{C}$-vars$_C$* (*var$_C$-of y$_A$*)
using *$\mathcal{C}$-vars-consistent* **by** *blast*

thus *?thesis* **using** *conc.$\mathcal{C}$-def* **by** *blast*
qed

lemma *doesnt-have-mode$_C$*:
x $\notin$ *mds$_A$-of mds$_C$ m* $\Longrightarrow$ *var$_C$-of x* $\notin$ *mds$_C$ m*
by(*simp add*: *doesnt-have-mode*)

lemma *has-mode$_A$*: *var$_C$-of* $x \in$ *mds$_C$* $m \Longrightarrow x \in$ *mds$_A$-of mds$_C$* m
using *doesnt-have-mode$_C$*
by *fastforce*

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$_A$-of mem$_{1C}$*) [$\|_1$ *A_A-of A_C*]) $x =$ *Low* $\wedge$ ($x \notin$ *mds$_A$-of mds$_C$ AsmNoReadOrWrite* $\vee$ $x \in \mathcal{C}_A$) $\longrightarrow$
(*mem$_A$-of mem$_{1C}$*) [$\|_1$ *A_A-of A_C*] $x =$ (*mem$_A$-of mem$_{2C}$*) [$\|_2$ *A_A-of A_C*] x
proof(*clarify*)
assume *x-is-Low*: *dma$_A$* ((*mem$_A$-of mem$_{1C}$*) [$\|_1$ *A_A-of A_C*]) $x =$ *Low*
assume *x-is-readable*: $x \notin$ *mds$_A$-of mds$_C$ AsmNoReadOrWrite* $\vee$ $x \in \mathcal{C}_A$

from *x-is-Low* **have** *x-is-Low$_C$*: *dma$_C$* (*mem$_{1C}$* [$\|_1$ *A_C*]) (*var$_C$-of* x) = *Low*
using *dma$_A$-adaptation-eq* **by** *simp*
from *x-is-readable* **have** *var$_C$-of* $x \notin$ *mds$_C$ AsmNoReadOrWrite* $\vee$ *var$_C$-of* $x \in \mathcal{C}_C$
using *doesnt-have-mode$_C$ var$_C$-of-in-$\mathcal{C}_C$* **by** *blast*
with *A_C-sec x-is-Low$_C$* **have** *mem$_{1C}$* [$\|_1$ *A_C*] (*var$_C$-of* x) = *mem$_{2C}$* [$\|_2$ *A_C*] (*var$_C$-of* x)
by *blast*
thus (*mem$_A$-of mem$_{1C}$*) [$\|_1$ *A_A-of A_C*] $x =$ (*mem$_A$-of mem$_{2C}$*) [$\|_2$ *A_A-of A_C*] x
by(*auto simp*: *mem$_A$-of-def apply-adaptation-def A_A-of-def split*: *option.splits*)
qed

lemma *apply-adaptation$_A$*:
(*mem$_A$-of mem$_{1C}$*) [$\|_1$ *A_A-of A_C*] = *mem$_A$-of* (*mem$_{1C}$* [$\|_1$ *A_C*])
(*mem$_A$-of mem$_{1C}$*) [$\|_2$ *A_A-of A_C*] = *mem$_A$-of* (*mem$_{1C}$* [$\|_2$ *A_C*])
by(*auto simp*: *mem$_A$-of-def A_A-of-def apply-adaptation-def split*: *option.splits*)

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-of* $\mathcal{R}_A$ $\mathcal{R}$ P)
unfolding *conc.closed-glob-consistent-def*
proof(*clarify*)
fix *c_{1C} mds$_C$ mem$_{1C}$ c_{2C} mem$_{2C}$ A_C*
assume ($\langle c_{1C},$ *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
from *this rr* **obtain** *c_{1A} c_{2A}* **where**
in-$\mathcal{R}_A$: ($\langle c_{1A},$ *mds$_A$-of mds$_C$*, *mem$_A$-of mem$_{1C}\rangle_A$*, $\langle c_{2A},$ *mds$_A$-of mds$_C$*, *mem$_A$-of mem$_{2C}\rangle_A$*) $\in \mathcal{R}_A$ **and**

in-$\mathcal{R}_1$: $(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{1C}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R}$ **and**
in-$\mathcal{R}_2$: $(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_{2C}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R}$
and
mds-eq: *conc.low-mds-eq* $mds_C\ mem_{1C}\ mem_{2C}$
and
P: $(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P$
by (*blast dest*: R_C*-ofD*)
assume A_C*-asm-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-asm-not-written* $mds_C\ x$
hence A_A*-asm-mem*: $\forall x.$ *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-asm-not-written* $(mds_A\text{-}of\ mds_C)\ x$
by(*metis* A_A*-asm-mem*)

assume A_C*-asm-dma*: $\forall x.\ dma_C\ (mem_{1C}\ [\|_1\ A_C])\ x \neq dma_C\ mem_{1C}\ x \longrightarrow \neg$ *var-asm-not-written* $mds_C\ x$
hence A_A*-asm-dma*: $\forall x_A.\ dma_A\ ((mem_A\text{-}of\ mem_{1C})\ [\|_1\ (A_A\text{-}of\ A_C)])\ x_A \neq dma_A\ (mem_A\text{-}of\ mem_{1C})\ x_A \longrightarrow \neg$ *var-asm-not-written* $(mds_A\text{-}of\ mds_C)\ x_A$
by(*metis* A_A*-asm-dma*)

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

from *rr* **have** *others*: *closed-others* $\mathcal{R}$
unfolding *secure-refinement-def* **by** *blast*
from *rr* **have** *P-cgc*: *conc.closed-glob-consistent* P
unfolding *secure-refinement-def* **by** *blast*
let *?*$mem_{1C}' = (mem_{1C}\ [\|_1\ A_C])$ **and**
?$mem_{2C}' = (mem_{2C}\ [\|_2\ A_C])$ **and**
?$mem_{1A} = (mem_A\text{-}of\ mem_{1C})$ **and**
?$mem_{2A} = (mem_A\text{-}of\ mem_{2C})$ **and**
?$mem_{1A}' = (mem_A\text{-}of\ mem_{1C})\ [\|_1\ A_A\text{-}of\ A_C]$ **and**
?$mem_{2A}' = (mem_A\text{-}of\ mem_{2C})\ [\|_2\ A_A\text{-}of\ A_C]$

have *mem'-simps*:
?$mem_{1A}' = mem_A\text{-}of$ *?*mem_{1C}'
?$mem_{2A}' = mem_A\text{-}of$ *?*mem_{2C}' **by**(*simp add*: *apply-adaptation*$_A$)+

from *cgc in-*$\mathcal{R}_A$ A_A*-asm-mem* A_A*-asm-dma* A_A*-sec* **have**

in-$\mathcal{R}_A$': $(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ (mem_A\text{-}of\ mem_{1C})\ [\|_1\ A_A\text{-}of\ A_C]\rangle_A,\ \langle c_{2A},\ mds_A\text{-}of\ mds_C,\ (mem_A\text{-}of\ mem_{2C})\ [\|_2\ A_A\text{-}of\ A_C]\rangle_A) \in \mathcal{R}_A$ **unfolding** *abs.closed-glob-consistent-def* **by** *blast*

from *A_C-asm-mem A_C-asm-dma* **have**
A_C-asm-mem$_1$': $\forall x.\ mem_{1C}\ x \neq ?mem_{1C}'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds_C\ x$ **and**
A_C-asm-dma$_1$': $\forall x.\ dma_C\ mem_{1C}\ x \neq dma_C\ ?mem_{1C}'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds_C\ x$
unfolding *apply-adaptation-def* **by**(*force split*: *option.splits*)+

from *A_C-asm-mem* **have**
A_C-asm-mem$_2$': $\forall x.\ mem_{2C}\ x \neq ?mem_{2C}'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds_C\ x$
unfolding *apply-adaptation-def* **by**(*force split*: *option.splits*)

from *in-$\mathcal{R}_1$ A_C-asm-mem$_1$' A_C-asm-dma$_1$' others* **have**
in-$\mathcal{R}_1$': $(\langle c_{1A},\ mds_A\text{-}of\ mds_C,\ ?mem_{1A}'\rangle_A,\ \langle c_{1C},\ mds_C,\ ?mem_{1C}'\rangle_C) \in \mathcal{R}$
unfolding *closed-others-def mem'-simps* **by** *blast*

from *mm*[*OF in-$\mathcal{R}_A$*] **have**
dma_C-eq: $dma_C\ mem_{1C} = dma_C\ mem_{2C}$ **by**(*rule abs-low-mds-eq-dma$_C$-eq*)
have *dma_C-eq'*: $dma_C\ ?mem_{1C}' = dma_C\ ?mem_{2C}'$
apply(*rule abs-low-mds-eq-dma$_C$-eq*[*OF mm*])
apply(*simp add*: *mem'-simps*[*symmetric*])
by(*rule in-$\mathcal{R}_A$'*)
from *dma_C-eq dma_C-eq' A_C-asm-dma$_1$'* **have**
A_C-asm-dma$_2$': $\forall x.\ dma_C\ mem_{2C}\ x \neq dma_C\ ?mem_{2C}'\ x \longrightarrow \neg\ var\text{-}asm\text{-}not\text{-}written\ mds_C\ x$
by *simp*

from *in-$\mathcal{R}_2$ A_C-asm-mem$_2$' A_C-asm-dma$_2$' others* **have**
in-$\mathcal{R}_2$': $(\langle c_{2A},\ mds_A\text{-}of\ mds_C,\ ?mem_{2A}'\rangle_A,\ \langle c_{2C},\ mds_C,\ ?mem_{2C}'\rangle_C) \in \mathcal{R}$
unfolding *closed-others-def mem'-simps* **by** *blast*

have *mds-eq'*: *conc.low-mds-eq mds_C $?mem_{1C}'$ $?mem_{2C}'$*
using *A_C-sec* **unfolding** *conc.low-mds-eq-def* **by** *blast*

from *P P-cgc A_C-asm-mem A_C-asm-dma A_C-sec* **have** *P'*: $(\langle c_{1C},\ mds_C,\ ?mem_{1C}'\rangle_C,\ \langle c_{2C},\ mds_C,\ ?mem_{2C}'\rangle_C) \in P$
unfolding *conc.closed-glob-consistent-def* **by** *blast*
from *in-$\mathcal{R}_A$' in-$\mathcal{R}_1$' in-$\mathcal{R}_2$' mem'-simps R_C-ofI mds-eq' P'* **show**
$(\langle c_{1C},\ mds_C,\ ?mem_{1C}'\rangle_C,\ \langle c_{2C},\ mds_C,\ ?mem_{2C}'\rangle_C) \in R_C\text{-}of\ \mathcal{R}_A\ \mathcal{R}\ P$
by(*metis*)
qed

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*$_{1C}$: $\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C \rightsquigarrow_C \langle c_{1C}',\ mds_C',\ mem_{1C}'\rangle_C$
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 −
from *rr in-R_C-of* **have**
P: $(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P$
by(*blast dest*: *R_C-ofD*)

let *?mds_A* = *mds_A-of mds_C* **and**
?mem_{1A} = *mem_A-of mem_{1C}* **and**
?mem_{2A} = *mem_A-of mem_{2C}* **and**
?mds_A' = *mds_A-of mds_C'* **and**
?mem_{1A}' = *mem_A-of mem_{1C}'*

from *rr in-R_C-of* **obtain** $c_{1A}\ c_{2A}$ **where**
in-$\mathcal{R}_1$: $(\langle c_{1A},\ ?mds_A,\ ?mem_{1A}\rangle_A,\ \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R}$ **and**
in-$\mathcal{R}_2$: $(\langle c_{2A},\ ?mds_A,\ ?mem_{2A}\rangle_A,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R}$ **and**
in-$\mathcal{R}_A$: $(\langle c_{1A},\ ?mds_A,\ ?mem_{1A}\rangle_A,\ \langle c_{2A},\ ?mds_A,\ ?mem_{2A}\rangle_A) \in \mathcal{R}_A$ **and**
low-mds-mds$_C$: *conc.low-mds-eq mds_C mem_{1C} mem_{2C}*
by(*blast dest*: *R_C-ofD*)+

from *rr in-$\mathcal{R}_1$ in-$\mathcal{R}_A$ in-$\mathcal{R}_2$ step*$_{1C}$ **obtain** $n\ c_{1A}'$ **where**
a: $(abs.neval\ \langle\ c_{1A},\ ?mds_A,\ ?mem_{1A}\ \rangle_A\ n\ \langle\ c_{1A}',\ ?mds_A',\ ?mem_{1A}'\ \rangle_A \wedge$
$(\langle\ c_{1A}',\ ?mds_A',\ ?mem_{1A}'\ \rangle_A,\ \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\forall\, c_{2A}'\ mem_{2A}'.$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\ \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P \wedge$
$abs.neval\ \langle\ c_{2A},\ ?mds_A,\ ?mem_{2A}\ \rangle_A\ n\ \langle\ c_{2A}',\ ?mds_A',\ mem_{2A}'\ \rangle_A \longrightarrow$
$(\exists\ c_{2C}'\ mem_{2C}'.\ \langle\ c_{2C},\ mds_C,\ mem_{2C}\ \rangle_C \rightsquigarrow_C \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C \wedge$
$(\langle\ c_{2A}',\ ?mds_A',\ mem_{2A}'\ \rangle_A,\ \langle\ c_{2C}',\ mds_C',\ mem_{2C}'\ \rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C}',\ mds_C',\ mem_{1C}'\rangle_C,\ \langle c_{2C}',\ mds_C',\ mem_{2C}'\rangle_C) \in P)))$
unfolding *secure-refinement-def2*
by *metis*

show *?thesis*
proof −
from *a* **have** *neval*$_{1A}$: $abs.neval\ \langle c_{1A},\ ?mds_A,\ ?mem_{1A}\rangle_A\ n\ \langle c_{1A}',\ ?mds_A',\ ?mem_{1A}'\rangle_A$ **and**
in-$\mathcal{R}_1$': $(\langle c_{1A}',\ ?mds_A',\ ?mem_{1A}'\rangle_A,\ \langle c_{1C}',\ mds_C',\ mem_{1C}'\rangle_C) \in \mathcal{R}$
by *blast*+
from *strong-low-bisim-neval*[*OF neval*$_{1A}$ *in-$\mathcal{R}_A$ bisim*] **obtain** $c_{2A}'\ mem_{2A}'$ **where**
neval$_{2A}$: $abs.neval\ \langle c_{2A},\ ?mds_A,\ ?mem_{2A}\rangle_A\ n\ \langle c_{2A}',\ ?mds_A',\ mem_{2A}'\rangle_A$ **and**
in-$\mathcal{R}_A$'-help: $(\langle c_{1A}',\ ?mds_A',\ ?mem_{1A}'\rangle_A,\ \langle c_{2A}',\ ?mds_A',\ mem_{2A}'\rangle_A) \in \mathcal{R}_A$
unfolding *abs.strong-low-bisim-mm-def*
by *blast*

```
    from a in-R_A in-R_2 neval_2A P obtain c_2C' mem_2C' where
         step_2C: ⟨c_2C, mds_C, mem_2C⟩_C ⇝_C ⟨c_2C', mds_C', mem_2C'⟩_C and
          in-R_2'-help: (⟨c_2A', ?mds_A', mem_2A'⟩_A, ⟨c_2C', mds_C', mem_2C'⟩_C) ∈ R
and
         P': (⟨c_1C', mds_C', mem_1C'⟩_C, ⟨c_2C', mds_C', mem_2C'⟩_C) ∈ P
      by blast

    let ?mem_2A' = mem_A-of mem_2C'
    from in-R_2'-help rr preserves-modes-memD have mem_2A' = ?mem_2A'
      unfolding secure-refinement-def by metis
    with in-R_2'-help in-R_A'-help have
      in-R_2': (⟨c_2A', ?mds_A', ?mem_2A'⟩_A, ⟨c_2C', mds_C', mem_2C'⟩_C) ∈ R and
      in-R_A': (⟨c_1A', ?mds_A', ?mem_1A'⟩_A, ⟨c_2A', ?mds_A', ?mem_2A'⟩_A) ∈ R_A
      by simp+

    have conc.low-mds-eq mds_C' mem_1C' mem_2C'
      apply(rule new-vars-private-does-the-thing[where R=R, OF - in-R_1 in-R_2
step_1C step_2C low-mds-mds_C])
       using rr apply(fastforce simp: secure-refinement-def)
      using in-R_A' bisim unfolding abs.strong-low-bisim-mm-def by blast

    with step_2C in-R_1' in-R_2' in-R_A' in-R_2' P' show ?thesis
      by(blast intro: R_C-ofI)
  qed
qed
```

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 R_A
  assumes rr: secure-refinement R_A R P
  assumes P-sym: sym P
  shows conc.strong-low-bisim-mm (R_C-of R_A R P)
  unfolding conc.strong-low-bisim-mm-def
  apply(intro conjI)
    apply(rule R_C-of-sym)
    using abs rr P-sym unfolding abs.strong-low-bisim-mm-def apply blast+
   apply(rule R_C-of-closed-glob-consistent)
    using abs unfolding abs.strong-low-bisim-mm-def apply blast+
   using rr apply blast
  apply safe
   apply(fastforce simp: R_C-of-def)
  apply(rule R_C-of-local-preservation)
     apply(rule rr)
    apply(rule abs)
   apply assumption+
  done
```

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-refinement-simpler $\mathcal{R}_A\ \mathcal{R}\ P$ *abs-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-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-steps*
shows *secure-refinement* $\mathcal{R}_A\ \mathcal{R}\ P$
unfolding *secure-refinement-def*
proof(*safe*)
from *rrs* **show** *closed-others* $\mathcal{R}$
unfolding *secure-refinement-simpler-def* **by** *blast*
next
from *rrs* **show** *preserves-modes-mem* $\mathcal{R}$
unfolding *secure-refinement-simpler-def* **by** *blast*
next
from *rrs* **show** *new-vars-private* $\mathcal{R}$
unfolding *secure-refinement-simpler-def* **by** *blast*
next
fix $c_{1A}\ mds_A\ mem_{1A}\ c_{1C}\ mds_C\ mem_{1C}\ c_{1C}'\ mds_C'\ mem_{1C}'$
let *?n* = *abs-steps* $\langle c_{1A},mds_A,mem_{1A}\rangle_A\ \langle c_{1C},mds_C,mem_{1C}\rangle_C$
assume *in-*$\mathcal{R}_1$: $(\langle c_{1A},\ mds_A,\ mem_{1A}\rangle_A, \langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C) \in \mathcal{R}$
and $eval_{1C}$: $\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C \rightsquigarrow_C \langle c_{1C}',\ mds_C',\ mem_{1C}'\rangle_C$
with *rrs* **obtain** $c_{1A}'\ mds_A'\ mem_{1A}'$ **where**
$neval_1$: *abs.neval* $\langle\ c_{1A},\ mds_A,\ mem_{1A}\ \rangle_A$ *?n* $\langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A$ **and**
in-$\mathcal{R}_1$': $(\langle\ c_{1A}',\ mds_A',\ mem_{1A}'\ \rangle_A, \langle\ c_{1C}',\ mds_C',\ mem_{1C}'\ \rangle_C) \in \mathcal{R}$
unfolding *secure-refinement-simpler-def* **by** *metis*
have $(\forall\ c_{2A}\ mem_{2A}\ c_{2C}\ mem_{2C}\ c_{2A}'\ mem_{2A}'.$
$(\langle c_{1A},\ mds_A,\ mem_{1A}\rangle_A, \langle c_{2A},\ mds_A,\ mem_{2A}\rangle_A) \in \mathcal{R}_A \wedge$
$(\langle c_{2A},\ mds_A,\ mem_{2A}\rangle_A, \langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C},\ mds_C,\ mem_{1C}\rangle_C,\langle c_{2C},\ mds_C,\ mem_{2C}\rangle_C) \in P \wedge abs.neval\ \langle c_{2A},$

mds_A, $mem_{2A}\rangle_A$ *?n* $\langle c_{2A}{}'$, $mds_A{}'$, $mem_{2A}{}'\rangle_A \longrightarrow$
$(\exists\, c_{2C}{}'\ mem_{2C}{}'.$
$\langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C \wedge$
$(\langle c_{2A}{}', mds_A{}', mem_{2A}{}'\rangle_A, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C}{}', mds_C{}', mem_{1C}{}'\rangle_C, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in P))$

proof(*clarsimp*)
fix c_{2A} mem_{2A} c_{2C} mem_{2C} $c_{2A}{}'$ $mem_{2A}{}'$
assume
in-$\mathcal{R}_A$: $(\langle c_{1A}, mds_A, mem_{1A}\rangle_A, \langle c_{2A}, mds_A, mem_{2A}\rangle_A) \in \mathcal{R}_A$ **and**
in-$\mathcal{R}_2$: $(\langle c_{2A}, mds_A, mem_{2A}\rangle_A, \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \in \mathcal{R}$ **and**
$neval_2$: *abs.neval* $\langle c_{2A}, mds_A, mem_{2A}\rangle_A$ *?n* $\langle c_{2A}{}', mds_A{}', mem_{2A}{}'\rangle_A$ **and**
in-P: $(\langle c_{1C}, mds_C, mem_{1C}\rangle_C, \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \in P$
have $\forall\, c_{2C}{}'\ mem_{2C}{}'.\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C \longrightarrow (\langle c_{2A}{}', mds_A{}', mem_{2A}{}'\rangle_A, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in \mathcal{R} \wedge (\langle c_{1C}{}', mds_C{}', mem_{1C}{}'\rangle_C, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in P$
proof(*clarify*)
fix $c_{2C}{}'\ mem_{2C}{}'$
assume $eval_{2C}$: $\langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C$
from *in-*$\mathcal{R}_2$ $eval_{2C}$ *in-P rrs* **obtain**
$c_{2A}{}''\ mds_A{}''\ mem_{2A}{}''$ **where**
$neval_2{}'$: *abs.neval* $\langle\ c_{2A}, mds_A, mem_{2A}\ \rangle_A$ (*abs-steps* $\langle c_{2A}, mds_A, mem_{2A}\rangle_A$ $\langle c_{2C}, mds_C, mem_{2C}\rangle_C$) $\langle\ c_{2A}{}'', mds_A{}'', mem_{2A}{}''\ \rangle_A$ **and**
in-$\mathcal{R}_2{}'$: $(\langle\ c_{2A}{}'', mds_A{}'', mem_{2A}{}''\ \rangle_A, \langle\ c_{2C}{}', mds_C{}', mem_{2C}{}'\ \rangle_C) \in \mathcal{R}$
unfolding *secure-refinement-simpler-def* **by** *blast*
let *?n*$'$ = (*abs-steps* $\langle c_{2A}, mds_A, mem_{2A}\rangle_A$ $\langle c_{2C}, mds_C, mem_{2C}\rangle_C$)
from *rrs* **have** *pe*: *simpler-refinement-safe* $\mathcal{R}_A$ $\mathcal{R}$ *P abs-steps*
unfolding *secure-refinement-simpler-def* **by** *blast*
with *in-*$\mathcal{R}_A$ *in-*$\mathcal{R}_1$ *in-*$\mathcal{R}_2$ *in-P*
have *?n*$'$ = *?n*
unfolding *simpler-refinement-safe-def* **by** *fastforce*
with $neval_2$ $neval_2{}'$ *abs.neval-det*
have [*simp*]: $c_{2A}{}'' = c_{2A}{}'$ **and** [*simp*]: $mds_A{}'' = mds_A{}'$ **and** [*simp*]: $mem_{2A}{}'' = mem_{2A}{}'$
by *auto*
from *in-*$\mathcal{R}_2{}'$ **have** *in-*$\mathcal{R}_2{}'$: $(\langle c_{2A}{}', mds_A{}', mem_{2A}{}'\rangle_A, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in \mathcal{R}$ **by** *simp*
from $eval_{1C}$ $eval_{2C}$ *in-P* **have**
in-P$'$: $(\langle c_{1C}{}', mds_C{}', mem_{1C}{}'\rangle_C, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in P$
using *rrs* **unfolding** *secure-refinement-simpler-def simpler-refinement-safe-def*
using *in-*$\mathcal{R}_A$ *in-*$\mathcal{R}_1$ *in-*$\mathcal{R}_2$ *in-P* **by** *auto*
with *in-*$\mathcal{R}_2{}'$
show $(\langle c_{2A}{}', mds_A{}', mem_{2A}{}'\rangle_A, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in \mathcal{R} \wedge (\langle c_{1C}{}', mds_C{}', mem_{1C}{}'\rangle_C, \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C) \in P$ **by** *blast*
qed
moreover have $\exists\, c_{2C}{}'\ mem_{2C}{}'.\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}{}', mds_C{}', mem_{2C}{}'\rangle_C$
proof −
from *rrs* **have** *pe*: *simpler-refinement-safe* $\mathcal{R}_A$ $\mathcal{R}$ *P abs-steps*

unfolding *secure-refinement-simpler-def* **by** *blast*
with *in-$\mathcal{R}_A$ in-$\mathcal{R}_1$ in-$\mathcal{R}_2$ in-P* **have** $stops_C\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C = stops_C\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C$
unfolding *simpler-refinement-safe-def* **by** *blast*
moreover from $eval_{1C}$ **have** $\neg\ stops_C\ \langle c_{1C}, mds_C, mem_{1C}\rangle_C$
unfolding *$stops_C$-def* **by** *blast*
ultimately have $\neg\ stops_C\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C$
by *simp*
from *this* **obtain** $c_{2C}'\ mds_C''\ mem_{2C}''$ **where** $eval_{2C}'$: $\langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}', mds_C'', mem_{2C}''\rangle_C$
unfolding *$stops_C$-def* **by** *auto*
with *pe* $eval_{1C}$ *in-$\mathcal{R}_A$ in-$\mathcal{R}_1$ in-$\mathcal{R}_2$ in-P* **have** *in-P'*: $(\langle c_{1C}', mds_C', mem_{1C}'\rangle_C, \langle c_{2C}', mds_C'', mem_{2C}''\rangle_C) \in P$
and [*simp*]: $mds_C'' = mds_C'$
unfolding *simpler-refinement-safe-def* **by** *blast+*
from *in-P'* $eval_{2C}'$
show $\exists\, c_{2C}'\ mem_{2C}'.\ \langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}', mds_C', mem_{2C}'\rangle_C$
by *fastforce*
qed
ultimately show
$\exists\, c_{2C}'\ mem_{2C}'.$
$\langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}', mds_C', mem_{2C}'\rangle_C \wedge (\langle c_{2A}', mds_A', mem_{2A}'\rangle_A, \langle c_{2C}', mds_C', mem_{2C}'\rangle_C) \in \mathcal{R} \wedge (\langle c_{1C}', mds_C', mem_{1C}'\rangle_C, \langle c_{2C}', mds_C', mem_{2C}'\rangle_C) \in P$
by *blast*
qed
with $neval_1$ *in-$\mathcal{R}_1$ in-$\mathcal{R}_1$'*
show $\exists\, n\ c_{1A}'\ mds_A'\ mem_{1A}'.$
$abs.neval\ \langle c_{1A}, mds_A, mem_{1A}\rangle_A\ n\ \langle c_{1A}', mds_A', mem_{1A}'\rangle_A \wedge$
$(\langle c_{1A}', mds_A', mem_{1A}'\rangle_A, \langle c_{1C}', mds_C', mem_{1C}'\rangle_C) \in \mathcal{R} \wedge$
$(\forall\, c_{2A}\ mem_{2A}\ c_{2C}\ mem_{2C}\ c_{2A}'\ mem_{2A}'.$
$(\langle c_{1A}, mds_A, mem_{1A}\rangle_A, \langle c_{2A}, mds_A, mem_{2A}\rangle_A) \in \mathcal{R}_A \wedge$
$(\langle c_{2A}, mds_A, mem_{2A}\rangle_A, \langle c_{2C}, mds_C, mem_{2C}\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C}, mds_C, mem_{1C}\rangle_C, \langle c_{2C}, mds_C, mem_{2C}\rangle_C)$
$\in P \wedge$
$abs.neval\ \langle c_{2A}, mds_A, mem_{2A}\rangle_A\ n\ \langle c_{2A}', mds_A', mem_{2A}'\rangle_A \longrightarrow$
$(\exists\, c_{2C}'\ mem_{2C}'.$
$\langle c_{2C}, mds_C, mem_{2C}\rangle_C \rightsquigarrow_C \langle c_{2C}', mds_C', mem_{2C}'\rangle_C \wedge$
$(\langle c_{2A}', mds_A', mem_{2A}'\rangle_A, \langle c_{2C}', mds_C', mem_{2C}'\rangle_C) \in \mathcal{R} \wedge$
$(\langle c_{1C}', mds_C', mem_{1C}'\rangle_C, \langle c_{2C}', mds_C', mem_{2C}'\rangle_C)$
$\in P))$
by *auto*
next
show *conc.closed-glob-consistent P*
using *rrs* **unfolding** *secure-refinement-simpler-def* **by** *blast*
qed

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

definition
secure-refinement-simple
where
secure-refinement-simple $\mathcal{R}_A\ \mathcal{R}\ abs\text{-}steps \equiv$
closed-others $\mathcal{R} \wedge$
preserves-modes-mem $\mathcal{R} \wedge$
new-vars-private $\mathcal{R} \wedge$
simple-refinement-safe $\mathcal{R}_A\ \mathcal{R}\ abs\text{-}steps \wedge$
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 *Isimple-closed-glob-consistent*:
conc.closed-glob-consistent Isimple
by(*auto simp*: *conc.closed-glob-consistent-def Isimple-def*)

lemma *secure-refinement-simple*:
assumes *srs*: *secure-refinement-simple* $\mathcal{R}_A$ $\mathcal{R}$ *abs-steps*
shows *secure-refinement-simpler* $\mathcal{R}_A$ $\mathcal{R}$ *Isimple abs-steps*
unfolding *secure-refinement-simpler-def*
proof(*safe* | *clarsimp*)+
from *srs* **show** *closed-others* $\mathcal{R}$
unfolding *secure-refinement-simple-def* **by** *blast*
next
from *srs* **show** *preserves-modes-mem* $\mathcal{R}$
unfolding *secure-refinement-simple-def* **by** *blast*
next
from *srs* **show** *new-vars-private* $\mathcal{R}$
unfolding *secure-refinement-simple-def* **by** *blast*
next
show *conc.closed-glob-consistent Isimple* **by** (*rule Isimple-closed-glob-consistent*)
next
from *srs* **have** *safe*: *simple-refinement-safe* $\mathcal{R}_A$ $\mathcal{R}$ *abs-steps*
unfolding *secure-refinement-simple-def* **by** *blast*
from *srs* **have** *simple*: *bisim-simple* $\mathcal{R}_A$
unfolding *secure-refinement-simple-def* **by** *fastforce*

from *safe simple* **show** *simpler-refinement-safe* $\mathcal{R}_A$ $\mathcal{R}$ *Isimple abs-steps*
by(*fastforce simp*: *simpler-refinement-safe-def Isimple-def simple-refinement-safe-def bisim-simple-def*)
next
fix c_{1A} mds_A mem_{1A} c_{1C} mds_C mem_{1C} c_{1C}' mds_C' mem_{1C}'
show $(\langle c_{1A}, mds_A, mem_{1A}\rangle_A, \langle c_{1C}, mds_C, mem_{1C}\rangle_C) \in \mathcal{R} \Longrightarrow$
$\langle c_{1C}, mds_C, mem_{1C}\rangle_C \rightsquigarrow_C \langle c_{1C}', mds_C', mem_{1C}'\rangle_C \Longrightarrow$
$\exists\, c_{1A}'\ mds_A'\ mem_{1A}'.$
abs.neval $\langle c_{1A}, mds_A, mem_{1A}\rangle_A$ (*abs-steps* $\langle c_{1A}, mds_A, mem_{1A}\rangle_A$ $\langle c_{1C}, mds_C, mem_{1C}\rangle_C$)
$\langle c_{1A}', mds_A', mem_{1A}'\rangle_A \wedge$
$(\langle c_{1A}', mds_A', mem_{1A}'\rangle_A, \langle c_{1C}', mds_C', mem_{1C}'\rangle_C) \in \mathcal{R}$
using *srs* **unfolding** *secure-refinement-simple-def* **by** *blast*
qed

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*))
apply(*clarsimp simp*: mds_A-*of-def*)
apply(*rule ext*)
apply(*force simp*: *image-def inv-f-f*[*OF* var_C-*of-inj*])
done

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*). $y \neq x$})
apply(*clarsimp simp*: mds_A-*of-def*)
apply(*rule ext*)
apply (*force simp*: *image-def inv-f-f*[*OF* var_C-*of-inj*])
done

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

lemma *secure-refinement-loc-reach*:
assumes *rr*: *secure-refinement* $\mathcal{R}_A$ $\mathcal{R}$ P
assumes *in-*$\mathcal{R}$: ($\langle c_A, mds_A, mem_A\rangle_A$, $\langle c_C, mds_C, mem_C\rangle_C$) $\in \mathcal{R}$
assumes *loc-reach*$_C$: $\langle c_C', mds_C', mem_C'\rangle_C \in$ *conc.loc-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-reach* $\langle c_A, mds_A, mem_A\rangle_A$
using *loc-reach*$_C$ **proof**(*induct rule*: *conc.loc-reach.induct*)
case (*refl*) **show** *?case*
using *in-*$\mathcal{R}$ *abs.loc-reach.refl* **by** *force*
next
case (*step* c_C' mds_C' mem_C' c_C'' mds_C'' mem_C'')
from *step*(*2*) **obtain** c_A' mds_A' mem_A' **where**
in-$\mathcal{R}'$: ($\langle c_A', mds_A', mem_A'\rangle_A$, $\langle c_C', mds_C', mem_C'\rangle_C$) $\in \mathcal{R}$ **and**
loc-reach$_A$: $\langle c_A', mds_A', mem_A'\rangle_A \in$ *abs.loc-reach* $\langle c_A, mds_A, mem_A\rangle_A$
by *blast*
from *rr in-*$\mathcal{R}'$ *step*(*3*)

obtain n c_A'' mds_A'' mem_A'' **where**
$neval_A$: $abs.neval\ \langle\ c_A',\ mds_A',\ mem_A'\ \rangle_A\ n\ \langle\ c_A'',\ mds_A'',\ mem_A''\ \rangle_A$ **and**
$in\text{-}\mathcal{R}''$: $(\langle\ c_A'',\ mds_A'',\ mem_A''\ \rangle_A,\ \langle\ c_C'',\ mds_C'',\ mem_C''\ \rangle_C) \in \mathcal{R}$
unfolding *secure-refinement-def* **by** *blast*
from $neval_A$ $loc\text{-}reach_A$ **have** $\langle c_A'',\ mds_A'',\ mem_A''\rangle_A \in abs.loc\text{-}reach\ \langle c_A,\ mds_A,\ mem_A\rangle_A$
using *abs.neval-loc-reach*
by *blast*
with $in\text{-}\mathcal{R}''$ **show** *?case* **by** *blast*
next
case (*mem-diff* c_C' mds_C' mem_C' mem_C'')
from *mem-diff*(*2*) **obtain** c_A' mds_A' mem_A' **where**
$in\text{-}\mathcal{R}'$: $(\langle c_A',\ mds_A',\ mem_A'\rangle_A,\ \langle c_C',\ mds_C',\ mem_C'\rangle_C) \in \mathcal{R}$ **and**
$loc\text{-}reach_A$: $\langle c_A',\ mds_A',\ mem_A'\rangle_A \in abs.loc\text{-}reach\ \langle c_A,\ mds_A,\ mem_A\rangle_A$
by *blast*
from *rr* **have** *mm*: *preserves-modes-mem* $\mathcal{R}$ **and** *co*: *closed-others* $\mathcal{R}$
unfolding *secure-refinement-def* **by** *blast+*
from *preserves-modes-memD*[*OF mm in-*$\mathcal{R}'$] **have**
$mem_A'\text{-}def$: $mem_A' = mem_A\text{-}of\ mem_C'$ **and** $mds_A'\text{-}def$: $mds_A' = mds_A\text{-}of\ mds_C'$
by *simp+*
hence $in\text{-}\mathcal{R}'$: $(\langle c_A',\ mds_A\text{-}of\ mds_C',\ mem_A\text{-}of\ mem_C'\rangle_A,\ \langle c_C',\ mds_C',\ mem_C'\rangle_C) \in \mathcal{R}$
and $loc\text{-}reach_A$: $(\langle c_A',\ mds_A\text{-}of\ mds_C',\ mem_A\text{-}of\ mem_C'\rangle_A) \in abs.loc\text{-}reach\ \langle c_A,\ mds_A,\ mem_A\rangle_A$
using $in\text{-}\mathcal{R}'$ $loc\text{-}reach_A$ **by** *simp+*
with *mem-diff*(*3*) *co*
have $(\langle c_A',\ mds_A\text{-}of\ mds_C',\ mem_A\text{-}of\ mem_C''\rangle_A,\ \langle c_C',\ mds_C',\ mem_C''\rangle_C) \in \mathcal{R}$
unfolding *closed-others-def* **by** *blast*
moreover have $\langle c_A',\ mds_A\text{-}of\ mds_C',\ mem_A\text{-}of\ mem_C''\rangle_A \in abs.loc\text{-}reach\ \langle c_A,\ mds_A,\ mem_A\rangle_A$
apply(*rule abs.loc-reach.mem-diff*)
apply(*rule* $loc\text{-}reach_A$)
using *mem-diff*(*3*)
using *calculation* $in\text{-}\mathcal{R}'$ $in\text{-}\mathcal{R}\text{-}dma'$ $mem_A\text{-}of\text{-}def$ *mm* $var\text{-}writable_A$ **by** *fastforce*

ultimately show *?case* **by** *blast*
qed

definition *preserves-local-guarantee-compliance* ::
('Com_A, 'Var_A, '*Val*, 'Com_C, 'Var_C) *state-relation* ⇒ *bool*
where
preserves-local-guarantee-compliance $\mathcal{R} \equiv$
$\forall\, cm_A\ mem_A\ cm_C\ mem_C.$
$abs.respects\text{-}own\text{-}guarantees\ cm_A \wedge$
$((cm_A,\ mem_A),\ (cm_C,\ mem_C)) \in \mathcal{R} \longrightarrow$
$conc.respects\text{-}own\text{-}guarantees\ cm_C$

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 $(c_C,\ mds_C)$
unfolding *preserves-local-guarantee-compliance-def*
by *simp*

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* $\mathcal{R}$
unfolding *preserves-locally-sound-mode-use-def*
proof(*clarsimp*)
fix $c_A\ mds_A\ mem_A\ c_C\ mds_C\ mem_C$
assume *locally-sound*$_A$: *abs.locally-sound-mode-use* $\langle c_A,\ mds_A,\ mem_A\rangle_A$ **and**
in-$\mathcal{R}$: $(\langle c_A,\ mds_A,\ mem_A\rangle_A,\ \langle c_C,\ mds_C,\ mem_C\rangle_C) \in \mathcal{R}$

show *conc.locally-sound-mode-use* $\langle c_C,\ mds_C,\ mem_C\rangle_C$
unfolding *conc.locally-sound-mode-use-def2*
proof(*clarsimp*)
fix $c_C'\ mds_C'\ mem_C'$
assume *loc-reach*$_C$: $\langle c_C',\ mds_C',\ mem_C'\rangle_C \in$ *conc.loc-reach* $\langle c_C,\ mds_C,\ mem_C\rangle_C$

from *rr in-*$\mathcal{R}$ *loc-reach*$_C$
obtain $c_A'\ mds_A'\ mem_A'$ **where**
in-$\mathcal{R}'$: $(\langle c_A',\ mds_A',\ mem_A'\rangle_A,\ \langle c_C',\ mds_C',\ mem_C'\rangle_C) \in \mathcal{R}$ **and**
loc-reach$_A$: $\langle c_A',\ mds_A',\ mem_A'\rangle_A \in$ *abs.loc-reach* $\langle c_A,\ mds_A,\ mem_A\rangle_A$
using *secure-refinement-loc-reach* **by** *blast*

from *locally-sound*$_A$ *loc-reach*$_A$
have *respects-guarantees*$_A'$: *abs.respects-own-guarantees* $(c_A',\ mds_A')$
unfolding *abs.locally-sound-mode-use-def2* **by** *auto*

with *preserves-guarantee-compliance in-*$\mathcal{R}'$
show *conc.respects-own-guarantees* $(c_C',\ mds_C')$
unfolding *preserves-local-guarantee-compliance-def* **by** *blast*
qed
qed

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*
  by(*unfold-locales*, *simp-all*)

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

lemma [*simp*]:
  gen-refine.new-vars-private $\mathcal{R}$
  unfolding *gen-refine.new-vars-private-def*
  by *simp*

definition
  preserves-modes-mem :: ('Com_A, '*Var*, '*Val*, 'Com_C, '*Var*) *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 ⟩$_C$) ∈ $\mathcal{R}$ ⟶
    $mem_A = mem_C \wedge mds_A = mds_C$)

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

lemma [*simp*]:
  $gen\text{-}refine.mds_A\text{-}of\ x = x$
  by(*simp add*: $gen\text{-}refine.mds_A\text{-}of\text{-}def$)

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

by(*simp add*: *gen-refine.mem$_A$-of-def*)

lemma [*simp*]:
preserves-modes-mem $\mathcal{R} \Longrightarrow$
gen-refine.closed-others $\mathcal{R}$ = *closed-others* $\mathcal{R}$
unfolding *closed-others-def*
gen-refine.closed-others-def
preserves-modes-mem-def
by *auto*

lemma [*simp*]:
gen-refine.preserves-modes-mem $\mathcal{R}$ = *preserves-modes-mem* $\mathcal{R}$
unfolding *gen-refine.preserves-modes-mem-def2 preserves-modes-mem-def*
by *simp*

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 $\mathcal{R}$ ∧
preserves-modes-mem $\mathcal{R}$ ∧
conc.closed-glob-consistent P ∧
(∀ c_{1A} *mds* mem_1 c_{1C}.
(⟨ c_{1A}, *mds*, mem_1 ⟩$_A$, ⟨ c_{1C}, *mds*, mem_1 ⟩$_C$) ∈ $\mathcal{R}$ ⟶
(∀ c_{1C}' *mds'* mem_1'. ⟨ c_{1C}, *mds*, mem_1 ⟩$_C$ ⇝$_C$ ⟨ c_{1C}', *mds'*, mem_1' ⟩$_C$ ⟶
(∃ n c_{1A}'. *abs.neval* ⟨ c_{1A}, *mds*, mem_1 ⟩$_A$ n ⟨ c_{1A}', *mds'*, mem_1' ⟩$_A$ ∧
(⟨ c_{1A}', *mds'*, mem_1' ⟩$_A$, ⟨ c_{1C}', *mds'*, mem_1' ⟩$_C$) ∈ $\mathcal{R}$ ∧
(∀ c_{2A} mem_2 c_{2C} c_{2A}' mem_2'.
(⟨ c_{1A}, *mds*, mem_1 ⟩$_A$, ⟨ c_{2A}, *mds*, mem_2 ⟩$_A$) ∈ $\mathcal{R}_A$ ∧
(⟨ c_{2A}, *mds*, mem_2 ⟩$_A$, ⟨ c_{2C}, *mds*, mem_2 ⟩$_C$) ∈ $\mathcal{R}$ ∧
(⟨c_{1C}, *mds*, mem_1⟩$_C$, ⟨c_{2C}, *mds*, mem_2⟩$_C$) ∈ P ∧
abs.neval ⟨ c_{2A}, *mds*, mem_2 ⟩$_A$ n ⟨ c_{2A}', *mds'*, mem_2' ⟩$_A$ ⟶
(∃ c_{2C}' . ⟨ c_{2C}, *mds*, mem_2 ⟩$_C$ ⇝$_C$ ⟨ c_{2C}', *mds'*, mem_2' ⟩$_C$ ∧
(⟨ c_{2A}', *mds'*, mem_2' ⟩$_A$, ⟨ c_{2C}', *mds'*, mem_2' ⟩$_C$) ∈ $\mathcal{R}$ ∧
(⟨c_{1C}', *mds'*, mem_1'⟩$_C$, ⟨c_{2C}', *mds'*, mem_2'⟩$_C$) ∈ P))))))

lemma *preserves-modes-memD*:
preserves-modes-mem $\mathcal{R}$ ⟹
(⟨ c_A, mds_A, mem_A ⟩$_A$, ⟨ c_C, mds_C, mem_C ⟩$_C$) ∈ $\mathcal{R}$ ⟹
$mem_A = mem_C \wedge mds_A = mds_C$
by(*auto simp*: *preserves-modes-mem-def*)

lemma [*simp*]:
gen-refine.secure-refinement $\mathcal{R}_A$ $\mathcal{R}$ P = *secure-refinement* $\mathcal{R}_A$ $\mathcal{R}$ P
unfolding *gen-refine.secure-refinement-def secure-refinement-def*
apply *safe*
apply *fastforce*

apply *fastforce*
defer
apply *fastforce*
apply *fastforce*
apply *fastforce*
defer
apply ((*drule spec*)+, *erule* (*1*) *impE*)
apply ((*drule spec*)+, *erule* (*1*) *impE*)
apply (*clarify*)
apply(*rename-tac n* $c_{1A}{}'$ $mds_A{}'$ $mem_{1A}{}'$)
apply(*rule-tac x*=*n* **in** *exI*)
apply(*rule-tac x*=$c_{1A}{}'$ **in** *exI*)
apply(*fastforce dest*: *preserves-modes-memD*)
apply (*frule* (*1*) *preserves-modes-memD*)
apply *clarify*
apply ((*drule spec*)+, *erule* (*1*) *impE*)
apply ((*drule spec*)+, *erule* (*1*) *impE*)
apply *clarify*
apply(*blast dest*: *preserves-modes-memD*)
done

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*)
using *abs rr gen-refine.*R_C*-of-strong-low-bisim-mm*[*OF - - P-sym*]
by *simp*

end

context *sifum-refinement* **begin**
lemma *use-secure-refinement-helper*:
secure-refinement $\mathcal{R}_A$ $\mathcal{R}$ *P* $\Longrightarrow$
($(cm_A,mem_A),(cm_C,mem_C)) \in \mathcal{R} \Longrightarrow (cm_C,mem_C) \rightsquigarrow_C (cm_C{}',mem_C{}') \Longrightarrow$
$(\exists\, cm_A{}'\ mem_A{}'\ n.\ abs.neval\ (cm_A,mem_A)\ n\ (cm_A{}',mem_A{}') \wedge$
$((cm_A{}',mem_A{}'), (cm_C{}',mem_C{}')) \in \mathcal{R})$
apply(*case-tac* cm_A, *case-tac* cm_C)
apply *clarsimp*
apply(*clarsimp simp*: *secure-refinement-def*)
by (*metis surjective-pairing*)

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 $mds_C\ x) \Longrightarrow$
$(\langle c_A,\ mds_A\text{-}of\ mds_C,\ mem_A\text{-}of\ mem_C{}'\rangle_A, \langle c_C,\ mds_C,\ mem_C{}'\rangle_C) \in \mathcal{R}$
unfolding *closed-others-def*

by *auto*
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$_A$*) *componentwise-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* ≠ *mem' x* ∨ *dma mem x* ≠ *dma mem' x*
shows ∃ *k cmn'' mem'' cmn''' mem'''*. *k* < *m* ∧ *neval* (*cms* ! *n*,*mem*) *k* (*cmn''*,*mem''*)
∧
(*cmn''*,*mem''*) ⇝ (*cmn'''*, *mem'''*) ∧
(*mem'' x* ≠ *mem''' x* ∨ *dma mem'' x* ≠ *dma mem''' x*)
using *nevaln lcn-def lcn'-def modified len*
proof(*induct arbitrary*: *cms cmn' mem mem' rule*: *neval.induct*)
case (*neval-0 lcn lcn'*)
from *neval-0* **show** *?case* **by** *simp*
next
case (*neval-S-n lcn lcn'' m lcn'*)
obtain *cmn'' mem''* **where** *lcn''-def*: *lcn''* = (*cmn''*, *mem''*) **by** *fastforce*
show *?case*
proof(*cases mem x* ≠ *mem'' x* ∨ *dma mem x* ≠ *dma mem'' x*)
assume *a*: *mem x* ≠ *mem'' x* ∨ *dma mem x* ≠ *dma mem'' x*
let *?k* = *0*::*nat*
let *?cmn''* = *cms* ! *n*
let *?mem''* = *mem*
have *?k* < *Suc m* ∧
neval (*cms* ! *n*, *mem*) *?k* (*?cmn''*, *?mem''*) ∧
(*?cmn''*, *?mem''*) ⇝ (*cmn''*, *mem''*) ∧ (*?mem'' x* ≠ *mem'' x* ∨
dma ?mem'' x ≠ *dma mem'' x*)
apply (*rule conjI*, *simp add*: *neval.neval-0*)+
apply (*simp only*: *a*)
by (*simp add*: *neval-S-n*(*1*)[*simplified neval-S-n lcn''-def*])
thus *?case* **by** *blast*
next
assume *a*: ¬ (*mem x* ≠ *mem'' x* ∨ *dma mem x* ≠ *dma mem'' x*)
hence *unchanged*: *mem'' x* = *mem x* ∧ *dma mem'' x* = *dma mem x*
by (*blast intro*: *sym*)
define *cms''* **where** *cms''* = *cms*[*n* := *cmn'*]
have *len''*: *n* < *length cms''*
by(*simp add*: *cms''-def neval-S-n*)
hence *lcn''-def2*: *lcn''* = (*cms''* ! *n*, *mem''*)
by(*simp add*: *lcn''-def cms''-def*)
from
neval-S-n(*3*)[*OF lcn''-def2 neval-S-n*(*5*), *simplified unchanged neval-S-n len''*]
obtain *k cmn'''' mem'''' cmn''''' mem''''* **where**
hyp: *k* < *m* ∧
neval (*cms''* ! *n*, *mem''*) *k* (*cmn''''*, *mem''''*) ∧
(*cmn''''*, *mem''''*) ⇝ (*cmn'''''*, *mem'''''*) ∧
(*mem'''' x* ≠ *mem''''' x* ∨ *dma mem'''' x* ≠ *dma mem''''' x*)

```
    by blast
  have neval (cms ! n, mem) (Suc k) (cmn''', mem''')
    apply(rule neval.neval-S-n)
     prefer 2
     using hyp apply fastforce
    apply(simp add: cms''-def neval-S-n)
    by(rule neval-S-n(1)[simplified neval-S-n lcn''-def])
  moreover have Suc k < Suc m using hyp by auto
  ultimately show ?case using hyp by fastforce
 qed
qed

lemma neval-sched-Nil [simp]:
 (cms, mem) →[] (cms, mem)
 by simp

lemma reachable-mode-states-refl:
 map snd cms ∈ reachable-mode-states (cms, mem)
 apply(clarsimp simp: reachable-mode-states-def)
 using neval-sched-Nil by blast

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')]) ∈ reachable-mode-states (cms, mem)
using neval lc-def len proof(induct arbitrary: cms mem rule: neval.induct)
case (neval-0 x y)
 thus ?case
 apply simp
 apply(drule sym, simp add: len reachable-mode-states-refl)
 done
next
case (neval-S-n x y n z)
 define cms' where cms' = cms[k := fst y]
 define mem' where mem' = snd y
 have y-def: y = (cms' ! k, mem')
  by(simp add: cms'-def mem'-def neval-S-n)
 moreover have len': k < length cms'
  by(simp add: cms'-def neval-S-n)
 ultimately have hyp: map snd (cms'[k := fst z]) ∈ reachable-mode-states (cms',
mem')
  using neval-S-n by metis
 have map snd (cms'[k := fst z]) = map snd (cms[k := fst z])
  unfolding cms'-def
  by simp
 moreover have (cms, mem) ⇝k (cms', mem')
  using meval-intro neval-S-n y-def cms'-def mem'-def len' by fastforce
 ultimately show ?case
```

using *reachable-modes-subset subsetD hyp* **by** *fastforce*
qed

lemma *meval-sched-sound-mode-use*:
sound-mode-use gc $\Longrightarrow$ *meval-sched sched gc gc'* $\Longrightarrow$ *sound-mode-use gc'*
proof(*induct rule*: *meval-sched.induct*)
case (*1 gc*)
thus *?case* **by** *simp*
next
case (*2 n ns gc gc'*)
from *2*(*3*) **obtain** *gc''* **where** *meval-abv gc n gc''* **and** *a*: *meval-sched ns gc'' gc'* **by** *force*
with *2*(*2*) *sound-modes-invariant* **have** *b*: *sound-mode-use gc''* **by** (*metis surjective-pairing*)
show *?case* **by** (*rule 2*(*1*)[*OF b a*])
qed

lemma *neval-meval*:
neval lcn k lcn' $\Longrightarrow$ *n* $<$ *length cms* $\Longrightarrow$ *lcn* = (*cms* ! *n*,*mem*) $\Longrightarrow$ *lcn'* = (*cmn'*, *mem'*) $\Longrightarrow$
meval-sched (*replicate k n*) (*cms*,*mem*) (*cms*[*n* := *cmn'*],*mem'*)
proof(*induct arbitrary*: *cms mem cmn' mem' rule*: *neval.induct*)
case (*neval-0 lcn lcn'*)
thus *?case* **by** *fastforce*
next
case (*neval-S-n lcn lcn'' k lcn'*)
define *cms''* **where** [*simp*]: *cms''* = *cms*[*n* := *fst lcn''*]
define *mem''* **where** [*simp*]: *mem''* = *snd lcn''*
have *len''* [*simp*]: *n* $<$ *length cms''* **by**(*simp add*: *neval-S-n*(*4*))
have *lcn''-def*: *lcn''* = (*cms''* ! *n*, *mem''*) **using** *len''* **by** *simp*
have *hyp*: (*cms''*, *mem''*) $\rightarrow_{replicate\ k\ n}$ (*cms''*[*n* := *cmn'*], *mem'*)
by (*rule neval-S-n*(*3*)[*OF len'' lcn''-def neval-S-n*(*6*)])
have *meval*: (*cms*, *mem*) $\rightsquigarrow_n$ (*cms''*, *mem''*)
using *cms''-def neval-S-n.hyps*(*1*) *neval-S-n.prems*(*1*) *neval-S-n.prems*(*2*) **by** *fastforce*
from *hyp meval* **show** *?case*
by *fastforce*
qed

lemma *meval-sched-app*:
meval-sched as gc gc' $\Longrightarrow$ *meval-sched bs gc' gc''* $\Longrightarrow$ *meval-sched* (*as*@*bs*) *gc gc''*
proof(*induct as arbitrary*: *gc gc' bs*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons a as*)
from *Cons*(*2*)
obtain *gc'''* **where** *a*: *meval-abv gc a gc'''* **and** *as*: *meval-sched as gc''' gc'* **by** *force*

from *Cons(1)*[*OF as Cons(3)*] *a*
have $gc \rightarrow_a \#\ (as\ @\ bs)\ gc''$
by (*metis meval-sched.simps*)
thus *?case* **by** *simp*
qed

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 −
from *sound-mode-use*$_A$ **have** *abs.respects-own-guarantees* $(cms_A\ !\ n)$
using *nlen len-eq abs.locally-sound-respects-guarantees*
unfolding *abs.sound-mode-use-def list-all-length*
by *fastforce*
with *in-*$\mathcal{R}n$ **have** *1*: *conc.respects-own-guarantees* $(cms_C\ !\ n)$
using *preserves*
unfolding *preserves-local-guarantee-compliance-def*
by *metis*
with *eval*$_C$ *modified*$_C$ **have** *2*: ¬ *conc.doesnt-modify* $(fst\ (cms_C\ !\ n))\ x_C$
unfolding *conc.doesnt-modify-def*
by (*metis surjective-pairing*)
then have ¬ *conc.doesnt-read-or-modify* $(fst\ (cms_C\ !\ n))\ x_C$
using *conc.doesnt-read-or-modify-doesnt-modify* **by** *metis*
with *1 2* **show** *?thesis*
unfolding *conc.respects-own-guarantees-def*
by *metis*
qed

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(*cases* $x_C \in range\ var_C\text{-}of$)
assume $x_C \in range\ var_C\text{-}of$
from *this* **obtain** x_A **where** $x_C\text{-}def$: $x_C = var_C\text{-}of\ x_A$ **by** *blast*
obtain $c_A n\ mds_A n$ **where** [*simp*]: $cms_A\ !\ n = (c_A n, mds_A n)$ **by** *fastforce*
obtain $c_C n\ mds_C n$ **where** [*simp*]: $cms_C\ !\ n = (c_C n, mds_C n)$ **by** *fastforce*
obtain $c_C n'\ mds_C n'$ **where** [*simp*]: $cm_C n' = (c_C n', mds_C n')$ **by** *fastforce*
obtain $c_A n'\ mds_A n'$ **where** [*simp*]: $cm_A n' = (c_A n', mds_A n')$ **by** *fastforce*

from *in-*$\mathcal{R}n$ *pmmn* **have** [*simp*]: $mem_A = mem_A\text{-}of\ mem_C$ **and** [*simp*]: $mds_A n = mds_A\text{-}of\ mds_C n$
using *preserves-modes-memD* **by** *auto*
from *in-*$\mathcal{R}n'$ *pmmn* **have** [*simp*]: $mem_A' = mem_A\text{-}of\ mem_C'$ **and** [*simp*]: $mds_A n' = mds_A\text{-}of\ mds_C n'$
using *preserves-modes-memD* **by** *auto*

from $modified_C$ *dma-consistent* **have**
$modified_A$: $mem_A\ x_A \neq mem_A'\ x_A \vee dma_A\ mem_A\ x_A \neq dma_A\ mem_A'\ x_A$
by (*simp add*: $mem_A\text{-}of\text{-}def\ x_C\text{-}def$)

from len_A *nlen* **have** $nlen_A$: $n < length\ cms_A$ **by** *simp*
from len_A *ilen* **have** $ilen_A$: $i < length\ cms_A$ **by** *simp*

from *abs.neval-modifies-helper*[*OF* $neval_A n$ *HOL.refl HOL.refl* $nlen_A$ $modified_A$]
obtain $k_A\ cm_A n''\ mem_A''\ cm_A n'''\ mem_A'''$
where $k_A < m_A$
and $neval_A n''$: $abs.neval\ (cms_A\ !\ n, mem_A)\ k_A\ (cm_A n'', mem_A'')$
and $eval_A n''$: $(cm_A n'', mem_A'') \rightsquigarrow_A (cm_A n''', mem_A''')$
and $modified_A''$: $(mem_A''\ x_A \neq mem_A'''\ x_A \vee dma_A\ mem_A''\ x_A \neq dma_A\ mem_A'''\ x_A)$ **by** *blast*
let $?c_A n'' = fst\ cm_A n''$

let *?mds$_A$n''* = *snd cm$_A$n''*
from *eval$_A$n'' modified$_A$''* **have** *modifies$_A$''*: ¬ *abs.doesnt-modify ?c$_A$n'' x$_A$*
unfolding *abs.doesnt-modify-def*
by (*metis surjective-pairing*)
have *loc-reach$_A$''*: (*cm$_A$n''*, *mem$_A$''*) ∈ *abs.loc-reach* (*cms$_A$* ! *n*, *mem$_A$*)
apply(*rule abs.neval-loc-reach*)
apply(*rule neval$_A$n''*)
using *abs.loc-reach.refl* **by** *simp*
have *locally-sound-mode-use$_A$n*: *abs.locally-sound-mode-use* (*cms$_A$* ! *n*, *mem$_A$*)
using *sound-mode-use$_A$ nlen$_A$*
unfolding *abs.sound-mode-use-def*
using *list-all-length* **by** *fastforce*
from *modifies$_A$'' loc-reach$_A$'' locally-sound-mode-use$_A$n abs.doesnt-read-or-modify-doesnt-modify*
have *no-guar$_A$n*: *x$_A$* ∉ *?mds$_A$n'' GuarNoReadOrWrite* ∧ *x$_A$* ∉ *?mds$_A$n'' GuarNoWrite*
unfolding *abs.locally-sound-mode-use-def*
by (*metis surjective-pairing*)
let *?mdss$_A$'' = map snd* (*cms$_A$*[*n* := *fst* (*cm$_A$n''*,*mem$_A$''*)])
have *?mdss$_A$''* ∈ *abs.reachable-mode-states* (*cms$_A$*, *mem$_A$*)
apply(*rule abs.neval-reachable-mode-states*)
apply(*rule neval$_A$n''*)
apply(*rule HOL.refl*)
by(*rule nlen$_A$*)
hence *compat*: *abs.compatible-modes ?mdss$_A$''*
using *sound-mode-use$_A$*
by(*simp add*: *abs.globally-sound-mode-use-def*)
have *n*: *?mdss$_A$''* ! *n* = *?mds$_A$n''*
by(*simp add*: *nlen$_A$*)
let *?mds$_A$i* = *snd* (*cms$_A$* ! *i*)
have *i*: *?mdss$_A$''* ! *i* = *?mds$_A$i*
apply(*simp add*: *ilen$_A$*)
by(*metis nth-list-update-neq neq*)
from *nlen$_A$* **have** *nlen$_A$''*: *n* < *length ?mdss$_A$''* **by** *simp*
from *ilen$_A$* **have** *ilen$_A$''*: *i* < *length ?mdss$_A$''* **by** *simp*
with *compat n i nlen$_A$'' ilen$_A$'' no-guar$_A$n neq*
have *no-asm$_A$i*: *x$_A$* ∉ *?mds$_A$i AsmNoWrite* ∧ *x$_A$* ∉ *?mds$_A$i AsmNoReadOrWrite*
unfolding *abs.compatible-modes-def*
by *metis*

obtain *c$_A$i mds$_A$i* **where** [*simp*]: *cms$_A$* ! *i* = (*c$_A$i*, *mds$_A$i*) **by** *fastforce*
obtain *c$_C$i mds$_C$i* **where** [*simp*]: *cms$_C$* ! *i* = (*c$_C$i*, *mds$_C$i*) **by** *fastforce*

from *in-$\mathcal{R}$i pmmi* **have** [*simp*]: *mds$_A$i* = *mds$_A$-of mds$_C$i*
using *preserves-modes-memD* **by** *auto*
have [*simp*]: *?mds$_A$i* = *mds$_A$i* **by** *simp*
from *no-asm$_A$i* **have** *no-asm$_C$i*: *x$_C$* ∉ *mds$_C$i AsmNoWrite* ∧ *x$_C$* ∉ *mds$_C$i AsmNoReadOrWrite*
using *x$_C$-def mds$_A$-of-def*
using *doesnt-have-mode* **by** *auto*
thus *?thesis*

unfolding *var-asm-not-written-def*
by *simp*
next
let *?mds$_C$ n* = *snd* (*cms$_C$* ! *n*)
let *?mds$_C$ i* = *snd* (*cms$_C$* ! *i*)

assume *new-var*: $x_C \notin$ *range var$_C$-of*
from *conc-respects-priv*[*OF new-var modified$_C$ eval$_C$ n in-$\mathcal{R}$ n preserves sound-mode-use$_A$ nlen len$_A$ len$_C$*]
have $x_C \notin$ *?mds$_C$ n GuarNoWrite* $\wedge$ $x_C \notin$ *?mds$_C$ n GuarNoReadOrWrite* **.**
with *priv-is-guar-priv nlen ilen neq*
have $x_C \notin$ *priv-mem$_C$* ! *i*
by *blast*
with *new-var new-asms-only-for-priv ilen*
have $x_C \notin$ *?mds$_C$ i AsmNoReadOrWrite* $\cup$ *?mds$_C$ i AsmNoWrite*
by *blast*
thus *?thesis*
unfolding *var-asm-not-written-def*
by *simp*
qed

definition
priv-is-asm-priv :: *'Var$_C$ Mds list* $\Rightarrow$ *bool*
where
priv-is-asm-priv mdss$_C$ $\equiv$ $\forall\, i <$ *length cms. priv-mem$_C$* ! *i* $\subseteq$ (*mdss$_C$* ! *i*) *AsmNoReadOrWrite*

definition
priv-is-guar-priv :: *'Var$_C$ Mds list* $\Rightarrow$ *bool*
where
priv-is-guar-priv mdss$_C$ $\equiv$
$\forall\, i <$ *length cms.* ($\forall\, j <$ *length cms.* $i \neq j \longrightarrow$ *priv-mem$_C$* ! *i* $\subseteq$ (*mdss$_C$* ! *j*) *GuarNoReadOrWrite*)

definition
new-asms-only-for-priv :: *'Var$_C$ Mds list* $\Rightarrow$ *bool*
where
new-asms-only-for-priv mdss$_C$ $\equiv$
$\forall\ i <$ *length cms.*
((*mdss$_C$* ! *i*) *AsmNoReadOrWrite* $\cup$ (*mdss$_C$* ! *i*) *AsmNoWrite*) $\cap$ ($-$ *range var$_C$-of*) $\subseteq$ *priv-mem$_C$* ! *i*

definition
new-asms-NoReadOrWrite-only :: *'Var$_C$ Mds list* $\Rightarrow$ *bool*
where
new-asms-NoReadOrWrite-only mdss$_C$ $\equiv$
$\forall\ i <$ *length cms.*
(*mdss$_C$* ! *i*) *AsmNoWrite* $\cap$ ($-$ *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*
unfolding *ignores-old-vars-def*
apply(*simp add*: *Rdef*)
apply(*intro impI allI*)
apply(*rule conj-cong*)
apply(*erule* (*1*) *iP*[*unfolded ignores-old-vars-def*, *rule-format*])
apply(*erule* (*1*) *iQ*[*unfolded ignores-old-vars-def*, *rule-format*])
done

lemma *nth-map-eq'*:
length xs = *length ys* ⟹ *map f xs* = *map g ys* ⟹ *i* < *length xs* ⟹ *f* (*xs* ! *i*) = *g* (*ys* ! *i*)
apply(*induct xs ys rule*: *list-induct2*)
apply *simp*
apply(*case-tac i*)
apply *force*
by (*metis length-map nth-map*)

lemma *nth-map-eq*:
map f xs = *map g ys* ⟹ *i* < *length xs* ⟹ *f* (*xs* ! *i*) = *g* (*ys* ! *i*)
apply(*rule nth-map-eq'*)
apply(*erule map-eq-imp-length-eq*)
apply *assumption*+
done

lemma *nth-in-Union-over-set*:
i < *length xs* ⟹ *xs* ! *i* ⊆ ⋃(*set xs*)

by (*simp add*: *Union-upper*)

lemma *priv-are-new-vars*:
x ∈ *priv-mem*$_C$! *i* ⟹ *i* < *length cms* ⟹ *x* ∉ *range var*$_C$*-of*
using *new-vars-priv nth-in-Union-over-set subsetD*
using *priv-mem*$_C$*-def* **by** *fastforce*

lemma *priv-is-asm-priv-ignores-old-vars*:
ignores-old-vars priv-is-asm-priv
apply(*clarsimp simp*: *ignores-old-vars-def priv-is-asm-priv-def*)
apply(*rule all-cong*)
apply(*drule nth-map-eq*)
apply *simp*
apply(*blast dest*: *priv-are-new-vars fun-cong*)
done

lemma *priv-is-guar-priv-ignores-old-vars*:
ignores-old-vars priv-is-guar-priv
apply(*clarsimp simp*: *ignores-old-vars-def priv-is-guar-priv-def*)
apply(*rule all-cong*)
apply(*rule all-cong*)
apply(*rule imp-cong*)
apply(*rule HOL.refl*)
apply(*frule nth-map-eq*)
apply *simp*
apply(*drule-tac i=j* **in** *nth-map-eq*)
apply *simp*
apply(*blast dest*: *priv-are-new-vars fun-cong*)
done

lemma *new-asms-only-for-priv-ignores-old-vars*:
ignores-old-vars new-asms-only-for-priv
apply(*clarsimp simp*: *ignores-old-vars-def new-asms-only-for-priv-def*)
apply(*rule all-cong*)
apply(*drule nth-map-eq*)
apply *simp*
apply(*blast dest*: *priv-are-new-vars fun-cong*)
done

lemma *new-asms-NoReadOrWrite-only-ignores-old-vars*:
ignores-old-vars new-asms-NoReadOrWrite-only
apply(*clarsimp simp*: *ignores-old-vars-def new-asms-NoReadOrWrite-only-def*)
apply(*rule all-cong*)
apply(*drule nth-map-eq*)
apply *simp*
apply(*blast dest*: *priv-are-new-vars fun-cong*)
done

lemma *modes-respect-priv-ignores-old-vars*:

ignores-old-vars modes-respect-priv
apply(*rule ignores-old-vars-conj*)
apply(*subst modes-respect-priv-def*)
apply(*rule HOL.refl*)
apply(*rule priv-is-asm-priv-ignores-old-vars*)
apply(*rule ignores-old-vars-conj*)
apply(*rule HOL.refl*)
apply(*rule priv-is-guar-priv-ignores-old-vars*)
apply(*rule ignores-old-vars-conj*)
apply(*rule HOL.refl*)
apply(*rule new-asms-only-for-priv-ignores-old-vars*)
apply(*rule new-asms-NoReadOrWrite-only-ignores-old-vars*)
done

lemma *ignores-old-varsD*:
ignores-old-vars P $\Longrightarrow$ *length mdss* = *length mdss'* $\Longrightarrow$ *length mdss'* = *length cms* $\Longrightarrow$
(*map* (λ*x m. x m* $\cap$ (− *range var*$_C$*-of*)) *mdss*) = (*map* (λ*x m. x m* $\cap$ (− *range var*$_C$*-of*)) *mdss'*) $\Longrightarrow$
P mdss = *P mdss'*
unfolding *ignores-old-vars-def*
by *force*

lemma *new-privs-preserved'*:
⟨*c*, *mds*, *mem*⟩$_C$ $\rightsquigarrow_C$ ⟨*c'*, *mds'*, *mem'*⟩$_C$ $\Longrightarrow$ (*mds m* $\cap$ (− *range var*$_C$*-of*)) = (*mds' m* $\cap$ (− *range var*$_C$*-of*))
using *new-privs-preserved* **by** *blast*

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*
apply(*induct xs ys rule*: *list-induct2*)
apply *simp*
apply *force*
done

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 −
obtain *cms*$_C$ *mem*$_C$ **where** [*simp*]: *gc*$_C$ = (*cms*$_C$, *mem*$_C$) **by** *fastforce*
obtain *cms*$_C$*'* *mem*$_C$*'* **where** [*simp*]: *gc*$_C$*'* = (*cms*$_C$*'*, *mem*$_C$*'*) **by** *fastforce*
from *meval* **obtain** *cmn' mem*$_C$*'* **where**
eval$_C$*n*: (*cms*$_C$! *n*, *mem*$_C$) $\rightsquigarrow_C$ (*cmn'*, *mem*$_C$*'*) **and** *len*: *n* < *length cms*$_C$ **and** *cms*$_C$*'-def*: *cms*$_C$*'* = *cms*$_C$[*n* := *cmn'*]
using *conc.meval.cases* **by** *fastforce*
have

```
    P (map snd cms_C) = P (map snd cms_C')
    apply(rule ignores-old-varsD[OF ignores])
     apply(simp add: cms_C'-def)
     using len-eq apply (simp add: cms_C'-def)
    apply(rule map-nth-eq)
     apply (simp add: cms_C'-def)
    apply(case-tac i = n)
     apply simp
     apply(rule ext)
     apply(simp add: cms_C'-def)
     using eval_C n new-privs-preserved' apply(metis surjective-pairing)
    by (simp add: cms_C'-def)
  thus ?thesis by simp
qed

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'))
using meval-sched len-eq proof(induct rule: conc.meval-sched.induct)
  case (1 gc gc')
  thus ?case by simp
next
  case (2 n ns gc gc')
 from 2(2) obtain gc'' where b: conc.meval-abv gc n gc'' and a: conc.meval-sched
ns gc'' gc' by force
  with 2 have length (fst gc'') = length cms
    using conc.meval.cases
    by (metis length-list-update surjective-pairing)
  with 2 a b show ?case
  using ignores-old-vars-conc-meval ignores by metis
qed

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'))
 by(blast dest!: ignores-old-vars-conc-meval-sched[OF modes-respect-priv-ignores-old-vars])

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'))
 by(blast dest!: ignores-old-vars-conc-meval[OF modes-respect-priv-ignores-old-vars])


lemma traces-refinement:
  ⋀gc_C gc_C' sched_C gc_A. conc.meval-sched sched_C gc_C gc_C' ⟹
```

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 −
fix gc_C gc_C' $sched_C$ gc_A
assume $meval_C$: *conc.meval-sched* $sched_C$ gc_C gc_C'
and *len-eq* [*simp*]: *length* (*fst* gc_A) = *length cms*
and *len-eq'*[*simp*]: *length* (*fst* gc_C) = *length cms*
and *in-*$\mathcal{R}$: ($\bigwedge i$. i < *length cms* $\Longrightarrow$ ((*fst* gc_A ! i, *snd* gc_A), (*fst* gc_C ! i, *snd* gc_C)) $\in$ $\mathcal{R}$-*rel* (*cms* ! i))
and *sound-mode-use*$_A$: *abs.sound-mode-use* gc_A
and *modes-respect-priv*: *modes-respect-priv* (*map snd* (*fst* gc_C))
thus
$\exists$ $sched_A$ 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(*induct arbitrary*: gc_A *rule*: *conc.meval-sched.induct*)
case (*1* cms_C cms_C')
from *1*(*1*) **have** cms_C'-*def* [*simp*]: cms_C' = cms_C **by** *simp*
with *1* **have** *abs.meval-sched* [] gc_A gc_A $\wedge$
($\forall i$<*length cms*.
((*fst* gc_A ! i, *snd* gc_A), *fst* cms_C' ! i, *snd* cms_C') $\in$ $\mathcal{R}$-*rel* (*cms* ! i)) $\wedge$
abs.sound-mode-use gc_A
by *simp*
thus *?case* **by** *blast*
next
case (*2 n ns* gc_C gc_c')
obtain cms_C mem_C **where** gc_C-*def* [*simp*]: gc_C = (cms_C, mem_C) **by** *force*
obtain cms_A mem_A **where** gc_A-*def* [*simp*]: gc_A = (cms_A, mem_A) **by** *force*
from *2*(*2*) gc_C-*def* **obtain** cms_C'' mem_C'' **where**
$meval_C$: ((cms_C,mem_C), n, (cms_C'',mem_C'')) $\in$ *conc.meval* **and**
meval-sched$_C$: *conc.meval-sched ns* (cms_C'',mem_C'') gc_c'
by *force*

let *?*$cm_C n$ = cms_C ! n
let *?*$cm_A n$ = cms_A ! n
let *?*$\mathcal{R} n$ = $\mathcal{R}$-*rel* (*cms* ! n)
from $meval_C$ **obtain** $cm_C n''$ **where**
$eval_C n$: (*?*$cm_C n$, mem_C) $\rightsquigarrow_C$ ($cm_C n''$, mem_C'') **and**
len: n < *length* cms_C **and**
cms_C''-*def*: cms_C'' = cms_C [n := $cm_C n''$] **by** (*blast elim*: *conc.meval.cases*)
from *len* **have** *len* [*simp*]: n < *length cms* **by** (*simp add*: *2*[*simplified*])
from cms_C''-*def 2* **have**

len-cms$_C$'' [*simp*]: *length cms$_C$'' = length cms* **by** *simp*
from *2 len* **have**
in-$\mathcal{R}n$: ((*?cm$_A$n*,*mem$_A$*), (*?cm$_C$n*,*mem$_C$*)) ∈ *?$\mathcal{R}n$*
by *simp*

with *eval$_C$n use-secure-refinement-helper*[*OF secure-refinements*[*OF len*]]
obtain *cm$_A$n'' mem$_A$'' m$_A$* **where**
neval$_A$n: *abs.neval* (*?cm$_A$n*, *mem$_A$*) *m$_A$* (*cm$_A$n''*, *mem$_A$''*) **and**
in-$\mathcal{R}n''$: ((*cm$_A$n''*,*mem$_A$''*),(*cm$_C$n''*,*mem$_C$''*)) ∈ *?$\mathcal{R}n$*
by *blast+*

define *cms$_A$''* **where** *cms$_A$'' = cms$_A$* [*n* := *cm$_A$n''*]
define *gc$_A$''* **where** [*simp*]: *gc$_A$'' = (cms$_A$'', mem$_A$'')*
have *len-cms$_A$''* [*simp*]: *length cms$_A$'' = length cms* **by**(*simp add*: *cms$_A$''-def 2*[*simplified*])

have *in-$\mathcal{R}''$*: (⋀*i*. *i* < *length cms* ⟹ ((*cms$_A$''* ! *i*, *mem$_A$''*), *cms$_C$''* ! *i*, *mem$_C$''*) ∈ *$\mathcal{R}$-rel* (*cms* ! *i*))
proof −
fix *i*
assume *i* < *length cms*
show *?thesis i*
proof(*cases i* = *n*)
assume *i* = *n*
hence *cms$_A$''* ! *i* = *cm$_A$n''*
using *cms$_A$''-def len-cms$_A$'' len* **by** *simp*
moreover from ‹*i* = *n*› **have** *cms$_C$''* ! *i* = *cm$_C$n''*
using *cms$_C$''-def len-cms$_C$'' len* **by** *simp*
ultimately show *?thesis*
using *in-$\mathcal{R}n''$* ‹*i* = *n*›
by *simp*
next
obtain *c$_A$i mds$_A$i* **where** *cms$_A$i-def* [*simp*]: (*cms$_A$* ! *i*) = (*c$_A$i*, *mds$_A$i*) **by** *fastforce*
obtain *c$_C$i mds$_C$i* **where** *cms$_C$i-def* [*simp*]: (*cms$_C$* ! *i*) = (*c$_C$i*, *mds$_C$i*) **by** *fastforce*
hence *mds$_C$i-def*: *mds$_C$i* = *snd* (*cms$_C$* ! *i*) **by** *simp*

from *2*(*5*) ‹*i* < *length cms*› **have**
in-$\mathcal{R}i$: ((*cms$_A$* ! *i*,*mem$_A$*), (*cms$_C$* ! *i*,*mem$_C$*)) ∈ *$\mathcal{R}$-rel* (*cms* ! *i*)
by *force*

from *in-$\mathcal{R}n''$ secure-refinements len preserves-modes-memD*
have *mem$_A$''-def* [*simp*]: *mem$_A$'' = mem$_A$-of mem$_C$''*
unfolding *secure-refinement-def*
by (*metis surjective-pairing*)

from *in-$\mathcal{R}i$ secure-refinements* ‹*i* < *length cms*› *preserves-modes-memD*
cms$_A$i-def cms$_C$i-def

```
      have mem_A-def [simp]: mem_A = mem_A-of mem_C and
          mds_A i-def [simp]: mds_A i = mds_A-of mds_C i
        unfolding secure-refinement-def
        by metis+

      assume i ≠ n
      hence cms_A'' ! i = cms_A ! i
        using cms_A''-def len-cms_A'' len by simp
      moreover from ‹i ≠ n› have cms_C'' ! i = cms_C ! i
        using cms_C''-def len-cms_C'' len by simp
      ultimately  show ?thesis

        using 2(5)[of i] ‹i ≠ n› ‹i < length cms›
        apply simp
        apply(rule closed-othersD)
              apply(rule secure-refinements[OF ‹i < length cms›, unfolded se-
cure-refinement-def, THEN conjunct1])
         apply assumption
        apply(simp only: mds_C i-def)
            apply(rule-tac Rn=R-rel (cms ! n)  and Ri=R-rel (cms ! i) in
modified-variables-are-not-assumed-not-written)
                        apply(rule 2(6)[unfolded gc_A-def])
                      using secure-refinements len secure-refinement-def apply blast
                       apply(rule in-Rn)
                      using secure-refinements secure-refinement-def apply blast
                     apply(rule in-Ri)
                    apply(rule len)
                    using 2 apply simp
                   using 2 apply simp
                   using 2(7) unfolding modes-respect-priv-def priv-is-asm-priv-def
gc_C-def
                   using 2.prems(3) apply auto[1]
                   using 2(7) unfolding modes-respect-priv-def priv-is-guar-priv-def
gc_C-def
                 using 2.prems(3) apply auto[1]
             using 2(7) unfolding modes-respect-priv-def new-asms-only-for-priv-def
gc_C-def
                  using 2.prems(3) apply auto[1]
                apply(rule eval_C n)
               apply(rule neval_A n)
              apply(rule in-Rn'')
             apply fastforce
            apply assumption
           apply assumption
          apply(rule local-guarantee-preservation)
          by simp
      qed
    qed
```

have *meval-sched*$_A$: *abs.meval-sched* (*replicate* m_A *n*) gc_A (cms_A'', mem_A'')
apply(*simp add*: cms_A''-*def*)
apply(*rule abs.neval-meval*[*OF - - HOL.refl HOL.refl*])
apply(*rule neval*$_A$*n*)
using *2.prems*(*2*) **by** *auto*

have *sound-mode-use*$_A''$: *abs.sound-mode-use* (cms_A'', mem_A'')
apply(*rule abs.meval-sched-sound-mode-use*)
apply(*rule 2*(*6*))
by(*rule meval-sched*$_A$)

have *respects''*: *modes-respect-priv* (*map snd* cms_C'')
apply(*rule meval-modes-respect-priv*[**where** gc_C'=(cms_C'',mem_C''), *simplified*])
apply(*rule meval*$_C$)
using *2.prems*(*3*) gc_C-*def* **apply** *blast*
using *2* **by** *simp*

from *respects''* *2*(*1*)[*OF meval-sched*$_C$, **where** $gc_A 7 = gc_A''$] *in-*$\mathcal{R}''$ *sound-mode-use*$_A''$
obtain *sched*$_A$ gc_A'
where *meval-sched*$_A''$: *abs.meval-sched sched*$_A$ gc_A'' gc_A' **and**
in-$\mathcal{R}'$: ($\forall$ *i*<*length cms.* ((*fst* gc_A' ! *i*, *snd* gc_A'), *fst* gc_c' ! *i*, *snd* gc_c') $\in$ $\mathcal{R}$-*rel* (*cms* ! *i*)) **and**
sound-mode-use$_A'$: *abs.sound-mode-use* gc_A' **by** *fastforce*
define *final-sched*$_A$ **where** *final-sched*$_A$ = (*replicate* m_A *n*) @ *sched*$_A$
have *meval-final-sched*$_A$: *abs.meval-sched final-sched*$_A$ gc_A gc_A'
using *meval-sched*$_A''$ *meval-sched*$_A$ *abs.meval-sched-app final-sched*$_A$-*def* gc_A''-*def* **by** *blast*

from *meval-final-sched*$_A$ *in-*$\mathcal{R}'$ *sound-mode-use*$_A'$
show *?case* **by** *blast*
qed
qed

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*
by(*auto simp*: *restrict-modes-def*)

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
unfolding compatible-modes-def
proof(safe)
  fix i x j
  assume ilen: i < length mdss
  assume jlen: j < length mdss
  assume neq: j ≠ i
  assume asm: x ∈ (mdss ! i) AsmNoReadOrWrite
  show x ∈ (mdss ! j) GuarNoReadOrWrite
  proof(cases x ∈ X)
    assume xin: x ∈ X
    let ?mdss_X = restrict-modes mdss X
    from asm xin have x ∈ (?mdss_X ! i) AsmNoReadOrWrite
      unfolding restrict-modes-def
      using ilen by auto

    with compat-X jlen ilen neq
    have x ∈ (?mdss_X ! j) GuarNoReadOrWrite
      unfolding compatible-modes-def
      by auto
    with xin jlen show ?thesis
      unfolding restrict-modes-def by auto
  next
    assume xnin: x ∉ X
    let ?mdss_X = restrict-modes mdss (− X)
    from asm xnin have x ∈ (?mdss_X ! i) AsmNoReadOrWrite
      unfolding restrict-modes-def
      using ilen by auto

    with compat-compX jlen ilen neq
    have x ∈ (?mdss_X ! j) GuarNoReadOrWrite
      unfolding compatible-modes-def
      by auto
    with xnin jlen show ?thesis
      unfolding restrict-modes-def by auto
  qed
next
  fix i x j
  assume ilen: i < length mdss
  assume jlen: j < length mdss
  assume neq: j ≠ i
  assume asm: x ∈ (mdss ! i) AsmNoWrite
  show x ∈ (mdss ! j) GuarNoWrite
  proof(cases x ∈ X)
    assume xin: x ∈ X
    let ?mdss_X = restrict-modes mdss X
    from asm xin have x ∈ (?mdss_X ! i) AsmNoWrite
      unfolding restrict-modes-def
```

```
      using ilen by auto

    with compat-X jlen ilen neq
    have x ∈ (?mdss_X ! j) GuarNoWrite
      unfolding compatible-modes-def
      by auto
    with xin jlen show ?thesis
      unfolding restrict-modes-def by auto
  next
    assume xnin: x ∉ X
    let ?mdss_X = restrict-modes mdss (− X)
    from asm xnin have x ∈ (?mdss_X ! i) AsmNoWrite
      unfolding restrict-modes-def
      using ilen by auto

    with compat-compX jlen ilen neq
    have x ∈ (?mdss_X ! j) GuarNoWrite
      unfolding compatible-modes-def
      by auto
    with xnin jlen show ?thesis
      unfolding restrict-modes-def by auto
  qed
qed

lemma in-restrict-modesD:
  i < length mdss ⟹ x ∈ ((restrict-modes mdss X) ! i) m ⟹ x ∈ X ∧ x ∈ (mdss
! i) m
  by(auto simp: restrict-modes-def)

lemma in-restrict-modesI:
  i < length mdss ⟹ x ∈ X ⟹ x ∈ (mdss ! i) m ⟹ x ∈ ((restrict-modes mdss
X) ! i) m
  by(auto simp: restrict-modes-def)

lemma meval-sched-length:
  meval-sched sched gc gc' ⟹ length (fst gc') = length (fst gc)
  apply(induct sched arbitrary: gc gc')
  by auto


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-R: (∀ i<length cms. ((cms_A ! i, mem_A), cms_C ! i, mem_C) ∈ R-rel
```

(*cms* ! *i*))
shows *conc.compatible-modes* (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*))
unfolding *conc.compatible-modes-def*
proof(*clarsimp*)
fix *i x*
assume *i-len*: *i* < *length cms*$_C$
let *?cms* = *cms* ! *i* **and**
?c$_A$ = *fst* (*cms*$_A$! *i*) **and** *?mds*$_A$ = *snd* (*cms*$_A$! *i*) **and**
?c$_C$ = *fst* (*cms*$_C$! *i*) **and** *?mds*$_C$ = *snd* (*cms*$_C$! *i*)

from *in-*$\mathcal{R}$ *i-len len*$_C$
have *in-*$\mathcal{R}$*-i*: ((*cms*$_A$! *i*, *mem*$_A$), *cms*$_C$! *i*, *mem*$_C$) ∈ $\mathcal{R}$*-rel ?cms* **by** *simp*

from *i-len* **have** *i* < *length* (*map snd cms*$_C$) **by** *simp*
hence *m-x-range*: ⋀*m*. *x* ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *i*) *m* ⟹ *x* ∈ *range var*$_C$*-of* ∧ *x* ∈ (*map snd cms*$_C$! *i*) *m*
using *conc.in-restrict-modesD i-len* **by** *blast+*
hence *m-x*$_C$*-i*: ⋀*m*. *x* ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *i*) *m* ⟹ *x* ∈ *?mds*$_C$ *m*
by (*simp add*: *i-len*)

from *secure-refinements i-len len*$_C$
have *secure-refinement* ($\mathcal{R}_A$*-rel ?cms*) ($\mathcal{R}$*-rel ?cms*) (*P-rel ?cms*) **by** *simp*
hence *preserves-modes-mem-*$\mathcal{R}$*-i*: *preserves-modes-mem* ($\mathcal{R}$*-rel ?cms*)
unfolding *secure-refinement-def* **by** *simp*

from *in-*$\mathcal{R}$*-i* **have** (⟨*?c*$_A$, *?mds*$_A$, *mem*$_A$⟩$_A$, ⟨*?c*$_C$, *?mds*$_C$, *mem*$_C$⟩$_C$) ∈ $\mathcal{R}$*-rel ?cms* **by** *clarsimp*
with *preserves-modes-mem-*$\mathcal{R}$*-i*
have (∀ *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*)
unfolding *preserves-modes-mem-def* **by** *blast*
with *m-x*$_C$*-i* **have** *m-x*$_A$: ⋀*m*. *x* ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *i*) *m* ⟹ *var*$_A$*-of x* ∈ *?mds*$_A$ *m*
unfolding *var*$_A$*-of-def* **using** *m-x-range inj-image-mem-iff var*$_C$*-of-inj* **by** *fastforce*

show (*x* ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *i*) *AsmNoReadOrWrite* ⟶
(∀ *j*<*length cms*$_C$. *j* ≠ *i* ⟶
x ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *j*) *GuarNoReadOrWrite*)) ∧
(*x* ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *i*) *AsmNoWrite* ⟶
(∀ *j*<*length cms*$_C$. *j* ≠ *i* ⟶
x ∈ (*conc.restrict-modes* (*map snd cms*$_C$) (*range var*$_C$*-of*) ! *j*) *GuarNoWrite*))
proof(*safe*)
fix *j*

assume *AsmNoRW-x_C*: $x \in$ (*conc.restrict-modes* (*map snd* cms_C) (*range* var_C*-of*) ! i) *AsmNoReadOrWrite* **and**
j-len: $j <$ *length* cms_C **and**
j-not-i: $j \neq i$
let *?cms'* = cms ! j **and**
$?c_A' = fst\ (cms_A\ !\ j)$ **and** $?mds_A' = snd\ (cms_A\ !\ j)$ **and**
$?c_C' = fst\ (cms_C\ !\ j)$ **and** $?mds_C' = snd\ (cms_C\ !\ j)$

from *AsmNoRW-x_C m-x-range*
have *x-range*: $x \in$ *range* var_C*-of* **by** *simp*

from *AsmNoRW-x_C m-x_A*
have var_A*-of* $x \in$ *?mds$_A$ AsmNoReadOrWrite* **by** *simp*
with *compatible-modes$_A$*
have *GuarNoRW-x_A*: var_A*-of* $x \in$ *?mds$_A$' GuarNoReadOrWrite*
unfolding *abs.compatible-modes-def* **using** *i-len* len_A len_C *j-len j-not-i* **by** *clarsimp*

from *in-$\mathcal{R}$ j-len* len_C
have *in-$\mathcal{R}$-j*: $((cms_A\ !\ j,\ mem_A),\ cms_C\ !\ j,\ mem_C) \in \mathcal{R}$*-rel ?cms'* **by** *simp*

from *j-len* **have** *j-len'*: $j <$ *length* (*map snd* cms_C) **by** *simp*

from *secure-refinements j-len* len_C
have *secure-refinement* ($\mathcal{R}_A$*-rel ?cms'*) ($\mathcal{R}$*-rel ?cms'*) (*P-rel ?cms'*) **by** *simp*
hence *preserves-modes-mem-$\mathcal{R}$-j*: *preserves-modes-mem* ($\mathcal{R}$*-rel ?cms'*)
unfolding *secure-refinement-def* **by** *simp*

from *in-$\mathcal{R}$-j* **have** $(\langle ?c_A', ?mds_A', mem_A\rangle_A, \langle ?c_C', ?mds_C', mem_C\rangle_C) \in \mathcal{R}$*-rel ?cms'* **by** *clarsimp*
with *preserves-modes-mem-$\mathcal{R}$-j*
have $(\forall x_A.\ mem_A\ x_A = mem_C\ (var_C\text{-}of\ x_A)) \wedge (\forall m.\ var_C\text{-}of\ `\ ?mds_A'\ m = range\ var_C\text{-}of \cap ?mds_C'\ m)$
unfolding *preserves-modes-mem-def* **by** *blast*

with *GuarNoRW-x_A j-len j-len' mds$_A$-of-def x-range conc.in-restrict-modesI* var_C*-of-inj*
show $x \in$ (*conc.restrict-modes* (*map snd* cms_C) (*range* var_C*-of*) ! j) *GuarNoReadOrWrite*
unfolding var_A*-of-def*
by (*metis* (*no-types*, *lifting*) *doesnt-have-mode f-inv-into-f image-inv-f-f nth-map*)
next

fix j
assume *AsmNoWrite-x_C*: $x \in$ (*conc.restrict-modes* (*map snd* cms_C) (*range* var_C*-of*) ! i) *AsmNoWrite* **and**
j-len: $j <$ *length* cms_C **and**
j-not-i: $j \neq i$

let *?cms′ = cms ! j* **and**
?c_A′ = fst (cms_A ! j) **and** *?mds_A′ = snd (cms_A ! j)* **and**
?c_C′ = fst (cms_C ! j) **and** *?mds_C′ = snd (cms_C ! j)*

from *AsmNoWrite-x_C m-x-range*
have *x-range*: *x ∈ range var_C-of* **by** *simp*

from *AsmNoWrite-x_C m-x_A*
have *var_A-of x ∈ ?mds_A AsmNoWrite* **by** *simp*
with *compatible-modes_A*
have *GuarNoWrite-x_A*: *var_A-of x ∈ ?mds_A′ GuarNoWrite*
unfolding *abs.compatible-modes-def* **using** *i-len len_A len_C j-len j-not-i* **by** *clarsimp*

from *in-R j-len len_C*
have *in-R-j*: *((cms_A ! j, mem_A), cms_C ! j, mem_C) ∈ R-rel ?cms′* **by** *simp*

from *j-len* **have** *j-len′*: *j < length (map snd cms_C)* **by** *simp*

from *secure-refinements j-len len_C*
have *secure-refinement (R_A-rel ?cms′) (R-rel ?cms′) (P-rel ?cms′)* **by** *simp*
hence *preserves-modes-mem-R-j*: *preserves-modes-mem (R-rel ?cms′)*
unfolding *secure-refinement-def* **by** *simp*

from *in-R-j* **have** *(⟨?c_A′, ?mds_A′, mem_A⟩_A, ⟨?c_C′, ?mds_C′, mem_C⟩_C) ∈ R-rel ?cms′* **by** *clarsimp*
with *preserves-modes-mem-R-j*
have *(∀ 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)*
unfolding *preserves-modes-mem-def* **by** *blast*

with *GuarNoWrite-x_A j-len j-len′ mds_A-of-def x-range conc.in-restrict-modesI var_C-of-inj*
show *x ∈ (conc.restrict-modes (map snd cms_C) (range var_C-of) ! j) GuarNoWrite*
unfolding *var_A-of-def*
by *(metis (no-types, lifting) doesnt-have-mode f-inv-into-f image-inv-f-f nth-map)*
qed
qed

lemma *compatible-modes-new-vars*:
length mdss = length cms ⟹ modes-respect-priv mdss ⟹ conc.compatible-modes (conc.restrict-modes mdss (− range var_C-of))
unfolding *conc.compatible-modes-def*
proof(*safe*)
let *?X = − range var_C-of*
let *?mdss_X = conc.restrict-modes mdss ?X*
assume *respect*: *modes-respect-priv mdss*
assume *len-eq*: *length mdss = length cms*

fix i x_C j
assume *ilen*: $i < length\ ?mdss_X$
assume *jlen*: $j < length\ ?mdss_X$
assume *neq*: $j \neq i$
assume asm_X: $x_C \in (?mdss_X\ !\ i)\ AsmNoWrite$
from *conc.in-restrict-modesD ilen* asm_X *conc.restrict-modes-length* **have**
xin: $x_C \in ?X$ **and**
asm: $x_C \in (mdss\ !\ i)\ AsmNoWrite$ **by** *metis+*
from *asm* **have** *False*
using *respect xin ilen conc.restrict-modes-length len-eq*
unfolding *modes-respect-priv-def new-asms-NoReadOrWrite-only-def*
by *force*
thus $x_C \in (?mdss_X\ !\ j)\ GuarNoWrite$ **by** *blast*
next
let $?X = -\ range\ var_C\text{-}of$
let $?mdss_X = conc.restrict\text{-}modes\ mdss\ ?X$
assume *respect*: *modes-respect-priv mdss*
assume *len-eq*: *length mdss* = *length cms*
fix i x_C j
assume *ilen*: $i < length\ ?mdss_X$
assume *jlen*: $j < length\ ?mdss_X$
assume *neq*: $j \neq i$
assume asm_X: $x_C \in (?mdss_X\ !\ i)\ AsmNoReadOrWrite$
from *conc.in-restrict-modesD ilen* asm_X *conc.restrict-modes-length* **have**
xin: $x_C \in ?X$ **and**
asm: $x_C \in (mdss\ !\ i)\ AsmNoReadOrWrite$ **by** *metis+*
from *respect asm xin ilen conc.restrict-modes-length len-eq* **have**
$x_C \in priv\text{-}mem_C\ !\ i$
unfolding *modes-respect-priv-def new-asms-only-for-priv-def*
by *force*
with *respect ilen jlen neq conc.restrict-modes-length len-eq* **have**
$x_C \in (mdss\ !\ j)\ GuarNoReadOrWrite$
unfolding *modes-respect-priv-def priv-is-guar-priv-def*
by *force*
with *jlen xin conc.in-restrict-modesI* **show**
$x_C \in (?mdss_X\ !\ j)\ GuarNoReadOrWrite$ **by** *force*
qed

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-mode-use $gc_A \Longrightarrow$ *modes-respect-priv* (*map snd* (*fst* gc_C)) $\Longrightarrow$
conc.sound-mode-use gc_C
proof −
fix gc_C gc_A
assume *len-eq* [*simp*]: *length* (*fst* gc_A) = *length cms*

and *len-eq'*[*simp*]: *length* (*fst* gc_C) = *length cms*
and *in-*$\mathcal{R}$: ($\bigwedge i$. i < *length cms* $\Longrightarrow$ ((*fst* gc_A ! i, *snd* gc_A), (*fst* gc_C ! i, *snd* gc_C)) $\in$ $\mathcal{R}$-*rel* (*cms* ! i))
and *sound-mode-use*$_A$: *abs.sound-mode-use* gc_A
and *modes-respect-priv*: *modes-respect-priv* (*map snd* (*fst* gc_C))
have *conc.globally-sound-mode-use* gc_C
unfolding *conc.globally-sound-mode-use-def*
proof(*clarsimp*)
fix $mdss_C'$
assume *in-reachable-modes*: $mdss_C' \in$ *conc.reachable-mode-states* gc_C
from *this* **obtain** cms_C' mem_C' $sched_C$ **where**
meval-sched$_C$: *conc.meval-sched* $sched_C$ gc_C (cms_C', mem_C') **and**
$mdss_C'$-*def*: $mdss_C' = map\ snd\ cms_C'$
unfolding *conc.reachable-mode-states-def* **by** *blast*
from *traces-refinement*[*OF meval-sched*$_C$, *OF len-eq len-eq' in-*$\mathcal{R}$ *sound-mode-use*$_A$ *modes-respect-priv*]
obtain $sched_A$ gc_A' cms_A' mem_A' **where** gc_A'-*def* [*simp*]: $gc_A' = (cms_A', mem_A')$ **and**
meval-sched$_A$: *abs.meval-sched* $sched_A$ gc_A gc_A' **and**
in-$\mathcal{R}$: ($\forall i$<*length cms*.
((cms_A' ! i, mem_A'), cms_C' ! i, mem_C') $\in$ $\mathcal{R}$-*rel* (*cms* ! i))
and *sound-mode-use*$_A$': *abs.sound-mode-use* gc_A'
by *fastforce*
let ?$mdss_A' = map\ snd\ cms_A'$
have ?$mdss_A' \in$ *abs.reachable-mode-states* gc_A
unfolding *abs.reachable-mode-states-def*
using *meval-sched*$_A$ **by** *fastforce*
hence *compatible-modes*$_A$': *abs.compatible-modes* ?$mdss_A'$
using *sound-mode-use*$_A$ **unfolding** *abs.sound-mode-use-def abs.globally-sound-mode-use-def*
by *fastforce*
let ?X = *range* var_C-*of*
show *conc.compatible-modes* $mdss_C'$
proof(*rule conc.compatible-modes-by-case-distinction*[**where** X=?X])
show *conc.compatible-modes* (*conc.restrict-modes* $mdss_C'$?X)
apply(*simp add*: $mdss_C'$-*def*)
apply(*rule compatible-modes-old-vars*[*OF - - - in-*$\mathcal{R}$])
apply(*rule compatible-modes*$_A$')
using *len-eq abs.meval-sched-length*[*OF meval-sched*$_A$] gc_A'-*def* **apply** *simp*
using *len-eq' conc.meval-sched-length*[*OF meval-sched*$_C$] **by** *simp*
next
show *conc.compatible-modes* (*conc.restrict-modes* $mdss_C'$ (− ?X))
apply(*rule compatible-modes-new-vars*)
using *len-eq' conc.meval-sched-length*[*OF meval-sched*$_C$] $mdss_C'$-*def* **apply** *simp*
apply(*simp add*: $mdss_C'$-*def*)
apply(*rule meval-sched-modes-respect-priv*[*OF meval-sched*$_C$, *simplified*])
using *modes-respect-priv* **by** *simp*
qed
qed

moreover have *list-all* (λ *cm. conc.locally-sound-mode-use* (*cm*, (*snd* gc_C))) (*fst* gc_C)
unfolding *list-all-length*
proof(*clarify*)
fix *i*
assume *i* < *length* (*fst* gc_C)
hence *len*: *i* < *length cms* **by** *simp*
have *preserves*: *preserves-locally-sound-mode-use* ($\mathcal{R}$*-rel* (*cms* ! *i*))
apply(*rule locally-sound-mode-use-preservation*)
using *secure-refinements len* **apply** *blast*
using *local-guarantee-preservation len* **by** *blast*
have *abs.locally-sound-mode-use* (*fst* gc_A ! *i*, *snd* gc_A)
using $sound\text{-}mode\text{-}use_A$ ‹*i* < *length cms*› *len-eq*
unfolding *abs.sound-mode-use-def list-all-length*
by (*simp add*: *case-prod-unfold*)

from *this in-*$\mathcal{R}$[*OF len*] *preserves*[*unfolded preserves-locally-sound-mode-use-def*]
show *conc.locally-sound-mode-use* (*fst* gc_C ! *i*, *snd* gc_C)
by *blast*
qed

ultimately show *?thesis* gc_C gc_A **unfolding** *conc.sound-mode-use-def*
by (*simp add*: *case-prod-unfold*)
qed

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 $sound\text{-}mode\text{-}use_A$: ($\bigwedge$ *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*$_C$
apply(*rule conc.sifum-compositionality-cont*)
apply(*clarsimp simp*: *list-all-length*)
apply(*clarsimp simp*: *conc.com-sifum-secure-def conc.low-indistinguishable-def*)
apply(*rule conc.mm-equiv.intros*)
apply(*rule* R_C*-of-strong-low-bisim-mm*)
apply(*fastforce intro*: *bisims*)
apply(*fastforce intro*: *secure-refinements*)
apply(*fastforce simp*: *Ps-sym*)
apply(*clarsimp simp*: R_C*-of-def*)
apply(*rename-tac i* c_C mds_C mem_C mem_C′)
apply(*rule-tac x*=*fst* (*cms*$_A$! *i*) **in** *exI*)

apply(*rule-tac x=snd* (cms_A ! *i*) **in** *exI*)
apply(*rule-tac x=*mem_A*-of* mem_C **in** *exI*)
apply(*rule conjI*)
using *in-*$\mathcal{R}$ **apply** *fastforce*
apply(*rule-tac x=fst* (cms_A ! *i*) **in** *exI*)
apply(*rule-tac x=snd* (cms_A ! *i*) **in** *exI*)
apply(*rule-tac x=*mem_A*-of* mem_C*'* **in** *exI*)
apply(*rule conjI*)
using *in-*$\mathcal{R}$ **apply** *fastforce*
apply(*fastforce simp*: *in-*$\mathcal{R}_A$ *Ps-refl-on-low-mds-eq*)
apply(*clarify*)
apply(*rename-tac* mem_C)
apply(*rule-tac* gc_A*=*(cms_A,mem_A*-of* mem_C) **in** *sound-mode-use-preservation*)
apply *simp*
apply *simp*
using *in-*$\mathcal{R}$ **apply** *fastforce*
apply(*rule sound-mode-use*$_A$)
apply *clarsimp*
by(*rule modes-respect-priv*)

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
apply(*rule refined-prog-secure*)
apply(*rule* len_A)
apply(*rule* len_C)
apply(*blast intro*: *in-*$\mathcal{R}$)
apply(*rule in-*$\mathcal{R}_A$)
apply *assumption*
apply(*subgoal-tac snd* (cms_A ! *i*) = mds_A*-of* (*snd* (cms_C ! *i*)))
using *low-mds-eq-from-conc-to-abs* **apply** *fastforce*
apply(*rule-tac* $\mathcal{R}$*1=*$\mathcal{R}$*-rel* (*cms* ! *i*) **and** c_A*1=fst* (cms_A ! *i*) **and** c_C*1=fst* (cms_C ! *i*) **in** *preserves-modes-memD*[*THEN conjunct2*])
using *secure-refinements* **unfolding** *secure-refinement-def* **apply** *fast*
apply *clarsimp*
using *in-*$\mathcal{R}$ **apply** *fastforce*
apply(*blast intro*: *sound-mode-use*$_A$)
by(*rule modes-respect-priv*)

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*
apply(*clarsimp simp*: *reachable-mems-def*)
apply(*rule-tac x*=[] **in** *exI*)
apply *fastforce*
done

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-*$\mathcal{R}$: (⋀*i* mem_C. *i* < *length cms* ⟹ ((cms_A ! *i*,mem_A*-of* mem_C),(cms_C ! *i*, mem_C)) ∈ $\mathcal{R}$*-rel* (*cms* ! *i*))
assumes *sound-mode-use*$_A$: (⋀ 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' ∈ *conc.reachable-mems* cms_C mem_C
shows mem_A*-of* mem_C' ∈ *abs.reachable-mems* cms_A (mem_A*-of* mem_C)
proof −
from *reachable*$_C$ **obtain** *sched*$_C$ cms_C' **where**
meval-sched$_C$: *conc.meval-sched sched*$_C$ (cms_C, mem_C) (cms_C', mem_C')
by (*fastforce simp*: *conc.reachable-mems-def*)

let *?*mem_A = mem_A*-of* mem_C

have *sound-mode-use*$_A$: *abs.sound-mode-use* (cms_A, *?*mem_A)
by(*rule sound-mode-use*$_A$)

from *traces-refinement*[**where** gc_A=(cms_A,*?*mem_A), *OF meval-sched*$_C$, *OF - - - sound-mode-use*$_A$]
in-$\mathcal{R}$[*of - mem*$_C$]
modes-respect-priv
obtain *sched*$_A$ cms_A' mem_A' **where**
meval-sched$_A$: *abs.meval-sched sched*$_A$ (cms_A, *?*mem_A) (cms_A', mem_A') **and**
in-$\mathcal{R}'$: (∀ *i*<*length cms*.
(((cms_A' ! *i*, mem_A'), cms_C' ! *i*, mem_C') ∈ $\mathcal{R}$*-rel* (*cms* ! *i*))
by *fastforce*

```
  hence reachable_A: mem_A' ∈ abs.reachable-mems cms_A ?mem_A
    by(fastforce simp: abs.reachable-mems-def)
  from sys-nonempty obtain i where ilen: i < length cms by blast
  let ?Ri = R-rel (cms ! i)
  from ilen secure-refinements have preserves-modes-mem ?Ri
    unfolding secure-refinement-def by blast
  from ilen in-R' preserves-modes-memD[OF this] have
    mem_A'-def: mem_A' = mem_A-of mem_C'
    by(metis surjective-pairing)
  with reachable_A show ?thesis by simp
qed

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.