Fully Composable and Adequate Verified Compilation with Direct Refinements between Open Modules

Ling Zhang [1] **Yuting Wang** [1] Jinhua Wu [1] Jérémie Koenig [2] Zhong Shao [2]

[1] Shanghai Jiao Tong University, China

[2] Yale University, USA

POPL, January 2024, London

Verified Compilation

CompCert: the State-of-the-Art

- Verified compilation of a subset of C into assembly in Coq
- Many Applications: CertiKOS, VST, Critical control system, etc.

Compiler Correctness = Refinement of Semantics

- $[\![M_1]\!] \preccurlyeq [\![M_2]\!]$ denotes the semantics of M_1 refines that of M_2

$$\begin{tikzcd}
\boxed{\texttt{foo.c}} \arrow[r, Rightarrow, "extract", "semantics"'] \arrow[d, "\text{CompCert}"] & {[\![\texttt{foo.c}]\!]} \arrow[d, phantom, "\rotatebox{90}{$\preccurlyeq$}"] \\
\boxed{\texttt{foo.s}} \arrow[r, Rightarrow, "extract", "semantics"'] & {[\![\texttt{foo.s}]\!]}
\end{tikzcd}$$

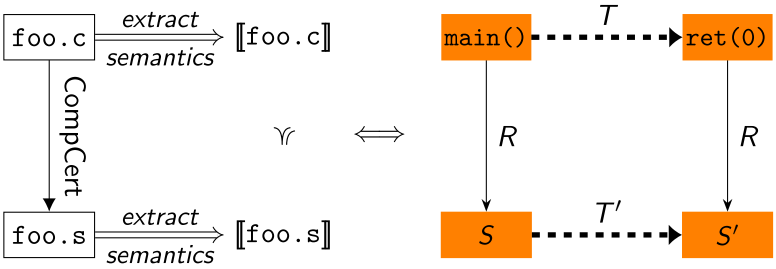
Verified Compilation

CompCert: the State-of-the-Art

- Verified compilation of a subset of C into assembly in Coq
- Many Applications: CertiKOS, VST, Critical control system, etc.

Compiler Correctness = Refinement of Semantics

- $[\![M_1]\!] \preccurlyeq [\![M_2]\!]$ denotes the semantics of M_1 refines that of M_2

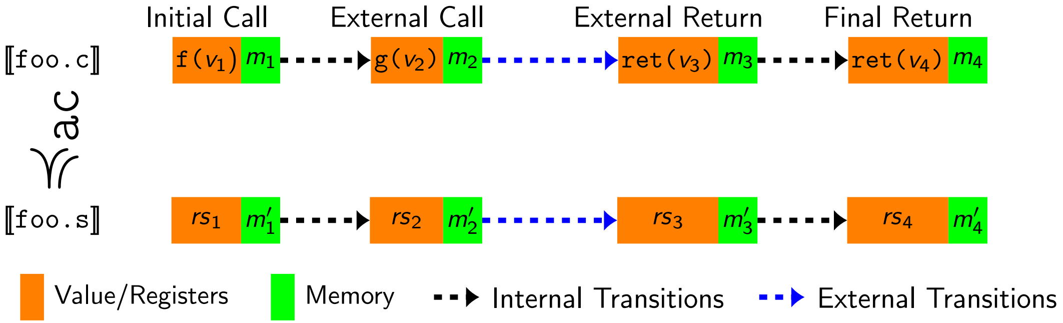


T: Transitions

R: Invariants

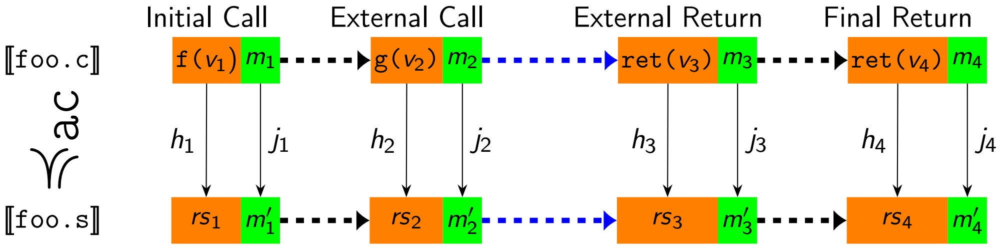
Verified Compilation of Open Modules

Intuition: Get a refinement directly relating semantics of C and assembly modules.



Verified Compilation of Open Modules

Intuition: Get a refinement directly relating semantics of C and assembly modules.



🟧 Value/Registers 🟩 Memory ▪▪▶ Internal Transitions ▪▪▶ External Transitions

h : Invariants of Language States j : Invariants of Memory States

Problem: Indirect Refinements

Observation: No existing work on CompCert produces direct refinement

Compositional CompCert [*POPL'15*]: $[\![\texttt{foo.s}]\!] \quad \preccurlyeq_{\mathcal{C}} \quad [\![\texttt{foo.c}]\!]$

CompCertM [*POPL'20*]: $[\![\texttt{foo.s}]\!] \preccurlyeq_1 + \preccurlyeq_2 + \ldots + \preccurlyeq_n [\![\texttt{foo.c}]\!]$

CompCertO [*PLDI'21*]: $[\![\texttt{foo.s}]\!] \quad \preccurlyeq_1 \cdot \preccurlyeq_2 \cdot \ldots \cdot \preccurlyeq_n \quad [\![\texttt{foo.c}]\!]$

Problem: Indirect Refinements

Observation: No existing work on CompCert produces direct refinement

Compositional CompCert [*POPL'15*]: $[\![\texttt{foo.s}]\!] \quad \underset{\substack{\uparrow \\ \text{Only with C interfaces}}}{\boxed{\preccurlyeq_{\mathcal{C}}}} \quad [\![\texttt{foo.c}]\!]$

CompCertM [*POPL'20*]: $[\![\texttt{foo.s}]\!] \preccurlyeq_1 + \preccurlyeq_2 + \ldots + \preccurlyeq_n [\![\texttt{foo.c}]\!]$

CompCertO [*PLDI'21*]: $[\![\texttt{foo.s}]\!] \quad \preccurlyeq_1 \cdot \preccurlyeq_2 \cdot \ldots \cdot \preccurlyeq_n \quad [\![\texttt{foo.c}]\!]$

Problem: Indirect Refinements

Observation: No existing work on CompCert produces direct refinement

Compositional CompCert [*POPL'15*]: $[\![\texttt{foo.s}]\!] \quad \boxed{\preccurlyeq_{\mathcal{C}}} \quad [\![\texttt{foo.c}]\!]$

↑

Only with C interfaces

CompCertM [*POPL'20*]: $[\![\texttt{foo.s}]\!]\boxed{\preccurlyeq_1 + \preccurlyeq_2 + \ldots + \preccurlyeq_n}[\![\texttt{foo.c}]\!]$

↑

Union of Refinements

CompCertO [*PLDI'21*]: $[\![\texttt{foo.s}]\!] \quad \preccurlyeq_1 \cdot \preccurlyeq_2 \cdot \ldots \cdot \preccurlyeq_n \quad [\![\texttt{foo.c}]\!]$

Problem: Indirect Refinements

Observation: No existing work on CompCert produces direct refinement

Compositional CompCert [*POPL'15*]: $[\![\texttt{foo.s}]\!] \quad \boxed{\preccurlyeq_{\mathcal{C}}} \quad [\![\texttt{foo.c}]\!]$

$\uparrow$

Only with C interfaces

CompCertM [*POPL'20*]: $[\![\texttt{foo.s}]\!] \boxed{\preccurlyeq_1 + \preccurlyeq_2 + \ldots + \preccurlyeq_n} [\![\texttt{foo.c}]\!]$

$\uparrow$

Union of Refinements

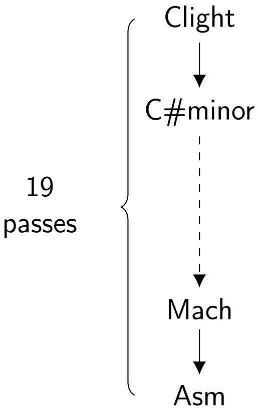
CompCertO [*PLDI'21*]: $[\![\texttt{foo.s}]\!] \quad \boxed{\preccurlyeq_1 \cdot \preccurlyeq_2 \cdot \ldots \cdot \preccurlyeq_n} \quad [\![\texttt{foo.c}]\!]$

$\uparrow$

Concatenation of Refinements

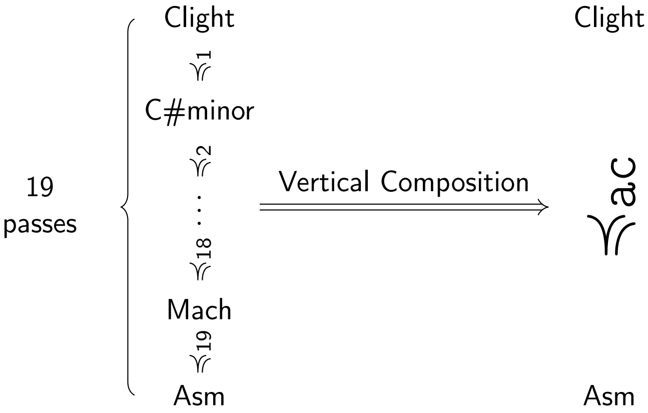
Main Challenge

Challenge: Vertical composition of direct refinements is difficult



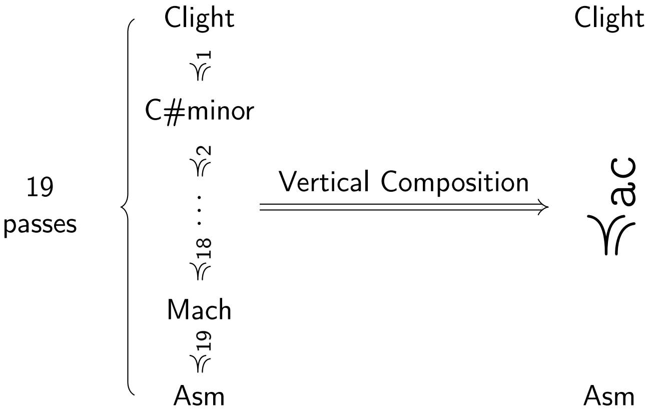
Main Challenge

Challenge: Vertical composition of direct refinements is difficult



Main Challenge

Challenge: Vertical composition of direct refinements is difficult



Q: Can we hide the intermediate semantics?

Our Contributions

Approach to Direct Refinements Supporting:

- Vertical and horizontal composition
- Equivalence of semantics and syntactic linking (i.e., Adequacy)
- Heterogeneous modules with mutual calls

Applications:

- CompCert's full compilation chain
- Extension to user-level verification

Notice: We focus on imperative programs with global memory and pointers.

A Running Example

Heterogeneous Modules with Callbacks and Pointer Passing:

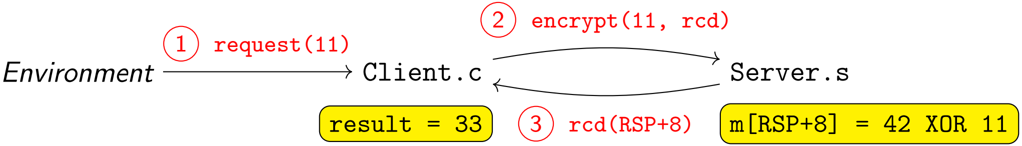
- A client written in C;
- An encryption server written in X86 assembly.

```
/* Client.c */
int result;
void encrypt(int i, void(*p)(int*));

static void rcd(int *r) {
  result = *r;
}
// Entry point
int request(int i) {
  encrypt(i,rcd);
  return result;
}
```

```
/* Server.s */
key: .long 42
encrypt:
  ... // Alloc 24-bytes frame
  // RSP[8] = key XOR i
  mov key RAX
  xor RAX RDI
  mov RDI 8(RSP)
  // call p(RSP + 8)
  lea 8(RSP) RDI
  call RSI
  ...
```

A Running Example



Environment

1 request(11)

Client.c

2 encrypt(11, rcd)

Server.s

3 rcd(RSP+8)

result = 33

m[RSP+8] = 42 XOR 11

```
/* Client.c */
int result;
void encrypt(int i, void(*p)(int*));

static void rcd(int *r) {
  result = *r;
}
// Entry point
int request(int i) {
  encrypt(i,rcd);
  return result;
}
```

```
/* Server.s */
key: .long 42
encrypt:
  ... // Alloc 24-bytes frame
  // RSP[8] = key XOR i
  mov key RAX
  xor RAX RDI
  mov RDI 8(RSP)
  // call p(RSP + 8)
  lea 8(RSP) RDI
  call RSI
  ...
```

Verification Steps

(1) Prove CompCert has the direct refinement $\preccurlyeq_{\mathtt{ac}}$;

(2) Prove $\llbracket \mathtt{Server.s} \rrbracket \preccurlyeq_{\mathtt{ac}} L_{\mathtt{S}}$;

(3) Exploit the compositionality and adequacy of $\preccurlyeq_{\mathtt{ac}}$.

$$\begin{tikzcd}
\llbracket \mathtt{Client.c} \rrbracket \arrow[d, "\mathtt{CompCert}"'] \arrow[d, phantom, shift left=4ex, "\preccurlyeq_{\mathtt{ac}}\ (1)"] & \oplus & L_{\mathtt{S}} \arrow[d, phantom, "\preccurlyeq_{\mathtt{ac}}\ (2)"] \\
\llbracket \mathtt{Client.s} \rrbracket & \oplus \arrow[d, phantom, "\preccurlyeq_{\mathtt{id}}"] & \llbracket \mathtt{Server.s} \rrbracket \\
 & \llbracket \mathtt{Client.s} + \mathtt{Server.s} \rrbracket \arrow[uur, bend right, "\preccurlyeq_{\mathtt{ac}}\ (3)"'] &
\end{tikzcd}$$

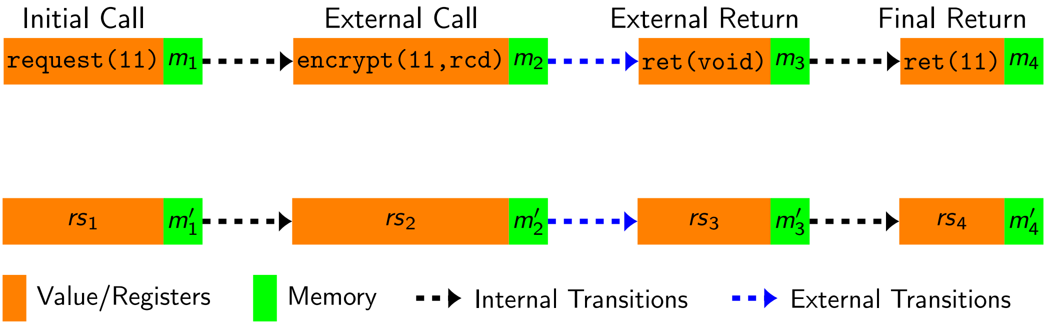
Key Ideas

1. Direct Refinements for Adequacy and Horizontal Composition
2. Transitive Kripke Memory Relation for Vertical Composition

Direct Refinement for CompCert

Direct refinement $\preccurlyeq_{\mathrm{ac}}$ as forward simulation with

- Invariant for source and target program states;

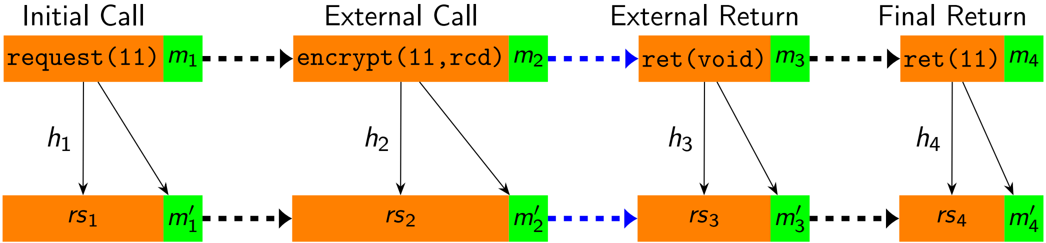


$$[\![\texttt{Client.s}]\!] \preccurlyeq_{\mathrm{ac}} [\![\texttt{Client.c}]\!]$$

Direct Refinement for CompCert

Direct refinement $\preccurlyeq_{ac}$ as forward simulation with

- Invariant for source and target program states;



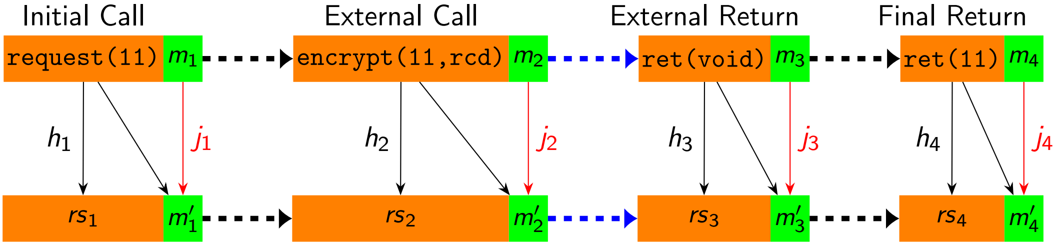
🟧 Value/Registers 🟩 Memory h : Invariants of Values

$$[\![\texttt{Client.s}]\!] \preccurlyeq_{ac} [\![\texttt{Client.c}]\!]$$

Direct Refinement for CompCert

Direct refinement $\preccurlyeq_{\mathrm{ac}}$ as forward simulation with

- Invariant for source and target program states;



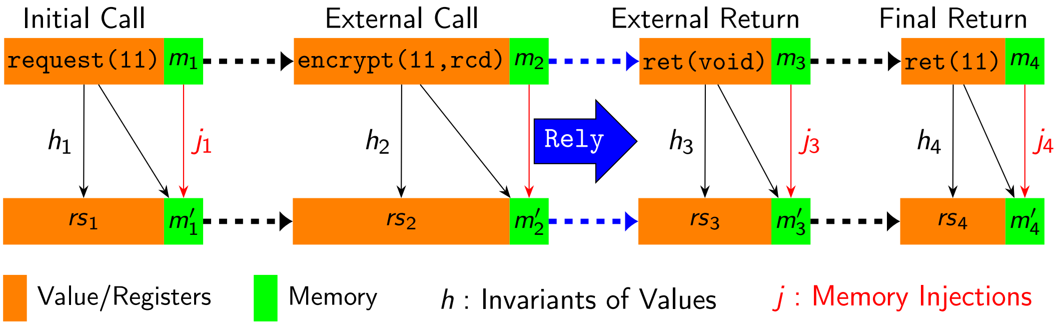
🟧 Value/Registers 🟩 Memory h : Invariants of Values j : Memory Injections

$$[\![\texttt{Client.s}]\!] \preccurlyeq_{\mathrm{ac}} [\![\texttt{Client.c}]\!]$$

Direct Refinement for CompCert

Direct refinement $\preccurlyeq_{\mathrm{ac}}$ as forward simulation with

- Invariant for source and target program states;
- Protection for program states across external calls.



$$[\![\texttt{Client.s}]\!] \preccurlyeq_{\mathrm{ac}} [\![\texttt{Client.c}]\!]$$

Adequacy of Direct Refinements

Adequacy trivially holds as invariants directly relates C and assembly states:

- Invariants formalize the CompCert C calling convention;
- Source function arguments are mapped directly to registers and the stack.

$$\begin{array}{c}[\![\texttt{Client.s}]\!] \qquad \oplus \qquad [\![\texttt{Server.s}]\!] \\ \rotatebox[origin=c]{90}{$\succcurlyeq_{\texttt{id}}$} \\ [\![\texttt{Client.s} + \texttt{Server.s}]\!]\end{array}$$

Horizontal Composition of Direct Refinements

Direct protection of private states against external calls :

- Callee-saved registers and stack pointer must be restored upon returning.
- Private stack memory (e.g., spilled registers) must not be modified

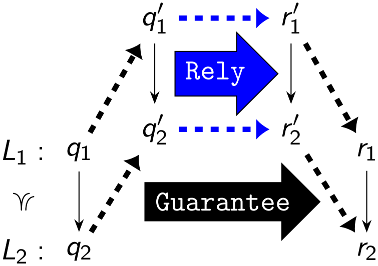
Rely-guarantee reasoning

Horizontal Composition of Direct Refinements

Direct protection of private states against external calls :

- Callee-saved registers and stack pointer must be restored upon returning.
- Private stack memory (e.g., spilled registers) must not be modified

Rely-guarantee reasoning

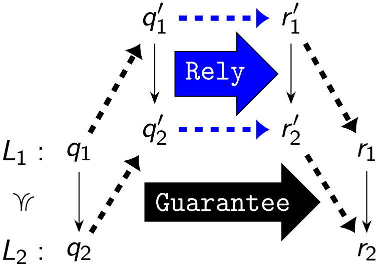


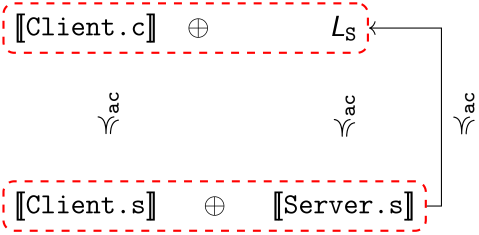
Horizontal Composition of Direct Refinements

Direct protection of private states against external calls :

- Callee-saved registers and stack pointer must be restored upon returning.
- Private stack memory (e.g., spilled registers) must not be modified

Rely-guarantee reasoning




$[\![$Client.c$]\!]$ $\oplus$ L_S
$\preccurlyeq_{ac}$
$\preccurlyeq_{ac}$
$\preccurlyeq_{ac}$
$[\![$Client.s$]\!]$ $\oplus$ $[\![$Server.s$]\!]$

($\preccurlyeq_{ac}$ has symmetric rely and guarantee conditions)

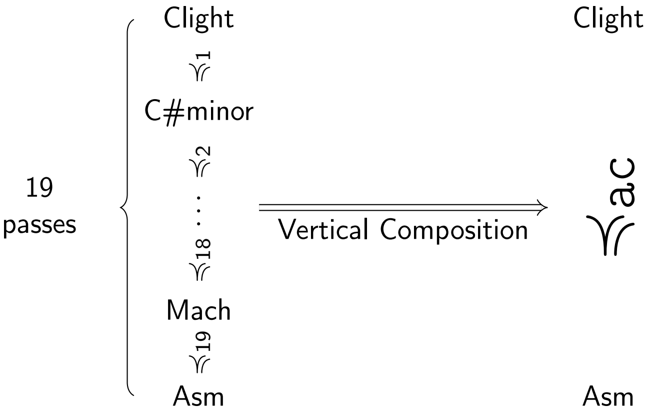
Key Ideas

1. Direct Refinements for Adequacy and Horizontal Composition

2. Transitive Kripke Memory Relation for Vertical Composition

Recall the Challenge

Challenge: Vertical composition of refinements



19 passes

Clight

$\preccurlyeq_1$

C#minor

$\preccurlyeq_2$

$\vdots$

$\preccurlyeq_{18}$

Mach

$\preccurlyeq_{19}$

Asm

Vertical Composition

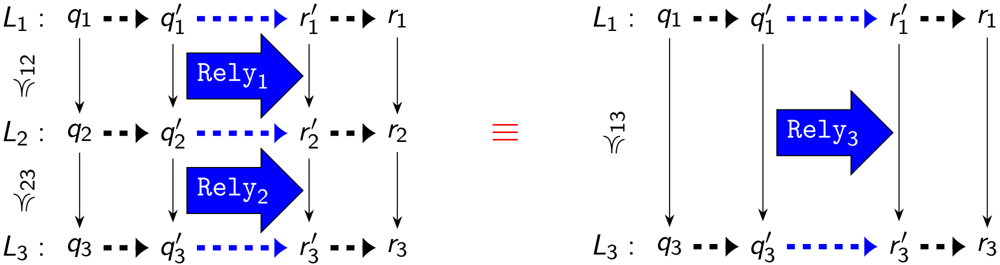
Clight

$\preccurlyeq_{ac}$

Asm

Recall the Challenge

Challenge: Vertical composition of rely-guarantee conditions

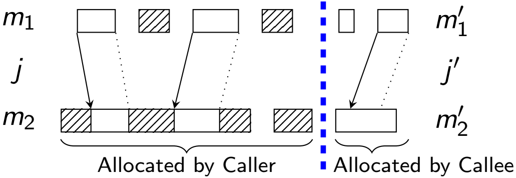


$$\begin{tikzcd}
L_1 : \arrow[d, phantom, "\preccurlyeq_{12}"] & q_1 \arrow[r, dashed] \arrow[d] & q_1' \arrow[r, dashed, "\mathtt{Rely}_1"'] \arrow[d] & r_1' \arrow[r, dashed] \arrow[d] & r_1 \arrow[d] \\
L_2 : \arrow[d, phantom, "\preccurlyeq_{23}"] & q_2 \arrow[r, dashed] \arrow[d] & q_2' \arrow[r, dashed, "\mathtt{Rely}_2"'] \arrow[d] & r_2' \arrow[r, dashed] \arrow[d] & r_2 \arrow[d] \\
L_3 : & q_3 \arrow[r, dashed] & q_3' \arrow[r, dashed] & r_3' \arrow[r, dashed] & r_3
\end{tikzcd}
\equiv
\begin{tikzcd}
L_1 : \arrow[d, phantom, "\preccurlyeq_{13}"] & q_1 \arrow[r, dashed] \arrow[d] & q_1' \arrow[r, dashed, "\mathtt{Rely}_3"'] \arrow[d] & r_1' \arrow[r, dashed] \arrow[d] & r_1 \arrow[d] \\
L_3 : & q_3 \arrow[r, dashed] & q_3' \arrow[r, dashed] & r_3' \arrow[r, dashed] & r_3
\end{tikzcd}$$

A Kripke Relation with Memory Protection

Kripke relation `injp` for protection:

- At an external call, infer private memory from the injection;
- No modification to private memory allowed during the call.

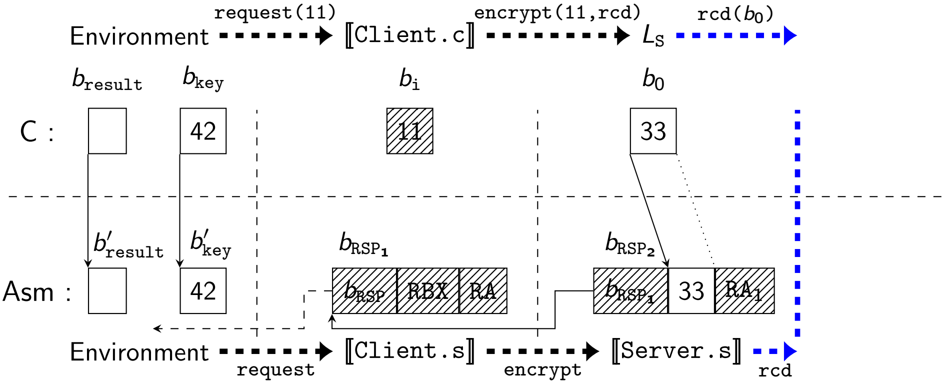


Private memory are the shaded areas, including

- Source caller's memory NOT in the domain of j
- Target caller's memory NOT in the image of j

Example of Memory Protection by `injp`

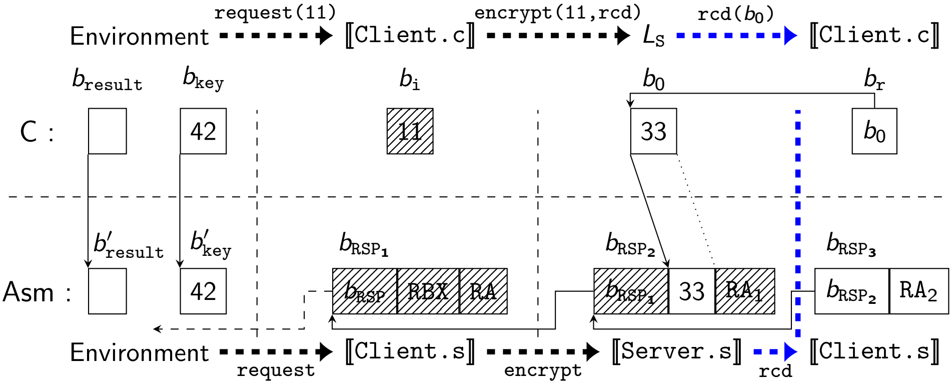
<span style="color:red">Before</span> the server calls back `rcd`:



Protected Memory: b_i, $b_{\texttt{RSP}_1}$, and part of $b_{\texttt{RSP}_2}$

Example of Memory Protection by `injp`

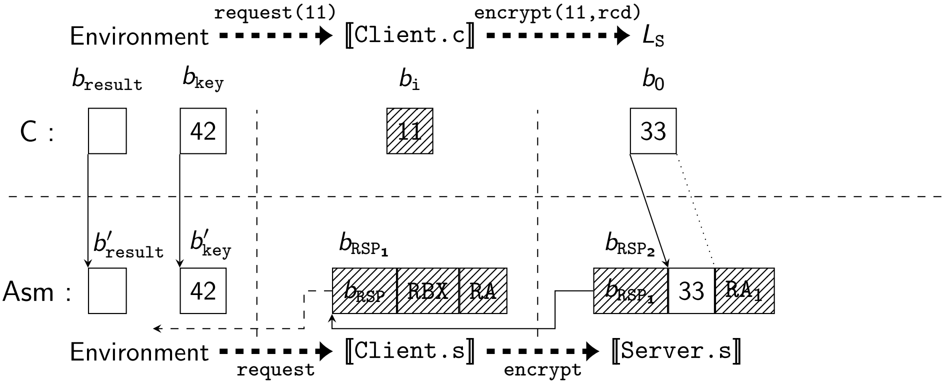
During the server calls back `rcd`:



Protected Memory: b_i, $b_{\texttt{RSP}_1}$, and part of $b_{\texttt{RSP}_2}$

Example of Memory Protection by `injp`

After the server calls back `rcd`:



Protected Memory: b_i, $b_{\texttt{RSP}_1}$, and part of $b_{\texttt{RSP}_2}$

Vertical Composition of Direct Refinements

Observations:

- injp is uniform: its protection works for all passes;
- injp is transitive: $\texttt{injp} \cdot \texttt{injp} \equiv \texttt{injp}$.

$$\begin{tikzcd}
L_1 : \arrow[d, phantom, "\succcurlyeq_{12}"] & q_1 \arrow[r, dashed] \arrow[d] & q_1' \arrow[r, dashed, blue] \arrow[d] \arrow[dr, phantom, "\texttt{injp}"] & r_1' \arrow[r, dashed] \arrow[d] & r_1 \arrow[d] \\
L_2 : \arrow[d, phantom, "\succcurlyeq_{23}"] & q_2 \arrow[r, dashed] \arrow[d] & q_2' \arrow[r, dashed, blue] \arrow[d] \arrow[dr, phantom, "\texttt{injp}"] & r_2' \arrow[r, dashed] \arrow[d] & r_2 \arrow[d] \\
L_3 : & q_3 \arrow[r, dashed] & q_3' \arrow[r, dashed, blue] & r_3' \arrow[r, dashed] & r_3
\end{tikzcd}$$

$$\equiv$$

$$\begin{tikzcd}
L_1 : \arrow[d, phantom, "\succcurlyeq_{13}"] & q_1 \arrow[r, dashed] \arrow[d] & q_1' \arrow[r, dashed, blue] \arrow[d] \arrow[dr, phantom, "\texttt{injp}"] & r_1' \arrow[r, dashed] \arrow[d] & r_1 \arrow[d] \\
L_3 : & q_3 \arrow[r, dashed] & q_3' \arrow[r, dashed, blue] & r_3' \arrow[r, dashed] & r_3
\end{tikzcd}$$

Transitivity of `injp`

Key to prove $\texttt{injp} \cdot \texttt{injp} \equiv \texttt{injp}$:

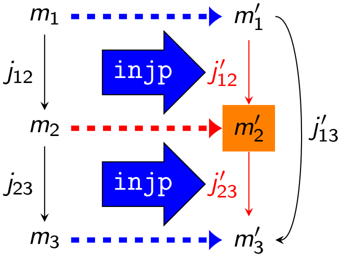
- Construct an interpolating state when the external call returns.

$$\begin{tikzcd}
m_1 \arrow[r, dashed] \arrow[d, "j_{12}"'] & m_1' \arrow[dd, "j_{13}'"] \\
m_2 \arrow[d, "j_{23}"'] \arrow[r, phantom, "\texttt{injp} \Rightarrow"] & \\
m_3 \arrow[r, dashed] & m_3'
\end{tikzcd}$$

Transitivity of `injp`

Key to prove $\texttt{injp} \cdot \texttt{injp} \equiv \texttt{injp}$:

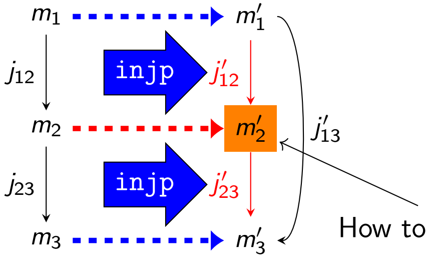
- Construct an interpolating state when the external call returns.



Transitivity of `injp`

Key to prove $\texttt{injp} \cdot \texttt{injp} \equiv \texttt{injp}$:

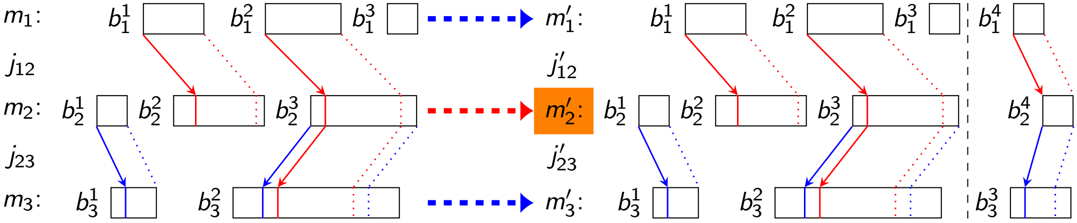
- Construct an interpolating state when the external call returns.



How to construct it?

Construction of the Interpolating State

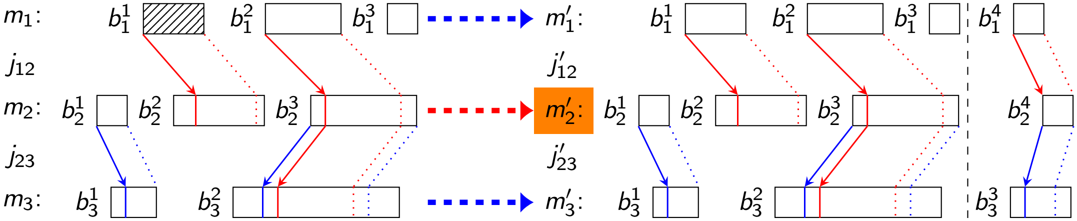
Protection after Composition $\geqslant$ Protection before Composition



- Public memory of $m_2' =$ (Image of j_{12}) $\cap$ (Domain of j_{23});
- m_2' is unchanged from m_2 except for its public memory is projected from m_1'.

Construction of the Interpolating State

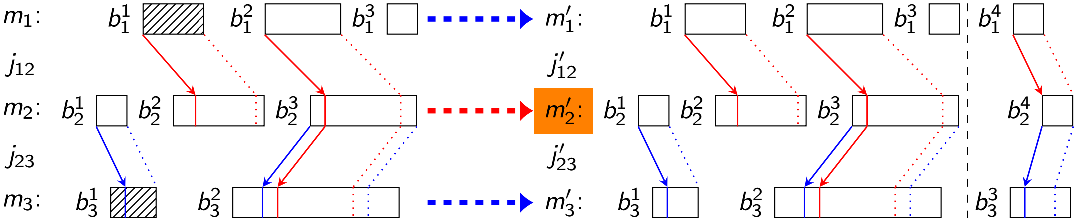
Protection after Composition $\geqslant$ Protection before Composition



- Public memory of m'_2 = (Image of j_{12}) $\cap$ (Domain of j_{23});
- m'_2 is unchanged from m_2 except for its public memory is projected from m'_1.

Construction of the Interpolating State

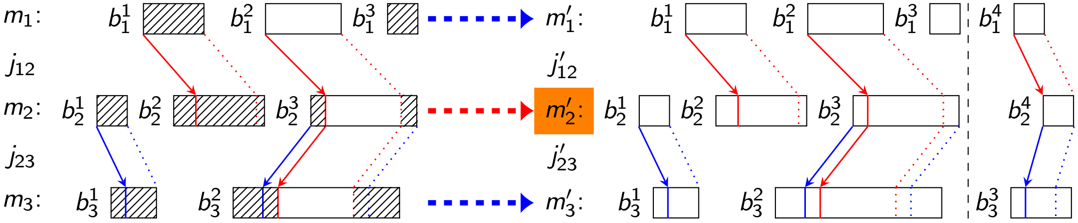
Protection after Composition $\geqslant$ Protection before Composition



- Public memory of m'_2 = (Image of j_{12}) $\cap$ (Domain of j_{23});
- m'_2 is unchanged from m_2 except for its public memory is projected from m'_1.

Construction of the Interpolating State

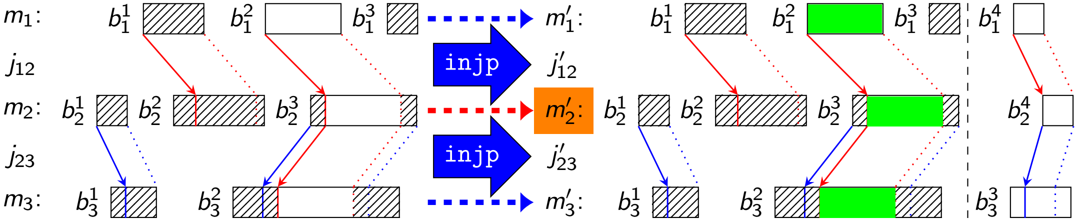
Protection after Composition $\geqslant$ Protection before Composition



- Public memory of m'_2 = (Image of j_{12}) $\cap$ (Domain of j_{23});
- m'_2 is unchanged from m_2 except for its public memory is projected from m'_1.

Construction of the Interpolating State

Protection after Composition $\geqslant$ Protection before Composition

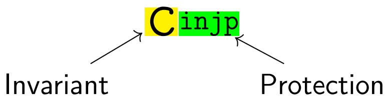


- Public memory of m'_2 = (Image of j_{12}) $\cap$ (Domain of j_{23});
- m'_2 is unchanged from m_2 except for its public memory is projected from m'_1.

CompCert(O) with Direct Refinement

Passes	Rely $\twoheadrightarrow$ Guarantee
Self-Sim	$\mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}}$
SimplLocals	$\mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathbb{C}_{\mathtt{inj}}$
Cminorgen	$\mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathbb{C}_{\mathtt{inj}}$
Selection	$\mathtt{wt} \cdot \mathbb{C}_{\mathtt{ext}} \twoheadrightarrow \mathtt{wt} \cdot \mathbb{C}_{\mathtt{ext}}$
RTLgen	$\mathbb{C}_{\mathtt{ext}} \twoheadrightarrow \mathbb{C}_{\mathtt{ext}}$
Self-Sim	$\mathbb{C}_{\mathtt{inj}} \twoheadrightarrow \mathbb{C}_{\mathtt{inj}}$
Tailcall	$\mathbb{C}_{\mathtt{ext}} \twoheadrightarrow \mathbb{C}_{\mathtt{ext}}$
Inlining	$\mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathbb{C}_{\mathtt{inj}}$
Self-Sim	$\mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathbb{C}_{\mathtt{injp}}$
Constprop	$\mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}}$
CSE	$\mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}}$
Deadcode	$\mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathbb{C}_{\mathtt{injp}}$
Unusedglob	$\mathbb{C}_{\mathtt{inj}} \twoheadrightarrow \mathbb{C}_{\mathtt{inj}}$
Allocation	$\mathtt{wt} \cdot \mathbb{C}_{\mathtt{ext}} \cdot \mathsf{CL} \twoheadrightarrow \mathtt{wt} \cdot \mathbb{C}_{\mathtt{ext}} \cdot \mathsf{CL}$
Tunneling	$\mathtt{ltl}_{\mathtt{ext}} \twoheadrightarrow \mathtt{ltl}_{\mathtt{ext}}$
Stacking	$\mathtt{ltl}_{\mathtt{injp}} \cdot \mathsf{LM} \twoheadrightarrow \mathsf{LM} \cdot \mathtt{mach}_{\mathtt{inj}}$
Asmgen	$\mathtt{mach}_{\mathtt{ext}} \cdot \mathsf{MA} \twoheadrightarrow \mathtt{mach}_{\mathtt{ext}} \cdot \mathsf{MA}$
Self-Sim	$\mathtt{asm}_{\mathtt{inj}} \cdot \mathtt{asm}_{\mathtt{injp}} \twoheadrightarrow \mathtt{asm}_{\mathtt{inj}} \cdot \mathtt{asm}_{\mathtt{injp}}$

Significant Passes



$$\mathbb{C}_{\mathtt{injp}} \cdot \mathbb{C}_{\mathtt{injp}} \equiv \mathbb{C}_{\mathtt{injp}}$$

$$\mathbb{C}_{\mathtt{ext}} \cdot \mathbb{C}_{\mathtt{injp}} \equiv \mathbb{C}_{\mathtt{injp}}$$

$$\ldots$$

CompCert(O) with Direct Refinement

Passes	Rely $\twoheadrightarrow$ Guarantee
Self-Sim	$\mathtt{ro}\cdot\mathtt{C_{injp}} \twoheadrightarrow \mathtt{ro}\cdot\mathtt{C_{injp}}$
SimplLocals	$\mathtt{C_{injp}} \twoheadrightarrow \mathtt{C_{inj}}$
Cminorgen	$\mathtt{C_{injp}} \twoheadrightarrow \mathtt{C_{inj}}$
Selection	$\mathtt{wt}\cdot\mathtt{C_{ext}} \twoheadrightarrow \mathtt{wt}\cdot\mathtt{C_{ext}}$
RTLgen	$\mathtt{C_{ext}} \twoheadrightarrow \mathtt{C_{ext}}$
Self-Sim	$\mathtt{C_{inj}} \twoheadrightarrow \mathtt{C_{inj}}$
Tailcall	$\mathtt{C_{ext}} \twoheadrightarrow \mathtt{C_{ext}}$
Inlining	$\mathtt{C_{injp}} \twoheadrightarrow \mathtt{C_{inj}}$
Self-Sim	$\mathtt{C_{injp}} \twoheadrightarrow \mathtt{C_{injp}}$
Constprop	$\mathtt{ro}\cdot\mathtt{C_{injp}} \twoheadrightarrow \mathtt{ro}\cdot\mathtt{C_{injp}}$
CSE	$\mathtt{ro}\cdot\mathtt{C_{injp}} \twoheadrightarrow \mathtt{ro}\cdot\mathtt{C_{injp}}$
Deadcode	$\mathtt{ro}\cdot\mathtt{C_{injp}} \twoheadrightarrow \mathtt{ro}\cdot\mathtt{C_{injp}}$
Unusedglob	$\mathtt{C_{inj}} \twoheadrightarrow \mathtt{C_{inj}}$
Allocation	$\mathtt{wt}\cdot\mathtt{C_{ext}}\cdot\mathtt{CL} \twoheadrightarrow \mathtt{wt}\cdot\mathtt{C_{ext}}\cdot\mathtt{CL}$
Tunneling	$\mathtt{ltl_{ext}} \twoheadrightarrow \mathtt{ltl_{ext}}$
Stacking	$\mathtt{ltl_{injp}}\cdot\mathtt{LM} \twoheadrightarrow \mathtt{LM}\cdot\mathtt{mach_{inj}}$
Asmgen	$\mathtt{mach_{ext}}\cdot\mathtt{MA} \twoheadrightarrow \mathtt{mach_{ext}}\cdot\mathtt{MA}$
Self-Sim	$\mathtt{asm_{inj}}\cdot\mathtt{asm_{injp}} \twoheadrightarrow \mathtt{asm_{inj}}\cdot\mathtt{asm_{injp}}$

Significant Passes

$$\preccurlyeq \mathrm{ac} :$$

$$\mathtt{ro}\cdot\mathtt{C_{injp}}\cdot\mathtt{C_{injp}}\cdot\mathtt{C_{injp}}\cdot\mathtt{wt}\cdot\mathtt{C_{ext}}\cdot\mathtt{C_{ext}}\cdot\mathtt{C_{inj}}$$
$$\cdot\mathtt{C_{ext}}\cdot\mathtt{C_{injp}}\cdot\mathtt{C_{injp}}\cdot\mathtt{ro}\cdot\mathtt{C_{injp}}\cdot\mathtt{ro}\cdot\mathtt{C_{injp}}\cdot\mathtt{ro}$$
$$\cdot\mathtt{C_{injp}}\cdot\mathtt{C_{inj}}\cdot\mathtt{wt}\cdot\mathtt{C_{ext}}\cdot\mathtt{CL}\cdot\mathtt{ltl_{ext}}\cdot\mathtt{ltl_{injp}}$$
$$\cdot\mathtt{LM}\cdot\mathtt{mach_{ext}}\cdot\mathtt{MA}\cdot\mathtt{asm_{inj}}\cdot\mathtt{asm_{injp}}$$

$$\twoheadrightarrow$$

$$\mathtt{ro}\cdot\mathtt{C_{injp}}\cdot\mathtt{C_{inj}}\cdot\mathtt{C_{inj}}\cdot\mathtt{wt}\cdot\mathtt{C_{ext}}\cdot\mathtt{C_{ext}}\cdot\mathtt{C_{inj}}$$
$$\cdot\mathtt{C_{ext}}\cdot\mathtt{C_{inj}}\cdot\mathtt{C_{injp}}\cdot\mathtt{ro}\cdot\mathtt{C_{injp}}\cdot\mathtt{ro}\cdot\mathtt{C_{injp}}\cdot\mathtt{ro}$$
$$\cdot\mathtt{C_{injp}}\cdot\mathtt{C_{inj}}\cdot\mathtt{wt}\cdot\mathtt{C_{ext}}\cdot\mathtt{CL}\cdot\mathtt{ltl_{ext}}\cdot\mathtt{LM}$$
$$\cdot\mathtt{mach_{inj}}\cdot\mathtt{mach_{ext}}\cdot\mathtt{MA}\cdot\mathtt{asm_{inj}}\cdot\mathtt{asm_{injp}}$$

CompCert(O) with Direct Refinement

Passes	Rely $\twoheadrightarrow$ Guarantee
Self-Sim	$\mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}}$
SimplLocals	$\mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{C}_{\mathtt{inj}}$
Cminorgen	$\mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{C}_{\mathtt{inj}}$
Selection	$\mathtt{wt} \cdot \mathtt{C}_{\mathtt{ext}} \twoheadrightarrow \mathtt{wt} \cdot \mathtt{C}_{\mathtt{ext}}$
RTLgen	$\mathtt{C}_{\mathtt{ext}} \twoheadrightarrow \mathtt{C}_{\mathtt{ext}}$
Self-Sim	$\mathtt{C}_{\mathtt{inj}} \twoheadrightarrow \mathtt{C}_{\mathtt{inj}}$
Tailcall	$\mathtt{C}_{\mathtt{ext}} \twoheadrightarrow \mathtt{C}_{\mathtt{ext}}$
Inlining	$\mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{C}_{\mathtt{inj}}$
Self-Sim	$\mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{C}_{\mathtt{injp}}$
Constprop	$\mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}}$
CSE	$\mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}}$
Deadcode	$\mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}} \twoheadrightarrow \mathtt{ro} \cdot \mathtt{C}_{\mathtt{injp}}$
Unusedglob	$\mathtt{C}_{\mathtt{inj}} \twoheadrightarrow \mathtt{C}_{\mathtt{inj}}$
Allocation	$\mathtt{wt} \cdot \mathtt{C}_{\mathtt{ext}} \cdot \mathsf{CL} \twoheadrightarrow \mathtt{wt} \cdot \mathtt{C}_{\mathtt{ext}} \cdot \mathsf{CL}$
Tunneling	$\mathtt{ltl}_{\mathtt{ext}} \twoheadrightarrow \mathtt{ltl}_{\mathtt{ext}}$
Stacking	$\mathtt{ltl}_{\mathtt{injp}} \cdot \mathsf{LM} \twoheadrightarrow \mathsf{LM} \cdot \mathtt{mach}_{\mathtt{inj}}$
Asmgen	$\mathtt{mach}_{\mathtt{ext}} \cdot \mathsf{MA} \twoheadrightarrow \mathtt{mach}_{\mathtt{ext}} \cdot \mathsf{MA}$
Self-Sim	$\mathtt{asm}_{\mathtt{inj}} \cdot \mathtt{asm}_{\mathtt{injp}} \twoheadrightarrow \mathtt{asm}_{\mathtt{inj}} \cdot \mathtt{asm}_{\mathtt{injp}}$

Significant Passes

$$\preccurlyeq_{ac}:$$

$$\mathtt{ro} \cdot \mathtt{wt} \cdot \mathtt{CA}_{\mathtt{injp}} \cdot \mathtt{asm}_{\mathtt{injp}}$$

$$\twoheadrightarrow$$

$$\mathtt{ro} \cdot \mathtt{wt} \cdot \mathtt{CA}_{\mathtt{injp}} \cdot \mathtt{asm}_{\mathtt{injp}}$$

Conclusion

Direct refinements of realistic verified compilers are feasible:

$$\begin{tikzcd}
\boxed{\texttt{foo.c}} \arrow[r, Rightarrow] \arrow[d, "\text{CompCert}"'] & \llbracket \texttt{foo.c} \rrbracket \arrow[d, phantom, sloped, "\preccurlyeq_{ac}"] \\
\boxed{\texttt{foo.s}} \arrow[r, Rightarrow] & \llbracket \texttt{foo.s} \rrbracket
\end{tikzcd}$$

Discovery: Transitivity of Kripke Relation with Memory Protection

Ongoing/Future work:

- Reduce to the original CompCert
- Connect with Program Verification
- Verified Compilation of Safe/Unsafe Rust

`https://doi.org/10.5281/zenodo.10036618`