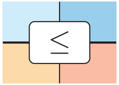
Three Dimensions of Compositionality in CompCert Semantics

Jérémie Koenig

Yale University

MFPS, June 21, 2023



Why Compilers are Interesting

One notion of compiler correctness is *semantics preservation*:

$$\mathsf{Compile}(p) = p' \quad\Longrightarrow\quad \mathsf{S}[\![p]\!] \;\leq\; \mathsf{T}[\![p']\!]$$

Why Compilers are Interesting

One notion of compiler correctness is *semantics preservation*:

$$\mathsf{Compile}(p) = p' \quad \Longrightarrow \quad \mathsf{S}[\![p]\!] \;\leq\; \mathsf{T}[\![p']\!]$$

In principle, we can get *compositional* compiler correctness by making $\mathsf{S}[\![-]\!]$ and $\mathsf{T}[\![-]\!]$ compositional semantics.

Why Compilers are Interesting

One notion of compiler correctness is *semantics preservation*:

$$\mathsf{Compile}(p) = p' \quad\Longrightarrow\quad \mathsf{S}[\![p]\!] \;\leq\; \mathsf{T}[\![p']\!]$$

In principle, we can get *compositional* compiler correctness by making $\mathsf{S}[\![-]\!]$ and $\mathsf{T}[\![-]\!]$ compositional semantics.

But in practice this is hard to do!

- S and T must be interpreted in the same domain
- We must formalize the **calling convention**

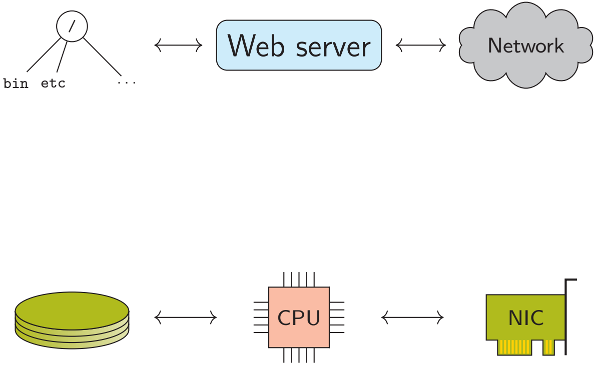
Traditional compositional semantics do not deal with that.

End-to-end verification of heterogeneous systems

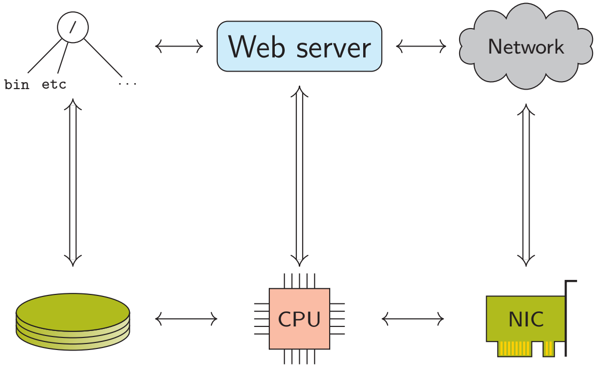
Web server

End-to-end verification of heterogeneous systems

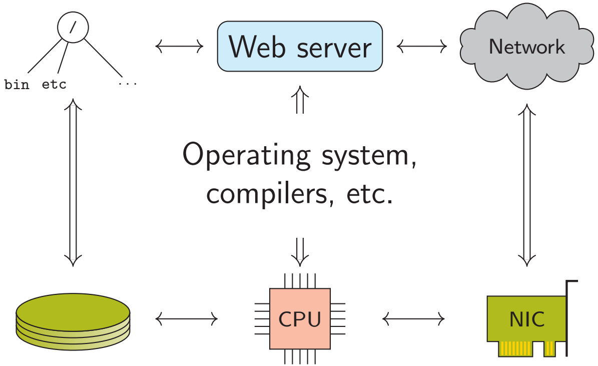
End-to-end verification of heterogeneous systems



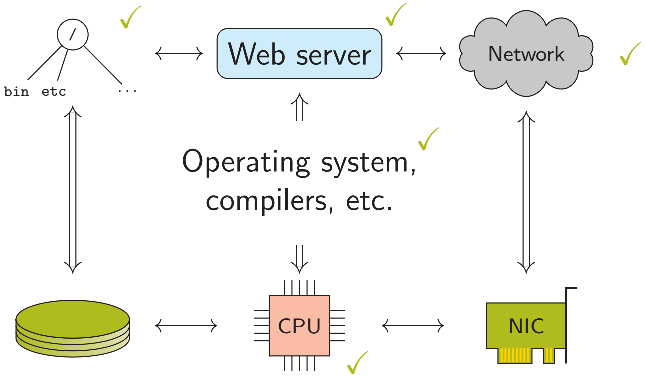
End-to-end verification of heterogeneous systems



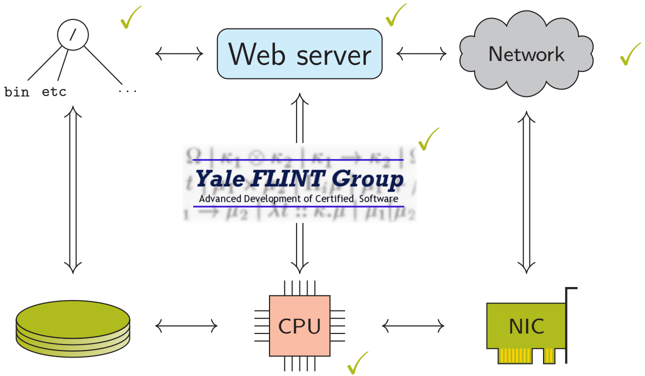
End-to-end verification of heterogeneous systems



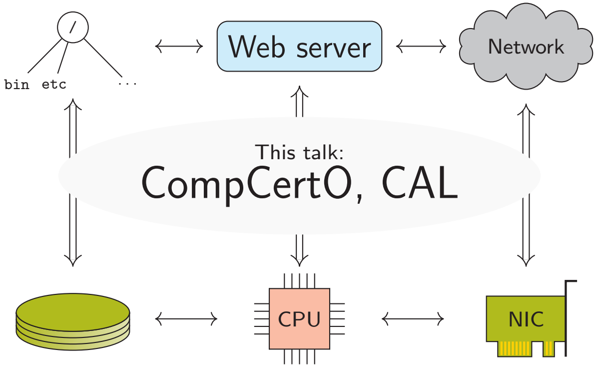
End-to-end verification of heterogeneous systems



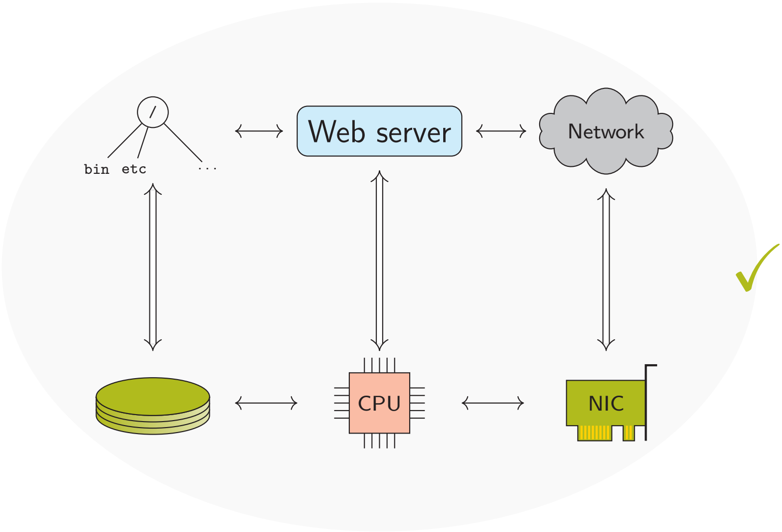
End-to-end verification of heterogeneous systems



End-to-end verification of heterogeneous systems



End-to-end verification of heterogeneous systems



Outline

Semantics in CompCert

For each source, target and intermediate (whole) program $p \in L$,
a transition system $L[p] = \langle S, \rightarrow, I, F \rangle$ is defined where:

$$S \in \mathsf{Set} \qquad \rightarrow \subseteq S \times S \qquad I \subseteq S \qquad F \subseteq S \times \mathsf{int}$$

Semantics in CompCert

For each source, target and intermediate (whole) program $p \in L$, a transition system $L[p] = \langle S, \to, I, F \rangle$ is defined where:

$$S \in \mathrm{Set} \qquad \to \subseteq S \times S \qquad I \subseteq S \qquad F \subseteq S \times \mathrm{int}$$

An execution of p corresponds to a transition sequence:

$$I \ni s_0 \to s_1 \to \cdots \to s_n \; F \; x$$

Semantics in CompCert

For each source, target and intermediate (whole) program $p \in L$, a transition system $L[p] = \langle S, \rightarrow, I, F \rangle$ is defined where:

$$S \in \text{Set} \qquad \rightarrow \subseteq S \times S \qquad I \subseteq S \qquad F \subseteq S \times \text{int}$$

An execution of p corresponds to a transition sequence:

$$I \ni s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n \; F \; x$$

CompCert's correctness is then established as a simulation property:

$$\text{CompCert}(p) = p' \quad \Longrightarrow \quad \text{Clight}[p] \; \leq \; \text{Asm}[p'] \,,$$

obtained by composing similar statements for each compilation pass.

Simulations

A simulation $\rho : L_1 \leq L_2 \in \mathrm{TS}$ of L_1 by L_2 is a relation $\rho \subseteq S_1 \times S_2$ such that:

- $s_1 \in I_1 \;\Rightarrow\; \exists s_2 \,.\, s_2 \in I_2 \;\wedge\; s_1 \;\rho\; s_2$
- $s_1 \;\rho\; s_2 \;\wedge\; s_1 \longrightarrow_1 s_1' \;\Rightarrow\; \exists s_2' \,.\, s_2 \longrightarrow_2 s_2' \;\wedge\; s_1' \;\rho\; s_2'$
- $s_1 \;\rho\; s_2 \;\wedge\; s_1 \; F_1 \; x \;\Rightarrow\; s_2 \; F_2 \; x$,

$$\begin{tikzcd}
s_1 \arrow[r] \arrow[d, dash, "\rho"'] & s_1' \arrow[d, dash, dashed, "\rho"] \\
s_2 \arrow[r, dashed] & s_2'
\end{tikzcd}$$

so that any execution of L_1 yields a a similar execution of L_2.

Simulations

A simulation $\rho : L_1 \leq L_2 \in \mathrm{TS}$ of L_1 by L_2 is a relation $\rho \subseteq S_1 \times S_2$ such that:

- $s_1 \in I_1 \;\Rightarrow\; \exists s_2 \,.\, s_2 \in I_2 \;\wedge\; s_1 \;\rho\; s_2$
- $s_1 \;\rho\; s_2 \;\wedge\; s_1 \rightarrow_1 s_1' \;\Rightarrow\; \exists s_2' \,.\, s_2 \rightarrow_2 s_2' \;\wedge\; s_1' \;\rho\; s_2'$
- $s_1 \;\rho\; s_2 \;\wedge\; s_1 \; F_1 \; x \;\Rightarrow\; s_2 \; F_2 \; x,$

$$\begin{tikzcd} s_1 \arrow[r] \arrow[d, no head, "\rho"'] & s_1' \arrow[d, no head, dashed, "\rho"] \\ s_2 \arrow[r, dashed] & s_2' \end{tikzcd}$$

so that any execution of L_1 yields a a similar execution of L_2.

Simulation relations compose in the expected way:

$$\frac{\rho : L_1 \leq L_2 \qquad \pi : L_2 \leq L_3}{(\rho \,;\, \pi) : L_1 \leq L_3}$$

making it possible to decompose the correctness proof.

Simulations

A simulation $\rho : L_1 \leq L_2 \in \mathrm{TS}$ of L_1 by L_2 is a relation $\rho \subseteq S_1 \times S_2$ such that:

- $s_1 \in I_1 \;\Rightarrow\; \exists s_2 \,.\, s_2 \in I_2 \;\wedge\; s_1 \;\rho\; s_2$
- $s_1 \;\rho\; s_2 \;\wedge\; s_1 \rightarrow_1 s_1' \;\Rightarrow\; \exists s_2' \,.\, s_2 \rightarrow_2 s_2' \;\wedge\; s_1' \;\rho\; s_2'$
- $s_1 \;\rho\; s_2 \;\wedge\; s_1 \; F_1 \; x \;\Rightarrow\; s_2 \; F_2 \; x,$

$$\begin{tikzcd} s_1 \arrow[r] \arrow[d, dash, "\rho"'] & s_1' \arrow[d, dash, dashed, "\rho"] \\ s_2 \arrow[r, dashed] & s_2' \end{tikzcd}$$

so that any execution of L_1 yields a a similar execution of L_2.

Simulation relations compose in the expected way:

$$\frac{\rho : L_1 \leq L_2 \qquad \pi : L_2 \leq L_3}{(\rho \,;\, \pi) : L_1 \leq L_3}$$

$$\begin{tikzcd} L_1 \arrow[d, Rightarrow, "\rho"] & & L_1 \arrow[dd, Rightarrow, "\rho;\pi"] \\ L_2 \arrow[d, Rightarrow, "\pi"] \arrow[rr, phantom, "\vdash"] & & \\ L_3 & & L_3 \end{tikzcd}$$

making it possible to decompose the correctness proof.

Vertical Composition $\boxed{;}$

In other words, transition systems and simulations form a category:

- the objects are transition systems
- the morphisms from L_1 to L_2 are the simulations $L_1 \leq L_2$
- The composition ; of simulations is associative
- $=$ is always a simulation relation and a unit for ;

Vertical Composition $\boxed{;}$

In other words, transition systems and simulations form a category:

- the objects are transition systems
- the morphisms from L_1 to L_2 are the simulations $L_1 \leq L_2$
- The composition ; of simulations is associative
- $=$ is always a simulation relation and a unit for ;

CompCert		
Object	Dimension	Operations
Transition system	0	
Simulation	1	;

Real programs are divided into pieces

Consider the following example:

rb.c

```
/* Encapsulated state */
static int c1, c2;
static V buf[N];

/* Accessors */
int inc1() { int i = c1++; c1 %= N; return i; }
int inc2() { int i = c2++; c2 %= N; return i; }
V get(int i) { return buf[i]; }
void set(int i, V val) { buf[i] = val; }
```

bq.c

```
/* Underlay signature */
extern int inc1(void);
extern int inc2(void);
extern V get(int i);
extern void set(int i, V val);

/* Layer implementation */
void enq(V val) { set(inc2(), val); }
V deq() { return get(inc1()); }
```

Real programs are divided into pieces

Consider the following example:

rb.c

```
/* Encapsulated state */
static int c1, c2;
static V buf[N];

/* Accessors */
int inc1() { int i = c1++; c1 %= N; return i; }
int inc2() { int i = c2++; c2 %= N; return i; }
V get(int i) { return buf[i]; }
void set(int i, V val) { buf[i] = val; }
```

bq.c

```
/* Underlay signature */
extern int inc1(void);
extern int inc2(void);
extern V get(int i);
extern void set(int i, V val);

/* Layer implementation */
void enq(V val) { set(inc2(), val); }
V deq() { return get(inc1()); }
```

These C translation units do not provide a main function,
so CompCert can compile them but provides no guarantee!

Real programs are divided into pieces

Consider the following example:

rb.c

```c
/* Encapsulated state */
static int c1, c2;
static V buf[N];

/* Accessors */
int inc1() { int i = c1++; c1 %= N; return i; }
int inc2() { int i = c2++; c2 %= N; return i; }
V get(int i) { return buf[i]; }
void set(int i, V val) { buf[i] = val; }
```

bq.c

```c
/* Underlay signature */
extern int inc1(void);
extern int inc2(void);
extern V get(int i);
extern void set(int i, V val);

/* Layer implementation */
void enq(V val) { set(inc2(), val); }
V deq() { return get(inc1()); }
```

These C translation units do not provide a main function, so CompCert can compile them but provides no guarantee!

To deal with this issue, *Compositional CompCert* gives semantics to individual translation units.

Semantics in Compositional CompCert

Transition systems are extended to $L = \langle S, \rightarrow, I, X, Y, F \rangle$.
The states S and internal steps $\rightarrow$ are as before.

Semantics in Compositional CompCert

Transition systems are extended to $L = \langle S, \rightarrow, I, X, Y, F \rangle$.
The states S and internal steps $\rightarrow$ are as before.

Initial and final states incorporate a *question* and an *answer* for the incoming call:

$$I \subseteq (\mathsf{ident} \times \mathsf{val}^* \times \mathsf{mem}) \times S \qquad\qquad F \subseteq S \times (\mathsf{val} \times \mathsf{mem})$$

Semantics in Compositional CompCert

Transition systems are extended to $L = \langle S, \to, I, X, Y, F \rangle$.
The states S and internal steps $\to$ are as before.

Initial and final states incorporate a *question* and an *answer* for the incoming call:

$$I \subseteq (\text{ident} \times \text{val}^* \times \text{mem}) \times S \qquad F \subseteq S \times (\text{val} \times \text{mem})$$

In addition, some states may perform outgoing calls:

$$X \subseteq S \times (\text{ident} \times \text{val}^* \times \text{mem}) \qquad Y^s \subseteq (\text{val} \times \text{mem}) \times S$$

Semantics in Compositional CompCert (example)

rb.c

```
/* Encapsulated state */
static int c1, c2;
static V buf[N];

/* Accessors */
int inc1() { int i = c1++; c1 %= N; return i; }
int inc2() { int i = c2++; c2 %= N; return i; }
V get(int i) { return buf[i]; }
void set(int i, V val) { buf[i] = val; }
```

Semantics in Compositional CompCert (example)

rb.c

```
/* Encapsulated state */
static int c1, c2;
static V buf[N];

/* Accessors */
int inc1() { int i = c1++; c1 %= N; return i; }
int inc2() { int i = c2++; c2 %= N; return i; }
V get(int i) { return buf[i]; }
void set(int i, V val) { buf[i] = val; }
```

An execution for the translation unit above may be:

$$\mathsf{inc}_1(\epsilon)@[c_1 \mapsto 3, \ldots] \; I \; s_0 \to^* s_n \; F \; 3@[c_1 \mapsto 4, \ldots]$$

Semantics in Compositional CompCert (example)

bq.c

```
/* Underlay signature */
extern int inc1(void);
extern int inc2(void);
extern V get(int i);
extern void set(int i, V val);

/* Layer implementation */
void enq(V val) { set(inc2(), val); }
V deq() { return get(inc1()); }
```

Semantics in Compositional CompCert (example)

bq.c

```
/* Underlay signature */
extern int inc1(void);
extern int inc2(void);
extern V get(int i);
extern void set(int i, V val);

/* Layer implementation */
void enq(V val) { set(inc2(), val); }
V deq() { return get(inc1()); }
```

An execution for the translation unit above may be:

$$\mathsf{enq}(v)@m \; I \; s_0 \to^* s_1 \; X \; \mathsf{inc}_2(\epsilon)@m_1 \leadsto 5@m_1' \; Y^{s_1} \; s_2 \cdots s_n \; F \; \mathsf{undef}@m'$$

Horizontal Composition $\oplus$

For $L_1, L_2 \in \mathrm{TS}$, their *semantic linking* $L_1 \oplus L_2 \in \mathrm{TS}$ is computed by letting them interact with each other.

Horizontal Composition $\oplus$

For $L_1, L_2 \in \mathsf{TS}$, their *semantic linking* $L_1 \oplus L_2 \in \mathsf{TS}$ is computed by letting them interact with each other.

Simulations now compose horizontally as well as vertically:

$$
\begin{tikzcd}
\bullet \arrow[r, bend left=50, "L_1", ""{name=A, below}] \arrow[r, bend right=50, "L_1'"', ""{name=B, above}] \arrow[Rightarrow, from=A, to=B, "\pi_1"] & \bullet \arrow[r, bend left=50, "L_2", ""{name=C, below}] \arrow[r, bend right=50, "L_2'"', ""{name=D, above}] \arrow[Rightarrow, from=C, to=D, "\pi_2"] & \bullet
\end{tikzcd}
\qquad
\frac{\pi_1 : L_1 \le L_1' \qquad \pi_2 : L_2 \le L_2'}{\pi_1 \oplus \pi_2 : L_1 \oplus L_2 \le L_1' \oplus L_2'}
\qquad
\begin{tikzcd}
\bullet \arrow[r, bend left=50, "L_1 \oplus L_2", ""{name=A, below}] \arrow[r, bend right=50, "L_1' \oplus L_2'"', ""{name=B, above}] \arrow[Rightarrow, from=A, to=B, "\pi_1 \oplus \pi_2"] & \bullet
\end{tikzcd}
$$

Horizontal Composition $\oplus$

For $L_1, L_2 \in \mathrm{TS}$, their *semantic linking* $L_1 \oplus L_2 \in \mathrm{TS}$ is computed by letting them interact with each other.

Simulations now compose horizontally as well as vertically:

$$
\begin{tikzcd}
\bullet \arrow[r, bend left, "L_1"] \arrow[r, bend right, "L_1'"'] \arrow[r, phantom, "\Downarrow \pi_1"] & \bullet \arrow[r, bend left, "L_2"] \arrow[r, bend right, "L_2'"'] \arrow[r, phantom, "\Downarrow \pi_2"] & \bullet
\end{tikzcd}
\qquad
\frac{\pi_1 : L_1 \le L_1' \qquad \pi_2 : L_2 \le L_2'}{\pi_1 \oplus \pi_2 : L_1 \oplus L_2 \le L_1' \oplus L_2'}
\qquad
\begin{tikzcd}
\bullet \arrow[r, bend left, "L_1 \oplus L_2"] \arrow[r, bend right, "L_1' \oplus L_2'"'] \arrow[r, phantom, "\Downarrow \pi_1 \oplus \pi_2"] & \bullet
\end{tikzcd}
$$

In other words we have a *monoidal category*:

Compositional CompCert		
Object	Dimension	Ops
Transition system	1	$\oplus$
Simulation	2	; $\oplus$

Compositional CompCert in Pasting Diagrams

$$
\begin{tikzcd}
L_1 \arrow[d, Rightarrow, "\phi"] \\
L_2 \arrow[d, Rightarrow, "\pi"] \\
L_3
\end{tikzcd}
$$

Compositional CompCert in Pasting Diagrams

$$\begin{tikzcd} L_1 \arrow[d, Rightarrow, "\phi;\pi"] \\ L_3 \end{tikzcd}$$

Compositional CompCert in Pasting Diagrams

$$
\begin{tikzcd}
\bullet \arrow[rr, dash, "L_{\mathsf{bq}}"] \arrow[d, equal] \arrow[drr, phantom, "\phi"] & & \bullet \arrow[d, equal] \\
\bullet \arrow[r, dash, "\mathsf{Clight(bq.c)}"] \arrow[d, equal] \arrow[dr, phantom, "\pi_{\mathsf{bq}}"] & \bullet \arrow[r, dash, "\mathsf{Clight(rb.c)}"] \arrow[d, equal] \arrow[dr, phantom, "\pi_{\mathsf{rb}}"] & \bullet \arrow[d, equal] \\
\bullet \arrow[r, dash, "\mathsf{Asm(bq.s)}"] & \bullet \arrow[r, dash, "\mathsf{Asm(rb.s)}"] & \bullet
\end{tikzcd}
$$

Compositional CompCert in Pasting Diagrams

$$
\begin{tikzcd}
\bullet \arrow[rr, dash, "L_{\mathsf{bq}}"] \arrow[d, equal] \arrow[drr, phantom, "\phi"] & & \bullet \arrow[d, equal] \\
\bullet \arrow[r, dash, "\mathsf{Clight}(\mathsf{bq.c})"] \arrow[d, equal] \arrow[drr, phantom, "\pi_{\mathsf{bq}} \oplus \pi_{\mathsf{rb}}"] & \bullet \arrow[r, dash, "\mathsf{Clight}(\mathsf{rb.c})"] & \bullet \arrow[d, equal] \\
\bullet \arrow[r, dash, "\mathsf{Asm}(\mathsf{bq.s})"] & \bullet \arrow[r, dash, "\mathsf{Asm}(\mathsf{rb.s})"] & \bullet
\end{tikzcd}
$$

Compositional CompCert in Pasting Diagrams

$$
\begin{tikzcd}
\bullet \arrow[rr, dash, "L_{\mathsf{bq}}"] \arrow[d, equal] \arrow[drr, phantom, "\phi \,;\, (\pi_{\mathsf{bq}} \oplus \pi_{\mathsf{rb}})"] & & \bullet \arrow[d, equal] \\
\bullet \arrow[r, dash, "\mathsf{Asm}(\mathsf{bq.s})"] & \bullet \arrow[r, dash, "\mathsf{Asm}(\mathsf{rb.s})"] & \bullet
\end{tikzcd}
$$

Compositional CompCert in Pasting Diagrams

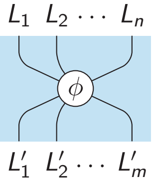
$$
\begin{tikzcd}
\bullet \arrow[r, no head, "C"] \arrow[d, equal] & \bullet \arrow[rr, no head, "L_{\mathsf{bq}}"] \arrow[d, equal] \arrow[drr, phantom, "\phi\, ;\, (\pi_{\mathsf{bq}} \oplus \pi_{\mathsf{rb}})"] & & \bullet \arrow[d, equal] \\
\bullet \arrow[r, no head, "C"] & \bullet \arrow[r, no head, "\mathrm{Asm}(\mathsf{bq.s})"] & \bullet \arrow[r, no head, "\mathrm{Asm}(\mathsf{rb.s})"] & \bullet
\end{tikzcd}
$$

Compositional CompCert in Pasting Diagrams

$$
\begin{tikzcd}
\bullet \arrow[r, dash, "C"] \arrow[d, equal] \arrow[drrr, phantom, "\mathrm{id}_C \oplus (\phi \,;\, (\pi_{\mathsf{bq}} \oplus \pi_{\mathsf{rb}}))"] & \bullet \arrow[rr, dash, "L_{\mathsf{bq}}"] & & \bullet \arrow[d, equal] \\
\bullet \arrow[r, dash, "C"] & \bullet \arrow[r, dash, "\mathsf{Asm}(\mathsf{bq.s})"] & \bullet \arrow[r, dash, "\mathsf{Asm}(\mathsf{rb.s})"] & \bullet
\end{tikzcd}
$$

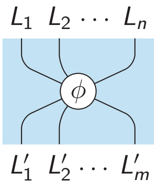
Compositional CompCert in String Diagrams

A simulation $\phi : L_1 \oplus \cdots \oplus L_n \leq L'_1 \oplus \cdots \oplus L'_m$ can be depicted as:

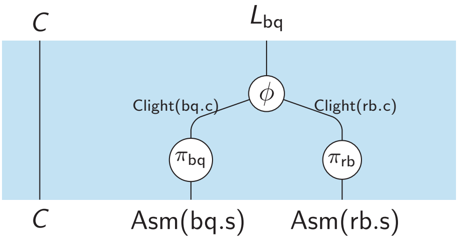


Compositional CompCert in String Diagrams

A simulation $\phi : L_1 \oplus \cdots \oplus L_n \leq L'_1 \oplus \cdots \oplus L'_m$ can be depicted as:



Our previous example is represented as:



Issues with Compositional CompCert

Compositional CompCert was a remarkable achievement
but it underscores the difficulty of using compositional semantics.

- Previously internal details become observable,
  so simulation relations are much more constrained.
- As a result, many proofs had to be redone
  and became much more complex.

Issues with Compositional CompCert

Compositional CompCert was a remarkable achievement
but it underscores the difficulty of using compositional semantics.

- Previously internal details become observable,
  so simulation relations are much more constrained.
- As a result, many proofs had to be redone
  and became much more complex.

CompCertO addresses this by dealing with each language and pass *on its own terms*:

- Languages can use their own questions and answers
- Calling conventions are modeled explicitly

Outline

Horizontal Morphisms

Recall the transition systems $L = \langle S, \rightarrow, I, X, Y, F \rangle \in \mathsf{TS}$ used in CompComp:

$$\begin{aligned} I &\subseteq (\mathsf{ident} \times \mathsf{val}^* \times \mathsf{mem}) \times S & \qquad F &\subseteq S \times (\mathsf{val} \times \mathsf{mem}) \\ X &\subseteq S \times (\mathsf{ident} \times \mathsf{val}^* \times \mathsf{mem}) & \qquad Y^s &\subseteq (\mathsf{val} \times \mathsf{mem}) \times S \end{aligned}$$

The questions and answers are hardcoded to correspond to C calls.

Horizontal Morphisms

In CompCertO, we generalize $L = \langle S, \to, I, X, Y, F \rangle : A \twoheadrightarrow B$ to:

$$I \subseteq B^\circ \times S \qquad F \subseteq S \times B^\bullet$$
$$X \subseteq S \times A^\circ \qquad Y^s \subseteq A^\bullet \times S$$

for the *language interfaces* $A = \langle A^\circ, A^\bullet \rangle$ and $B = \langle B^\circ, B^\bullet \rangle$.

Horizontal Morphisms

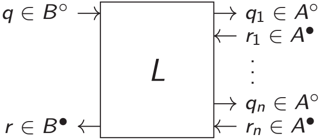
In CompCertO, we generalize $L = \langle S, \to, I, X, Y, F \rangle : A \twoheadrightarrow B$ to:

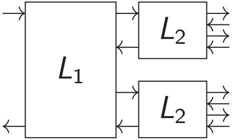
$$I \subseteq B^\circ \times S \qquad F \subseteq S \times B^\bullet$$
$$X \subseteq S \times A^\circ \qquad Y^s \subseteq A^\bullet \times S$$

for the *language interfaces* $A = \langle A^\circ, A^\bullet \rangle$ and $B = \langle B^\circ, B^\bullet \rangle$.

For our purposes, we use the following notion of composition $L_1 \odot L_2$:

$$B \twoheadleftarrow A \qquad\qquad C \twoheadleftarrow B \twoheadleftarrow A$$





Vertical Morphisms

To connect source and target languages such as

$$\mathrm{Clight}(p) : \mathcal{C} \twoheadrightarrow \mathcal{C} \qquad \mathit{vs} \qquad \mathrm{Asm}(p') : \mathcal{A} \twoheadrightarrow \mathcal{A}\,,$$

CompCertO introduces *simulation conventions* $\mathbb{R} : \mathcal{C} \leftrightarrow \mathcal{A}$, which parameterize the relationship between source- and target-level interactions:

$$\begin{tikzcd}
\mathcal{C}^\circ \ni & q_1 \arrow[r, dash, "I_1"] \arrow[d, dash, "\mathbb{R}^\circ"'] & s_1 \arrow[d, dash, dashed, "\rho"] \\
\mathcal{A}^\circ \ni & q_2 \arrow[r, dash, dashed, "I_2"'] & s_2
\end{tikzcd}
\qquad
\begin{tikzcd}
s_1 \arrow[r] \arrow[d, dash, "\rho"'] & s_1' \arrow[d, dash, dashed, "\rho"] \\
s_2 \arrow[r, dashed] & s_2'
\end{tikzcd}
\qquad
\begin{tikzcd}
s_1 \arrow[r, dash, "F_1"] \arrow[d, dash, "\rho"'] & r_1 \arrow[d, dash, dashed, "\mathbb{R}^\bullet"] & \in \mathcal{C}^\bullet \\
s_2 \arrow[r, dash, dashed, "F_2"] & r_2 & \in \mathcal{A}^\bullet
\end{tikzcd}$$

Simulation conventions compose vertically as $\mathbb{R}_1 \,;\, \mathbb{R}_2$.

Simulations

Simulations now have two-dimentional types and compose in both directions:

$$\pi : L_1 \leq_{\mathrm{R}_A \twoheadrightarrow \mathrm{R}_B} L_2$$

$$\begin{tikzcd} A_1 \arrow[r, two heads, "L_1"] \arrow[d, leftrightarrow, "\mathrm{R}_A"'] & B_1 \arrow[d, leftrightarrow, "\mathrm{R}_B"] \\ A_2 \arrow[r, two heads, "L_2"'] & B_2 \end{tikzcd}$$

Simulations

Simulations now have two-dimentional types and compose in both directions:

$$\pi : L_1 \leq_{\mathrm{R}_A \twoheadrightarrow \mathrm{R}_B} L_2$$

$$\begin{tikzcd}
A_1 \arrow[r, two heads, "L_1"] \arrow[d, leftrightarrow, "\mathrm{R}_A"'] & B_1 \arrow[d, leftrightarrow, "\mathrm{R}_B"] \\
A_2 \arrow[r, two heads, "L_2"'] & B_2
\end{tikzcd}$$

CompCertO			
Object		Dimension	Ops
Language interface	A	0	
Transition system	$L : A \twoheadrightarrow B$	1	$\odot$
Simulation convention	$\mathbb{R} : A \leftrightarrow B$	1	;
Simulation	$\rho : L_1 \leq_{\mathbb{R} \twoheadrightarrow \mathbb{S}} L_2$	2	; $\odot$

CompCertO in Pasting Diagrams

$$\begin{tikzcd} L_1 \arrow[d, Rightarrow, "\phi"] \\ L_2 \arrow[d, Rightarrow, "\pi"] \\ L_3 \end{tikzcd}$$

CompCertO in Pasting Diagrams

$$
\begin{tikzcd}
\bullet \arrow[rr, no head, "L_{\mathsf{bq}}"] \arrow[d, equal] \arrow[drr, phantom, "\phi"] & & \bullet \arrow[d, equal] \\
\bullet \arrow[r, no head, "\mathsf{Clight(bq.c)}"] \arrow[d, equal] \arrow[dr, phantom, "\pi_{\mathsf{bq}}"] & \bullet \arrow[d, equal] \arrow[dr, phantom, "\pi_{\mathsf{rb}}"] & \bullet \arrow[l, two heads, "\mathsf{Clight(rb.c)}"'] \arrow[d, equal] \\
\bullet & \bullet \arrow[l, two heads, "\mathsf{Asm(bq.s)}"'] & \bullet \arrow[l, two heads, "\mathsf{Asm(rb.s)}"']
\end{tikzcd}
$$

CompCertO in Pasting Diagrams

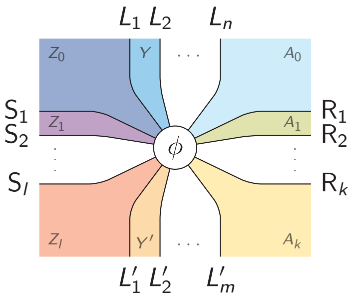
$$
\begin{tikzcd}
\mathcal{C} \arrow[d, equal] \arrow[drr, phantom, "\phi"] & & \top \arrow[ll, two heads, "L_{\mathsf{bq}}"'] \arrow[d, leftrightarrow, "\varnothing"] \\
\mathcal{C} \arrow[d, leftrightarrow, "\mathbb{C}"] \arrow[dr, phantom, "\pi_{\mathsf{bq}}"] & \mathcal{C} \arrow[l, two heads, "\mathsf{Clight(bq.c)}"'] \arrow[d, leftrightarrow, "\mathbb{C}"] \arrow[dr, phantom, "\pi_{\mathsf{rb}}"] & \mathcal{C} \arrow[l, two heads, "\mathsf{Clight(rb.c)}"'] \arrow[d, leftrightarrow, "\mathbb{C}"] \\
\mathcal{A} & \mathcal{A} \arrow[l, two heads, "\mathsf{Asm(bq.s)}"'] & \mathcal{A} \arrow[l, two heads, "\mathsf{Asm(rb.s)}"']
\end{tikzcd}
$$

CompCertO in String Diagrams

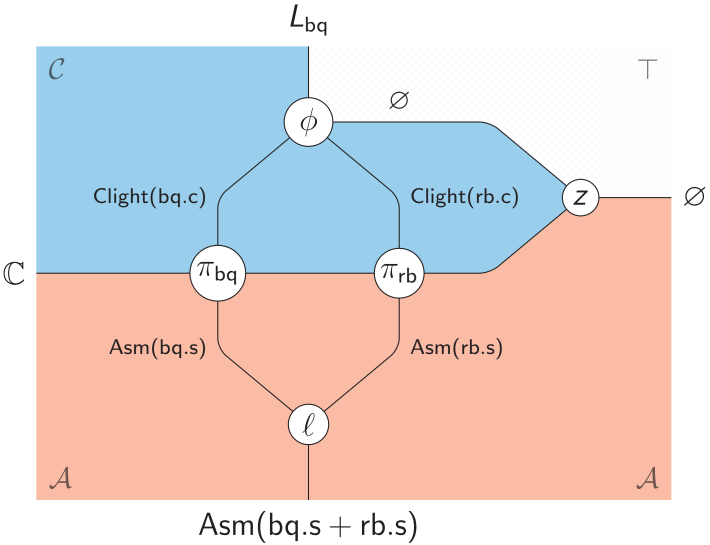
A simulation

$$\phi : L_1 \odot \cdots \odot L_n \leq_{\mathbb{R}_1;\cdots;\mathbb{R}_k \twoheadrightarrow \mathbb{S}_1;\cdots;\mathbb{S}_l} L'_1 \odot \cdots \odot L'_m$$

can be represented as:



CompCertO in String Diagrams (example)



Outline

Abstract Specifications

What would be a good specification for our example:

rb.c

```
/* Encapsulated state */
static int c1, c2;
static V buf[N];

/* Accessors */
int inc1() { int i = c1++; c1 %= N; return i; }
int inc2() { int i = c2++; c2 %= N; return i; }
V get(int i) { return buf[i]; }
void set(int i, V val) { buf[i] = val; }
```

bq.c

```
/* Underlay signature */
extern int inc1(void);
extern int inc2(void);
extern V get(int i);
extern void set(int i, V val);

/* Layer implementation */
void enq(V val) { set(inc2(), val); }
V deq() { return get(inc1()); }
```

As a user, we would prefer not to deal with low-level details about the memory, and rely instead on an abstract description:

$$\begin{array}{lll} \Gamma_{\mathsf{bq}} \vDash \mathsf{enq}(v)@\vec{q} \rightarrowtail *@\vec{q}v \\ \Gamma_{\mathsf{bq}} \vDash \mathsf{deq}()@v\vec{q} \rightarrowtail v@\vec{q} \end{array} \quad \text{where} \quad \vec{q} \in D_{\mathsf{bq}} := \mathsf{val}^*$$

Abstract Specifications

Likewise, to prove the implementation correct, we may want to rely on

$$\begin{aligned}
\Gamma_{\text{rb}} &\models \mathsf{get}(i)@(b, c_1, c_2) \rightarrowtail b_i@(b, c_1, c_2) \\
\Gamma_{\text{rb}} &\models \mathsf{set}(i, v)@(b, c_1, c_2) \rightarrowtail *@(b[i := v], c_1, c_2) \\
\Gamma_{\text{rb}} &\models \mathsf{inc1}()@(b, c_1, c_2) \rightarrowtail c_1@(b, c_1{+}1, c_2) \\
\Gamma_{\text{rb}} &\models \mathsf{inc2}()@(b, c_1, c_2) \rightarrowtail c_2@(b, c_1, c_2{+}1)
\end{aligned}$$

which specifies rb.c in terms of $D_{\text{rb}} := V^N \times \mathbb{N} \times \mathbb{N}$

Abstract Specifications

Likewise, to prove the implementation correct, we may want to rely on

$$
\begin{aligned}
\Gamma_{\mathrm{rb}} &\vDash \mathrm{get}(i)@(b, c_1, c_2) \rightarrowtail b_i@(b, c_1, c_2) \\
\Gamma_{\mathrm{rb}} &\vDash \mathrm{set}(i, v)@(b, c_1, c_2) \rightarrowtail *@(b[i := v], c_1, c_2) \\
\Gamma_{\mathrm{rb}} &\vDash \mathrm{inc1}()@(b, c_1, c_2) \rightarrowtail c_1@(b, c_1{+}1, c_2) \\
\Gamma_{\mathrm{rb}} &\vDash \mathrm{inc2}()@(b, c_1, c_2) \rightarrowtail c_2@(b, c_1, c_2{+}1)
\end{aligned}
$$

which specifies rb.c in terms of $D_{\mathrm{rb}} := V^N \times \mathbb{N} \times \mathbb{N}$

CompCertO's language interfaces and simulation conventions can help us do this!
But it requires a good way to deal with abstract state.

Adjoining state to language interfaces

Consider the following construction on language interfaces:

$$A @ U := \langle A^\circ \times U,\ A^\bullet \times U \rangle$$

That is, we extend A with a state component taken in the set U.

Adjoining state to language interfaces

Consider the following construction on language interfaces:

$$A \mathbin{@} U := \langle A^\circ \times U,\ A^\bullet \times U \rangle$$

That is, we extend A with a state component taken in the set U.

The language interface used for C semantics can be described as:

$$\mathsf{Clight}(M) : \mathcal{C} \mathbin{@} \mathsf{mem} \twoheadrightarrow \mathcal{C} \mathbin{@} \mathsf{mem} \qquad \text{where} \quad \mathcal{C} = \langle \mathsf{ident} \times \mathsf{val}^*,\ \mathsf{val} \rangle$$

Adjoining state to language interfaces

Consider the following construction on language interfaces:

$$A @ U := \langle A^\circ \times U,\ A^\bullet \times U \rangle$$

That is, we extend A with a state component taken in the set U.

The language interface used for C semantics can be described as:

$$\mathrm{Clight}(M) : \mathcal{C} @ \mathrm{mem} \twoheadrightarrow \mathcal{C} @ \mathrm{mem} \qquad \text{where} \quad \mathcal{C} = \langle \mathrm{ident} \times \mathrm{val}^*, \mathrm{val} \rangle$$

The abstract specifications are typed as:

$$\begin{array}{l} \Gamma_{\mathrm{bq}} : \top \twoheadrightarrow \mathcal{C} @ D_{\mathrm{bq}} \\ \Gamma_{\mathrm{rb}} : \top \twoheadrightarrow \mathcal{C} @ D_{\mathrm{rb}} \end{array} \qquad \text{where} \quad \top = \langle \varnothing, \varnothing \rangle$$

But how can we interface client code with abstract specifications?

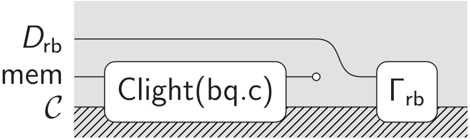
Spatial compositionality

We turn @ into yet another dimension of compositionality:

- Transition systems combine with a *lens* $f : U \leftrightarrows V$
- Simulation conventions easily combine with a relation $R \subseteq U \times V$
- Simulations can be extended in a similar way.

As before, we can use string diagrams to combine two kinds of composition. For example, to interface bq.c with Γ_{rb}:

$$L_{\mathrm{bq}} := (\mathrm{Clight}(\mathrm{bq.c}) \mathbin{@} D_{\mathrm{rb}}) \odot (\mathcal{C} \mathbin{@} \langle\mathrm{mem}] \mathbin{@} D_{\mathrm{rb}}) \odot \Gamma_{\mathrm{rb}} =$$



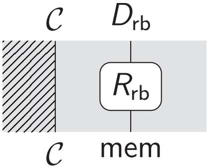
$\top$,

where $\langle\mathrm{mem}]$ is a trivial lens which "bounces" the memory state unchanged.

Concretizing state

Consider $\Gamma_{\mathrm{rb}} : \top \twoheadrightarrow \mathcal{C} @ D_{\mathrm{rb}}$ vs. its implementation rb.c in terms of $\mathcal{C} @ \mathrm{mem}$. The corresponding correctness property can be expressed as:

$$\Gamma_{\mathrm{rb}} : \varnothing \twoheadrightarrow \mathcal{C} @ R_{\mathrm{rb}}$$



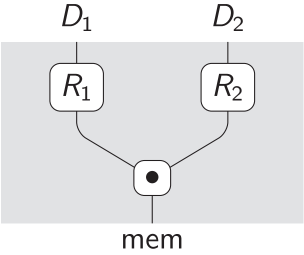
where $R_{\mathrm{rb}} \subseteq D_{\mathrm{rb}} \times \mathrm{mem}$ explains how abstract data is implemented.

Summary of the framework

	Role	Components	Notation	Compose			Diagrams	
				H	V	S	$\mathbb{H}$	$\mathbb{V}$
Active	Interface	Language interfaces	A, B, C			@		
	Behavior	Transition systems	$L : A \twoheadrightarrow B \in \mathsf{TS}$	$\odot$		@	$\odot$	@
	Abstraction	Simulation conventions	$\mathbb{R} : A \leftrightarrow B \in \mathsf{SC}$		;	@	@	;
	Refinement	Simulations	$\pi : L^\sharp \leq_{\mathbb{R} \twoheadrightarrow \mathbb{S}} L^\flat \in \mathsf{TSC}$	$\odot$	;	@	$\odot$	;
Passive	Interface	Sets	U, V			$\times$		
	Behavior	Lenses	$f : U \leftrightarrows V \in \mathsf{Lens}$	$\circ$		$\times$	$\circ$	$\times$
	Abstraction	Simulation relations	$R \subseteq U \times V \in \mathsf{Rel}$		;	$\times$	$\times$	;
	Refinement	Bisimulations	$\phi : f \equiv_{R \leftrightarrows S} g \in \mathsf{LSR}$	$\circ$	;	$\times$	$\circ$	;

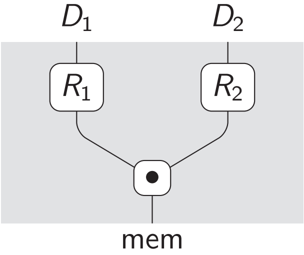
Compositional state vs. CompCert memory

To preserve compositionality when concretizing abstract state,
we can use separation algebra as a relation $\bullet \subseteq (\text{mem} \times \text{mem}) \times \text{mem}$:



Compositional state vs. CompCert memory

To preserve compositionality when concretizing abstract state,
we can use separation algebra as a relation $\bullet \subseteq (\text{mem} \times \text{mem}) \times \text{mem}$:

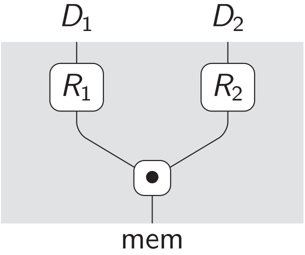


Properties of interest can be expressed as simulations, for example

- $\text{id}_{\text{mem}\times\text{mem}\times\text{mem}} \equiv_{(\bullet @ \text{mem});\bullet \twoheadrightarrow (\text{mem} @ \bullet);\bullet} \text{id}_{\text{mem}}$

Compositional state vs. CompCert memory

To preserve compositionality when concretizing abstract state,
we can use separation algebra as a relation $\bullet \subseteq (\mathrm{mem} \times \mathrm{mem}) \times \mathrm{mem}$:

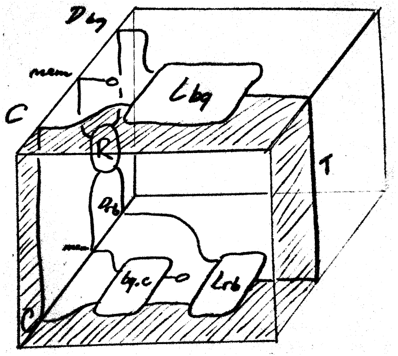


Properties of interest can be expressed as simulations, for example

- $\mathrm{id}_{\mathrm{mem}\times\mathrm{mem}\times\mathrm{mem}} \equiv_{(\bullet @ \mathrm{mem});\bullet \twoheadrightarrow (\mathrm{mem} @ \bullet);\bullet} \mathrm{id}_{\mathrm{mem}}$
- $\mathrm{Clight}(M) @ \mathrm{mem} \leq_{\mathcal{C}@\bullet \twoheadrightarrow \mathcal{C}@\bullet} \mathrm{Clight}(M)$

Three dimenstions of compositionality

Simulation string diagrams can also incorporate @ as *depth*.



Overall, many properties of interest can be put into the form of simulation "cubes" of the kind above, and proofs can be glued together geometrically.

Outline

Conclusion

This is joint work with Yu Zhang, Zhong Shao and Yuting Wang.
We are hoping to use thi framework for large-scale verification applications.

Some other things we did:

- Model of certified abstraction layers within this framework
- Encapsulated state
- ClightP language with private variables

Conclusion

This is joint work with Yu Zhang, Zhong Shao and Yuting Wang.
We are hoping to use thi framework for large-scale verification applications.

Some other things we did:

- Model of certified abstraction layers within this framework
- Encapsulated state
- ClightP language with private variables

Last thoughts:

- Compositional semantics for compilers are hard but interesting
- Semantics and higher category theory can be useful for engineering

Please feel free to request our paper draft from me!
(`jeremie.koenig@gmail.com`)