Refinement-Based Game Semantics for Certified Abstraction Layers

Jérémie Koenig   Zhong Shao

Yale University

Thirty-Fifth Annual ACM/IEEE Symposium
on Logic in Computer Science (LICS)
8–11 July 2020

Scaling up certified software

Certified software this past decade:

- C compiler (CompCert) and program logic (VST)
- Operating system kernel (CertiKOS), file system (FSCQ)
- Processor designs (Bluespec), ...

Scaling up certified software

Certified software this past decade:

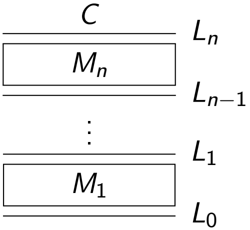
- C compiler (CompCert) and program logic (VST)
- Operating system kernel (CertiKOS), file system (FSCQ)
- Processor designs (Bluespec), . . .

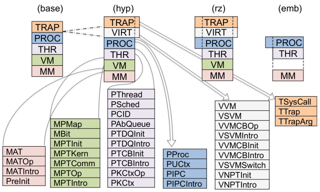
To scale up verification further, we need a compositional glue:

- Heterogenous: general-purpose model, embed various components
- Composition and abstraction: high-level algebraic structures

Case study: CertiKOS

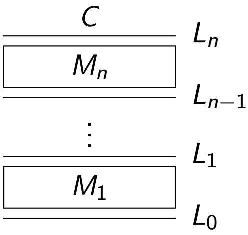
Software systems use abstraction layers. In CertiKOS:

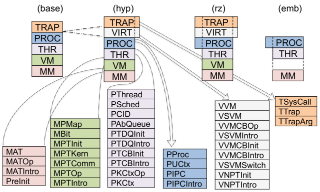




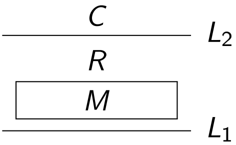
Case study: CertiKOS

Software systems use abstraction layers. In CertiKOS:





Our verification effort uses *certified abstraction layers*:



$$\boxed{L_1 \vdash M : L_2}$$

$$\forall C \cdot [\![C]\!]_{L_2} \leq_R [\![C \oplus M]\!]_{L_1}$$

Great research not used in large-scale verification

Good news: There is lot of research that we can draw from!

Bad news: Few applications to large-scale verification.

Semantics research	Typical verification project
Game semantics Linear logic	Transition systems
Refinement calculus	Simulations
Logical relations	Hoare logic
Algebraic effects	Closed systems

Why this gap?

Challenges:

- Sophisticated models hard to mechanize in proof assistants
- Not clear how these techniques can work together

For example:

- Game semantics: not much emphasis on refinement
- Refinement calculus: imperative programming and specification

Contributions

We combine various paradigms to introduce:

Refinement-Based Game Semantics

Contributions

We combine various paradigms to introduce:

Refinement-Based Game Semantics

Our models provide:

- **Compositionality:** categories with symmetric monoidal structures
- **Refinement:** uniform treatment of programs and specifications
- **Dual nondeterminism:** for expressivity and data abstraction

Contributions

We combine various paradigms to introduce:

Refinement-Based Game Semantics

Our models provide:

- **Compositionality:** categories with symmetric monoidal structures
- **Refinement:** uniform treatment of programs and specifications
- **Dual nondeterminism:** for expressivity and data abstraction

Key insights:

- Reinterpret strategies as inherently nondeterministic
- Upgrade to dual nondeterminism and lift all restrictions

Section 1

Dual nondeterminism and refinement

Refinement and nondeterminism

Stepwise refinement

Key idea: uniform treatment of programs and specifications

$$\boxed{C_1 \sqsubseteq C_2}$$

$$P\{C\}Q \Leftrightarrow \langle P|Q\rangle \sqsubseteq C$$

$$S \sqsubseteq C_1 \sqsubseteq \cdots \sqsubseteq C_n$$

Refinement and nondeterminism

Stepwise refinement

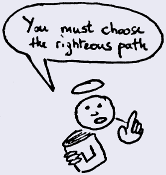
Key idea: uniform treatment of programs and specifications

$$\boxed{C_1 \sqsubseteq C_2}$$

$$P\{C\}Q \Leftrightarrow \langle P|Q\rangle \sqsubseteq C$$

$$S \sqsubseteq C_1 \sqsubseteq \cdots \sqsubseteq C_n$$

Nondeterminism in specifications



$$\boxed{S_1 \sqcup S_2 \sqsubseteq C}$$

$$\frac{S_1 \sqsubseteq C \qquad S_2 \sqsubseteq C}{S_1 \sqcup S_2 \sqsubseteq C}$$

Refinement and nondeterminism

Stepwise refinement

Key idea: uniform treatment of programs and specifications

$$\boxed{C_1 \sqsubseteq C_2}$$

$$P\{C\}Q \Leftrightarrow \langle P|Q\rangle \sqsubseteq C$$

$$S \sqsubseteq C_1 \sqsubseteq \cdots \sqsubseteq C_n$$

Nondeterminism in specifications

$$\boxed{S_1 \sqcap S_2 \sqsubseteq C}$$

$$\frac{S_1 \sqsubseteq C}{S_1 \sqcap S_2 \sqsubseteq C} \qquad \frac{S_2 \sqsubseteq C}{S_1 \sqcap S_2 \sqsubseteq C}$$

Dual nondeterminism and distributive lattices

$\sqsubseteq$, $\sqcup$, $\sqcap$ work together as a *completely distributive lattice*:

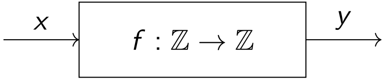
- Associativity of $\sqcup$, $\sqcap$: insensitive to branching
- Complete distributivity:

$$\bigsqcup_{i \in I} \bigsqcap_{j \in J_i} x_{i,j} = \bigsqcap_{f \in \prod_{i \in I} J_i} \bigsqcup_{i \in I} x_{i,f(i)}$$

  Angelic and demonic choice also commute with each other
- Refinement increases $\sqcup$, decreases $\sqcap$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:

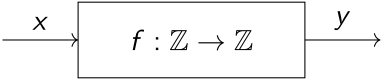


The specification $x \mapsto y$ means:

When the input is x*, the output must be* y*.*

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

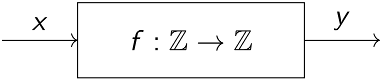
When the input is x, the output must be y.

The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

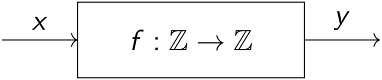
When the input is x, the output must be y.

The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

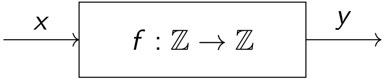
When the input is x, the output must be y.

The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2 \;\sqsubseteq\; \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x)$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

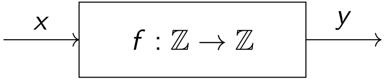
When the input is x, the output must be y.

The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2 \;\sqsubseteq\; \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x) \;=\; f$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

When the input is x, the output must be y.

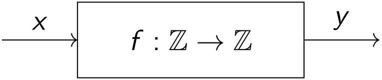
The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2 \;\sqsubseteq\; \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x) \;=\; f$$

$$\bigsqcap_{x \in \mathbb{Z}} (x \mapsto x + 1)$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

When the input is x, the output must be y.

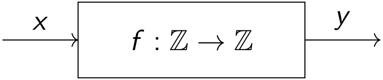
The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2 \;\sqsubseteq\; \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x) \;=\; f$$

$$\bigsqcap_{x \in \mathbb{Z}} (x \mapsto x + 1) \;\sqsubseteq\; 0 \mapsto 1 \sqcap 1 \mapsto 2$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

When the input is x, the output must be y.

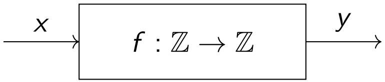
The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2 \;\sqsubseteq\; \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x) \;=\; f$$

$$\bigsqcap_{x \in \mathbb{Z}} (x \mapsto x + 1) \;\sqsubseteq\; 0 \mapsto 1 \sqcap 1 \mapsto 2 \;\sqsubseteq\; 1 \mapsto 2$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

When the input is x, the output must be y.

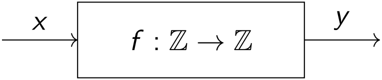
The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \ \sqsubseteq \ 0 \mapsto 0 \sqcup 1 \mapsto 2 \ \sqsubseteq \ \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x) \ = \ f$$

$$\bigsqcap_{x \in \mathbb{Z}} (x \mapsto x + 1) \ \sqsubseteq \ 0 \mapsto 1 \sqcap 1 \mapsto 2 \ \sqsubseteq \ 1 \mapsto 2 \ \sqsubseteq \ f$$

Example: nondeterministic functions

A function $f : \mathbb{Z} \to \mathbb{Z}$ can be seen as a simple system:



The specification $x \mapsto y$ means:

When the input is x, the output must be y.

The function $f(x) := 2x$ statisfies the specifications:

$$0 \mapsto 0 \;\sqsubseteq\; 0 \mapsto 0 \sqcup 1 \mapsto 2 \;\sqsubseteq\; \bigsqcup_{x \in \mathbb{Z}} (x \mapsto 2x) \;=\; f$$

$$\bigsqcap_{x \in \mathbb{Z}} (x \mapsto x + 1) \;\sqsubseteq\; 0 \mapsto 1 \sqcap 1 \mapsto 2 \;\sqsubseteq\; 1 \mapsto 2 \;\sqsubseteq\; f$$

$$\bigsqcup_{x \text{ odd}} \; \bigsqcap_{y \text{ even}} (x \mapsto y) \;\sqsubseteq\; f$$

Dual nondetermimism and data abstraction

Consider integers $x \in \mathbb{Z}$ as pairs of naturals $n = (n_1, n_2) \in \mathbb{N}^2$:

$$x \; R \; n \;\Leftrightarrow\; x = n_1 - n_2$$

Dual nondetermimism and data abstraction

Consider integers $x \in \mathbb{Z}$ as pairs of naturals $n = (n_1, n_2) \in \mathbb{N}^2$:

$$x \, R \, n \Leftrightarrow x = n_1 - n_2$$

Then $f : \mathbb{Z} \to \mathbb{Z}$ is implemented by $g : \mathbb{N}^2 \to \mathbb{N}^2$ when:

$$(\forall) \qquad \begin{array}{ccc} x & \xrightarrow{\;f\;} & f(x) \\ R \Big| & & \vdots \, R \\ n & \xrightarrow[\;g\;]{} & g(n) \end{array} \qquad (\exists)$$

Dual nondetermimism and data abstraction

Consider integers $x \in \mathbb{Z}$ as pairs of naturals $n = (n_1, n_2) \in \mathbb{N}^2$:

$$x \; R \; n \;\Leftrightarrow\; x = n_1 - n_2$$

Then $f : \mathbb{Z} \to \mathbb{Z}$ is implemented by $g : \mathbb{N}^2 \to \mathbb{N}^2$ when:

$$(\forall) \qquad \begin{array}{ccc} x & \xrightarrow{\;f\;} & f(x) \\ R \,\Big| & & \Big\vdots\, R \\ n & \xrightarrow[\;g\;]{} & g(n) \end{array} \qquad (\exists)$$

With dual nondeterminism:

$$R^*(f) \sqsubseteq g \qquad\qquad R^*(f) := \bigsqcup_{n \in \mathbb{N}^2} \; \bigsqcup_{x \, R \, n} \; \bigsqcap_{m \, R^{-1} \, f(x)} (n \mapsto m)$$

Consider integers $x \in \mathbb{Z}$ as pairs of naturals $n = (n_1, n_2) \in \mathbb{N}^2$:

$$x \, R \, n \;\Leftrightarrow\; x = n_1 - n_2$$

Then $f : \mathbb{Z} \to \mathbb{Z}$ is implemented by $g : \mathbb{N}^2 \to \mathbb{N}^2$ when:

$$(\forall) \qquad \begin{array}{ccc} x & \xrightarrow{\;f\;} & f(x) \\ R \Big| & & \vdots R \\ n & \xrightarrow[\;g\;]{} & g(n) \end{array} \qquad (\exists)$$

With dual nondeterminism:

$$f \sqsubseteq R_*(g) \qquad\qquad R_*(g) := \bigsqcup_{x \in \mathbb{Z}} \; \bigsqcap_{n \, R^{-1} \, x} \; \bigsqcup_{y \, R \, g(n)} (x \mapsto y)$$

Nondeterminism in game semantics

Game semantics describes strategies with sets of plays:

$$\{0 \mapsto 0,\ 1 \mapsto 1,\ 1 \mapsto -1\}$$

Nondeterminism in game semantics

Game semantics describes strategies with sets of plays:

$$\{0 \mapsto 0,\ 1 \mapsto 1,\ 1 \mapsto -1\}$$

We can interpret the nondeterminism above as:

$$0 \mapsto 0 \ \sqcup\ (1 \mapsto 1 \ \sqcap\ 1 \mapsto -1)$$

Nondeterminism in game semantics

Game semantics describes strategies with sets of plays:

$$\{0 \mapsto 0,\ 1 \mapsto 1,\ 1 \mapsto -1\}$$

We can interpret the nondeterminism above as:

$$0 \mapsto 0 \ \sqcup\ (1 \mapsto 1 \ \sqcap\ 1 \mapsto -1)$$

However, the resulting refinement ordering is complicated to describe:

$$\begin{aligned} \{0 \mapsto 0\} &\sqsubseteq \{0 \mapsto 0,\ 1 \mapsto 1,\ 1 \mapsto -1\} \\ \{0 \mapsto 0,\ 1 \mapsto 1,\ 1 \mapsto -1\} &\sqsubseteq \{0 \mapsto 0,\ 1 \mapsto 1\} \end{aligned}$$

Dual nondeterminism and strategy specifications

Instead, we embrace unrestricted dual nondeterminism:

- Single play: "if environment does x then system does y"

Dual nondeterminism and strategy specifications

Instead, we embrace unrestricted dual nondeterminism:

- Single play: “if environment does x then system does y”
- Strategy: range over environment choices (angelic)
  Set of plays ordered by inclusion ($\subseteq$)

Dual nondeterminism and strategy specifications

Instead, we embrace unrestricted dual nondeterminism:

- Single play: "if environment does x then system does y"
- Strategy: range over environment choices (angelic)
  Set of plays ordered by inclusion ($\subseteq$)
- Strategy specification: add system choices (demonic)
  Set of strategies ordered by containment ($\supseteq$)

Dual nondeterminism as an effect

The **FCD** monad extends any poset with dual nondeterminism.

Dual nondeterminism as an effect

The **FCD** monad extends any poset with dual nondeterminism.

Definition

$\mathbf{FCD}(A)$ is the *free completely distributive lattice* generated by A.
Every element $x \in \mathbf{FCD}(A)$ can be described as:

$$x = \bigsqcap_{i \in I} \bigsqcup_{j \in J_i} x_{ij} \qquad (x_{ij} \in A)$$

The monadic structure is:

$$a \leftarrow x \,;\, f(a) := \bigsqcap_{i \in I} \bigsqcup_{j \in J_i} f(x_{ij}) \qquad (x \in \mathbf{FCD}(A), f : A \to \mathbf{FCD}(B))$$

$$\eta(a) := \bigsqcap_{i \in \mathbb{1}} \bigsqcup_{j \in \mathbb{1}} a \qquad (a \in A)$$

Section 2

Refinement-based game semantics

First-order signatures as games

Definition (Signature)

$$E = \{m_1 : N_1, \ldots, m_j : N_j\}$$

Each $m_i : N_i \in E$ is a *question*, with $n_i \in N_i$ a corresponding *answer*.

First-order signatures as games

Definition (Signature)

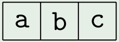
$$E = \{m_1 : N_1, \ldots, m_j : N_j\}$$

Each $m_i : N_i \in E$ is a *question*, with $n_i \in N_i$ a corresponding *answer*.

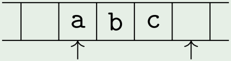
Example (Bounded queue)

We implement a queue using an array and two counters:

$$E_{\mathrm{q}} := \{\mathrm{enq}[v] : \mathbb{1},\ \mathrm{deq} : V \mid v \in V\}$$



$$E_{\mathrm{a}} := \{\mathrm{get}[i]{:}V,\ \mathrm{set}[i, v]{:}\mathbb{1},\ \mathrm{inc}_1{:}\mathbb{N},\ \mathrm{inc}_2{:}\mathbb{N} \mid i \in \mathbb{N}, v \in V\}$$



Individual interactions

Definition (Plays)

We use **odd-length** plays $s \in P_E(A)$ of the form:

$$s \sqsubseteq_{\text{odd}} \underline{m_1} n_1 \cdots \underline{m_j} n_j \underline{v} \qquad (m_i : N_i \in E, n_i \in N_i, v \in A)$$

Individual interactions

Definition (Plays)

We use **odd-length** plays $s \in P_E(A)$ of the form:

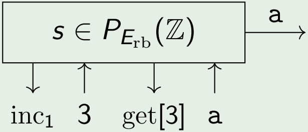
$$s \ \sqsubseteq_{\mathrm{odd}} \ \underline{m_1} n_1 \cdots \underline{m_j} n_j \underline{v} \qquad (m_i : N_i \in E, n_i \in N_i, v \in A)$$

Example (Dequeuing from an array)

The play:

$$s := \underline{\mathrm{inc}_1} \cdot 3 \cdot \underline{\mathrm{get}[3]} \cdot \mathtt{a} \cdot \underline{\mathtt{a}}$$

can be depicted as:



Interaction specifications

Definition (Interaction specifications)

For a signature E and a set A:

$$\mathcal{I}_E(A) := \mathbf{FCD}(P_E(A))$$

Interaction specifications

Definition (Interaction specifications)

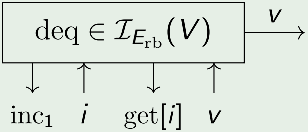
For a signature E and a set A:

$$\mathcal{I}_E(A) := \mathbf{FCD}(P_E(A))$$

Example (Dequeuing from an array)

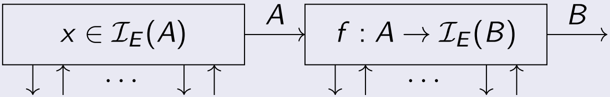
Implementing deq in terms of E_a:

$$\mathrm{deq} := \bigsqcup_{i \in \mathbb{N}} \bigsqcup_{v \in V} \underline{\mathrm{inc}_1} \cdot i \cdot \underline{\mathrm{get}[i]} \cdot v \cdot \underline{v}$$



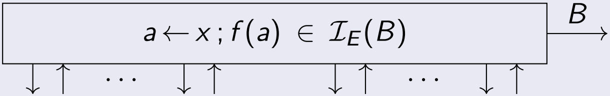
Sequential composition

Monadic structure



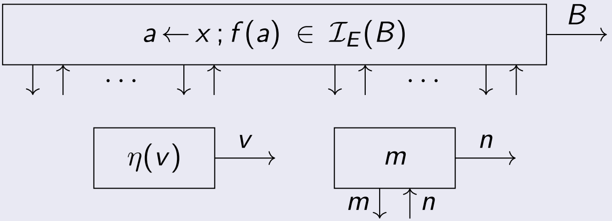
Sequential composition

Monadic structure



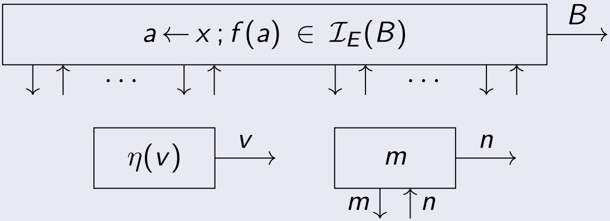
Sequential composition

Monadic structure



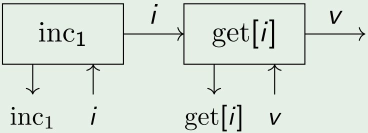
Sequential composition

Monadic structure



Example (Dequeuing from an array)

$$\mathrm{deq} := i \leftarrow \mathrm{inc}_1 \;;\; \mathrm{get}[i]$$

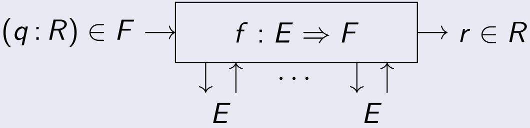


Two-sided strategies

Definition

A morphism $f : E \Rightarrow F$ is a family:

$$f \in \prod_{(q:R) \in F} \mathcal{I}_E(R)$$

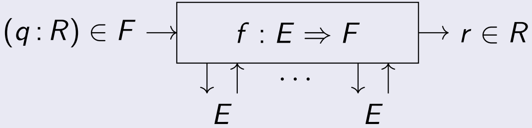


Two-sided strategies

Definition

A morphism $f : E \Rightarrow F$ is a family:

$$f \in \prod_{(q:R) \in F} \mathcal{I}_E(R)$$



Example (Queue implementation)

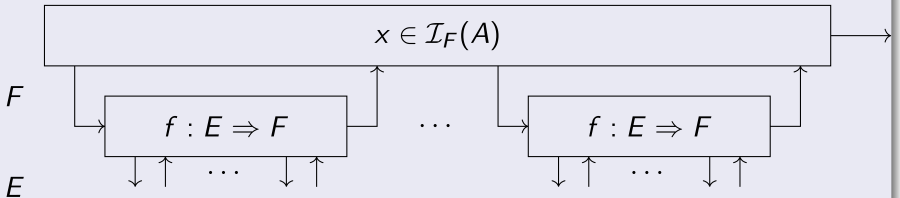
The morphism $M_{\mathrm{q}} : E_{\mathrm{a}} \Rightarrow E_{\mathrm{q}}$ is defined by:

$$\begin{aligned} \mathrm{enq}[v] &:= i \leftarrow \mathrm{inc}_2 \,;\, \mathrm{set}[i, v] \\ \mathrm{deq} &:= i \leftarrow \mathrm{inc}_1 \,;\, \mathrm{get}[i] \end{aligned}$$

Composition

Substitution

The substitution $x[f]$ has the shape:

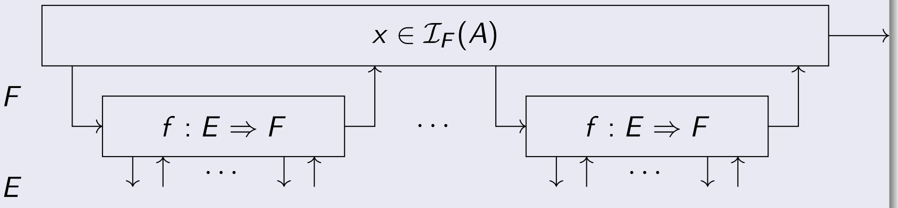


It can be used to define composition of morphisms.

Composition

Substitution

The substitution $x[f]$ has the shape:



It can be used to define composition of morphisms.

Example (Queue rotation)

$$\begin{aligned} \mathrm{rot} \in \mathcal{I}_{E_\mathrm{q}}(\mathbb{1}) \quad &:= \quad v \leftarrow \mathrm{deq} \, ; \mathrm{enq}[v] \\ \mathrm{rot}[M_\mathrm{q}] \in \mathcal{I}_{E_\mathrm{a}}(\mathbb{1}) \quad &:= \quad i \leftarrow \mathrm{inc}_1 \, ; v \leftarrow \mathrm{get}[i] \, ; j \leftarrow \mathrm{inc}_2 \, ; \mathrm{set}[j, v] \end{aligned}$$

State

Definition (Extending a signature with state)

We can annotate all calls and returns in E with a state $k \in S$:

$$E@S := \{m@k : N \times S \mid m{:}N \in E, k \in S\}$$

State

Definition (Extending a signature with state)

We can annotate all calls and returns in E with a state $k \in S$:

$$E@S := \{m@k : N \times S \mid m{:}N \in E, k \in S\}$$

Example (Queue layer interface)

$$S_\mathrm{q} := V^* \qquad L_\mathrm{q} : \varnothing \Rightarrow E_\mathrm{q}@S_\mathrm{q}$$

$$\mathrm{enq}[v]@\vec{q} := \eta(*@\vec{q}v)$$
$$\mathrm{deq}@\vec{q} := \bigsqcup_{v\vec{p}=\vec{q}} \eta(v@\vec{p})$$

$$\mathrm{rot}@S_\mathrm{q} : S_\mathrm{q} \rightarrow \mathcal{I}_{E_\mathrm{q}@S_\mathrm{q}}(\mathbb{1} \times S_\mathrm{q})$$

$$\mathrm{rot}@S_\mathrm{q}[L_\mathrm{q}] : S_\mathrm{q} \rightarrow \mathcal{I}_{\varnothing}(\mathbb{1} \times S_\mathrm{q})$$

State

Definition (Extending a signature with state)

We can annotate all calls and returns in E with a state $k \in S$:

$$E@S := \{m@k : N \times S \mid m{:}N \in E, k \in S\}$$

Example (Array layer interface)

$$S_\mathrm{a} := V^\mathbb{N} \times \mathbb{N} \times \mathbb{N} \qquad L_\mathrm{a} : \varnothing \Rightarrow E_\mathrm{a}@S_\mathrm{a}$$

$$\begin{aligned}
\mathrm{get}[i]@(t, c_1, c_2) &:= \eta(t_i@(t, c_1, c_2)) \\
\mathrm{set}[i, v]@(t, c_1, c_2) &:= \eta(*@(t[i \leftarrow v], c_1, c_2)) \\
\mathrm{inc}_1@(t, c_1, c_2) &:= \eta(c_1@(t, c_1 + 1, c_2)) \\
\mathrm{inc}_2@(t, c_1, c_2) &:= \eta(c_2@(t, c_1, c_2 + 1))
\end{aligned}$$

$$M_\mathrm{q}@S_\mathrm{a} : E_\mathrm{a}@S_\mathrm{a} \Rightarrow E_\mathrm{q}@S_\mathrm{a} \qquad M_\mathrm{q}@S_\mathrm{a} \circ L_\mathrm{a} : \varnothing \Rightarrow E_\mathrm{q}@S_\mathrm{a}$$

Data abstraction

Definition

A simulation relation $R \subseteq S_2 \times S_1$ can be encoded as a morphism:

$$R_E^* : E@S_2 \Rightarrow E@S_1 \qquad R_*^E : E@S_1 \Rightarrow E@S_2$$

$$R_E^* \circ L_2 \sqsubseteq L_1 \qquad \Leftrightarrow \qquad L_2 \sqsubseteq R_*^E \circ L_1$$

Data abstraction

Definition

A simulation relation $R \subseteq S_2 \times S_1$ can be encoded as a morphism:

$$R_E^* : E@S_2 \Rightarrow E@S_1 \qquad R_*^E : E@S_1 \Rightarrow E@S_2$$

$$R_E^* \circ L_2 \sqsubseteq L_1 \qquad \Leftrightarrow \qquad L_2 \sqsubseteq R_*^E \circ L_1$$

Example (Translating between array and queue states)

$$\vec{q}\ R\ (t, c_1, c_2) \quad \Leftrightarrow \quad c_1 \leq c_2 \wedge \vec{q} = t_{c_1} \cdots t_{c_2 - 1}$$

The layer interface $R_{E_q}^* \circ L_q : \varnothing \Rightarrow E_\mathrm{q}@S_\mathrm{a}$ becomes:

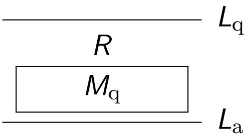
$$\mathrm{enq}[v]@(t, c_1, c_2) := \bigsqcup_{\vec{q} = t_{c_1} \cdots t_{c_2}} \ \bigsqcap_{(t', c_1', c_2') | \vec{q}v = t'_{c_1'} \cdots t'_{c_2'}} \eta(*@(t', c_1', c_2'))$$

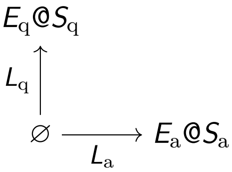
$$\mathrm{deq}@(t, c_1, c_2) := \bigsqcup_{v\vec{q} = t_{c_1} \cdots t_{c_2}} \ \bigsqcap_{(t', c_1', c_2') | \vec{q} = t'_{c_1'} \cdots t'_{c_2'}} \eta(v@(t', c_1', c_2'))$$

Certified abstraction layers

$L_{\mathrm{a}} \vdash M_{\mathrm{q}} : L_{\mathrm{q}}$

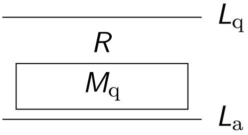
$R^*_{E_{\mathrm{q}}} \circ L_{\mathrm{q}} \sqsubseteq M_{\mathrm{q}} @ S_{\mathrm{a}} \circ L_{\mathrm{a}}$

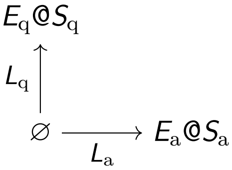




Certified abstraction layers

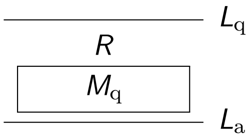
$$L_\mathrm{a} \vdash M_\mathrm{q} : L_\mathrm{q} \qquad\qquad R^*_{E_\mathrm{q}} \circ L_\mathrm{q} \sqsubseteq M_\mathrm{q} @ S_\mathrm{a} \circ L_\mathrm{a}$$

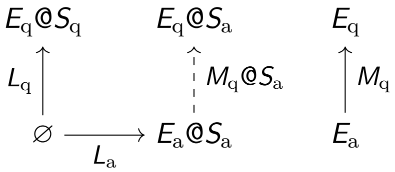




Certified abstraction layers

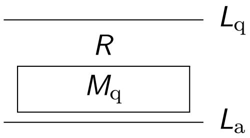
$$L_\mathrm{a} \vdash M_\mathrm{q} : L_\mathrm{q} \qquad\qquad R^*_{E_\mathrm{q}} \circ L_\mathrm{q} \sqsubseteq M_\mathrm{q} @ S_\mathrm{a} \circ L_\mathrm{a}$$

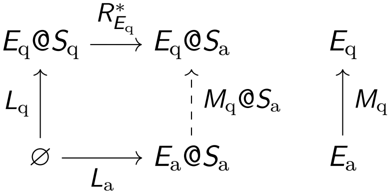




Certified abstraction layers

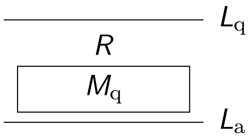
$$L_{\mathrm{a}} \vdash M_{\mathrm{q}} : L_{\mathrm{q}} \qquad\qquad R^*_{E_{\mathrm{q}}} \circ L_{\mathrm{q}} \sqsubseteq M_{\mathrm{q}} @ S_{\mathrm{a}} \circ L_{\mathrm{a}}$$

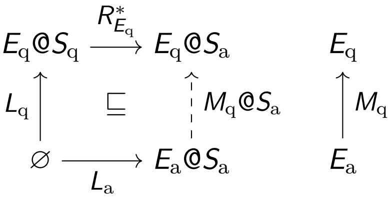




Certified abstraction layers

$$L_a \vdash M_q : L_q \qquad\qquad R^*_{E_q} \circ L_q \sqsubseteq M_q @ S_a \circ L_a$$





Section 3

Conclusion

Conclusion

Game semantics and dual nondeterminism go hand-in-hand:

- Angelic nondeterminism is already present in strategies
- Unrestricted dual nondeterminism completes the symmetry

Refinement-based game introduces several innovations:

- Combine game semantics and the refinement calculus
- Nondeterminism decoupled from the structure of plays
- Supports heterogenous components and data abstraction

Thank you!