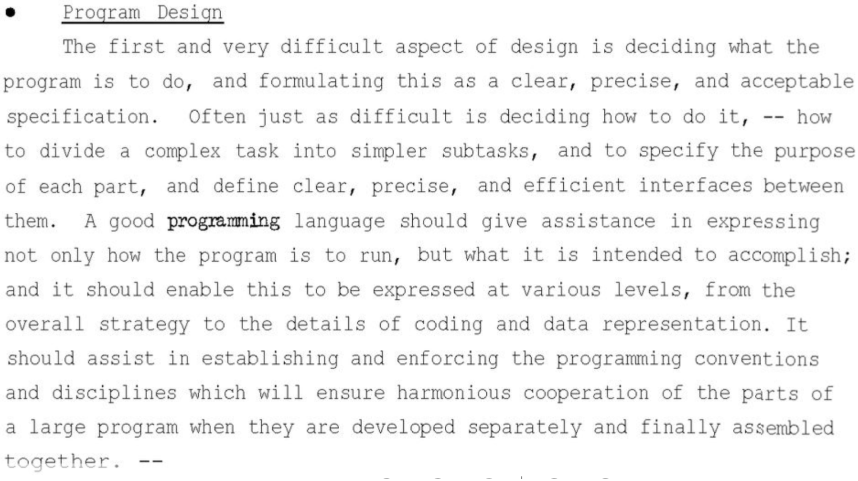
CS428 Lecture 9: Layered and Object-Based Game Semantics

Zhong Shao

Joint work with Arthur Oliveira Vale, Paul-André Melliès, Jérémie Koenig, and Leo Stefanesco

February 15, 2024

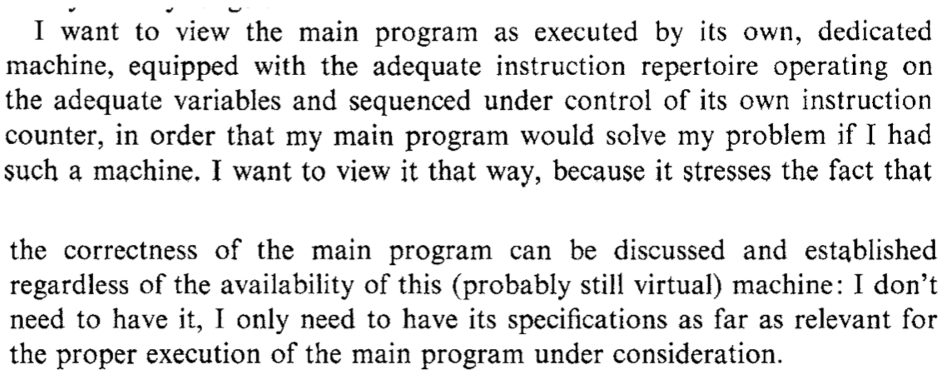
Hints on Programming Language Design (Hoare 1973)



- <u>Program Design</u>

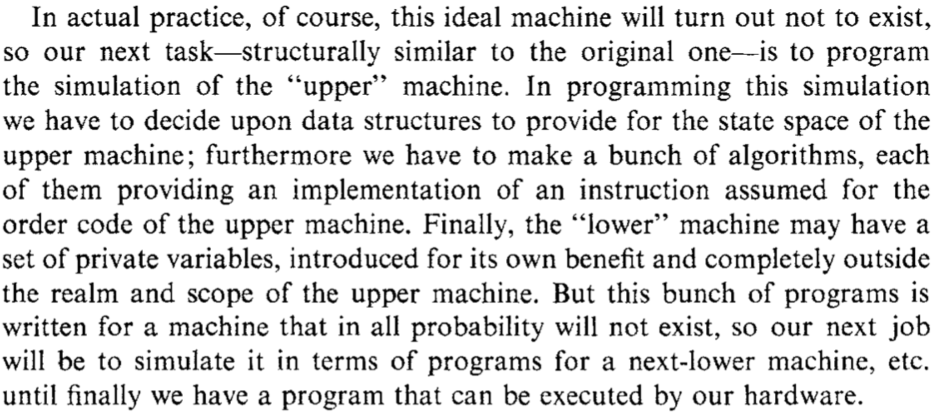
The first and very difficult aspect of design is deciding what the program is to do, and formulating this as a clear, precise, and acceptable specification. Often just as difficult is deciding how to do it, -- how to divide a complex task into simpler subtasks, and to specify the purpose of each part, and define clear, precise, and efficient interfaces between them. A good **programming** language should give assistance in expressing not only how the program is to run, but what it is intended to accomplish; and it should enable this to be expressed at various levels, from the overall strategy to the details of coding and data representation. It should assist in establishing and enforcing the programming conventions and disciplines which will ensure harmonious cooperation of the parts of a large program when they are developed separately and finally assembled together. --

Notes on Structured Programming (Dijkstra 1972)



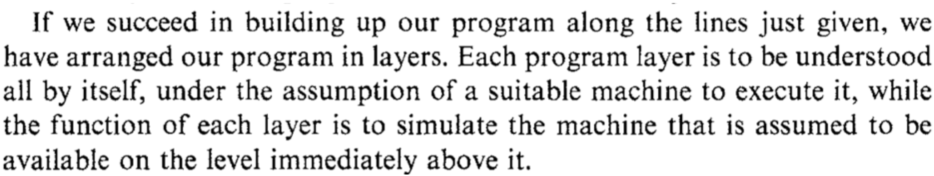
I want to view the main program as executed by its own, dedicated machine, equipped with the adequate instruction repertoire operating on the adequate variables and sequenced under control of its own instruction counter, in order that my main program would solve my problem if I had such a machine. I want to view it that way, because it stresses the fact that the correctness of the main program can be discussed and established regardless of the availability of this (probably still virtual) machine: I don't need to have it, I only need to have its specifications as far as relevant for the proper execution of the main program under consideration.

Notes on Structured Programming (Dijkstra 1972)



In actual practice, of course, this ideal machine will turn out not to exist, so our next task—structurally similar to the original one—is to program the simulation of the "upper" machine. In programming this simulation we have to decide upon data structures to provide for the state space of the upper machine; furthermore we have to make a bunch of algorithms, each of them providing an implementation of an instruction assumed for the order code of the upper machine. Finally, the "lower" machine may have a set of private variables, introduced for its own benefit and completely outside the realm and scope of the upper machine. But this bunch of programs is written for a machine that in all probability will not exist, so our next job will be to simulate it in terms of programs for a next-lower machine, etc. until finally we have a program that can be executed by our hardware.

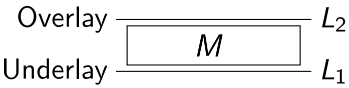
Notes on Structured Programming (Dijkstra 1972)



If we succeed in building up our program along the lines just given, we have arranged our program in layers. Each program layer is to be understood all by itself, under the assumption of a suitable machine to execute it, while the function of each layer is to simulate the machine that is assumed to be available on the level immediately above it.

Certified Abstraction Layers

- **Layer:**

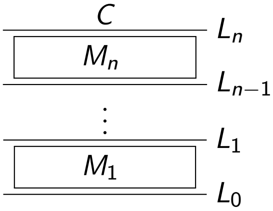


Overlay L_2

M

Underlay L_1

- **Layering:**



C

L_n

M_n

L_{n-1}

$\vdots$

L_1

M_1

L_0

- **Certified Implementation:**

$$L_1 \vdash M : L_2$$

- **Encapsulation:**

???

↑

Abstract State

↑

Concrete State

Modeling State: Global vs Local State

Global State:

- C Memory Model
- ML Reference Types
- Haskell State Monads
- Algebraic Effects (Global)

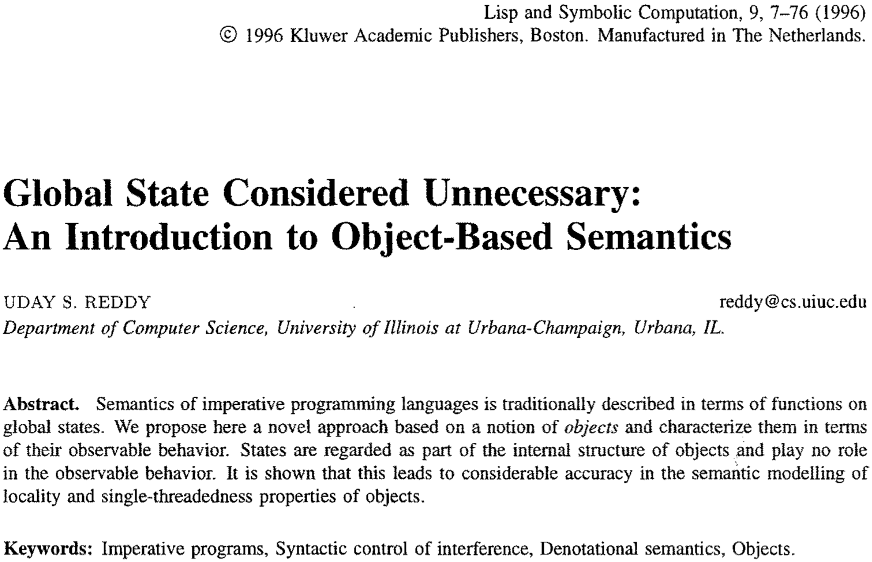
  ⋮

Local State:

- Algebraic Effects (Local)
- Object-Based Semantics

  ⋮

Object-Based Semantics



Lisp and Symbolic Computation, 9, 7–76 (1996)

Global State Considered Unnecessary: An Introduction to Object-Based Semantics

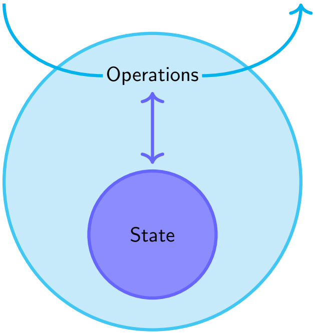
UDAY S. REDDY reddy@cs.uiuc.edu

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

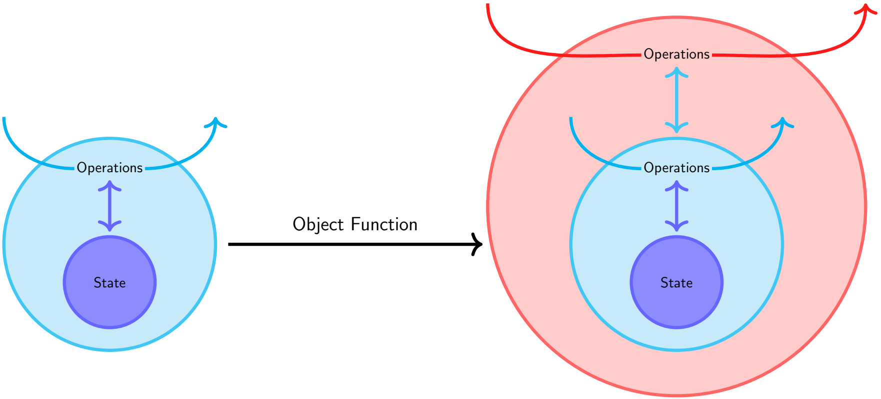
Abstract. Semantics of imperative programming languages is traditionally described in terms of functions on global states. We propose here a novel approach based on a notion of *objects* and characterize them in terms of their observable behavior. States are regarded as part of the internal structure of objects and play no role in the observable behavior. It is shown that this leads to considerable accuracy in the semantic modelling of locality and single-threadedness properties of objects.

Keywords: Imperative programs, Syntactic control of interference, Denotational semantics, Objects.

Objects



Object Functions



Object-Based Semantics and Certified Abstraction Layers

$$
\begin{aligned}
\text{Object Type} &\approx \text{Layer Signature} \\
\text{Object} &\approx \text{Layer Specification} \\
\text{Object Function} &\approx \text{Layer Implementation} \\
\text{???} &\approx \text{Certified Layer}
\end{aligned}
$$

How to build compositional semantic models for certifying large heterogenous systems?

Our Main Results (Oliveira Vale et al POPL 2022)

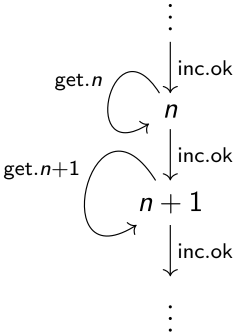
- A novel model of Certified Abstraction Layers.
  - No explicit state
  - Rooted in linear logic
  - Clarifies some aspects of the abstract semantics of CAL
  - Supports an operational account as a game semantics model
  - Supports a denotational account as a domain-theoretic model
- Generalized notion of layer interface that faithfully encapsulates state
- An extension to handle non-determinism in layer interfaces
- An extension to support concurrency

From Code to External Behaviors

```
inc() {
  i ← get();
  set(i+1);
  return ok;
}
.
.
.
```

$\Rightarrow$

$(S_{\mathsf{Counter}}, \rightarrow_{\mathsf{Counter}})$



$\Rightarrow$

$V_{\mathsf{Counter}} \subseteq \dagger\mathsf{Counter}$

$$\begin{aligned} V_{\mathsf{Counter}} = \{ & \epsilon, \mathsf{get}, \mathsf{inc} \\ & \mathsf{get} \cdot 0, \mathsf{inc} \cdot \mathsf{ok}, \\ & \mathsf{get} \cdot 0 \cdot \mathsf{get}, \\ & \mathsf{get} \cdot 0 \cdot \mathsf{inc}, \\ & \mathsf{inc} \cdot \mathsf{ok} \cdot \mathsf{get}, \\ & \mathsf{inc} \cdot \mathsf{ok} \cdot \mathsf{inc}, \\ & \mathsf{get} \cdot 0 \cdot \mathsf{get} \cdot 0, \\ & \ldots, \\ \} & \end{aligned}$$

Layer Signatures

Every layer receives a type describing the operations it supports.

Example

$\mathsf{Var} := \{\mathsf{get} : \mathbf{1} \to \mathbb{N}, \mathsf{set} : \mathbb{N} \to \mathbf{1}\}$ $\mathsf{Counter} := \{\mathsf{get} : \mathbf{1} \to \mathbb{N}, \mathsf{set} : \mathbb{N} \to \mathbf{1}, \mathsf{init} : \mathbf{1} \to \mathbf{1}\}$

Which implicitly describe small interactions:

$$\mathsf{get} \longrightarrow n$$

$$\mathsf{set}(m) \longrightarrow \mathsf{ok}$$

$$\mathsf{get} \longrightarrow n$$

$$\mathsf{inc} \longrightarrow \mathsf{ok}$$

$$\mathsf{init} \longrightarrow \mathsf{ok}$$

Effect Signatures as Types for Systems

Definition

An **effect signature** consists of a set E of **operations** together with an assignment $\mathrm{ar}(-) : E \to \mathbf{Set}$ of a set to each effect.

$$E := \{e_1 : \mathrm{ar}(e_1), e_2 : \mathrm{ar}(e_2), \ldots\}$$

The set $\mathrm{ar}(e)$ is called the **arity** of e.

Its associated game E has, for every $e \in E$ and $v \in \mathrm{ar}(e)$, plays:

$$\begin{tikzcd}
env \arrow[d, dotted, no head] & \\
e \arrow[r] & v \arrow[d, dotted, no head] \\
& sys
\end{tikzcd}$$

The Replay Modality

A signature such as Var allows interactions with a single operation:

$$\mathsf{get} \longrightarrow n \qquad\qquad \mathsf{set}(m) \longrightarrow \mathsf{ok}$$

Layers generally support many consecutive interactions:

$$\mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{set}(3) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get} \longrightarrow 3$$

The Replay Modality

The type $\dagger E$ has as corresponding game $\dagger E$ which consists of plays:

$$\begin{tikzcd}
env \arrow[d, dotted, no head] & & env \arrow[d, dotted, no head] & & & env \arrow[d, dotted, no head] & \\
e_1 \arrow[r] & v_1 \arrow[r] \arrow[d, dotted, no head] & e_2 \arrow[r] & v_2 \arrow[r] \arrow[d, dotted, no head] & \dots \arrow[r] & e_n \arrow[r, dashed] & v_n \arrow[d, dotted, no head] \\
& sys & & sys & & & sys
\end{tikzcd}$$

where for every $i \leq n$, $e_i \in E$ and $v_i \in \mathrm{ar}(e_i)$.

The Replay Modality

Example

The game for $\dagger\mathsf{Var}$ supports plays such as

1. $\mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{set}(3) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get} \longrightarrow 3$
2. $\mathsf{get} \longrightarrow 5$
3. $\mathsf{set}(5) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get} \longrightarrow 42 \longrightarrow \mathsf{get} \longrightarrow 17$
4. $\mathsf{set}(5) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{set}(3) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get}$

External Behaviors from Transition Systems

$$\textit{Interaction} : \qquad \text{get} \longrightarrow$$

$$\textit{State} : \qquad 0 \xrightarrow{\text{get}.0} {\color{red}0}$$

External Behaviors from Transition Systems

$$\textit{Interaction}: \quad \text{get} \longrightarrow 0$$

$$\textit{State}: \quad 0 \xrightarrow{\text{get}.0} 0$$

External Behaviors from Transition Systems

$$\textit{Interaction}: \qquad \mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{set}(3)$$

$$\textit{State}: \qquad 0 \xrightarrow{\qquad\mathsf{get}.0\qquad} 0$$

External Behaviors from Transition Systems

$$\textit{Interaction}: \quad \mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{set}(3) \longrightarrow \mathsf{ok}$$

$$\textit{State}: \quad 0 \xrightarrow{\qquad\quad \mathsf{get}.0 \qquad\quad} 0 \xrightarrow{\mathsf{set}.3} 3$$

External Behaviors from Transition Systems

$$\textit{Interaction}: \quad \mathsf{get} \longrightarrow 0 \longrightarrow \mathsf{set}(3) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get}$$

$$\textit{State}: \quad 0 \xrightarrow{\quad\mathsf{get.0}\quad} 0 \xrightarrow{\quad\mathsf{set(3).ok}\quad} 3$$

External Behaviors from Transition Systems

$$\mathit{Interaction}: \qquad \mathrm{get} \longrightarrow 0 \longrightarrow \mathrm{set}(3) \longrightarrow \mathrm{ok} \longrightarrow \mathrm{get} \longrightarrow$$

$$\mathit{State}: \qquad 0 \xrightarrow{\quad\mathrm{get}.0\quad} 0 \xrightarrow{\quad\mathrm{set}(3).\mathrm{ok}\quad} 3 \xrightarrow{\mathit{get}.3} 3$$

External Behaviors from Transition Systems

$$\mathit{Interaction}: \quad \mathrm{get} \longrightarrow 0 \longrightarrow \mathrm{set}(3) \longrightarrow \mathrm{ok} \longrightarrow \mathrm{get} \longrightarrow 3$$

$$\mathit{State}: \quad 0 \xrightarrow{\mathrm{get}.0} 0 \xrightarrow{\mathrm{set}(3).\mathrm{ok}} 3 \xrightarrow{get.3} 3$$

Layer Specifications from State Transition Systems

Given a transition system $L = (S, \rightarrow \subseteq S \times (\cup_{e \in E} e \times \mathsf{ar}(e)) \times S)$ we recursively construct a set of plays $L\sharp q$ by

$$\epsilon \in L\sharp q \qquad \forall e \in E \Rightarrow e \in L\sharp q \qquad e \rightarrow v \xrightarrow{s} \in L\sharp q \iff \exists q' \in S.q \xrightarrow{e.v} q' \wedge s \in L\sharp q'$$

Example (Bounded Queue)

Signature $E_{\mathsf{bq}} = \{\mathsf{enq} : \mathbb{U} \rightarrow \mathbf{1}, \mathsf{deq} : \mathbf{1} \rightarrow \mathbb{U}\}$

States $S_{\mathsf{bq}} = \mathbb{U}^*$

Transitions

- $|\vec{q}| < N \Rightarrow \vec{q} \xrightarrow{\mathsf{enq}(v).\mathsf{ok}} \vec{q}v$
- $\vec{q} = v\vec{q'} \Rightarrow \vec{q} \xrightarrow{\mathsf{deq}.v} \vec{q'}$

We can define a layer specification for E_{bq} as

$$V_{\mathsf{bq}} := (S_{\mathsf{bq}}, \rightarrow)\sharp\langle\rangle$$

Example: Variable Object

Example

The usual Var semantics is encoded as an object as the set V_{Var} of plays s of $\dagger\mathsf{Var}$ representing its externally observable behaviors:

- If get is the first operation then it returns 0

$$s = \ \mathsf{get} \longrightarrow v \longrightarrow \dots \ \Rightarrow v = 0$$

- Consecutive calls to get return the same value:

$$s = \ \xrightarrow{s_1} \mathsf{get} \longrightarrow v_1 \longrightarrow \mathsf{get} \longrightarrow v_2 \xrightarrow{s_2} \ \Rightarrow v_1 = v_2$$

- A call to get after a successful set(n) returns n

$$s = \ \xrightarrow{s_1} \mathsf{set}(n) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get} \longrightarrow v \xrightarrow{s_2} \ \Rightarrow v = n$$

Example: Counter Object

Example

Similarly, the object V_{Counter} encoding the usual Counter semantics is defined as the set of all plays s of †Counter such that:

- Any non-empty play must start with the initialization procedure:

$$s = \quad e \longrightarrow v \xrightarrow{s'} \quad \Rightarrow e = \mathsf{init}$$

- Calls to get return the same number of calls to inc that happened since the last init:

$$s = \quad \xrightarrow{s_1} \mathsf{get} \longrightarrow v \xrightarrow{s_2} \quad \Rightarrow v = \#\mathsf{inc}(s_1)$$

Layer Specifications

We use objects over types E as layer specifications. Generally:

Definition

A **layer specification** V_E for an effect signature E is a **deterministic strategy** $V_E : \dagger E$. That is, it is a set of plays of $\dagger E$ satisfying:

- V_E is non-empty
- V_E is prefix-closed
- V_E is receptive: If $\xrightarrow{s} \;\in V_E$ is even-length then $\forall e \in E.\ \xrightarrow{s} e \;\in V_E$
- V_E is deterministic: $$\begin{tikzcd} & & v \\ {} \arrow[r, "s"] & e \arrow[ru] \arrow[rd] & \\ & & v' \end{tikzcd} \in V_E \Rightarrow v = v'$$

Determinism

$$\begin{array}{ll} e_1 \longrightarrow v_1 \longrightarrow \ldots \longrightarrow e_k \longrightarrow v_k \longrightarrow \ldots \longrightarrow e_n \dashrightarrow v_n & \\ & \in V_E \Rightarrow v_k = v'_k \\ e_1 \longrightarrow v_1 \longrightarrow \ldots \longrightarrow e_k \longrightarrow v'_k \longrightarrow \ldots \longrightarrow e'_m \dashrightarrow v'_m & \end{array}$$

Layer Specifications are Stateful

$$S_E = \text{plays of } \dagger E \qquad\qquad s \xrightarrow{e.v} s \cdot e \cdot v \iff s \cdot e \cdot v \in V_E$$

Implementations

Example

We would like to define an implementation

$$M : \mathsf{Var} \to \mathsf{Counter}$$

encoding the code:

```
inc() {
  i ← get();
  set(i+1);
  return ok;
}
```

```
get() {
  i ← get();
  return i;
 }
```

```
init() {
  set(0);
  return ok;
  }
```

Object Functions

As the implementation must map objects to objects it needs to be able to handle many consecutive operations:

$$\widehat{M} : \dagger\mathsf{Var} \multimap \dagger\mathsf{Counter}$$

But as it must handle any object of type Var it must not assume anything about the state of the object across invocations

$$\forall n_1, n_2 \in \mathbb{N}. \quad \begin{tikzcd}
\mathsf{inc} \arrow[dr, bend right] & & & & & \mathsf{ok} \arrow[r] & \mathsf{get} \arrow[dr, bend right] & & & n_2 \\
& \mathsf{get} \arrow[r] & n_1 \arrow[r] & \mathsf{set}(n_1 + 1) \arrow[r] & \mathsf{ok} \arrow[ur, bend right] & & & \mathsf{get} \arrow[r] & n_2 \arrow[ur, bend right] &
\end{tikzcd}$$

$$\in \widehat{M}$$

Theorem (Reddy)

$$\dagger E \longrightarrow_{\mathsf{Reg}} \dagger F \cong \dagger E \multimap F$$

Object Functions

As the implementation must map objects to objects it needs to be able to handle many consecutive operations:

$$\widehat{M} : \dagger\mathsf{Var} \multimap \dagger\mathsf{Counter}$$

But as it must handle any object of type Var it must not assume anything about the state of the object across invocations

$$\begin{tikzcd}
\forall n_1, n_2 \in \mathbb{N}. & \text{inc} \arrow[dr, bend right] & & & & & \text{ok} \arrow[r] & \text{get} \arrow[dr, bend right] & & & n_2 \\
& & \text{get} \arrow[r] & n_1 \arrow[r] & \text{set}(n_1 + 1) \arrow[r] & \text{ok} \arrow[ur, bend right] & & & \text{get} \arrow[r] & n_2 \arrow[ur, bend right] & \\
& & & & \in \widehat{M} & & & & & &
\end{tikzcd}$$

Theorem (Reddy)

$$\dagger E \longrightarrow_{\mathsf{Reg}} \dagger F \cong \dagger E \multimap F$$

Example: Implementing Counter

We will define the implementation M

$$M : \mathsf{Var} \to \mathsf{Counter} \text{ a linear map } M : \dagger\mathsf{Var} \multimap \mathsf{Counter}$$

as:

$$M := M^{\mathsf{inc}} \cup M^{\mathsf{get}} \cup M^{\mathsf{init}}$$

where:

$$M^{\mathsf{inc}} := \downarrow \left\{ \begin{tikzcd} \mathsf{inc} \arrow[dr, bend right] & & & & & \mathsf{ok} \\ & \mathsf{get} \arrow[r] & n \arrow[r] & \mathsf{set}(n+1) \arrow[r] & \mathsf{ok} \arrow[ur, bend right] & \end{tikzcd} \middle| \; n \in \mathbb{N} \right\}$$

$$M^{\mathsf{get}} := \downarrow \left\{ \begin{tikzcd} \mathsf{get} \arrow[dr, bend right] & & & n \\ & \mathsf{get} \arrow[r] & n \arrow[ur, bend right] & \end{tikzcd} \middle| \; n \in \mathbb{N} \right\} M^{\mathsf{init}} := \downarrow \left\{ \begin{tikzcd} \mathsf{init} \arrow[dr, bend right] & & & \mathsf{ok} \\ & \mathsf{set}(0) \arrow[r] & \mathsf{ok} \arrow[ur, bend right] & \end{tikzcd} \right.$$

Regular Map Correspondence

$$\begin{tikzcd}
\mathsf{inc} \arrow[rrrrr, dotted, no head, "M"] \arrow[dr, bend right] & & & & & \mathsf{ok} \arrow[r] & \mathsf{get} \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & n_2 & \in \widehat{M} \\
 & \mathsf{get} \arrow[r] & n_1 \arrow[r] & \mathsf{set}(n_1 + 1) \arrow[r] & \mathsf{ok} \arrow[ur, bend right] & & & \mathsf{get} \arrow[r] & n_2 \arrow[ur, bend right] & &
\end{tikzcd}$$

Linear maps $\dagger\mathsf{Var} \multimap \mathsf{Counter}$ can be extended to regular maps $\dagger\mathsf{Var} \multimap \dagger\mathsf{Counter}$ similarly to a Kleene star:

$$\widehat{M} \approx M^*$$

Regular maps have no state!

Game Semantics of Implementations

Plays of $\dagger E \multimap F$ are of the form

$$\begin{tikzcd}
f \arrow[dr, bend right] & & & & & & & & v \\
& e_1 \arrow[r] & v_1 \arrow[r] & e_2 \arrow[r] & v_2 \arrow[r] & \ldots \arrow[r] & e_n \arrow[r] & v_n \arrow[ur, bend right] &
\end{tikzcd}$$

Layer Implementations

Definition

An **implementation** $M : E \to F$ is a **deterministic strategy** of type $\dagger E \multimap F$. That is, M is a set of plays of $\dagger E \multimap F$ such that:

- M is non-empty and prefix-closed
- M is receptive:
  - For all $f \in F$ the play f is in M
  - $$\begin{tikzcd} {} \arrow[dr, bend right, "s"] & \\ & e \end{tikzcd} \in M \Rightarrow \begin{tikzcd} {} \arrow[dr, bend right, "s"] & & \\ & e \arrow[r] & v \end{tikzcd} \in M$$
- M is deterministic
  - $$\begin{tikzcd} {} \arrow[dr, bend right, "s"] & \\ & e \end{tikzcd} \in M \wedge \begin{tikzcd} {} \arrow[dr, bend right, "s"] & \\ & e' \end{tikzcd} \in M \Rightarrow e = e'$$
  - $$\begin{tikzcd} f \arrow[dr, bend right, dash] & & & & v \\ & {} & {} \arrow[r, "s"] & {} \arrow[ur, bend right] & \end{tikzcd} \in M \wedge \begin{tikzcd} f \arrow[dr, bend right, dash] & & & & \\ & {} & {} \arrow[r, "s"] & {} \arrow[r] & m \end{tikzcd} \in M \Rightarrow m = v$$

Regular Extension of Implementations

Definition

Given an implementation $M : E \to F$ its **regular extension** $\widehat{M}$ is a deterministic strategy of type $\dagger E \multimap \dagger F$ defined as

$$\widehat{M} := \{ s_1 \cdot \ldots \cdot s_n \mid s_1, \ldots, s_n \in M \}$$

Essentially, the plays of $\widehat{M}$ are of the form:

$$\begin{tikzcd}
f_1 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_1 \arrow[r, bend left] & f_2 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_2 \arrow[r, bend left] & \ldots \arrow[r, bend left] & f_n \arrow[rrr, dotted, no head, "M"] \arrow[dr, dashed, bend right] & & & v_n \\
& {} \arrow[r, "s_1"] & {} \arrow[ur, bend right] & & & {} \arrow[r, "s_2"] & {} \arrow[ur, bend right] & & & & {} \arrow[r, dashed, "s_n"] & {} \arrow[ur, dashed, bend right] &
\end{tikzcd}$$

Implementation Composition

An implementation $N : F \to G$ makes several invocations to operations in F:

$$\begin{tikzcd}
g \arrow[rrrrrrrr, dotted, no head, "N"] \arrow[dr, bend right] & & & & & & & & v \\
& f_1 \arrow[r] & v_1 \arrow[r] & f_2 \arrow[r] & v_2 \arrow[r] & \dots \arrow[r] & f_n \arrow[r] & v_n \arrow[ur, bend right] &
\end{tikzcd}$$

While $M : E \to F$ describes plays

$$\begin{tikzcd}
f \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v \\
& {} \arrow[r, "s \in \dagger E"] & {} \arrow[ur, bend right] &
\end{tikzcd}$$

Which involves a single invocation of F.

Composing Strategies

Essentially, interactions have shape

$$\begin{tikzcd}
g \arrow[rrrrrrrrrrrrrr, dotted, no head, "N"] \arrow[dr, bend left] & & & & & & & & & & & & & & v \\
& f_1 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_1 \arrow[r, bend left] & f_2 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_2 \arrow[r, bend left] & \cdots \arrow[r, bend left] & f_n \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_n \arrow[ur, bend left] & \\
& & {} \arrow[r, "s_1"] & {} \arrow[ur, bend right] & & & {} \arrow[r, "s_2"] & {} \arrow[ur, bend right] & & & & {} \arrow[r, "s_1"] & {} \arrow[ur, bend right] & &
\end{tikzcd}$$

Which after hiding look like

$$\begin{tikzcd}
g \arrow[rrrrrrrrr, dotted, no head, "N \circ M"] \arrow[dr, bend right] & & & & & & & & & v \\
& {} \arrow[r, "s_1"] & {} \arrow[r, "s_2"] & {} \arrow[r, "s_3"] & {} \arrow[r, "s_4"] & {} & \cdots \arrow[r, "s_{n-1}"] & {} \arrow[r, "s_n"] & {} \arrow[ur, bend right] &
\end{tikzcd}$$

Refinement

Definition

For specifications V_E and V'_E over E refinement is defined by

$$V_E \sqsubseteq V'_E := V_E \subseteq V'_E$$

Certified Layer Implementations

Definition

A **certified layer implementation** $M : L_E \to L_F$ between layer interfaces $L_E = (E, V_E)$ and $L_F = (F, V_F)$ consists of an implementation $M : E \to F$ satisfying the refinement property:

$$V_F \sqsubseteq V_E ; \widehat{M}$$

The refinement property explicitly reads as:

$$V_F \sqsubseteq V_E ; \widehat{M} \iff \forall s \in V_F . \exists t \in V_E . \exists p \in \widehat{M} . p\restriction_E = s \land p\restriction_F = t$$

Example

$$M : (\mathrm{Var}, V_{\mathrm{Var}}) \to (\mathrm{Counter}, V_{\mathrm{Counter}})$$

Certified Layer Implementations

$$f_1 \longrightarrow v_1 \longrightarrow f_2 \longrightarrow v_2 \longrightarrow \dots \longrightarrow f_n \longrightarrow v_n \;\in V_E ; \widehat{M}$$

$$\Longleftrightarrow$$

$$\begin{tikzcd}
f_1 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_1 \arrow[r, bend left] & f_2 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_2 \arrow[r, bend left] & \dots \arrow[r, bend left] & f_n \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_n & \in \widehat{M} \\
& {} \arrow[r, "s_1"] \arrow[rrrrrrrrrr, dotted, no head, bend right, "s_1 \cdot \ldots \cdot s_n \in V_E"'] & {} \arrow[ur, bend right] & & & {} \arrow[r, "s_2"] & {} \arrow[ur, bend right] & & & & {} \arrow[r, "s_n"] & {} \arrow[ur, bend right] & &
\end{tikzcd}$$

Outline

Introduction

Layer Signatures: E

Implementations: $M : \dagger E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Outline

Introduction

Layer Signatures: E

Implementations: $M : \dagger E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Layer Signatures

Definition

An **effect signature** consists of

- A set E of **operations**
- A set $\mathrm{ar}(e)$ to each operation e called the **arity** of e

Example

- $\mathrm{Var} := \{\mathrm{get} : \mathbf{1} \to \mathbb{N}, \mathrm{set} : \mathbb{N} \to \mathbf{1}\}$:

  Operations $\mathrm{Var} = \{\mathrm{get}, \mathrm{set}(n) \mid n \in \mathbb{N}\}$

  Arities
  - $\mathrm{ar}(\mathrm{get}) = \mathbb{N}$
  - $\mathrm{ar}(\mathrm{set}(n)) = \mathbf{1} \cong \{\mathrm{ok}\}$

Layer Signatures

Definition

An **effect signature** consists of

- A set E of **operations**
- A set $\mathrm{ar}(e)$ to each operation e called the **arity** of e

Example

- $\mathrm{Var} := \{\mathrm{get} : \mathbf{1} \to \mathbb{N}, \mathrm{set} : \mathbb{N} \to \mathbf{1}\}$:

Operations $\mathrm{Var} = \{\mathrm{get}, \mathrm{set}(n) \mid n \in \mathbb{N}\}$

Arities
- $\mathrm{ar}(\mathrm{get}) = \mathbb{N}$
- $\mathrm{ar}(\mathrm{set}(n)) = \mathbf{1} \cong \{\mathrm{ok}\}$

State-Based Layer Specifications

$$\mathsf{Var} := \{\mathsf{get} : \mathbf{1} \to \mathbb{N}, \mathsf{set} : \mathbb{N} \to \mathbf{1}\}$$

State: $n, m \in S_{\mathsf{Var}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{get.n} n$

$n \xrightarrow{\mathsf{set}(m).\mathsf{ok}} m$

$$\mathsf{FAI} := \{\mathsf{fai} : \mathbf{1} \to \mathbb{N}\}$$

State: $n \in S_{\mathsf{FAI}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{\mathsf{fai}.n} n+1$

State-Based Layer Specifications

$$\text{Var} := \{\text{get} : \mathbf{1} \to \mathbb{N}, \text{set} : \mathbb{N} \to \mathbf{1}\}$$

State: $n, m \in S_{\text{Var}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{get.n} n$

$n \xrightarrow{\text{set}(m).\text{ok}} m$

$$\text{FAI} := \{\text{fai} : \mathbf{1} \to \mathbb{N}\}$$

State: $n \in S_{\text{FAI}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{\text{fai}.n} n + 1$

Bounded Queue

Signature

$$E_{\text{bq}} := \{\text{enq} : \mathbb{U} \to \mathbf{1}, \text{deq} : \mathbf{1} \to \mathbb{U}\}$$

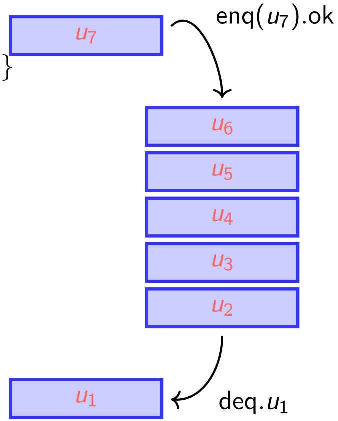
States $S_{\text{bq}} = \mathbb{U}^*$

Initial State ϵ

Transitions

- $|\vec{q}| < N \Rightarrow \vec{q} \xrightarrow{\text{enq}(v).\text{ok}} \vec{q}v$
- $\vec{q} = v\vec{q'} \Rightarrow \vec{q} \xrightarrow{\text{deq}.v} \vec{q'}$

Queue Semantics:



Bounded Queue

Signature

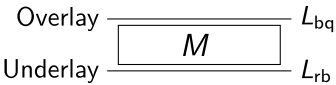
$$E_{\mathsf{bq}} := \{\mathsf{enq} : \mathbb{U} \to \mathbf{1}, \mathsf{deq} : \mathbf{1} \to \mathbb{U}\}$$

States $S_{\mathsf{bq}} = \mathbb{U}^*$

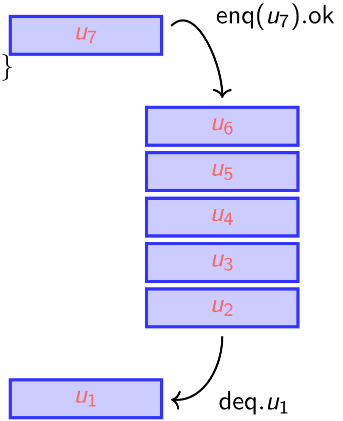
Initial State ϵ

Transitions

- $|\vec{q}| < N \Rightarrow \vec{q} \xrightarrow{\mathsf{enq}(v).\mathsf{ok}} \vec{q}v$
- $\vec{q} = v\vec{q'} \Rightarrow \vec{q} \xrightarrow{\mathsf{deq}.v} \vec{q'}$



Queue Semantics:



Ring Buffer

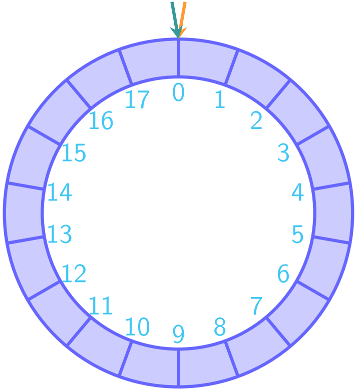
Signature:

$$E_{\mathsf{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\}$$
$$\cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathsf{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

Initial State:



Ring Buffer

Signature:

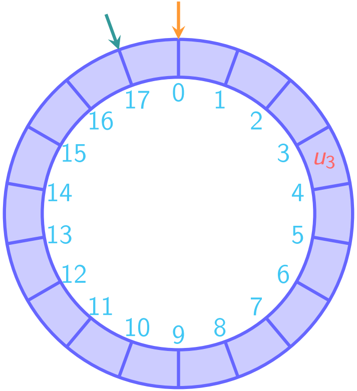
$$E_{\mathrm{rb}} := \{\mathrm{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathrm{get} : \mathbb{N} \to \mathbb{U}\}$$
$$\cup \{\mathrm{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathrm{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathrm{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

set Semantics:

$$(\varnothing, 0, 17) \xrightarrow{\mathrm{set}(3, u_3).\mathrm{ok}} (\{3 \mapsto u_3\}, 0, 17)$$



Ring Buffer

Signature:

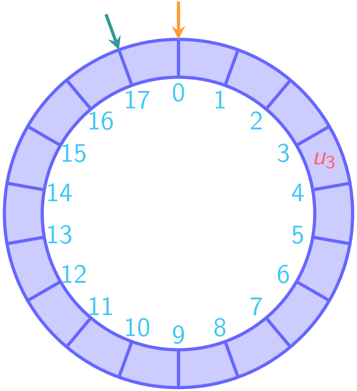
$$E_{\mathrm{rb}} := \{\mathrm{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathrm{get} : \mathbb{N} \to \mathbb{U}\} \cup \{\mathrm{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathrm{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathrm{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

get Semantics:

$$(\{3 \mapsto u_3\}, 0, 17) \xrightarrow{\mathrm{get}(3).u_3} (\{3 \mapsto u_3\}, 0, 17)$$



Ring Buffer

Signature:

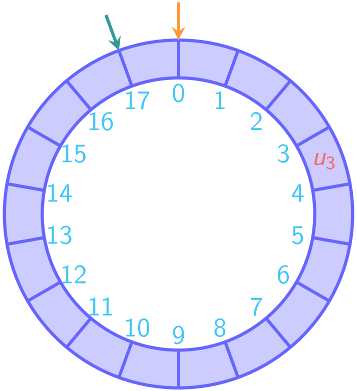
$$E_{\mathsf{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\}$$
$$\cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathsf{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

get Semantics:

$$(\{3 \mapsto u_3\}, 0, 17) \cancel{\xrightarrow{\mathsf{get}(0).u_3}}$$



Ring Buffer

Signature:

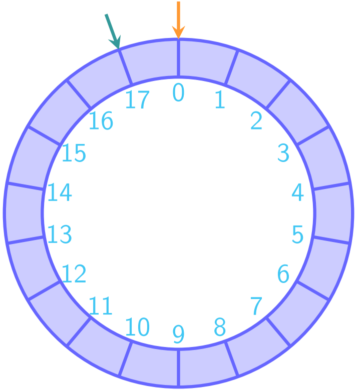
$$E_{\mathsf{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\} \cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathsf{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

fai Semantics:

$$(\varnothing, 0, 17) \xrightarrow{\mathsf{fai}_1.0} (\varnothing, 1, 17)$$



Ring Buffer

Signature:

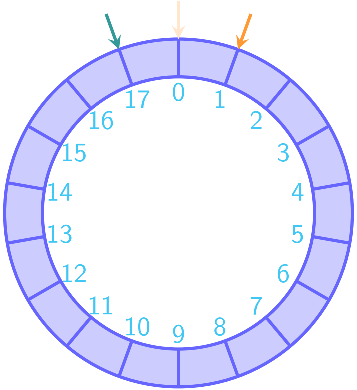
$$E_{\mathsf{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\} \\ \cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathsf{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

fai Semantics:

$$(\varnothing, 0, 17) \xrightarrow{\mathsf{fai}_1.0} (\varnothing, 1, 17)$$



Ring Buffer

Signature:

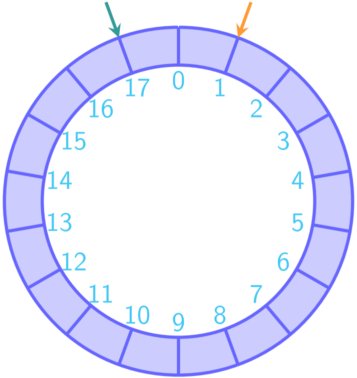
$$E_{\mathsf{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\}$$
$$\cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathsf{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

Initial State: $(\varnothing, 0, 0)$

fai Semantics:

$$(\varnothing, 1, 17) \xrightarrow{\mathsf{fai}_2.17} (\varnothing, 1, 0)$$



Ring Buffer

Signature:

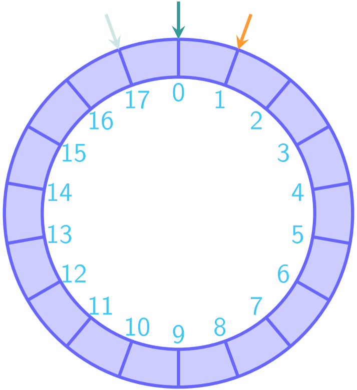
$$E_{\mathsf{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\} \cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

State: $S_{\mathsf{rb}} := \mathbb{U}^{\mathbb{N}} \times \mathbb{N} \times \mathbb{N}$

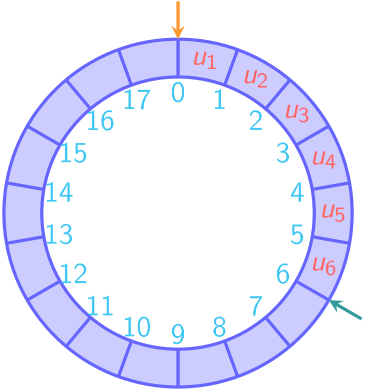
Initial State: $(\varnothing, 0, 0)$

fai Semantics:

$$(\varnothing, 1, 17) \xrightarrow{\mathsf{fai}_2.17} (\varnothing, 1, 0)$$



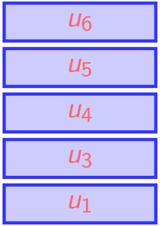
Implementing a Bounded Queue using a Ring Buffer



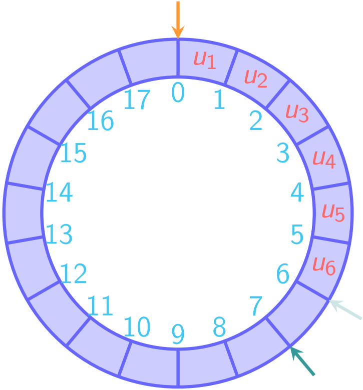
M

```
enq(u) {
  i ← fai_2();
  set(i,u);
  return ok
}
deq() {
  i ← fai_1();
  get(i)
}
```

enq(u_7)

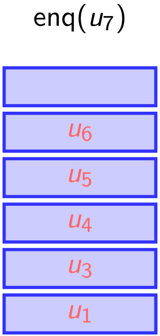


Implementing a Bounded Queue using a Ring Buffer

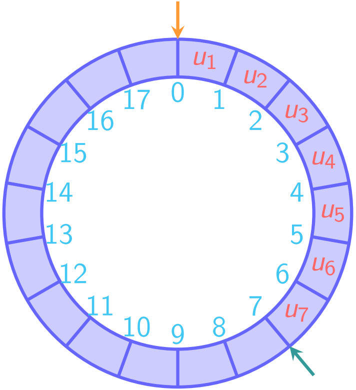


M

```
enq(u) {
  i ← $\mathrm{fai}_2$();
  set(i,u);
  return ok
}
deq() {
  i ← $\mathrm{fai}_1$();
  get(i)
}
```



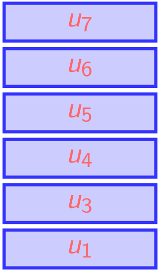
Implementing a Bounded Queue using a Ring Buffer



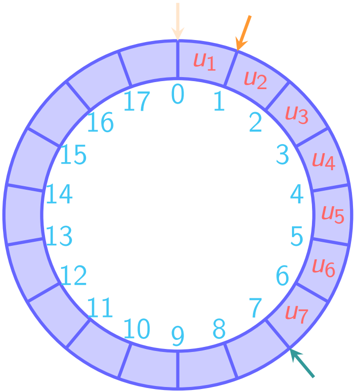
M

```
enq(u) {
   i ← $\mathrm{fai}_2$();
   set(i,u);
   return ok
}
deq() {
   i ← $\mathrm{fai}_1$();
   get(i)
}
```

enq(u_7).ok



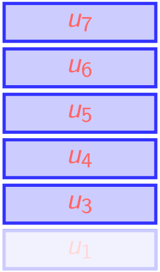
Implementing a Bounded Queue using a Ring Buffer



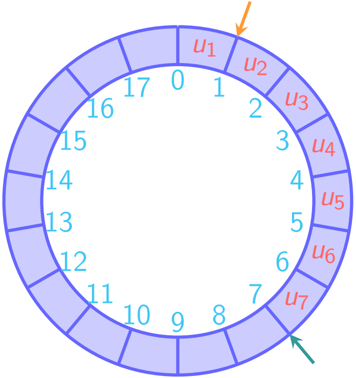
M

```
enq(u) {
  i ← fai_2();
  set(i,u);
  return ok
}
deq() {
  i ← fai_1();
  get(i)
}
```

deq



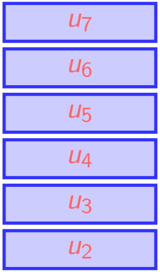
Implementing a Bounded Queue using a Ring Buffer



M

```
enq(u) {
   i ← $\mathsf{fai}_2()$;
   set(i,u);
   return ok
}
deq() {
   i ← $\mathsf{fai}_1()$;
   get(i)
}
```

deq.u_1



Outline

Introduction

Layer Signatures: E

Implementations: $M : {\dagger}E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Layer Signatures and Interactions

Every effect signature E has an associated game, which for every $e \in E$ and $v \in \mathrm{ar}(e)$, has a play:

$$\begin{tikzcd}
\color{blue}{\mathit{env}} \arrow[d, dotted, no head] & \\
e \arrow[r] & v \arrow[d, dotted, no head] \\
& \color{green}{\mathit{sys}}
\end{tikzcd}$$

Example

$$E_{\mathrm{rb}} := \{\mathsf{set} : \mathbb{N} \times \mathbb{U} \to \mathbf{1}, \mathsf{get} : \mathbb{N} \to \mathbb{U}\} \\ \cup \{\mathsf{fai}_1 : \mathbf{1} \to \mathbb{N}, \mathsf{fai}_2 : \mathbf{1} \to \mathbb{N}\}$$

$$\mathsf{set}(i, u) \longrightarrow \mathsf{ok}$$

$$\mathsf{get}(i) \longrightarrow u$$

$$\mathsf{fai}_1 \longrightarrow v$$

$$\mathsf{fai}_2 \longrightarrow v$$

The Replay Modality $\dagger E$

The game $\dagger E$ consists of plays:

$$\begin{tikzcd}
\mathit{env} & & \mathit{env} & & & \mathit{env} & \\
e_1 \arrow[u, dotted, no head] \arrow[r] & v_1 \arrow[d, dotted, no head] \arrow[r] & e_2 \arrow[u, dotted, no head] \arrow[r] & v_2 \arrow[d, dotted, no head] \arrow[r] & \dots \arrow[r] & e_n \arrow[u, dotted, no head] \arrow[r, dashed] & v_n \arrow[d, dotted, no head] \\
& \mathit{sys} & & \mathit{sys} & & & \mathit{sys}
\end{tikzcd}$$

Example

A linear logic modality $\dagger E_{\mathrm{rb}}$ describes plays as sequences of E_{rb} interactions:

- $\mathrm{fai}_1 \longrightarrow 0 \longrightarrow \mathrm{fai}_1 \longrightarrow 1 \longrightarrow \mathrm{fai}_1 \longrightarrow 2$
- $\mathrm{fai}_1 \longrightarrow 7 \longrightarrow \mathrm{get}(3) \longrightarrow u_3 \longrightarrow \mathrm{set}(2, u_2) \longrightarrow \mathrm{ok} \longrightarrow \mathrm{fai}_2 \longrightarrow 13$

Implementation Plays: $\dagger E \multimap F$

Plays of $\dagger E \multimap F$ are of the form

$$\begin{tikzcd}
f \arrow[dr, bend right] & & & & & & & & v \\
& e_1 \arrow[r] & v_1 \arrow[r] & e_2 \arrow[r] & v_2 \arrow[r] & \dots \arrow[r] & e_n \arrow[r] & v_n \arrow[ur, bend right] &
\end{tikzcd}$$

Implementations: Example

$$M := \left(\cup_{u \in \mathbb{U}} M^{\mathsf{enq}(u)} \right) \cup M^{\mathsf{deq}}$$

```
enq(u) {
  i ← $\mathsf{fai}_2()$;
  set(i,u);
  return ok
}
```

$$\Rightarrow \quad M^{\mathsf{enq}(u)} := \downarrow \left\{ \begin{tikzcd} \mathsf{enq}(u) \arrow[dr, bend right] & & & & & \mathsf{ok} \\ & \mathsf{fai}_2 \arrow[r] & i \arrow[r] & \mathsf{set}(i,u) \arrow[r] & \mathsf{ok} \arrow[ur, bend right] & \end{tikzcd} \mid i \in \mathbb{N} \right\}$$

```
deq() {
  i ← $\mathsf{fai}_1()$;
  get(i)
}
```

$$\Rightarrow \quad M^{\mathsf{deq}} := \downarrow \left\{ \begin{tikzcd} \mathsf{deq} \arrow[dr, bend right] & & & & & u \\ & \mathsf{fai}_1 \arrow[r] & i \arrow[r] & \mathsf{get}(i) \arrow[r] & u \arrow[ur, bend right] & \end{tikzcd} \mid i \in \mathbb{N}, u \in \mathbb{U} \right\}$$

Regular Extensions

From an implementation:

$$M : \dagger E \multimap F$$

We build its regular extension:

$$\widehat{M} : \dagger E \multimap \dagger F$$

by

$$\begin{tikzcd}
f_1 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_1 \arrow[r, bend left] & f_2 \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right] & & & v_2 \arrow[r, bend left] & \cdots \arrow[r, bend left] & f_n \arrow[rrr, dotted, no head, "M"] \arrow[dr, bend right, dashed] & & & v_n & \in \widehat{M} \\
& {} \arrow[r, "s_1"] & {} \arrow[ur, bend right] & & & {} \arrow[r, "s_2"] & {} \arrow[ur, bend right] & & & & {} \arrow[r, dashed, "s_n"] & {} \arrow[ur, bend right, dashed] & &
\end{tikzcd}$$

In general, approximately:

$$\widehat{M} \approx M^*$$

With local states implementations are stateless!

Implementations as Strategies

An implementation $M : E \to F$ is a **deterministic strategy** of type $\dagger E \multimap F$.
That is, M is a set of plays that is:

- **non-empty**,
- **prefix-closed**,
- **deterministic**

The Structure of the Replay Modality

The replay modality $\dagger -$ is a Comonad:

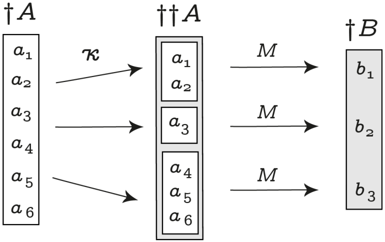
$$\epsilon_A : \dagger A \multimap A$$

$$\kappa_A : \dagger A \multimap \dagger\dagger A$$

The regular extension is just the Kleisli morphism:

$$M : \dagger A \multimap B$$

$$\dagger A \xrightarrow{\widehat{M}} \dagger B = \dagger A \xrightarrow{\kappa_A} \dagger\dagger A \xrightarrow{\dagger M} \dagger B$$



Outline

Introduction

Layer Signatures: E

Implementations: $M : \dagger E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Layer Specifications from State Transition Systems

How to get rid of state-based specifications?

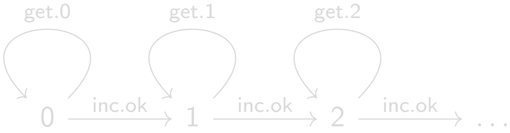
$\text{Counter} := \{\text{get} : \mathbf{1} \to \mathbb{N}, \text{inc} : \mathbf{1} \to \mathbf{1}\}$

State: $n \in S_{\text{Counter}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{get.n} n$

$n \xrightarrow{\text{inc.ok}} n + 1$



Denote the observable behaviors at state q as:

$$(S, \to)\sharp q$$

$$(S_{\text{Counter}}, \to_{\text{Counter}})\sharp 0 = \{\epsilon, \text{get} \cdot 0, \text{inc} \cdot \text{ok}, \text{get} \cdot 0 \cdot \text{get} \cdot 0, \text{inc} \cdot \text{ok} \cdot \text{get} \cdot 1, \\ \text{get} \cdot 0 \cdot \text{inc} \cdot \text{ok}, \text{inc} \cdot \text{ok} \cdot \text{inc} \cdot \text{ok}, \text{get} \cdot 0 \cdot \text{inc} \cdot \text{ok} \cdot \text{get} \cdot 1, \ldots\}$$

Layer Specifications from State Transition Systems

How to get rid of state-based specifications?

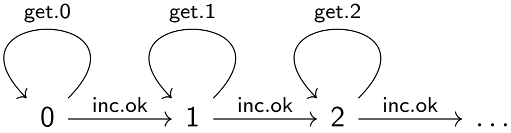
$$\text{Counter} := \{\text{get} : \mathbf{1} \to \mathbb{N}, \text{inc} : \mathbf{1} \to \mathbf{1}\}$$

State: $n \in S_{\text{Counter}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{get.n} n$

$n \xrightarrow{\text{inc.ok}} n+1$



Denote the observable behaviors at state q as:

$$(S, \to)\sharp q$$

$$(S_{\text{Counter}}, \to_{\text{Counter}})\sharp 0 = \{\epsilon, \text{get}\cdot 0, \text{inc}\cdot\text{ok}, \text{get}\cdot 0\cdot\text{get}\cdot 0, \text{inc}\cdot\text{ok}\cdot\text{get}\cdot 1,$$
$$\text{get}\cdot 0\cdot\text{inc}\cdot\text{ok}, \text{inc}\cdot\text{ok}\cdot\text{inc}\cdot\text{ok}, \text{get}\cdot 0\cdot\text{inc}\cdot\text{ok}\cdot\text{get}\cdot 1, \ldots\}$$

Layer Specifications from State Transition Systems

How to get rid of state-based specifications?

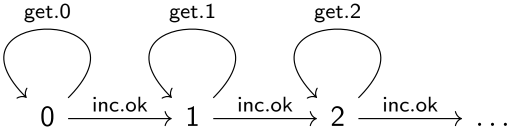
$$\text{Counter} := \{\text{get} : \mathbf{1} \to \mathbb{N}, \text{inc} : \mathbf{1} \to \mathbf{1}\}$$

State: $n \in S_{\text{Counter}} := \mathbb{N}$

Initial State: 0

Transitions: $n \xrightarrow{get.n} n$

$n \xrightarrow{\text{inc.ok}} n + 1$



Denote the observable behaviors at state q as:

$$(S, \to)\sharp q$$

$$(S_{\text{Counter}}, \to_{\text{Counter}})\sharp 0 = \{\epsilon, \text{get} \cdot 0, \text{inc} \cdot \text{ok}, \text{get} \cdot 0 \cdot \text{get} \cdot 0, \text{inc} \cdot \text{ok} \cdot \text{get} \cdot 1,$$
$$\text{get} \cdot 0 \cdot \text{inc} \cdot \text{ok}, \text{inc} \cdot \text{ok} \cdot \text{inc} \cdot \text{ok}, \text{get} \cdot 0 \cdot \text{inc} \cdot \text{ok} \cdot \text{get} \cdot 1, \ldots\}$$

Bounded Queue Specification: V_{bq}

Example (Bounded Queue)

Signature $E_{\mathsf{bq}} = \{\mathsf{enq} : \mathbb{U} \to \mathbf{1}, \mathsf{deq} : \mathbf{1} \to \mathbb{U}\}$

States $S_{\mathsf{bq}} = \mathbb{U}^*$

Initial State ϵ

Transitions

- $|\vec{q}| < N \Rightarrow \vec{q} \xrightarrow{\mathsf{enq}(v).\mathsf{ok}} \vec{q}v$
- $\vec{q} = v\vec{q'} \Rightarrow \vec{q} \xrightarrow{\mathsf{deq}.v} \vec{q'}$

We can define a layer specification for E_{bq} as

$$V_{\mathsf{bq}} := (S_{\mathsf{bq}}, \rightarrow)\sharp\epsilon$$

Stateless Variable Specification: V_{Var}

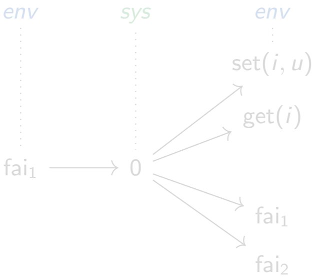
Example

V_{Var} is the set of plays s of $\dagger\mathsf{Var}$ satisfying:

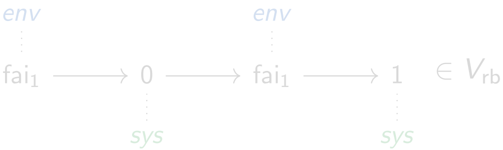
- $$s = \quad \mathsf{get} \longrightarrow v \longrightarrow \ldots \quad \Rightarrow v = 0$$
- $$s = \quad \ldots \ \mathsf{get} \longrightarrow v_1 \longrightarrow \mathsf{get} \longrightarrow v_2 \ \ldots \quad \Rightarrow v_1 = v_2$$
- $$s = \quad \ldots \ \mathsf{set}(n) \longrightarrow \mathsf{ok} \longrightarrow \mathsf{get} \longrightarrow v \ \ldots \quad \Rightarrow v = n$$

Structure of Trace Sets

- Two sides: system and environment
- Interfaces play as system.
- The environment is unpredictable hence non-deterministic:



- The system is deterministic:

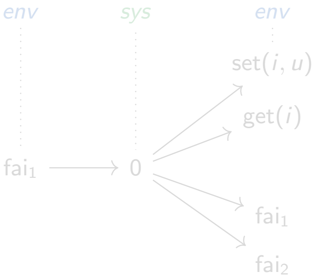


Structure of Trace Sets

- Two sides: system and environment

- Interfaces play as system.

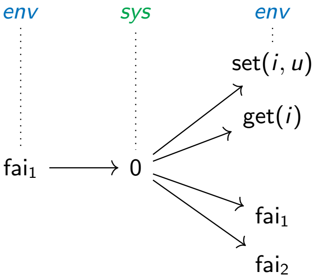
- The environment is unpredictable hence non-deterministic:



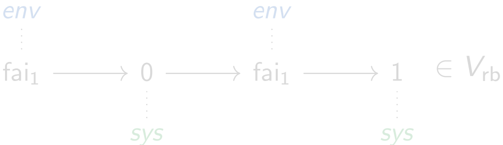
- The system is deterministic: $\overset{env}{\mathrm{fai}_1} \longrightarrow \underset{sys}{0} \longrightarrow \overset{env}{\mathrm{fai}_1} \longrightarrow \underset{sys}{1} \in V_{\mathrm{rb}}$

Structure of Trace Sets

- Two sides: system and environment
- Interfaces play as system.
- The environment is unpredictable hence non-deterministic:

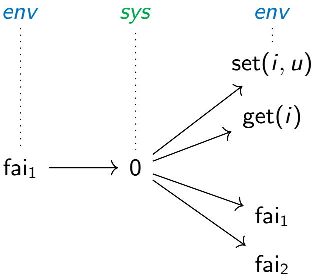


- The system is deterministic: $\mathsf{fai}_1 \longrightarrow 0 \longrightarrow \mathsf{fai}_1 \longrightarrow 1 \in V_{\mathrm{rb}}$



Structure of Trace Sets

- Two sides: system and environment
- Interfaces play as system.
- The environment is unpredictable hence non-deterministic:



- The system is deterministic:

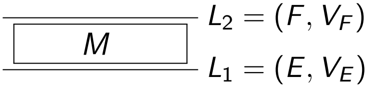
$$\begin{tikzcd} \mathit{env} \arrow[d, dash, dotted] & & \mathit{env} \arrow[d, dash, dotted] & & \\ \mathrm{fai}_1 \arrow[r] & 0 \arrow[r] & \mathrm{fai}_1 \arrow[r] & 1 & \in V_{\mathrm{rb}} \\ & \mathit{sys} \arrow[u, dash, dotted] & & \mathit{sys} \arrow[u, dash, dotted] & \end{tikzcd}$$

Layer Specifications

A **layer specification** V_E over E is a **deterministic strategy** of type $\dagger E$.
That is, V_E is a set of $\dagger E$ plays that is:

- **non-empty**,
- **prefix-closed**,
- **deterministic**

Layer Interfaces



M

$L_2 = (F, V_F)$

$L_1 = (E, V_E)$

Definition

A **layer interface** is a pair $L = (E, V_E : \dagger E)$.

Outline

Introduction

Layer Signatures: E

Implementations: $M : \dagger E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Certified Implementations

- $M : E_{\text{rb}} \to E_{\text{bq}}$ does not know rb semantics or bq semantics.
- Composing with V_{rb} "gives" the rb semantics to $\widehat{M}$:

$$f_1 \longrightarrow v_1 \longrightarrow f_2 \longrightarrow v_2 \longrightarrow \dots \longrightarrow f_n \longrightarrow v_n \quad \in V_E ; \widehat{M}$$

$$\Longleftrightarrow$$

$$\begin{tikzcd}
f_1 \arrow[rr, dotted, no head, "M"] \arrow[dr, bend right] & {} & v_1 \arrow[r, bend left] & f_2 \arrow[rr, dotted, no head, "M"] \arrow[dr, bend right] & {} & v_2 \arrow[r, bend left] & \dots \arrow[r, bend left] & f_n \arrow[rr, dotted, no head, "M"] \arrow[dr, bend right] & {} & v_n & \in \widehat{M} \\
{} & \xrightarrow{s_1} \arrow[ur, bend right] & {} & {} & \xrightarrow{s_2} \arrow[ur, bend right] & {} & {} & {} & \xrightarrow{s_n} \arrow[ur, bend right] & {} & {}
\end{tikzcd}$$

$$\xrightarrow{\qquad s_1 \qquad} \quad \xrightarrow{\qquad s_2 \qquad} \quad \xrightarrow{\qquad s_3 \qquad} \dots \xrightarrow{\qquad s_{n-1} \qquad} \quad \xrightarrow{\qquad s_n \qquad} \quad \in V_E$$

Certified Layer Implementations

Definition

A **certified layer implementation** $L_E \vdash M : V_F$ consists of:

Underlay:	Overlay:	Implementation:
$L_E = (E, V_E)$	$L_F = (F, V_F)$	$M : E \to F$

Correctness:

$$V_F \subseteq V_E ; \widehat{M}$$

Example

$$(E_{\mathsf{rb}}, V_{\mathsf{rb}}) \vdash M : (E_{\mathsf{bq}}, V_{\mathsf{bq}})$$

"For every V_{bq} behavior there is some V_{rb} that can be used by M to implement the V_{bq} behavior"

Outline

Introduction

Layer Signatures: E

Implementations: $M : \dagger E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Non-Deterministic Layer Specifications

A **non-deterministic layer specification** $\mathcal{V}_E$ for E is a non-empty set of $V_E : \dagger E$.

A **non-deterministic layer interface** is pair $(E, \mathcal{V}_E)$.

Example (Non-Deterministic rb Initialization)

- **Bounded queue:** $\mathcal{V}_{\text{bq}} := \{V_{\text{bq}}\}$
- **Ring buffer:**

Deterministic:

$$V_{\text{rb}} := L_{\text{rb}} \sharp (\varnothing, 0, 0)$$

Non-Deterministic:

$$\mathcal{V}_{\text{rb}} := \{L^S_{\text{rb}} \sharp (f, c, c) \mid f \in \mathbb{U}^N, c < N\}$$

Non-Deterministic Certified Abstraction Layers

Definition

A **non-deterministic certified abstraction layer** $L_E \vdash M : L_F$ consists of:

Underlay: $L_E = (E, \mathcal{V}_E)$

Overlay: $L_F = (F, \mathcal{V}_F)$

Implementation: $M : E \to F$

Correctness:

$$\forall V_E \in \mathcal{V}_E. \quad \exists V_F \in \mathcal{V}_F. \quad (E, V_E) \vdash M : (F, V_F)$$

Example

$$(E_{\mathsf{rb}}, \mathcal{V}_{\mathsf{rb}}) \vdash M : (E_{\mathsf{bq}}, \mathcal{V}_{\mathsf{bq}})$$

Outline

Introduction

Layer Signatures: E

Implementations: $M : \dagger E \multimap F$

Layer Interfaces: $L = (E, V_E)$

Certified Layer Implementations: $L_1 \vdash M : L_2$

Non-Deterministic Layer Specifications: $L = (E, \mathcal{V}_E)$

Concurrent Layers: $L = (E, R, V_E)$

Conclusion

Certified Concurrent Layers

- A **concurrent layer interface** is a triple (E, R, V_E) where R is a **coherent congruence** between plays of $\dagger E$.
- R is essentially a "determinism preserving" equational theory:

$$s \cdot e_1 \cdot v_1 \cdot e_2 \cdot v_2 \cdot t \quad R \quad s \cdot e_2 \cdot v_2 \cdot e_1 \cdot v_1 \cdot t$$

- Implementations work on equivalence classes under coherent congruences:

$$M : \dagger_R E \multimap \dagger_S F$$

$$[s]_R \mapsto [t]_S$$

Check paper for details!

Equational Theories and †-Coalgebras

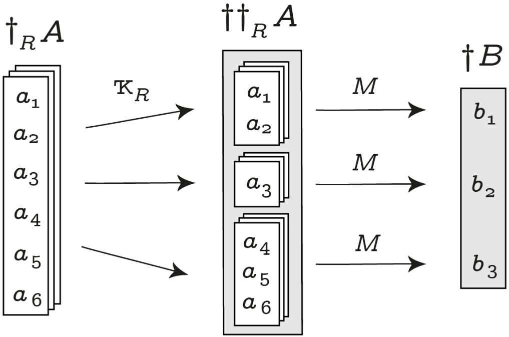
Given equational theory $R \in \mathbf{Rel}(\dagger A, \dagger A)$ we construct a †-Coalgebra:

$$(\dagger_R A, \kappa_R : \dagger_R \multimap \dagger\dagger_R)$$

Generalized regular extension:

$$M : \dagger_R A \multimap B$$

$$\dagger_R A \xrightarrow{\widehat{M}} \dagger B = \dagger_R A \xrightarrow{\kappa_A} \dagger\dagger_R A \xrightarrow{\dagger M} \dagger B$$



Certified Concurrent Layers

This allows us to:

- Reasoning up to the equational theory R
- Express independent products of layers

$$(E, R, V_E) \otimes (F, S, V_F) := (E \uplus F, R \otimes S, V_E \bullet V_F)$$

- Express specific patterns of state sharing (e.g. lock-based)
- Express the implementation of synchronization primitives (e.g. ticket lock)

Conclusion

- We have a novel way of building models of CAL
  - No explicit state
  - Rooted in linear logic
  - Extensible
- Promising direction in compositional semantics for reasoning about large systems.
-

$$\text{New CAL [POPL '22]} \xrightarrow{\text{Semantics}} \text{DeepSEA [OOPSLA '19]} \xrightarrow{\text{Compilation}} \text{CompCertO [PLDI '21]}$$

Layer Implementations are Regular Maps

Implementations are regular maps:

$$\widehat{M} : \dagger E \multimap \dagger F$$

Regular maps are linear maps that are “replayable”:

$$\begin{tikzcd}
f_1 \arrow[rrr, dotted, no head, "\widehat{M}"'] \arrow[rrrrrrrrrrrr, dotted, no head, bend left=15, "\widehat{M}"] \arrow[dr, bend right] & & & v_1 \arrow[r, bend left] & f_2 \arrow[rrr, dotted, no head, "\widehat{M}"'] \arrow[dr, bend right] & & & v_2 \arrow[r, bend left] & \cdots \arrow[r, bend left] & f_n \arrow[rrr, dotted, no head, "\widehat{M}"'] \arrow[dr, dashed, bend right] & & & v_n \\
& {} \arrow[r, "s_1"] & {} \arrow[ur, bend right] & & & {} \arrow[r, "s_2"] & {} \arrow[ur, bend right] & & & & {} \arrow[r, dashed, "s_n"] & {} \arrow[ur, dashed, bend right] &
\end{tikzcd}$$

Theorem (Reddy)

$$\dagger E \longrightarrow_{\mathrm{Reg}} \dagger F \cong \dagger E \multimap F$$

Layer Specifications are Stateful

$$V_{\mathsf{Counter}} = \{\epsilon, \mathsf{get}, \mathsf{inc}, \mathsf{get}\cdot 0, \mathsf{inc}\cdot\mathsf{ok}, \mathsf{inc}\cdot\mathsf{ok}\cdot\mathsf{get}, \mathsf{inc}\cdot\mathsf{ok}\cdot\mathsf{get}\cdot 1, \ldots\}$$

State as Past

$$S_E = \text{plays of } V_E \qquad \text{Initial: } \epsilon \qquad s \xrightarrow{e.v} s\cdot e\cdot v \iff s\cdot e\cdot v \in V_E$$

$$\begin{tikzcd}
\vdots & \vdots & \iddots \\
\mathsf{inc}\cdot\mathsf{ok} \arrow[u, "\mathsf{inc.ok}"] \arrow[r, "\mathsf{get.1}"] & \mathsf{inc}\cdot\mathsf{ok}\cdot\mathsf{get}\cdot 1 \arrow[u, "\mathsf{inc.ok}"] \arrow[ur, "\mathsf{get.1}"] & \iddots \\
\epsilon \arrow[u, "\mathsf{inc.ok}"] \arrow[r, "\mathsf{get.0}"] & \mathsf{get}\cdot 0 \arrow[ur, "\mathsf{inc.ok}"] \arrow[r, "\mathsf{get.0}"] & \ldots
\end{tikzcd}$$