A Logical Relation for Monadic Encapsulation of State

Proving contextual equivalences in the presence of runST

Amin Timany Léo Stefanesco Morten Krogh-Jespersen
Lars Birkedal

imec-DistriNet, KU Leuven

May 18th 2017

KU-Leuven

On the occasion of Ph.D. defence of Jesper Cockx

ST Monad (Launchbury and Peyton Jones 1994)

The idea

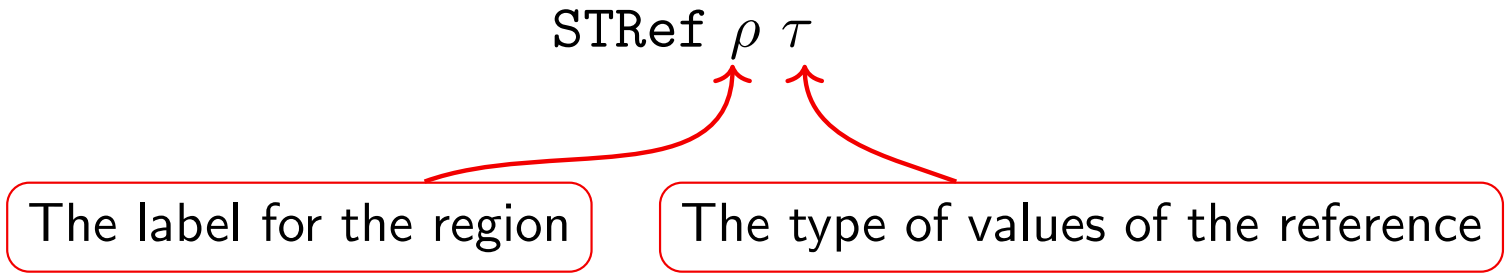
Encapsulating the effect of using the **heap**

ST Monad (Launchbury and Peyton Jones 1994)

The idea

Encapsulating the effect of using the **heap**

- The idea: label references with regions (also: state threads)

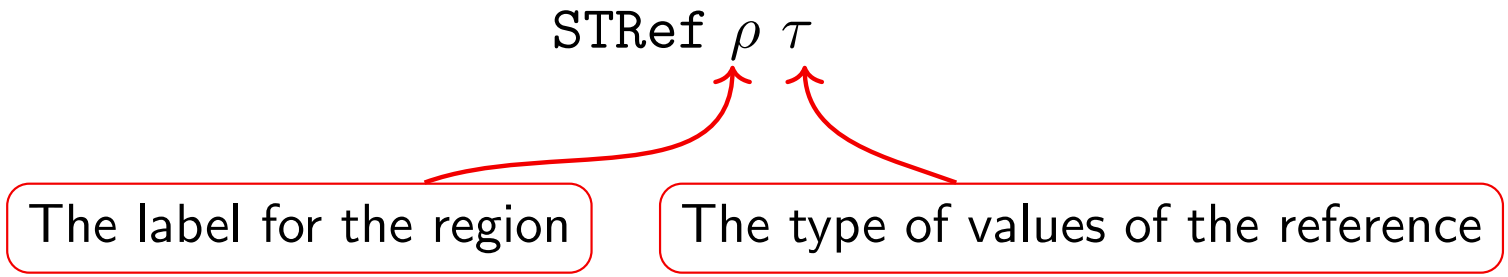


ST Monad (Launchbury and Peyton Jones 1994)

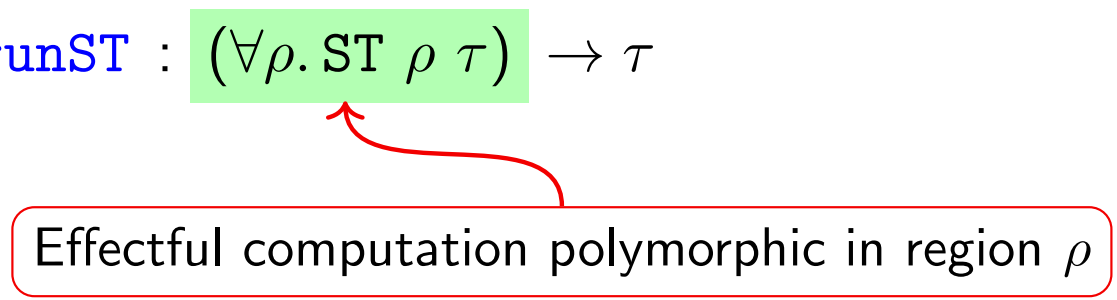
The idea

Encapsulating the effect of using the **heap**

- The idea: label references with regions (also: state threads)



- The function `runST` to *run* computations and produce a value



- Because of polymorphism ρ cannot appear in τ
- Therefore, *effects (references) cannot escape computations*

Implemented in Haskell

ST Monad (forming computations)

Functions to form (**not execute**) effectful computations:

$$\texttt{return} : \tau \to \texttt{ST}\ \rho\ \tau \qquad \text{monadic return}$$

$$\texttt{bind} : \texttt{ST}\ \rho\ \tau_1 \to (\tau_1 \to \texttt{ST}\ \rho\ \tau_2) \to \texttt{ST}\ \rho\ \tau_2 \qquad \text{monadic bind}$$

$$\texttt{ref} : \tau \to \texttt{ST}\ \rho\ (\texttt{STRef}\ \rho\ \tau) \qquad \text{allocation}$$

$$! : \texttt{STRef}\ \rho\ \tau \to \texttt{ST}\ \rho\ \tau \qquad \text{dereference}$$

$$(\leftarrow) : \texttt{STRef}\ \rho\ \tau \to \tau \to \texttt{ST}\ \rho\ \tau \qquad \text{write}$$

Non-effectful reference comparison function

$$(==) : \texttt{STRef}\ \rho\ \tau \to \texttt{STRef}\ \rho\ \tau \to \mathbb{B} \qquad \text{reference comparison}$$

In this talk

Justifying ST properly encapsulates state by proving equations (contextual refinements/equivalences)

- that are expected to hold for pure programs
- but fail for effectful programs (e.g., ML programs)
- example : $\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\mathsf{ctx}} (e, e) : \tau \times \tau$

In this talk

Justifying ST properly encapsulates state by proving equations (contextual refinements/equivalences)

- that are expected to hold for pure programs
- but fail for effectful programs (e.g., ML programs)
- example : $\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\text{ctx}} (e, e) : \tau \times \tau$

STLang: A CBV programming language featuring:

- impredicative polymorphism, recursive types
- *labeled* ML-style *higher-order* references and `ST` monad
- Semantics of STLang: effectful programs (manipulation of heap) performed on a **global heap**, closer to actual implementation

In this talk

Justifying ST properly encapsulates state by proving equations (contextual refinements/equivalences)

- that are expected to hold for pure programs
- but fail for effectful programs (e.g., ML programs)
- example : $\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\text{ctx}} (e, e) : \tau \times \tau$

STLang: A CBV programming language featuring:

- impredicative polymorphism, recursive types
- *labeled* ML-style *higher-order* references and `ST` monad
- Semantics of STLang: effectful programs (manipulation of heap) performed on a **global heap**, closer to actual implementation

By defining a binary logical relation model for STLang
In Iris: a state-of-the-art higher-order separation logic

- Designed for program verification and **constructing semantic models of programming languages**

In this talk

Justifying ST properly encapsulates state by proving equations (contextual refinements/equivalences)

- that are expected to hold for pure programs
- but fail for effectful programs (e.g., ML programs)
- example : $\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\text{ctx}} (e, e) : \tau \times \tau$

STLang: A CBV programming language featuring:

- impredicative polymorphism, recursive types
- *labeled* ML-style *higher-order* references and `ST` monad
- Semantics of STLang: effectful programs (manipulation of heap) performed on a **global heap**, closer to actual implementation

By defining a binary logical relation model for STLang
In Iris: a state-of-the-art higher-order separation logic

- Designed for program verification and constructing semantic models of programming languages

I assume no familiarity with Iris or program verification

Example of operational semantics

$\langle \emptyset, (\lambda\, x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(2+1)\, \texttt{in}\, (\lambda\, r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda\, \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda\, x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow$

Example of operational semantics

$\langle \emptyset, (\lambda\, x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(2+1)\, \texttt{in}\, (\lambda\, r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda\, \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda\, x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow \langle \emptyset, (\lambda\, x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda\, r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda\, \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda\, x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow$

Example of operational semantics

$\langle \emptyset, (\lambda\, x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(2+1)\, \texttt{in}\, (\lambda\, r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda\, \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda\, x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow \langle \emptyset, (\lambda\, x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda\, r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda\, \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda\, x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow \langle \emptyset, \texttt{runST}\ \{\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda\, r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda\, \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda\, x.\, \texttt{return}\, x)))\} \rangle$

$\rightarrow$

Example of operational semantics

$\langle \emptyset, (\lambda x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(2+1)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow \langle \emptyset, (\lambda x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))) \rangle$

$\rightarrow \langle \emptyset, \texttt{runST}\ \{\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))\} \rangle$

$\rightarrow \langle [l \mapsto 3], \texttt{runST}\ \{\texttt{bind}\, (\texttt{return}\, l)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))\} \rangle$

$\rightarrow$

Example of operational semantics

$$\langle \emptyset, (\lambda x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(2+1)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))) \rangle$$

$$\rightarrow \langle \emptyset, (\lambda x.\, \texttt{runST}\ \{x\})\ (\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))) \rangle$$

$$\rightarrow \langle \emptyset, \texttt{runST}\ \{\texttt{bind}\, \texttt{ref}(3)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))\} \rangle$$

$$\rightarrow \langle [l \mapsto 3], \texttt{runST}\ \{\texttt{bind}\, (\texttt{return}\, l)\, \texttt{in}\, (\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)))\} \rangle$$

$$\rightarrow \langle [l \mapsto 3], \texttt{runST}\ \{(\lambda r.\, \texttt{bind}\, (r \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, r\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x))\ l\} \rangle$$

$$\rightarrow \langle [l \mapsto 3], \texttt{runST}\ \{\texttt{bind}\, (l \leftarrow 7)\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, l\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], \texttt{runST}\ \{\texttt{bind}\, (\texttt{return}\, ())\, \texttt{in}\, (\lambda \_.\, \texttt{bind}\, !\, l\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], \texttt{runST}\ \{(\lambda \_.\, \texttt{bind}\, !\, l\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)\ ()\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], \texttt{runST}\ \{\texttt{bind}\, !\, l\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], \texttt{runST}\ \{\texttt{bind}\, (\texttt{return}\, 7)\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x)\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], \texttt{runST}\ \{(\lambda x.\, \texttt{return}\, x)\ 7\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], \texttt{runST}\ \{\texttt{return}\, 7\} \rangle$$

$$\rightarrow \langle [l \mapsto 7], 7 \rangle$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$
$$\langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell' \mapsto 3]; [\ell \mapsto 2], \texttt{runST}\ \{\texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$
$$\langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$
$$\langle [\ell' \mapsto 3]; [\ell \mapsto 2], \texttt{runST}\ \{\texttt{runST}\ \{\texttt{return}\ ()\}\};\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\rightarrow^* \langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ ()\};\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell' \mapsto 3]; [\ell \mapsto 2], \texttt{runST}\ \{\texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ ()\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell' \mapsto 3]; [\ell \mapsto 2], \texttt{runST}\ \{\texttt{runST}\ \{\texttt{return}\ ()\}\};\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ ()\};\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle [\ell'' \mapsto 4], \texttt{runST}\ \{\texttt{return}\ ()\}\rangle$$

Related work (E.g., Semmelroth and Sabry 1999; Moggi and Sabry 2001)

The challenge: different semantics

Consider the program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In previous work: a stack of stores was used for the semantics:

$$\langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle [\ell' \mapsto 3]; [\ell \mapsto 2], \texttt{runST}\ \{\texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle [\ell \mapsto 2], \texttt{runST}\ \{\texttt{return}\ ()\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle [\ell'' \mapsto 4], \texttt{runST}\ \{\texttt{return}\ ()\}\rangle$$
$$\to^* \langle \cdot, () \rangle$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \to^*$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

$$\langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

$$\langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

$$\rightarrow^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ ()\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ ()\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\}\,;$$
$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\,\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\}\,;\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\}\,;\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle \to^*$$
$$\langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{return}\ ()\}\}\,;\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ ()\}\,;\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\,\_.\ \texttt{return}\ ()\}\rangle$$
$$\to^* \langle \{\ell'' \mapsto 4, \ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ ()\}\rangle$$

Related work

The challenge: different semantics

Consider the same program

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}$$

In this work: a global heap for the semantics:

$$\langle \emptyset, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(2)\ \texttt{in}\ \lambda\ell.\ \texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

$$\langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3)\ \texttt{in}\ \texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle \rightarrow^*$$

$$\langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ \texttt{runST}\ \{\texttt{return}\ ()\}\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

$$\rightarrow^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ ()\};\ \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

$$\rightarrow^* \langle \{\ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda\_.\ \texttt{return}\ ()\}\rangle$$

$$\rightarrow^* \langle \{\ell'' \mapsto 4, \ell' \mapsto 3, \ell \mapsto 2\}, \texttt{runST}\ \{\texttt{return}\ ()\}\rangle$$

$$\rightarrow^* \langle \{\ell'' \mapsto 4, \ell' \mapsto 3, \ell \mapsto 2\}, ()\rangle$$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda \_.\, \texttt{return}\, ()\}$$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$$
$$\langle [\ell \mapsto 2], \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$
$\langle [\ell \mapsto 2], \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow \mathit{err}$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$
$\langle [\ell \mapsto 2], \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow \mathit{err}$

Using a global heap:

$\langle \emptyset, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \texttt{return}\, ()\}\rangle$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$
$\langle [\ell \mapsto 2], \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow \mathit{err}$

Using a global heap:

$\langle \emptyset, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 2\}, \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$
$\langle [\ell \mapsto 2], \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow \mathit{err}$

Using a global heap:

$\langle \emptyset, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 2\}, \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$

Related work

The challenge: different semantics

Consider the **ill-typed** program

$$\texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}$$

Using a stack of stores:

$\langle \cdot, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow^*$
$\langle [\ell \mapsto 2], \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \cdot, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle \rightarrow \mathit{err}$

Using a global heap:

$\langle \emptyset, \texttt{let}\, x = \texttt{runST}\ \{\texttt{ref}(2)\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 2\}, \texttt{let}\, x = \texttt{runST}\ \{\texttt{return}\, \ell\}\ \texttt{in}\ \texttt{runST}\ \{\texttt{bind}\, x \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 2\}, \texttt{runST}\ \{\texttt{bind}\, \ell \leftarrow 3\, \texttt{in}\, \lambda\_.\, \texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 3\}, \texttt{runST}\ \{\texttt{return}\, ()\}\rangle$
$\rightarrow^* \langle \{\ell \mapsto 3\}, ()\rangle$

Related work

The challenge: proving equations

Previous work:

- Prove type safety
- Which is a strong result given that programs that violate encapsulation produce errors

(Moggi and Sabry 2001) in the conclusion part:

"On the other hand, substantially more work is needed to establish soundness of equational reasoning w.r.t. our dynamic semantics (even for something as unsurprising as β-equivalence)."

Contextual refinement and equivalence

Contextual equivalence defined through contextual refinement:

$$\Xi \mid \Gamma \vDash e \approx_{\mathsf{ctx}} e' : \tau \triangleq (\Xi \mid \Gamma \vDash e \preceq_{\mathsf{ctx}} e' : \tau) \wedge (\Xi \mid \Gamma \vDash e' \preceq_{\mathsf{ctx}} e : \tau)$$

Contextual refinement and equivalence

Contextual equivalence defined through contextual refinement:

$$\Xi \mid \Gamma \vDash e \approx_{\text{ctx}} e' : \tau \triangleq (\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau) \wedge (\Xi \mid \Gamma \vDash e' \preceq_{\text{ctx}} e : \tau)$$

Contextual refinement (idea): Two programs are equal if no context can distinguish them

- Usually (for a programming language with heap):

$$\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau \triangleq \Xi \mid \Gamma \vdash e : \tau \wedge \Xi \mid \Gamma \vdash e' : \tau \wedge$$
$$\forall C.\ C : (\Xi \mid \Gamma; \tau) \rightsquigarrow (\cdot \mid \cdot; 1) \wedge \langle \emptyset, C[e] \rangle \downarrow \Rightarrow \langle \emptyset, C[e'] \rangle \downarrow$$

Contextual refinement and equivalence

Contextual equivalence defined through contextual refinement:

$$\Xi \mid \Gamma \vDash e \approx_{\text{ctx}} e' : \tau \triangleq (\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau) \wedge (\Xi \mid \Gamma \vDash e' \preceq_{\text{ctx}} e : \tau)$$

Contextual refinement (idea): Two programs are equal if no context can distinguish them

- Usually (for a programming language with heap):

$$\cancel{\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau \triangleq \Xi \mid \Gamma \vdash e : \tau \wedge \Xi \mid \Gamma \vdash e' : \tau \wedge}$$
$$\cancel{\forall C.\ C : (\Xi \mid \Gamma; \tau) \rightsquigarrow (\cdot \mid \cdot; 1) \wedge \langle \emptyset, C[e] \rangle \downarrow \Rightarrow \langle \emptyset, C[e'] \rangle \downarrow}$$

- Our definition:

$$\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau \triangleq \Xi \mid \Gamma \vdash e : \tau \wedge \Xi \mid \Gamma \vdash e' : \tau \wedge$$
$$\forall\, \text{h}, \text{h'}, C.\ C : (\Xi \mid \Gamma; \tau) \rightsquigarrow (\cdot \mid \cdot; 1) \wedge \left\langle \text{h}, C[e] \right\rangle \downarrow \Rightarrow \left\langle \text{h'}, C[e'] \right\rangle \downarrow$$

Contextual refinement and equivalence

Contextual equivalence defined through contextual refinement:

$$\Xi \mid \Gamma \vDash e \approx_{\text{ctx}} e' : \tau \triangleq (\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau) \wedge (\Xi \mid \Gamma \vDash e' \preceq_{\text{ctx}} e : \tau)$$

Contextual refinement (idea): Two programs are equal if no context can distinguish them

- Usually (for a programming language with heap):

$$\cancel{\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau \triangleq \Xi \mid \Gamma \vdash e : \tau \wedge \Xi \mid \Gamma \vdash e' : \tau \wedge}$$
$$\cancel{\forall C.\ C : (\Xi \mid \Gamma; \tau) \rightsquigarrow (\cdot \mid \cdot; 1) \wedge \langle \emptyset, C[e] \rangle \downarrow \Rightarrow \langle \emptyset, C[e'] \rangle \downarrow}$$

- Our definition:

$$\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau \triangleq \Xi \mid \Gamma \vdash e : \tau \wedge \Xi \mid \Gamma \vdash e' : \tau \wedge$$
$$\forall \text{ h }, \text{ h' }, C.\ C : (\Xi \mid \Gamma; \tau) \rightsquigarrow (\cdot \mid \cdot; 1) \wedge \left\langle \text{ h }, C[e] \right\rangle \downarrow \Rightarrow \left\langle \text{ h' }, C[e'] \right\rangle \downarrow$$

Programs are independent of heap!

- We prove that $\preceq_{\text{ctx}}$ is reflexive which means
- $\cdot \mid \cdot \vdash e : 1$ implies $(\exists h.\ \langle h, e \rangle \downarrow) \Longrightarrow \forall h'.\ \langle h', e \rangle \downarrow$

Proven Equations

$$e \preceq_{\mathsf{ctx}} () : 1 \qquad \text{(Neutrality)}$$

$$\texttt{let}\, x = e_2\, \texttt{in}\, (e_1, x) \approx_{\mathsf{ctx}} (e_1, e_2) : \tau_1 \times \tau_2 \qquad \text{(Commutativity)}$$

$$\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\mathsf{ctx}} (e, e) : \tau \times \tau \qquad \text{(Idempotency)}$$

Proven Equations

$$e \preceq_{\mathsf{ctx}} () : 1 \qquad \text{(Neutrality)}$$

$$\texttt{let}\, x = e_2\, \texttt{in}\, (e_1, x) \approx_{\mathsf{ctx}} (e_1, e_2) : \tau_1 \times \tau_2 \qquad \text{(Commutativity)}$$

$$\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\mathsf{ctx}} (e, e) : \tau \times \tau \qquad \text{(Idempotency)}$$

$$\texttt{let}\, y = e_1\, \texttt{in}\, \texttt{rec}\, f(x) = e_2 \preceq_{\mathsf{ctx}} \texttt{rec}\, f(x) = \texttt{let}\, y = e_1\, \texttt{in}\, e_2 : \tau_1 \to \tau_2$$
(Rec hoisting)

$$\texttt{let}\, y = e_1\, \texttt{in}\, \Lambda\, e_2 \preceq_{\mathsf{ctx}} \Lambda\, (\texttt{let}\, y = e_1\, \texttt{in}\, e_2) : \forall X.\, \tau \qquad (\Lambda \text{ hoisting})$$

Proven Equations

$$e \preceq_{\mathsf{ctx}} () : 1 \qquad \text{(Neutrality)}$$

$$\texttt{let}\, x = e_2\, \texttt{in}\, (e_1, x) \approx_{\mathsf{ctx}} (e_1, e_2) : \tau_1 \times \tau_2 \qquad \text{(Commutativity)}$$

$$\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\mathsf{ctx}} (e, e) : \tau \times \tau \qquad \text{(Idempotency)}$$

$$\texttt{let}\, y = e_1\, \texttt{in}\, \texttt{rec}\, f(x) = e_2 \preceq_{\mathsf{ctx}} \texttt{rec}\, f(x) = \texttt{let}\, y = e_1\, \texttt{in}\, e_2 : \tau_1 \to \tau_2$$
(Rec hoisting)

$$\texttt{let}\, y = e_1\, \texttt{in}\, \Lambda\, e_2 \preceq_{\mathsf{ctx}} \Lambda\, (\texttt{let}\, y = e_1\, \texttt{in}\, e_2) : \forall X.\, \tau \qquad (\Lambda \text{ hoisting})$$

$$e \preceq_{\mathsf{ctx}} \texttt{rec}\, f(x) = (e\ x) : \tau_1 \to \tau_2$$
(η expansion for rec)

$$e \preceq_{\mathsf{ctx}} \Lambda\, (e\ \_) : \forall X.\, \tau \qquad (\eta \text{ expansion for } \Lambda)$$

Proven Equations

$$e \preceq_{\mathsf{ctx}} () : 1 \qquad \text{(Neutrality)}$$

$$\texttt{let}\, x = e_2\, \texttt{in}\, (e_1, x) \approx_{\mathsf{ctx}} (e_1, e_2) : \tau_1 \times \tau_2 \qquad \text{(Commutativity)}$$

$$\texttt{let}\, x = e\, \texttt{in}\, (x, x) \approx_{\mathsf{ctx}} (e, e) : \tau \times \tau \qquad \text{(Idempotency)}$$

$$\texttt{let}\, y = e_1\, \texttt{in}\, \texttt{rec}\, f(x) = e_2 \preceq_{\mathsf{ctx}} \texttt{rec}\, f(x) = \texttt{let}\, y = e_1\, \texttt{in}\, e_2 : \tau_1 \to \tau_2$$
(Rec hoisting)

$$\texttt{let}\, y = e_1\, \texttt{in}\, \Lambda\, e_2 \preceq_{\mathsf{ctx}} \Lambda\, (\texttt{let}\, y = e_1\, \texttt{in}\, e_2) : \forall X.\, \tau \quad (\Lambda \text{ hoisting})$$

$$e \preceq_{\mathsf{ctx}} \texttt{rec}\, f(x) = (e\ x) : \tau_1 \to \tau_2$$
(η expansion for rec)

$$e \preceq_{\mathsf{ctx}} \Lambda\, (e\ \_) : \forall X.\, \tau \qquad (\eta \text{ expansion for } \Lambda)$$

$$(\texttt{rec}\, f(x) = e_1)\ e_2 \preceq_{\mathsf{ctx}} e_1[e_2, (\texttt{rec}\, f(x) = e_1)/x, f] : \tau$$
(β reduction for rec)

$$(\Lambda\, e)\ \_ \preceq_{\mathsf{ctx}} e : \tau' \qquad (\beta \text{ reduction for } \Lambda)$$

Binary logical relations

To establish contextual refinement

The idea: Use a binary logical relation such that being related implies contextual refinement

Binary logical relations

To establish contextual refinement

The idea: Use a binary logical relation such that being related implies contextual refinement

Define *semantics* of types (by recursion on τ):
$\mathcal{E}[\![\tau]\!] : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

Binary logical relations

To establish contextual refinement

The idea: Use a binary logical relation such that being related implies contextual refinement

Define *semantics* of types (by recursion on τ):
$\mathcal{E}[\![\tau]\!] : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

1. Prove *Adequacy*: $\mathcal{E}[\![\tau]\!](e, e') \Rightarrow \forall h, h'.\ \langle h, e\rangle\downarrow \Rightarrow \langle h', e'\rangle\downarrow$

Binary logical relations

To establish contextual refinement

The idea: Use a binary logical relation such that being related implies contextual refinement

Define *semantics* of types (by recursion on τ):
$\mathcal{E}[\![\tau]\!] : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

1. Prove *Adequacy*: $\mathcal{E}[\![\tau]\!](e, e') \Rightarrow \forall h, h'.\ \langle h, e\rangle\downarrow \Rightarrow \langle h', e'\rangle\downarrow$
2. Prove *compatibility* lemmas for typing rules, e.g.:

$$\frac{\mathcal{E}[\![\tau_1]\!](e_1, e_1') \qquad \mathcal{E}[\![\tau]\!](e_2, e_2')}{\mathcal{E}[\![\tau_1 \times \tau_2]\!]((e_1, e_2), (e_1', e_2'))}$$

Binary logical relations

To establish contextual refinement

The idea: Use a binary logical relation such that being related implies contextual refinement

Define *semantics* of types (by recursion on τ):
$\mathcal{E}[\![\tau]\!] : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

1. Prove *Adequacy*: $\mathcal{E}[\![\tau]\!](e, e') \Rightarrow \forall h, h'.\ \langle h, e\rangle\downarrow \Rightarrow \langle h', e'\rangle\downarrow$
2. Prove *compatibility* lemmas for typing rules, e.g.:

$$\frac{\mathcal{E}[\![\tau_1]\!](e_1, e_1') \qquad \mathcal{E}[\![\tau]\!](e_2, e_2')}{\mathcal{E}[\![\tau_1 \times \tau_2]\!]((e_1, e_2), (e_1', e_2'))}$$

3. Corollary (*soundness*):
   $\Xi \mid \Gamma \vDash e \preceq_{\mathsf{log}} e' : \tau \Rightarrow \Xi \mid \Gamma \vdash e \preceq_{\mathsf{ctx}} e' : \tau$

Why use Iris?

We do not need to construct a model!

$$
\begin{array}{rcl}
\mathrm{HeapAtom}_n & \stackrel{\text{def}}{=} & \{(W,h_1,h_2) \mid W \in \mathrm{World}_n\} \\
\mathrm{HeapRel}_n & \stackrel{\text{def}}{=} & \{\psi \subseteq \mathrm{HeapAtom}_n \mid \forall (W,h_1,h_2) \in \psi.\ \forall W' \sqsupseteq W.\ (W',h_1,h_2) \in \psi\} \\
\mathrm{Island}_n & \stackrel{\text{def}}{=} & \{\iota = (s,\delta,\varphi,\natural,H) \mid s \in \mathrm{State} \wedge \delta \subseteq \mathrm{State}^2 \wedge \varphi \subseteq \delta \wedge \delta,\varphi \text{ reflexive} \wedge \\
 & & \quad \delta,\varphi \text{ transitive} \wedge \natural \subseteq \mathrm{State} \wedge H \in \mathrm{State} \to \mathrm{HeapRel}_n\} \\
\mathrm{World}_n & \stackrel{\text{def}}{=} & \{W = (k,\Sigma_1,\Sigma_2,\omega) \mid k < n \wedge \exists m.\ \omega \in \mathrm{Island}_k^m\} \\
\mathrm{ContAtom}_n[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \{(W,K_1,K_2) \mid W \in \mathrm{World}_n \wedge W.\Sigma_1;\cdot;\cdot \vdash K_1 \div \tau_1 \wedge W.\Sigma_2;\cdot;\cdot \vdash K_2 \div \tau_2\} \\
\mathrm{TermAtom}_n[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \{(W,e_1,e_2) \mid W \in \mathrm{World}_n \wedge W.\Sigma_1;\cdot;\cdot \vdash e_1 : \tau_1 \wedge W.\Sigma_2;\cdot;\cdot \vdash e_2 : \tau_2\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{HeapAtom}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{HeapAtom}_n[\tau_1,\tau_2] \\
\mathrm{World} & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{World}_n \\
\mathrm{ContAtom}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{ContAtom}_n[\tau_1,\tau_2] \\
\mathrm{TermAtom}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{TermAtom}_n[\tau_1,\tau_2]
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{ValRel}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \{r \subseteq \mathrm{TermAtom}^{\mathsf{val}}[\tau_1,\tau_2] \mid \forall (W,v_1,v_2) \in r.\ \forall W' \sqsupseteq W.\ (W',v_1,v_2) \in r\} \\
\mathrm{SomeValRel} & \stackrel{\text{def}}{=} & \{R = (\tau_1,\tau_2,r) \mid r \in \mathrm{ValRel}[\tau_1,\tau_2]\}
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
\lfloor(\iota_1,\ldots,\iota_m)\rfloor_k & \stackrel{\text{def}}{=} & (\lfloor\iota_1\rfloor_k,\ldots,\lfloor\iota_m\rfloor_k) & \lfloor H\rfloor_k & \stackrel{\text{def}}{=} & \lambda s.\lfloor H(s)\rfloor_k \\
\lfloor(s,\delta,\varphi,\natural,H)\rfloor_k & \stackrel{\text{def}}{=} & (s,\delta,\varphi,\natural,\lfloor H\rfloor_k) & \lfloor\psi\rfloor_k & \stackrel{\text{def}}{=} & \{(W,h_1,h_2) \in r \mid W.k < k\}
\end{array}
$$

$$
\begin{array}{rcl}
\triangleright(k+1,\Sigma_1,\Sigma_2,\omega) & \stackrel{\text{def}}{=} & (k,\Sigma_1,\Sigma_2,\lfloor\omega\rfloor_k) \\
\triangleright r & \stackrel{\text{def}}{=} & \{(W,e_1,e_2) \mid W.k > 0 \Rightarrow (\triangleright W,e_1,e_2) \in r\}
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
(k',\Sigma_1',\Sigma_2',\omega') & \sqsupseteq & (k,\Sigma_1,\Sigma_2,\omega) & \stackrel{\text{def}}{=} & k' \le k \wedge \Sigma_1' \supseteq \Sigma_1 \wedge \Sigma_2' \supseteq \Sigma_2 \wedge \omega' \sqsupseteq \lfloor\omega\rfloor_{k'} \\
(\iota_1',\ldots,\iota_{m'}') & \sqsupseteq & (\iota_1,\ldots,\iota_m) & \stackrel{\text{def}}{=} & m' \ge m \wedge \forall j \in \{1,\ldots,m\}.\ \iota_j' \sqsupseteq \iota_j \\
(s',\delta',\varphi',\natural',H') & \sqsupseteq & (s,\delta,\varphi,\natural,H) & \stackrel{\text{def}}{=} & (\delta',\varphi',\natural',H') = (\delta,\varphi,\natural,H) \wedge (s,s') \in \delta
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
(k',\Sigma_1',\Sigma_2',\omega') & \sqsupseteq^{\mathsf{pub}} & (k,\Sigma_1,\Sigma_2,\omega) & \stackrel{\text{def}}{=} & k' \le k \wedge \Sigma_1' \supseteq \Sigma_1 \wedge \Sigma_2' \supseteq \Sigma_2 \wedge \omega' \sqsupseteq^{\mathsf{pub}} \lfloor\omega\rfloor_{k'} \\
(\iota_1',\ldots,\iota_{m'}') & \sqsupseteq^{\mathsf{pub}} & (\iota_1,\ldots,\iota_m) & \stackrel{\text{def}}{=} & m' \ge m \wedge \forall j \in \{1,\ldots,m\}.\ \iota_j' \sqsupseteq^{\mathsf{pub}} \iota_j \wedge \\
 & & & & & \forall j \in \{m+1,\ldots,m'\}.\ \mathrm{safe}(\iota_j') \\
(s',\delta',\varphi',\natural',H') & \sqsupseteq^{\mathsf{pub}} & (s,\delta,\varphi,\natural,H) & \stackrel{\text{def}}{=} & (\delta',\varphi',\natural',H') = (\delta,\varphi,\natural,H) \wedge (s,s') \in \varphi
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
\mathrm{safe}(W) & \stackrel{\text{def}}{=} & \forall \iota \in W.\omega.\ \mathrm{safe}(\iota) & \mathrm{safe}(\iota) & \stackrel{\text{def}}{=} & \forall s'.\ (\iota.s,s') \in \iota.\varphi \Rightarrow s' \notin \iota.\natural
\end{array}
$$

$$
\mathrm{consistent}(W) \stackrel{\text{def}}{=} \nexists \iota \in W.\omega.\ \iota.s \in \iota.\natural
$$

$$
\psi \otimes \psi' \stackrel{\text{def}}{=} \{(W,h_1 \uplus h_1',h_2 \uplus h_2') \mid (W,h_1,h_2) \in \psi \wedge (W,h_1',h_2') \in \psi'\}
$$

$$
(h_1,h_2) : W \stackrel{\text{def}}{=} \vdash h_1 : W.\Sigma_1 \wedge \vdash h_2 : W.\Sigma_2 \wedge (W.k > 0 \Rightarrow (\triangleright W,h_1,h_2) \in \textstyle\bigotimes\{\iota.H(\iota.s) \mid \iota \in W.\omega\})
$$

Fig. 5. Worlds and auxiliary definitions

Using Iris to construct logical relation

Define in two steps

- Value relation for types $[\![\tau]\!]_\Delta : \mathit{Val} \to \mathit{Val} \to \mathit{iProp}$
- Expression relation $\mathcal{E}\,[\![\tau]\!]_\Delta : \mathit{Val} \to \mathit{Val} \to \mathit{iProp}$
  - based on $[\![\tau]\!]_\Delta$
  - intuitively: $\mathcal{E}\,[\![\tau]\!]_\Delta\,(e, e')$ if whenever $(h, e) \to^* v$ then $(h', e') \to^* v'$ such that $[\![\tau]\!]_\Delta(v, v')$
- $\Delta : \mathit{Tvar} \to (((\mathit{Val} \times \mathit{Val}) \to \mathit{iProp}) \times \{\tau \in \mathit{Types} \mid \mathrm{FV}(\tau) = \emptyset\})$

Value interpretations

except `ST` and `STRef`

$$[\![X]\!]_\Delta \triangleq (\Delta(X)).1$$

Value interpretations

except ST and STRef

$$[\![X]\!]_\Delta \triangleq (\Delta(X)).1$$

$$[\![\mathbb{B}]\!]_\Delta(v, v') \triangleq v = v' \in \{\texttt{true}, \texttt{false}\}$$

Value interpretations

except ST and STRef

$$[\![X]\!]_\Delta \triangleq (\Delta(X)).1$$

$$[\![\mathbb{B}]\!]_\Delta(v, v') \triangleq v = v' \in \{\texttt{true}, \texttt{false}\}$$

$$[\![\tau \times \tau]\!]_\Delta(v, v') \triangleq \exists w_1, w_2, w_1', w_2'.\ v = (w_1, w_2) \wedge v' = (w_1', w_2') \wedge [\![\tau]\!]_\Delta(w_1, w_1') \wedge [\![\tau']\!]_\Delta(w_2, w_2')$$

Value interpretations

except ST and STRef

$$[\![X]\!]_\Delta \triangleq (\Delta(X)).1$$

$$[\![\mathbb{B}]\!]_\Delta(v, v') \triangleq v = v' \in \{\texttt{true}, \texttt{false}\}$$

$$[\![\tau \times \tau]\!]_\Delta(v, v') \triangleq \exists w_1, w_2, w_1', w_2'.\ v = (w_1, w_2) \wedge v' = (w_1', w_2') \wedge [\![\tau]\!]_\Delta(w_1, w_1') \wedge [\![\tau']\!]_\Delta(w_2, w_2')$$

$$[\![\tau \to \tau']\!]_\Delta(v, v') \triangleq \Box\left(\forall(w, w').\ [\![\tau]\!]_\Delta(w, w') \Rightarrow \mathcal{E}\,[\![\tau]\!]_\Delta(v\ w, v'\ w')\right)$$

Value interpretations

except ST and STRef

$$[\![X]\!]_\Delta \triangleq (\Delta(X)).1$$

$$[\![\mathbb{B}]\!]_\Delta(v, v') \triangleq v = v' \in \{\texttt{true}, \texttt{false}\}$$

$$[\![\tau \times \tau]\!]_\Delta(v, v') \triangleq \exists w_1, w_2, w_1', w_2'.\ v = (w_1, w_2) \wedge v' = (w_1', w_2') \wedge [\![\tau]\!]_\Delta(w_1, w_1') \wedge [\![\tau']\!]_\Delta(w_2, w_2')$$

$$[\![\tau \to \tau']\!]_\Delta(v, v') \triangleq \Box\left(\forall (w, w').\ [\![\tau]\!]_\Delta(w, w') \Rightarrow \mathcal{E}\,[\![\tau]\!]_\Delta\,(v\ w, v'\ w')\right)$$

$$[\![\forall X.\,\tau]\!]_\Delta(v, v') \triangleq \Box\left(\forall f, \tau'.\ \mathsf{FV}(\tau') = \emptyset \wedge \mathsf{persistent}(f) \Rightarrow \mathcal{E}\,[\![\Xi, X \vdash \tau]\!]_{\Delta, X \mapsto (f, \tau')}\,(v\ \_, v'\ \_)\right)$$

Value interpretations

except ST and STRef

$$[\![X]\!]_\Delta \triangleq (\Delta(X)).1$$

$$[\![\mathbb{B}]\!]_\Delta(v, v') \triangleq v = v' \in \{\texttt{true}, \texttt{false}\}$$

$$[\![\tau \times \tau]\!]_\Delta(v, v') \triangleq \exists w_1, w_2, w_1', w_2'.\ v = (w_1, w_2) \wedge v' = (w_1', w_2') \wedge [\![\tau]\!]_\Delta(w_1, w_1') \wedge [\![\tau']\!]_\Delta(w_2, w_2')$$

$$[\![\tau \to \tau']\!]_\Delta(v, v') \triangleq \Box\left(\forall(w, w').\ [\![\tau]\!]_\Delta(w, w') \Rightarrow \mathcal{E}\,[\![\tau]\!]_\Delta(v\ w, v'\ w')\right)$$

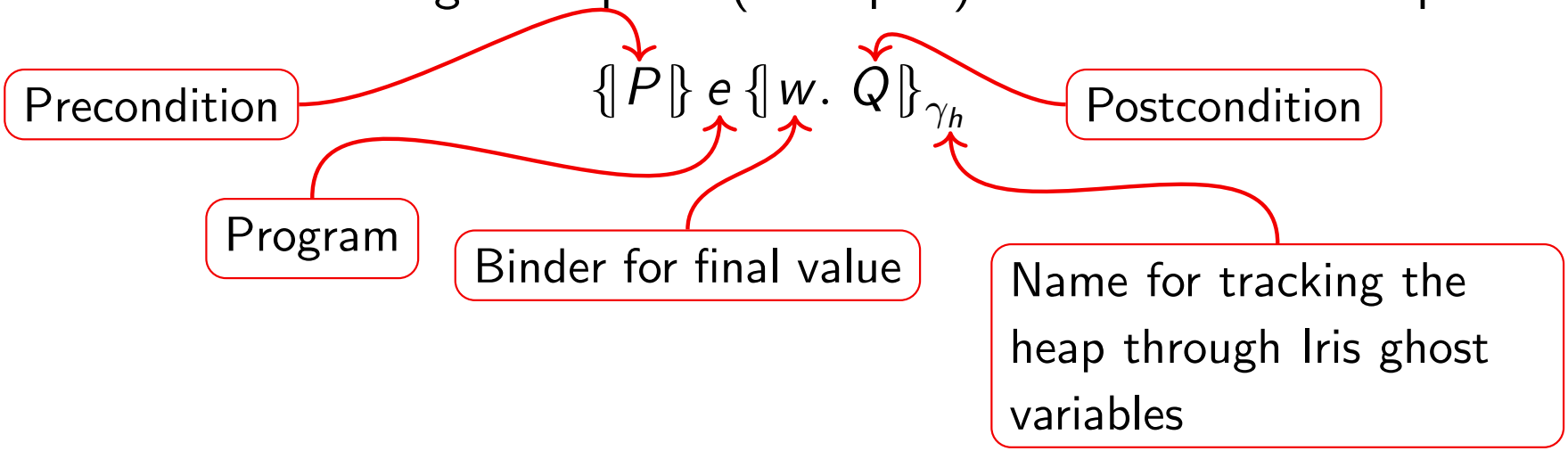
$$[\![\forall X.\,\tau]\!]_\Delta(v, v') \triangleq \Box\left(\forall f, \tau'.\ \mathsf{FV}(\tau') = \emptyset \wedge \mathsf{persistent}(f) \Rightarrow \mathcal{E}\,[\![\Xi, X \vdash \tau]\!]_{\Delta, X \mapsto (f, \tau')}(v\ \_, v'\ \_)\right)$$

$$[\![\mu X.\,\tau]\!]_\Delta(v, v') \triangleq \mu f.\ \exists w, w'.\ v = \texttt{fold}\, w \wedge v' = \texttt{fold}\, w' \wedge \triangleright [\![\Xi, X \vdash \tau]\!]_{\Delta, X \mapsto (f, \mathsf{close}(\Delta, \mu X.\,\tau))}(w, w')$$

IC-triples

Used for defining the expression relation

We use "If Convergent triples" (IC-triples) similar to Hoare-triples

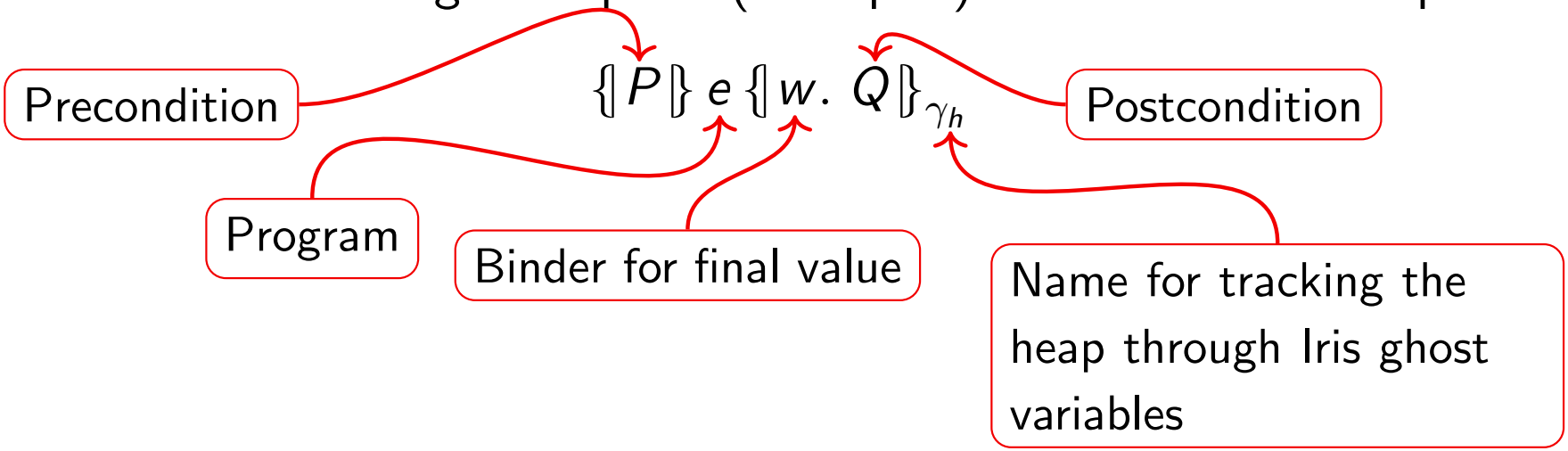


Intuitively: if P holds **and** e **converges** to a value w then Q holds.

IC-triples

Used for defining the expression relation

We use "If Convergent triples" (IC-triples) similar to Hoare-triples



Intuitively: if P holds **and** e **converges** to a value w then Q holds.

Examples:

Iris ghost variable: evidence of the value stored in ℓ in the heap tracked by γ_h

$$\{\!|\, \texttt{true} \,|\!\}\ \texttt{runST}\ \{\texttt{ref}(v)\}\ \left\{\!\left|\, \ell.\ \boxed{\circ\, \ell \mapsto v}^{\gamma_h} \,\right|\!\right\}_{\gamma_h}$$

$$\left\{\!\left|\, \boxed{\circ\, \ell \mapsto v}^{\gamma_h} \,\right|\!\right\}\ \texttt{runST}\ \{\ell \leftarrow u\}\ \left\{\!\left|\, w.\ w = () * \boxed{\circ\, \ell \mapsto u}^{\gamma_h} \,\right|\!\right\}_{\gamma_h}$$

The expression relation

Iris ghost variable: the heaped tracked by γ_h

Lemma: $\boxed{\bullet\, h}^{\gamma_h'} * \boxed{\circ\, \ell \mapsto v}^{\gamma_h'} \vdash \exists h_2.\ h = h_2 \uplus \{(\ell, v)\}$

Lemma: $\boxed{\circ\, \ell \mapsto v}^{\gamma_h'} * \boxed{\circ\, \ell \mapsto v'}^{\gamma_h'} \vdash \texttt{false}$

$$\mathcal{E}\,[\![\tau]\!]_\Delta\,(e, e') \triangleq \forall \gamma_h, \gamma_h', h_1'.\ \left\{ \boxed{\bullet\, h_1'}^{\gamma_h'} * \text{regions} \right\} e \left\{ w.\ (h_1', e') \Downarrow^{\gamma_h'}_{[\![\tau]\!]_\Delta(w,\cdot)} \right\}_{\gamma_h}$$

Invariant keeping track of all regions

$$(h_1', e') \Downarrow^{\gamma_h'}_{[\![\tau]\!]_\Delta(w,\cdot)} \triangleq \exists h_2', v'.\ \langle h_1', e' \rangle \to_d^* \langle h_2', v' \rangle * \boxed{\bullet\, h_2'}^{\gamma_h'} * [\![\tau]\!]_\Delta(w, v')$$

The heap tracked by γ_h, $\boxed{\bullet\, h}^{\gamma_h}$, appears in the definition of IC-triples

Value relation for ST

We require, and guarantee at the end, the whole region corresponding to ρ

$$[\![\mathtt{ST}\ \rho\ \tau]\!]_\Delta(v, v') \triangleq \forall \gamma_h, \gamma_h', h_1'.$$

$$\left\{ \left| \bullet \mathsf{excl}(h_1')^{\gamma_h'} * \mathsf{regions} * \mathsf{region}(\mathsf{close}(\Delta, \rho), \gamma_h, \gamma_h') \right| \right\}$$

$$\mathtt{runST}\ \{v\}$$

$$\left\{ \left| w.\ (h_1', \mathtt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\tau]\!]_\Delta(w,\cdot)} * \mathsf{region}(\mathsf{close}(\Delta, \rho), \gamma_h, \gamma_h') \right| \right\}_{\gamma_h}$$

Value relation for STRef and regions

The name $\mathsf{close}(\Delta, \rho)$ associated with $\mathcal{R}$

Lemma: $\mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}) * \mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}') \vdash \mathcal{R} = \mathcal{R}'$

The region

$$[\![\texttt{STRef}\ \rho\ \tau]\!]_\Delta(\ell, \ell') \triangleq \exists \mathcal{R}.\ \mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}) * (\ell, \ell') \in \mathcal{R}_{\mathsf{Bij}} * \mathcal{R}_{\mathsf{Rels}}(\ell, \ell') = [\![\tau]\!]_\Delta$$

The bijection on related locations

The relation that **should** relate ℓ and ℓ'

Value relation for STRef and regions

The name $\mathsf{close}(\Delta, \rho)$ associated with $\mathcal{R}$

Lemma: $\mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}) * \mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}') \vdash \mathcal{R} = \mathcal{R}'$

The region

$$[\![\texttt{STRef}\ \rho\ \tau]\!]_\Delta(\ell, \ell') \triangleq \exists \mathcal{R}.\ \mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}) * (\ell, \ell') \in \mathcal{R}_{\mathsf{Bij}} * \mathcal{R}_{\mathsf{Rels}}(\ell, \ell') = [\![\tau]\!]_\Delta$$

The bijection on related locations

The relation that **should** relate ℓ and ℓ'

$$\mathsf{region}(\rho, \gamma_h, \gamma_h') \triangleq \exists \mathcal{R}.\ \mathsf{RegOf}(\mathsf{close}(\Delta, \rho), \mathcal{R}) * \ast_{(\ell,\ell') \in \mathcal{R}_{\mathsf{Bij}}} \left(\exists v, v'.\ \boxed{\circ\, \ell \mapsto v}^{\gamma_h} * \boxed{\circ\, \ell' \mapsto v'}^{\gamma_h'} * \triangleright \mathcal{R}_{\mathsf{Rels}}(\ell, \ell')(v, v') \right)$$

The regions predicate just keeps track of regions ($\mathcal{R}_{\mathsf{Bij}}$ and $\mathcal{R}_{\mathsf{Rels}}$)

An excerpt of the proof of compatibility lemma for runST

To prove $(\mathcal{E}[\![\tau]\!]_\Delta(\texttt{runST}\ \{v\}, \texttt{runST}\ \{v'\}))$:

$$\left\{\left| \boxed{\bullet\, h_1'}^{\gamma_h'} * \text{regions} \right|\right\}$$

$$\texttt{runST}\ \{v\}$$

$$\left\{\left| w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\tau]\!]_\Delta(w,\cdot)} \right|\right\}_{\gamma_h}$$

An excerpt of the proof of compatibility lemma for runST

To prove $(\mathcal{E}\,[\![\tau]\!]_\Delta\,(\texttt{runST}\ \{v\}, \texttt{runST}\ \{v'\}))$:

$$\left\{\left| \boxed{\bullet\, h_1'}^{\gamma_h'} * \text{regions} \right|\right\}$$

$$\texttt{runST}\ \{v\}$$

$$\left\{\left| w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\tau]\!]_\Delta(w,\cdot)} \right|\right\}_{\gamma_h}$$

We have $([\![\forall\rho.\,\mathtt{ST}\ \rho\ \tau]\!]_\Delta(v, v'))$:

$$\forall f, \tau'.\ \ldots \Rightarrow \left\{\left| \boxed{\bullet\, h_1'}^{\gamma_h'} * \text{regions} * \text{region}(\text{close}((\Delta, \rho \mapsto (f, \tau')), \rho), \gamma_h, \gamma_h') \right|\right\}$$

$$\texttt{runST}\ \{v\}$$

$$\left\{\left| w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\tau]\!]_{\Delta,\rho\mapsto(f,\tau')}(w,\cdot)} * \text{region}(\text{close}((\Delta, \rho \mapsto (f, \tau')), \rho), \gamma_h, \gamma_h') \right|\right\}_{\gamma_h}$$

An excerpt of the proof of compatibility lemma for runST

To prove $(\mathcal{E}[\![\tau]\!]_\Delta(\texttt{runST}\ \{v\}, \texttt{runST}\ \{v'\}))$:

$$\left\{\left| \boxed{\bullet\, h_1'}^{\gamma_h'} * \text{regions} \right|\right\}$$
$$\texttt{runST}\ \{v\}$$
$$\left\{\left| w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\tau]\!]_\Delta(w,\cdot)} \right|\right\}_{\gamma_h}$$

We have $([\![\forall\rho.\, \mathtt{ST}\ \rho\ \tau]\!]_\Delta(v, v'))$:

$$\forall f, \tau'.\ \ldots \Rightarrow \left\{\left| \boxed{\bullet\, h_1'}^{\gamma_h'} * \text{regions} * \text{region}(\text{close}((\Delta, \rho \mapsto (f, \tau')), \rho), \gamma_h, \gamma_h') \right|\right\}$$
$$\texttt{runST}\ \{v\}$$
$$\left\{\left| w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\tau]\!]_{\Delta, \rho\mapsto(f,\tau')}(w,\cdot)} * \text{region}(\text{close}((\Delta, \rho \mapsto (f, \tau')), \rho), \gamma_h, \gamma_h') \right|\right\}_{\gamma_h}$$

Simply **allocate** (inside regions) a new region $\mathcal{R}$ (empty $\mathcal{R}_{\text{Bij}}$) and assign (a new a closed type τ') to it to obtain $\text{RegOf}(\tau', \mathcal{R})$ and $\text{region}(\tau', \gamma_h, \gamma_h')$.

In the draft and the technical appendix

In the draft

- The exact definition of regions, region predicates
- Construction of regions in Iris: $\mathcal{R}_{\mathsf{Bij}}$ and $\mathcal{R}_{\mathsf{Rels}}$
- High level explanation of the proofs of all equations

In the draft and the technical appendix

In the draft

- The exact definition of regions, region predicates
- Construction of regions in Iris: $\mathcal{R}_{\mathsf{Bij}}$ and $\mathcal{R}_{\mathsf{Rels}}$
- High level explanation of the proofs of all equations

In the technical appendix

- The exact proof of all the equations
- Discussion on why we can't use Hoare triples for expression relation as we usually do for logical relations in Iris
- Discussion on the relation between defining logical relations in Iris using IC-triples and the usual recursive world constructions