A Logical Relation for Monadic Encapsulation of State[1]

Proving contextual equivalences in the presence of runST

Amin Timany[2]

imec-Distrinet, KU Leuven

December 7th, 2017

Prosecco, Inria, Paris

[1] To appear in POPL'18. See `http://www.iris-project.org`.

[2] joint work with: Léo Stefanesco (IRIF, Universite Paris Diderot & CNRS), Morten Krogh-Jespersen (Aarhus University) and Lars Birkedal (Aarhus University)

Introduction

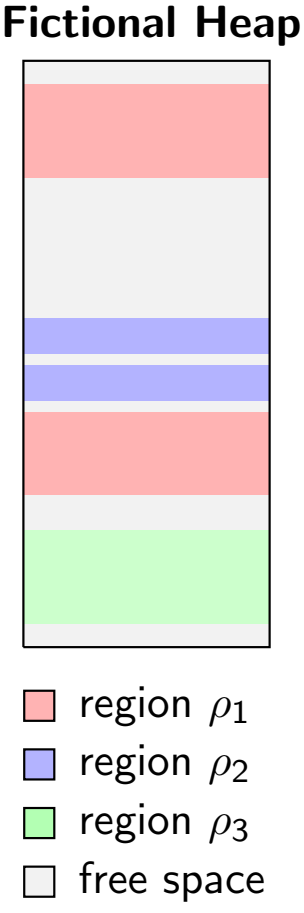
Idea: ST monad (Launchbury and Peyton Jones [1994]):
Using the **type system** to encapsulate effectful computations in a **pure** functional programming language (featured in $\mathrm{Haskell}$), **retaining purity**

Introduction

Idea: ST monad (Launchbury and Peyton Jones [1994]):

Using the **type system** to encapsulate effectful computations in a **pure** functional programming language (featured in $\mathrm{Haskell}$), **retaining purity**

- A family of monads $\mathtt{ST}\ \rho$ for **effectful** computations
  - ρ represents a **fictional disjoint** region of the heap
  - Regions grow **dynamically**
- $\mathtt{ST}\ \rho\ \tau$: the type of effectful **suspended** computations
- Type system ensures that
  - Expressions of type $\mathtt{ST}\ \rho\ \tau$ produce a value of type τ
  - Effects are **restricted** to region ρ of the heap
- `runST` to run encapsulated suspended computations
  - Runs programs starting from **empty** region
  - Types ensures: **no reference** escapes encapsulation

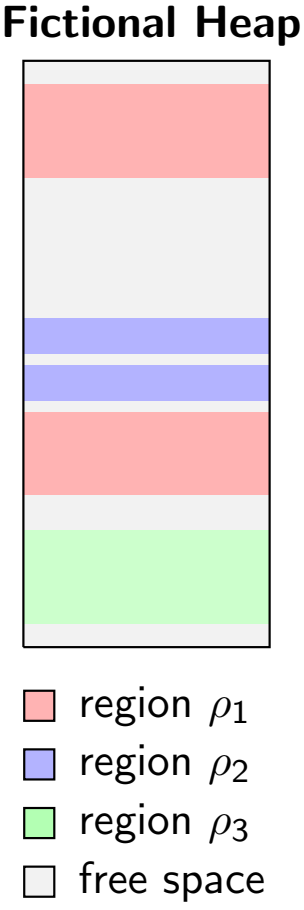


Introduction

Idea: ST monad (Launchbury and Peyton Jones [1994]):
Using the **type system** to encapsulate effectful computations in a **pure** functional programming language (featured in Haskell), **retaining purity**

- A family of monads ST ρ for **effectful** computations
  - ρ represents a **fictional disjoint** region of the heap
  - Regions grow **dynamically**
- ST ρ τ: the type of effectful **suspended** computations
- Type system ensures that
  - Expressions of type ST ρ τ produce a value of type τ
  - Effects are **restricted** to region ρ of the heap
- `runST` to run encapsulated suspended computations
  - Runs programs starting from **empty** region
  - Types ensures: **no reference** escapes encapsulation

In this work we formally (in Coq) justify that the purity is retained by the ST monad and `runST`.



Introduction

In particular, we

- Consider STLang: a call-by-value programming language with
  - Impredicative polymorphism, recursive types and the ST monad and `runST`
  - Higher-order mutable heap
    - **Dynamically** allocated references with **destructive** updates
    - Using the ST monad for encapsulating heap effects
    - Close to the actual implementation of ST monad in practice
- Construct a logical relations model for STLang on top of the Iris base logic
- Prove the state-independence theorem
  - Intuitively, well-typed programs are heap independent
- Prove that contextual equations hold for all well-typed programs of STLang that only hold for **pure** programs

We use Iris because

- We can avoid the well-known problems, e.g., type-world circularity, when reasoning about higher-order dynamically allocated heap
- Iris' formalization in Coq allows us to mechanize our results in Coq

Introduction

In the rest of this talk I will present:

- STLang:
  - Typing rules for STmonad and `runST` (intuitive reason for encapsulation)
  - Example program
- The state-independence theorem and equations we have prove
- The logical relations that we use for proving the theorems above
  - Some basic concepts in Iris base logic as needed
  - Our constructions on top of Iris base logic

STLang: intuition for why types ensures encapsulation

- Type system ensures effects are **restricted**

$$\textsc{TNew}\ \frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{ref}(e) : \texttt{ST}\ \rho\ (\texttt{STRef}\ \rho\ \tau)}$$

STLang: intuition for why types ensures encapsulation

- Type system ensures effects are **restricted**

$$\textsc{Tnew}\ \frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{ref}(e) : \texttt{ST}\ \rho\ (\texttt{STRef}\ \rho\ \tau)} \qquad \textsc{Tderef}\ \frac{\Xi \mid \Gamma \vdash e : \texttt{STRef}\ \rho\ \tau}{\Xi \mid \Gamma \vdash\ !\, e : \texttt{ST}\ \rho\ \tau}$$

$$\textsc{Tgets}\ \frac{\Xi \mid \Gamma \vdash e : \texttt{STRef}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \tau}{\Xi \mid \Gamma \vdash e \leftarrow e' : \texttt{ST}\ \rho\ \mathbf{1}}$$

$$\textsc{Treturn}\ \frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{return}\ e : \texttt{ST}\ \rho\ \tau} \qquad \textsc{Tbind}\ \frac{\Xi \mid \Gamma \vdash e : \texttt{ST}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \tau \to (\texttt{ST}\ \rho\ \tau')}{\Xi \mid \Gamma \vdash \texttt{bind}\ e\ \texttt{in}\ e' : \texttt{ST}\ \rho\ \tau'}$$

STLang: intuition for why types ensures encapsulation

- Type system ensures effects are **restricted**

$$\textsc{TNew}\ \frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{ref}(e) : \texttt{ST}\ \rho\ (\texttt{STRef}\ \rho\ \tau)}$$

$$\textsc{TDeref}\ \frac{\Xi \mid \Gamma \vdash e : \texttt{STRef}\ \rho\ \tau}{\Xi \mid \Gamma \vdash\ !\, e : \texttt{ST}\ \rho\ \tau}$$

$$\textsc{TGets}\ \frac{\Xi \mid \Gamma \vdash e : \texttt{STRef}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \tau}{\Xi \mid \Gamma \vdash e \leftarrow e' : \texttt{ST}\ \rho\ \mathbf{1}}$$

$$\textsc{TReturn}\ \frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi \vdash \rho}{\Xi \mid \Gamma \vdash \texttt{return}\ e : \texttt{ST}\ \rho\ \tau}$$

$$\textsc{TBind}\ \frac{\Xi \mid \Gamma \vdash e : \texttt{ST}\ \rho\ \tau \qquad \Xi \mid \Gamma \vdash e' : \tau \to (\texttt{ST}\ \rho\ \tau')}{\Xi \mid \Gamma \vdash \texttt{bind}\ e\ \texttt{in}\ e' : \texttt{ST}\ \rho\ \tau'}$$

- `runST` runs a suspended computation that is **region independent**

$$\textsc{TRunST}\ \frac{\Xi, X \mid \Gamma \vdash e : \texttt{ST}\ X\ \tau \qquad \Xi \vdash \tau}{\Xi \mid \Gamma \vdash \texttt{runST}\ \{e\} : \tau}$$

Example

$$\texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(3+1)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}$$

Example

$$\cdot \mid \cdot \vdash \square : \mathbb{Z}$$

$$\texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3+1)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !L\}$$

Example

$$X \mid \cdot \vdash \square : \mathtt{ST}\ X\ \mathbb{Z}$$

$$\mathtt{runST}\ \{\mathtt{bind}\ \mathtt{ref}(3+1)\ \mathtt{in}\ \lambda L.\ \mathtt{bind}\ !L\ \mathtt{in}\ \lambda y.\ \mathtt{bind}\ L \leftarrow (y+1)\ \mathtt{in}\ \lambda z.\ !L\}$$

Example

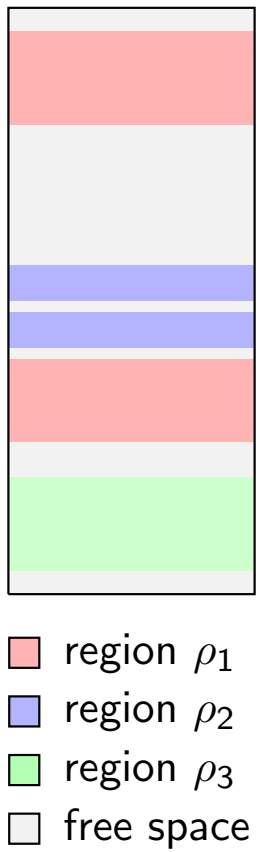
$X \mid \cdot \vdash \square : \mathtt{ST}\ X\ (\mathtt{STRef}\ X\ \mathbb{Z})$ $\quad$ $X \mid \cdot \vdash \square : (\mathtt{STRef}\ X\ \mathbb{Z}) \to \mathtt{ST}\ X\ \mathbb{Z}$

$$\mathtt{runST}\ \{\mathtt{bind}\ \mathtt{ref}(3+1)\ \mathtt{in}\ \lambda L.\ \mathtt{bind}\ !L\ \mathtt{in}\ \lambda y.\ \mathtt{bind}\ L \leftarrow (y+1)\ \mathtt{in}\ \lambda z.\ !L\}$$

Example

$$\langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(3+1)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\,\rangle$$

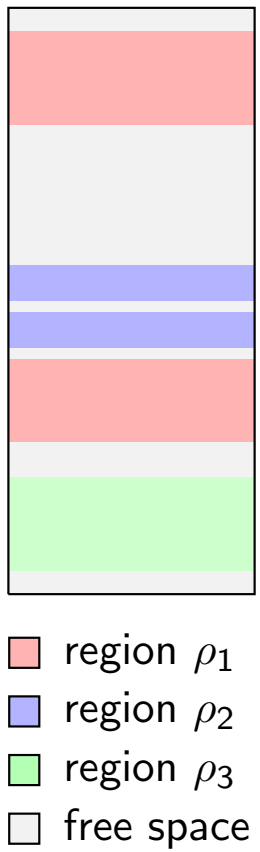
Fictional Heap



Example

$$\langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(3+1)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\rangle$$
$$\rightarrow \langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(4)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\rangle$$

Fictional Heap

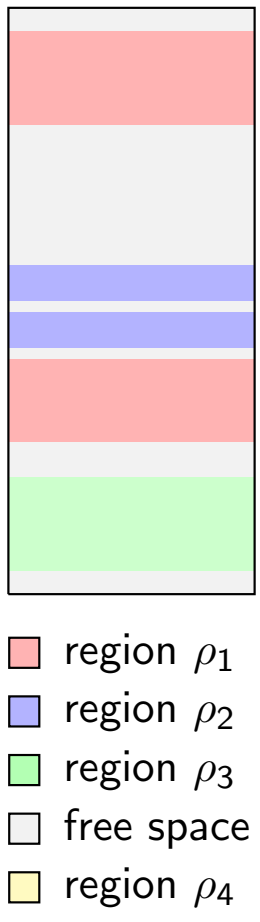


Example

$$\langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(3+1)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\rangle$$

$$\rightarrow \langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(4)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\rangle$$

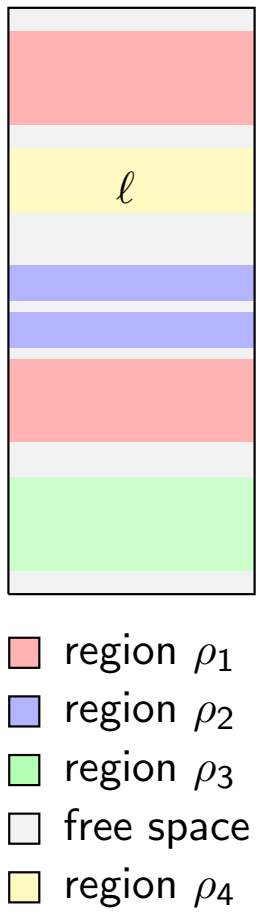
Fictional Heap



Example

$$
\begin{aligned}
& \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3+1)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\ \rangle \\
\rightarrow\ & \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\}\,, \\
& \texttt{runST}\ \{\texttt{bind}\ \texttt{return}\ \ell\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle
\end{aligned}
$$

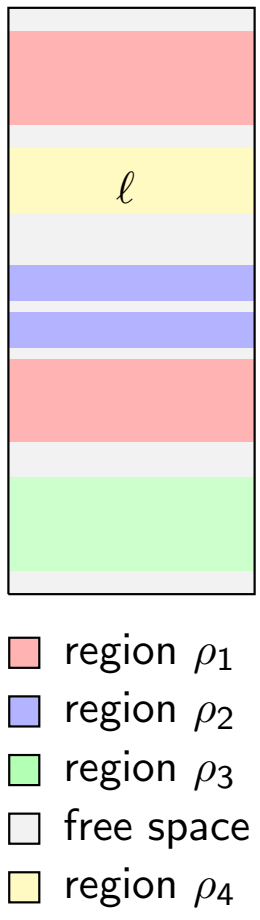
Fictional Heap



Example

$$
\begin{aligned}
& \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3+1)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\ \rangle \\
\rightarrow\ & \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\}\,, \\
& \texttt{runST}\ \{\texttt{bind}\ \texttt{return}\ \ell\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\}\,, \texttt{runST}\ \{(\lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L)\ \ell\}\rangle
\end{aligned}
$$

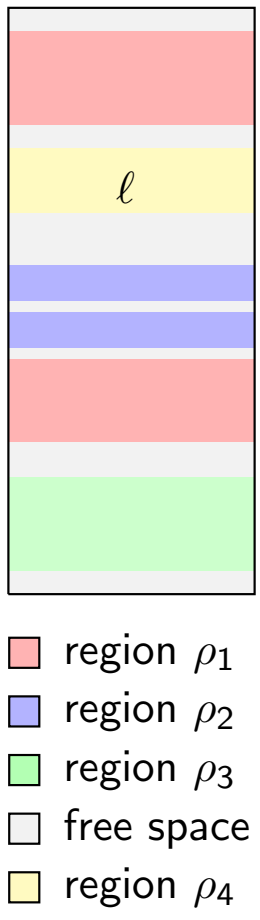
Fictional Heap



Example

$$
\begin{aligned}
& \langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(3+1)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\,\rangle \\
\rightarrow\ & \langle h, \texttt{runST}\ \{\texttt{bind}\,\texttt{ref}(4)\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\}\,, \\
& \texttt{runST}\ \{\texttt{bind}\,\texttt{return}\,\ell\,\texttt{in}\,\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\}\,, \texttt{runST}\ \{(\lambda L.\,\texttt{bind}\,!\,L\,\texttt{in}\,\lambda y.\,\texttt{bind}\,L \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,L)\ \ell\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\}\,, \texttt{runST}\ \{\texttt{bind}\,!\,\ell\,\texttt{in}\,\lambda y.\,\texttt{bind}\,\ell \leftarrow (y+1)\,\texttt{in}\,\lambda z.\,!\,\ell\}\rangle
\end{aligned}
$$

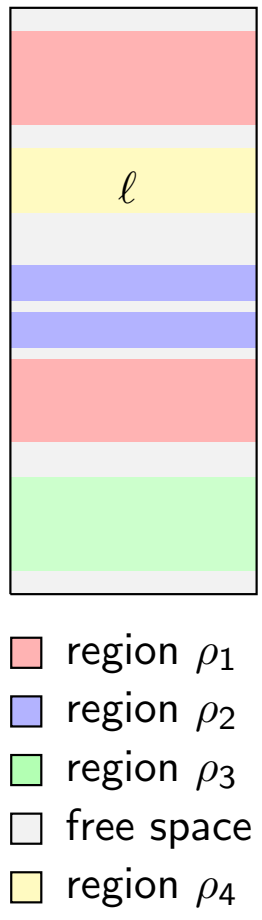
Fictional Heap



Example

$$
\begin{aligned}
& \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3+1)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\rightarrow\ & \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\},\ \\
& \texttt{runST}\ \{\texttt{bind}\ \texttt{return}\ \ell\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{(\lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L)\ \ell\}\rangle \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{\texttt{bind}\ !\,\ell\ \texttt{in}\ \lambda y.\ \texttt{bind}\ \ell \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,\ell\}\rangle \\
& \vdots \\
\rightarrow\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{\texttt{bind}\ \ell \leftarrow 5\ \texttt{in}\ \lambda z.\ !\,\ell\}\rangle
\end{aligned}
$$

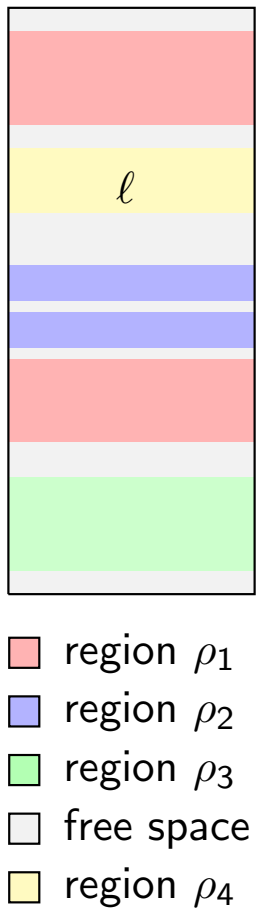
Fictional Heap



Example

$$
\begin{aligned}
& \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(3+1)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\to\ & \langle h, \texttt{runST}\ \{\texttt{bind}\ \texttt{ref}(4)\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\to\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{\texttt{bind}\ \texttt{return}\ \ell\ \texttt{in}\ \lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L\}\rangle \\
\to\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{(\lambda L.\ \texttt{bind}\ !\,L\ \texttt{in}\ \lambda y.\ \texttt{bind}\ L \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,L)\ \ell\}\rangle \\
\to\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{\texttt{bind}\ !\,\ell\ \texttt{in}\ \lambda y.\ \texttt{bind}\ \ell \leftarrow (y+1)\ \texttt{in}\ \lambda z.\ !\,\ell\}\rangle \\
& \vdots \\
\to\ & \langle h \uplus \{(\ell, 4)\},\ \texttt{runST}\ \{\texttt{bind}\ \ell \leftarrow 5\ \texttt{in}\ \lambda z.\ !\,\ell\}\rangle \\
& \vdots \\
\to\ & \langle h \uplus \{(\ell, 5)\},\ \texttt{runST}\ \{!\,\ell\}\rangle \\
\to\ & \langle h \uplus \{(\ell, 5)\},\ \texttt{runST}\ \{\texttt{return}\ 5\}\rangle \\
\to\ & \langle h \uplus \{(\ell, 5)\},\ 5\rangle
\end{aligned}
$$

Fictional Heap



State-independence theorem

Suppose

$$\cdot \mid x : \texttt{STRef}\ \rho\ \tau' \vdash e : \tau$$

and that there are some h_1, ℓ, h_2 and v such that

$$\langle h_1, e[\ell/x] \rangle \to^* \langle h_2, v \rangle$$

Then

$$\forall h_1', \ell'.\ \exists h_2', v'.\ \left\langle h_1', e[\ell'/x] \right\rangle \to^* \left\langle h_2', v' \right\rangle \wedge h_1' \subseteq h_2'$$

Intuitively, if e can be executed under **some** heap h_1, then it can be executed under **any** h_1' and it can only allocate on top of h_1' and not change h_1'!

Contextual refinement/equivalence

We say e contextually refines e', $\Xi \mid \Gamma \vDash e \preceq_{\text{ctx}} e' : \tau$, if $\Xi \mid \Gamma \vdash e : \tau$ and $\Xi \mid \Gamma \vdash e' : \tau$ and that

$$\forall h, h', C.\ \ C : (\Xi \mid \Gamma; \tau) \rightsquigarrow (\cdot \mid \cdot; \mathbf{1}) \ \wedge\ (h, C[e])\downarrow \ \implies\ (h', C[e'])\downarrow$$

where $(h, e)\downarrow \ \triangleq\ \exists h', v.\ (h, e) \to^* (h', v)$

We say e and e' are contextually equivalent, $\Xi \mid \Gamma \vDash e \approx_{\text{ctx}} e' : \tau$, if

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

Intuitively, context C cannot distinguish e and e'!

Contextual equations we prove

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

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

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

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

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

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

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

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

$$(\Lambda\, e)\ \_ \approx_{\mathsf{ctx}} e : \tau[\tau'/X] \qquad (\beta \text{ reduction for } \Lambda)$$

$$\texttt{bind}\, e\, \texttt{in}\, (\lambda x.\, \texttt{return}\, x) \approx_{\mathsf{ctx}} e : \texttt{ST}\ \rho\ \tau \qquad \text{(Left Identity)}$$

$$e_2\ e_1 \preceq_{\mathsf{ctx}} \texttt{bind}\, (\texttt{return}\, e_1)\, \texttt{in}\, e_2 : \texttt{ST}\ \rho\ \tau \qquad \text{(Right Identity)}$$

$$\texttt{bind}\, (\texttt{bind}\, e_1\, \texttt{in}\, e_2)\, \texttt{in}\, e_3 \preceq_{\mathsf{ctx}} \texttt{bind}\, e_1\, \texttt{in}\, (\lambda x.\, \texttt{bind}\, (e_2\ x)\, \texttt{in}\, e_3) : \texttt{ST}\ \rho\ \tau' \qquad \text{(Associativity)}$$

How to prove contextual refinements?

Well-known way to prove contextual equations:
use a binary logical relations model

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.:

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

3. Corollary (*fundamental theorem of LR*):

$$\Xi \mid \Gamma \vdash e : \tau \Rightarrow \Xi \mid \Gamma \vDash e \preceq_{\mathsf{log}} e : \tau$$

4. Corollary (*soundness*):

$$\Xi \mid \Gamma \vDash e \preceq_{\mathsf{log}} e' : \tau \Rightarrow \Xi \mid \Gamma \vdash e \preceq_{\mathsf{ctx}} e' : \tau$$

Using Iris to construct logical relation

Define in two steps

- Value relation for types $[\![\Xi \vdash \tau]\!]_\Delta : \mathit{Val} \to \mathit{Val} \to \mathit{iProp}$
- Expression relation $\mathcal{E}\,[\![\Xi \vdash \tau]\!]_\Delta : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$
  - based on $[\![\Xi \vdash \tau]\!]_\Delta$
  - intuitively: $\mathcal{E}\,[\![\Xi \vdash \tau]\!]_\Delta\,(e, e')$ if whenever $(h, e) \to^* v$ then $(h', e') \to^* v'$ such that $[\![\Xi \vdash \tau]\!]_\Delta(v, v')$
- Δ: maps type variables in Ξ to a pair of a value relations and a region name

$$\Delta : \mathit{Tvar} \to ((\mathit{Val} \times \mathit{Val}) \to \mathit{iProp}) \times \mathbb{N}$$

Value interpretations (an excerpt)

Almost standard value interpretation for types (except for ST and `STRef`)

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

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

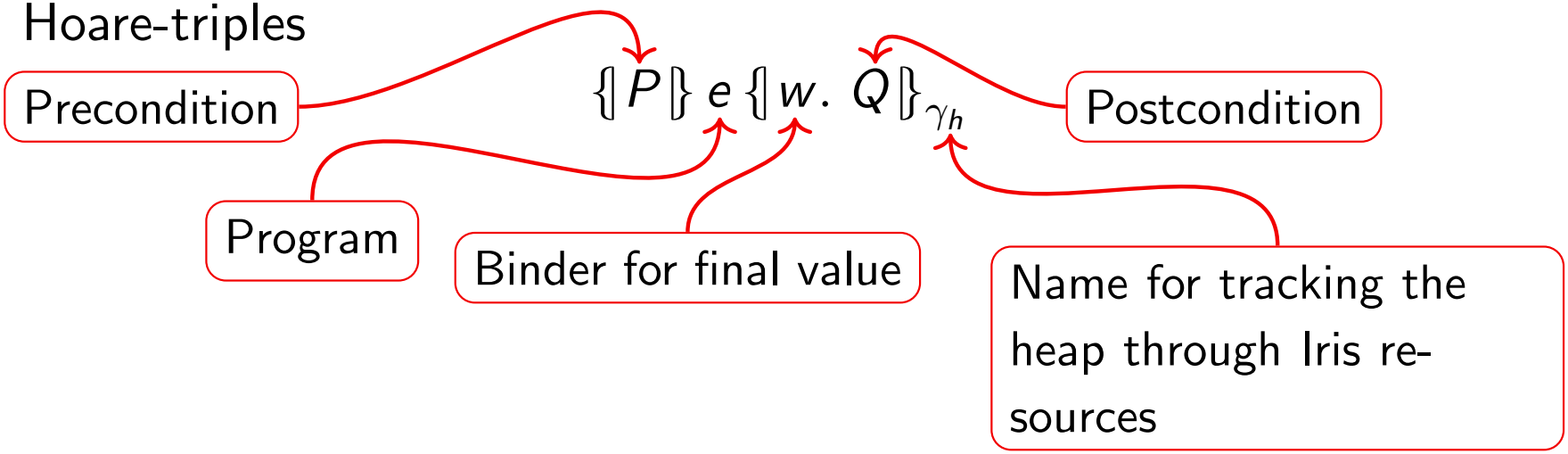
$$[\![\Xi \vdash \tau \times \tau']\!]_\Delta(v, v') \triangleq \exists w_1, w_2, w_1', w_2'.\ v = (w_1, w_2) \wedge v' = (w_1', w_2') \wedge [\![\Xi \vdash \tau]\!]_\Delta(w_1, w_1') \wedge [\![\Xi \vdash \tau']\!]_\Delta(w_2, w_2')$$

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

$$[\![\Xi \vdash \forall X.\,\tau]\!]_\Delta(v, v') \triangleq \Box\Big(\forall f.\ \mathsf{persistent}(f) \Rightarrow \mathcal{E}\,[\![\Xi, X \vdash \tau]\!]_{\Delta, X \mapsto (f, 0)}\,(v\ \_, v'\ \_)\Big)$$

IC-triples (used for defining the expression relation)

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

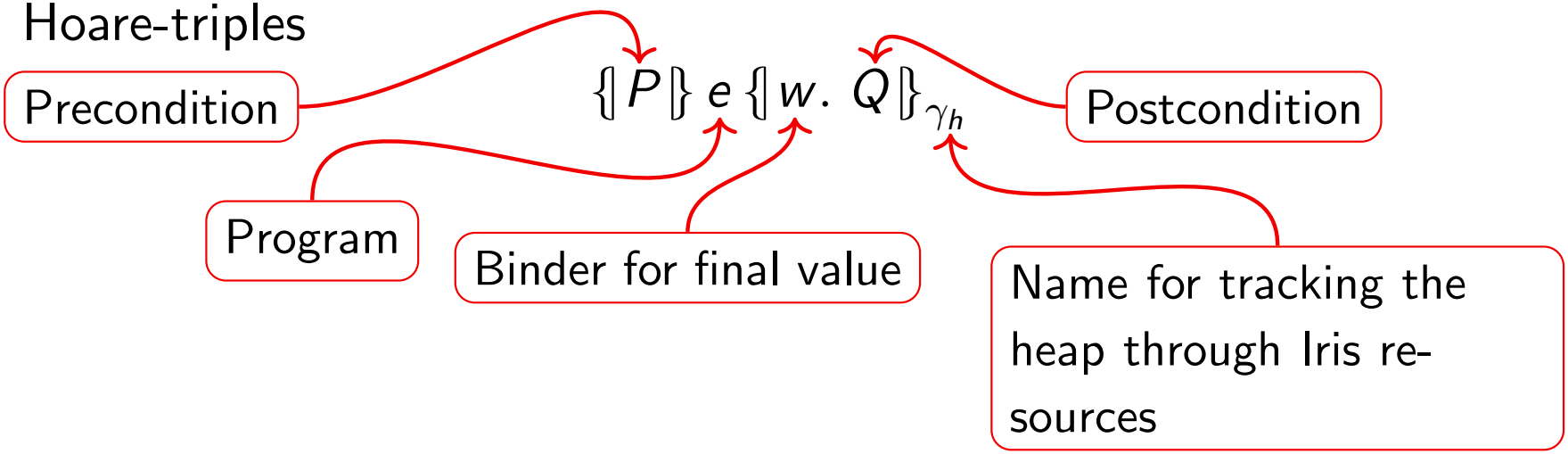


Intuitively: if P holds **and** e **converges** to a value w then Q holds.

IC-triples do not guarantee safety (unlike Hoare-triples) but allow for a more **fine-grained** relation between computations

IC-triples (used for defining the expression relation)

We define "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 do not guarantee safety (unlike Hoare-triples) but allow for a more **fine-grained** relation between computations

We use Iris resources to keep tracking the heap: $heap_\gamma(h)$

Intuitively: this predicate states that the heap being tracked under the **resource name** γ is h.

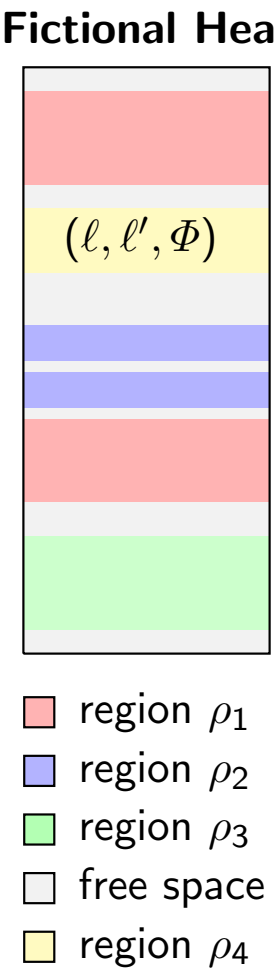
The expression relation

$$\mathcal{E}\,[\![\Xi \vdash \tau]\!]_\Delta\,(e, e') \triangleq \forall \gamma_h, \gamma_h', h_1'.$$

$$\left\{\left| \mathit{heap}_{\gamma_h'}(h_1') * \mathsf{regions} \right|\right\}\ e\ \left\{\left| w.\ (h_1', e') \Downarrow^{\gamma_h'}_{[\![\Xi \vdash \tau]\!]_\Delta(w,\cdot)} \right|\right\}_{\gamma_h}$$

where

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



The regions predicate keeps track of regions (fictional heap) mapping pairs of memory locations to predicates.

Intuitively, values stored in ℓ and ℓ' should be related by Φ.

Value relation for ST

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

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

$$\left\{ \left| \mathit{heap}_{\gamma_h'}(h_1') * \mathsf{regions} * \mathsf{region}(\mathsf{toRgn}(\Delta, \rho), \gamma_h, \gamma_h') \right| \right\}$$

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

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

Intuitively, $\mathsf{region}(\mathsf{toRgn}(\Delta, \rho), \gamma_h, \gamma_h')$ states that we own Iris resources corresponding to the region $\mathsf{toRgn}(\Delta, \rho)$ (in the fictional heap) in heaps tracked by γ_h and γ_h'.

Value relation for STRef and region ownership

Region name $\mathsf{toRgn}(\Delta, \rho)$ is **uniquely** tied to **semantic region** r

$$[\![\Xi \vdash \texttt{STRef}\ \rho\ \tau]\!]_\Delta(\ell, \ell') \triangleq \exists r.\ \mathsf{RegOf}(\mathsf{toRgn}(\Delta, \rho), r) * \mathsf{rel}(r, \ell, \ell', [\![\Xi \vdash \tau]\!]_\Delta)$$

Defined by Iris resources: $(\ell, \ell', [\![\Xi \vdash \tau]\!]_\Delta) \in r$

$$\mathsf{region}(n, \gamma_h, \gamma_h') \triangleq \exists r.\ \mathsf{RegOf}(n, r) * \mathop{*}_{(\ell,\ell',\Phi)\in r} \begin{pmatrix} \exists v, v'.\ \ell \mapsto_{\gamma_h} v * \ell' \mapsto_{\gamma_h'} v' * \\ \triangleright \Phi(\ell, \ell')(v, v') \end{pmatrix}$$

These definitions are slightly simplified see the paper for more details.

An excerpt of the proof of compatibility lemma for runST

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

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

An excerpt of the proof of compatibility lemma for runST

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

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

We have $([\![\Xi, X \vdash \mathtt{ST}\ X\ \tau]\!]_\Delta(v, v'))$:

$$\forall f, n.\ \cdots \Rightarrow \left\{\left| \begin{array}{l} \mathit{heap}_{\gamma_h'}(h_1') * \text{regions} * \\ \text{region}(\text{toRgn}((\Delta, \rho \mapsto (f, n)), \rho), \gamma_h, \gamma_h') \end{array} \right|\right\}$$
$$\texttt{runST}\ \{v\}$$
$$\left\{\left| \begin{array}{l} w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\Xi, X \vdash \tau]\!]_{\Delta, \rho \mapsto (f,n)}(w,\cdot)} * \\ \text{region}(\text{toRgn}((\Delta, \rho \mapsto (f, n)), \rho), \gamma_h, \gamma_h') \end{array} \right|\right\}_{\gamma_h}$$

An excerpt of the proof of compatibility lemma for runST

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

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

We have $([\![\Xi, X \vdash \texttt{ST}\ X\ \tau]\!]_\Delta(v, v'))$:

$$\forall f, n.\ \cdots \Rightarrow \left\{\left| \begin{array}{l} \mathit{heap}_{\gamma_h'}(h_1') * \text{regions} * \\ \text{region}(\text{toRgn}((\Delta, \rho \mapsto (f, n)), \rho), \gamma_h, \gamma_h') \end{array} \right|\right\}$$
$$\texttt{runST}\ \{v\}$$
$$\left\{\left| \begin{array}{l} w.\ (h_1', \texttt{runST}\ \{v'\}) \Downarrow^{\gamma_h'}_{[\![\Xi, X \vdash \tau]\!]_{\Delta, \rho \mapsto (f,n)}(w,\cdot)} * \\ \text{region}(\text{toRgn}((\Delta, \rho \mapsto (f, n)), \rho), \gamma_h, \gamma_h') \end{array} \right|\right\}_{\gamma_h}$$

We simply **allocate** a new **empty** semantic region r and assign a new region name m to it to obtain $\text{RegOf}(m, r)$ and hence $\text{region}(m, \gamma_h, \gamma_h')$.

Conclusion

- We construct a logical relation in Iris for STLang
- Mechanized in Coq
- Prove proper encapsulation of state by the ST monad:
  - We proved state-independence theorem (direct consequence of fundamental theorem and soundness of LR)
  - We used our LR to show contextual equations that only hold for pure programs

More details

In the paper:

- The exact definition of LR, e.g.:
  - regions, region and other resources used in LR
  - IC triples, future modality[3]
  - NN-logical relations [4]
- Sketches of proofs of all equations
- ...

In the Coq development:[5]

- The construction of the logical relations
- Mechanized proof of all equations
- Mechanized proof of the state-independence theorem

---

[3] Used in the definition of IC triples. Omitted in this talk.

[4] Slightly stronger than the presented LR, needed for some equations, e.g., hoisting. Omitted in this talk.

[5] Available at: `http://www.iris-project.org`

Bibliography

John Launchbury and Simon L. Peyton Jones. Lazy functional state threads. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, PLDI '94, pages 24–35, New York, NY, USA, 1994. ACM. ISBN 0-89791-662-X.