Transfinite Step-Indexing

Filip Sieczkowski
with Kasper Svendsen and Lars Birkedal

Step-Indexed Logical Relations at a Glance

- Handles various constructs: general recursion, recursive type, <u>general references</u> (mutable state)
- Natural number determines how many steps of evaluation expression is allowed to take
- For general references, impredicativity means models are difficult to define

LR for State and Abstraction

- A functional PL with references and existential types
  $$\tau, \sigma ::= 1 \mid \mathbb{N} \mid \tau \times \sigma \mid \tau \to \sigma \mid \tau \text{ ref } \mid \exists \alpha.\, \tau \mid \alpha$$
- Allow relating parts of heap of different sizes through *impredicative invariants*
  $\mathbf{W} = \mathbb{N} \to_f \mathbf{Inv}$ $\quad$ $\mathbf{Inv} \backsimeq \mathbf{W} \to_m \mathbf{UPred}(\mathrm{Heap} \times \mathrm{Heap})$
- Invariants well-defined due to step-indexing, using one forces to take an evaluation step
- Semantic types can refer to invariants
  $\mathbf{Type} = \mathbf{W} \to_m \mathbf{UPred}(\mathrm{Val} \times \mathrm{Val})$

LR for State and Abstraction

$V[\![\sigma \to \tau]\!]_\rho(w) = \{(n, v_1, v_2) \mid \forall m \le n.\ \forall w' \ge w.\ \forall u_1, u_2.$

$\quad (m, u_1, u_2) \in V[\![\sigma]\!]_\rho(w') \Rightarrow (m, v_1\ u_1, v_2\ u_2) \in \mathcal{E}(V[\![\tau]\!]_\rho)(w')\}$

$V[\![\exists\alpha.\ \tau]\!]_\rho(w) = \{(n, \mathsf{pack}\ v_1, \mathsf{pack}\ v_2) \mid \exists \nu \in \mathbf{Type}.\ (n, v_1, v_2) \in V[\![\tau]\!]_{\rho[\alpha \mapsto \nu]}(w)\}$

$V[\![\mathsf{ref}\ \tau]\!]_\rho(w) = \{(n, l_1, l_2) \mid \exists\iota.\ w(\iota) \doteq_n \mathsf{inv}(V[\![\tau]\!]_\rho, l_1, l_2)\}$

$\mathcal{E}(\nu)(w) = \{(n, e_1, e_2) \mid \forall i < n, e_1'.\ e_1 \to^i e_1' \Rightarrow$

$\quad \exists w' \ge w, e_2'.\ e_2 \to e_2' \wedge (n-i, e_1', e_2') \in \nu(w')\}$

Example: Counting Up and Down

```
τ = ∃α. (1 → α × α → nat)
```

```
countUp = pack (ref nat,
  (λ_. ref 0,
   λc. let v = !c
       in c := v+1; v))
```

```
countDown = pack (ref nat,
  (λ_. ref 0,
   λc. let v = !c
       in c := v-1; -v))
```

How do we show equivalence?

Example: Counting Up and Down

```
countUp = pack (ref nat,
  (λ_. ref 0,
   λc. let v = !c
       in c := v+1; v))
```

```
countDown = pack (ref nat,
  (λ_. ref 0,
   λc. let v = !c
       in c := v-1; -v))
```

Invariant: $S(l_u, l_d)\,(W) = \{(n, h_u, h_d) \mid h_u(l_u) = -\,h_d(l_d)\}$

Relation: $v(W) = \{(n, v_u, v_d) \mid \exists\iota.\ W(\iota) \mathrel{\backsimeq_n} S(v_u, v_d)\}$

Trouble Ahead

```
f(C) =
  let (α, (new, inc)) = unpack(C)
  in pack (ref α,
    (λ_. ref (new ()),
     λc. inc (!c)))
```

- We can show `countUp` $=_{log}$ `f(countUp)`, `countUp` $=_{log}$ `f(countDown)`, etc.

- But can we show that `x` $=_{log}$ `f(x)` *without* knowing the implementation of `x`?

Trouble Ahead

```
f(C) =
  let (α, (new, inc)) = unpack(C)
  in pack (ref α,
    (λ_. ref (new ()),
     λc. inc (!c)))
```

$S(v_s, l_i)(w) =$
$\{(n, h_s, h_i) \mid (n, v_s, h_i(l_i)) \in v(w)\}$

- Need to introduce a *fresh* invariant to express the additional indirection
- Unfolding this invariant requires taking a step
- In the second function, we need arguments ("c" above) related *without* taking steps

Why is this a problem?

- Evidence that relationship between evaluation and recursive construction of invariants may be too close
- In program logics, this corresponds to *layering of abstractions*
- Clients impose *additional* constraints on specification provided by libraries through new invariants: need to open multiple layers of invariants in single step
- Investigate LR as a more isolated case

Our approach

- Decouple operational steps from solution of recursive domain equation
- Index the construction over *ordered pairs* of numbers
- Associate operational step with the first component
- Allow for arbitrary finite number of unfoldings between each steps

The Transfinite Definition

$V[\![\sigma \to \tau]\!]_\rho(w) = \{(n, m, v_1, v_2) \mid \forall n' < n.\ \forall w' \geq w.\ \forall u_1, u_2.$
$\quad (\forall m'.\ (n', m', u_1, u_2) \in V[\![\sigma]\!]_\rho(w')) \Rightarrow (n'+1, v_1\ u_1, v_2\ u_2) \in \mathcal{E}(V[\![\tau]\!]_\rho)(w')\}$

$V[\![\exists \alpha.\ \tau]\!]_\rho(w) = \{(n, m, \text{pack } v_1, \text{pack } v_2) \mid$
$\quad \exists \nu \in \text{Type}.\ (n, m, v_1, v_2) \in V[\![\tau]\!]_{\rho[\alpha \mapsto \nu]}(w)\}$

$V[\![\text{ref } \tau]\!]_\rho(w) = \{(n, m, l_1, l_2) \mid \exists \iota.\ w(\iota) \simeq_{n,m} \text{inv}(V[\![\tau]\!]_\rho, l_1, l_2)\}$

$\mathcal{E}(\nu)(w) = \{(n, e_1, e_2) \mid \forall i < n,\ e_1'.\ e_1 \to^i e_1' \Rightarrow$

$\quad \exists w' \geq w,\ e_2'.\ e_2 \to e_2' \wedge \forall m.\ (n\text{-}i, m, e_1', e_2') \in \nu(w')\}$

The Transfinite Definition

For our example, we are free to pick larger m' to allow for additional unfolding

$$V[\![\sigma \to \tau]\!]_\rho(w) = \{(n, m, v_1, v_2) \mid \forall n' < n.\ \forall w' \geq w.\ \forall u_1, u_2.$$
$$(\forall m'.\ (n', m', u_1, u_2) \in V[\![\sigma]\!]_\rho(w')) \Rightarrow (n'+1, v_1\ u_1, v_2\ u_2) \in \mathscr{E}(V[\![\tau]\!]_\rho)(w')\}$$

$$V[\![\exists \alpha.\ \tau]\!]_\rho(w) = \{(n, m, \mathsf{pack}\ v_1, \mathsf{pack}\ v_2) \mid$$
$$\exists \nu \in \mathbf{Type}.\ (n, m, v_1, v_2) \in V[\![\tau]\!]_{\rho[\alpha \mapsto \nu]}(w)\}$$

$$V[\![\mathsf{ref}\ \tau]\!]_\rho(w) = \{(n, m, l_1, l_2) \mid \exists \iota.\ w(\iota) \backsimeq_{n,m} \mathrm{inv}(V[\![\tau]\!]_\rho, l_1, l_2)\}$$

$$\mathscr{E}(\nu)(w) = \{(n, e_1, e_2) \mid \forall i < n, e_1'.\ e_1 \to^i e_1' \Rightarrow$$
$$\exists w' \geq w, e_2'.\ e_2 \to e_2' \land \forall m.\ (n-i, m, e_1', e_2') \in \nu(w')\}$$

Key technical difficulty is hidden behind this approximate equality

Conclusions

- We solved the recursive domain equation over pairs of natural numbers (ω^2)
- We loosened the tight coupling between operational steps and unfolding of impredicative invariants
- Logical relation provides proof-of-concept that this approach can handle *layered abstractions*