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, general references (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
  $$\tau, \sigma ::= 1 \mid \mathbb{N} \mid \tau \times \sigma \mid \tau \rightarrow \sigma \mid \tau \text{ ref} \mid \exists \alpha. \tau \mid \alpha$$

• Allow relating parts of heap of different sizes through *impredicative invariants*
  $$\mathcal{W} = \mathbb{N} \rightarrowf \text{ Inv} \quad \text{Inv} = \mathcal{W} \rightarrowm \text{ UPred}(\text{Heap} \times \text{Heap})$$

• Invariants well-defined due to step-indexing, using one forces to take an evaluation step

• Semantic types can refer to invariants
  $$\text{Type} = \mathcal{W} \rightarrowm \text{ UPred}(\text{Val} \times \text{Val})$$
LR for State and Abstraction

\[ V[\sigma \rightarrow \tau]_\rho(w) = \{(n, v_1, v_2) \mid \forall m \leq n. \ \forall w' \geq w. \ \forall u_1, u_2. \]
\[ (m, u_1, u_2) \in V[\sigma]_\rho(w') \Rightarrow (m, v_1 u_1, v_2 u_2) \in \mathcal{E}(V[\tau]_\rho(w')) \]

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

\[ V[\text{ref } \tau]_\rho(w) = \{(n, l_1, l_2) \mid \exists t. \ w(t) =_n \text{inv}(V[\tau]_\rho, l_1, l_2) \}
\]

\[ \mathcal{E}(\nu)(w) = \{(n, e_1, e_2) \mid \forall i < n. \ e_1' . \ e_1 \rightarrow^i e_1' \Rightarrow \]
\[ \exists w' \geq w. \ e_2'. \ e_2 \rightarrow e_2' \land (n-i, e_1', e_2') \in v(w') \} \]
Example: Counting Up and Down

\[ \tau = \exists \alpha. \ (1 \to \alpha \times \alpha \to \text{nat}) \]

\[
\begin{align*}
\text{countUp} &= \text{pack} \ (\text{ref nat,} \\
& \quad (\lambda_. \ \text{ref 0,} \\
& \quad \lambda c. \ \text{let v = !c} \\
& \quad \text{in c := v+1; v})) \\
\text{countDown} &= \text{pack} \ (\text{ref nat,} \\
& \quad (\lambda_. \ \text{ref 0,} \\
& \quad \lambda c. \ \text{let v = !c} \\
& \quad \text{in c := v-1; -v}))
\end{align*}
\]

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

Relation: $v(W) = \{(n, v_u, v_d) | \exists l. W(l) \equiv_n S(v_u, v_d)\}$
Trouble Ahead

\[ f(C) = \]
\[
\text{let } (\alpha, (\text{new, inc})) = \text{unpack}(C) \]
\[
\text{in pack } (\text{ref } \alpha, \\
(\lambda_. \text{ ref } (\text{new }()), \\
\lambda c. \text{ inc } (!c)))
\]

• We can show \( \text{countUp} =_{\text{log}} f(\text{countUp}) \),
  \( \text{countUp} =_{\text{log}} f(\text{countDown}) \), etc.

• But can we show that \( x =_{\text{log}} f(x) \) \textit{without}
  knowing the implementation of \( x \)?
Trouble Ahead

\[
f(C) = \begin{align*}
\text{let } (\alpha, (\text{new, inc})) &= \text{unpack}(C) \\
\text{in } \text{pack}(\text{ref } \alpha, \\
& \quad (\lambda_. \text{ref } (\text{new } ()), \\
& \quad \lambda c. \text{inc } (!c)))
\end{align*}
\]

• 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

\[ \forall [\sigma \to \tau]_\rho(w) = \{(n, m, v_1, v_2) \mid \forall n' < n. \forall w' \geq w. \forall u_1, u_2. \]
\[ (\forall m'. (n', m', u_1, u_2) \in V[\sigma]_\rho(w')) \Rightarrow (n'+1, v_1 u_1, v_2 u_2) \in \mathcal{E}(V[\tau]_\rho(w')) \]  
\[ V[\exists \alpha. \tau]_\rho(w) = \{(n, m, \text{pack } v_1, \text{pack } v_2) \mid \exists \nu \in \text{Type}. (n, m, v_1, v_2) \in V[\tau]_\rho[\alpha \mapsto \nu](w) \} \]
\[ V[\text{ref } \tau]_\rho(w) = \{(n, m, l_1, l_2) \mid \exists l. w(l) =_{n,m} \text{inv}(V[\tau]_\rho, l_1, l_2) \} \]

\[ \mathcal{E}(v)(w) = \{(n, e_1, e_2) \mid \forall i < n. e_1'. e_1 \to^i e_1' \Rightarrow \]
\[ \exists w' \geq w, e_2'. e_2 \to e_2' \land \forall m. (n-i, m, e_1', e_2') \in v(w') \} \]
The Transfinite Definition

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

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

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

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

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

Key technical difficulty is hidden behind this approximate equality.
Conclusions

• We solved the recursive domain equation over pairs of natural numbers ($\omega^2$)

• 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