A Mechanized Proof in Coq of the Type Soundness of Core L^3

Milestone 3

Yawar Raza

Section 1

Background

$L^3$and Type Soundness

- Supports *strong updates*: updating a pointer's contents to a value of a different type.
- Trade-off: the language is *linear*, meaning all variables are used exactly once.
- The rules in L^3's *type system* enforce such restrictions that allow strong updates to be safe.
- We can formally prove that these rules don't permit erroneous programs. This property is called *type soundness*.

Mechanization

- Proofs about programming languages are traditionally only worked out by hand.
- Nowadays, PL researchers often also use software tools to construct proofs. I am using a tool called Coq.
- The software checks that these *mechanized* proofs are correct.
- We must translate our proof appropriately so the software will understand it.
- There are different ways to translate the constructs used in our proof, some of which are easier to use than others.

Section 2

Representation Decisions

Locally Nameless Representation

- In paper proofs, variables can be implicitly renamed to prevent conflicts.
- Coq can't do this, so we need to carefully consider how variables are represented. I used the locally nameless representation.
- Bound variables use de Bruijn indices: A variable is represented by a number indicating the *relative* place that variable was introduced.
  - Checking if two types are equal is easy.
- Free variables use explicit variable names.
  - Environments for mapping these variables are simple.

Environments

- Environments map variables to some other value. I used several different types of environments.
- I initially represented them using functions: $f(x) = v$ means x maps to v.
    - Problem: No concrete access to the variables it binds.
    - Problem: Need to separately specify finiteness.
- Changed to using a list of pairs $[(x_1, v_1), (x_2, v_2), ...]$.
    - Used an external library called TLC.
    - Potential problem: Permuted environment isn't recognized as equivalent. Ended up not being an issue.

Semantic Interpretations

- $\mathcal{V}[\![\tau]\!]$: Interpret type τ as a set of configurations (σ, e).
- Initially implemented as relation $\mathcal{V}(\tau, \sigma, e)$.
- Then implemented as a function $\mathcal{V}(\tau)$ that returned a relation $R(\sigma, e)$.
- To prove termination, added an extra function parameter: $\mathcal{V}(\tau, \tau')$.
- Then, I needed to change the return type to: $R(\delta, \sigma, e)$.
  - δ is an environment for substituting location variables.

Section 3

Progress

Progress

- Previous Milestones
    - Syntax, operational semantics, static semantics
- This Milestone
    - Lots of refactoring; migrated to the TLC library
    - Implemented semantic interpretations
    - Started type soundness proof cases
- Remaining Work
    - Remaining soundness cases
    - Requires proving basic properties of previous definitions