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

Project Description

Yawar Raza

Section 1

Motivation for Type Soundness

Type Checking

- Consider the problematic expression `"abababa" / 4.5`.
- Python detects the problem when the expression is evaluated at run-time.
  - This might not happen in all code paths.
- Java detects the problem at compile-time using its type checker, without needing to run the program at all.
- The set of rules implemented by a type checker is called a **type system**.

When Type Systems Are Wrong

```
1 String[] strings = new String[2];
2 Object[] objects = strings;
3 objects[0] = "hello";
4 objects[1] = new Scanner(System.in);
5 for (String s : strings) {
6     System.out.println(s.toUpperCase());
7 }
```

Line 4 throws an `ArrayStoreException`, preemptively preventing errors like line 6 calling `toUpperCase` on a `Scanner` object.

What is Type Soundness?

- Programming languages can be defined using a set of formal rules.
- The evaluation rules tell us whether an expression can be evaluated.
- The typing rules tell us whether an expression is well-typed.
- Thus, we can determine if the following property holds:

$$\forall\, expr.\ expr \text{ is well-typed} \Rightarrow expr \text{ can be evaluated}$$

  This property is called **type soundness**.
- Type soundness can be proved mathematically based on the language's formal rules.

Section 2

Motivation for Mechanization

Overview of Mechanization

- Mechanization is when a user inputs a proof to a piece of software that tells them if their proof is correct.
- This software includes Coq (which I'll be using), Agda, and Twelf.
- Proofs are often input as *computer programs* and are checked using an advanced *type checker*.
  - Any relation to the previous section is just coincidence!
- Tools often have special features that aid in writing proof programs, like interactivity and automation.

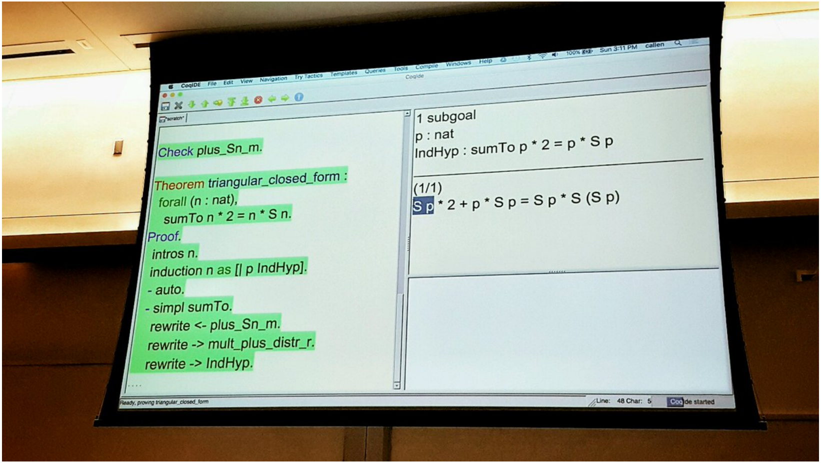

Figure: A screenshot of CoqIDE being used to write a proof.

Benefits of Mechanization

- The proof is machine-checked for correctness.
  - Complex type systems often require advanced proof techniques where mistakes can be hard to notice.
- Boilerplate aspects of proofs can be automated.
  - Some cases are simple enough that a built-in search can find the proof.
  - Cases with similar proofs can use the same generalized proof code.
- Extending a proof after adding new language features is easier and safer.
  - No changes need to be made to cases that still machine-check.
  - The cases that don't machine-check highlight where further research is needed.

Section 3

Project Goal and Milestones

Overview of L^3

- L^3 is introduced by the paper *L^3: A Linear Language with Locations*, by Ahmed, Fluet, and Morrisett.
- The goal of L^3 is to support *strong updates*. A strong update assigns a value of a different type to a reference cell.
- To make this sound, L^3 uses a linear type system; linear values must be used exactly once.
- L^3 uses linear "capability" values to ensure that reads are made using the most up-to-date type of the cell's contents.
- The paper presents both Core L^3 and Extended L^3, the latter having additional functionality.

Project Goal and Milestones

- The L^3 paper provides handwritten type soundness proofs for both Core L^3 and Extended L^3, but did not mechanize them.
- This project will mechanize the paper's type soundness proof for Core L^3 in the Coq proof assistant.
- Milestones:
  - Milestone 1: Mechanize the syntax and the operational semantics
  - Milestone 2: Mechanize the typing rules and the semantic interpretations
  - Milestone 3: Complete most of the cases of the type soundness proof