How it works:

▶ Looked for standard solution that’s suitable here.
▶ The software tool I used is the Coq proof assistant.

Researchers conceive and study ideas for programming language features.
▶ Formally and precisely define a small language.
▶ Traditionally by-hand; now often mechanized to prevent mistakes.

Why?

Core L³: a programming language introduced in [1].
▶ Included a type soundness proof, but it was not mechanized.

Project goal: Mechanize the paper’s type soundness proof of Core L³.
▶ Interesting to mechanize: linear type system, semantic soundness proof.

Each construct from the paper has multiple ways it can be mechanized.
▶ I figured out the best one to use for each; lots of experimentation.

Conclusion: I didn’t finish implementing the cases of the proof.
▶ Should be straightforward now that I mechanized all the paper’s constructs.
▶ I also proved some “helper lemmas” used by the main proof.

Syntax and Variables

▶ Syntax defines the structure of a program.
▶ Core L³ has locations, types, and expressions.

Variables: standard problem for mechanization.
▶ Why? The following expressions behave equivalently, but are not equal in Coq:
\[
\lambda x. (\lambda y. f x y) x \quad \lambda a. (\lambda b. f a b) a
\]

▶ Looked for standard solution that’s suitable here.
▶ Settled on the locally nameless representation [2].

How it works: Replace bound variables with relative offsets called de Bruijn indices.
▶ The above expressions are normalized to:
\[
\lambda. (\lambda f 1 0) 0
\]

▶ Free variables (such as f above) remain named.

Semantics and Environments

▶ Operational semantics: rules for how a program can be executed.
▶ Uses a store: a way to formalize “memory”.

Static semantics: rules for how an expression can be assigned a type.
▶ Uses a typing context: assigns types to variables.

Environments are a construct that generalizes stores and typing contexts.
▶ In the paper, they are unordered and finite.

Try 1: A function: \( E(x) = v \Leftrightarrow x \text{ maps to } v \).
▶ Needs external finiteness proof: difficult.

Try 2: A list of pairs: \((x, v) \in E \Leftrightarrow x \text{ maps to } v\).
▶ Explicit reordering: not an issue in practice.

Summary

▶ Core L³: a programming language introduced in [1].
▶ Included a type soundness proof, but it was not mechanized.

Project goal: Mechanize the paper’s type soundness proof of Core L³.
▶ Interesting to mechanize: linear type system, semantic soundness proof.

Each construct from the paper has multiple ways it can be mechanized.
▶ I figured out the best one to use for each; lots of experimentation.

Conclusion: I didn’t finish implementing the cases of the proof.
▶ Should be straightforward now that I mechanized all the paper’s constructs.
▶ I also proved some “helper lemmas” used by the main proof.

Semantic Interpretations

▶ \( V \) is one of the paper’s semantic interpretations.
▶ \( V \) views a type \( \tau \) as the set of values \( v \) with that type, each paired with a store \( \sigma \).

Coq has restrictions to ensure proofs are correct.
▶ Needed to work around them to mechanize \( V \).

Try 1: A ternary (three-way) relation \( V(\tau, \sigma, v) \).
▶ Problem: One of the rules had a recursive use of \( V \) not allowed in Coq relations.

Try 2: A function \( V(\tau) \); returns a relation \( R(\sigma, v) \).
▶ Problem: Coq needs to ensure the function always terminated, but couldn’t.

Try 3: Add a dummy parameter \( \tau_{\eta \rightarrow (\cdot)} \) to Try 2.
▶ It helps Coq see that the function terminates.

[2] “The Locally Nameless Representation” by Charguéraud