Mechanized semantics: when machines reason about their languages

Concluding remarks

Xavier Leroy

2020-02-13

Collège de France, chaire de sciences du logiciel

Summary of the course

We studied how

1. To formalize the semantics of a programming languages
   - Operational semantics
     (by reduction, natural, using a bounded interpreter, ...).
   - Denotational semantics.
2. To verify and to transform programs
   - Type systems.
   - Program logics.
   - Static analyses.
   - Basic compilation (translation to machine code).
   - Optimizing compilation.
3. To use the semantics for proving the soundness of verifications and the correctness of transformations.

Benefits of the Coq mechanization

Definitions are precise (no ambiguities, no omissions) and clear
(except when bound variables are involved…)

Proofs are often cumbersome, but it is not necessary to read them.

It suffices to read (and publish) the intermediate statements (lemmas).

Shared interfaces (the semantics of IMP)
$\Rightarrow$ all the pieces fit together.

Enables us to safely explore advanced approaches:
intrinsically-typed syntax, semantic typing, coinductive approaches, …

A challenge: scaling up

A few examples of large-scale formalizations were described or mentioned in the seminar: JSCert, CompCert and Verasco, RustBelt, CakeML, etc.

The effort required to formalize a real-world language remains high, bordering on unreasonable.

Specialized formalisms exist to describe programming languages (PLT Redex, Ott, K, …). Effective for some uses, not very helpful to build specifications appropriate for formal proof.

A challenge: reusing formalized components

A few libraries are highly reusable:

- Variables and binders: de Bruijn (`autosubst`), locally nameless.
- Program logics: IRIS
- Scott domains.
- Abstract domains for static analysis.

The description of a language (syntax, semantics, typing) remains monolithic and not reusable other than by copy-and-paste.

A lot of “let’s restart from scratch!”.

A lot of fragmentation between the various proof assistants.

Mechanization is one of the best things that happened to programming language research in the last 20 years.

Let’s make the best out of it!