A Verified Compiler for a Higher-Order Language

Thesis Proposal by Ramana Kumar (rk436)

September 20, 2012

Introduction Computer systems typically separate into several layers of abstraction: at the high level are languages suitable for humans to program and reason about, and at the low level is machine code and hardware suitable for physical computation. Compilers translate from high to low. Reasoning about programs is typically with respect to a high-level semantics, implicitly assuming a correct implementation. Thus, a useful assumption for complete formal verification is that the compiler and runtime system preserves the high-level semantics. This dissertation shows a way to discharge that assumption. Moreover, we allow high-level programs written in higher-order logic itself, eliminating the need for leaps of faith between formal specifications and executable machine code.

Thesis Statement Simple type theory suffices to define and verify a semantics-preserving, usable, maintainable compiler and interactive runtime system for a language with higher-order values.

Context There are three landmarks in the field of compiler verification by which we can judge the present proposal:

- The CompCert project[1]: the CompCert C Compiler [2, 3, 1] is a formally verified realistic compiler for a large subset of the C language to real machine code.

  The main extension in this proposal is compiling a language with higher-order values.

  Other work on verified realistic compilers for first-order source languages includes: Jitawa, CerCo, ...?

- The Lambda Tamer project[2] includes a compiler for a higher order language to idealised assembly code verified in Coq ¡ref¿. The emphasis is on proof automation in part enabled by novel representations.

  The main difference in this proposal is the use of simple, rather than dependent, type theory. We still achieve reasonable proof automation and flexbility (hopefully!). By writing our compiler in its own source language, we can produce (by bootstrapping) a verified read-eval-print loop (with incremental compilation), not just a one-shot compiler. An additional minor extension in this proposal is compiling to a real machine language, which primarily involves dealing with finite machine integers, finite stack space, and garbage collection.

  Other work on verified compilers for higher-order languages includes: ¡compcert future work, benton and hur, maude/k/haskell guys, partial pieces(cps/uncurrying/java stuff?)¿

[1] `http://compcert.inria.fr/`
[2] `http://ltamer.sourceforge.net/`

- The CLI stack: ¡complete system¿

  ¡main extension: actually finish, real applications, Moore grand challenge¿ ¡any others doing complete systems?: ...¿

Contributions To summarise, the proposed contributions are:

- An approach to formally defining and proving compiler correctness in higher-order logic when the source language has higher-order values.
- A complete, usable, verified interactive runtime system. In particular, this means:
  - systems programming with a pure functional language
  - parsing, type checking, other bits of front-end? - (joint with Scott/Michael)
  - garbage collection, and finite heap/stack? - (joint with Magnus)
  - dealing with finite machine words etc.? - (joint with Magnus)
  - connecting up the pieces, bootstrapping, repl
- An approach to program verification with an extremely small trust base, namely, the hardware model and the theorem prover kernel. This is made possible in two independent ways. The first option is to compile constants (functions) defined in the logic, that is, to compile executable specifications. The second is to compile the result of a verified parser on traditional source code; in this case, one must also trust the models of the source language syntax and semantics.
- (If time permits.) New techniques for verifying compiler optimisations and/or program complexity bounds.

Demonstrations

- compiler
- theorem prover
- crypto stick

Limitations and Future Work

- compositionality
- more sophisticated type system and/or module system
- more optimisations
- mutable state
- other impure features

References

[1] Xavier Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In *33rd ACM symposium on Principles of Programming Languages*, pages 42–54. ACM Press, 2006.

[2] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[3] Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363–446, 2009.