Designing a trustworthy, extensible proof checker for formal systems verification

Jared Davis

Department of Computer Science, The University of Texas at Austin

Introduction

The author of **theorem proving** software must strike a balance between competing goals.

Trustworthiness

Only a small amount of code should need to be trusted, and there should be a clear logical story explaining why this software works.

Capability

The software should be as automatic as possible. It should provide a useful mixture of decision procedures and reasoning strategies which can be guided by the user. It should be able to follow large proof steps on its own, so that proofs can be kept current as a system's design changes.

Each new capability adds complexity. For example, the source code for **ACL2** is almost 8 megabytes, at times very sophisticated, and is mostly unverified. How can it be trusted?

Worse, how can it be **extended**? If we are not careful, our extensions might not be sound. If being careful means understanding deeply how our changes will impact the rest of the system, then only experts can be trusted to add new functionality.

Our approach

We start with a simple proof checker that really operates at the level of formal proofs. In other words, we begin with a trustworthy **core**.

The core is written in its own logic, and can thus reason about encoded formulas and proofs. We can talk about the **provability** of formulas.

We write our extensions in the same logic. This means the core can reason about extensions, and we can prove that extensions are **sound**. The net result: we need not trust these extensions.

The core proof checker

Terms, formulas, and proofs can be encoded as data. resulting in **proof objects**. These are simply tuples with a *method*, a *conclusion*, and (possibly) some subproofs or additional information.

We write a function to check each type of proof step. To check an entire proof, we simply check each step throughout it.

Total size (estimate): ~1,500 lines of Lisp, versus ~200,000 lines for ACL2. That's much less to trust, and we have clear **correspondence** between the implementation and the logic.

Extensions

The core checker is too limited to use in practice, so we write extensions to add capabilities such as: tautology checking, substitution of equals for equals, evaluation of arbitrary ground terms, term rewriting, and so forth.

How can we trust these extensions? We write them in our logic, so that the core proof checker can reason about them. Then, we **prove** (using the core checker) that each extension is sound.

Example. To verify our tautology checking extension, we need to show that our tautology checker accepts only provable formulas.

Reflection

Finally, we add a rule of reflection. Loosely, this rule allows us to conclude that *F* is true by showing that *F* is provable. This allows us to make use of our extensions by running them on concrete formulas.

Example. Suppose we can show (tautologyp F). Then, by the soundness theorem above, we can conclude (provablep F). Hence, by the reflection rule, we can conclude F is true.

Conclusions

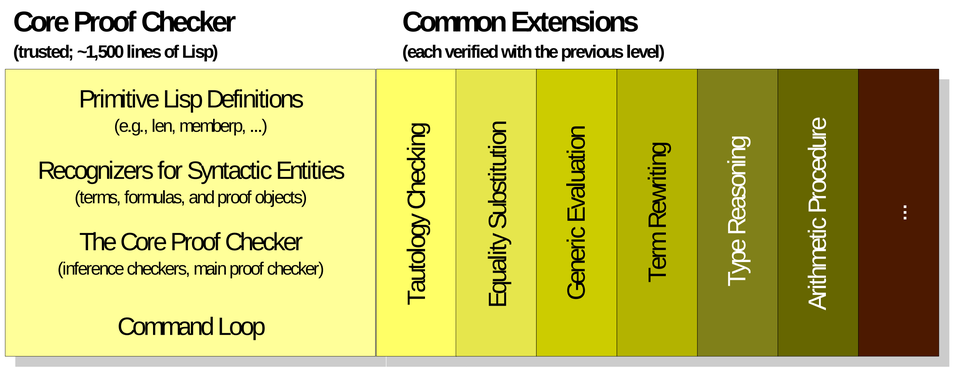
Our approach leads to a proof checker which is highly trustworthy and yet capable of dealing with the complexity of software and hardware designs.

Because the system is developed as a sequence of extensions, it becomes clear what must be done to further extend the system. Users should be able to develop their own extensions without author involvement, and can be confident in the results.

Finally, because our logic is so similar to that of ACL2, we have some hope of being able to recreate existing, useful ACL2 libraries for our tool, and some hope of being able to port ACL2 proofs to our system. This is attractive because ACL2 is already widely used for hardware and software verification

Computing Basis
Choose one or many

Our Proof System



Verification Project

For further information

Please contact jared@cs.utexas.edu, or see my homepage:
http://www.cs.utexas.edu/~jared/

Also, more information on ACL2 can be found at:
http://www.cs.utexas.edu/~moore/acl2/