Efficient proof search in the logical framework

Brigitte Pientka

Carnegie Mellon University

One of the challenges in computer science today is to provide some form of guarantee about the run-time behaviour of programs. This problem is addressed by research on "certified code" where programs are equipped with a certificate (proof) that asserts certain safety properties. Before executing the program, the host machine can then quickly verify the code's safety properties by checking the certificate against the program. It has been shown that a wide range of safety properties, varying from type safety to more general security properties can be checked efficiently.

Safety properties and proofs about them are by and large represented in the logical framework LF which is a meta-language for the specification of logical systems. The inference rules describing the safety policy are described as a higher-order logic program. To verify that a given program fulfills a specified safety policy, the specification is then executed by a logic programming interpreter. A unique feature of the logical framework is that it not only allows reasoning within a given safety policy but also about it. This is especially important as an inconsistent or unsound safety policy compromises the integrity of the certificate. In previous work, it has been demonstrated that it is feasible in principle to automate reasoning *with* and *about* safety policies in the logical framework. However, many straightforward safety specifications are not executable and many properties about safety policies cannot be established automatically, thereby limiting its practical impact on applications such as certified code.

The goal of the thesis is to demonstrate that we can efficiently automate reasoning with and about safety languages. The principal contributions of work up to date to achieve this goal are two-fold: First, I have designed and implemented a logic programming interpreter for LF which eliminates redundant computation by memoizing sub-computation and re-using its result later. Preliminary experiments indicate that this technique allows efficient reasoning *with* safety policies. Second, I have developed a practical system for verifying proofs by complete induction based on a logical view towards termination checking. This work is instrumental in making the reasoning *about* safety languages more powerful. Based on these prior accomplishments, I propose to design an automated reasoning system for LF, develop efficient implementation techniques, and demonstrate that this allows many more specifications to be executed and properties proven.

1. Introduction
2. A motivating example (one of the following two)
   - lambda calculus and proofs about the lambda-calculus [1]
   - pi-calculus and proof about it[2]
3. Previous work
   - Elf: logic programming
   - Twelf: meta-logic programming
4. Overview of proposed work
   - Work already done:
     - Checking proofs by complete induction
     - Prototype for tabled evaluation
   - Work proposed:
     - Incorporate complete induction into meta-theorem prover
     - Incorporate tabled evaluation into meta-theorem prover
     - Design efficient indexing techniques for proof search in LF
5. Related Work
   XSB system, lambdaProlog, theorem proving, etc
6. Basic abstract machine for tabled computation
7. Extensions for efficient theorem-proving for LF
   - Term abstraction
   - Strengthening of hypothesis and contraction
   - Scheduling strategies
   - Interaction between non-tabled and tabled predicates
   - Indexing techniques
   - Efficient and compact proof term generation
   - Efficient linear logic programming ?
8. Efficient meta-theorem prover
   - Incorporating tabled computation into meta-theorem prover
   - Incorporating complete induction into meta-theorem prover
   - Splitting ?
9. Experiments
   - Complete induction
   - Tabled computation
10. Theoretical foundation for efficient proof search
    - Proof irrelevance
    - Soundness of tabled abstract machine
    - Completeness (?) of tabled abstract machine
    - Soundness of meta-theorem prover ?

---

[1] advantage is that it is already all there and I tested tabling on it; not so interesting from a certifying code perspective

[2] maybe more interesting, because it is new, talks about mobile computation, guarantees about interaction between processes. it also opens up questions about inductive and co-inductive arguments. implementation is a little experimental right now.