Efficient proof search in the logical framework

Brigitte Pientka

Carnegie Mellon University

One of the major benefits of logic programming is its declarative nature which makes it ideally suited for representing, verifying and debugging logical specifications. Higer-order logic programming based on the logical framework LF takes this paradigm even one step further by offering a unified approach for the specification of logical systems and also proofs about them. This line of work lies at the heart of applications such as proof-carrying code where a program is accompanied by a proof which guarantees that the execution of the program is safe according to a given safety policy. The inference rules describing the safety policy can be represented 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.

However, higher-order logic programming in LF suffers from the same well-known problem as classical logic programming: completeness of the underlying semantics is sacrificed in order to obtain an efficient interpreter for logic programs. But the problem with the classical logic programming semantics is in fact even more fundamental as has been pointed out by D. H. Warren. The classical semantics of logic programming, there are two kinds of branches, paths which lead to success, i.e. a solution, and branches which do not lead to an answer. However, it does not distinguish between branches which are infinite and branches which can be proven false.

This is untenable from the programmer's point of view but also from an efficient implementation point of view. [blah]

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* logical systems..

The goal of the thesis is to demonstrate that we can efficiently automate reasoning with and about safety languages. To achieve this, I propose to design a revised 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. Basic abstract machine for tabled computation for LF
   - Illustrating the method
   - Formal description
   - Strategies
     - OLD
     - XSB
4. 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 ?
5. High-level Optimizations
   - Subformula property
   - Orderings
6. Previous work
   - Elf: logic programming
   - Twelf: meta-logic programming
7. Related Work
   XSB system, lambdaProlog, theorem proving, etc
8. Overview of proposed work
   - Prototype for tabled evaluation
9. Work proposed:
   - Incorporate tabled evaluation into meta-theorem prover
   - Design efficient indexing techniques for proof search in LF
10. Efficient meta-theorem prover
    - Incorporating tabled computation into meta-theorem prover
11. Experiments
    - subtyping
    - polymorphism

[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.

- lambda calculus
- transition graph

12. Theoretical foundation for efficient proof search
    - Proof irrelevance
    - Soundness of tabled abstract machine
    - Completeness (?) of tabled abstract machine
    - Soundness of meta-theorem prover ?