Evaluation Opportunities in Mechanized Theories (Invited Talk Abstract)

Joe Hurd
Galois, Inc.
joe@galois.com

1 Introduction

When evaluating logic solvers, it is essential to take into account the application domain in which they will operate. For example, a logic solver for reasoning about programs could be applied either inside a compiler to justify the validity of optimizing transformations, or in an IDE plug-in to help developers find bugs in their programs. Despite the similarity of the domain, these two applications lead to different success criteria for the underlying logic solver. Soundness is essential in the compiler application, since no optimizing transformation is permitted to change the observable behavior of the program, but soundness in the bug-finding application can be traded-off if it results in a more productive false positive/false negative rate for the human developer [1].

This invited talk will look at logic solvers through the application lens of constructing and processing a theory library of mechanized mathematics. In fact, constructing and processing theories are two distinct applications, and each will be considered in turn. Construction is carried out by formalizing a mathematical theory using an interactive theorem prover, and logic solvers can remove much of the drudgery by automating common reasoning tasks. At the theory library level, logic solvers can provide assistance with *theory engineering* tasks such as compressing theories, managing dependencies, and constructing new theories from reusable theory components.

2 Theory Construction

The task of formalizing mathematics in an interactive theorem prover involves breaking down proofs of mathematical theorems into small enough steps that the theorem prover accepts them. In the case of a theorem prover in the LCF design [2], a *'small enough step'* is one of the *primitive inference rules* in the logical kernel. This is extremely tedious for a human to do manually, but a full programming language is provided to write logic solvers, called *tactics*, to help with this process. The role of the human is to guide the proof, orchestrating little engines of proof carrying out small-scale automation.

A diverse variety of logic solvers have been implemented as tactics, including term rewriters, decision procedures and first order provers. To investigate the success criteria imposed by this application domain, one particular logic solver will be examined in detail: the *Metis tactic*. Metis [5] is an automated theorem prover for first order logic, and has been implemented as a tactic in the HOL4 [9] and Isabelle [8] interactive theorem provers for higher order logic.[1]. The steps of the Metis tactic are as follows:

1. The initial higher order logic goal is negated and converted by proof to conjunctive normal form. Definitional conjunctive normal form is used to avoid exponential blow-up.

2. A suitable logical interface (m,t) is selected, consisting of a mapping m from higher order logic formulas to first order clauses, and also a translation t that lifts first order refutations to higher order logic proofs [4]. The conjuncts are mapped to first order logic clauses.

[1] Metis is available for download from `http://gilith.com/software/metis`

3. A refutation of the clauses is found, where the search is performed using the ordered paramodulation calculus [7].

4. The refutation is replayed as primitive inferences of higher order logic, completing the proof of the initial goal.

The Metis tactic is effective on many classes of higher order logic goal, particularly those that require a combination of deductive and equality reasoning. It automatically selects an interface for the goal, first trying a fast one that discards type information, and then a more robust one that includes type information in the first order clauses. A syntactic check detects whether the goal contains higher order features such as quantification over functions, and if so an interface is selected that maps higher order logic function application to a first order function symbol (i.e., $f(x)$ is mapped to $\mathsf{app}(f,x)$). This automatic interface selection makes Metis more efficient and also gives it wider coverage than might be expected of a first order proof tactic.

The user explicitly invokes the Metis tactic, so it must respond quickly to be useful. The range of goals that the Metis tactic can prove within a few seconds is more significant than the range that it can prove within a few hours, even though the conventional wisdom in the first order proving community has it that the latter is more indicative of a strong theorem prover. Also against conventional wisdom, completeness is an important feature of a first order proof tactic, not to improve the underlying proof engine, but rather to effectively communicate to the user the possible range of goals that the tactic can handle. Nothing is more frustrating to a user than a tactic failing to prove an *'easy'* goal that is supposedly within its range.

In contrast to completeness, soundness is not an essential feature of a first order proof tactic, because any logical inconsistencies will be detected when the refutation is replayed as primitive inferences. Indeed, the Metis logical interfaces that discard type information are unsound and sometimes result in a bad proof. When that occurs, the tactic simply tries again with a logical interface that preserves type information.

Although not on the scale of a knowledge base, there are a large number of definitions and theorems present in a typical proof context, so a logic solver implemented as a tactic must address the *large theory problem*. The Metis proof tactic sidesteps this by requiring the user to supply the relevant theorems at invocation. However, the *sledgehammer* tactic in Isabelle attempts this automatically by using a relevance filter to choose a subset of the available theorems, and passing them to a state of the art first order theorem prover. If a refutation is found, the theorems that it contains are passed to the Metis tactic to (hopefully) generate a proof that can be replayed with primitive inferences. This has the added advantage of removing the dependency on an external tool in the proof script.

3 Theory Operations

The most robust way to archive a proof of higher order logic is to expand it into primitive inferences. Replaying such a proof does not depend on any tactics, and thus can even take place on a different system with the ability to simulate the primitive inferences. The goal of the OpenTheory project [6] is to transfer the benefits of package management to aid the development of logical theories[2]. A theory package $\Gamma \triangleright \Delta$ of higher order logic consists of:

1. A set Γ of assumptions.

2. A set Δ of theorems.

[2]The OpenTheory project homepage is `http://gilith.com/research/opentheory`

3. A primitive inference proof that the theorems in Δ logically derive from the assumptions in Γ.

Storing the primitive inferences that result from tactics imposes some success criteria on the output from logical solvers implemented as tactics. Above all, the proofs should be short, since a proof format based on primitive inference is necessarily verbose. The compressed form of the theories distributed with the HOL Light theorem prover [3] consist of 769,138 primitive inferences stored in 18Mb. For example, in the current distribution of HOL4, there are 1,136 successful calls to the Metis tactic, which result in 2,880,406 primitive inferences. However, 78% of these primitive inferences are spent converting the goal to conjunctive normal form. Even though Metis implements a powerful resolution calculus, from the perspective of storing proofs, perhaps better results would be obtained by a less powerful tableau method that does not require normalization.

Theory libraries are a rich source of challenge problems for logic solvers. For example, one challenge problem is to take an existing theory package and compress the proof, following the pioneering work of Wos compressing first order logic refutations [10]. Another challenge problem is to take a theory and prove the theorems from the assumptions without looking at the given proof. This large-scale automation of deep mathematical proofs is extremely hard in general, but even logical solvers with limited range could perform much useful work in a theory library. For example, suppose a theory $\Gamma_1 \triangleright \Delta_1$ is upgraded to $\Gamma_2 \triangleright \Delta_2$ in the theory library. It is possible to automatically port theories that depend on the old version to the new version if a proof can be found for the *upgrade theory* $\Gamma_2 \cup \Delta_2 \triangleright \Delta_1$.

References

[1] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles H. Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM*, 53(2):66–75, February 2010.

[2] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979.

[3] John Harrison. HOL light: A tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer, 1996.

[4] Joe Hurd. An LCF-style interface between HOL and first-order logic. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 134–138. Springer, July 2002.

[5] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, September 2003.

[6] Joe Hurd. OpenTheory: Package management for higher order logic theories. In Gabriel Dos Reis and Laurent Théry, editors, *PLMMS '09: Proceedings of the ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems*, pages 31–37. ACM, August 2009.

[7] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[8] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[9] Michael Norrish and Konrad Slind. A thread of HOL development. *The Computer Journal*, 41(1):37–45, 2002.

[10] Larry Wos and Gail W. Pieper. *A fascinating country in the world of computing: your guide to automated reasoning*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.