Theory Engineering: Proving in the Large

Joe Hurd

Galois, Inc.
joe@galois.com

ITP Workshop
Monday 24 August 2009

galois

Talk Plan

1. Introduction
2. Theory Packages
3. Verified Software
4. Summary

galois

Motivation

- Interactive theorem proving is growing up.
- It has moved beyond toy examples of mathematics and program verification.
  - The FlySpeck project is driving the HOL Light theorem prover towards a formal proof of the Kepler sphere-packing conjecture.
  - The CompCert project used the Coq theorem prover to verify an optimizing compiler from a large subset of C to PowerPC assembly code.
- There is a need for theory engineering techniques to support these major verification efforts.
  - Theory engineering is to proving as software engineering is to programming. *"Proving in the large."*

galois

Software Engineering for Theories

An incomplete list of software engineering techniques applicable to the world of theories:

- **Standards:** Programming languages, basis libraries.
- **Abstraction:** Module systems to manage the namespace and promote reuse.
- **Multi-Language:** Tight/efficient (e.g., FFIs) to loose/flexible (e.g., SOAs).
- **Distribution:** Package repos with dependency tracking and automatic installation.

galois

The OpenTheory Project

- The OpenTheory project aims to apply software engineering principles to theories of higher order logic.[1]
- The initial case study for the project is Church's simple theory of types, extended with Hindley-Milner polymorphism.
  - The logic implemented by HOL4, HOL Light and ProofPower.
- By focusing on a concrete case study we aim to investigate the issues surrounding:
  - Exchanging theories between theorem prover implementations.
  - Building a common library of higher order logic theories.
  - Discovering design techniques for theories that compose well.
  - Installing and upgrading theories while respecting their dependencies.

[1] OpenTheory was started in 2004 with Rob Arthan.

galois

OpenTheory Vision

- **Goal:** A distributed package management system for theories.
  - Includes formalized mathematics (a.k.a. specifications).
  - Includes verified higher order logic functions.
  - Includes embeddings of hardware platforms (e.g., ARM) and programming languages (e.g., C), with verified software.
- **Central Problem:** Managing theory dependencies, to support:
  - Installing theories on top of native theories.
  - Authors releasing new versions of theories.
  - Minimizing obsolete theories.
- Take inspiration from successful package management systems (e.g., apt-get, cabal, Nix).

Theory Dependencies

- A theory of higher order logic consists of:
  1. An import list of theorems Γ that the theory requires.
  2. An export list of theorems Δ that the theory provides.
  3. A formal proof $\Gamma \vdash \Delta$ that the theorems in Δ logically derive from the theorems in Γ.
- By binding the type operators and constants in Γ, theories behave like ML functors.
  - This naturally supports installing theories on top of native theories.
  - Also supports a limited form of theory interpretation.
- There is a theory engineering challenge to design theories that can be applied in many contexts.

galois

Theory Installation

- **The Theory Installation Problem:** Given a set of available theorems Θ, find a binding σ for a theory $\Gamma \vdash \Delta$ such that $\Gamma\sigma \subseteq \Theta$.
- **After Installation:** The new set of available theorems is $\Theta \cup \Delta\sigma$.
- It may be impossible to install a theory, but possible if some other theories are installed first.
- Modern package managers typically offer a simple interface to such recursive installation:

```
opentheory install complex-analysis
```

galois

Theory Upgrade

- Offer theory developers the capability to mark that a theory package obsoletes others.
  - Typically used to prefer newer versions of the same theory.
  - Can also be used to merge parallel theory developments.
- Modern package managers typically offer a simple interface to upgrading all installed packages:

```
opentheory upgrade all
```

- Can statically check that all the theory dependencies will match up after an upgrade.

Packaging Verified Software

Using theory packages for verified software addresses many of the logistical needs:

- **Distribution:** Download software from repos, check the proofs, and install on your local machine.
- **Versioning:** Developers can release new versions of software, obsolete packages can be marked.
- **Upgrade:** Can statically guarantee that an upgrade will be safe, so long as the required properties still hold of the new version.

galois

Semi-Formal Verification of Software

- The expressivity of higher order logic allows it to naturally span the gap between abstract specifications and executable higher order functions.
- Haskell can efficiently execute higher order functions, and its purity has led to recent successes on multicore architectures.
- Makes sense to implement a Haskell back end for OpenTheory higher order functions (like Haskabelle for Isabelle).
- This is a promising approach to developing correct and efficient code for future architectures.

High Assurance Software

- For the highest level of assurance, verify properties of embedded programs w.r.t. a formalized semantics of their hardware platform/programming language.
- The formalized platform semantics and program specification must live in theory packages where upgrades are restricted.
- The embedded programs can live in regular theory packages, where the developer is free to make upgrades that still satisfy the spec.

galois

Summary

- This talk has presented the next steps for the OpenTheory project, which aims to apply software engineering principles to theories of higher order logic.
- Package management techniques support distribution, installation and safe upgrade.
- Theory packages also support construction of verified software libraries at different points in the effort/formality trade-off.
- The project web page:

  `http://gilith.com/research/opentheory`

galois