The *Lean* Theorem Prover

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

(Lean's principal developer is Leonardo de Moura,
Microsoft Research, Redmond)

September 2014

Automated theorem proving and formal verification

- Automated theorem proving: want *powerful, fast methods*
- Formal verification: want *secure guarantees*

These pull in different directions.

Automated theorem proving

Domain general

- fast satisfiability methods
- equational theorem proving
- first-order theorem proving (resolution, tableau)

Domain specific

- integer / linear arithmetic
- nonlinear real arithmetic
- numerical methods
- algebraic methods

Combination methods aim to get the best of both worlds.

Interactive theorem proving

Some systems: Mizar, HOL, Isabelle, Coq, HOL-light, ACL2, PVS, Agda, ...

The user works interactively with the system to construct a formal proof.

Design space:

- Logic: first-order, simple types, dependent types
- Classical vs. constructive
- Interaction with computation (internal vs. external)

Lean

Aims to bring the two worlds together:

- An interactive theorem prover with powerful automation.
- An automated reasoning tool that
    - produces proofs,
    - has a rich language,
    - can be used interactively, and
    - is built on a verified mathematical library.

Lean

The Lean theorem prover is being developed by

- Leonardo de Moura (Microsoft Research)
- Soonho Kong (CMU, a student of Ed's!)

The Lean standard library is being developed by

- Jeremy Avigad (CMU)
- Floris van Doorn (CMU)
- Leonardo de Moura (Microsoft Research)

Contributors

- Cody Roux (Draper)
- Robert Lewis (CMU)
- Parikshit Khanna (Indian Institute of Technology, Kanpur)

The logical framework

Lean's default logical framework is a version of the Calculus of Constructions with:

- an impredicative, proof-irrelevant type Prop of propositions
- a non-cumulative hierarchy of universes, Type 1, Type 2, ... above Prop
- universe polymorphism
- inductively defined types

Features:

- The core is constructive.
- Can comfortably import classical logic.
- Can work in homotopy type theory.

```
inductive nat : Type :=
  zero : nat,
  succ : nat → nat

namespace nat

notation `ℕ` := nat

theorem zero_or_succ_pred (n : ℕ) :
    n = 0 ∨ n = succ (pred n) :=
induction_on n
  (or.inl rfl)
  (take m IH, or.inr
    (show succ m = succ (pred (succ m)),
      from congr_arg succ pred_succ⁻¹))
```

```
inductive decidable (p : Prop) : Type :=
inl : p  → decidable p,
inr : ¬p → decidable p

theorem em (p : Prop) {H : decidable p} :
    p ∨ ¬p :=
induction_on H
   (λ Hp, or.inl Hp)
   (λ Hnp, or.inr Hnp)

theorem and_decidable [instance] {a b : Prop}
    (Ha : decidable a) (Hb : decidable b) :
  decidable (a ∧ b)

theorem has_decidable_eq [instance] [protected] :
  decidable_eq ℕ
```

The elaborator

Can handle:

- Dependent type theory
- Implicit arguments, higher-order unification
- Overloading
- Coercions
- Type classes

Features:

- No other proof system handles all of these.
- The elaborator uses nonchronological backtracking.
- It is really fast.

The implementation

- Written in C++, for performance.
- Functional data structures for backtracking, parallelization.
- Small kernel (7,000 lines of C++ code).
- Lua bindings, for user-defined tactics and parser extensions.

The user interface

Features:

- text files, with unicode symbols
- emacs
- a "Lean server" tracking changes and answering queries
- flycheck checks in the background, highlights errors
- the ninja build system maintains dependencies
- the Lean server provides type information, goals
- robust autocompletion

For a demo, see: `https://asciinema.org/a/12277`

Short term goals

A release in early 2015, with:

- A stable kernel and elaborator.
- A stable user interface.
- A basic tactic language.
- Some automation (e.g. the term simplifier)
- The beginnings of a standard library:
  - basic data types: nat, int, lists, ...
  - algebraic structures: orderings, equivalence relations, groups, rings, categories, ...

See: `https://github.com/leanprover/lean`

Long terms goals

A powerful system for

- reasoning about complex systems,
- reasoning about mathematics, and
- verifying claims about both.