Automation and Computation in the Lean Theorem Prover

Robert Y. Lewis[1] Leonardo de Moura[2]

[1]Carnegie Mellon University

[2]Microsoft Research, Redmond

April 6, 2016

Goals for this talk

- Introduce Lean: a new proof assistant based on dependent type theory
- Discuss the current state of, and future prospects for, automation in Lean
- Introduce Polya: a system for verifying real nonlinear inequalities

Credit to...

- Leonardo de Moura
- Soonho Kong, Floris van Doorn, Daniel Selsam
- Jeremy Avigad, Cody Roux
- Many others....

Lean details

- Open source
- Constructive dependent type theory
- Designed with automation in mind
  - Interactive theorem prover with strong automation
  - Automated theorem prover with verified mathematical library
- "Standard" and "homotopy type theory" flavors
  - Standard: proof-irrelevant, impredicative Prop, classical logic available, quotient types
  - HoTT: proof-relevant, no impredicative Prop, univalence, HIT
- Seamlessly integrate classical reasoning

Lean details

- Small kernel
  - No termination checker, pattern matching, etc.
- Reference type checker
- Mixed tactic and declarative proof styles
- Powerful elaborator with strong type class inference mechanism
- May see similarities to other systems: not surprising

- Very young system!

Lean details

The standard library already has:

- datatypes: booleans, lists, tuples, finsets, sets
- number systems: nat, int, rat, real, complex
- the algebraic hiearachy, through ordered fields
- "big operations": finite sums and products, etc.
- elementary number theory (e.g. primes, gcd's, unique factorization, etc.)
- elementary set theory
- elementary group theory (Sylow's theorem)
- beginnings of analysis: topological spaces, limits, continuity, the intermediate value theorem

Lean details

Currently working on:

- topology (connectedness, compactness)
- linear algebra
- analysis: transcendental functions, the Frechet derivative
- measure theory (Lebesgue integration)
- group theory

Example

```
definition infinite_primes (n : nat) : {p | p ≥ n ∧ prime p} :=
let m := fact (n + 1) in
have m ≥ 1,     from le_of_lt_succ (succ_lt_succ (fact_pos _)),
have m + 1 ≥ 2, from succ_le_succ this,
obtain p `prime p` `p | m + 1`, from sub_prime_and_dvd this,
have p ≥ 2,     from ge_two_of_prime `prime p`,
have p > 0,     from lt_of_succ_lt (lt_of_succ_le `p ≥ 2`),
have p ≥ n,     from by_contradiction
 (suppose ¬ p ≥ n,
   have p < n,   from lt_of_not_ge this,
   have p ≤ n + 1, from le_of_lt (lt.step this),
   have p | m,   from dvd_fact `p > 0` this,
   have p | 1,   from
         dvd_of_dvd_add_right (!add.comm ▸ `p | m + 1`) this,
   have p ≤ 1,   from le_of_dvd zero_lt_one this,
   absurd (le.trans `2 ≤ p` `p ≤ 1`) dec_trivial),
subtype.tag p (and.intro this `prime p`)
```

Type class inference

- Can declare classes and instances
- Variables marked with [ ] are inferred by searching for instances of the correct types
- Search is recursive and backtracking and caches aggressively

Type class inference

```
inductive inhabited [class] (A : Type) : Type :=
mk : A → inhabited A

definition default (A : Type) [h : inhabited A] : A :=
inhabited.rec (λ a, a) h

definition prop_inhabited [instance] : inhabited Prop :=
inhabited.mk true

definition fun_inhabited [instance]
    (A B : Type) [h : inhabited B] : inhabited (A → B) :=
inhabited.mk (λ x : A, default B)

definition prod_inhabited [instance]
    (A B : Type) [ha : inhabited A] [hb : inhabited B] : inhabited (A × B)
    :=
inhabited.mk (default A, default B)

eval default (nat → nat × Prop)
-- λ (a : nat), (0, true)
```

Algebraic hierarchy

Type class inference lets us construct the algebraic hierarchy in a uniform way.

```
structure semigroup [class] (A : Type) extends has_mul A :=
(mul_assoc : ∀ a b c, mul (mul a b) c = mul a (mul b c))

structure monoid [class] (A : Type) extends semigroup A, has_one A :=
(one_mul : ∀ a, mul one a = a) (mul_one : ∀ a, mul a one = a)

structure group [class] (A : Type) extends monoid A, has_inv A :=
(mul_left_inv : ∀ a, mul (inv a) a = one)

theorem inv_mul_cancel_left {A : Type} [H : group A] (a b : A) :
          a⁻¹ · (a · b) = b :=
 by rewrite [←mul.assoc, mul.left_inv, one_mul]

structure linear_ordered_field [class] (A : Type) extends
    linear_ordered_ring A, field A
```

Algebraic hierarchy

```
structure left_module [class] (R M : Type) [ringR : ring R]
  extends has_scalar R M, add_comm_group M :=
(smul_left_distrib : ∀ (r : R) (x y : M),
   smul r (add x y) = (add (smul r x) (smul r y)))
(smul_right_distrib : ∀ (r s : R) (x : M),
   smul (ring.add r s) x = (add (smul r x) (smul s x)))
(mul_smul : ∀ r s x, smul (mul r s) x = smul r (smul s x))
(one_smul : ∀ x, smul one x = x)

definition m_left_module [instance] (A : Type) [ring A] (m n : ℕ) :
           left_module A (matrix A m n) := ...
```

Algebraic hierarchy

The standard library has

- order structures (including lattices, complete lattices)
- additive and multiplicative semigroups, monoids, groups, ...
- rings, fields, ordered rings, ordered fields, ...
- modules over arbitrary rings, vector spaces, normed spaces, ...
- homomorphisms preserving appropriate parts of structures

Concrete number structures

When types instantiate these algebraic structures, all theorems proved in the general case are immediately available in the concrete setting.

```
definition real_ord_ring [reducible] [instance] : ordered_ring ℝ :=
{ ordered_ring, real.comm_ring,
  le_refl := real.le_refl,
  le_trans := @real.le_trans,
  mul_pos := real.mul_pos,
  mul_nonneg := real.mul_nonneg,
  zero_ne_one := real.zero_ne_one,
  add_le_add_left := real.add_le_add_left,
  le_antisymm := @real.eq_of_le_of_ge,
  lt_irrefl := real.lt_irrefl,
  lt_of_le_of_lt := @real.lt_of_le_of_lt,
  lt_of_lt_of_le := @real.lt_of_lt_of_le,
  le_of_lt := @real.le_of_lt,
  add_lt_add_left := real.add_lt_add_left
}
```

Concrete number structures

When types instantiate these algebraic structures, all theorems proved in the general case are immediately available in the concrete setting.

```
theorem translate_cts {f : ℝ → ℝ} (Hcon : continuous f) (a : ℝ) :
      continuous (λ x, (f x) + a) :=
 begin
  intros x ϵ Hϵ,
  cases Hcon x Hϵ with δ Hδ,
  cases Hδ with Hδ₁ Hδ₂,
  existsi δ,
  split,
  assumption,
  intros x' Hx',
  rewrite [add_sub_comm, sub_self, add_zero],
  apply Hδ₂,
  assumption
 end
```

Numeral computation

Binary numerals can be used in Lean in any structure that has 0, 1, and +.

```
definition bit0 {A : Type} [s : has_add A]  (a  : A) : A := add a a
definition bit1 {A : Type} [s : has_one A] [t : has_add A] (a : A) : A :=
    add (bit0 a) one
```

With the right instances present, numerical simplification can be done efficiently in an arbitrary type using binary arithmetic.

```
example : 1900 + 220*4 = (2780 : ℕ) :=
 by norm_num

example : (253 + 5 * 10) / 3 = (101 : ℝ) :=
 by norm_num

example (A : Type) [linear_ordered_field A] : (11 + 25) / 8 = (9 : A) / 2
 := by norm_num
```

Upshots for automation

This uniform development is helpful for automation.

- Theorems are not duplicated, so strategies apply across structures.
- Numerals behave identically, and easy to identify when the necessary properties are present.

Present/forthcoming:

- Term simplifier
- Fourier-Motzkin linear inequality solver
- Simplex solver
- Blast (general purpose auto proof search)
- Blast (machine learning)
- Polya: nonlinear inequalities

Broader questions

- How do we adapt standard proof search techniques to dependent type theory?
- How can we incorporate AI methods in the short term? Long term?

Polya

Polya: a tool for heuristically verifying real-valued inequalities over extensions of RCF

- Lightweight
- Flexible, extensible
- "Reasonably" constructive
- NOT a decision procedure
- Avigad, Lewis, Roux. *A heuristic prover for real inequalities* (2014)

A motivating example

$$0 < x < y,\ u < v$$
$$\Longrightarrow$$
$$2u + \exp(1 + x + x^4) < 2v + \exp(1 + y + y^4)$$

- This inference is not contained in linear arithmetic or real closed fields.
- This inference is tight: symbolic or numeric approximations to exp are not useful.
- Backchaining using monotonicity properties suggests many equally plausible subgoals.
- But, the inference is completely straightforward.

A new method

We propose and implement a method based on this type of heuristically guided forward reasoning. Our method:

- Verifies inequalities on which other procedures fail.
- Is relatively easy to implement in Lean.
- Captures natural, human-like inferences.
- Performs well on real-life problems.

- Is not complete.
- Is not guaranteed to terminate.

We envision it as a complement, not a replacement, to other verification procedures.

Implementations

- Python prototype: not proof-producing, but can experiment
- Lean version: on the way!

Terms and normal forms

The inequality

$$15 < 3(3y + 5x + 4xy)^2 f(u + v)^{-1}$$

is expressed canonically as

$$\underbrace{\underbrace{\mathbf{1}}_{t_0} < 5 \cdot \underbrace{(\underbrace{\underbrace{x}_{t_1} + \frac{3}{5} \cdot \underbrace{y}_{t_2} + \frac{4}{5} \cdot \underbrace{xy}_{t_3 = t_1 t_2}}_{t_6 = t_1 + \frac{3}{5} t_2 + \frac{4}{5} t_3})^2 \underbrace{f(\underbrace{\underbrace{u}_{t_4} + \underbrace{v}_{t_5}}_{t_7 = t_4 + t_5})}_{t_8 = f(t_7)}}_{t_9 = t_6^2 t_8^{-1}}}_{t_0 \leq 5 t_9}$$

Modules and database

Any comparison between canonical terms can be expressed as $t_i \bowtie 0$ or $t_i \bowtie c \cdot t_j$, where $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$. This is in the common language of addition and multiplication.

A central database (the blackboard) stores term definitions and comparisons of this form.

Modules use this information to learn and assert new comparisons.

The procedure has succeeded in verifying an implication when modules assert contradictory information.

Arithmetic modules

If we restrict out attention to atomic comparisons and only additive (or only multiplicative) definitions, then linear methods can be used.

Given additive equations $\{t_i = \sum_j c_j \cdot t_{k_j}\}$ and atomic comparisons $\{t_i \bowtie c \cdot t_j\}$ and $\{t_i \bowtie 0\}$, want to saturate the blackboard with the strongest implied atomic comparisons. Two methods:

- Fourier-Motzkin elimination
- Geometric techniques

The geometric method scales better, but the two methods have the same output.
Modulo some concerns about sign information and irrational numbers, we can do the same with multiplicative equations.

Fourier-Motzkin additive module

To find comparisons between t_1 and t_2, eliminate t_3:

$$\begin{aligned} 3t_1 + 2t_2 - t_3 &> 0 \\ 4t_1 + t_2 + t_3 &\geq 0 \\ 2t_1 - t_2 - 2t_3 &\geq 0 \\ -2t_2 - t_3 &> 0 \end{aligned}$$

Fourier-Motzkin additive module

To find comparisons between t_1 and t_2, eliminate t_3:

$$\begin{aligned} 3t_1 + 2t_2 - t_3 &> 0 \\ 4t_1 + t_2 + t_3 &\geq 0 \\ 2t_1 - t_2 - 2t_3 &\geq 0 \\ -2t_2 - t_3 &> 0 \end{aligned} \quad \Longrightarrow \quad 7t_1 + 3t_2 > 0$$

Fourier-Motzkin additive module

To find comparisons between t_1 and t_2, eliminate t_3:

$$\begin{array}{r} 3t_1 + 2t_2 - t_3 > 0 \\ 4t_1 + t_2 + t_3 \geq 0 \\ 2t_1 - t_2 - 2t_3 \geq 0 \\ -2t_2 - t_3 > 0 \end{array} \quad \Longrightarrow \quad \begin{array}{l} 7t_1 + 3t_2 > 0 \\ 10t_1 + t_2 \geq 0 \end{array}$$

Fourier-Motzkin additive module

To find comparisons between t_1 and t_2, eliminate t_3:

$$\begin{array}{r} 3t_1 + 2t_2 - t_3 > 0 \\ 4t_1 + t_2 + t_3 \geq 0 \\ 2t_1 - t_2 - 2t_3 \geq 0 \\ -2t_2 - t_3 > 0 \end{array} \quad \Longrightarrow \quad \begin{array}{r} 7t_1 + 3t_2 > 0 \\ 10t_1 + t_2 \geq 0 \\ 4t_1 - t_2 > 0 \end{array}$$

Fourier-Motzkin additive module

To find comparisons between t_1 and t_2, find the strongest pair:

$$\begin{aligned} 3t_1 + 2t_2 - t_3 &> 0 \\ 4t_1 + t_2 + t_3 &\geq 0 \\ 2t_1 - t_2 - 2t_3 &\geq 0 \\ -2t_2 - t_3 &> 0 \end{aligned} \implies \begin{aligned} 7t_1 + 3t_2 &> 0 \\ 10t_1 + t_2 &\geq 0 \\ 4t_1 - t_2 &> 0 \end{aligned} \implies \begin{aligned} t_1 &> -\frac{3}{7}t_2 \\ t_1 &\geq -\frac{1}{10}t_2 \\ t_1 &> \frac{1}{4}t_2 \end{aligned}$$

Fourier-Motzkin additive module

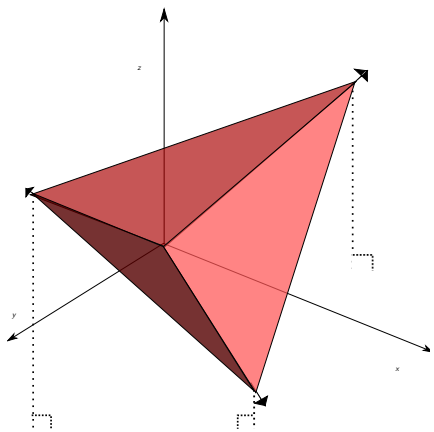
To find comparisons between t_1 and t_2, find the strongest pair:

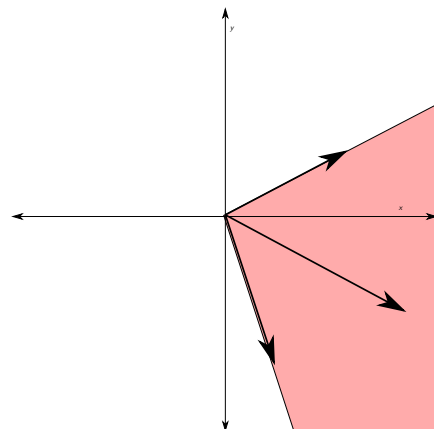
$$\begin{aligned} 3t_1 + 2t_2 - t_3 &> 0 \\ 4t_1 + t_2 + t_3 &\geq 0 \\ 2t_1 - t_2 - 2t_3 &\geq 0 \\ -2t_2 - t_3 &> 0 \end{aligned} \implies \begin{aligned} 7t_1 + 3t_2 &> 0 \\ 10t_1 + t_2 &\geq 0 \\ 4t_1 - t_2 &> 0 \end{aligned} \implies \begin{aligned} t_1 &> -\frac{3}{7}t_2 \\ t_1 &\geq -\frac{1}{10}t_2 \\ t_1 &> \frac{1}{4}t_2 \end{aligned}$$

Geometric additive module

The equalities and inequalities in the blackboard describe a polyhedron with vertex at the origin, in halfplane representation.

By projecting this polyhedron to the $t_i t_j$ plane, one can find the strongest implied comparisons between t_i and t_j.





We use the computational geometry packages cdd and lrs for the conversion from half-plane representation to vertex representation.

Axiom instantiation module

- Users can define function terms and axiomatize their behavior.
  - f increasing, positive, etc.
- This module instantiates these axioms heuristically.
- Built in handling for min/max, sin/cos, exp/log

Successes

Our implementation in Python successfully proves many theorems, some of which are not proved by other systems.

$$0 < x < 1 \implies 1/(1-x) > 1/(1-x^2) \tag{1}$$

$$0 < u,\ u < v,\ 0 < z,\ z+1 < w \implies (u+v+z)^3 < (u+v+w)^5 \tag{2}$$

$$\left(\forall x, y.\ x \leq y \to f(x) \leq f(y)\right),\ u < v,\ 1 < v,\ x \leq y \implies u + f(x) \leq v^2 + f(y) \tag{3}$$

Successes

$$(\forall x, y.\ f(x+y) = f(x)f(y)),\ f(a+b) > 2,\ f(c+d) > 2 \implies f(a+c+b+d) > 4 \tag{4}$$

$$0 \leq n,\ n < (K/2)x,\ 0 < c,\ 0 < \epsilon < 1 \implies \left(1 + \frac{\epsilon}{3(C+3)} \cdot n < Kx\right) \tag{5}$$

$$x < y,\ u \leq v \implies u + \min(x + 2u, y + 2v) \leq x + 3v \tag{6}$$

$$y > \max(2, 3x),\ x > 0 \implies \exp(4y - 3x) > \exp(6) \tag{7}$$

Limitations

Since our method is incomplete, it fails on a wide class of problems where other methods succeed.

$$x > 0,\ xyz < 0,\ xw > 0 \implies w > yz \tag{8}$$

$$x^2 + 2x + 1 \geq 0 \tag{9}$$

$$\begin{aligned} 4 \leq x_i \leq 6.3504 \implies & \\ & x_1x_4(-x_1 + x_2 + x_3 - x_4 + x_5 + x_6) \\ & + x_2x_5(x_1 - x_2 + x_3 + x_4 - x_5 + x_6) \\ & + x_3x_6(x_1 + x_2 - x_3 + x_4 + x_5 + -x_6) \\ & - x_2x_3x_4 - x_1x_3x_5 - x_1x_2x_6 - x_4x_5x_6 > 0 \end{aligned} \tag{10}$$

KeYmaera examples

On a collection of 4442 problems generated automatically by KeYmaera, we solve 4255 (96%) with a 3-second timeout.

- 8 minutes using geometric packages

Example

```
Hypothesis:  ru10**2 == (1/3)*x1u0**2
Hypothesis:  x1u0 <= 0
Hypothesis:  ru10 > 0
Hypothesis:  d1 == -1 * om * (h2 + -1*x2)
Hypothesis:  d2 == om * (h1 + -1*x1)
Hypothesis:  (h1 + -1*x1)**2 + (h2 + -1*x2)**2 == r**2
Hypothesis:  1 != ru10**-1 * ru10
Conclusion:  False
```

Connections to other systems

- SMTLIB input
- Why3 driver (rudimentary)

Lean implementation:

- Work in progress!
- Question: how to formalize geometric version?

Thanks for listening!

Lean:

- `http://leanprover.github.io` (interactive tutorial)
- de Moura, Kong, Roux. *Elaboration in dependent type theory.* (Available online)

Polya:

- `http://github.com/avigad/polya` (source code and directions)
- Avigad, Lewis, Roux. *A heuristic prover for real inequalities.* (JAR, 2016)