The Lean HoTT library

Floris van Doorn

Department of Philosophy
Carnegie Mellon University
github.com/leanprover/lean2

7 July 2017

The Lean Theorem Prover

Lean is an interactive theorem prover announced in 2015.

It is developed principally by Leonardo de Moura at Microsoft Research, Redmond.

It is open source, released under a permissive license, Apache 2.0.

Lean 2 vs Lean 3

The newest version — Lean 3 — doesn't support HoTT (yet?).

The HoTT library is actively developed in Lean 2, an older but stable version.

The Lean Theorem Prover

Notable features:

- implements dependent type theory
- written in C++, with multi-core support
- small, trusted kernel with multiple independent type checkers
- standard and HoTT instantiations
- powerful elaborator
- can use proof terms or tactics
- Emacs mode with proof-checking on the fly
- browser version runs in javascript

Lean's kernel

Lean's kernel for HoTT implements dependent type theory with

- a hierarchy of (non-cumulative) universes:
  `Type.{0} : Type.{1} : Type.{2} : Type.{3} : ...`
- universe polymorphism:
  `definition id.{u} {A : Type.{u}} : A → A := λa, a`
- dependent products: `Πx : A, B`
- inductive types (à la Dybjer, constructors and recursors)

There are multiple reference checkers with about $1500 - 2000$ lines of code.

Lean's kernel

The kernel is smaller and simpler than those of Coq and Agda.

The kernel does not contain

- a termination checker
- fixpoint operators
- Pattern matching
- coinductive types
- inductive-inductive or inductive-recursive types
- universe cumulativity
- the eta rule for records

HITs and UA

In addition to the kernel, you need to trust:

- the univalence axiom
- two higher inductive types: quotients and truncations (as a kernel extension).

```
HIT quotient (A : Type) (R : A → A → Type) : Type :=
| i : A → quotient A R
| e : Π{x y : A}, R x y → i x = i y
```

We no not use type-in-type or any resizing rules.

Simplicial and cubical sets should model Lean.

HoTT mode

The quotient can define many HITs:

- the pushout, hence also the suspension, smash, join, spheres, ...;
- sequential colimits;
- HITs with 2-constructors, such as the torus and Eilenberg-MacLane spaces $K(G, 1)$.
- the propositional truncation (vD);
- (not formalized) n-truncations (Rijke) and certain (ω-compact) localizations (Rijke, vD);

Lean's elaborator

Lean has a powerful elaborator that handles:

- implicit universe levels
- first-order and higher-order unification
- computational reductions
- overloading
- coercions
- type class inference
- Definitions by pattern matching
- tactic proofs

Pattern matching

Definitions like these are compiled down to recursors:

```
definition tail {A : Type} :
  Π{n}, vector A (succ n) → vector A n
| tail (h :: t) := t

definition zip {A B : Type} :
  Π{n}, vector A n → vector B n → vector (A × B) n
| zip nil     nil     := nil
| zip (a::va) (b::vb) := (a, b) :: zip va vb

definition diag : Π{n}, vector (vector A n) n → vector A n
| diag nil                := nil
| diag ((a :: v) :: M) := a :: diag (map tail M)
```

Calculational proofs

$$\sum_{(a,b):\Sigma_{a:A}B(a)} C(a) \simeq \sum_{(a,c):\Sigma_{a:A}C(a)} B(a)$$

```
definition sigma_assoc_comm_equiv {A : Type} (B C : A → Type)
  : (Σ(v : Σa, B a), C v.1) ≃ (Σ(u : Σa, C a), B u.1) :=
calc
  (Σ(v : Σa, B a), C v.1)
      ≃ (Σa (b : B a), C a)      : sigma_assoc_equiv
  ... ≃ (Σa (c : C a), B a)      : sigma_equiv_sigma_right
                                     (λa, !comm_equiv_nondep)
  ... ≃ (Σ(u : Σa, C a), B u.1) : sigma_assoc_equiv
```

Example tactic proof

```
variable (P : S¹ → Type)
definition circle.rec_unc (v : Σ(p : P base), p =[loop] p)
  : Π(x : S¹), P x :=
begin
  intro x, cases v with p q, induction x,
  { exact p },
  { exact q }
end

definition circle_pi_equiv
  : (Π(x : S¹), P x) ≃ Σ(p : P base), p =[loop] p :=
begin
  fapply equiv.MK,
  { intro f, exact ⟨f base, apd f loop⟩ },
  { exact circle.rec_unc P },
  { intro v, induction v with p q, fapply sigma_eq,
    { reflexivity },
    { esimp, apply pathover_idp_of_eq, apply rec_loop }},
  { intro f, apply eq_of_homotopy, intro x, induction x,
    { reflexivity },
    { apply eq_pathover_dep, apply hdeg_squareover, esimp, apply rec_loop }}
end
```

The HoTT library

The library is separated into two repositories, the HoTT library and the Spectral repository.[1]

The main goal is to explore and formalize synthetic homotopy theory.

Lines of code: (rounded)

Library	files	blank	comment	code
HoTT-library	168	8700	3600	32800
Spectral	57	2600	2000	10900

Contributors: vD, Jakob von Raumer, Ulrik Buchholtz, Jeremy Avigad, Egbert Rijke, Steve Awodey, Mike Shulman and others.

[1] `https://github.com/cmu-phil/Spectral`

Synthetic homotopy theory

The library contains:

- Freudenthal suspension theorem
- Whitehead's Theorem
- Seifert-Van Kampen theorem
- long exact sequence of homotopy groups
- complex and quaternionic Hopf fibration
- $\pi_k(\mathbb{S}^n)$ for $k \leq n$ and $\pi_3(\mathbb{S}^2)$.
- adjunction between smash and pointed maps.
- Cohomology theory
- The Serre Spectral sequence (almost!)

Synthetic homotopy theory

The library contains:

- Freudenthal suspension theorem
- Whitehead's Theorem
- Seifert-Van Kampen theorem
- long exact sequence of homotopy groups
- complex and quaternionic Hopf fibration
- $\pi_k(\mathbb{S}^n)$ for $k \leq n$ and $\pi_3(\mathbb{S}^2)$.
- adjunction between smash and pointed maps.
- Cohomology theory
- The Serre Spectral sequence (almost!)

Synthetic homotopy theory

Given a pointed map $f : X \to Y$. Then the following is a long exact sequence:

$$\vdots$$

$$\pi_2(F) \xrightarrow[\pi_2(p_1)]{} \pi_2(X) \xrightarrow{\pi_2(f)} \pi_2(Y)$$

$$\xrightarrow{\pi_1(\delta)}$$

$$\pi_1(F) \xrightarrow[\pi_1(p_1)]{} \pi_1(X) \xrightarrow{\pi_1(f)} \pi_1(Y)$$

$$\xrightarrow{\pi_0(\delta)}$$

$$\pi_0(F) \xrightarrow[\pi_0(p_1)]{} \pi_0(X) \xrightarrow{\pi_0(f)} \pi_0(Y)$$

Here F is the fiber of f and p_1 is the first projection.

Synthetic homotopy theory

The Hopf fibration (by Ulrik Buchholtz):

```
variables (A : Type) [H : h_space A] [K : is_conn 0 A]

definition hopf : susp A → Type :=
susp.elim_type A A
  (λa, equiv.mk (λx, a * x) !is_equiv_mul_left)

definition hopf.total : sigma (hopf A) ≃ join A A

definition circle_h_space : h_space S¹
definition sphere_three_h_space : h_space (S 3)
```

Synthetic homotopy theory

Smash is adjoint to pointed maps:

```
definition smash_adjoint_pmap (A B C : Type*) :
  ppmap (A ∧ B) C ≃* ppmap B (ppmap A C)

definition smash_assoc (A B C : Type*) :
  A ∧ (B ∧ C) ≃* (A ∧ B) ∧ C

definition smash_assoc_natural
  (f : A →* A') (g : B →* B') (h : C →* C') :
  psquare (smash_assoc A B C) (smash_assoc A' B' C')
          (f ∧→ (g ∧→ h))((f ∧→ g) ∧→ h)
```

Synthetic homotopy theory

```
structure ptruncconntype (n : ℕ₋₂) : Type :=
  (A : Type*)
  (H1 : is_conn n A)
  (H2 : is_trunc (n+1) A)

notation cType*[n] := /- category on ptruncconntype n -/

definition Grp_equivalence_cptruncconntype :
  Grp ≃c cType*[0]

definition AbGrp_equivalence_cptruncconntype (n : ℕ) :
  AbGrp ≃c cType*[n+1]
```

Serre Spectral Sequence

Theorem

Given a simply connected pointed type (X, x_0)*, a family* $F : X \to \mathrm{Type}$ *and a group* G*. Then*

$$H^p(X, H^q(F(x_0); G)) \Longrightarrow H^{p+q}(\Sigma_{x:X}, F(x); G).$$

This means that the cohomology group $H^{p+q}(Z; Y)$ *is "built up from"* $H^p(X, H^q(F; Y))$ *in some technical way.*

Parametrized version in Lean: ($95+\%$ done)

```
variables {X : Type} (F : X → Type) (Y : spectrum)
          (s₀ : ℤ) (H : is_strunc s₀ Y)
definition serre_convergence :
  (λp q, H^-p[(x : X), H^-q[F x, Y]]) ⟹[g]
  (λn, H^-n[Σ(x : X), F x, Y])
```

Extra: pointed maps

A pointed map from $A : \mathrm{Type}^*$ to $B : \mathrm{Type}^*$ is a map $f : A \to B$ which respects the basepoint: $f_0 : f(a_0) = b_0$.

Extra: pointed maps

A pointed map from $A : \mathrm{Type}^*$ to $B : \mathrm{Type}^*$ is a map $f : A \to B$ which respects the basepoint: $f_0 : f(a_0) = b_0$.

A pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}^*$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

Extra: pointed maps

A pointed map from $A : \mathrm{Type}^*$ to $B : \mathrm{Type}^*$ is a map $f : A \to B$ which respects the basepoint: $f_0 : f(a_0) = b_0$.

A pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}^*$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

A more general pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}$ with point $b_0 : B(a_0)$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

Extra: pointed maps

A pointed map from $A : \mathrm{Type}^*$ to $B : \mathrm{Type}^*$ is a map $f : A \to B$ which respects the basepoint: $f_0 : f(a_0) = b_0$.

A pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}^*$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

A more general pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}$ with point $b_0 : B(a_0)$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

Now we can define a pointed homotopy between f and g as an element of $\Pi^*_{a:A} f(a) = g(a)$ where $f(a_0) = g(a_0)$ has basepoint $f_0 \cdot g_0^{-1}$

Extra: pointed maps

A pointed map from $A : \mathrm{Type}^*$ to $B : \mathrm{Type}^*$ is a map $f : A \to B$ which respects the basepoint: $f_0 : f(a_0) = b_0$.

A pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}^*$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

A more general pointed dependent map from $A : \mathrm{Type}^*$ to $B : A \to \mathrm{Type}$ with point $b_0 : B(a_0)$ is a map $f : \Pi_{a:A} B(a)$ such that $f_0 : f(a_0) = b_0$.

Now we can define a pointed homotopy between f and g as an element of $\Pi^*_{a:A} f(a) = g(a)$ where $f(a_0) = g(a_0)$ has basepoint $f_0 \cdot g_0^{-1}$

Advantage: we can define a pointed 2-homotopy between $h, k : f \sim^* g$ as $h \sim^* k$

Advantages of Lean

- Powerful unification allows you to make higher-order arguments implicit
- Small kernel
- Very large library for synthetic homotopy theory
- Fast elaborator and type-checker

Disdvantages of Lean

- Powerful unification causes degration of error messages
- No active development of the source code
- Need better tools to print/simplify the goal you are proving

Thank you