The Lean Theorem Prover and Homotopy Type Theory

Jeremy Avigad and Floris van Doorn

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University
http://leanprover.github.io

May 2016

Outline

- Formal verification and interactive theorem proving
- The Lean Theorem Prover
- Lean's kernel
- Lean's elaborator
- The standard and HoTT libraries
- Automation
- Education

Then I will yield the floor to Floris, who will tell you about the HoTT library.

Formal verification

Formal methods can be used to verify correctness:

- verifying that a circuit description, an algorithm, or a network or security protocol meets its specification; or
- verifying that a mathematical statement is true.

Formal verification in industry

Formal methods are commonly used:

- Intel and AMD use ITP to verify processors.
- Microsoft uses formal tools such as Boogie and SLAM to verify programs and drivers.
- CompCert has verified the correctness of a C compiler.
- Airbus uses formal methods to verify avionics software.
- Toyota uses formal methods for hybrid systems to verify control systems.
- Formal methods were used to verify Paris' driverless line 14 of the Metro.
- The NSA uses (it seems) formal methods to verify cryptographic algorithms.

Formal verification in mathematics

There is no sharp line between industrial and mathematical verification:

- Designs and specifications are expressed in mathematical terms.
- Claims rely on background mathematical knowledge.

Mathematics has a different character, however:

- Problems are conceptually deeper, less heterogeneous.
- More user interaction is needed.

Interactive theorem proving

Working with a proof assistant, users construct a formal axiomatic proof.

In most systems, this proof object can be extracted and verified independently.

Interactive theorem proving

Some systems with large mathematical libraries:

- Mizar (set theory)
- HOL (simple type theory)
- Isabelle (simple type theory)
- HOL light (simple type theory)
- Coq (constructive dependent type theory)
- ACL2 (primitive recursive arithmetic)
- PVS (classical dependent type theory)

Interactive theorem proving

Some theorems formalized to date:

- the prime number theorem
- the four-color theorem
- the Jordan curve theorem
- Gödel's first and second incompleteness theorems
- Dirichlet's theorem on primes in an arithmetic progression
- Cartan fixed-point theorems
- the central limit theorem

There are good libraries for elementary number theory, real and complex analysis, point-set topology, measure-theoretic probability, abstract algebra, Galois theory, ...

Interactive theorem proving

Georges Gonthier and coworkers verified the Feit-Thompson Odd Order Theorem in Coq.

- The original 1963 journal publication ran 255 pages.
- The formal proof is constructive.
- The development includes libraries for finite group theory, linear algebra, and representation theory.

The project was completed on September 20, 2012, with roughly

- 150,000 lines of code,
- 4,000 definitions, and
- 13,000 lemmas and theorems.

Interactive theorem proving

Hales announced the completion of the formal verification of the Kepler conjecture (*Flyspeck*) in August 2014.

- Most of the proof was verified in HOL light.
- The classification of tame graphs was verified in Isabelle.
- Verifying several hundred nonlinear inequalities required roughly 5000 processor hours on the Microsoft Azure cloud.

Homotopy Type Theory

Homotopy Type Theory opens up new avenues for formal verification.

It provides a framework for:

- "synthetic" reasoning about spaces up to homotopy
- reasoning about structures of interest in algebraic topology
- reasoning structures in general, and higher-order equivalences between them
- reasoning about generic computation and invariance under change of representation

The Lean Theorem Prover

Lean is a new interactive theorem prover, developed principally by Leonardo de Moura at Microsoft Research, Redmond.

It was "announced" in the summer of 2015.

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

The goal is to make it a community project, like Clang.

The Lean Theorem Prover

The aim is to bring interactive and automated reasoning together, and build

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

The Lean Theorem Prover

Lean is a designed to be a mature system, rather than an experimental one.

- Take advantage of existing theory.
- Build on strengths of existing interactive and automated theorem provers.
- Craft clean but pragmatic solutions.

We have drawn ideas and inspiration from Coq, SSReflect, Isabelle, Agda, and Nuprl, among others.

The Lean Theorem Prover

Goals:

- Verify hardware, software, and hybrid systems.
- Verify mathematics.
- Combine powerful automation with user interaction.
- Support reasoning and exploration.
- Support formal methods in education.
- Create an eminently powerful, usable system.
- Bring formal methods to the masses.

The Lean Theorem Prover

Notable features:

- based on a powerful dependent type theory
- written in C++, with multi-core support
- small, trusted kernel with an independent type checker
- 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
- already has a respectable library
- automation is now the main focus

Contributors

Currently working on the code base: Leonardo de Moura, Daniel Selsam, Lev Nachman, Soonho Kong

Currently working the standard library: Me, Rob Lewis, Jacob Gross

Currently working on the HoTT library: Floris van Doorn, Ulrik Buchholtz, Jakob von Raumer

Contributors: Cody Roux, Joe Hendrix, Parikshit Khanna, Sebastian Ullrich, Haitao Zhang, Andrew Zipperer, and many others.

Lean's kernel

Lean's is based on a version of dependent type theory.

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

Daniel Selsam has written an independent type checker, in Haskell.

It is less than 2,000 lines long.

There are currently two modes supported, a standard mode and the HoTT mode.

Lean's kernel

Common to both modes:

- universes:
  `Type.{0} : Type.{1} : Type.{2} : Type.{3} : ...`
- dependent products: `Π x : A, B`
- inductive types (à la Dybjer, constructors and recursors)

Specific to the standard mode:

- `Type.{0}` (aka `Prop`) is impredicative and proof irrelevant.
- quotient types (`lift f H (quot.mk x) = f x` definitionally).

We use additional axioms for classical reasoning: propositional extensionality, Hilbert choice.

Lean's kernel

Definitions like these are compiled down to recursors:

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

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
```

Lean's kernel

Lean keeps track of which definitions are computable.

```
definition map (f : A → B) : list A → list B
| []       := []
| (a :: l) := f a :: map l

noncomputable definition inv_fun (f : X → Y)
  (s : set X) (dflt : X) (y : Y) : X :=
if H : ∃₀ x ∈ s, f x = y then some H else dflt

definition add (x y : ℝ) : ℝ := ...

noncomputable definition div (x y : ℝ) : ℝ :=
x * y⁻¹
```

Lean's kernel

In the HoTT mode:

- No `Prop`.
- quotient HIT (non-truncated, for an arbitrary type-valued relation)
- n-truncation HIT

Many HITs can be constructed from these two.

Additional axiom: univalence, of course.

Lean's elaborator

Fully detailed expressions in dependent type theory are quite long.

Systems of dependent type theory allow users to leave a lot of information implicit.

Such systems therefore:

- Parse user input.
- Fill in the implicit information.

The last step is known as "elaboration."

Lean's elaborator

Lean has a powerful elaborator that handles:

- higher-order type inference
- computational reductions
- ad-hoc overloading
- type class inference

Leo is rewriting the elaborator:

- avoid backtracking search, to make it more predictable and reliable.
- use a stronger unifier (e.g. with unification hints)

Lean's elaborator

Structures and type class inference have been optimized to handle the 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

Type classes work beautifully.

Lean's elaborator

```
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)

definition pow {A : Type} [s : monoid A] (a : A) : ℕ → A
| 0     := 1
| (n+1) := pow n * a

theorem pow_add (a : A) (m : ℕ) : ∀ n, a^(m + n) = a^m * a^n
| 0        := by rewrite [nat.add_zero, pow_zero, mul_one]
| (succ n) := by rewrite [add_succ, *pow_succ, pow_add,
                          mul.assoc]

definition int.linear_ordered_comm_ring [instance] :
  linear_ordered_comm_ring int := ...
```

Lean's elaborator

Type classes are also used:

- to infer straightforward facts (`finite s`, `is_subgroup G`)
- to simulate classical reasoning constructively (`decidable p`)

Type classes are extensively in the HoTT library:

- for algebraic structures (as in the standard library)
- for category theory
- to infer truncatedness (`is_trunc n A`)
- to infer half-adjoint equivalence (`is_equiv f`)

Lean's elaborator

```
namespace nat_trans

infixl ` ⟹ `:25 := nat_trans
variables {B C D E : Precategory} {F G H I : C ⇒ D}
attribute natural_map [coercion]

protected definition compose [constructor] (η : G ⟹ H)
  (θ : F ⟹ G) : F ⟹ H :=
nat_trans.mk
  (λ a, η a ∘ θ a)
  (λ a b f,
    calc
      H f ∘ (η a ∘ θ a)
          = (H f ∘ η a) ∘ θ a : by rewrite assoc
      ... = (η b ∘ G f) ∘ θ a : by rewrite naturality
      ... = η b ∘ (G f ∘ θ a) : by rewrite assoc
      ... = η b ∘ (θ b ∘ F f) : by rewrite naturality
      ... = (η b ∘ θ b) ∘ F f : by rewrite assoc)
```

Lean's elaborator

In Lean, we can present a proof as a term, as in Agda, with nice syntactic sugar.

We can also use tactics.

The two modes can be mixed freely:

- anywhere a term is expected, `begin ... end` or `by` enter tactic mode.
- within tactic mode, `have ..., from ...` or `show ..., from ...` or `exact` specify terms.

Lean's elaborator

```
theorem sqrt_two_irrational {a b : ℕ} (co : coprime a b) :
  a^2 ≠ 2 * b^2 :=
assume H : a^2 = 2 * b^2,
have even (a^2),
  from even_of_exists (exists.intro _ H),
have even a,
  from even_of_even_pow this,
obtain (c : ℕ) (aeq : a = 2 * c),
  from exists_of_even this,
have 2 * (2 * c^2) = 2 * b^2,
  by rewrite [-H, aeq, *pow_two, mul.assoc, mul.left_comm c],
have 2 * c^2 = b^2,
  from eq_of_mul_eq_mul_left dec_trivial this,
have even (b^2),
  from even_of_exists (exists.intro _ (eq.symm this)),
have even b,
  from even_of_even_pow this,
assert 2 ∣ gcd a b,
  from dvd_gcd (dvd_of_even `even a`) (dvd_of_even `even b`),
have 2 ∣ 1,
  by rewrite [gcd_eq_one_of_coprime co at this]; exact this,
show false,
  from absurd `2 ∣ 1` dec_trivial
```

Lean's elaborator

```
theorem is_conn_susp [instance] (n : ℕ₋₂) (A : Type)
  [H : is_conn n A] : is_conn (n .+1) (susp A) :=
is_contr.mk (tr north)
begin
  apply trunc.rec, fapply susp.rec,
  { reflexivity },
  { exact (trunc.rec (λa, ap tr (merid a)) (center (trunc n A))) },
  { intro a, generalize (center (trunc n A)),
    apply trunc.rec, intro a', apply pathover_of_tr_eq,
    rewrite [transport_eq_Fr,idp_con],
    revert H, induction n with [n, IH],
    { intro H, apply is_prop.elim },
    { intros H,
      change ap (@tr n .+2 (susp A)) (merid a) = ap tr (merid a'),
      generalize a',
      apply is_conn_fun.elim n
            (is_conn_fun_from_unit n A a)
            (λx : A, trunctype.mk' n
              (ap (@tr n .+2 (susp A)) (merid a) = ap tr (merid x))),
      intros,
      change ap (@tr n .+2 (susp A)) (merid a) = ap tr (merid a),
      reflexivity } }
end
```

Lean's standard library

Already has:

- datatypes: booleans, lists, tuples, finsets, sets
- number systems: nat, int, rat, real, complex
- the algebraic hierarchy, 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
- beginnings of analysis: topological spaces, limits, continuity, the intermediate value theorem

Lean's HoTT library

Already has:

- most of the HoTT book, excluding Chapter 10 (set theory) and Chapter 11 (the real numbers)
- pathovers and cubical types (à la Licata and Brunerie)
- novel constructions of higher-inductive types (van Doorn, Rijke)
- real and complex Hopf fibrations (Buchholtz, Rijke)
- long exact sequences (van Doorn)

Current plans

Leo is doing a major rewrite of the system now.

- The elaborator will be slightly less general, but much more stable and predictable.
- "Let" definitions will be added to the kernel.
- There will be a better foundation for automation (goals with extra information, e.g. with indexed information).
- There will be a byte-code compiler and fast evaluator for Lean.

Lean as a programming language

The fast evaluator make it possible to use Lean as an interpreted programming language.

Monadic interfaces will make it possible to write custom tactics or extend the parser, from within Lean.

A PhD student at the University of Washington, Jared Roesch, is working on code extraction to C++.

Automation

Plans for automation:

- A general theorem prover and term simplifier, with
  - awareness of type classes
  - powerful unification and e-matching
- Custom methods for arithmetic reasoning, linear and nonlinear inequalities.
- A flexible framework for adding domain specific tools.

This opens up exciting possibilities for HoTT as well.

Education

We have been using Lean as an educational tool:

- I taught a grad-level seminar on dependent type theory and Lean in spring 2015.
- We have an interactive online tutorial (which has been favorably received).
- I used Lean in an undergraduate introduction to logic and proof in fall 2015.
- It will be offered again in fall 2016, as an optional replacement for "concepts of mathematics" for CS majors.
- I will teach a course on interactive theorem proving in spring 2017.

Conclusions

Take home messages:

- We have made good progress.
- There's a lot going on.
- We still have a long way to go.
- It is an exciting project.

And now Floris will tell you about the HoTT library.

The Lean-HoTT library

- The Lean-HoTT library is under active development
- Development started 1.5 years ago
- The main contributors are me, Jakob von Raumer and Ulrik Buchholtz.
- It is available at
  `github.com/leanprover/lean/blob/master/hott/hott.md`

Theorems from the HoTT book

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1					+	+	+	+	+		+	+			
2	+	+	+	+		+	+	+	+	+	+	+	+	+	+
3	+	+	+	+	$\frac{1}{2}$	+	+	+	+		+				
4	-	+	+	+		+	+	+	+						
5	-		$\frac{1}{2}$	-	-			$\frac{1}{2}$							
6		+	+	+	+	+	+	+	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	+			
7	+	+	+	-	$\frac{3}{4}$	-	-								
8	+	+	+	+	+	$\frac{3}{4}$	-	+	+	$\frac{1}{4}$					
9	$\frac{3}{4}$	+	+	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{1}{2}$	-	-	-						
10	$\frac{1}{4}$	-	-	-	-										
11	-	-	-	-	-	-									

Statistics

HoTT-Lean: ∼28k LOC
HoTT-Coq: ∼31k LOC (in `theories/` folder)
HoTT-Agda: ∼18k LOC (excluding `old/` folder)
UniMath: ∼52k LOC

(excluding blank lines and single-line comments)

Higher-inductive types

There are two primitive HITs in Lean: the n-truncation and *quotients*.

Given $A : \mathcal{U}$ and $R : A \to A \to \mathcal{U}$ the quotient is:
`HIT` $quotient_A(R) :=$

- $q : A \to quotient_A(R)$
- $\Pi(x, y : A),\ R(x, y) \to q(x) = q(y)$

We define all other HITs in terms of these.

Higher-inductive types

There are two primitive HITs in Lean: the n-truncation and *quotients*.

Given $A : \mathcal{U}$ and $R : A \to A \to \mathcal{U}$ the quotient is:
`HIT` $quotient_A(R) :=$

- $q : A \to quotient_A(R)$
- $\Pi(x, y : A),\ R(x, y) \to q(x) = q(y)$

We define all other HITs in terms of these. We can define

- colimits;
- pushouts, hence also suspensions, spheres, joins, ...

Higher-inductive types

There are two primitive HITs in Lean: the n-truncation and *quotients*.

Given $A : \mathcal{U}$ and $R : A \to A \to \mathcal{U}$ the quotient is:
`HIT` $quotient_A(R) :=$

- $q : A \to quotient_A(R)$
- $\Pi(x, y : A),\ R(x, y) \to q(x) = q(y)$

We define all other HITs in terms of these. We can define

- colimits;
- pushouts, hence also suspensions, spheres, joins, ...
- the propositional truncation;
- HITs with 2-constructors, such as the torus and Eilenberg-MacLane spaces $K(G, 1)$;

Higher-inductive types

There are two primitive HITs in Lean: the n-truncation and *quotients*.

Given $A : \mathcal{U}$ and $R : A \to A \to \mathcal{U}$ the quotient is:
`HIT` $quotient_A(R) :=$

- $q : A \to quotient_A(R)$
- $\Pi(x, y : A),\ R(x, y) \to q(x) = q(y)$

We define all other HITs in terms of these. We can define

- colimits;
- pushouts, hence also suspensions, spheres, joins, ...
- the propositional truncation;
- HITs with 2-constructors, such as the torus and Eilenberg-MacLane spaces $K(G, 1)$;
- [WIP] n-truncations and certain (ω-compact) localizations (Egbert Rijke).

Lean-HoTT library

Furthermore, the library includes:

- A library of squares, cubes, pathovers, squareovers, cubeovers (based on the paper by Licata and Brunerie)

```
definition circle.rec {P : S¹ → Type}
  (Pbase : P base) (Ploop : Pbase =[loop] Pbase)
  (x : S¹) : P x
```

Lean-HoTT library

Furthermore, the library includes:

- A library of squares, cubes, pathovers, squareovers, cubeovers (based on the paper by Licata and Brunerie)

```
definition circle.rec {P : S¹ → Type}
  (Pbase : P base) (Ploop : Pbase =[loop] Pbase)
  (x : S¹) : P x
```

- A library of pointed types, pointed maps, pointed homotopies, pointed equivalences

```
definition loopn_ptrunc_pequiv
  (n : ℕ₋₂) (k : ℕ) (A : Type*) :
  Ω[k] (ptrunc (n+k) A) ≃* ptrunc n (Ω[k] A)
```

Lean-HoTT library

Furthermore, the library includes:

- A library of squares, cubes, pathovers, squareovers, cubeovers (based on the paper by Licata and Brunerie)

```
definition circle.rec {P : S¹ → Type}
  (Pbase : P base) (Ploop : Pbase =[loop] Pbase)
  (x : S¹) : P x
```

- A library of pointed types, pointed maps, pointed homotopies, pointed equivalences

```
definition loopn_ptrunc_pequiv
  (n : ℕ₋₂) (k : ℕ) (A : Type*) :
  Ω[k] (ptrunc (n+k) A) ≃* ptrunc n (Ω[k] A)
```

- Some category theory which is not in the book, e.g. limits, colimits and exponential laws:

```
definition functor_functor_iso
  (C D E : Precategory) :
  (C ^c D) ^c E ≅c C ^c (E ×c D)
```

Homotopy Theory in Lean

Loop space of the circle:

```
definition base_eq_base_equiv : base = base ≃ ℤ
definition fundamental_group_of_circle : π₁(S¹.) = gℤ
```

Homotopy Theory in Lean

Connectedness of suspensions (by Ulrik Buchholtz):

```
definition is_conn_susp (n : ℕ₋₂) (A : Type)
  [H : is_conn n A] : is_conn (n .+1) (susp A)
```

Homotopy Theory in Lean

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 (A : Type) [H : h_space A]
  [K : is_conn 0 A] : sigma (hopf A) ≃ join A A

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

Homotopy Theory in Lean

Some results are ported from Agda, such as the Freudenthal equivalence

```
definition freudenthal_pequiv (A : Type*) {n k : ℕ}
  [is_conn n A] (H : k ≤ 2 * n) :
  ptrunc k A ≃* ptrunc k (Ω (psusp A))
```

and the associativity of join (by Jakob von Raumer)

```
definition join.assoc (A B C : Type) :
  join (join A B) C ≃ join A (join B C)
```

Homotopy Theory in Lean

Truncated Whitehead's principle:

```
definition whitehead_principle (n : ℕ₋₂) {A B : Type}
  [HA : is_trunc n A] [HB : is_trunc n B] (f : A → B)
  (H' : is_equiv (trunc_functor 0 f))
  (H : Πa k, is_equiv
                (π→*[k + 1] (pmap_of_map f a))) :
  is_equiv f
```

Here '`pmap_of_map f a`' is the pointed map $(A, a) \xrightarrow{f} (B, f(a))$.

Homotopy Theory in Lean

Eilenberg-MacLane spaces (based on the paper by Licata and Finster):

```
definition EM : CommGroup → ℕ → Type*
variables (G : CommGroup) (n : ℕ)
definition homotopy_group_EM :
  πg[n+1] (EM G (n+1)) ≃g G
theorem is_conn_EM : is_conn (n.-1) (EM G n)
theorem is_trunc_EM : is_trunc n (EM G n)

definition EM_pequiv_1 {X : Type*} (e : π₁ X ≃g G)
  [is_conn 0 X] [is_trunc 1 X] : EM G 1 ≃* X
-- TODO for n > 1
```

LES of homotopy groups

The long exact sequence of homotopy groups.

Given a pointed map $f : X \to Y$.

$$X \xrightarrow{f} Y$$

LES of homotopy groups

The long exact sequence of homotopy groups.

Given a pointed map $f : X \to Y$.

$$F \xrightarrow{p_1} X \xrightarrow{f} Y$$

LES of homotopy groups

The long exact sequence of homotopy groups.

Given a pointed map $f : X \to Y$.

$$F^{(6)} \xrightarrow{p_1^{(6)}} F^{(5)} \xrightarrow{p_1^{(5)}} F^{(4)} \xrightarrow{p_1^{(4)}} F^{(3)} \xrightarrow{p_1^{(3)}} F^{(2)} \xrightarrow{p_1^{(2)}} F \xrightarrow{p_1} X \xrightarrow{f} Y$$

LES of homotopy groups

The long exact sequence of homotopy groups.

Given a pointed map $f : X \to Y$.

$$\begin{array}{ccccccccccccc} F^{(6)} & \xrightarrow{p_1^{(6)}} & F^{(5)} & \xrightarrow{p_1^{(5)}} & F^{(4)} & \xrightarrow{p_1^{(4)}} & F^{(3)} & \xrightarrow{p_1^{(3)}} & F^{(2)} & \xrightarrow{p_1^{(2)}} & F & \xrightarrow{p_1} & X \xrightarrow{f} Y \\ \wr| & & \wr| & & \wr| & & \wr| & & \wr| & & & & \\ \Omega F^{(3)} & & \Omega F^{(2)} & & \Omega F & & \Omega X & & \Omega Y & & & & \end{array}$$

LES of homotopy groups

The long exact sequence of homotopy groups.

Given a pointed map $f : X \to Y$.

$$\begin{array}{ccccccccccccc}
F^{(6)} & \xrightarrow{p_1^{(6)}} & F^{(5)} & \xrightarrow{p_1^{(5)}} & F^{(4)} & \xrightarrow{p_1^{(4)}} & F^{(3)} & \xrightarrow{p_1^{(3)}} & F^{(2)} & \xrightarrow{p_1^{(2)}} & F & \xrightarrow{p_1} & X & \xrightarrow{f} & Y \\
\wr\downarrow & & \wr\downarrow & & \wr\downarrow & & \wr\downarrow & & \wr\downarrow & & & & & & \\
\Omega F^{(3)} & \xrightarrow{-\Omega p_1^{(3)}} & \Omega F^{(2)} & \xrightarrow{-\Omega p_1^{(2)}} & \Omega F & \xrightarrow{-\Omega p_1} & \Omega X & \xrightarrow{-\Omega f} & \Omega Y & & & & & &
\end{array}$$

LES of homotopy groups

The long exact sequence of homotopy groups.

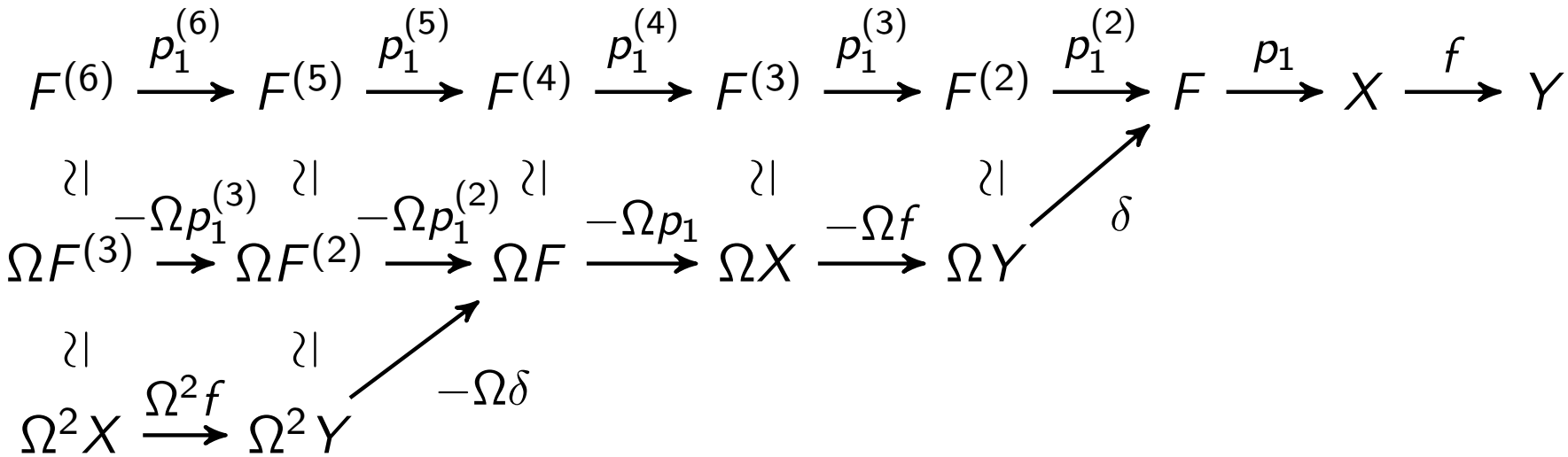
Given a pointed map $f : X \to Y$.

$$
\begin{array}{ccccccccccccccc}
F^{(6)} & \xrightarrow{p_1^{(6)}} & F^{(5)} & \xrightarrow{p_1^{(5)}} & F^{(4)} & \xrightarrow{p_1^{(4)}} & F^{(3)} & \xrightarrow{p_1^{(3)}} & F^{(2)} & \xrightarrow{p_1^{(2)}} & F & \xrightarrow{p_1} & X & \xrightarrow{f} & Y \\
\wr\downarrow & & \wr\downarrow & & \wr\downarrow & & \wr\downarrow & & \wr\downarrow & & & & & & \\
\Omega F^{(3)} & \xrightarrow{-\Omega p_1^{(3)}} & \Omega F^{(2)} & \xrightarrow{-\Omega p_1^{(2)}} & \Omega F & \xrightarrow{-\Omega p_1} & \Omega X & \xrightarrow{-\Omega f} & \Omega Y & & & & & & \\
\wr\downarrow & & \wr\downarrow & & & & & & & & & & & & \\
\Omega^2 X & \xrightarrow{\Omega^2 f} & \Omega^2 Y & & & & & & & & & & & &
\end{array}
$$

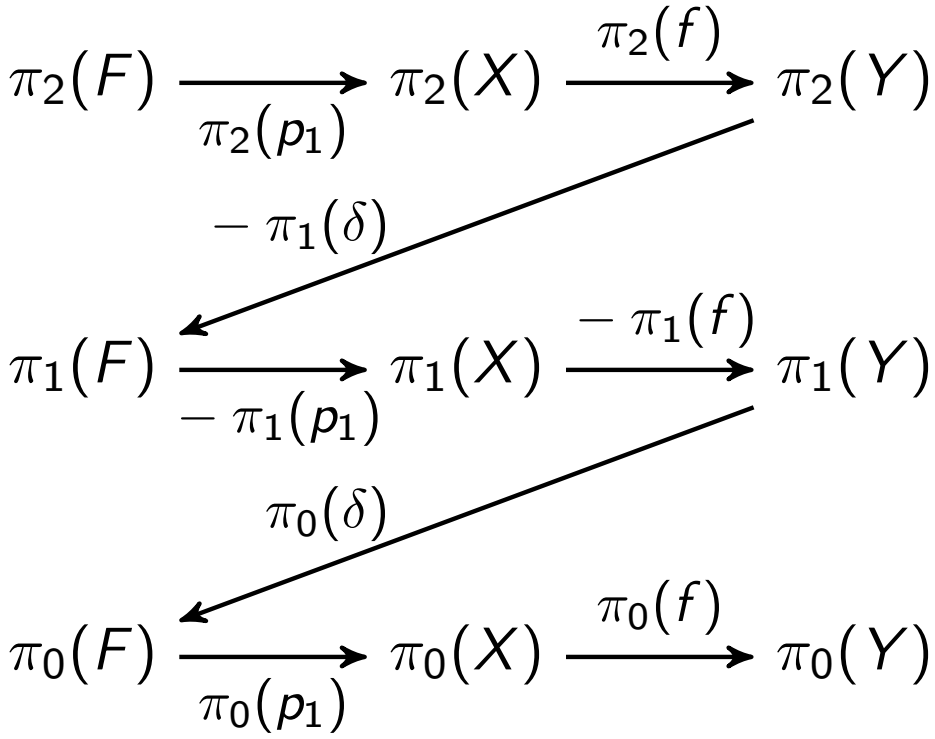
LES of homotopy groups

The long exact sequence of homotopy groups.

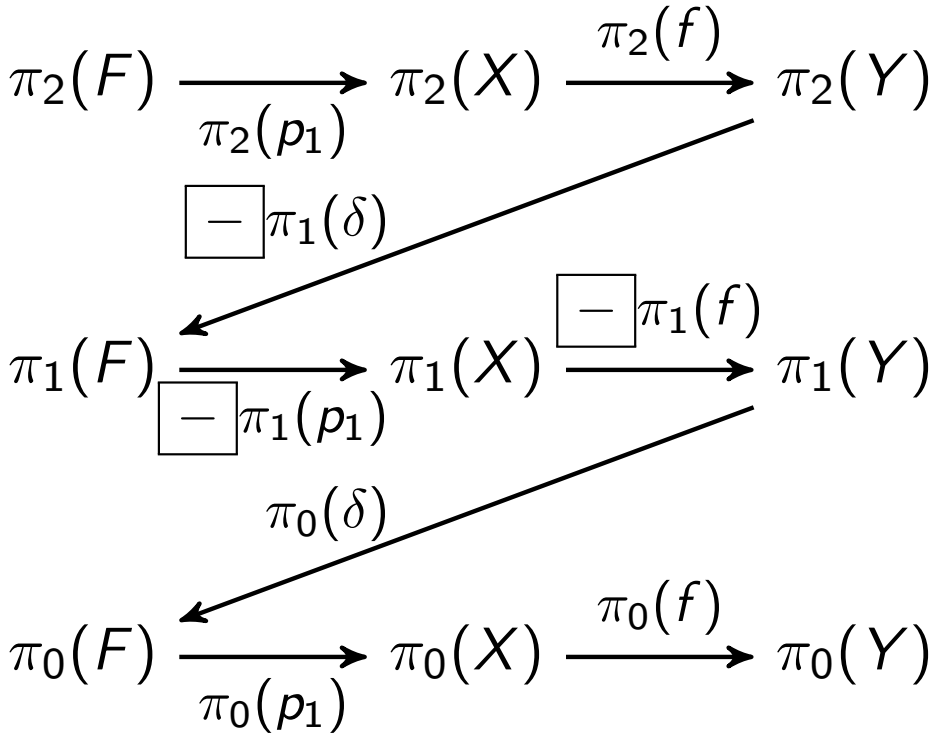
Given a pointed map $f : X \to Y$.



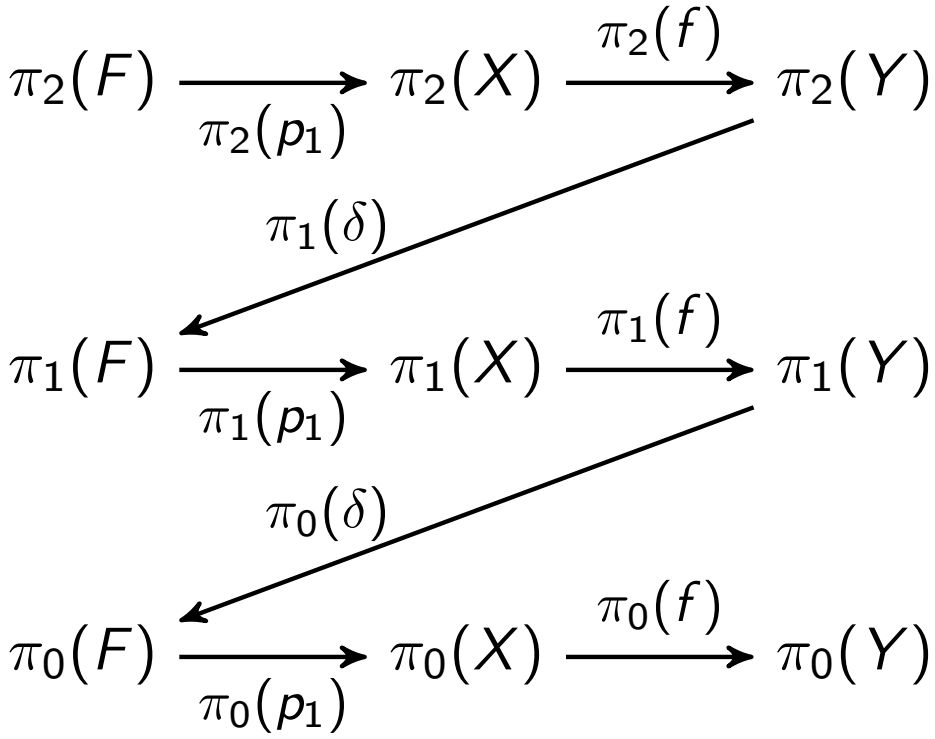
LES of homotopy groups



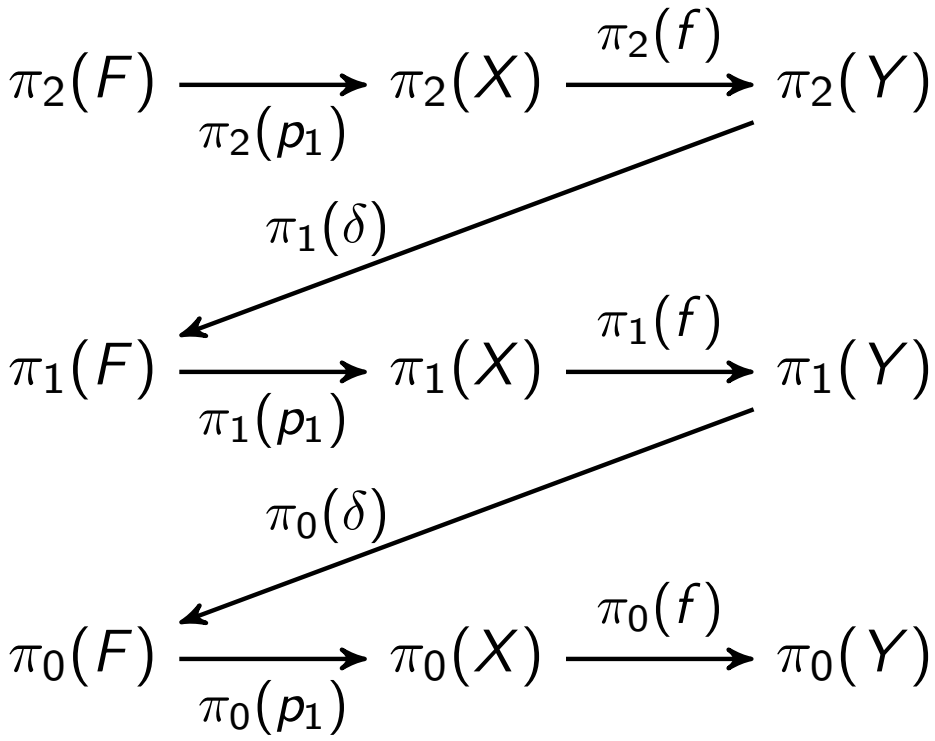
LES of homotopy groups



LES of homotopy groups



LES of homotopy groups



How do we formalize this?

The obvious thing is to have a sequence $Z : \mathbb{N} \to \mathcal{U}$ and maps $f_n : Z_{n+1} \to Z_n$.

Problem: $Z_{3n} = \pi_n(Y)$ doesn't hold definitionally, hence $f_{3n} = \pi_n(f)$ isn't even well-typed.

LES of homotopy groups

Better: Take $Z : \mathbb{N} \times 3 \to \mathcal{U}$, we can define Z by

$$Z_{(n,0)} = \pi_n(Y) \qquad Z_{(n,1)} = \pi_n(X) \qquad Z_{(n,2)} = \pi_n(F).$$

Then we need maps $f_x : Z_{succ(x)} \to Z_x$, where *succ* is the successor function for $\mathbb{N} \times 3$.

LES of homotopy groups

Better: Take $Z : \mathbb{N} \times 3 \to \mathcal{U}$, we can define Z by

$$Z_{(n,0)} = \pi_n(Y) \qquad Z_{(n,1)} = \pi_n(X) \qquad Z_{(n,2)} = \pi_n(F).$$

Then we need maps $f_x : Z_{succ(x)} \to Z_x$, where *succ* is the successor function for $\mathbb{N} \times 3$.

We define chain complexes over an arbitrary type with a successor operation.

LES of homotopy groups

```
definition homotopy_groups : +3ℕ → Set*
| (n, fin.mk 0 H) := π*[n] Y
| (n, fin.mk 1 H) := π*[n] X
| (n, fin.mk k H) := π*[n] (pfiber f)

definition homotopy_groups_fun : Π(n : +3ℕ),
  homotopy_groups (S n) →* homotopy_groups n
| (n, fin.mk 0 H) := π→*[n] f
| (n, fin.mk 1 H) := π→*[n] (ppoint f)
| (n, fin.mk 2 H) := π→*[n] boundary_map ○*
    pcast (ap (ptrunc 0) (loop_space_succ_eq_in Y n))
| (n, fin.mk (k+3) H) := begin exfalso,
    apply lt_le_antisymm H, apply le_add_left end
```

LES of homotopy groups

Then we prove:

- These maps form a chain complex
- This chain complex is exact
- `homotopy_groups (n + 2, k)` are commutative groups.
- `homotopy_groups (1, k)` are groups.
- `homotopy_groups_fun (n + 1, k)` are group homomorphisms.

LES of homotopy groups

Corollaries:

```
theorem is_equiv_π_of_is_connected {A B : Type*}
  {n k : ℕ} (f : A →* B) [H : is_conn_fun n f]
  (H2 : k ≤ n) : is_equiv (π→[k] f)
```

Combine with Hopf fibration:

```
definition π2S2 : πg[1+1] (S. 2) ≃g gℤ
definition πnS3_eq_πnS2 (n : ℕ) :
  πg[n+2 +1] (S. 3) ≃g πg[n+2 +1] (S. 2)
```

Combine with Freudenthal Suspension Theorem:

```
definition πnSn (n : ℕ) :
  πg[n+1] (S. (succ n)) ≃g gℤ
definition π3S2 : πg[2+1] (S. 2) ≃g gℤ
```

Conclusion

- The Lean-HoTT library is quite big and growing quickly.
- Many existing results have been formalized, but there are also results formalized for the first time.
- There is currently an ongoing project to formalize the Serre Spectral Sequence (j.w.w. Ulrik Buchholtz, Mike Shulman, Jeremy Avigad, Steve Awodey, Egbert Rijke, Clive Newstead)

Thank you