Homotopy Type Theory in Lean

Floris van Doorn

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

25 June 2016

Outline

- The Lean Theorem Prover
- Lean's kernel
- Lean's elaborator
- Demo
- The HoTT library

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.

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: Jeremy Avigad, 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 kernel 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)

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

Daniel Selsam has written an independent type checker in Haskell, which is less than 2,000 lines long.

The kernel can be instantiated into two modes, a standard mode and the HoTT mode.

Lean's kernel

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)
```

Standard mode

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 keeps track of which definitions are computable.

```
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
```

HoTT mode

In the HoTT mode:

- There is no `Prop`.
- 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` $quotient_A(R) :=$
  - $q : A \to quotient_A(R)$
  - $\Pi(x, y : A),\ R(x, y) \to q(x) = q(y)$

Many HITs can be constructed from these two.

The Univalence Axiom is assumed.

Lean's elaborator

Fully detailed expressions in dependent type theory are 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:

- implicit universe levels
- higher-order unification
- computational reductions
- ad-hoc overloading
- coercions
- type class inference
- tactic proofs

Higher-order unification

```
variables {A : Type} {B : A → Type} {C : Πa, B a → Type}
definition sigma_transport {a₁ a₂ : A} (p : a₁ = a₂)
  (x : Σ(b : B a₁), C a₁ b) : p ▷ x = ⟨p ▷ x.1, p ▷D x.2⟩ :=
by induction p; induction x; reflexivity
```

$\triangleright$ is transport. `.1` and `.2` are projections.

The first transport is along the family `λ(a : A), Σ(b : B a), C a b`

The second transport is along `B`

The last transport is a "dependent transport" which is a map `C a₁ x.1 → C a₂ (p ▷ x.1)`

Type classes

Structures and type class inference have been optimized to handle the algebraic hierarchy.

The algebraic hierarchy consist of:

- 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

```
structure has_mul [class] (A : Type) := (mul : A → A → A)

structure semigroup [class] (A : Type) extends has_mul A :=
  (is_set_carrier : is_set 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)

variables {A : Type} [monoid A]
definition pow (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]
| (n+1) := by rewrite [add_succ, *pow_succ, pow_add, mul.assoc]

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

Type classes

Type classes are also used in the standard library:

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

They are used in the HoTT library:

- for category theory
- to infer truncatedness (`is_trunc n A`)
- to infer half-adjoint equivalence (`is_equiv f`)

Calculational proofs

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

```
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
```

Tactics and terms

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.

Example term proof

```
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
```

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, induction 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
```

Current plans

Leo is now doing a major rewrite of the system.

- 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 (e.g. goals with indexed hypotheses).
- There will be a byte-code compiler and fast evaluator for Lean. This makes it possible to use Lean as an interpreted programming language.
- Monadic interfaces will make it possible to write custom tactics from within Lean.

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.

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

Demo

Browser version available at:
`https://leanprover.github.io/tutorial/?live&hott`

HoTT library

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

(excluding blank lines and single-line comments)

Higher-inductive types

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

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, vD).

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)
```

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)
```

Loop space of the circle:

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

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)
```

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)
```

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 $(A, a) \xrightarrow{f} (B, f(a))$.

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
```

Seifert-van Kampen Theorem:

```
definition vankampen {S A B C : Type} (k : S → C)
  (f : C → A) (g : C → B) [is_surjective k]
  (x y : A + B) :
  @hom (Π₁ (pushout f g)) _
    (pushout_of_sum f g x) (pushout_of_sum f g y) ≃
  @hom (Groupoid_bpushout k (Π₁⇒ f) (Π₁⇒ g)) _ x y
```

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 $f : X \to Y$.

Define $F :\equiv \mathrm{fiber}_f(y_0) :\equiv (\Sigma(x : X),\ f(x) = y_0)$.

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

LES of homotopy groups

The long exact sequence of homotopy groups.

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

Define $F :\equiv \mathrm{fiber}_f(y_0) :\equiv (\Sigma(x : X),\ f(x) = y_0)$.

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

LES of homotopy groups

The long exact sequence of homotopy groups.

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

Define $F :\equiv \mathrm{fiber}_f(y_0) :\equiv (\Sigma(x : X),\ f(x) = y_0)$.

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

LES of homotopy groups

The long exact sequence of homotopy groups.

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

Define $F :\equiv \mathrm{fiber}_f(y_0) :\equiv (\Sigma(x : X),\ f(x) = y_0)$.

$$
\begin{tikzcd}
F^{(6)} \arrow[r, "p_1^{(6)}"] & F^{(5)} \arrow[r, "p_1^{(5)}"] & F^{(4)} \arrow[r, "p_1^{(4)}"] & F^{(3)} \arrow[r, "p_1^{(3)}"] & F^{(2)} \arrow[r, "p_1^{(2)}"] & F \arrow[r, "p_1"] & X \arrow[r, "f"] & Y \\
\Omega F^{(3)} \arrow[u, phantom, "\wr|"] \arrow[r, "-\Omega p_1^{(3)}"] & \Omega F^{(2)} \arrow[u, phantom, "\wr|"] \arrow[r, "-\Omega p_1^{(2)}"] & \Omega F \arrow[u, phantom, "\wr|"] \arrow[r, "-\Omega p_1"] & \Omega X \arrow[u, phantom, "\wr|"] \arrow[r, "-\Omega f"] & \Omega Y \arrow[u, phantom, "\wr|"]
\end{tikzcd}
$$

LES of homotopy groups

The long exact sequence of homotopy groups.

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

Define $F :\equiv \mathrm{fiber}_f(y_0) :\equiv (\Sigma(x : X),\ f(x) = y_0)$.

$$
\begin{tikzcd}
F^{(6)} \arrow[r, "p_1^{(6)}"] \arrow[d, phantom, sloped, "\simeq"] & F^{(5)} \arrow[r, "p_1^{(5)}"] \arrow[d, phantom, sloped, "\simeq"] & F^{(4)} \arrow[r, "p_1^{(4)}"] \arrow[d, phantom, sloped, "\simeq"] & F^{(3)} \arrow[r, "p_1^{(3)}"] \arrow[d, phantom, sloped, "\simeq"] & F^{(2)} \arrow[r, "p_1^{(2)}"] \arrow[d, phantom, sloped, "\simeq"] & F \arrow[r, "p_1"] & X \arrow[r, "f"] & Y \\
\Omega F^{(3)} \arrow[r, "-\Omega p_1^{(3)}"] \arrow[d, phantom, sloped, "\simeq"] & \Omega F^{(2)} \arrow[r, "-\Omega p_1^{(2)}"] \arrow[d, phantom, sloped, "\simeq"] & \Omega F \arrow[r, "-\Omega p_1"] & \Omega X \arrow[r, "-\Omega f"] & \Omega Y \\
\Omega^2 X \arrow[r, "\Omega^2 f"] & \Omega^2 Y
\end{tikzcd}
$$

LES of homotopy groups

The long exact sequence of homotopy groups.

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

Define $F :\equiv \mathrm{fiber}_f(y_0) :\equiv (\Sigma(x : X),\ f(x) = y_0)$.

$$
\begin{tikzcd}
F^{(6)} \arrow[r, "p_1^{(6)}"] \arrow[d, phantom, "\wr"] & F^{(5)} \arrow[r, "p_1^{(5)}"] \arrow[d, phantom, "\wr"] & F^{(4)} \arrow[r, "p_1^{(4)}"] \arrow[d, phantom, "\wr"] & F^{(3)} \arrow[r, "p_1^{(3)}"] \arrow[d, phantom, "\wr"] & F^{(2)} \arrow[r, "p_1^{(2)}"] \arrow[d, phantom, "\wr"] & F \arrow[r, "p_1"] & X \arrow[r, "f"] & Y \\
\Omega F^{(3)} \arrow[r, "-\Omega p_1^{(3)}"] \arrow[d, phantom, "\wr"] & \Omega F^{(2)} \arrow[r, "-\Omega p_1^{(2)}"] \arrow[d, phantom, "\wr"] & \Omega F \arrow[r, "-\Omega p_1"] & \Omega X \arrow[r, "-\Omega f"] & \Omega Y \arrow[ur, "\delta"'] & & & \\
\Omega^2 X \arrow[r, "\Omega^2 f"] & \Omega^2 Y \arrow[ur, "-\Omega\delta"'] & & & & & &
\end{tikzcd}
$$

LES of homotopy groups

$$
\begin{tikzcd}
\pi_2(F) \arrow[r, "\pi_2(p_1)"'] & \pi_2(X) \arrow[r, "\pi_2(f)"] & \pi_2(Y) \arrow[dll, "-\pi_1(\delta)"'] \\
\pi_1(F) \arrow[r, "-\pi_1(p_1)"'] & \pi_1(X) \arrow[r, "-\pi_1(f)"] & \pi_1(Y) \arrow[dll, "\pi_0(\delta)"'] \\
\pi_0(F) \arrow[r, "\pi_0(p_1)"'] & \pi_0(X) \arrow[r, "\pi_0(f)"] & \pi_0(Y)
\end{tikzcd}
$$

LES of homotopy groups

$$\begin{tikzcd}
\pi_2(F) \arrow[r, "\pi_2(p_1)"'] & \pi_2(X) \arrow[r, "\pi_2(f)"] & \pi_2(Y) \arrow[dll, "\boxed{-}\pi_1(\delta)"'] \\
\pi_1(F) \arrow[r, "\boxed{-}\pi_1(p_1)"'] & \pi_1(X) \arrow[r, "\boxed{-}\pi_1(f)"] & \pi_1(Y) \arrow[dll, "\pi_0(\delta)"'] \\
\pi_0(F) \arrow[r, "\pi_0(p_1)"'] & \pi_0(X) \arrow[r, "\pi_0(f)"] & \pi_0(Y)
\end{tikzcd}$$

LES of homotopy groups

$$
\begin{tikzcd}
\pi_2(F) \arrow[r, "\pi_2(p_1)"'] & \pi_2(X) \arrow[r, "\pi_2(f)"] & \pi_2(Y) \arrow[dll, "\pi_1(\delta)"'] \\
\pi_1(F) \arrow[r, "\pi_1(p_1)"'] & \pi_1(X) \arrow[r, "\pi_1(f)"] & \pi_1(Y) \arrow[dll, "\pi_0(\delta)"'] \\
\pi_0(F) \arrow[r, "\pi_0(p_1)"'] & \pi_0(X) \arrow[r, "\pi_0(f)"] & \pi_0(Y)
\end{tikzcd}
$$

LES of homotopy groups

$$
\begin{tikzcd}
\pi_2(F) \arrow[r, "\pi_2(p_1)"'] & \pi_2(X) \arrow[r, "\pi_2(f)"] & \pi_2(Y) \arrow[dll, "\pi_1(\delta)"'] \\
\pi_1(F) \arrow[r, "\pi_1(p_1)"'] & \pi_1(X) \arrow[r, "\pi_1(f)"] & \pi_1(Y) \arrow[dll, "\pi_0(\delta)"'] \\
\pi_0(F) \arrow[r, "\pi_0(p_1)"'] & \pi_0(X) \arrow[r, "\pi_0(f)"] & \pi_0(Y)
\end{tikzcd}
$$

How do we formulate 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 can define the 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 can define the 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

- Lean is an exciting new proof assistant.
- The Lean-HoTT library is quite big and growing quickly.
- The Lean-HoTT library contains a good basis for serious formalizations.
- There is currently an ongoing collaboration to formalize the Serre Spectral Sequence

Thank you